

# **COMPUTER PROGRAMMING**

**Including**

**HTML & JavaScript  
Database Concepts  
ADO.NET in VB.NET  
SQL  
OOP**

for the subject

*COMPUTER PROGRAMMING*

**FET NCV NQF Level 4**

(TOPICS 1,2,4,5 – Practical component)

## **Notes prepared by:**

Buitendag AAK  
Kirsten JNA  
Labuschagne WA  
Roux L  
van Heerden ME

## © COPYRIGHT: THE AUTHORS

All rights reserved.

**These notes remain the intellectual property of the authors.** Copyright assigned to FET-colleges and to the DoE to duplicate the document in hard copy format only. Any other reproduction method is prohibited. No changes or alterations, in any way, are permitted. Duplication of the CD content is restricted to FET lecturers only.

The authors of each section of these notes take full responsibility regarding copyright issues pertaining to the individual content developed.

Suggestions and comments may be e-mailed to:

[BertieBuitendag@gmail.com](mailto:BertieBuitendag@gmail.com)

May 2009



# PREFACE

The following notes were developed to cover the missing outcomes from the amended Subject and Assessment Guidelines of December 2007 (NCV-document) for the subject COMPUTER PROGRAMMING Level 4.

These notes are intended to be used as a source to cover the newly included **practical** subject outcomes for Topics 1,2,4,5. The nature of this document and subject reflect and exhibit cross curricular content. Therefore many concepts covered in the notes will also address some of the subject outcomes of other topics in the NCV-document.

The notes reflect the interpretation of the authors regarding Topics 1,2,4,5 of the NCV-document. It remains the responsibility of the lecturer to ensure that all subject outcomes in the NCV-document are covered.

Some of the content presented as part of the notes is of a theoretical nature. It is advised that the content presented as part of these notes is studied in conjunction with the prescribed text for the subject.

Certain topics such as the MDI concept of application development, could be well implemented in other programming scenarios, such as the development of a MDI database application.

The appendix on SQL and the development of a query tool are included as a supplement to the current content. Take note that the SQL language include various constructs and differ slightly from DBMS to DBMS, it is therefor advisable to do a study on the various functionalities included as part of the SQL construct of the specific DMBS driver used, for this reason language specific build-in SQL functions such as DATE functions have not been covered. Please consult the prescribed text in that regard.

Most excersises are supplied with solutions, there are however a few excersises which does not have suggested solutions. These excersises were included additionally and are of such a nature that it closely corresponds to the examples given in the text.

The following topics and sections are covered.

Topic 1: Object Oriented Programming

Topic 2: Database application design

Topic 4: Design and build a web-site using HTML

Topic 5: Create multimedia, web-based applications with scripting

- Languages Implemented. VB.NET
- OOP – VB.NET Classes and implementation of Objects in a VB Application.
- OLEDBConnection (Manual connection)
- SQL – Notes (Using SQL in a VB.NET DB Application)
- HTML – Elementary concepts
- JavaScript, Cascading Style Sheets

Various problems and examples have been supplied and the subsequent steps to develop solutions have been provided and discussed in detail. The authors followed a step-by-step approach from problem to solution in many of the examples.

The appendix covers cross cirruclular content, and it is advisable to cover the content presented in the appendices as well.

In some examples regarding VB.NET the event handler agruments has been shorted for printing purposes with a (. . .)



```
Private Sub btnClear_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles btnClear.Click
```

Will be

```
Private Sub btnClear_Click(. . .) Handles btnClear.Click
```

in the text.

We hope that these notes will kindle your enthusiasm for problems solving and programming and motivate you to work hard in achieving your goals to become a developer in the IT-Field.

### **HOW TO USE THIS NOTES:**

Many well vised programmers and developers in industry will agree that one of the best sources of knowledge is by looking at examples, and implementing the constructs in your own code. It is suggested that all programs which appear in the text are to be rewritten by the student as well as the lecturer. This will not only strengthen the learning and mastering of the content but it will hone the student's program development and debugging skills as well.

Attempt the excersises without looking at the suggested solutions. Consult other sources of knowledge as well; some of the best programmers admit that they frequently visit other sources and text for help.

**ENJOY THE SUBJECT!**  
**EMBRACE THE LEARNING!**

**ThE aUtHoRs**



# TABLE OF CONTENTS

SECTION	PAGE
Introduction to HTML	1
Introduction to JavaScript	79
Introduction to OOP in VB.NET	163
Introduction to ADO databases in VB.NET	257
<b>APPENDIXES</b>	
A Introduction to SQL	347
B Custom Query tool in VB.NET	378
C Multiple Document Interface	387
D EasyPong	397

# Introduction to HTML



**AUBREY LABUSCHAGNE**  
**© 2009**

# Practical guide to using HTML

## Introduction to HTML

Notes compiled by: [WA Labuschagne](#)

### Background information about HTML and the environment of Web sites

#### What is a web page?

A web page can contain numerous types of information, which is able to be seen, heard or interact by the end user. A web page contains Hypertext Markup Language (HTML) code.

#### What is Hypertext Markup Language (HTML)?

HTML is a language for describing how pages of text, graphics, and other information are organized, formatted, and linked together.

#### What is a web browser?

A web browser is a software application that allows the user to access resources on the World Wide Web (WWW). Most web browsers provide a graphical interface to the web. These browsers include Microsoft Internet Explorer, Mozilla Firefox, Opera, Safari, Netscape Navigator, and Konqueror.

The web browser renders the web page in a graphical representation to display the web page to the user. The web browser displays the translation rather than the coding. It also interprets HTML commands to collect, arrange, and display the parts of the web page. This allows anyone to "browse the Web" by simple *point and click* navigation, bypassing the need to know commands used in software languages.

#### What is a URL?

Uniform Resource Locator (URL) specifies where an identified resource is available and the mechanism for retrieving it.

A URL comprises a number of elements:

- Protocol used for the request
- The domain or the IP of the server
- The port number
- Subdirectory path
- Name of the resource

For example:

- <http://www.google.com/index.html>
- <http://www.supersolution.co.za:8080/document/services.html>
- <ftp://ftp.google.com>



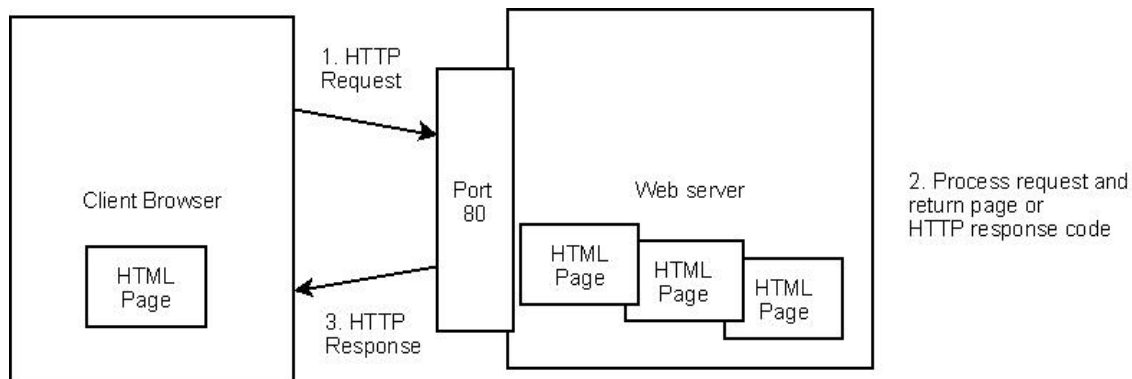
## What is HTTP?

Hypertext transfer Protocol is used by the World Wide Web to send information. HTTP is a request-response protocol. Whereby the client will send a HTTP request from the web browser, the web server receives the request and then responds with a HTTP Response. It usually uses port 80.

## What is a web server?

A web browser is a computer program that is responsible for accepting HTTP requests from clients and serving them HTTP responses along with optional data contents, which usually are web pages such as HTML documents and linked objects (images, etc.).

## How does a user request a web page?



The User will type a URL address into the web browser (1). A HTTP request will be send to a web server as referenced by the URL that was entered. The web server receives the HTTP request and creates a HTTP Response that will be send back(2). This HTTP Response will send the HTML page back to the web browser. Once the web browser receives the HTTP Response from the web server and start to interpret the HTML code to display the web page that was requested by the user (3).

## Tags that every HTML page must have

Most HTML tags will have an opening tag, which indicates where a piece of text begins, and a closing tag, which indicates where the piece of text ends. Closing tags start with a / (forward slash)

For example

```
<b>Hello</b>
```

<b> - is the opening tag

Hello - is the text that must be made bold

</b> - closing tag

Every web page must have the following tags:

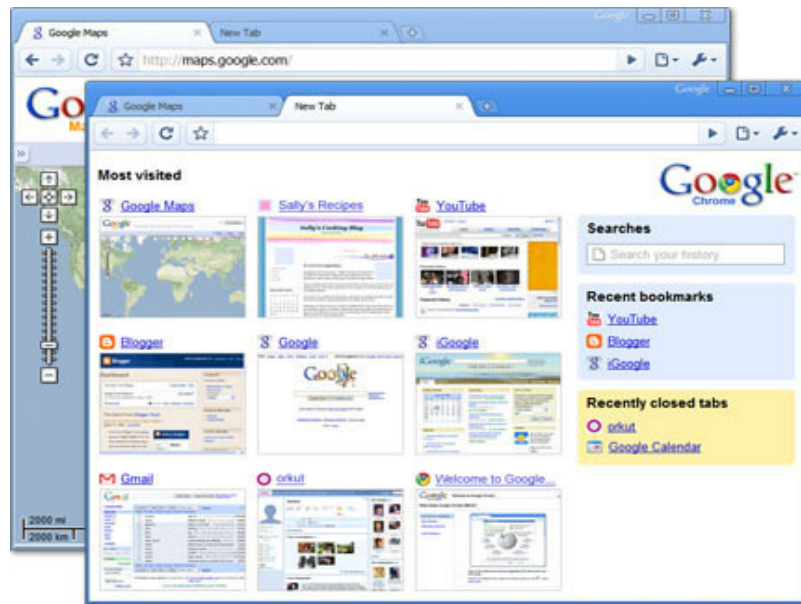
<html> - Indicates that this is a web page

<body>- Everything between these tags will be visible in the web browser

<head> - Provide header information with the HTTP request

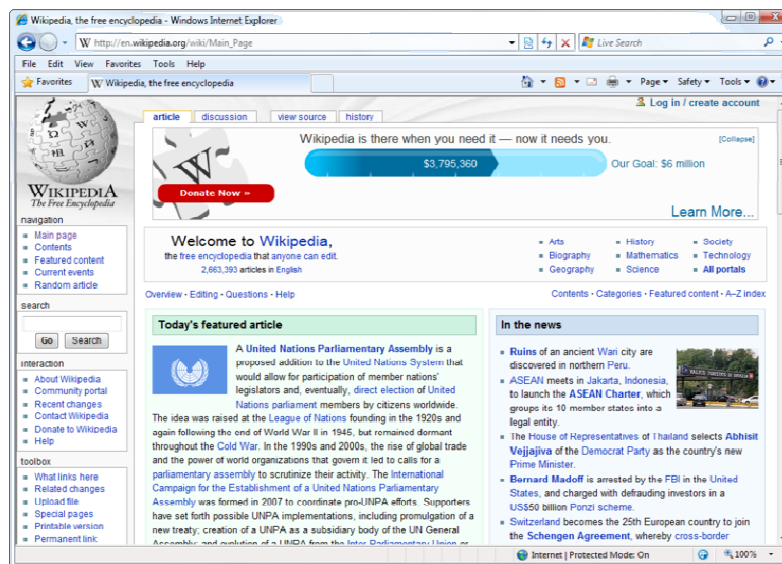
<title>- The title text that will be used to identify the web page

## Google Chrome



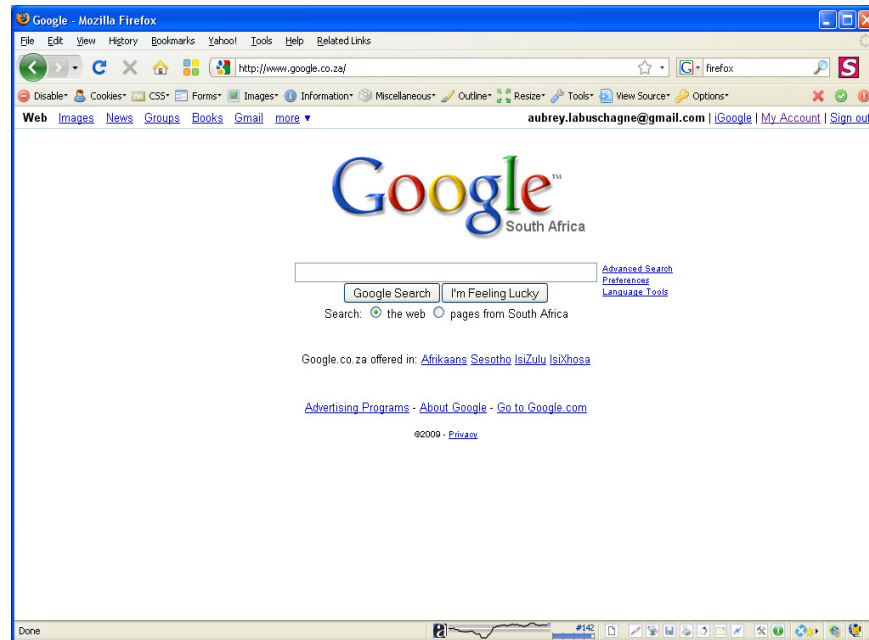
Link to Google Chrome:  
<http://www.google.com/chrome>

## Internet Explorer



Link to Internet Explorer:  
<http://www.microsoft.com/windows/downloads/ie/getitnow.mspx>

## Mozilla Firefox



Link to Mozilla Firefox:

<http://www.mozilla-europe.org/en/firefox/>

### Viewing the source code of a web page using the browser

Every web page contains HTML tags that are rendered by the web browser. The web browser as mentioned before will then interpret the HTML tags and display the presentational version of the web page in the web browser.

The user can view the HTML tags of the web page by viewing the page source of the web page. Different web browsers have the feature to display the page source also known as the source code of the web page.

### Using Internet Explorer



**Step 1:** Click on **View**

**Step 2:** Select **Source** from the drop down menu



## Using Firefox



**Step 1:** Click on **View**

**Step 2:** Select **Page Source** from the drop down menu

Shortcut key: CTRL+U

## Using Google Chrome



Google Chrome

**Step 1:** Click on the **"Control the current page"** icon

**Step 2:** Select **developer** from the drop down menu

**Step 3:** Select **View Source** from the drop down menu

Shortcut key: CTRL+U

## What is XHTML?

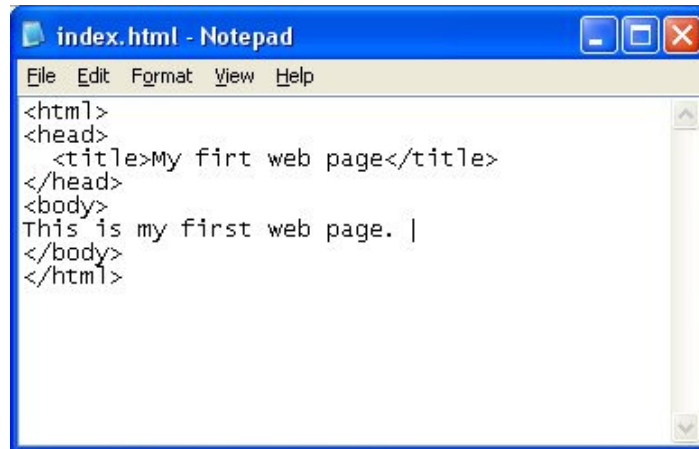
The Extensible Hypertext Markup Language, or XHTML, is a markup language that has the same depth of expression as HTML, but also conforms to XML syntax. XHTML is a stricter and cleaner version of HTML. Differences between HTML and XHTML:

- XHTML elements must be properly nested  
Example: `<p><b>This is a sentence.</b></p>`
- XHTML elements must always be closed  
Example `<p>This text is between the opening and closing tags</p>`  
Another Example `<br /> <hr />`
- XHTML elements must be in lowercase  
Example `<h2>This is a header</h2>`
- XHTML documents must have one root element  
Example `<html></html>`

## Step by step to create your first web page

**Step 1:** Open Windows Notepad or any text editor

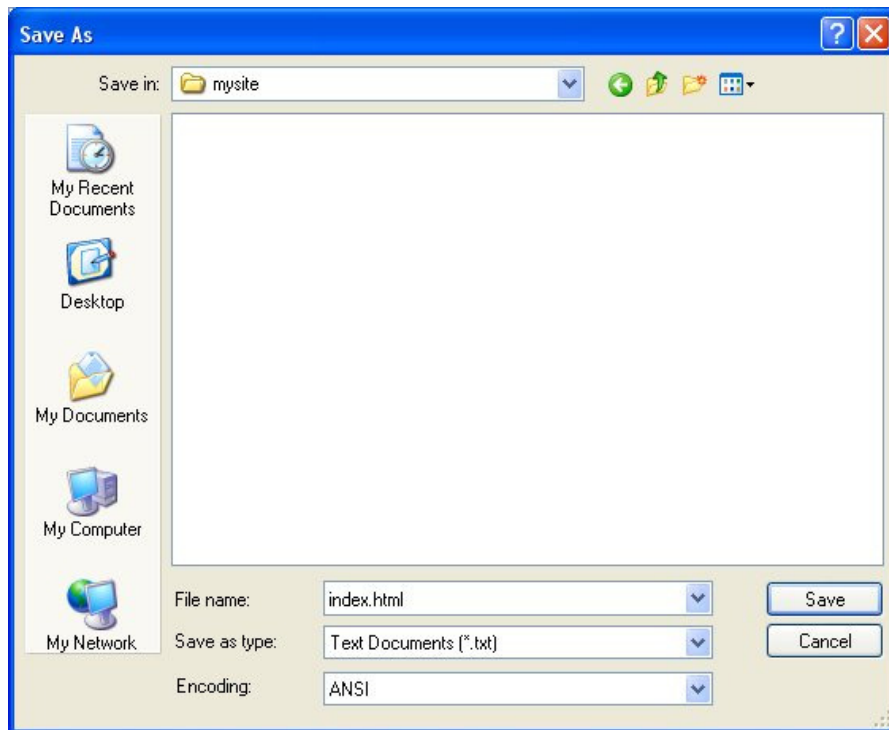
**Step 2:** Type all the text including the HTML tags in the figure below



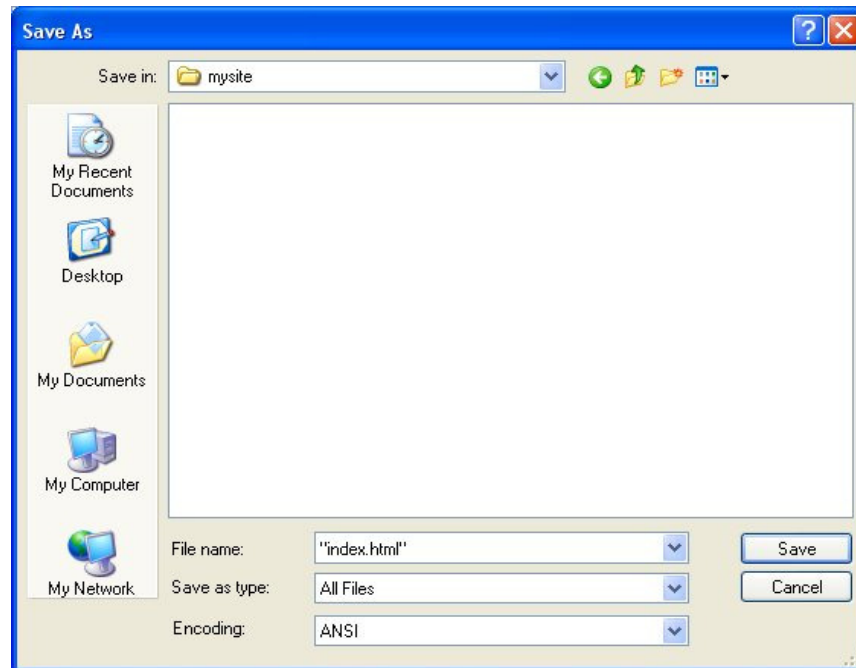
**Here is the HTML code:**

```
<html>
<head><title>My first web page</title></head>
<body>
This is my first web page.
</body>
</html>
```

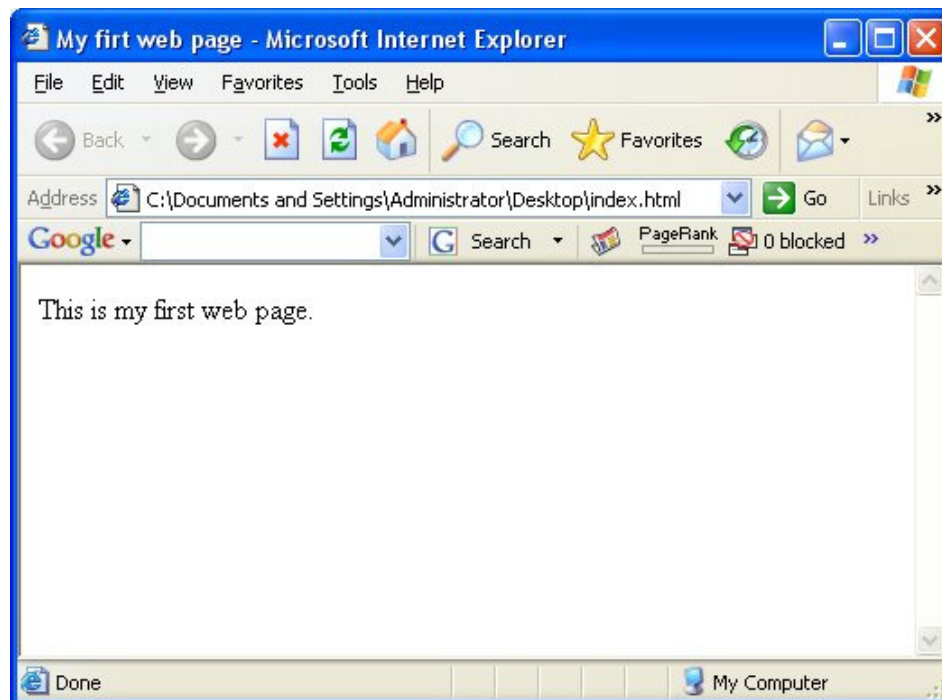
**Step 3:** Be sure to save the file as plain, standard ASCII text. Notepad always saves files as plain text, but you may choose it as an option (after selecting File, Save As) should you use another program.



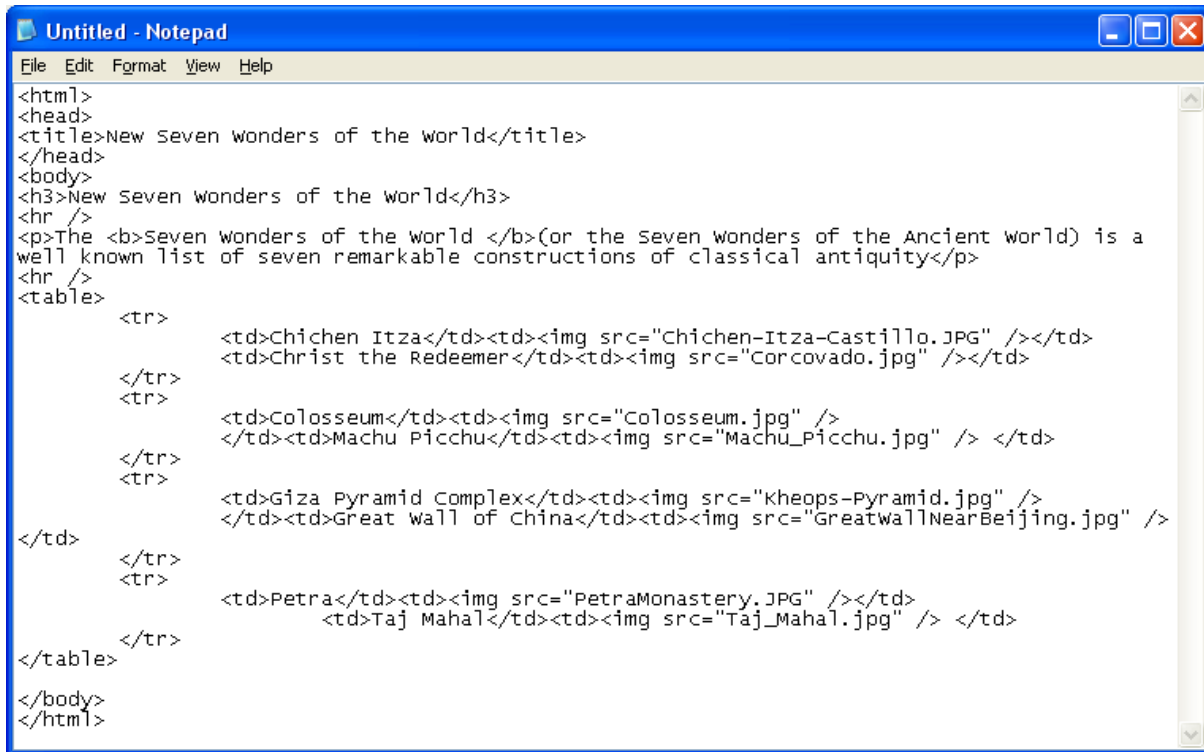
**Step 4:** Always give the filename containing the HTML tags a name ending in .htm (or .html) when you save them. Add double quotes in front of the filename and at the end of the filename for example, "index.html". If you forget to type the .htm or .html at the end of the filename when you save the file, the editor will add .txt or .doc as the file extension. This will cause the web page not to display correctly in a web browser.



**Step 5:** Once the file has been saved, close Notepad and you will see a file called index.html. Double click on the file called index.html and the web browser will open file.



## Example of HTML Code in action

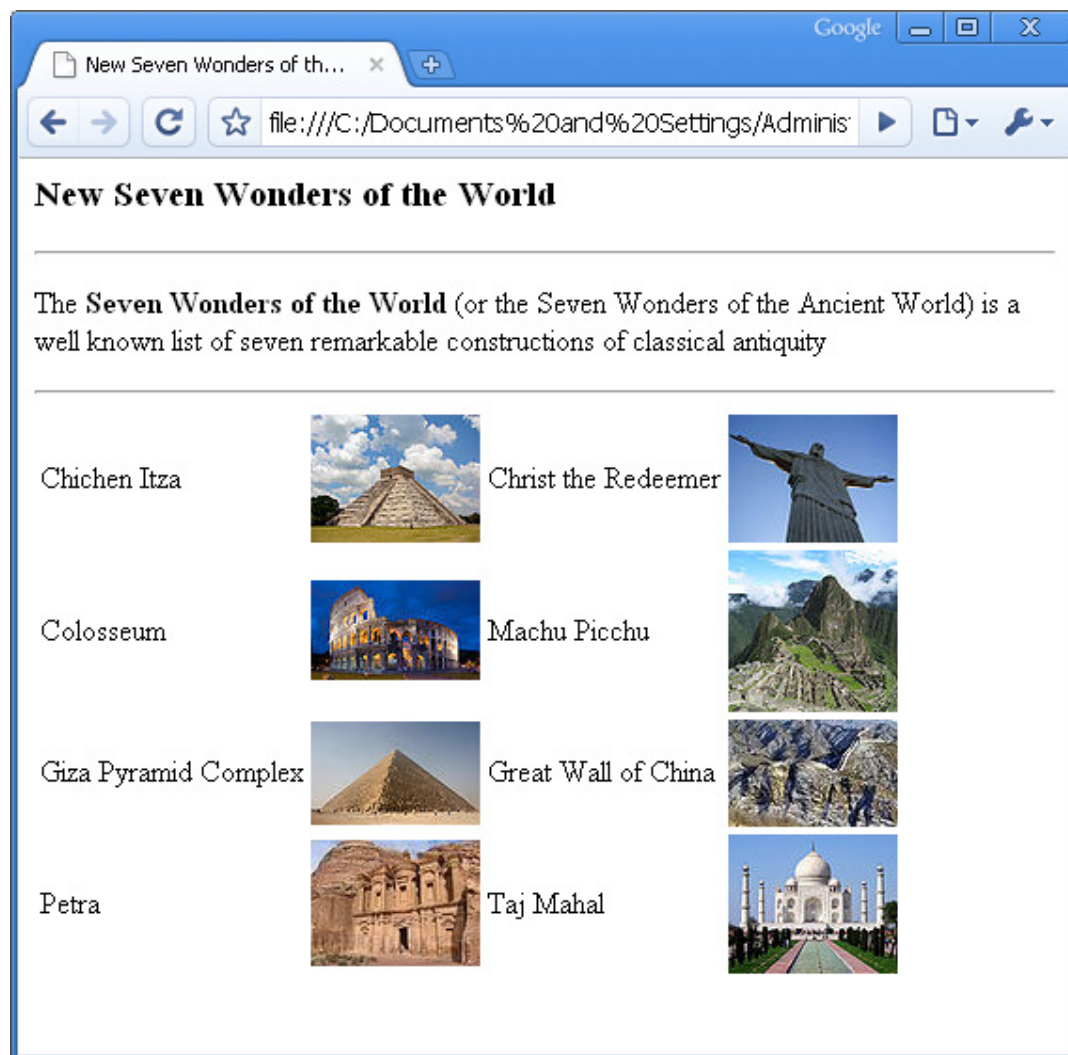


```
<html>
<head>
<title>New Seven Wonders of the world</title>
</head>
<body>
<h3>New Seven Wonders of the world</h3>
<hr />
<p>The <b>Seven wonders of the world </b>(or the Seven wonders of the Ancient world) is a
well known list of seven remarkable constructions of classical antiquity</p>
<hr />
<table>
  <tr>
    <td>Chichen Itza</td><td></td>
    <td>Christ the Redeemer</td><td></td>
  </tr>
  <tr>
    <td>Colosseum</td><td>
    </td><td>Machu Picchu</td><td> </td>
  </tr>
  <tr>
    <td>Giza Pyramid Complex</td><td>
    </td><td>Great wall of China</td><td>
  </tr>
  <tr>
    <td>Petra</td><td></td>
    <td>Taj Mahal</td><td> </td>
  </tr>
</table>
</body>
</html>
```

### Here is the HTML code:

```
<html>
<head>
<title>New Seven Wonders of the World</title>
</head>
<body>
<h3>New Seven Wonders of the World</h3>
<hr />
<p>The <b>Seven Wonders of the World </b>(or the Seven Wonders of the Ancient World) is a well
known list of seven remarkable constructions of classical antiquity</p>
<hr />
<table>
  <tr>
    <td>Chichen Itza</td><td></td>
    <td>Christ the Redeemer</td><td></td>
  </tr>
  <tr>
    <td>Colosseum</td><td>
    </td><td>Machu Picchu</td><td> </td>
  </tr>
  <tr>
    <td>Giza Pyramid Complex</td><td>
    </td><td>Great Wall of China</td><td>
  </tr>
  <tr>
    <td>Petra</td><td></td>
    <td>Taj Mahal</td><td> </td>
  </tr>
</table>
</body>
</html>
```

## The HTML displayed in the web browser



## Effects of the different tags









`<title> New Seven Wonders of the World</title>`

`<h3>New Seven Wonders of the World</h3>`

**New Seven Wonders of the World**

---

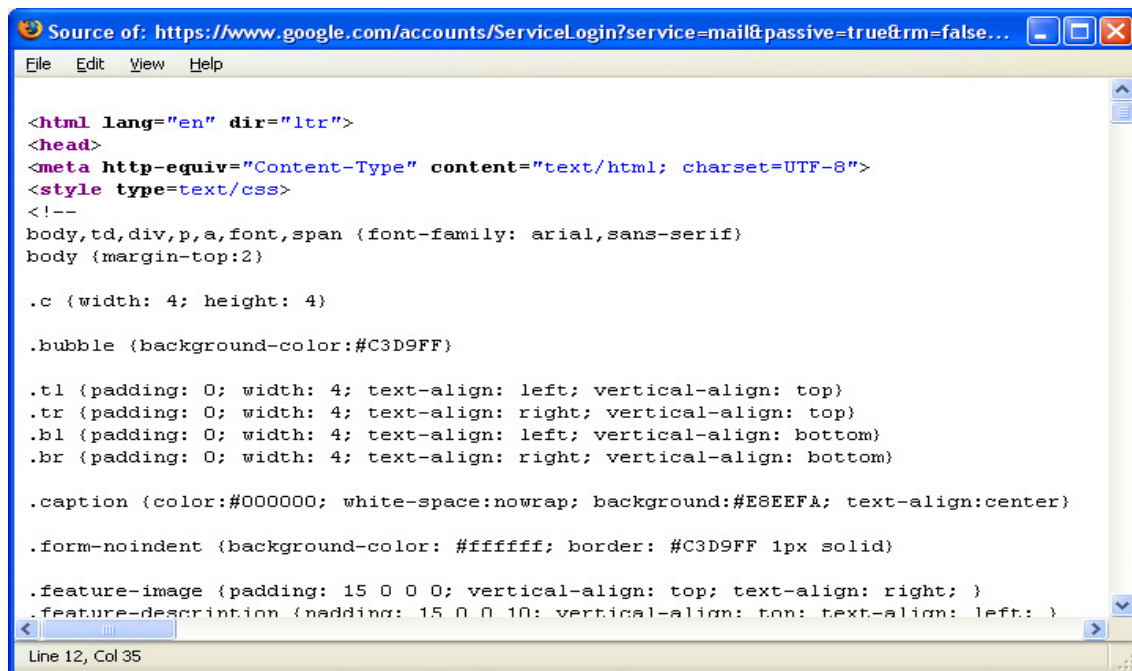
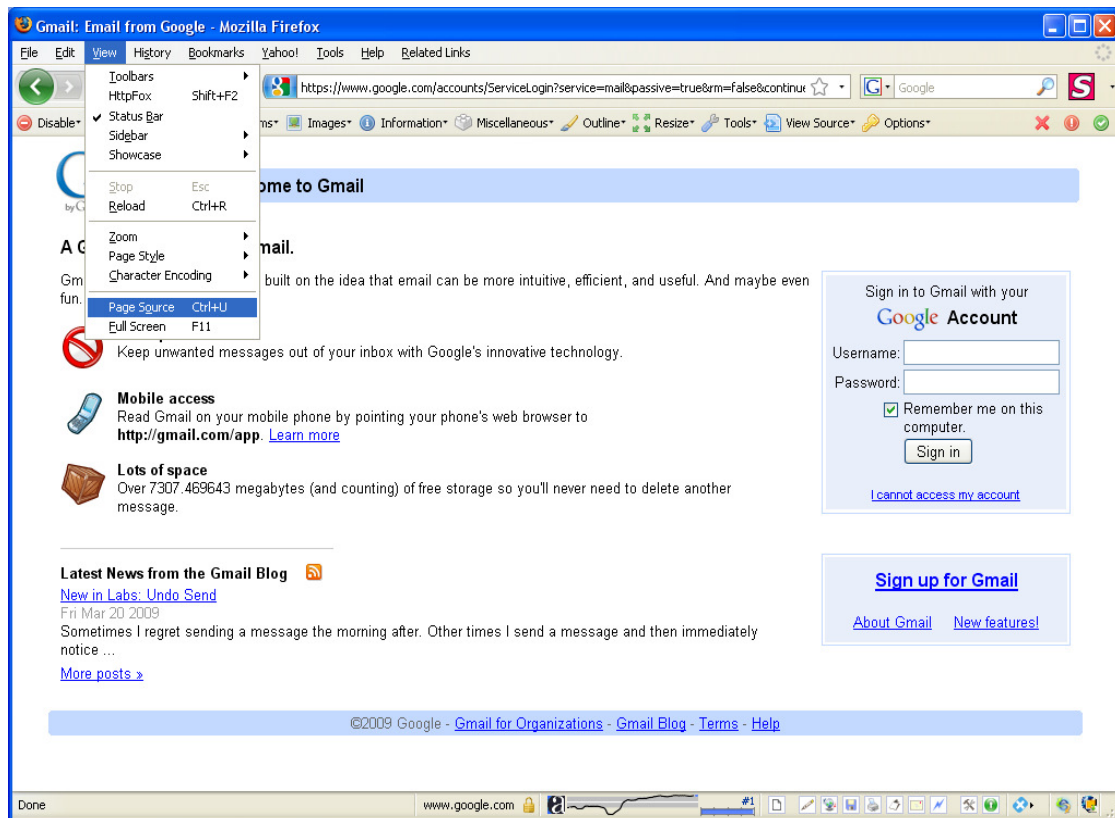
The **Seven Wonders of the World** (or the Seven Wonders of the Ancient World) is a well known list of seven remarkable constructions of classical antiquity

Chichen Itza		Christ the Redeemer	
Colosseum		Machu Picchu	
Giza Pyramid Complex		Great Wall of China	
Petra		Taj Mahal	

`<hr />`

`<p>The <b>Seven Wonders of the World</b> (or the Seven Wonders of the Ancient World) is a well known list of seven remarkable constructions of classical antiquity</p>`

## Example of viewing page source code using Firefox



## Validation of HTML

### What is Markup validation?

The Markup Validation Service by the World Wide Web Consortium (W3C) allows Internet users to check HTML documents for conformance to HTML or XHTML standards. It also provides a quick method for web page authors to check their posted pages for mark-up errors.

### How does it work?


HTML validators operated by comparing the mark-up on a web page to the W3C standards. The standards vary depending upon the declared version and so the validator will start by reading the DOCTYPE declaration to see which set of standards to apply.

Once the validator has read the page and determined the applicable standards it looks for such things as missing opening or closing tags, missing quotation marks and other hand-coding errors.

The validator then provides a report indicating that the coding is correct or not. Errors are noted in a list. One error, such as neglecting to close a tag, can cause a cascade of errors through the page, producing dozens or even hundreds of noted errors. However when the page author addresses the first error listed it will also eliminate the "cascade errors"

### Example of using Markup validation

**Step 1:** We will create a HTML document that uses the HTML 4.01 Transitional DOCTYPE. The web page will contain a header



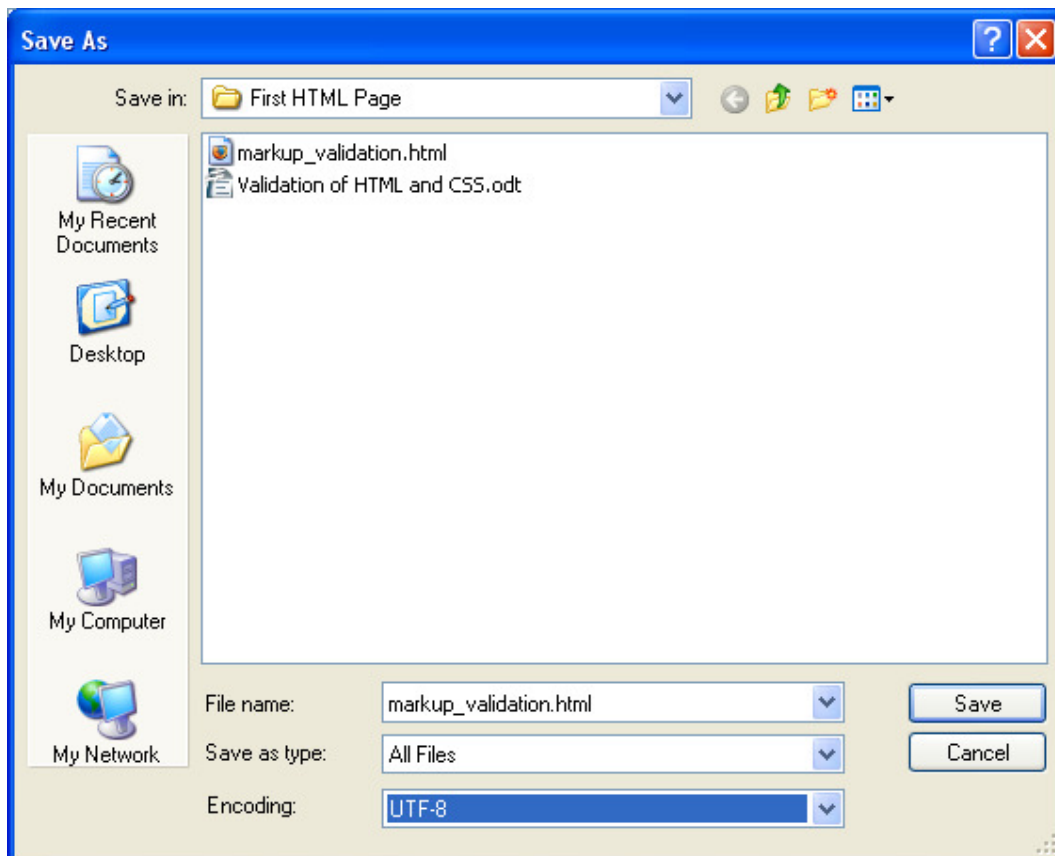
```
markup_validation.html - Notepad
File Edit Format View Help
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>

<head>
<title>Testing markup validation</title>
<meta http-equiv="Content-Type"
content="text/html; charset=utf-8" >
</head>

<body>
<h3>Testing Markup Validation</h3>
<p>The Markup Validation Service by the world wide web
Consortium (W3C)
allows Internet users to check HTML documents for
conformance to HTML
or XHTML standards. It also provides a quick method for
web page authors to check their posted pages for mark-up
errors. </p>
</body>
</html>
```

**Step 2:** Save the document as markup\_validation.html.





**Step 3:** Open the following URL in your web browser:

<http://validator.w3.org/>

The screenshot shows a web browser window with the address bar displaying 'http://validator.w3.org/'. The page title is 'The W3C Markup Validation...'. The main heading is 'W3C® Markup Validation Service' with the subtitle 'Check the markup (HTML, XHTML, ...) of Web documents'. There are three tabs: 'Validate by URI', 'Validate by File Upload', and 'Validate by Direct Input'. The 'Validate by URI' tab is active. It contains the text 'Validate by URI' and 'Validate a document online:'. Below this is a text input field labeled 'Address:'. A link 'More Options' is visible. At the bottom right is a 'Check' button.

**Step 4:** Select "Validate by File Upload"


The screenshot shows the same W3C Markup Validation Service page, but the 'Validate by File Upload' tab is now selected. The heading is 'Validate by File Upload' and the text is 'Upload a document for validation:'. Below this is a file input field labeled 'File:'. The 'Browse...' button next to the file input field is circled in red. A link 'More Options' is visible. At the bottom right is a 'Check' button.

**Step 5:** Select HTML file

To be able to select our html file we will need to click on the **Browse** button and select the HTML file created in Step 1.

## Step 6: Validate the HTML file

To validate the selected HTML file we will click on the **Check** button

**Markup Validation Service**  
Check the markup (HTML, XHTML, ...) of Web documents

[Jump To:](#) [Congratulations](#) · [Icons](#)

**This document was successfully checked as HTML 4.01 Transitional!**

<b>Result:</b>	Passed	
<b>File :</b>	<input type="text"/> <input type="button" value="Browse..."/> <small>Use the file selection box above if you wish to re-validate the uploaded file markup_validation.html</small>	
<b>Encoding :</b>	utf-8	(detect automatically) ▼
<b>Doctype :</b>	HTML 4.01 Transitional	(detect automatically) ▼
<b>Root Element:</b>	HTML	

The following message will appear if our markup conforms to the HTML 4.01 Transitional:

*The uploaded document "markup\_validation.html" was successfully checked as HTML 4.01 Transitional. This means that the resource in question identified itself as "HTML 4.01 Transitional" and that we successfully performed a formal validation using an SGML, HTML5 and/or XML Parser(s) (depending on the markup language used).*

*"valid" Icon(s) on your Web page*

*To show your readers that you have taken the care to create an interoperable Web page, you may display this icon on any page that validates. Here is the HTML you could use to add this icon to your Web page:*



```
<p>
  <a href="http://validator.w3.org/check?uri=referer"></a>
</p>
```



```
<p>
  <a href="http://validator.w3.org/check?uri=referer"></a>
</p>
```

We can add the HTML code to our HTML document created in Step1.



## Markup Validation Service

Check the markup (HTML, XHTML, ...) of Web documents

Jump To: [Congratulations](#) · [Icons](#)

This document was successfully checked as HTML 4.01 Transitional!

Result:	Passed	
File :	<input type="text"/> <input type="button" value="Browse..."/>	
	Use the file selection box above if you wish to re-validate the uploaded file markup_validation.html	
Encoding :	utf-8	(detect automatically) ▼
Doctype :	HTML 4.01 Transitional	(detect automatically) ▼
Root Element:	HTML	

### Options

- ☐ Show Source      ☐ Show Outline      ☒ List Messages Sequentially      ☐ Group Error Messages by Type
- ☐ Validate error pages      ☐ Verbose Output      ☐ Clean up Markup with HTML Tidy

[Help](#) on the options is available.

### Congratulations

The uploaded document "markup\_validation.html" was successfully checked as HTML 4.01 Transitional. This means that the resource in question identified itself as "HTML 4.01 Transitional" and that we successfully performed a formal validation using an SGML, HTML5 and/or XML Parser(s) (depending on the markup language used).

#### "valid" Icon(s) on your Web page

To show your readers that you have taken the care to create an interoperable Web page, you may display this icon on any page that validates. Here is the HTML you could use to add this icon to your Web page:



```
<p>
<a href="http://validator.w3.org/check?uri=referer"></a>
</p>
```



```
<p>
<a href="http://validator.w3.org/check?uri=referer"></a>
</p>
```

A [full list](#) of icons, with links to alternate formats and colors, is available. If you like, you can download a copy of the icons to keep in your local web directory, and change the HTML fragment above to reference your local image rather than the one on this server.

#### Validating CSS Style Sheets

If you use [CSS](#) in your document, you can check it using the W3C [CSS Validation Service](#).

[↑ TOP](#)

[Home](#) [About...](#) [News](#) [Docs](#) [Help & FAQ](#) [Feedback](#) [Contribute](#)



This is the W3C Markup Validator, [v0.8.4](#)  
COPYRIGHT © 1994-2008 W3C® (MIT, ERCIM, KEIO), ALL RIGHTS RESERVED. W3C LIABILITY, TRADEMARK, DOCUMENT USE AND  
SOFTWARE LICENSING RULES APPLY. YOUR INTERACTIONS WITH THIS SITE ARE IN ACCORDANCE WITH OUR PUBLIC AND MEMBER  
PRIVACY STATEMENTS.



## Exercise 1:

Create the following XHTML document and validate the XHTML document. The document needs to conform to XHTML 1.0 Strict.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>My personal information</title>
</head>

<body>
<h3>My personal information</h3>

<table>
<tr><td>Name</td><td>Surname</td></tr>
</table>

<div>
<h4>My Pictures</h4>
</div>

<p>
    <a href="http://validator.w3.org/check?uri=referer">
</a>
</p>

</body>
</html>
```

## ELEMENTS OF A WEB PAGE

### What are the elements of a web page?

After creating a blank web page in the previous section we will need to start adding data onto the web page. This content needs to be presented in its intended format. The presentation will be handled in a specific way. A web page will consist of different functions for example a header, an image, paragraphs, content in tables, and links to other pages. These different functions are the different elements of the web page. All these elements can be defined with the use of HTML tags.

### Text formatting and font control

#### Using the <b> HTML tag

For boldface text, we will put the <b> opening tag at the beginning of the text and the </b> closing tag at the end.

This is a <b>bold</b> text.

#### Using the <i> HTML tag

For italic text, we will put the <i> opening tag at the beginning of the text and the </i> closing tag at the end.

<i>"To be or nor to be!"</i>

#### Using the <u> HTML tag

To underline text, we will put the <u> opening tag at the beginning of the text and the </u> closing tag at the end.

<u>underline</u>

Important tip: Use the <u> tag sparingly since web users expect that underlined text to indicate a link.

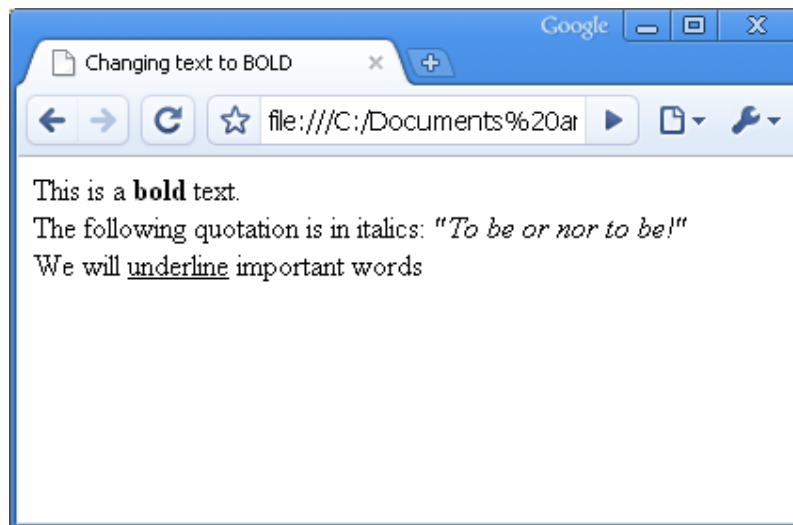
### Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Changing text using Bold, Italics and Underline </title>
</head>

<body>

This is a <b>bold</b> text. <br />
The following quotation is in italics:
<i>"To be or nor to be!"</i> <br />
We will <u>underline</u> important words

</body>
</html>
```



## Changing the Font Size and Font Color

We can make use of the <font> tag to change the presentation of the font used on the web page.

### Changing the font size

We will make use of the size attribute of the <font> tag to specify the font size

<font size="10">font is size is changed</font>

### Changing the font color

We will make use of the color attribute of the <font> tag to specify the font size

<font color="blue">font</font>

### Example

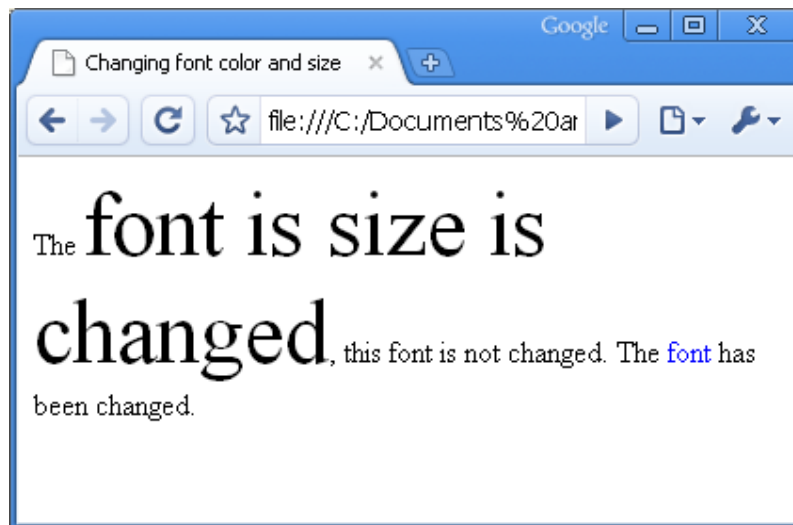
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Changing font color and size </title>
</head>
```

```
<body>
```

The <font size="10">font is size is changed</font>, this font is not changed.

The <font color="blue">font</font> has been changed.

```
</body>
</html>
```



### Choosing the typeface of the font

With the use of the face attribute of the `<font>` tag you can specify the font typeface.

If the web browser does not find the typeface defined it will revert back to the default font typeface.

### Using the `<font face>` to change the typeface

`<font face="Helvetica">`This changes the typeface of the font`</font>`

### Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Choosing the typeface of the font </title>
</head>

<body>
This changes<font face="Helvetica"> the typeface of </font>the font. This
can be
<font face="Courier New, Times New Roman">used to change</font> the font
again.
</body>
</html>
```





## Making use of Special Characters

You can insert special characters into the HTML document by looking up the codes of the character entities.

Should we want to create a copyright (© ) character that can be used on the web page we will make use of the HTML Entity or the code.

Example using HTML entity (Ensure that the & is in front of the entity and the ; is at the end of the HTML entity:

This page is &copy; 2009 SuperSolutions CC

Example using code entity (Ensure that the & is in front of the code and the ; is at the end of the HTML code:

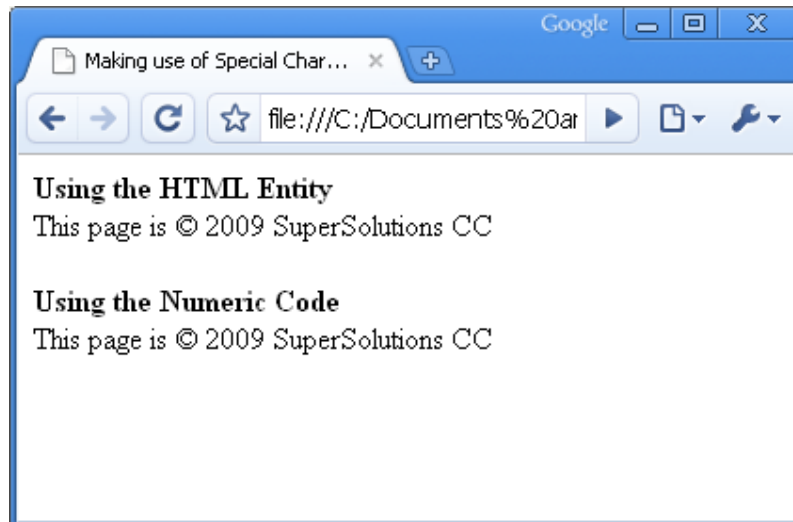
This page is &#169; 2009 SuperSolutions CC

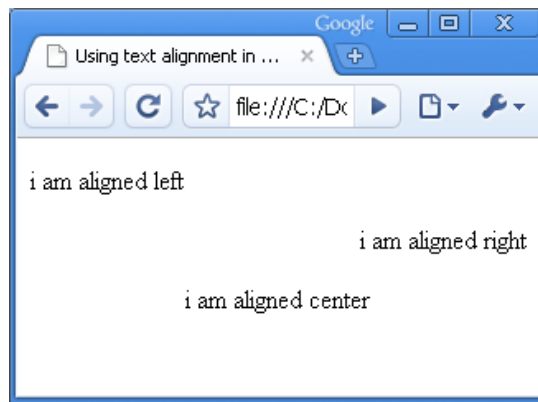
### Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Making use of Special Characters</title>
</head>

<body>
  <b>Using the HTML Entity</b><br/>
  This page is &copy; 2009 SuperSolutions CC<br/><br />
  <b>Using the Numeric Code</b><br/>
  This page is &#169; 2009 SuperSolutions CC
</body>

</html>
```





**Table of some of the special characters**

Sign/ Character	HTML Entity (case sensitive)	Numeric Code	Name or meaning
&	&amp;	&#38;	ampersand
¢	&cent;	&#162;	Cent
©	&copy;	&#169;	copyright
÷	&divide;	&#247;	divide
>	&gt;	&#62;	greater than
<	&lt;	&#60;	less than
™	&trade;	&#153;	trademark

### Aligning text using HTML tags

Some of the HTML tags allows you to align text to the left margin, right margin or center of the page. This is done using the align attribute of the HTML tag to define the placement.

```
<p align="left">i am aligned left</p>
```

#### Example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Using text alignment in your web page</title>
</head>

<body>

<p align="left">i am aligned left</p>
<p align="right">i am aligned right</p>
<p align="center">i am aligned center</p>
</body>
</html>
```

## Using the <h> HTML tag to make use of headings

Headings are used to make the text appear larger and bolder. Usually its used for headings of pages or to emphasize some text on the web page. There are 6 levels available for defining a heading.

We make use of the <h> HTML tag to define headings

```
<h3>Heading</h3>
```

### Example

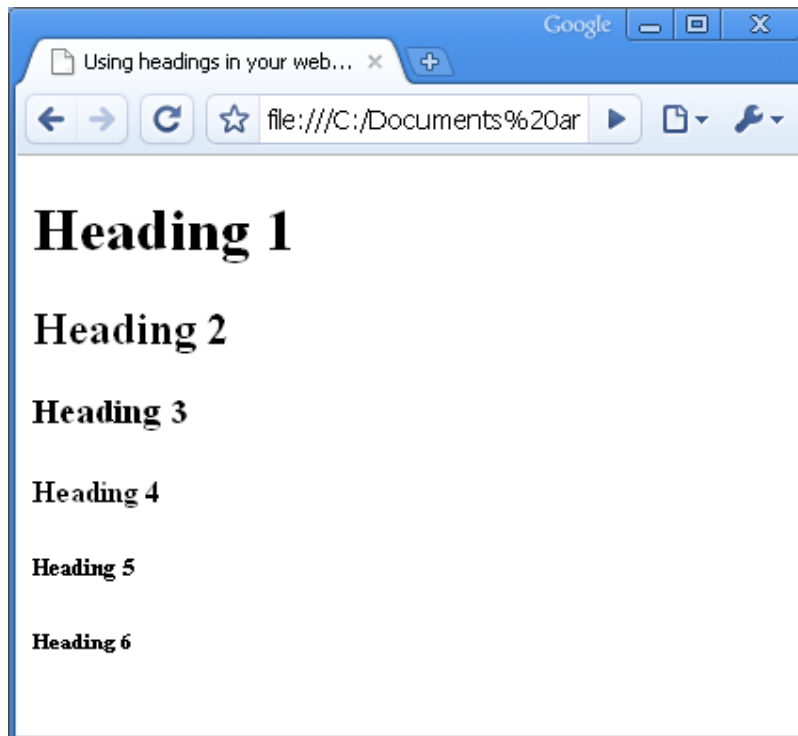
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Using headings in your web page</title>
</head>

<body>

<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>

</body>

</html>
```



## Using the <hr /> HTML tag to make use of horizontal lines

To be able to draw an horizontal lines that will Separate sections of text we will make use of the <hr /> HTML tag. Do note that the <hr /> does not have a closing tag and to conform to XHTML specifications you must add the trailing / to the tag definition.

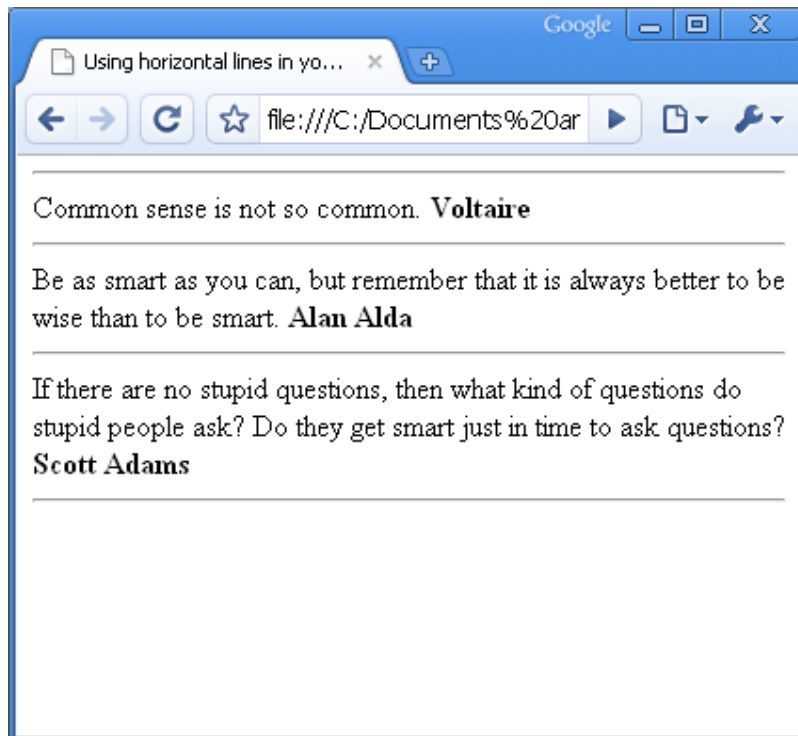
<hr />

### Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Using horizontal in your web page</title>
</head>

<body>

<hr />
Common sense is not so common. <b>Voltaire</b>
<hr />
Be as smart as you can, but remember that it is always better to be
wise than to be smart. <b>Alan Alda </b>
<hr />
If there are no stupid questions,
then what kind of questions do stupid people ask? Do they get smart just in
time to ask questions? <b>Scott Adams </b>
<hr />
</body>
</html>
```



## Using the <br /> HTML tag to make use of line breaks

The <br /> HTML tag forces a line break within a paragraph. Do note that the <br /> does not have a closing tag and to conform to XHTML specifications you must add the trailing / to the tag definition.

<br />

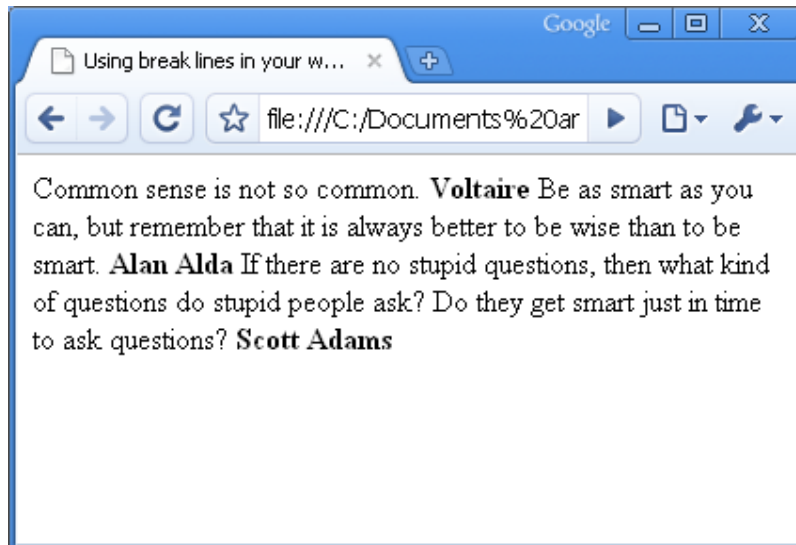
### Example not using break lines

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Using break lines in your web page</title>
</head>

<body>

Common sense is not so common. <b>Voltaire</b>
Be as smart as you can, but remember that it is always better to be
wise than to be smart. <b>Alan Alda </b>
If there are no stupid questions,
then what kind of questions do stupid people ask? Do they get smart just in
time to ask questions? <b>Scott Adams </b>

</body>
</html>
```



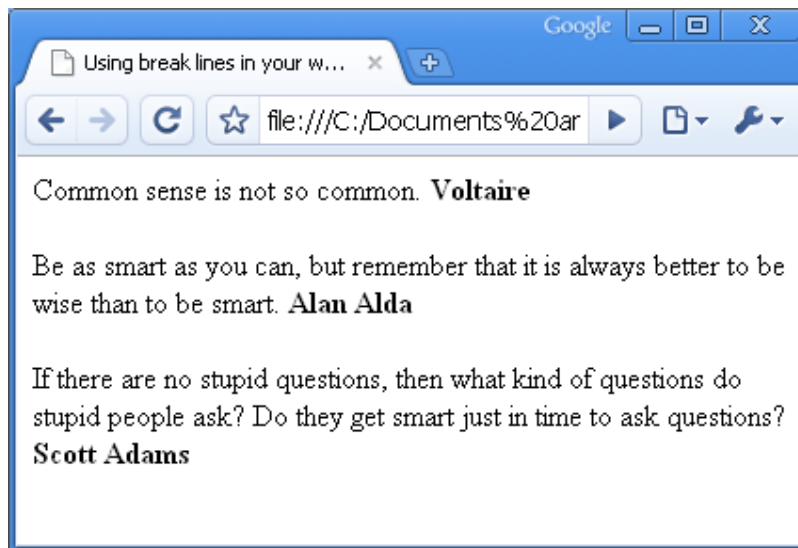
## Example using break lines

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Using break lines in your web page</title>
</head>

<body>

Common sense is not so common. <b>Voltaire</b><br /><br />
Be as smart as you can, but remember that it is always better to be
wise than to be smart. <b>Alan Alda </b><br /><br />
If there are no stupid questions,
then what kind of questions do stupid people ask? Do they get smart just in
time to ask questions? <b>Scott Adams </b>

</body>
</html>
```



## Using the <p> HTML tag to define paragraphs

To define paragraph break in HTML we will make use of the <p> HTML tag. All text that you want to define as part of the paragraph must be between the <p> opening tag and </p> closing tag

<p>Everyone is a genius at least once a year. The real geniuses simply have their bright ideas closer together. </p>

### Example

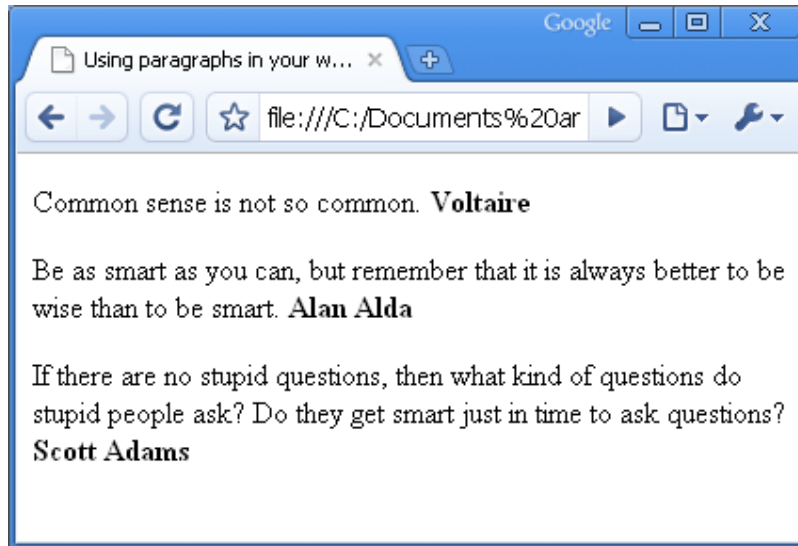
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Using paragraphs in your web page</title>
</head>
```

```

<body>

<p>Common sense is not so common. <b>Voltaire</b></p>
<p>Be as smart as you can, but remember that it is always better to be
wise than to be smart. <b>Alan Alda </b></p>
<p>
If there are no stupid questions,
then what kind of questions do stupid people ask? Do they get smart just in
time to ask questions? <b>Scott Adams </b>
</p>
</body>

```



```

</html>

```

## Making use of lists in HTML

There are 3 basic types of HTML lists. They are:

- The bulleted list is called an unordered list
- The numbered list is called an ordered list
- The definition list is a list of terms and their meanings

All items that form part of the lists need to be between the opening **<li>** HTML tag and closing **</li>** HTML tag. The **<li>** HTML tag needs to be between the defined list opening and closing HTML tags

## Creating the bulleted list with the **<ul>** HTML tags

```

<ul>
<li>One</li>
<li>Two</li>
<li>Three</li>
<li>Four</li>
</ul>

```

## Creating the numbered list with the <ol> HTML tags

```
<ol>
<li>Monday</li>
<li>Tuesday</li>
<li>Wednesday</li>
<li>Thursday</li>
<li>Friday</li>
</ol>
```

## Creating a definition list with the <dl> HTML tag

It starts with the opening HTML tag **<dl>** and closes with the closing **</dl>** HTML tag. The opening **<dt>** HTML tag goes in front of each term to be defined, with a **<dd>** tag in front of each definition. Line break and indentation appears automatically.

```
<dl>
  <dt>Asset</dt>
  <dd>A useful or valuable thing or person</dd>

  <dt>Network</dt>
  <dd>In information technology, a network is a series of points
      or nodes interconnected by communication paths. Networks can
interconnect with other networks and contain subnetworks.
  </dd>
</dl>
```

## Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Using lists</title>
</head>

<body>

<h3>Unordered List</h3>
<ul>
<li>One</li>
<li>Two</li>
<li>Three</li>
<li>Four</li>
</ul>

<h3>Ordered List</h3>
<ol>
<li>Monday</li>
<li>Tuesday</li>
<li>Wednesday</li>
<li>Thursday</li>
<li>Friday</li>
</ol>

<h3>Definition List</h3>
<dl>
  <dt>Asset</dt>
  <dd>A useful or valuable thing or person</dd>

  <dt>Network</dt>
```



```

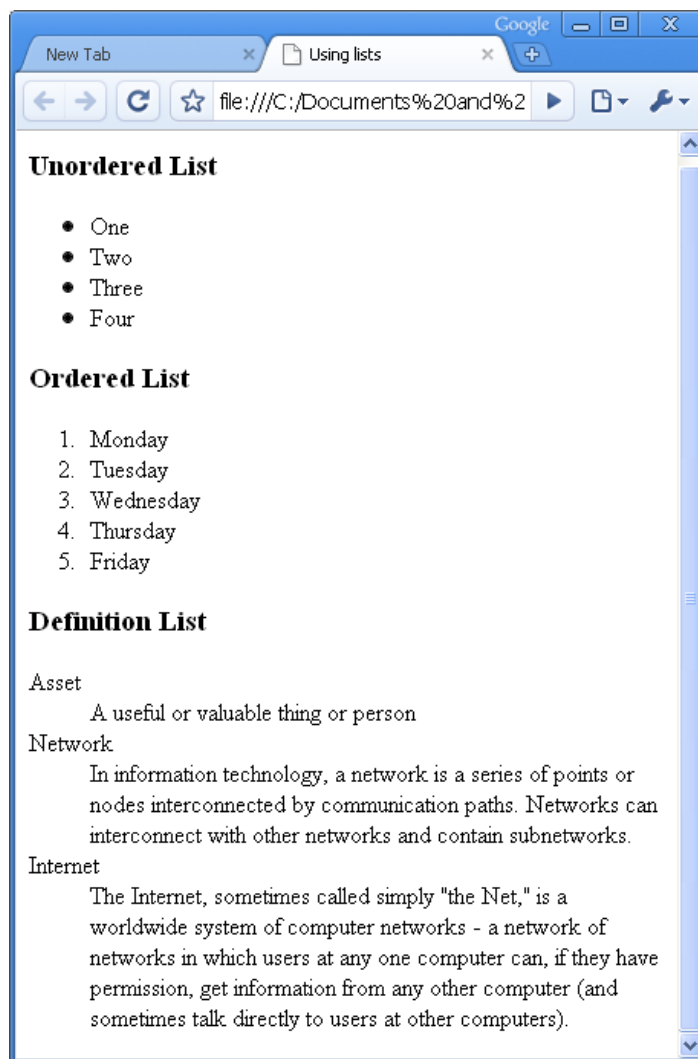
<dd>In information technology, a network is a series of points
    or nodes interconnected by communication paths. Networks can
interconnect with    other networks and contain subnetworks.
</dd>

<dt>Internet</dt>
<dd>The Internet, sometimes called simply "the Net," is a worldwide system
    of computer    networks -
    a network of networks in which users at any one computer can,
    if they have permission, get information from any other computer
        (and sometimes talk directly to users at other computers).
</dd>

</dl>

</body>
</html>

```



## Nested Lists

One can create lists within lists hence ordered and unordered lists can be nested inside one another, down as many levels as you want.

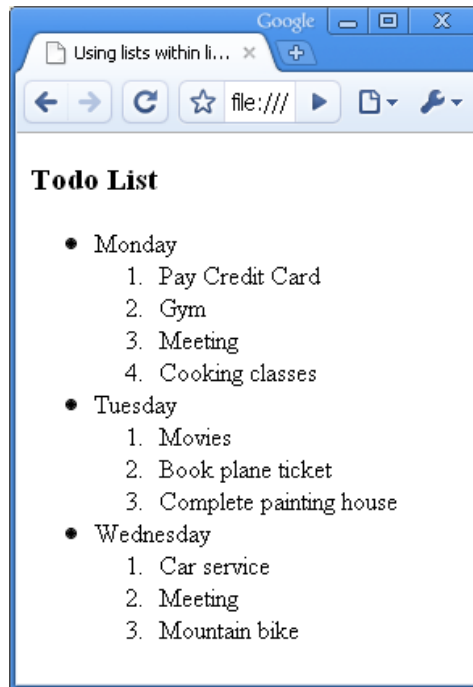
### Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Using lists within lists</title>
</head>

<body>

<h3>Todo List</h3>
<ul>
<li>Monday
    <ol>
        <li>Pay Credit Card</li>
        <li>Gym</li>
        <li>Meeting </li>
        <li>Cooking classes</li>
    </ol>
</li>
<li>Tuesday
    <ol>
        <li>Movies</li>
        <li>Book plane ticket</li>
        <li>Complete painting house</li>
    </ol>
</li>
<li>Wednesday
    <ol>
        <li>Car service</li>
        <li>Meeting </li>
        <li>Mountain bike </li>
    </ol>
</li>
</ul>

</body>
</html>
```



## Linking to other pages

A hyperlink is a reference to a resource on the web. Hyperlinks can point to any resource on the web: an HTML page, an image, a sound file, a movie, etc. An anchor is a term used to define a hyperlink destination inside a document.

To be able to make use of hyperlinks we will make use of the **<a>** HTML tag.

### Using the **<a>** HTML Tag to link to other resources:

`<a href="url">Link text</a>`

The element part (Link text) will be displayed on the HTML document, this can also be an image. The **href** attribute defines where the web user will be directed once the link has been clicked.

- **<A>** Opening tag and stands for **A**ncor. It begins the link to another page.
- **HREF** stands for **H**ypertext **REF**erence. That's a nice, short way of saying to the browser, "This is where the link is going to go."
- **url** is the FULL ADDRESS of the link for example `http://www.google.com`
- Where it reads **"link text"** is where you write the text you want to appear on the page. What is in that space will appear on the page for the viewer to click.
- **</A>** closing tag for the anchor

Example of linking to other web sites:

`<a href="http://www.google.com" >Google</a>`

Example of linking to another web page:

`<a href="contact.html" >Click here to go to Contact Us page</a>`

## Using an anchor to define a hyperlink destination inside a document.

We can also create hyperlinks that will link to specific section inside a defined document.

Step 1: Define the section with the **name** attribute in the document

```
<a name="resources">Resources</a>
```

Step 2: Define the `<a>` HTML tag in the document and link to the name attribute defined in Step 1

```
<a href="home.html#resources"> Click here to view Resources</a>
```

### Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Linking to other pages </title>
</head>

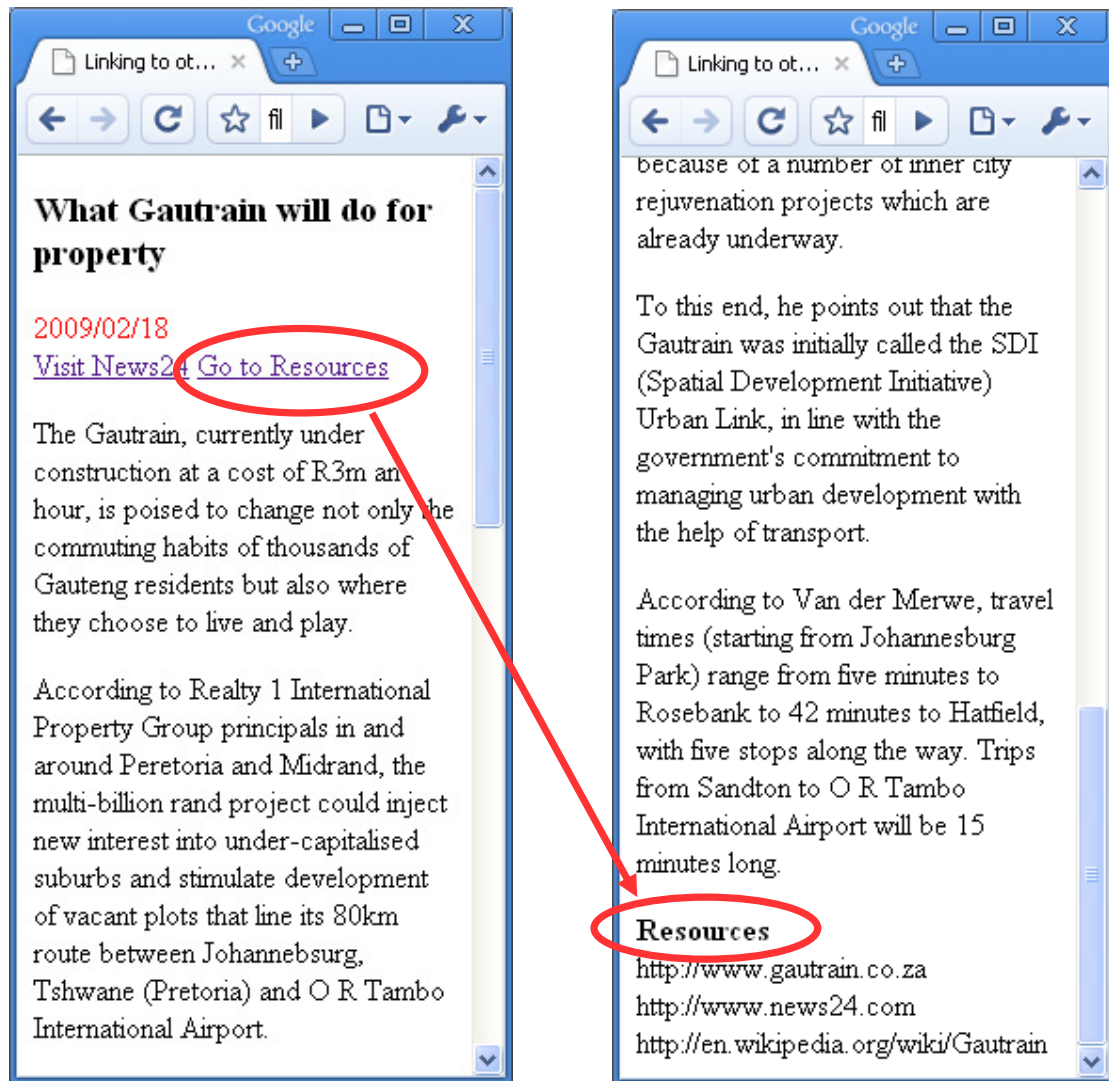
<body>
<h3>
What Gautrain will do for property
</h3>
<font color="red">2009/02/18</font><br />
<a href="http://www.news24.com">Visit News24</a>
<a href="linking.html#resources">Go to Resources</a>

<p>
The Gautrain, currently under construction at a cost of R3m an hour, is
poised to change not only the commuting habits of thousands of Gauteng
residents but also where they choose to live and play.
</p>
<p>
According to Realty 1 International Property Group principals in and around
Peretoria and Midrand, the multi-billion rand project could inject new
interest into under-capitalised suburbs and stimulate development of vacant
plots that line its 80km route between Johanneburg, Tshwane (Pretoria) and
O R Tambo International Airport.
</p>
<p>
While buying and new development activity has slowed in most parts of South
Africa, the suburbs dotted along the rapid rail network are set to become
high-density, economic cores, says Jack van der Merwe CEO and project
Leader of the Gautrain Management Agency. This is not only because the new
system will offer a cost-effective alternative to private commuting on
heavily congested link roads, but also because of a number of inner city
rejuvenation projects which are already underway.
</p>
<p>
To this end, he points out that the Gautrain was initially called the SDI
(Spatial Development Initiative) Urban Link, in line with the government's
commitment to managing urban development with the help of transport.
</p>
<p>
According to Van der Merwe, travel times (starting from Johannesburg Park)
range from five minutes to Rosebank to 42 minutes to Hatfield, with five
```

stops along the way. Trips from Sandton to O R Tambo International Airport will be 15 minutes long.

```
<a name="resources"><b>Resources</b></a><br />
http://www.gautrain.co.za<br />
http://www.news24.com<br />
http://en.wikipedia.org/wiki/Gautrain

</body>
</html>
```



## Linking your email address into a web page

Web users can click on the clickable link that contains your email address and then send you email.

To be able to create an HTML link to your email address we will use the **<a>** HTML tag and edit the format of the **href** attribute. We need to add the mailto parameter and the email address.

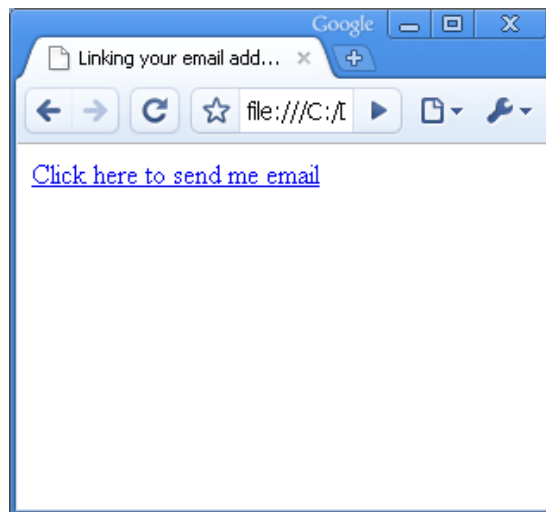
```
<a href="mailto:admin@schools.com">Click here to send me email</a>
```

### For example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Linking your email address into you web page</title>
</head>

<body>
<a href="mailto:admin@schools.com">Click here to send me email</a>

</body>
</html>
```



## Exercise 2

Create a web site with 5 pages that will be used as your CV. Your home page will have be used to introduce yourself also include a picture of yourself. Your Home Page will then link to four other web pages: Ensure that all the pages are linked together.

- Work Experience
- Education
- Personal Details
- References

## WEB PAGE DESIGN

### Using tables for layout in HTML

A table is an orderly arrangement of text and graphics into vertical and horizontal rows. Tables are defined with the <table> HTML tag. The rows of data needs to be defined between the opening <table> HTML tag and the closing </table> HTML tag. A table is divided into rows with the use of the <tr> HTML tag, and each row is divided into data cells with the use of the <td> HTML tag. A data cell can contain text, images, lists, paragraphs, forms, horizontal rules, tables, etc.

```
<table>
<tr>
<td></td><td></td>
</tr>
</table>
```

### Creating a 2 by 3 Table using HTML

**Step 1:** Create a wireframe of the layout

Row 1, cell 1	Row 1, cell 2
Row 2, cell 1	Row 2, cell 2
Row 3, cell 1	Row 3, cell 2

**Step 2:** Create the HTML code for the table

```
<table>
<tr>
<td>Row 1, cell 1</td><td>Row 1, cell 2</td>
</tr>
<tr>
<td>Row 2, cell 1</td><td>Row 2, cell 2</td>
</tr>
<tr>
<td>Row 3, cell 1</td><td>Row 3, cell 2</td>
</tr>
</table>
```

### Final version:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Creating a table in HTML</title>
</head>

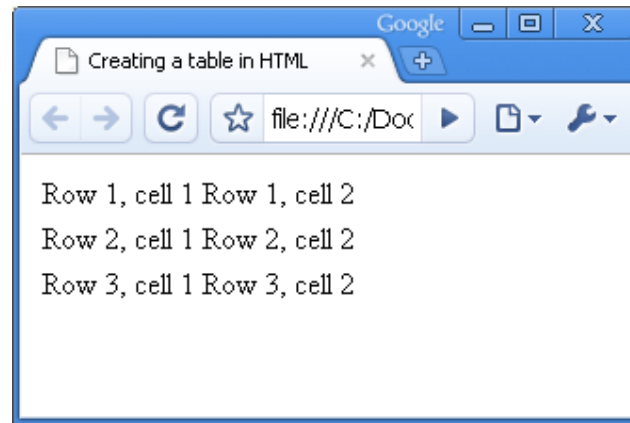
<body>

<table>
<tr>
<td>Row 1, cell 1</td><td>Row 1, cell 2</td>
</tr>
<tr>
<td>Row 2, cell 1</td><td>Row 2, cell 2</td>
</tr>
```

```

<tr>
<td>Row 3, cell 1</td><td>Row 3, cell 2</td>
</tr>
</table>
</body>
</html>

```



### Formatting the table using the attributes of the <table> HTML tag

We will use the **width** and **height** attribute to control the exact size of the entire table.

```
<table width="450" height="400">
```

To control the background color of the table we will use the **bgcolor** attribute

```
<table bgcolor="#aeaeae">
```

We can also control the space around the borders of a table with the **cellpadding** and **cellspacing** attributes. The **cellspacing** attribute set the amount of space between table borders and between table cells. The **cellpadding** attribute set the amount of space around the edges of information in the cell

```
<table cellpadding="1" cellspacing="1">
```

To add a border to the table we will use the **border** attributes. A value of 0 for the **border** attribute will not apply the border to the table.

```
<table border="1">
```

We can align the content of a cell with the **align** and **valign** attributes. Content can be horizontally aligned left, right or center using the **align** attribute. Content can also be vertically aligned using the **valign** attribute, the values can be top, bottom or middle.



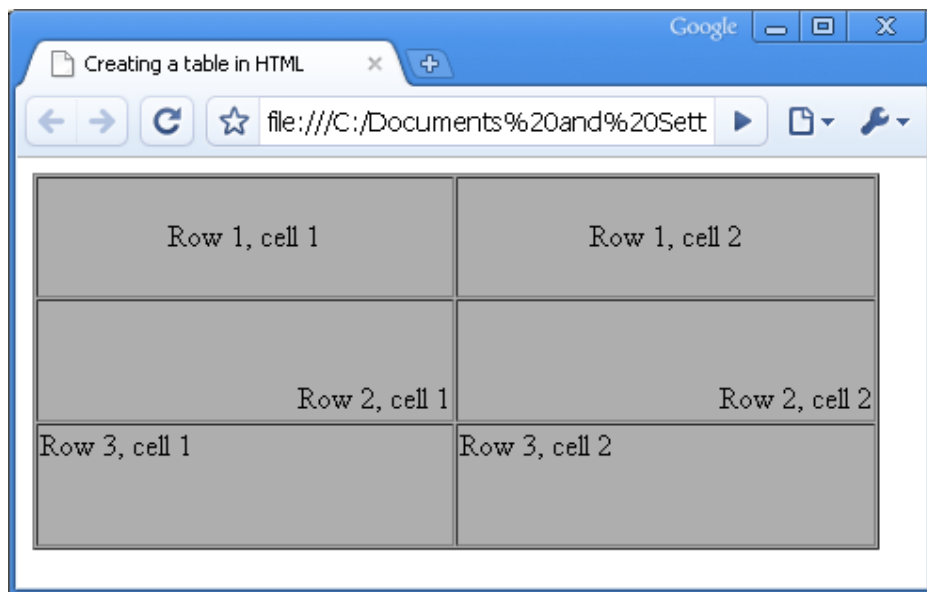
## Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Creating a table in HTML</title>
</head>

<body>

<table width="450" height="200" border="1" bgcolor="#aeeeee"
cellpadding="1" cellspacing="1">
<tr valign="middle" align="center">
<td>Row 1, cell 1</td><td>Row 1, cell 2</td>
</tr>
<tr valign="bottom" align="right">
<td>Row 2, cell 1</td><td>Row 2, cell 2</td>
</tr>
<tr valign="top" align="left">
<td>Row 3, cell 1</td><td>Row 3, cell 2</td>
</tr>
</table>

</body>
</html>
```



## Creative tables using colspan and rowspan attributes

The **colspan** attribute is used to define the number of columns the cell will span. The **rowspan** attribute is used to define the number of rows the cell will span.

### Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Creating a table in HTML</title>
</head>

<body>

<table border="1" cellpadding="1" cellspacing="1">

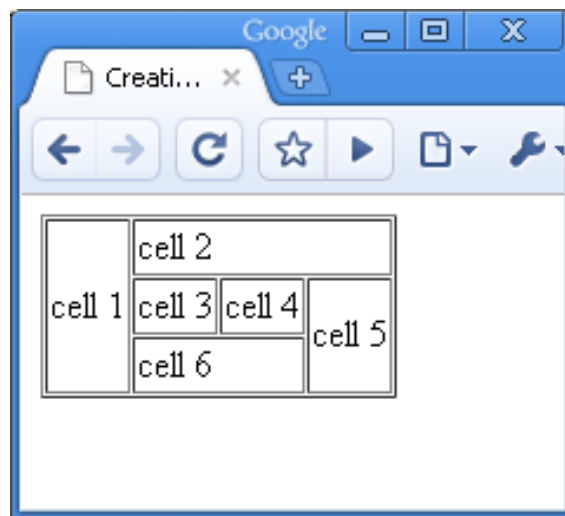
<tr>
<td rowspan="3">cell 1</td>
<td colspan="3">cell 2</td>
</tr>

<tr>
<td>cell 3</td>
<td>cell 4</td>
<td rowspan="2">cell 5</td>
</tr>

<tr>
<td colspan="2">cell 6</td>
</tr>

</table>

</body>
</html>
```



A screenshot of a web browser window with a blue title bar and address bar. The browser displays a table with the following structure:

cell 1	cell 2		
	cell 3	cell 4	cell 5
	cell 6		

## Forms and HTML

Forms are used to allow users to submit data. Based on this data that was entered the website will then provide specific information back to the user. XHTML provides a mechanism called a form to collect the user data.

Usually the data is sent to a script that will capture the data entered and then run the script on the web server and send the result back. The **<form>** HTML tag has two important attributes that allow that the data is sent to a specified resource and in a specified way. The **action** attribute specifies the URL of a script. The **method** attribute defines how the form's data is sent to the web server. The **method** attribute has two possible values: **post** is used to append form data to the browser request, the **get** method appends the form data directly to the end of the URL.

The **<input>** HTML tag will add a form element that can be represented as a text box, password text box, check box, radio button, submit button, reset button, hidden input field, image (which acts as a submit button), file selection box or general button. Remember that the **<input>** HTML tag ends with a **/** to conform to XHTML standards. That means the **<input>** HTML tag does not have an ending tag. The **type** attribute will define which component will be created on the form.

### Creating forms in HTML

**Step 1:** Create a basic form

```
<form action="forms_plain.html" method="get">

</form>
```

**Step 2:** Add components to the form using the **<input>** attribute.

The attribute **type** defines a textfield, the **size** attribute defines that the textfield has a size value of 15 characters. The **maxlength** attribute will allow the user to enter a maximum of 10 characters.

```
Name <input type="text" name="name" size="15" maxlength="10" />
```

**Step 3:** Add a button to submit data entered and a reset button to clear data entered.

The button that will send the data to the specified URL is defined as submit hence the **type** attribute is assigned the value of **submit**. The **name** attribute is used to define the name of the HTML component and the **value** attribute defines the text that must appear on the button.

The button that will clear all the data entered into the form is defined as reset hence the **type** attribute is assigned the value of **reset**. The **value** attribute defines the text that must appear on the button.

```
<input type="submit" name="submit" value="Send data" />
<input type="reset" value="Clear form" />
```

### Final version

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Using forms in HTML</title>
</head>

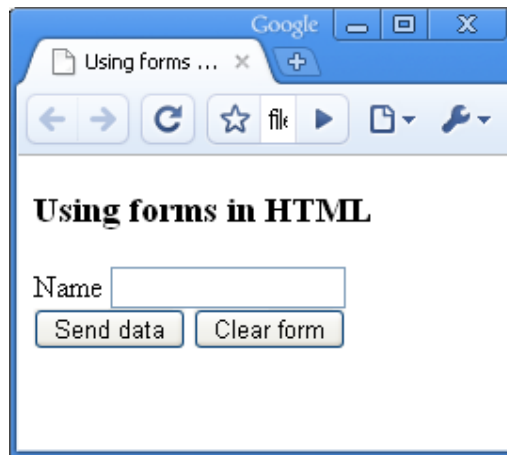
<body>
```

```

    <h3>Using forms in HTML</h3>
    <form action="forms_plain.html" method="get">
      Name <input type="text" name="name" size="15" maxlength="10" /><br />
      <input type="submit" name="submit" value="Send data" />
      <input type="reset" value="Clear form" />
    </form>

  </body>
</html>

```



### Creating a text area on a HTML form

The `textarea` is a multi-row text area form element. The `<textarea>` HTML tag have an opening and a `</textarea>` closing tag. The **rows** attribute defines the number of rows the text area will have. The **cols** attribute defines the number of columns the text area will have. The **name** attribute is used for identification purposes. The initial value of the text area can be placed in between the opening and closing tags.

#### Step 1: Create a basic form

```

<form action="forms.html" method="get">

</form>

```

#### Step 2: Add the text area

```

<textarea name="comments" rows="10" cols="30">
enter your comments here.
</textarea>

```

### Final Version:

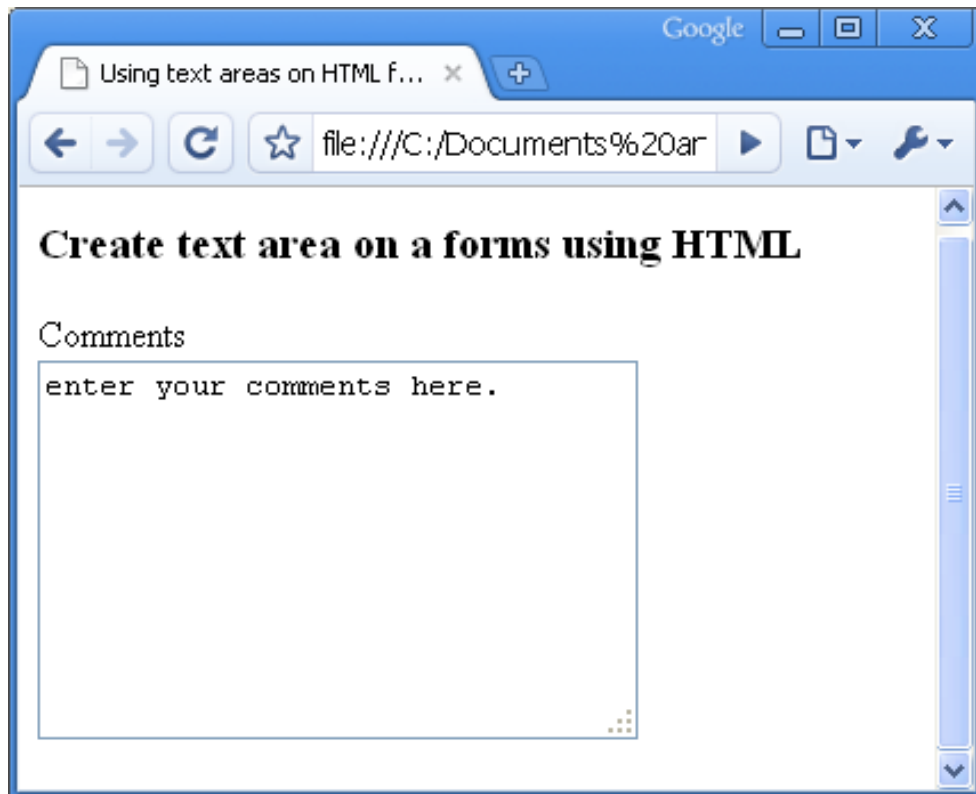
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Using text areas on HTML forms</title>
</head>

<body>
  <h3>Create text area on a forms using HTML</h3>

  <form action="forms.html" method="get">
Comments<br/>
<textarea name="comments" rows="10" cols="30">
enter your comments here.
</textarea>

</form>

</body>
</html>
```



## Using the <select> HTML tag to create a drop down list on a HTML form

The <select> HTML tag is used for a drop-down list form element. The <option> HTML tag elements within the select element define each list item. The <option> HTML tag must be between the opening <select> HTML tag and the closing </select> HTML tag. The data that must be displayed on the drop down list must be placed between the opening <option> HTML tag and closing </option> HTML tag. The **value** attribute for the <option> HTML tag provides the value of the selected element.

### Step 1: Create a basic form

```
<form action="forms.html" method="get">  
  
</form>
```

### Step 2: Add the drop down list

```
<select name="months">  
<option value="jan">January</option>  
<option value="feb">February</option>  
<option value="mar">March</option>  
<option value="apr">April</option>  
<option value="may">May</option>  
<option value="jun">June</option>  
<option value="jul">July</option>  
<option value="aug">August</option>  
<option value="sep">September</option>  
<option value="oct">October</option>  
<option value="nov">November</option>  
<option value="dec">December</option>  
</select>
```

### Final version:

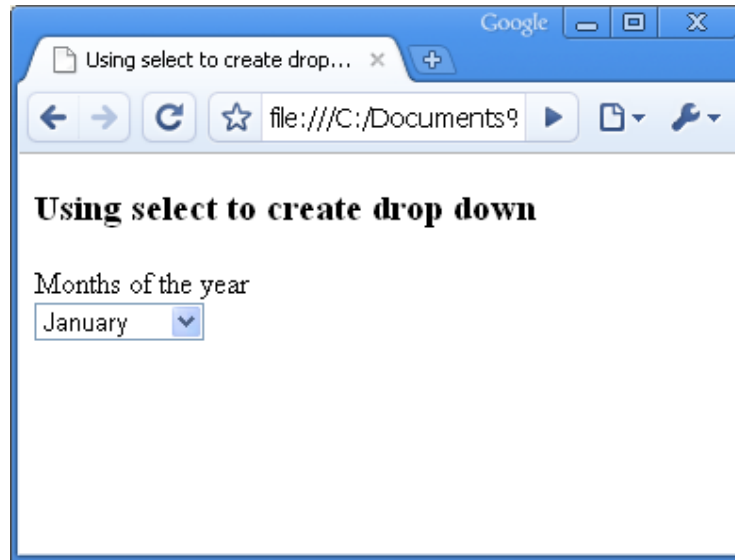
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"  
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
  <title>Using select to create drop down list on HTML forms</title>  
</head>  
  
<body>  
  <h3>Using select to create drop down</h3>  
  
  <form action="forms.html" method="get">  
    Months of the year<br/>  
  
    <select name="comments" name="months">  
      <option value="jan">January</option>  
      <option value="feb">February</option>  
      <option value="mar">March</option>  
      <option value="apr">April</option>  
      <option value="may">May</option>  
      <option value="jun">June</option>  
      <option value="jul">July</option>  
      <option value="aug">August</option>  
      <option value="sep">September</option>  
      <option value="oct">October</option>  
      <option value="nov">November</option>
```

```

<option value="dec">December</option>
</select>

</form>
</body>
</html>

```



### Creating a feedback form for your website.

We will create a form that will capture the feedback of a user of the web site. The following information will be captured, I have indicated the element that will be used for that function:

- Name of user (text field)
- Email address if we need to follow up on comments (text field)
- A rating of the user on a scale between 1 and 10 (drop down list)
- The comments of the user (text area)
- A few aspects that the user liked (check box)
- Option to send the user a newsletter (radio)
- A button to send data (submit button)
- A button to clear data entered (reset button)

### Final version:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Feedback</title>
</head>

<body>
  <h3>Feedback From</h3>

  <form action="feedback.php" method="post">
    Name <input type="text" name="name" size="15" maxlength="150" /><br />
    Email <input type="text" name="email_address" size="15" maxlength="150" /><br />

```

```

Rating of the site: (10- Excellent, 1 - Poor) <br/>
<select name="rating">
<option value="1">1</option>
<option value="2">2</option>
<option value="3">3</option>
<option value="4">4</option>
<option value="5">5</option>
<option value="6">6</option>
<option value="7">7</option>
<option value="8">8</option>
<option value="9">9</option>
<option value="10">10</option>
</select>
<br />
Comments<br/>
<textarea name="comments" rows="5" cols="30">
Enter your comments here.
</textarea>
<br />
<b>Things you liked about the site</b><br />
Speed of the site<input type="checkbox" name="liked" value="speed" checked />
Functionality of the site<input type="checkbox" name="liked" value="functions" />
Design of the site<input type="checkbox" name="liked" value="design" />
<br/>
<b>would you like to receive newsletters</b><br />
<input type="radio" name="newsletter" value="yes" checked />Yes <br />
<input type="radio" name="newsletter" value="no" />No

<br />
<input type="submit" name="submit" value="Send data" />
<input type="reset" value="Clear form" />

</form>
</body>
</html>

```

**Feedback From**

Name

Email

Rating of the site: (10- Excellent, 1 - Poor)

1

Comments

Enter your comments here.

**Things you liked about the site**

Speed of the site ☒ Functionality of the site ☐ Design of the site ☐

**would you like to receive newsletters**

☒ Yes ☐ No



## Using Frames to create a multipage layout

With frames, you can display more than one HTML document in the same browser window. Each HTML document is called a frame, and each frame is independent of the others. Frames are like tables in that they allow you to arrange text and graphics into rows and columns.

### Creating a Frameset document

A Frameset document contains no content. It defines how the other pages will be loaded and how to arrange them in the browser window. The Frameset document will also specify which web page should be displayed in each frame.

To create the Frameset document we will make use of the `<frameset>` HTML tag. The `rows` or `cols` attribute is used to define the layout of the frameset. The `rows` attribute defines the number of rows the frameset will have. The values assigned to the `rows` attribute can be a precise pixel value, percentage of the total size of the window or use the asterisk (\*) to indicate that a frame should fill whatever space is available in the window. The number of regions can be defined with the use of a comma (,)

Note that the opening `<frameset>` HTML tag and the closing `</frameset>` HTML tag replaces the opening `<body>` HTML tag and closing `</body>` HTML tag in the document that defines the frameset document.

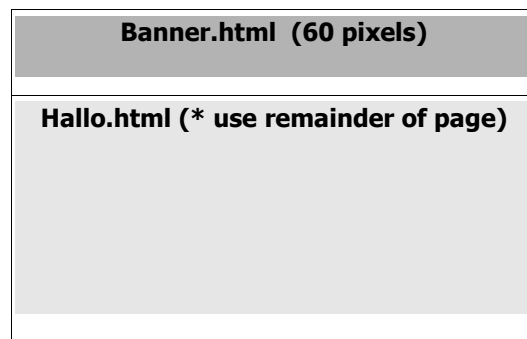
The `<noframes>` HTML tag is used to display alternative content if the web browser does not support frames.

### Example of using rows attribute

The following HTML code will divide the frameset into two rows. The one region will be allocated 60 pixels of web page space, this region will be occupied with the content of `banner.html`. The other region will then be filled with whatever space is available and the region will be filled with the content of `hello.html`.

```
<frameset rows="60,*">
  <frame src="banner.html" name="Banner" />
  <frame src="hello.html" name="Hello" />
</frameset>
```

### Basic wireframe layout of the `Basic_rows_frames.html` frameset document.



## Basic\_rows\_frames.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Basic Frames and HTML </title>
</head>

<frameset rows="60,*">
  <frame src="banner.html" name="Banner" />
  <frame src="hello.html" name="Hello" />
</frameset>

<noframes>
<body>
  Your browser does not support Frames.
</body>
</noframes>
</html>
```

## Hallo.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Hello</title>
</head>

<body>

<h2>Welcome to Frames</h2>

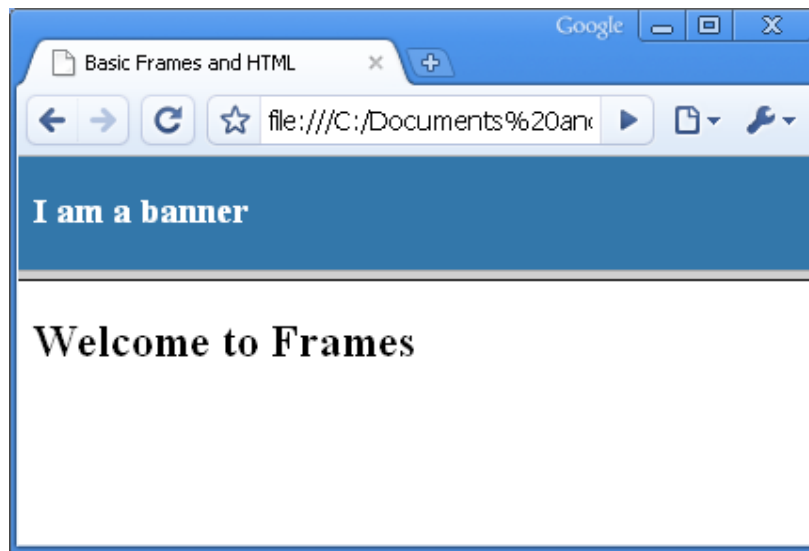
</body>
</html>
```

## Banner.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Banner</title>
</head>

<body bgcolor="#3377AA">
<h3><font color="white"> I am a banner</font></h3>

</body>
</html>
```



### Linking Frames and Windows

We will now create a frameset that will contain a navigation bar and will change the one region of the frameset based on the link that was clicked on the navigationbar.

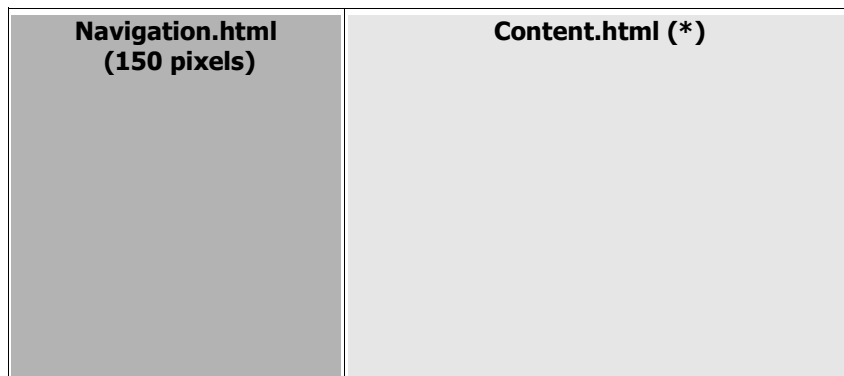
To allow specific regions to be changed when a link has been clicked we will make use of the name attribute for the <frame> HTML tag.

```
<frame src="content.html" name="main" />
```

This name attribute specified will be called once a link has been clicked that made use of the target attribute of the <a> HTML tag.

```
<a href="about.html" target="main">About Us </a>
```

### Basic wireframe layout of the frame\_website.html frameset document.



## frame\_website.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Linking Frames and HTML </title>
</head>

<frameset cols="150,*">
  <frame src="navigation.html" name="navigation" />
  <frame src="content.html" name="main" />
</frameset>

<noframes>
<body>
  Your browser does not support Frames.
</body>
</noframes>
</html>
```

## navigation.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Navigation</title>
</head>

<body>

<h2>Navigation</h2>
<a href="content.html" target="main">Content Page</a><br />
<a href="about.html" target="main">About Us </a>

</body>
</html>
```

## content.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Content</title>
</head>

<body>

<h2>Content Page</h2>

</body>
</html>
```

## about.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>About Us</title>
</head>
<body>
<h2>About Us</h2>
</body>

</html>
```

When we click on the About Us link the frame will be updated with the content of the About.html web page.



## Web page graphics

### Adding a background color to your web page

We can change the background of the web page with the **bgcolor** attribute of the **<body>** opening HTML tag

HTML colors are defined using a hexadecimal (hex) notation for the combination of Red, Green, and Blue color values (RGB). The lowest value that can be given to one of the light sources is 0 (hex 00). The highest value is 255 (hex FF).

Hex values are written as 3 double digit numbers, starting with a # sign.

```
<body bgcolor="#331122" >
```

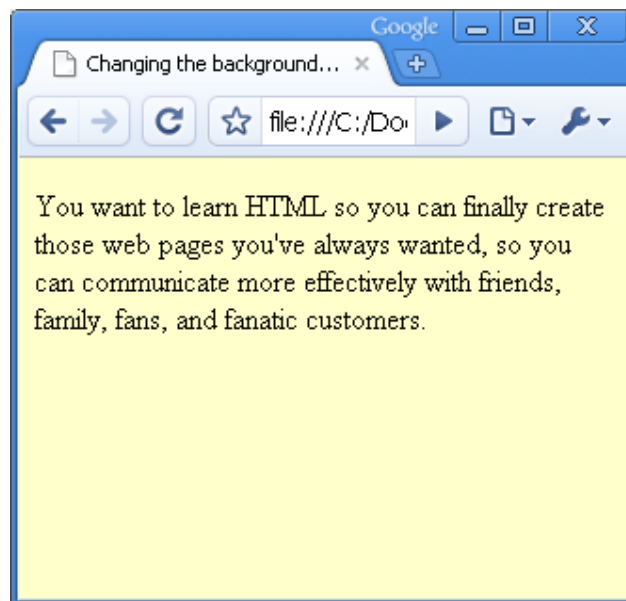
### Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Changing the background colors</title>
</head>

<body bgcolor="#ffffcc">
```

```
<p>You want to learn HTML so you can finally create those web pages you've
always wanted, so you can communicate more effectively with friends,
family, fans, and fanatic customers. </p>
```

```
</body>
</html>
```



## Adding a background image to your HTML web page

To add a background image to the web page we will use the **background** attribute on the opening **<body>** HTML tag. If the image is smaller than the page, the image will repeat itself.

```
<body background="clouds.gif">
```

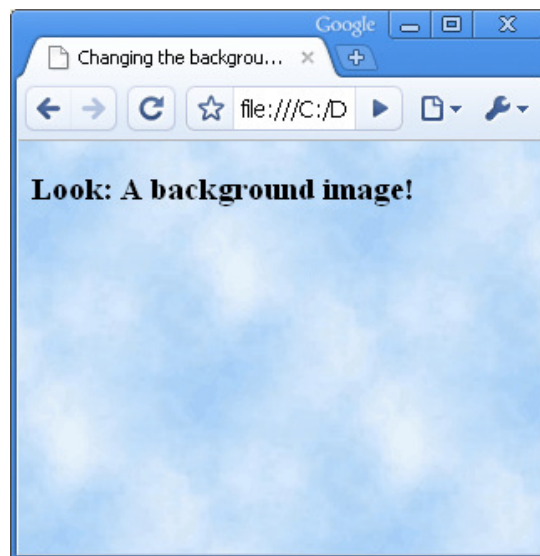
### Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Changing the background colors</title>
</head>

<body background="clouds.gif">

<h3>Look: A background image!</h3>

</body>
</html>
```



## Adding images to your web page

To add images to your web page we will make use of the **<img>** HTML tag. The **src** attribute will be used to specify the location of the image that needs to be display on the web page. The **alt** attribute is used to display alternative text should the image not be displayed. The **alt** attribute is important for creating accessible web pages for users with disabilities, especially those with vision impairments and text based browsers. The **width** and **height** attributes are used to allocate screen space to the image on the web page.

Note that the **<img>** HTML tag does not have a closing tag and needs to have the / in the opening **<img>** tag to conform to the XHTML standard.

```

```

We can wrap text around an image with the use of the **align** attribute. The image will appear along the right hand side of the paragraph if we should assign **right** as the value for the **align** attribute..

```

```

### Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Add images to your HTML page</title>
</head>

<body>




</body>
</html>
```





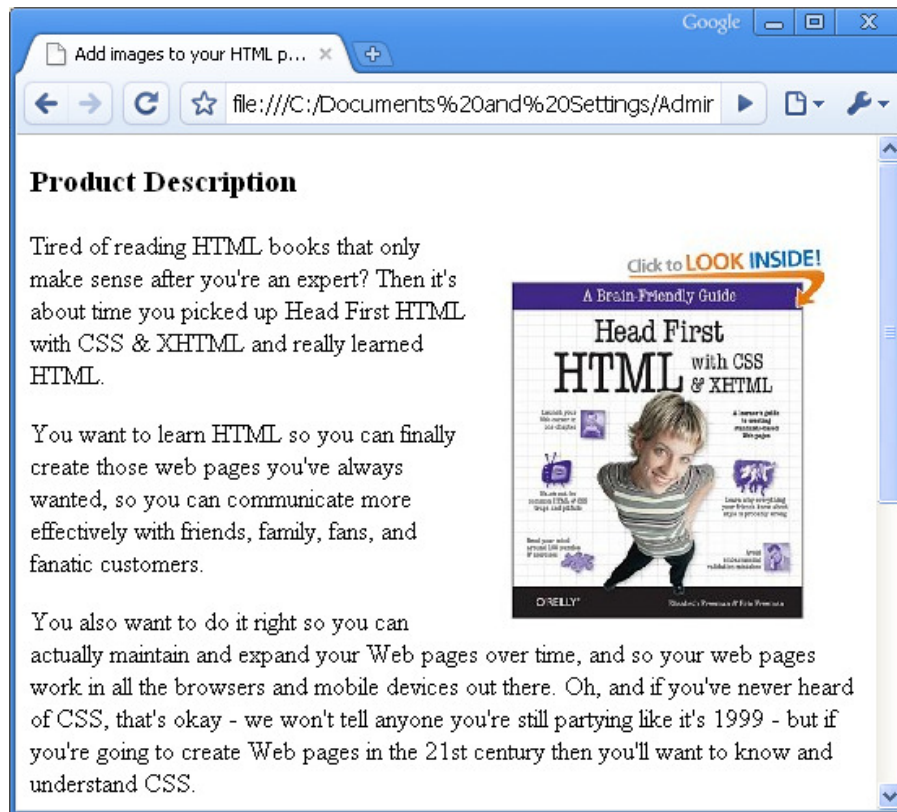
## Example of wrapping text around an image

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Wrapping text around images</title>
</head>

<body>
<h3>Product Description</h3>
<p>
Tired of reading HTML books that only make sense after you're an expert? Then it's
about time you picked up Head First HTML with CSS & XHTML and really learned HTML.
</p>

<p>You want to learn HTML so you can finally create those web pages you've always
wanted, so you can communicate more effectively with friends, family, fans, and
fanatic customers. </p>
<p>You also want to do it right so you can actually maintain and expand your Web
pages over time, and so your web pages work in all the browsers and mobile devices
out there. Oh, and if you've never heard of CSS, that's okay - we won't tell anyone
you're still partying like it's 1999 - but if you're going to create Web pages in
the 21st century then you'll want to know and understand CSS.</p>
<p>
Learn the real secrets of creating Web pages, and why everything your boss told you
about HTML tables is probably wrong (and what to do instead). Most importantly,
hold your own with your co-worker (and impress cocktail party guests) when he
casually mentions how his HTML is now strict, and his CSS is in an external style
sheet.
</p>

</body>
</html>
```



## Making use of Image maps

### What are Image Maps?

An image map provides the ability to define one or many areas on a bitmap (that is any photograph or bit-image picture that can be published on the World Wide Web) and treat each defined area as a separate HTML (or JavaScript) hypertext link. A defined area is called a hotspot and hotspots can be almost any shape and size.

### Information that we need before we start

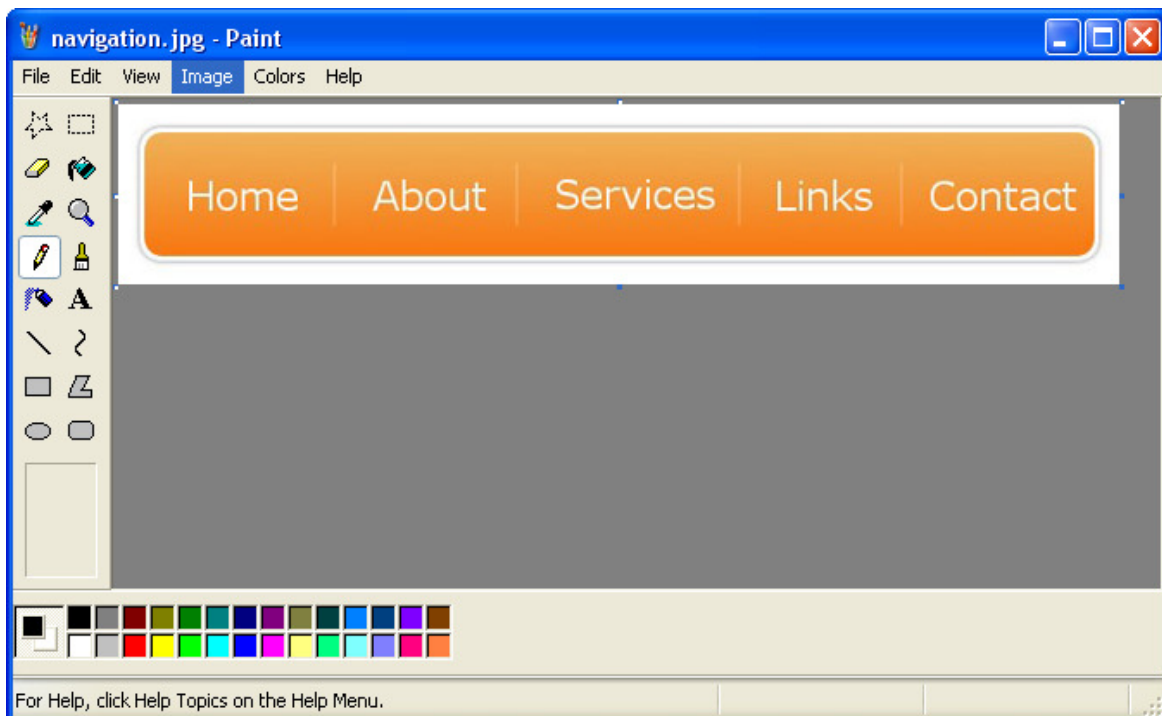
When creating an image map you need to get certain advance information about the image and this includes the size of the image and all the x,y coordinates of the various shapes that you want to define in your image map.

Below is an example of a navigation bar. We will create 5 "hot spots" that will link to other web pages.



We need to determine the size of the image.

This can be done by opening the image in an image editing tool for example Paint.

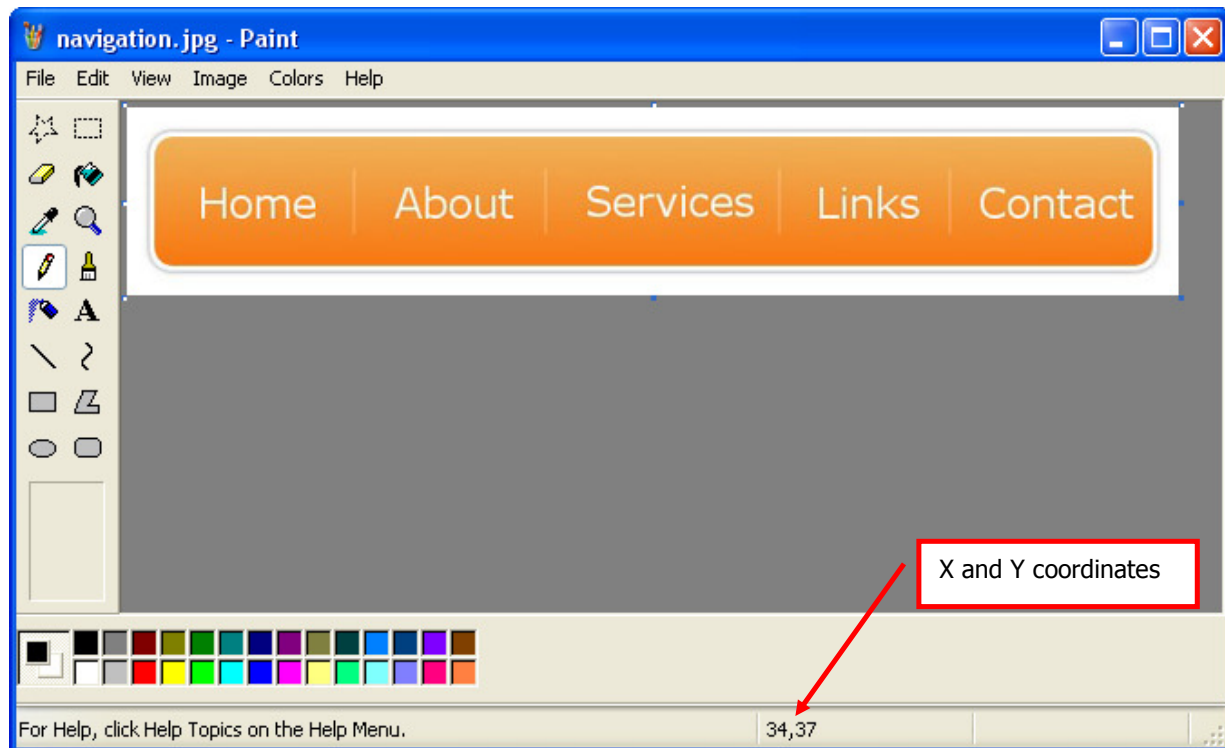


To get the size of the image, simply select **Image->Attributes** from the toolbar



In this example the width is 580 and the height is 103. We will use this information to setup the "hot spots".

Next step will be to determine the coordinates of the "hot spots".



The x and y coordinates are displayed in the bottom right hand side of the window by moving the mouse pointer over the image.

With defining rectangular "hot spots" we only need two coordinates. The top left coordinate and the bottom right coordinate.

## Tags used in Image Maps

### HTML <map> tag

The <map> tag is used to define a client-side image-map. An image-map is an image with clickable areas.

The name attribute is required in the map element. This attribute is associated with the <img>'s usemap attribute and creates a relationship between the image and the map.

Example:

```
<map name="map">
```

### HTML <area> tag

The region that will be used will be defined by the <area> tag.

The area tag has three important attributes:

- shape
- coords
- href

The *shape* attribute is used to define the region it can be one of the following:

- Rectangular (shape="rect")
- Irregular polygon (shape="poly")
- Circle (shape="circle")

The *coords* attribute give the exact pixel coordinates for the region.

- Rectangular – provide the x,y coordinates of the top left corner and bottom right corner
- Irregular polygon – List all the x,y coordinates of all the corners, in sequence.
- Circle – Provide the x,y coordinate of the center point followed by the radius in pixels

The *href* attribute is used to specify the page the region links to.

Example:

```
<area shape="rect" coords="34,36,109,68" alt="Home" href="index.html" />  
<area shape="poly" href="plan.htm" coords="244,3 254,47 251,60 214,68 202,12" />  
<area shape="circle" coords="124,58,8" href="venus.htm" alt="Venus" />
```

### HTML <img> tag

A simple addition to the <img> tag will tie the graphic to the image map. You do this by adding the 'usemap' attribute to the image tag in the following manner.

Example:

```

```

## Example using an Image map

There are three steps that needs to be followed to make use of Image maps:

Step 1: Get the x , y coordinates of top left coordinate and bottom right coordinate and the dimensions of the image

Step 2: Create an image map using the <map> HTML tag

Step 3: Add the image to the HTML web page and link to the image map defined

Step 1: Get the x , y coordinates of top left coordinate and bottom right coordinate and the dimensions of the image



Using the technique that was discussed in the section **Information that we need before we start** we will create rectangular hot spots over the areas to define the links on the image to other pages. The areas have been indicated in the image above.

Area	Top Left Coordinate (X,Y)	Bottom Right Coordinate (X,Y)	Linking to
Home	34,36	109,68	Index.html
About	141,37	222,67	Aboutus.htm
Services	247,39	352,66	Services.html
Links	373,39	442,67	Links.html
Contact	466,39	559,68	Contactus.html

Step 2: Create an image map using the <map> HTML tag

```
<map name="map">
```

```
<area shape="rect" coords="141,37,222,67" alt="About" href="aboutus.html" />
```

```
<area shape="rect" coords="247,39,352,66" alt="Services" href="services" />
```

```
<area shape="rect" coords="373,39,442,67" alt="Links" href="links.html" />
```

```
<area shape="rect" coords="466,39,559,68" alt="Contact" href="contactus.html" />
```

```
</map>
```

Step 3: Add the image to the HTML web page and link to the image map defined

```

```

## Final Version:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Using an image map</title>
</head>

<body>


<map name="map">
<area shape="rect" coords="34,36,109,68" alt="Home" href="index.html" />
<area shape="rect" coords="141,37,222,67" alt="About" href="aboutus.html" />
<area shape="rect" coords="247,39,352,66" alt="Services" href="services" />
<area shape="rect" coords="373,39,442,67" alt="Links" href="links.html" />
<area shape="rect" coords="466,39,559,68" alt="Contact" href="contactus.html" />
</map>

</body>
</html>
```

## Exercise 3:



You have been approached by a friend that would like to advertise his home. He has asked you to create a website that will describe all the different rooms. The content of the different rooms will be provided by him. You have been provided with the floor layout of the house.

Create an image map of the floor layout that will link to the different rooms in the house.

#### **Exercise 4:**

Create a web site that will provide more information about the different continents in the world. The website will consist of 8 pages. The home page will have the world map. Create an image map of the continent map below.



Create 7 web pages that will give more information about the 6 different continents. All the pages must be linked.

The continents are:

- North America
- South America
- Africa
- Europe
- Asia
- Australia

## CASCADING STYLE SHEETS (CSS) – Adding Some Style

### What is CSS?

Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in a markup language. One can create a single style sheet document that specifies the colors, backgrounds, fonts and other aspects to establish a unique look.

It is designed primarily to enable the separation of document content (written in HTML or a similar markup language) from document presentation (written in CSS). This stylesheet will then be linked to every document that needs to apply the presentation.

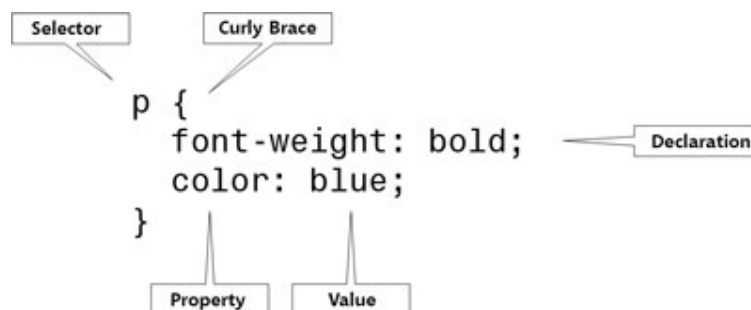
All changes made to one of the CSS documents will reflect on the documents at the same time. A style sheet is a single page of formatting instructions that can control the appearance of many HTML pages at one.

A style sheet is made up of style rules that tell a browser how to present a document

### CSS syntax

The CSS syntax is made up of three parts: a selector, a property and a value:

selector {property: value}



For example:

```
body {color: black}
p {font-family: "sans serif"}
```

If you wish to specify more than one property, you must separate each property with a semicolon.

```
p
{
text-align: center;
color: black;
font-family: arial
}
```



You can group selectors. Separate each selector with a comma.

```
h1,h2,h3,h4,h5,h6
{
color: green
}
```

With the class selector you can define different styles for the same type of HTML element.

```
p.right {text-align: right}
p.center {text-align: center}
```

Use in HTML of class selector

```
<p class="right">
This paragraph will be right-aligned.
</p>
```

```
<p class="center">
This paragraph will be center-aligned.
</p>
```

### **Example of Inline Styles**

Open NotePad and type the following HTML code, save the document as inline.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Inline Styles</title>
</head>

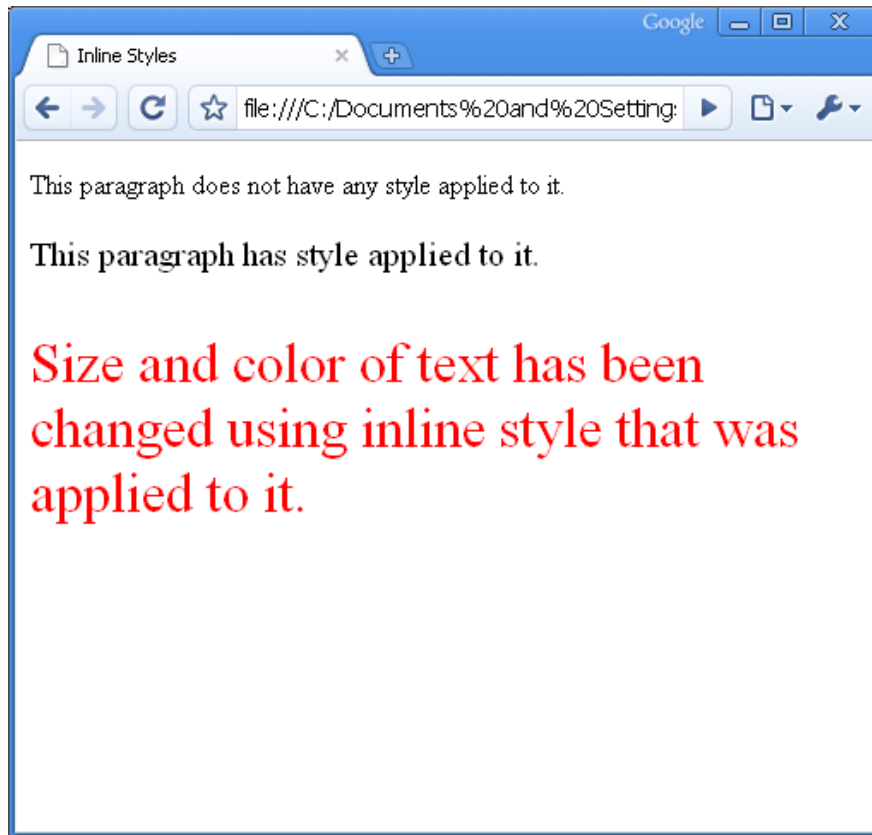
<body>

<p>This paragraph does not have any style applied to it.</p>

<p style="font-size: 15pt;">This paragraph has style applied to it.</p>

<p style="font-size: 25pt; color:red">Size and color of text has been
changed using
inline style that was applied to it.</p>

</body>
</html>
```



### Example of Embedded Style Sheet

Open NotePad and type the following HTML code, save the document as embedded.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Embedded Styles</title>

  <style type="text/css">
    h1 {font-family: arial}
    p {font-size:16pt}
    .special {color: red}
    em {background-color: yellow; color: black}

  </style>
</head>

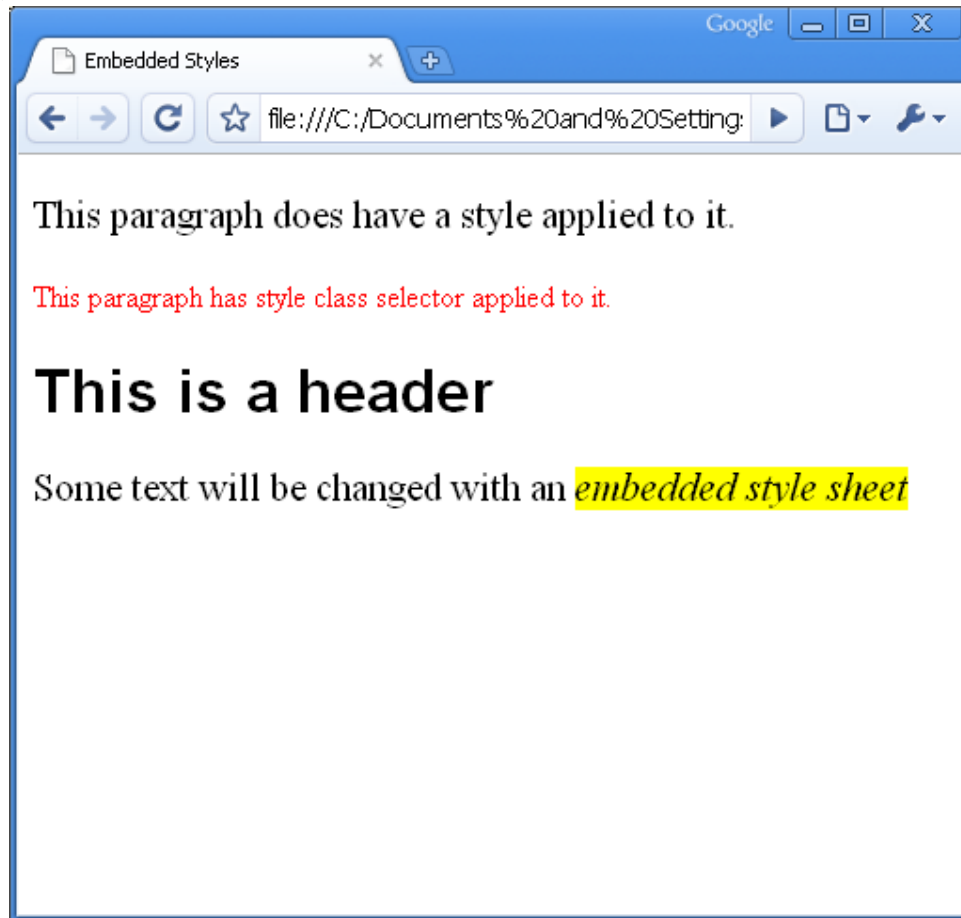
<body>

<p>This paragraph does have a style applied to it.</p>

<div class="special">This paragraph has style class selector applied to
it.</div>

<h1>This is a header</h1>
```

```
<p>Some text will be changed with an <em> embedded style sheet</em></p>  
</body>  
</html>
```



### Linking to external style sheets

**Note how the external document is added to the current HTML document**

```
<link rel="stylesheet" type="text/css" href="stylesheet.css" />
```

Step 1: We would need to create a style sheet document. Open NotePad and type the following CSS into the document. Save as stylesheet.css.

```
h1 {font-family: arial}  
p {font-size:16pt}  
.special {color: red}  
em {background-color: yellow; color: black}
```

Step 2: Create the following HTML document and save as external.html

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>External Styles Sheets</title>

<link rel="stylesheet" type="text/css" href="stylesheet.css" />
</head>

<body>

<h1>External Styles Sheets</h1>

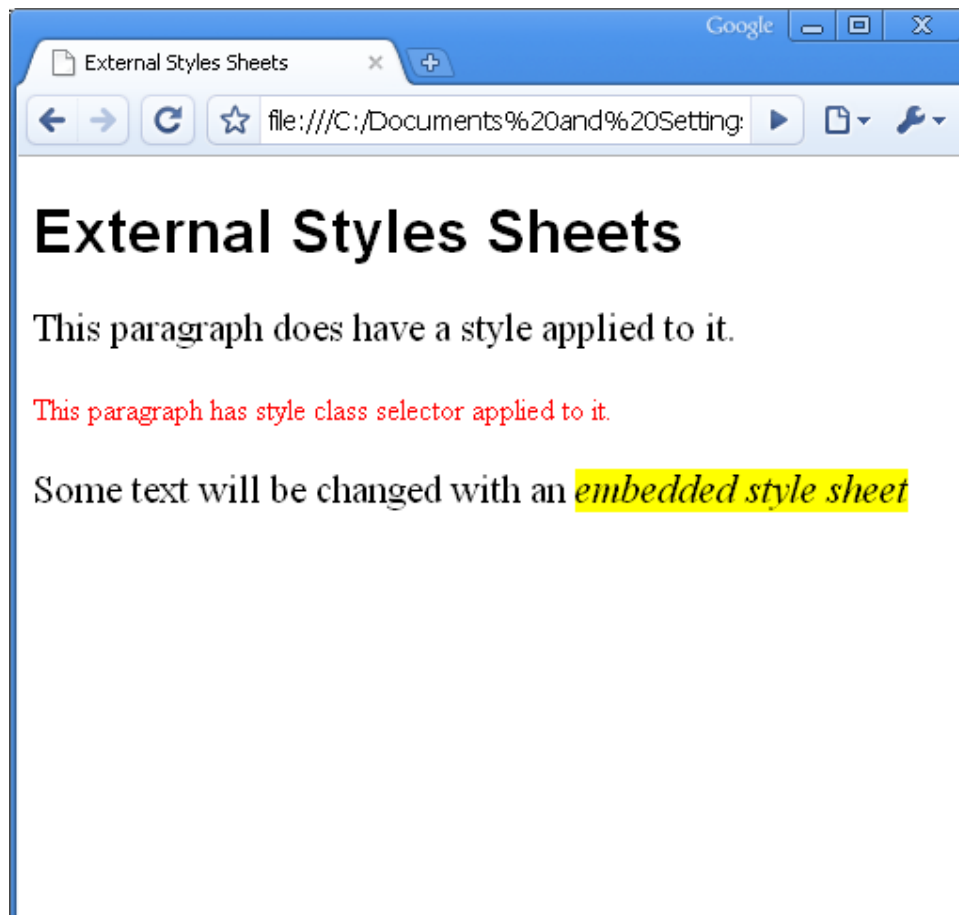
<p>This paragraph does have a style applied to it.</p>

<div class="special">This paragraph has style class selector applied to
it.</div>

<p>Some text will be changed with an <em> embedded style sheet</em></p>

</body>
</html>

```



## Using two style sheets to change the presentation of the HTML

In this example I will show the effect of two different CSS files on the same HTML file. We will need to create two style sheets that will be applied to the HTML file.

### Style1.css

```
body{background-color: #C1FFC1 }
div{text-align:left; border-style:groove; border-color: #3300cc }
p{font-family:sans-serif;font-size:12;font-style:oblique; }
h3{color:#008000 }
a{text-decoration:none }
```

### Style2.css

```
body{background-color: #FFFF80 }
div{text-align:center; border-style:dotted; border-color: #3300cc }
p{font-family:sans-serif;font-size:12;font-style:italic; width:50%; }
h3{color:#9911ff }
```

### Here is the HTML document (home.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>

<head>
  <title></title>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
  <link rel="stylesheet" type="text/css" href="style2.css" title="default" />
/>

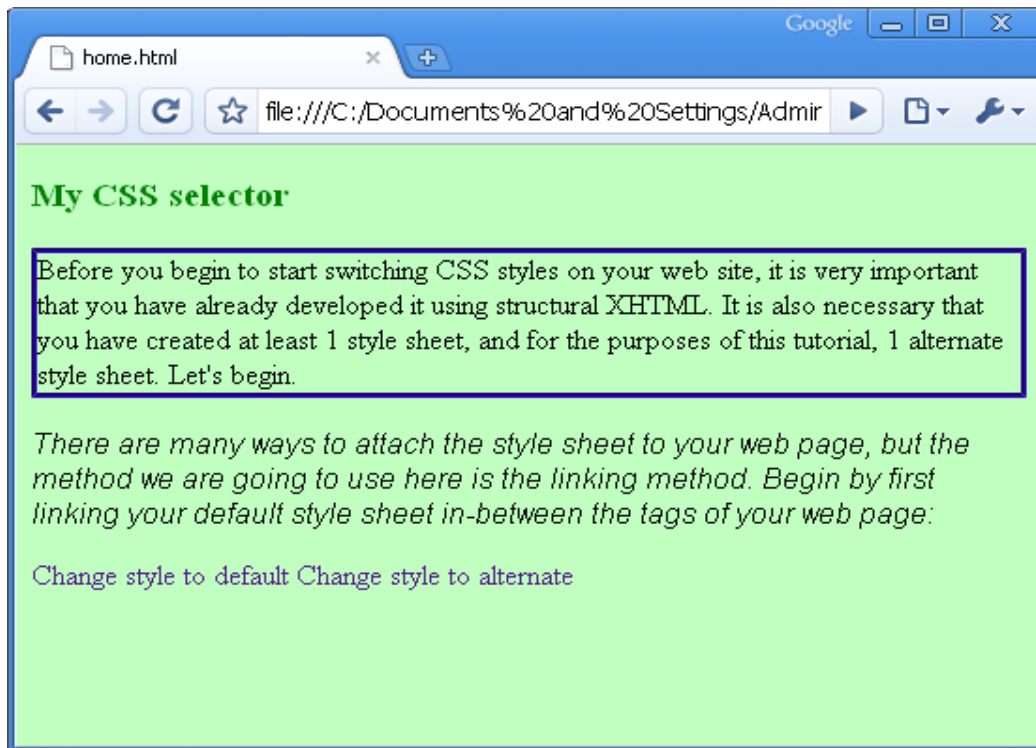
</head>
<body>
<h3>My CSS selector</h3>

<div>
Before you begin to start switching CSS styles on your web site,
it is very important that you have already developed it using structural XHTML.
It is also necessary that you have created at least 1 style sheet,
and for the purposes of this tutorial, 1 alternate style sheet.
Let's begin.
</div>

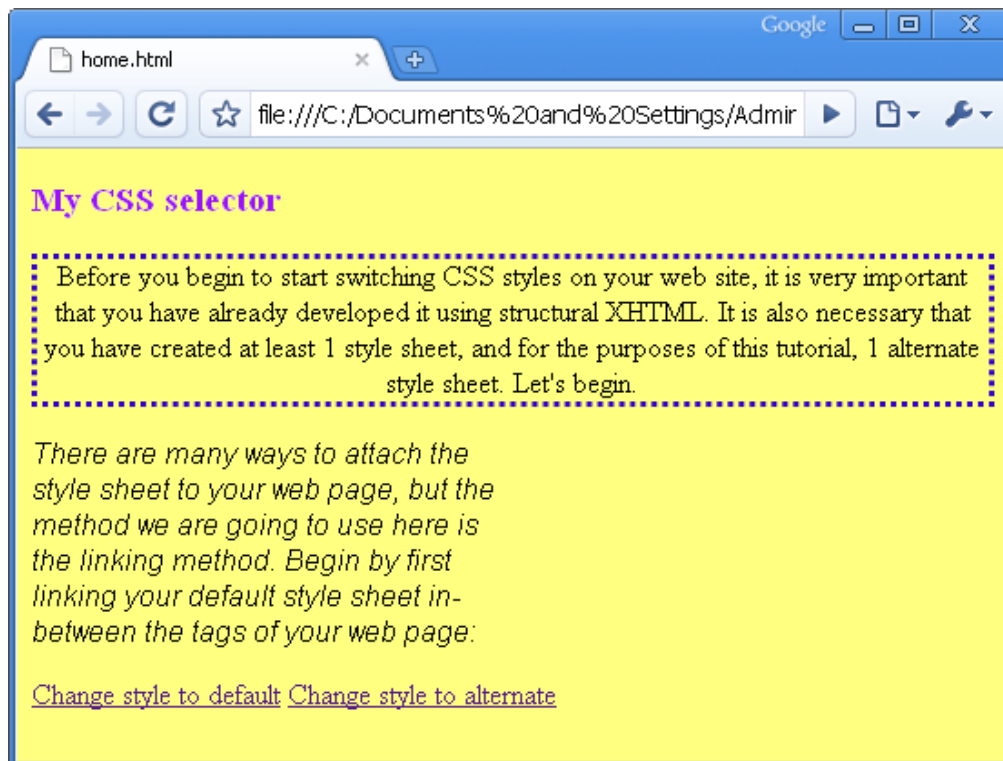
<p>There are many ways to attach the style sheet to your web page,
but the method we are going to use here is the linking method.
Begin by first linking your default style sheet in-between the <head> tags
of your web page:</p>

<a href="home.html" >Change style to default</a>
<a href="home.html" >Change style to alternate</a>
```

Here is the output when Style1.css is applied



Here is the output when Style2.css is applied



## COMPLETE CASE STUDY

### SCUBA STEVE

#### Case Study: Creating a web site for Scuba Steve using XHTML



## Scuba Steve

[Home Page](#)

[Equipment](#)

[Resources](#)

[Top 5 Spots](#)

[Contact Us](#)

#### Welcome to Scuba Steve

Scuba Steve is about experiencing an adventure delight equal to the best in its class & enhances exploring the opulence of this Indian Ocean destination. Scuba Steve caters for all adventure seekers, specializing in diving and dive training. Community based, we support a plethora of activities locally hosted and supplied. Join us as we further realize that sport diving adds to conservation.

#### What is Scuba?

Scuba (originally an acronym for Self Contained Underwater Breathing Apparatus) diving is swimming underwater, or taking part in another activity, while using a scuba set. By carrying a source of breathing gas (usually compressed air), the scuba diver is able to stay underwater longer than with the simple breath-holding techniques used in snorkeling and free-diving, and is not hindered by air lines to a remote air source. The scuba diver typically swims underwater by using fins attached to the feet. However, some divers also move around with the assistance of a DPV (diver propulsion vehicle), commonly called a "scooter", or by using surface-tethered devices called sleds pulled by a boat.



We are approached by Steve who owns a scuba diving company, Scuba Steve. The company specializes in creating scuba diving packages around the world. They target the 5 best diving spots in the world. They also sell scuba diving gear. With the experience of 5000 scuba dives Steve is an expert and likes to share his knowledge on the topic. He wants us to develop a website for him to help create a web presence on the Internet.

We will follow I custom methodology to develop the website for the client.

- Step 1: Define the requirements for the project from the client
- Step 2: Decide on the design in terms of colors, logos, images, theme and HTML standards
- Step 3: Create a wireframe for each web page
- Step 4: Create a site map for the website
- Step 5: Create navigationbar
- Step 6: Add links between pages according to the sitemap
- Step 7: Add content
- Step 8: Validate each page against the standards decided on

Step 9: Add validation icon to each page

Step 10: Sign off of project

### **Step 1: Define the requirements for the project from the client**

We setup a meeting with Steve to obtain the requirements for the web site. We need to ensure that we meet the requirements that Steve wants and hence we need to ensure that we are aligned with Steve in terms of what he wants to achieve with the website. We also need to establish how we will know when the web site has been successful.

The requirements for the website will include the following questions:

What are your specific goals?

How should your Website fit with current promotional and marketing strategies and materials?

What are the schedule or deadline requirements?

What are the budgetary constraints?

How will you measure the success of the site?

Based on the questions the following information was gathered:

We must create a 5 page website that will consists of the following web pages:

Home Page – Page that will be used to tell the web user what the web site is about

Equipment – Describe all the equipment used in scuba, must include images

Resources- Links to web sites with similar information about scuba diving

Contact Us – Contact details page, must include contact person, email address and contact numbers

5 best diving spots – Describe the 5 best places to dive in the world, must include images

The web site must be completed in 4 weeks. The first week the Home Page and Contact Us page must be completed and deployed on the web hosting company. Then the other pages must be completed each week and after accepted by the client deployed to the web hosting company.

Payment will be done by client, 50% upfront and the remaining upon delivery of the last portion of the website

The domain name and web hosting company has already been established. The client has a web master and he will upload the pages and maintain the web site after sign off.

The processes has been defined as follows. Once a web page has been accepted by the client, we need to send the complete web page and images to the web master.

We also established after the meeting the following:

The client will provide all the content.

We are responsible to search for images for use on the website

The main colors that must be used are Grey, white and blue.

The client wants two column website: One navigation and the other content area

The main objective of the client is to have a web presence with the website with the required pages.



## Step 2: Decide on the design in terms of colors, logos, images, theme and HTML standards

The colors have been defined by the client. We made use of a image manipulation program called GIMP to ensure that the correct colors are used on the web site.

Blue: 256ae5  
White: ffffff  
Grey: a0a1ae

Images will be bought from websites that sells images based on themes. We will make use of the following sites:

<http://www.fotosearch.com/photos-images/websites.html>  
<http://www.freedigitalphotos.net/>  
<http://www.istockphoto.com/>

We have decided to develop the website using the XHTML 1.0 Transitional

Hence each web page should conform the standard. Each each will have the following page DOC declaration

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

## Step 3: Create a wireframe for each web page

A **wireframe** is a grayscale block diagram that illustrates the overall navigation and the blocks of elements such as content, functionality, etc. that will go on the screen. It does not contain pictures and doesn't necessarily need to link to anything. It just demonstrates what elements a web page or application screen will contain and roughly where they might go—although the location can change. It does not include visual design. We will create the wireframe using HTML. Adding the the content description to the position where the content will be added. This is the visual layout of the wireframe.

Logo	Scuba Steve
Navigation Bar	Content
Validation Logo	

Here is a copy of the HTML code that will be used for each page.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<title>Scuba Steve Home Page</title>
<meta http-equiv="Content-type" content="text/html; charset=UTF-8" />
</head>

<body>

<table border="1" width="92%">
<tr><td>Logo</td><td>Scuba Steve</td></tr>
```

```

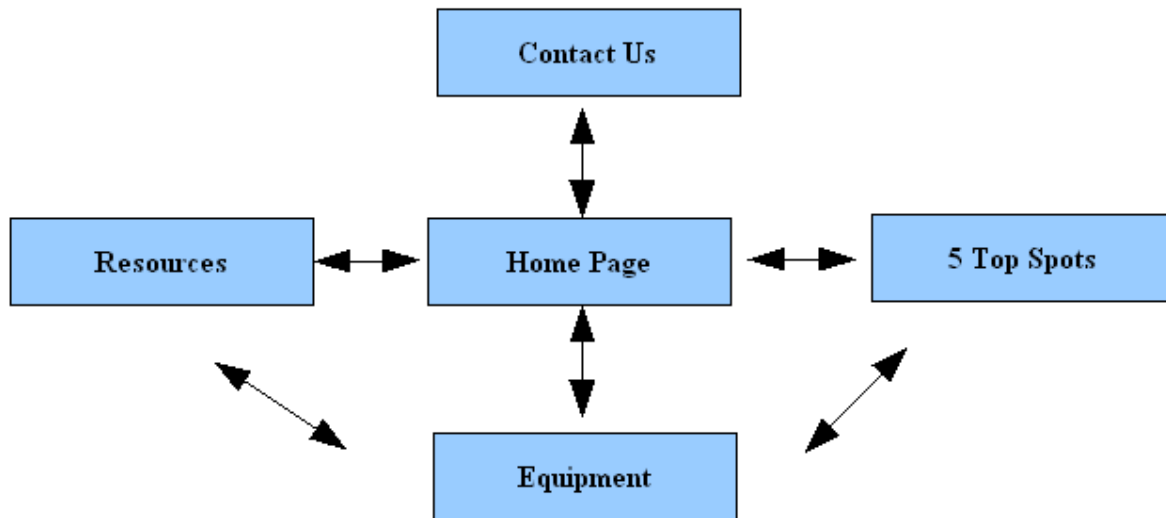
<tr><td>Navigation Bar</td><td>Content</td></tr>
<tr><td colspan="2">Validation Logo</td></tr>
</table>

</body>
</html>

```

#### Step 4: Create a site map for the website

A site map (or sitemap) is a representation of the architecture of a web site. We will create a visual site map that will show how the pages are linked together.



The information will be used to create the navigation bar for each page.

#### Step 5: Create navigationbar

A navigation bar should be created for each page. We will make use of the site map to generate each navigation bar.

The Home Page needs to link to:

Resources  
Equipment  
5 Top Spots  
Equipment  
Contact Us

Here is the HTML code, we will create a text based navigation bar using HTML tables

```

<table>
<tr><td><a href="home.html">Home Page</a></td></tr>
<tr><td><a href="equipment.html">Equipment</a></td></tr>
<tr><td><a href="resources.html">Resources</a></td></tr>
<tr><td><a href="top.html">Top 5 Spots</a></td></tr>

```

```
<tr><td><a href="contact.html">Contact Us</a></td></tr>
</table>
```

Logo	Scuba Steve
Home Page Equipment Resources Top 5 Spots Contact Us	Content
Validation Logo	

### Step 6: Add links between pages according to the sitemap

Here is the HTML code to link the pages.

```
<table>
<tr><td><a href="home.html">Home Page</a></td></tr>
<tr><td><a href="equipment.html">Equipment</a></td></tr>
<tr><td><a href="resources.html">Resources</a></td></tr>
<tr><td><a href="top.html">Top 5 Spots</a></td></tr>
<tr><td><a href="contact.html">Contact Us</a></td></tr>
</table>
```

The Resources Page needs to link to:

Home Page  
Equipment  
5 Top Spots

```
<table>
<tr><td><a href="home.html">Home Page</a></td></tr>
<tr><td><a href="equipment.html">Equipment</a></td></tr>
<tr><td><a href="top.html">Top 5 Spots</a></td></tr>
</table>
```

The Equipment Page links to:

Resources  
5 Top Spots  
Home Page

```
<table>
<tr><td><a href="home.html">Home Page</a></td></tr>
<tr><td><a href="resources.html">Resources</a></td></tr>
<tr><td><a href="top.html">Top 5 Spots</a></td></tr>
</table>
```

The Contact Us Page only links to the Home Page

```
<table>
<tr><td><a href="home.html">Home Page</a></td></tr>
</table>
```

The 5 Top Spots Page links to the following:

Equipment  
Resources  
Home Page

```
<table>
<tr><td><a href="home.html">Home Page</a></td></tr>
<tr><td><a href="equipment.html">Equipment</a></td></tr>
<tr><td><a href="resources.html">Resources</a></td></tr>
</table>
```

This is how our web page looks like after adding the navigation bar to the wireframe template for the Home Page.

Logo	Scuba Steve
<a href="#">Home Page</a> <a href="#">Equipment</a> <a href="#">Resources</a> <a href="#">Top 5 Spots</a> <a href="#">Contact Us</a>	Content
Validation Logo	

**Step 7: Add content**

We will add the content to the areas that were defined with our wireframe.



The logo was send by the client  
The Scuba Steve banner



A document containing the content for the Home Page was send by the client. Here is an extract:

*"Scuba Steve is about experiencing an adventure delight equal to the best in its class & enhances exploring the opulence of this Indian Ocean destination. Scuba Steve caters for all adventure seekers, specializing in diving and dive training. Community based, we support a plethora of activities locally hosted and supplied. Join us as we further realize that sport diving adds to conservation"*

Images that will be used on the home page:



Please note that the images used are from the public domain. Some of the images does include there watermark as to reference and credit the website where the image can be found and bought without the watermark.

- [www.gettyimages.com](http://www.gettyimages.com) (80586879)
- [www.photosearch.com](http://www.photosearch.com)

The HTML code including the changes to the wireframe

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Scuba Steve Home Page</title>
<meta http-equiv="Content-type" content="text/html; charset=UTF-8" />

</head>

<body>
<table border="0" width="92%">
<tr><td></td><td></td></tr>
<tr><td colspan="2" valign="top">
<table>
<tr><td><a href="home.html">Home Page</a></td></tr>
<tr><td><a href="equipment.html">Equipment</a></td></tr>
<tr><td><a href="resources.html">Resources</a></td></tr>
<tr><td><a href="top.html">Top 5 Spots</a></td></tr>
<tr><td><a href="contact.html">Contact Us</a></td></tr>
</table>
</td><td><h3>Welcome to Scuba Steve</h3>Scuba Steve is about experiencing an
adventure delight equal to the best in its
class & enhances exploring the opulence of this Indian Ocean destination. 

```

Scuba Steve caters for all adventure seekers, specializing in diving and dive training.  
 Community based, we support a plethora of activities locally hosted and supplied.  
 Join us as we further realize that sport diving adds to conservation.

### What is Scuba?

Scuba (originally an acronym for Self Contained Underwater Breathing Apparatus) diving is swimming underwater, or taking part in another activity, while using a scuba set. By carrying a source of breathing gas (usually compressed air), the scuba diver is able to stay underwater longer than with the simple breath-holding techniques used in snorkeling and free-diving, and is not hindered by air lines to a remote air source. The scuba diver typically swims underwater by using fins attached to the feet. However, some divers also move around with the assistance of a DPV (diver propulsion vehicle), commonly called a "scooter", or by using surface-tethered devices called sleds pulled by a boat.

```

<p>






</p>

</td></tr>
<tr ><td colspan="2">Validation Logo</td></tr>
</table>

</body>
</html>

```

The Complete Home Page (Note that the wireframe is still visible. We need to remove it by making the value of the border="0" and the navigation bar needs to be aligned to the top of the table)

	<h1>Scuba Steve</h1>
<a href="#">Home Page</a> <a href="#">Equipment</a> <a href="#">Resources</a> <a href="#">Top 5 Spots</a> <a href="#">Contact Us</a>	<p><b>Welcome to Scuba Steve</b></p> <p>Scuba Steve is about experiencing an adventure delight equal to the best in its class &amp; enhances exploring the opulence of this Indian Ocean destination. Scuba Steve caters for all adventure seekers, specializing in diving and dive training. Community based, we support a plethora of activities locally hosted and supplied. Join us as we further realize that sport diving adds to conservation.</p> <p><b>What is Scuba?</b></p> <p>Scuba (originally an acronym for Self Contained Underwater Breathing Apparatus) diving is swimming underwater, or taking part in another activity, while using a scuba set. By carrying a source of breathing gas (usually compressed air), the scuba diver is able to stay underwater longer than with the simple breath-holding techniques used in snorkeling and free-diving, and is not hindered by air lines to a remote air source. The scuba diver typically swims underwater by using fins attached to the feet. However, some divers also move around with the assistance of a DPV (diver propulsion vehicle), commonly called a "scooter", or by using surface-tethered devices called sleds pulled by a boat.</p> <div>    </div>
Validation Logo	

## Step 8: Validate each page against the standard

To validate each page we will make use of the following website, <http://validator.w3.org/>

The screenshot shows the 'Validate by File Upload' tab selected. It includes a file input field with a 'Browse...' button, a 'More Options' section with various checkboxes and dropdowns, and a 'Check' button at the bottom.

**Validate by URI**   **Validate by File Upload**   **Validate by Direct Input**

### Validate by File Upload

Upload a document for validation:

File:  

▼ **More Options**

**Character Encoding**  ☐ Only if missing

**Document Type**  ☐ Only if missing

☒ List Messages Sequentially   ☐ Group Error Messages by Type

☐ Show Source   ☐ Clean up Markup with HTML Tidy

☐ Show Outline   ☐ Validate error pages   ☐ Verbose Output

Step 1: Click on "Validate by File Upload"

Step 2: Click on "More Options", ensure that the "Document Type" is selected according the standard selected.

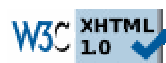
Step 3: Click on the "Browse" button and select the file

Step 4: Click on "Check"

## Step 9: Add validation icon to each page

Once a page had been validated we can then add the following the area defined for the validation icon in the wireframe.

```
<p>
  <a href="http://validator.w3.org/check?uri=referer"></a>
</p>
```



## Final Version of the Scuba Steve Home Page after validation

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Scuba Steve Home Page</title>
<meta http-equiv="Content-type" content="text/html; charset=UTF-8" />

</head>

<body>
<table border="0" width="92%">
<tr><td></td><td></td></tr>
<tr><td valign="top">
<table>
<tr><td><a href="home.html">Home Page</a></td></tr>
<tr><td><a href="equipment.html">Equipment</a></td></tr>
<tr><td><a href="resources.html">Resources</a></td></tr>
<tr><td><a href="top.html">Top 5 Spots</a></td></tr>
<tr><td><a href="contact.html">Contact Us</a></td></tr>
</table>
</td><td><h3>Welcome to Scuba Steve</h3>Scuba Steve is about experiencing an
adventure delight equal to the best in its
class &amp; enhances exploring the opulence of this Indian Ocean destination. 
Scuba Steve caters for all adventure seekers, specializing in diving and dive
training.
Community based, we support a plethora of activities locally hosted and supplied.
Join us as we further realize that sport diving adds to conservation. <br /><br
/><h3>What is Scuba?</h3>Scuba (originally an acronym for Self
Contained Underwater Breathing Apparatus)
diving is swimming underwater,
or taking part in another activity, while using a scuba set. By carrying a source
of breathing gas (usually compressed air), the scuba diver
is able to stay underwater longer than with the simple breath-holding techniques
used in snorkeling and free-diving, and is not hindered by air lines
to a remote air source. The scuba diver typically swims underwater by using fins
attached to the feet.
However, some divers also move around with the assistance of a DPV (diver
propulsion vehicle), commonly called a "scooter", or by using surface-tethered
devices called sleds pulled by a boat.

<p>


</p>

</td></tr>
<tr><td colspan="2" align="center"> <p>
<a href="http://validator.w3.org/check?uri=referer"></a>
</p>
</td></tr>
</table>

</body>
</html>
```



This process will be repeated for all the other pages

### Step 10: Sign off of project

After completion of all the pages the client will sign off on the project. During the requirements process (Step 1) certain objectives were defined that would be used to measure the success of the project

We will create a quick checklist that the client will sign off each objective that was defined

Objectives	Completed (Yes/No)	Date	Sign to Accept
Home Page			
Resources			
Contact Us			
Equipment			
5 Top Spots			
Defined Colors used			
Two column website			
HTML Standards used			

## HTML - Links to Resources / List of references

### Resources:

<http://validator.w3.org/>  
<http://jigsaw.w3.org/css-validator/>  
<http://www.validome.org/>  
[http://en.wikipedia.org/wiki/W3C\\_Markup\\_Validation\\_Service](http://en.wikipedia.org/wiki/W3C_Markup_Validation_Service)  
<http://www.w3.org/QA/Tools/>  
<http://htmlhelp.com/tools/validator/>  
<http://www.w3.org/International/O-charset>  
<http://www.htmlite.com/XH002.php>  
<http://anthologyoi.com/dev/xhtml-vs-html-round-2.html>  
[http://htmlfixit.com/tutes/tutorial\\_XHTML\\_and\\_HTML\\_-\\_The\\_differences.shtml](http://htmlfixit.com/tutes/tutorial_XHTML_and_HTML_-_The_differences.shtml)  
[http://www.w3schools.com/XHTML/xhtml\\_html.asp](http://www.w3schools.com/XHTML/xhtml_html.asp)  
<http://www.xhtml.org/>  
<http://en.wikipedia.org/wiki/XHTML>  
<http://www.sitepoint.com/print/html-or-xhtml-does-it-matter/>  
[http://www.w3schools.com/HTML/html\\_formatting.asp](http://www.w3schools.com/HTML/html_formatting.asp)  
<http://www.yourhtmlsource.com/myfirstsite/basicformatting.html>  
<http://www.echoecho.com/htmltext04.htm>  
<http://www.web-wise-wizard.com/html-tutorials/html-text-format-formatting-tags.html>  
<http://www.utexas.edu/learn/html/spchar.html>  
<http://www.chami.com/tips/internet/050798I.html>  
[http://www.w3schools.com/HTML/html\\_links.asp](http://www.w3schools.com/HTML/html_links.asp)  
<http://www.htmlgoodies.com/primers/html/article.php/3478171>  
<http://www.pageresource.com/html/linking.htm>  
<http://www.tizag.com/htmlT/htmlheadings.php>  
[http://www.w3schools.com/html/html\\_primary.asp](http://www.w3schools.com/html/html_primary.asp)  
<http://www.htmldog.com/guides/htmlbeginner/headings/>  
[http://www.w3schools.com/HTML/html\\_lists.asp](http://www.w3schools.com/HTML/html_lists.asp)  
<http://www.htmlcodetutorial.com/lists.html>  
<http://www.w3.org/TR/html401/struct/lists.html>  
<http://konstruktors.com/blog/design-suggetions/125-how-to-create-beautiful-and-elegant-html-lists-using-css/>  
<http://jobsearch.about.com/od/cvsamples/a/cvexample1.htm>  
[http://www.cvtips.com/CV\\_example.html](http://www.cvtips.com/CV_example.html)  
<http://www.cv-resume.org/>  
[http://www.cv-resume.org/pages/resume\\_samples.php](http://www.cv-resume.org/pages/resume_samples.php)  
[http://www.w3schools.com/html/html\\_tables.asp](http://www.w3schools.com/html/html_tables.asp)

[http://www.htmlcodetutorial.com/tables/index\\_famsupp\\_27.html](http://www.htmlcodetutorial.com/tables/index_famsupp_27.html)  
<http://www.tizag.com/htmlT/tables.php>  
<http://www.pageresource.com/html/table1.htm>  
<http://www.echoecho.com/htmltables.htm>  
[http://www.w3schools.com/html/html\\_forms.asp](http://www.w3schools.com/html/html_forms.asp)  
<http://www.echoecho.com/htmlforms.htm>  
<http://www.tizag.com/htmlT/forms.php>  
<http://www.cs.tut.fi/~jkorpela/forms/>  
<http://www.sitepoint.com/print/fancy-form-design-css/>  
<http://www.htmldog.com/guides/htmlbeginner/forms/>  
[http://www.iso.org/iso/country\\_codes/iso\\_3166\\_code\\_lists.htm](http://www.iso.org/iso/country_codes/iso_3166_code_lists.htm)  
[http://www.w3schools.com/html/html\\_frames.asp](http://www.w3schools.com/html/html_frames.asp)  
<http://www.echoecho.com/htmlframes.htm>  
<http://www.tizag.com/htmlT/frames.php>  
<http://www.html-faq.com/htmlframes/?framesareevil>  
<http://www.quackit.com/html/examples/frames/>  
<http://www.echoecho.com/htmlframes08.htm>  
[http://en.wikipedia.org/wiki/Image\\_map](http://en.wikipedia.org/wiki/Image_map)  
<http://www.w3.org/TR/REC-html40/struct/objects>  
<http://www.elated.com/articles/creating-image-maps/>  
<http://www.javascriptkit.com/howto/imagemap.shtml>  
[http://www.htmlcodetutorial.com/images/images\\_famsupp\\_220.html](http://www.htmlcodetutorial.com/images/images_famsupp_220.html)  
[http://www.accessiblelearning.net/mod9/9\\_16.php](http://www.accessiblelearning.net/mod9/9_16.php)  
[http://www.w3schools.com/TAGS/tag\\_map.asp](http://www.w3schools.com/TAGS/tag_map.asp)  
<http://www.web-wise-wizard.com/web-graphics-design/html-client-side-image-maps.html>  
<http://www.chami.com/tips/internet/050698I.html>  
[http://www.w3schools.com/HTML/html\\_images.asp](http://www.w3schools.com/HTML/html_images.asp)  
<http://www.htmlgoodies.com/primers/html/article.php/3478181>  
<http://www.htmlgoodies.com/primers/html/article.php/3478191>  
<http://www.tizag.com/htmlT/images.php>  
<http://htmldog.com/guides/htmlbeginner/images/>  
[http://www.w3schools.com/HTML/html\\_colors.asp](http://www.w3schools.com/HTML/html_colors.asp)  
<http://www.echoecho.com/htmlbackgrounds02.htm>  
[http://www.quackit.com/html/codes/html\\_background\\_image\\_codes.cfm](http://www.quackit.com/html/codes/html_background_image_codes.cfm)  
<http://www.sitepoint.com/print/css-anthology-tips-tricks-1/>  
<http://www.sitepoint.com/subcat/css>  
<http://www.w3schools.com/css/>  
[http://en.wikipedia.org/wiki/Cascading\\_Style\\_Sheets](http://en.wikipedia.org/wiki/Cascading_Style_Sheets)  
<http://htmlhelp.com/reference/css/>  
<http://jigsaw.w3.org/css-validator/>  
<http://www.bboxesandarrows.com/view/prototyping-with>  
<http://www.sitepoint.com/print/tools-prototyping-wireframing/>  
<http://99designs.com/>  
<http://www.sitepoint.com/print/site-planner/>  
<http://www.sitepoint.com/print/importance-web-standards/>  
<http://www.sitepoint.com/print/requirements-gathering/>  
<http://www.gimp.org/>  
<http://www.w3.org/TR/xhtml-basic/>  
<http://www.hongkiat.com/blog/50-websites-for-free-vector-images-download/>  
<http://validator.w3.org/>  
<http://searchenginewatch.com/2167931>  
[http://www.gettyimages.com \(80586879\)](http://www.gettyimages.com (80586879))  
<http://www.photosearch.com>



Dalize van Heerden  
© 2009

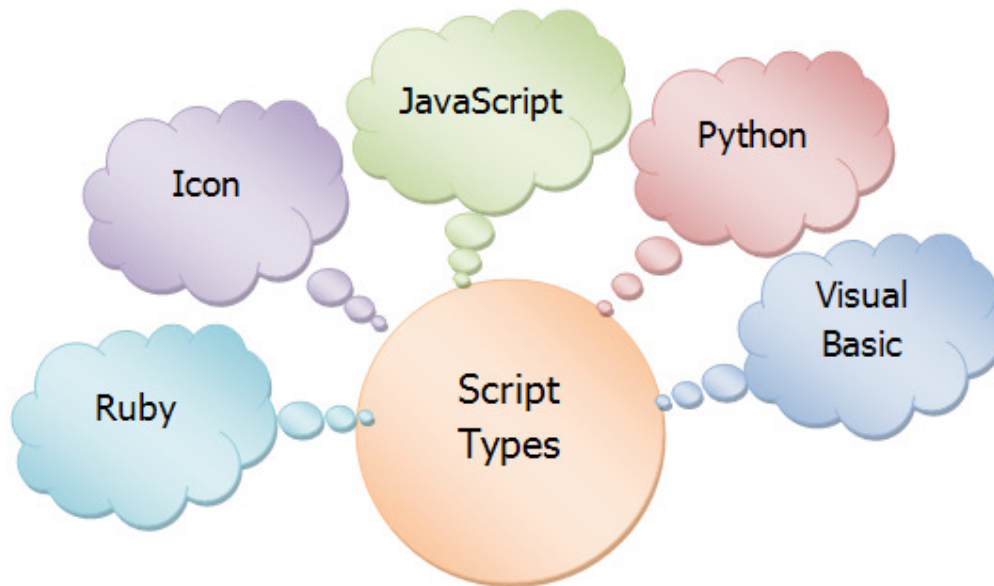
# Introduction to JavaScript

## Introduction to JavaScript

Notes compiled by: [D van Heerden](#)

### Introduction

Scripting language has been around since the 1960's. They were originally used as command languages for computer operator tasks. Scripting languages, as we know them today, only started developing in the late 1980's.



Scripting languages are not used to develop large scale applications, system programming languages such as C, Java and C++ are used for this. The system programming languages are used to program complex algorithms and to manipulate large amounts of data.

Scripting languages are used to join pre-existing components, develop graphical user interfaces and doing basic text manipulation. As with all technology scripting languages are still developing and as time pass more components are added and more complex features becomes available.

### What is JavaScript?

JavaScript is a scripting language developed by Netscape. It is used to add interactive functions to HTML documents. JavaScript is widely supported by web browsers and other web tools.

JavaScript is a client-side scripting language, indicating that it runs on the local browser instead of on a web server. Each browser, like Internet Explorer and Firefox have a scripting engine that executes scripting code. The interpreter is the program that executes or renders the scripting code.

It is an object-based programming language, making use of pre-defined objects to manipulate text to be rendered in a user-friendly way.

JavaScript exists within HTML code and cannot be rendered on its own.

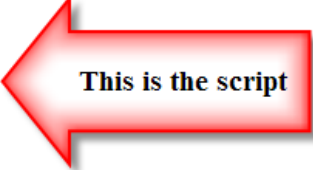
**Example:**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>My First Script </title>
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1"/>
  <link rel="stylesheet" href=" " type="text/css" />
  <script type="text/javascript">
    <!-- HIDE FROM INCOMPATIBLE BROWSERS
    document.write("<h1>This is the first script I've written</h1><hr />");
    // STOP HIDING FROM INCOMPATIBLE BROWSERS -->
  </script>
</head>

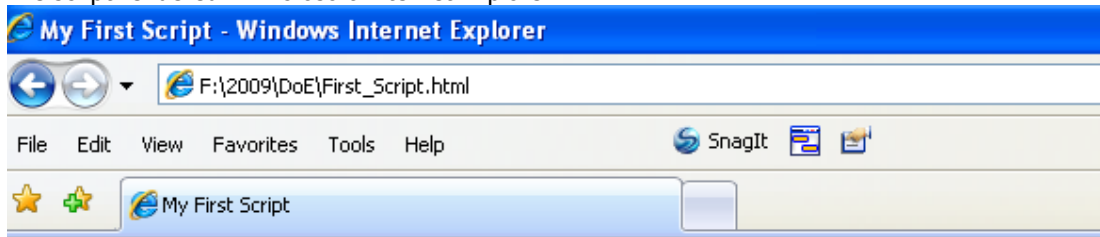
<body>

</body>
</html>
```



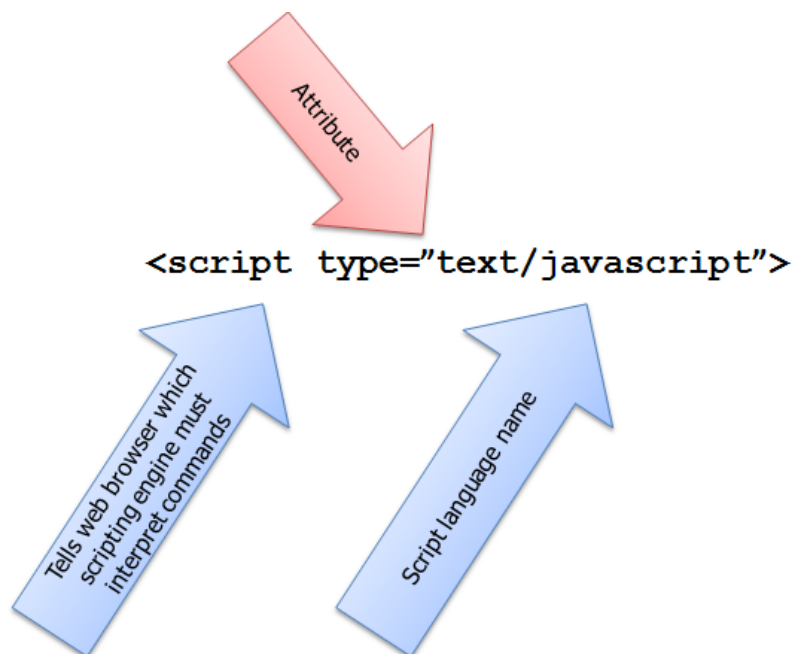
**This is the script**

The script rendered in Microsoft Internet Explorer:



**This is the first script I've written**

---



```
statement= document.write("<p>Welcome to South Africa!</p>");
```



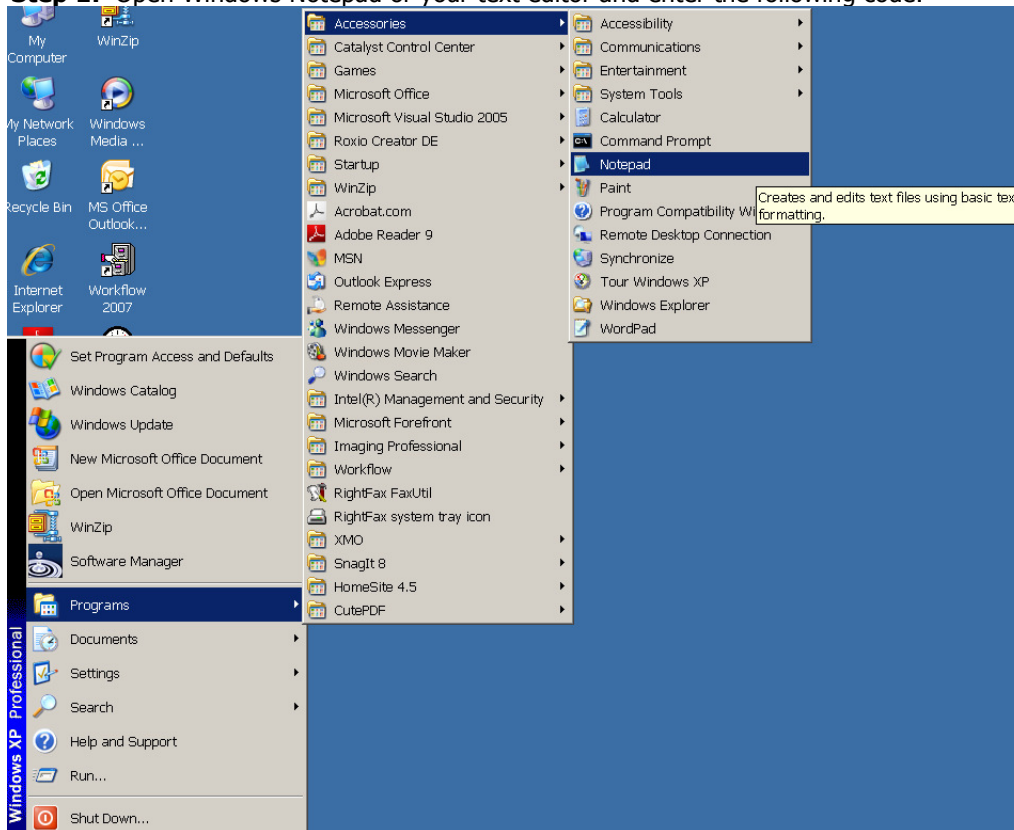
JavaScript is both an interpreted scripting language and a low level object-based programming language. An object is programming code and data that can be treated as an individual unit or component. Methods are procedures associated with an object and properties are piece of data associated with an object. Procedures are individual statements used in a computer program grouped into logical units, used to perform specific tasks

#### Example:

Object	The body	Loan
Procedures	wriggle / twist / bend / rotate	pay / tax / interest
Method	Arm	Calculate
Property	Fingers / hand / elbow / shoulder	amount / rate / instalment

#### Step-by-Step create your first JavaScript Program

**Step 1:** Open Windows Notepad or your text editor and enter the following code.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

**Comment 1:** The `<!DOCTYPE>` declaration determines the Document Type Definition (DTD) to which the document complies. It defines the attributes and elements used by the document as well as the rules the document must follow to ensure it is well formed.

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

**Comment 2:** The XMLNS attribute is required in the `<html>` element to ensure the document is well formed.

```
<head>
```

**Comment 3:** The `<head>` element contains information that is used by the browser.

```
<title>Welcome SA</title>
```

**Comment 4:** The `<title>` element contains text that will appear in the browser title bar.

```
<link rel="stylesheet" href="styles.css" type="text/css" />
```

**Comment 5:** This line links the document to the external style sheet. The three attributes in the link element is the *rel* attribute that is assigned a value of style sheet, the *href* attribute that is assigned the URL of the style sheet and the *type* attribute which is assigned the same text/css value as the type attribute used in the `<style>` element.

```
<meta http-equiv="content-type" content="text/html;
charset=iso-8859-1" />
```

**Comment 6:** The `<meta>` element specifies the content type that the document uses which is sent to the Web browser to provide information the browser needs to render the page

```
</head>
```

**Comment 7:** Very important to remember that every opening element must have a closing element to ensure the document is well formed

```
<body>
```

**Comment 8:** The `<body>` element refers to the document body, this is where the document is assembled and formatted according to the specified instruction.

```
<h1>Welcome to South Africa!</h1><hr />
```

**Comment 9:** The `<h?>` element changes the appearance of the font and the `<hr />` element adds a horizontal rule. Note that the `<hr />` element is an opening and closing element in one.

```
<h2>Land of splendour and beauty</h2>
<script type="text/javascript">
```

**Comment 10:** The `<script>` element tells the Web browser that the scripting engine must interpret the commands it contains. The *type* attribute tells the browser which scripting language is used and what the version of the language is.

```
/* <![CDATA[ */
```

**Comment 11:** CDATA prevents the validator from attempting to parse the JavaScript statements.

```
document.write("<p><h3>Sawubona!</h3></p>");
```

**Comment 12:** The *Document* object represents the content of a browser's window and you add text to the Web page using the *write()* or *writeln()* methods.

```
document.write("<p>Places to visit:</p>");
document.write("<ul>");
document.write("<li>Cape Town Waterfront</li>");
document.write("<li>Pilansberg National Park</li>");
```



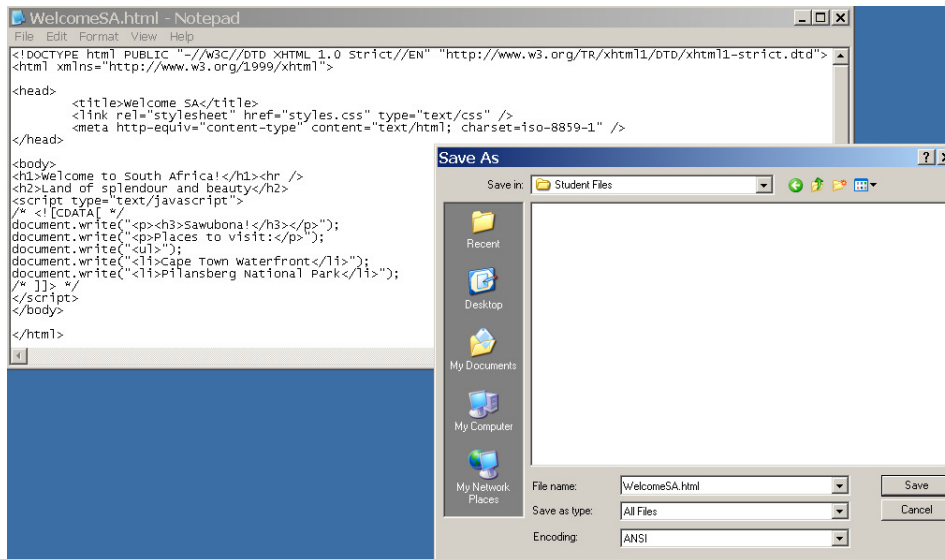
```

/* ]]> */
</script>
</body>

</html>

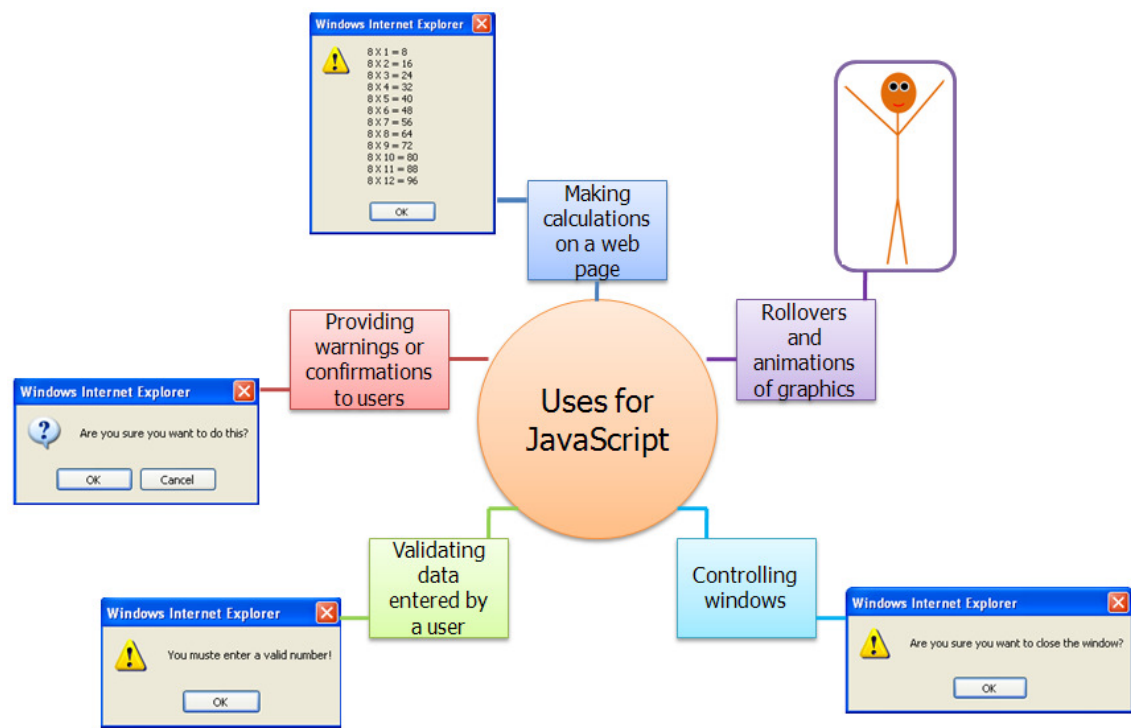
```

**Step 2:** Save the code you entered as welcomeSA.html.





Uses for JavaScript



# Logic Design

## Introduction

When you are learning to speak a new language like French or Zulu, you can learn a few phrases by heart without really knowing what each word means. You will, however, never be able to fully communicate in French or in Zulu.

The same principal goes for learning a programming language. Unless you truly understand what the problem is about, how can you tell the computer, in its language, how to solve the problem?

## Reading a problem statement

Reading a problem statement like the one below might seem to be a simple task. The catch is to understand what it is you are reading, because if you don't understand it, how can you ever hope to solve it.

The following steps will help you in understanding a problem statement:

1. Read the problem entirely.
2. Re-read the problem and write down all the words you do not know the meaning of.
3. Use a dictionary, like the Oxford Dictionary, to find out what the words mean and write their meaning next to them.
4. Re-write the problem in your own words.

This might seem a very long process, but as your knowledge grows, the process will grow shorter until you no longer need it.

## Example

**Step 1:** Read the problem statement.

*Design a simple website that prompts the user to enter 4 integers. Using the information entered by the user determine the sum and the average of the 4 numbers.*

**Step 2:** Re-read the problem and write down all the words you do not know the meaning of.

*Design – plan a program*

*Prompts – ask to do something*

*Integer – whole number without decimal spaces*

*Determine – calculate*

**Step 3:** Use a dictionary, like the Oxford Dictionary, to find out what the words mean and write their meaning next to them.

**Step 4:** Re-write the problem in your own words.

*Plan a simple website that asks the user to enter 4 whole numbers. Using the information entered by the user calculate the sum and average of the 4 numbers.*

## Identifying key elements in a problem statement

Most problem statements will require you to do some kind of mathematical operation. These operations will not always be provided as a mathematical statement e.g.  $a = b + c$  and therefore you need to know what to look for in the problem statement.

The following is “key” words to look for in a problem statement that will indicate certain mathematical operations:

#### **Addition (+)**

increased  
by  
more than  
combined  
together  
total of  
sym  
added to

#### **Subtraction (-)**

decreased by  
minus  
less than  
difference  
between / of  
fewer than

#### **Multiplication (x)**

of  
times  
multiplied by  
product of  
by a factor of

#### **Division (/)**

per  
a  
out of  
ratio of  
quotient of  
percent (divide by 100)

#### **Equals (=)**

is / are  
was / were  
will be  
gives  
yields  
sold for

These are just some of the key words and it is advised that, as you go along with your studies, you add other key words you come across to the table.

#### **Example**

I drove 90 kilometres on 10 litres of petrol, so I got 9 kilometres **per** litre

90 / 10

He makes R1.50 an hour **less** than me

Wage – R1.50

What is the **sum** of 8 and y?

8 + y

The **quotient** of y and 13 is ...

y / 13

The **ratio** of 9 **more than** y to x is ...

(9 + y) / x

### **Planning the solution**

Now that you have the ability to read a problem statement and understand it, to identify key elements in a problem statement and are able apply the basic operators and operands, you can examine the problem statement and find the information needed to solve the problem. This is done by asking some basic questions like:

- What do they want me to do?
- How should this be done?
- What do I need to be able to do this?

These are just basic questions, but they will be the beginning of the solution to the problem statement. As you move forward you will notice that each answer you get to these basic questions will broken down into smaller and smaller pieces, which in the end will lead to a working solution.

#### **Example**

*Plan a simple website that asks the user to enter 4 whole numbers. Using the information entered by the user calculate the sum and average of the 4 numbers.*

*What do they want me to do?*



Calculate the sum and average of the 4 numbers

*How should this be done?*

Number 1 + Number 2 + Number 3 + Number 4 = Sum

(Number 1 + Number 2 + Number 3 + Number 4) / 4 = Average

*What will I need to be able to do this?*

Number 1, Number 2, Number 3, Number 4

## Pseudo code

Pseudo code is NOT a programming language; it is more a thought process. It does not focus on writing code in a particular programming language. The idea of pseudo code is to draw a "map" in order to solve a problem. Pseudo code is written in plain English, in a format that can be converted into a programming language.

There is no formal standard for pseudo code; this means that like programming, no two people's pseudo code will ever look the same. Remember that pseudo code is merely writing down the steps to solve a problem.

Take a look at the following recipe:

### Ingredients

1/2 cup diced onions	2 1/2 cup chicken broth
1/2 cup diced celery	5 cups milk
1/2 cup diced green pepper	1 cup half and half cream
2 cloves minced garlic	1 cup creamy peanut butter
3T butter	2T Tomato Paste
3 to 4T all purpose flour	3T Soy sauce

### Process

Chopped peanuts for garnish

Sauté onions, celery, green pepper and garlic in the butter until onions are translucent.

Add flour and cook for 2 to 3 minutes, until you have a light roux.

Stir in water.

Add milk slowly.

Bring to a boil.

Add peanut butter, tomato paste (optional), and soy sauce.

Reduce heat and simmer approximately 15 minutes.

Serve hot with chopped peanuts for garnish.

The ingredients can be seen as the variables that we will be using in our program and the process as the pseudo code. As you will notice the process is written in normal English and explains exactly what you should do in every step, the same method is followed when writing pseudo code.

## How to write pseudo code

There are six basic computer operations

1. A computer can receive information  
Read (information from a file)  
Get (information from the keyboard)
2. A computer can put out information  
Write (information to a file)  
Display (information to the screen)
3. A computer can perform arithmetic  
Use actual mathematical symbols or the words for the symbols  
Add number to total  
Total = total + number

- + , - , \* , /  
Calculate, Compute also used
- 4. A computer can assign a value to a piece of data  
3 cases
  - to give data an initial value  
Initialize, Set
  - To assign a value as a result of some processing  
=?  
 $x = 5 + y$
  - to keep a piece of information for later use  
Save, Store
- 5. A computer can compare two piece of information and select one of two alternative actions  
IF condition THEN  
    some action  
ELSE  
    alternative action  
ENDIF
- 6. A computer can repeat a group of actions
  - WHILE condition (is true)  
    some action  
    ENDWHILE
  - FOR a number of times  
    some action  
    ENDFOR

The last two operations will be discussed in more detail in following chapters.

### Example

*Plan a simple website that asks the user to enter 4 whole numbers. Using the information entered by the user calculate the sum and average of the 4 numbers.*

Actions needed to solve the problem

*CalculateNumber Program*

*Get input*

*Calculate sum*

*Calculate average*

*Display output*

Design the solution

How do I get the input?

Ask the user to enter the numbers

*Get Number1*

*Get Number2*

*Get Number3*

*Get Number4*

*Do the Calculations*

What does the calculation look like?

$Sum = Number1 + Number2 + Number3 + Number4$

$Average = Sum / 4$

What is the outcome of the program?

*Display Sum*

*Display Average*

## Storyboard

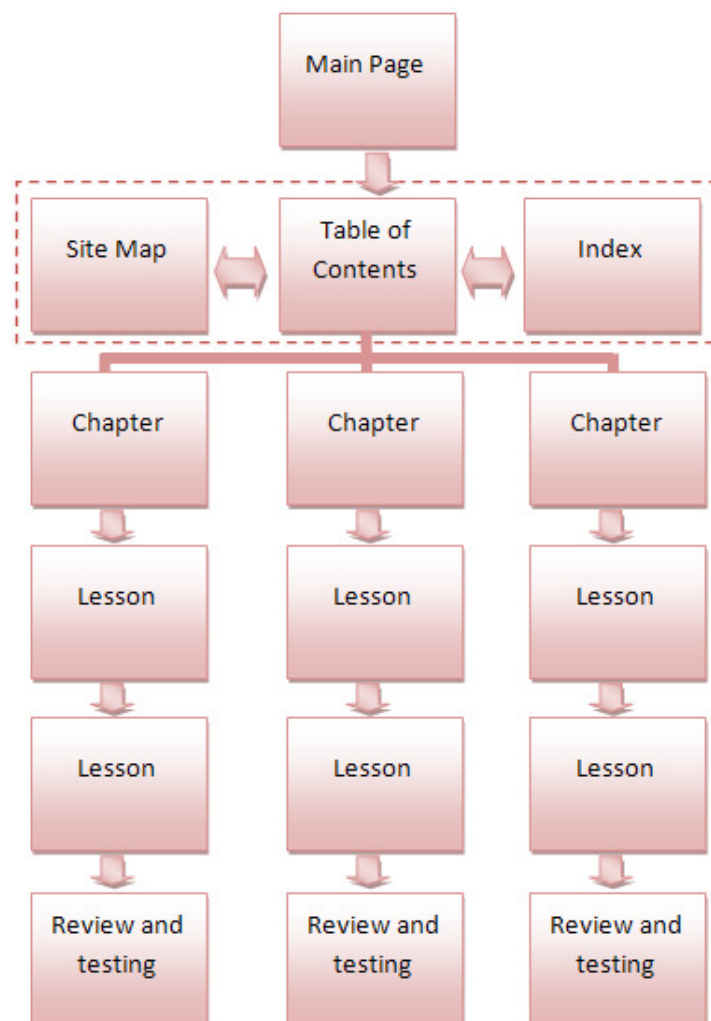
Once you've designed the solution for the programming code you need to think about the structure and logic of the content of your Web site and how the user should navigate through the site. Planning the layout of your Web site is done by creating a storyboard. This is one of the most important steps in planning your website.

You can draw a storyboard using pen and paper, flow chart software or even post-it notes to help you plan the structure of the website visually. The aim of the storyboard is to help you plan the navigation of the site and to assist you in visualizing the entire Web site.

There are various different sample structures for storyboards; you need to decide which of the sample structures will suite your needs and instructions best. Following is a short description and an example of each of these sample structures. You may use these examples as a starting point and adjust them to suite your specific needs by adding more or having fewer pages, sections, topics or links.

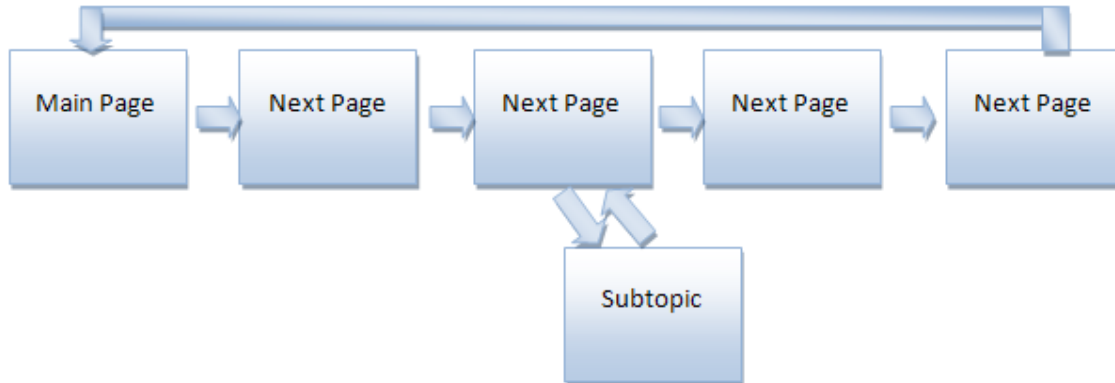
### Tutorial Structure

The tutorial structure builds on the linear structure and is mainly used for Web sites that contain lessons, tutorials or task-oriented pages. The user can choose which lesson to go to, work on it, leave and return later. The site map, table of content and index are linked to and from all the pages in the site.



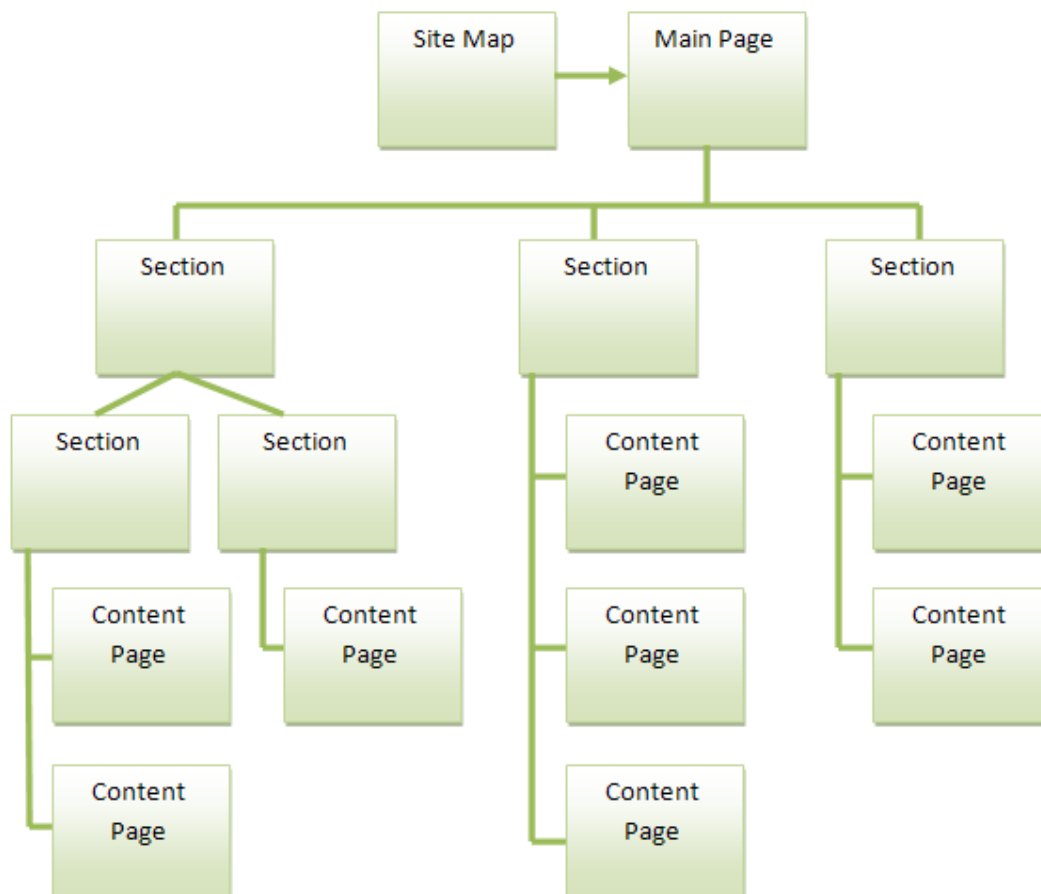
## Linear Structure

This structure is like reading a book; you can go one page forward or backwards. If you choose you may have a link from each page back to the main page, you may have links to subtopic, but they return to the "calling" page.



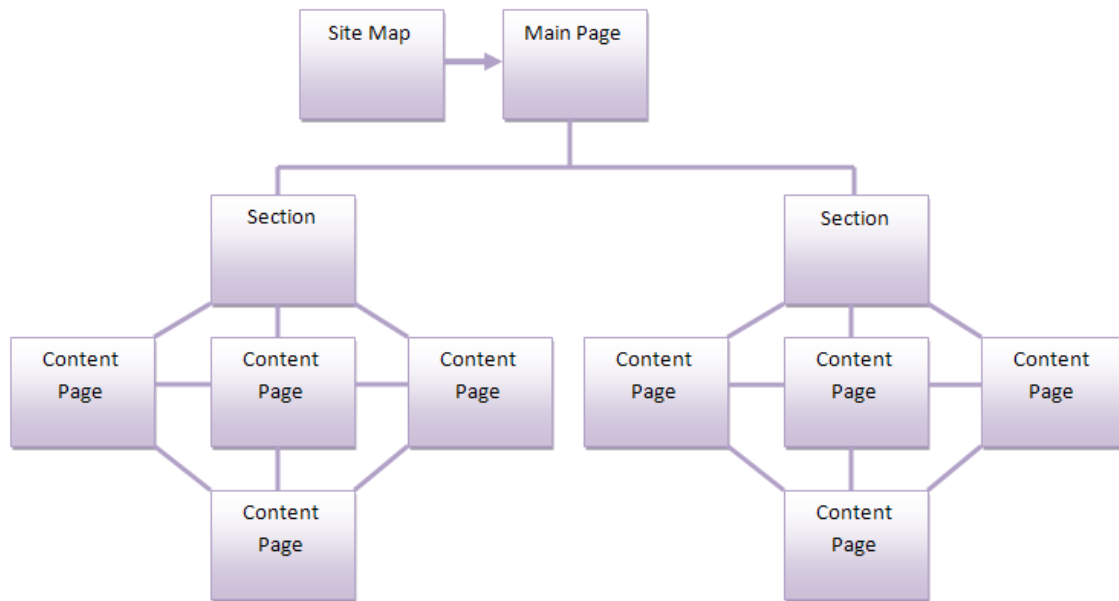
## Hierarchical Structure

This is the most largely used navigational structure. The Web site is broken down into logical sections and each section contains its own content. Navigation is linear through the site; the user goes from one page to the next and back, and from one section to the next and back. The site map allows the user to navigate throughout the site and each page has a navigation bar that takes the user to the site map, main page or section page.



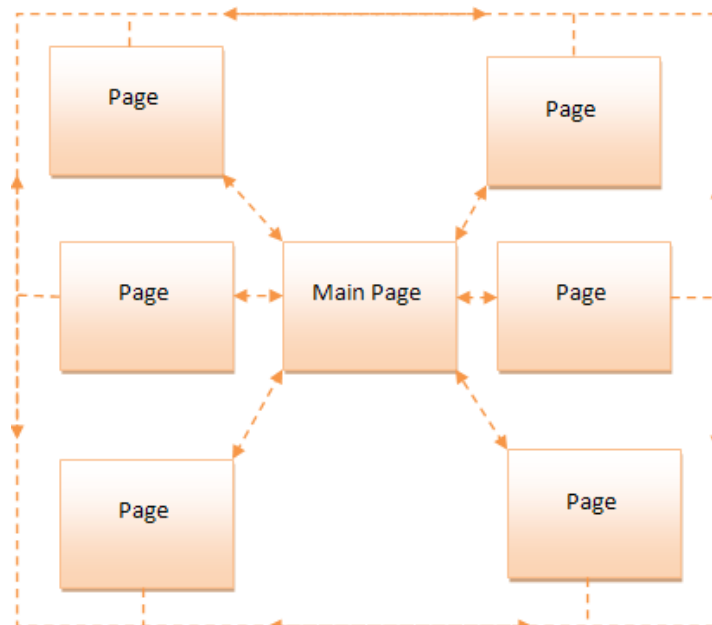
## Cluster Structure

This structure is almost the same as the hierarchical structure in the sense that information is broken down into logical sections. The difference is that there is no linear order inside of each of the sections. The user may move between the content pages in any order. The site map allows the user to navigate throughout the site and each page has a navigation bar that takes the user to the site map, main page or section page.



## Web Structure

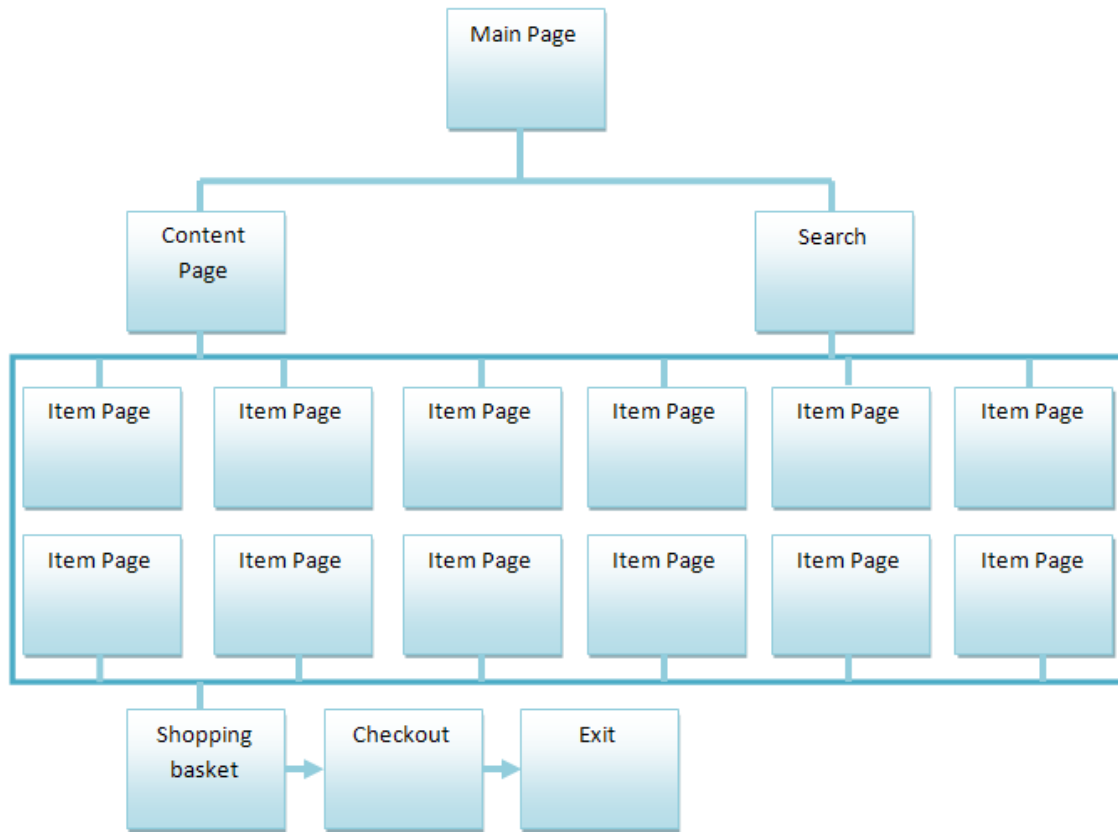
This structure is non-linear and jumps from any page to any other page. Clear location information and a standardized navigation bar must be used to inform the user so the user will know at all times where in the Web site he or she is and where they can go to from there.





## Catalog Structure

This structure is used for electronic shopping websites. The user searches for a specific product or browses through the site to find it, views the products page and can then add the product to a shopping basket if they want to purchase it. After selecting all the products they which to purchase the can review the shopping basket after which they check out and pay for their shopping.



## Step-by-Step create the JavaScript Program

*Design a simple website that prompts the user to enter 4 integers. Using the information entered by the user determine the sum and the average of the 4 numbers.*

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Number Calculations</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

  <script type="text/javascript">
    /* <![CDATA[ */

    function calculate() {
```

**Comment 1:** Definition of the function named calculate()

```
var sum = parseInt(calc.number1.value) + parseInt(calc.number2.value) +
parseInt(calc.number3.value) + parseInt(calc.number4.value);
```

**Comment 2:** Declare a variable named sum and assign the value of the sum of the 4 numbers to the variable. parseInt() function ensures value is converted to integer. To get the actual value entered by the user use the form name, input field name and then the value e.g. calc.number1.value.

```
var average = sum / 4;
```

**Comment 3:** Calculate the average of the four numbers the user entered.

```
document.writeln("Sum of the four numbers = " + sum + "<br />");  
document.writeln("Average of the four numbers = " + average);
```

**Comment 4:** Display the output

```
}  
  
/* ]]> */  
</script>  
</head>  
  
<body>  
  <form name = "calc" action = "">
```

**Comment 5:** Create the form the user must use to enter the data

```
Enter number 1: <input type = "text" name = "number1" /><br />  
Enter number 2: <input type = "text" name = "number2" /><br />  
Enter number 3: <input type = "text" name = "number3" /><br />  
Enter number 4: <input type = "text" name = "number4" />
```

**Comment 6:** Create the fields into which the user enter the numbers

```
<p><input type = "button" value = "Calculate" onclick = "calculate()" /></p>
```

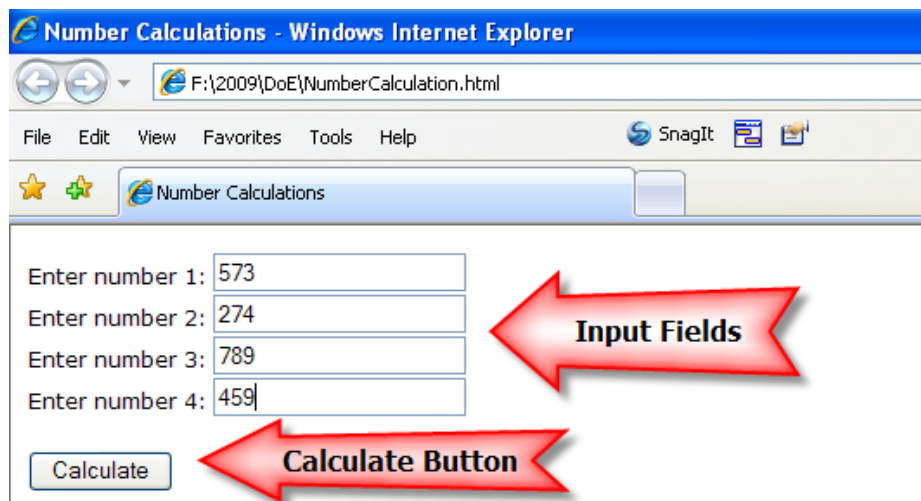
**Comment 7:** Create the button which calls the calculate function in the script

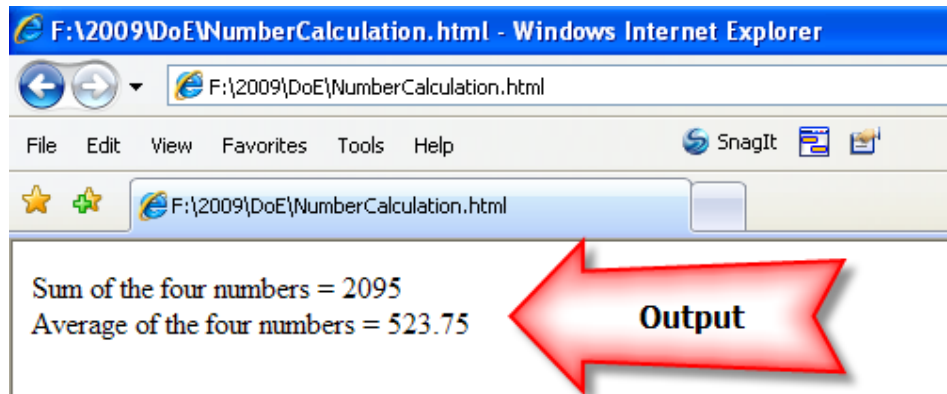
```
</form>  
</body>  
  
</html>
```

**Step 2:** Save the code you entered as NumberCalculation.html.

**Step 3:** Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.

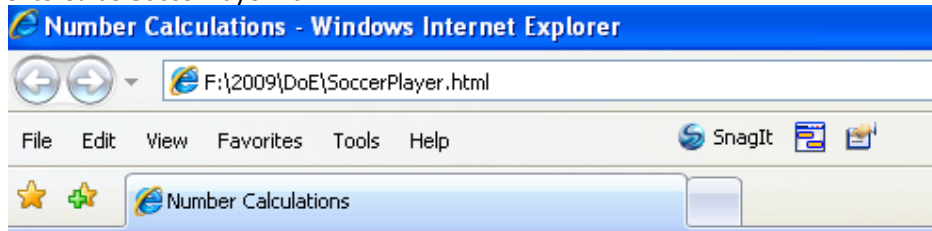
**Step 4:** Once the code has been validated open the browser and view your code.





### Exercises:

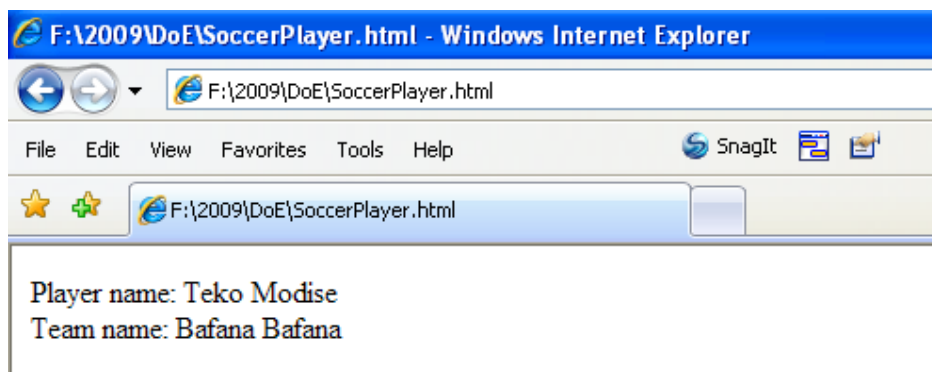
1. Develop a simple Web page where the user is requested to enter the name of his or her favorite soccer player as well as the name of the team they are play for. Add a button that, once pressed, shows the name of the soccer player and his team name. Save the code you have entered as SoccerPlayer.html



## Soccer Information

Enter the player name:

Enter the team name:



## Data Types

### Variables

A variable is a symbol or name that stands for a value. It is a placeholder in the computer's memory where a certain value can be stored. For example in the expression:

A + B

A and B are variables.


Definition of variable: often changing or likely to change; not consistent; able to be changed; able to take on different values.

Variables are a very important part of programming because they enable programmers to write flexible programs. Instead of entering data directly into a program, variables are used to represent the data. When the program is running, the variables are replaced with data. This makes it possible to use the same program for different sets of data. E.g. all banks can use the same system even though they have different customers.

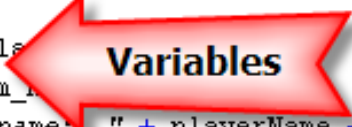
Each variable has a unique name and a data type. Every time a program is run that has a specific variable name in it, the value of the variable might change.

#### Example:

```
function calculate() {  
    var sum = pars  
    var average = sum / 4  
    document.writeln("Sum of the four numbers: " + sum);  
    document.writeln("Average of the four numbers: " + average);  
}
```



```
function display() {  
    var playerName = soccer.playerName;  
    var teamName = soccer.teamName;  
    document.writeln("Player name: " + playerName + "<br />");  
    document.writeln("Team name: " + teamName);  
}
```



### Variable names

Most languages have rules like starting a person's name with capital letters e.g. John Schoeman. In the same fashion there are rules to follow when naming variables. Naming conventions are the rules that will make your programs readable and understandable to other programmers. They are the norm by which all variable names should be chosen.

#### Names should have meaning

When the variable is named, it should indicate what the name is representing e.g. salary, pension, last name.

#### Avoid names with more than one meaning

The variable name should clearly indicate the meaning of the variable e.g. number is not a clear name as it can contain any number, be specific e.g. personel\_number, phone\_number.

### **Keep the name as short as possible**

It must be long enough to be descriptive, but short enough to remember and use. Good variable names use one word or by putting together two or three word, separated by an underscore "\_" e.g. start\_time, phone\_number.

### **Using Uppercase and lowercase**

Over the years programmers invented special naming rules for variables to distinguish them at a glance. All lower case words separated by an underscore and upper and lower case mixed with no separators indicate variables e.g. total\_salary, TotalSalary.

### **Be consistent**

When using all lower case or upper and lower case mixed to indicate variables, apply the same rule to all variable names. Do not make some only lower case and others upper and lower case mixed.

### **Group related elements**

When a group of elements refer to the same thing, give them related names to indicate they belong together e.g. employee\_name, employee\_number, employee\_adres.

### **Short names**

In some cases it is allowed to use a single character as a variable name. Short names such as x, y and I are acceptable when their meaning is clear and a longer name will not add information or clarity e.g. "i" is frequently used in programming for index.

### **Don't use special symbols**

Most programming languages don't allow the use of special symbols like %, \$, etc. in variable names e.g. \$\_value is an invalid name.

### **No spaces**

When using more than one word as a name they are not allowed to be separated by a space as the computer will then read them as two separate things e.g. inch per squire is an invalid name and should be inch\_per\_squire.

### **No numbers**

A name may not exist of numbers only e.g. 13 is an invalid name. Numbers may be used with characters e.g. month1, month2.

### **Reserved words**

Certain words in programming languages are reserved by the language to perform a specific task. These words may not be used as names e.g. getdate.

### **Avoid similar names**

Don't use names that have slight differences, make each name obviously different from the other e.g. instead of total and totals use daily\_total and monthly\_total.

### **Describing variables**

Choosing proper names for variables creates programs that are easy to read and maintain, this, however, does not work alone. Each variable need to be described in order to clarify exactly what its purpose is. The solution to this problem is by following each variable with a comment describing its purpose when the variable is declared.

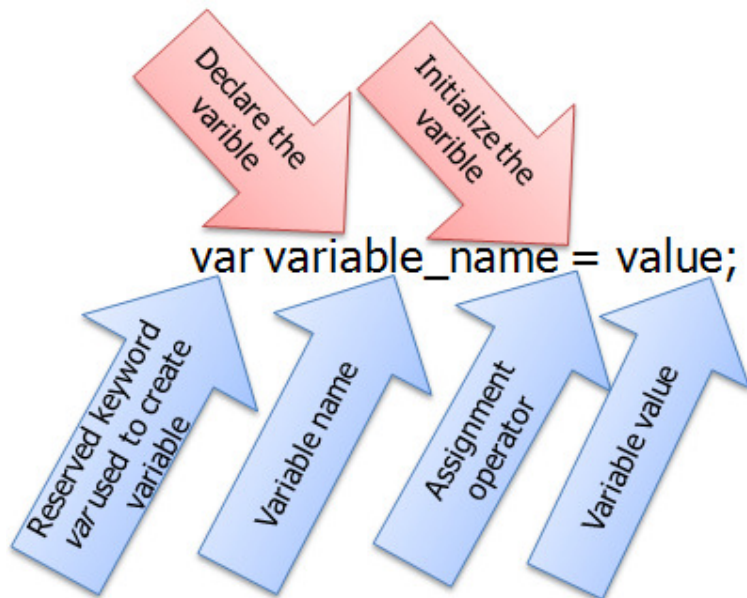
### **Example:**

salary	/* basic salary before deductions
tax	/* percentage tax payable
emp_num	/* unique employee number

## Assigning variables

When a value is assigned to a variable the value is stored in the variable. The equal sign = is used to assign the value to the variable

Declare variable and initialize:



When a literal string (text) is assigned to a variable it should be enclosed in quotation marks for example:

```
var my_name = "Thabong Modisang";
```

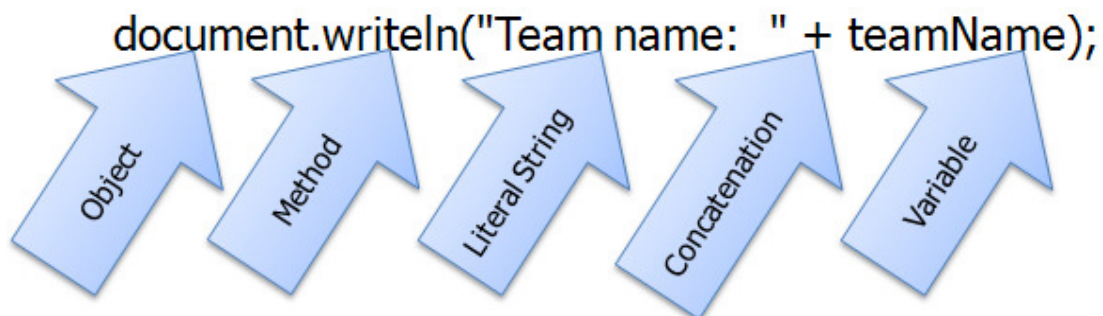
When a numeric value is assigned to a variable it is not enclosed in quotation marks, for example:

```
var my_age = 25;
```

Good programming practice is to always initialize a variable when you declare it.

## Displaying Variables

To display a variable on screen pass the variable name to the `document.write()` or `document.writeln()` methods. The + (plus sign) is used to combine a literal string with a variable in order to display the output on the screen, for example:



## Changing Variables

A variables value can be changed at any point in a script. This is done using a statement that includes the variable's name, followed by = (equal sign), followed by the value to be assigned to the variable, for example:

```
var my_age = 35;
var my_age = var my_age + 5;
```

### Step-by-Step create the JavaScript Program

*Develop a simple Web page in which you request the user to enter his or her age. The program should then calculate what the user's age was 10 year ago as well as what age the use would be in another 10 years.*

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
    <title>Age Calculation</title>
    <link rel="stylesheet" href="styles.css" type="text/css" />
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

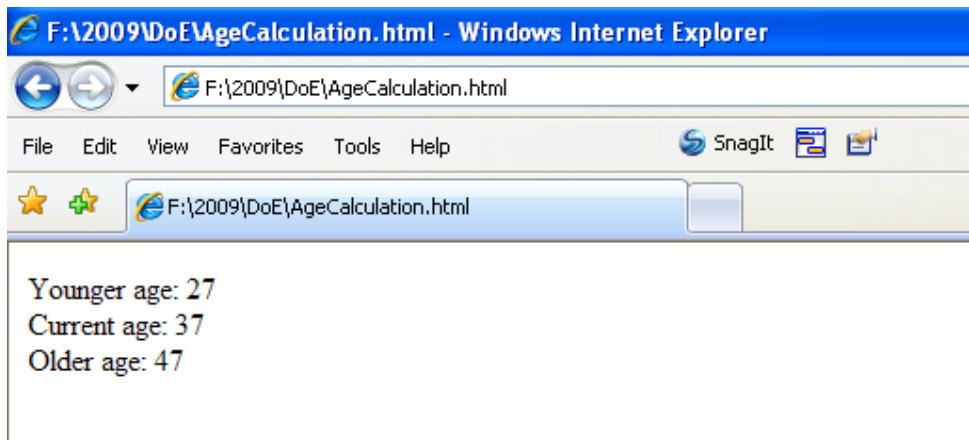
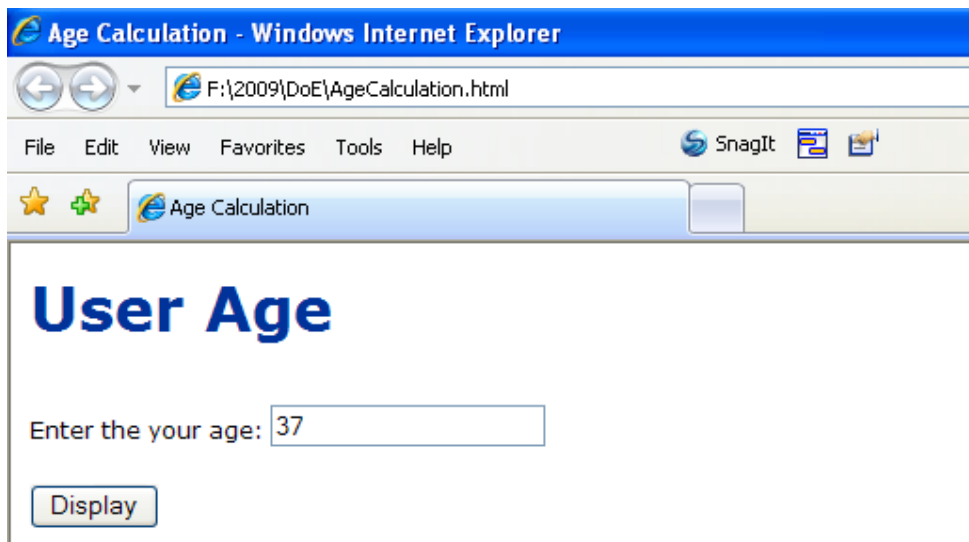
<script type="text/javascript">
/*  */

function display() {
    var current_age = parseInt(age.currentAge.value);
    var younger_age = current_age - 10;
    var older_age = current_age + 10;

    document.write("Younger age: " + younger_age + "&lt;br /&gt;");
    document.write("Current age: " + current_age + "&lt;br /&gt;");
    document.write("Older age: " + older_age);
}
/* ]]&gt; */
&lt;/script&gt;
&lt;/head&gt;

&lt;body&gt;
&lt;h1&gt;User Age&lt;/h1&gt;
    &lt;form name = "age" action = ""&gt;
        Enter the your age: &lt;input type = "text" name = "currentAge" /&gt;&lt;br /&gt;
        &lt;p&gt;&lt;input type = "button" value = "Display" onclick = "display()" /&gt;&lt;/p&gt;
    &lt;/form&gt;
&lt;/body&gt;

&lt;/html&gt;</pre></div><div data-bbox="147 756 563 772" data-label="Text"><p><b>Step 2:</b> Save the code you entered as AgeCalculation.html.</p></div><div data-bbox="147 784 812 813" data-label="Text"><p><b>Step 3:</b> Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.</p></div><div data-bbox="147 826 713 843" data-label="Text"><p><b>Step 4:</b> Once the code has been validated open the browser and view your code.</p></div><div data-bbox="849 886 886 936" data-label="Page-Footer"><p>100</p></div>
```



## Variable Scope

A variable's scope depends on where in the program it is declared and thus where it can be used. Global variables are declared outside a function and are available to all other functions in the program. Local variables are declared inside a function and are only available for use in the function it is declared in. Function will be discussed later.

## Types of Data

The data type is a specific kind of value that a variable contains. The kind of operation that can be performed on a variable is determined by its type.

Type	Description	Example
<b>Number</b>	Used for arithmetic calculations	
<b>Integer</b>	Positive or negative number without decimal places	60, -15, 100001, -785
<b>Floating-point</b>	Positive or negative number with decimal places	60.01, -15.72, 100001.0001, 785.867
<b>Exponential notation</b>	Shortened format for writing large positive and negative numbers with or without decimal places	1e24, 1e-18, 1e15, 13-21
<b>String</b>	Text consisting of all the letters of the	Abc, !@#\$\$%^&*()



	alphabet and special characters	
<b>Boolean</b>	Logical values used for deciding which part of a program should execute	True or False
<b>Undefined</b>	A variable that has never been assigned a value used to determine if a variable is used in another part of the program	
<b>Null</b>	An empty value to ensure a variable does not contain data	null

In JavaScript you do not need to declare a variable's type as the interpreter automatically detects the type of data stored in a variable. It is, however, very important to know the types of variable to use them correctly.

### Example

```
variableType = "My JavaScript Programs";    // String
variableType = 345689;                     // Integer number
variableType = 345.990                     // Floating-point number
variableType = true;                       // Boolean
variableType = null;                       // null
```

### Using Data Entered by the User

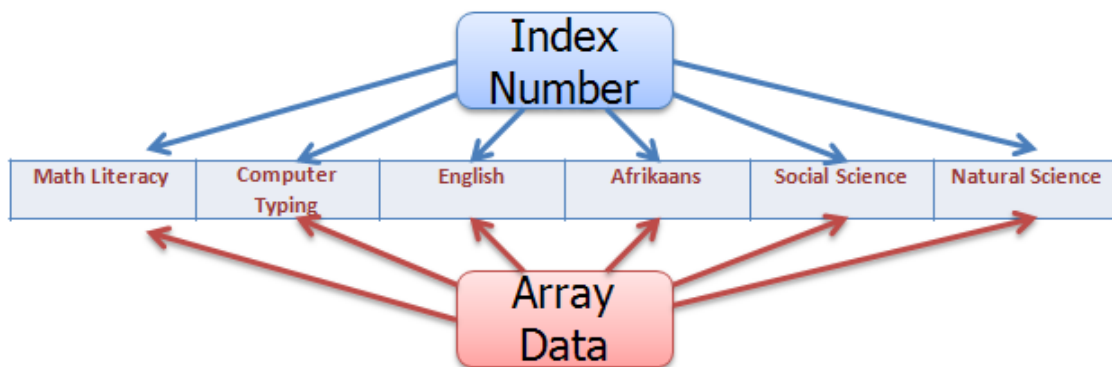
All variable values entered by the user into a text box are considered to be of variable type String. If you want to use a number entered by a user for calculations you need to convert the string to an integer using the *parseInt()* function, or if you want to use the string as a floating-point number you must convert it using the *parseFloat()* function. For example:

```
var current_age = parseInt(age.currentAge.value);
```

### Arrays

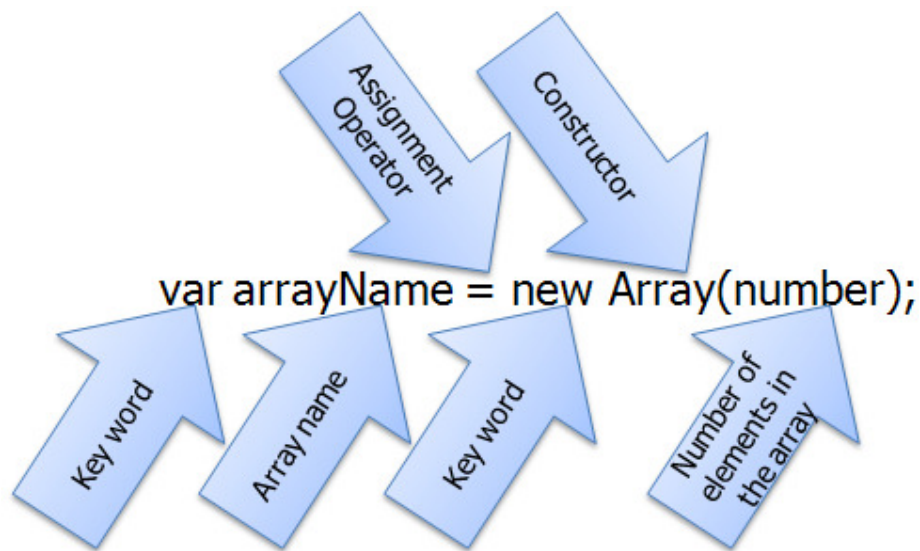
An array is a collection of variables within a single variable. An array is normally used to store a group or list of related information in a single place.

schoolSubjects array



### Declaring and Initializing Arrays

JavaScript has an object called *Array* which is used to represent arrays, the *Array* object has an *Array()* constructor which is used to declare the array with. The constructor informs JavaScript to build an array.



### Example

```
schoolSubjects = new Array();
schoolSubjects[0] = "Math Literacy";
schoolSubjects[1] = "Computer Typing";
schoolSubjects[2] = "English";
schoolSubjects[3] = "Afrikaans";
schoolSubjects[4] = "Social Science";
schoolSubjects[5] = "Natural Science";
```

### Accessing and Modifying Array Elements

To access the value of an array element you include the elements index number in brackets.

```
document.writeln(schoolSubjects[2]); // prints "English"
```

Changing the value of an array element is done the same way you would change the value of any variable, except you include the index number of the element.

```
schoolSubjects[4] = "Economic Management" // changes "Social Science" to "Economic Management"
```

### How many elements in the array

In most cases the programmer will not know how many elements an array will contain, it is thus very important to be able to determine how many elements there are to know which one to refer to. The *Array* class has a single property named *length*, this property counts the number of elements in an array. If you add the property to the name of the array it will count the number of elements in the array.

```
schoolSubjects.length // gives the answer 6
```

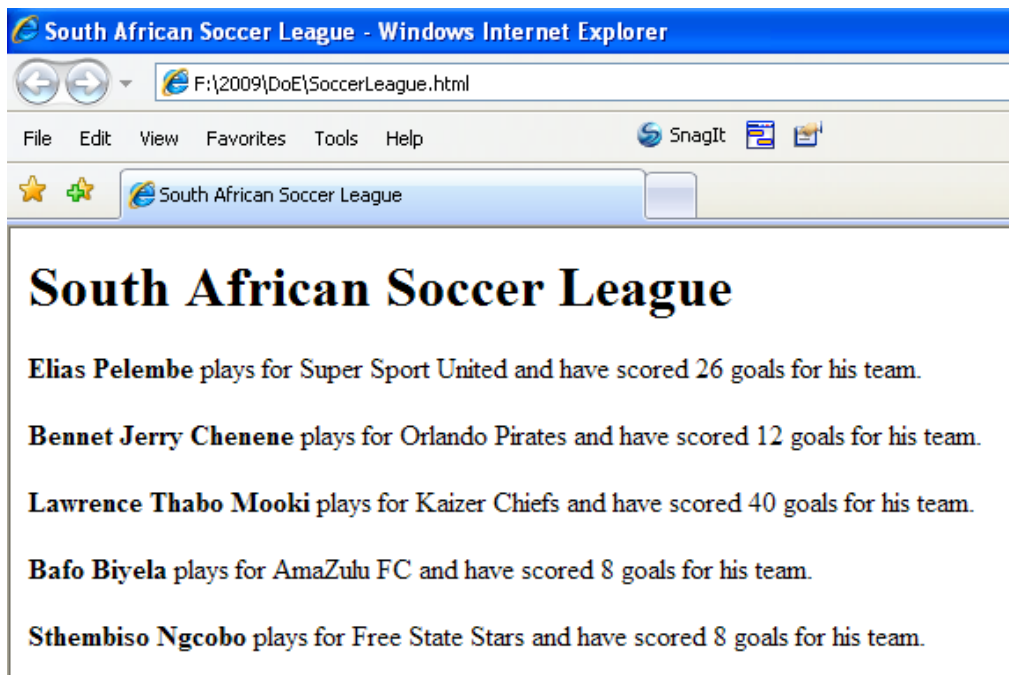
### Step-by-Step create the JavaScript Program

*Develop a simple Web page in which you enter the names of 5 soccer players in an array, the name of the teams the soccer players play for in a separate array and the number of goals each player have scored during the season in a third array. Print each players name, team name and score in a single line.*

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>South African Soccer League</title>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
<link rel="stylesheet" href="js_styles.css" type="text/css" />
</head>
<body>
<h1>South African Soccer League</h1>
<script type="text/javascript">
/*  */
var player = new Array(6);
player[0] = "Elias Pelembe";
player[1] = "Bennet Jerry Chenene";
player[2] = "Lawrence Thabo Mooki";
player[3] = "Bafo Biyela ";
player[4] = "Sthembiso Ngcobo";
</pre>
</div>
<div data-bbox="147 288 680 304" data-label="Text">
<p><b>Comment 1:</b> Create the array containing the names of the 5 soccer players.</p>
</div>
<div data-bbox="147 312 380 379" data-label="Text">
<pre>
var team = new Array(10);
team[0] = "Super Sport United";
team[1] = "Orlando Pirates";
team[2] = "Kaizer Chiefs";
team[3] = "AmaZulu FC";
team[4] = "Free State Stars";
</pre>
</div>
<div data-bbox="147 389 720 404" data-label="Text">
<p><b>Comment 2:</b> Create the array containing the team names of the 5 soccer players.</p>
</div>
<div data-bbox="147 413 336 479" data-label="Text">
<pre>
var goals = new Array(6);
goals[0] = 26;
goals[1] = 12;
goals[2] = 40;
goals[3] = 8;
goals[4] = 8;
</pre>
</div>
<div data-bbox="147 489 806 504" data-label="Text">
<p><b>Comment 3:</b> Create the array containing the number of goals each player scored for his team.</p>
</div>
<div data-bbox="147 513 849 536" data-label="Text">
<pre>
document.write("&lt;p&gt;&lt;strong&gt;" + player[0] + " &lt;/strong&gt;" + " plays for " + team[0] + " and have
scored " + goals[0] + " goals for his team.&lt;/p&gt;");
</pre>
</div>
<div data-bbox="147 546 674 561" data-label="Text">
<p><b>Comment 4:</b> Display the information for each of the players on the screen.</p>
</div>
<div data-bbox="147 569 849 700" data-label="Text">
<pre>
document.write("&lt;p&gt;&lt;strong&gt;" + player[1] + " &lt;/strong&gt;" + " plays for " + team[1] + " and have
scored " + goals[1] + " goals for his team.&lt;/p&gt;");
document.write("&lt;p&gt;&lt;strong&gt;" + player[2] + " &lt;/strong&gt;" + " plays for " + team[2] + " and have
scored " + goals[2] + " goals for his team.&lt;/p&gt;");
document.write("&lt;p&gt;&lt;strong&gt;" + player[3] + " &lt;/strong&gt;" + " plays for " + team[3] + " and have
scored " + goals[3] + " goals for his team.&lt;/p&gt;");
document.write("&lt;p&gt;&lt;strong&gt;" + player[4] + " &lt;/strong&gt;" + " plays for " + team[4] + " and have
scored " + goals[4] + " goals for his team.&lt;/p&gt;");
/* ]]&gt; */
&lt;/script&gt;
&lt;/body&gt;
&lt;/html&gt;
</pre>
</div>
<div data-bbox="147 714 556 730" data-label="Text">
<p><b>Step 2:</b> Save the code you entered as SoccerLeague.html.</p>
</div>
<div data-bbox="147 742 812 771" data-label="Text">
<p><b>Step 3:</b> Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.</p>
</div>
<div data-bbox="147 785 713 801" data-label="Text">
<p><b>Step 4:</b> Once the code has been validated open the browser and view your code.</p>
</div>
<div data-bbox="849 886 886 936" data-label="Page-Footer">
<p>104</p>
</div>
```



### Exercise

Using the code in the Step-by-Step example above modify the code to display how many elements there are in each of the arrays as well as how many array elements there are in total. Add the "Spurs" team name to the team array. Save the code as SoccerLeague.html



## Building Expressions

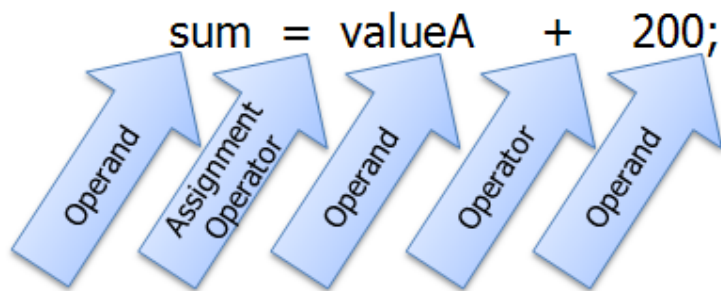
### Introduction

Having data is of no use if you cannot do anything with it. You have to turn your data into information for it to become useful. The process of turning the data into information is done using expressions.

### Expressions

An expression is a combination of values, variables, operators and other expressions which the JavaScript interpreter evaluates to get a result.

The values and variables in an expression is referred to as the operands. The symbols, like + (plus) and / (divide), in the expression is referred to as the operators.



### Operators

Operator Name	Operator	Description	Example
<b>Arithmetic Binary Operators – used to perform mathematical calculations</b>			
<b>Addition</b>	+	Adds operands	<code>10 = 6 + 4</code>
<b>Subtraction</b>	-	Subtract one operand from another	<code>4 = 10 - 6</code>
<b>Multiplication</b>	*	Multiplies one operand by another	<code>12 = 3 * 4</code>
<b>Division</b>	/	Divides one operand by another	<code>3 = 12 / 4</code>
<b>Modulus</b>	%	Divides one operand by another and returns the remainder	<code>1 = 13 % 4</code>
<b>Exponentiation</b>	^	Multiplies a number by itself a certain number of times	<code>2^3 = 8</code>
<b>Arithmetic Unary Operators – used to perform mathematical calculations with a single operand</b>			
<b>Increment</b>	++	Increases operand by value of one	<code>++5 = 6</code>
<b>Decrement</b>	--	Decreases operand by value of one	<code>--5 = 4</code>
<b>Negation</b>	-	Returns opposite value (negative or positive) of an operand	<code>-(-5) = 5</code>
<b>Assignment Operators – Used to assign a value to a variable</b>			
<b>Assignment</b>	=	Assigns the value of the right operand to the left operand	<code>A = 6</code>
<b>Compound addition assignment</b>	+=	Combines the value of the right operand with the value of the left operand or adds the value of the right operand to the value of the left operand and assigns the new value to the left operand	<code>A = 6</code> <code>B = 7</code> <code>A += B</code> <code>A = 13</code>
<b>Compound subtraction</b>	-=	Subtracts the value of the right operand from the value of the left operand and assigns the new	<code>A = 13</code> <code>B = 7</code>

<b>assignment</b>		value to the left operand	A -= B A = 6
<b>Compound multiplication assignment</b>	*=	Multiplies the value of the right operand from the value of the left operand and assigns the new value to the left operand	A = 6 B = 7 A *= B A = 42
<b>Compound division assignment</b>	/=	Divides the value of the right operand from the value of the left operand and assigns the new value to the left operand	A = 42 B = 7 A /= B A = 6
<b>Compound modulus assignment</b>	%=	Divides the value of the right operand from the value of the left operand and assigns the remainder value to the left operand	A = 43 B = 7 A%= B A = 1
<b>Comparison Operators – used to compare two operands, a Boolean value of true or false is returned</b>			
<b>Equal</b>	==	Returns true if operands are equal	5 = 5
<b>Strict equal</b>	===	Returns true if operands are equal and of same type	A = A
<b>Not equal</b>	!=	Returns true if the operands are not equal	5 != 4
<b>Strict not equal</b>	!==	Returns true if operands are not equal or not of the same type	A !== B or A !== a
<b>Greater than</b>	>	Returns true if left operand is greater than right operand	7 > 9
<b>Less than</b>	<	Returns true if left operand is less than right operand	9 < 7
<b>Greater than or equal to</b>	>=	Returns true if left operand is greater than or equal to right operand	7 >= 9 or 7>=7
<b>Less than or equal to</b>	<=	Returns true if left operand less than or equal to right operand	9 <= 7 or 7 <=7
<b>Conditional Operator – executes one of two expressions based on results of a conditional expression</b>			
	?	If the conditional expression is true the first expression is executed, else the second expression is executed	(7 >=9) ? a + 1 : b + 2
<b>Logical Operators – used for comparing two Boolean operands</b>			
<b>And</b>	&&	Returns true if both left operand and right operand return value of true	7 > 9 && 9 < 7
<b>Or</b>		Returns true if either left operand or right operand returns value of true	7 > 9    10 > 9
<b>Not</b>	!	Returns true if an expression is false and returns false if and expression is true	!(7 > 9) = false

### Step-by-Step create the JavaScript Program

Develop a simple Web page in which you request the user to enter his or her age as well as his or her spouses' age. The Web page should then display the following information:

- Who is the eldest?
- What is their combined age?
- How old will the husband be if you multiply his age by 2?
- How old will the wife be if you divide her age by 2?
- How many years remain if you divide the husband's age by 3?

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Age Calculation</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

  <script type="text/javascript">
    /*  */

    function display() {
      var husband_age = parseInt(age.husbandAge.value);
      var wife_age = parseInt(age.wifeAge.value);
</pre>
</div>
<div data-bbox="148 257 743 272" data-label="Text">
<p><b>Comment 1:</b> Use the <i>parseInt()</i> function to ensure the value is converted to integer.</p>
</div>
<div data-bbox="203 281 609 325" data-label="Text">
<pre>
      var eldest;
      (husband_age &gt; wife_age) ?
        eldest = "The husband is older than the wife" :
        eldest = "The wife is older than the husband";
</pre>
</div>
<div data-bbox="148 335 850 364" data-label="Text">
<p><b>Comment 2:</b> Use the <i>conditional operator</i> to determine if who is the eldest and to assign a value to the eldest variable.</p>
</div>
<div data-bbox="203 375 517 386" data-label="Text">
<pre>
      var combined_age = husband_age + wife_age;
</pre>
</div>
<div data-bbox="148 396 674 411" data-label="Text">
<p><b>Comment 3:</b> Add the two ages together to determine their combined age.</p>
</div>
<div data-bbox="203 421 509 433" data-label="Text">
<pre>
      var husband_multiply = (husband_age * 2);
</pre>
</div>
<div data-bbox="148 443 480 458" data-label="Text">
<p><b>Comment 4:</b> Multiply the husbands' age by 2.</p>
</div>
<div data-bbox="203 468 450 479" data-label="Text">
<pre>
      var wife_divide = (wife_age / 2);
</pre>
</div>
<div data-bbox="148 489 440 504" data-label="Text">
<p><b>Comment 5:</b> Divide the wife's age by 2.</p>
</div>
<div data-bbox="203 514 517 525" data-label="Text">
<pre>
      var husband_remainder = (husband_age % 3);
</pre>
</div>
<div data-bbox="148 535 850 565" data-label="Text">
<p><b>Comment 6:</b> Use the modules operand to determine what the remainder will be when the husband's age is divided by 3.</p>
</div>
<div data-bbox="148 574 849 672" data-label="Text">
<pre>
      document.write("The husband is " + husband_age + " years old.&lt;br /&gt;");
      document.write("The wife is " + wife_age + " years old.&lt;br /&gt;");
      document.write(eldest + "&lt;br /&gt;");
      document.write("Combined they are " + combined_age + " years old.&lt;br /&gt;");
      document.write("The husbands age multiplied by 2 is " + husband_multiply + " years.&lt;br /&gt;");
      document.write("The wifes age divided by 2 is " + wife_divide + " years.&lt;br /&gt;");
      document.write("After dividing the husbands age by 3 there are " + husband_remainder +
" years left.&lt;br /&gt;");
</pre>
</div>
<div data-bbox="148 683 487 697" data-label="Text">
<p><b>Comment 4:</b> Display the output on the screen.</p>
</div>
<div data-bbox="148 708 773 868" data-label="Text">
<pre>
    }
    /* ]]&gt; */
  &lt;/script&gt;
&lt;/head&gt;

&lt;body&gt;
&lt;h1&gt;User Age&lt;/h1&gt;
  &lt;form name = "age" action = ""&gt;
    Enter the husband's age: &lt;input type = "text" name = "husbandAge" /&gt;&lt;br /&gt;
    Enter the wife's age: &lt;input type = "text" name = "wifeAge" /&gt;&lt;br /&gt;
    &lt;p&gt;&lt;input type = "button" value = "Display" onclick = "display()" /&gt;&lt;/p&gt;
  &lt;/form&gt;
&lt;/body&gt;

&lt;/html&gt;
</pre>
</div>
<div data-bbox="148 882 556 898" data-label="Text">
<p><b>Step 2:</b> Save the code you entered as AgeOperators.html.</p>
</div>
<div data-bbox="849 886 886 936" data-label="Page-Footer">
<p>108</p>
</div>
```

**Step 3:** Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.

**Step 4:** Once the code has been validated open the browser and view your code.

Age Calculation - Windows Internet Explorer

F:\2009\DoE\AgeOperators.html

File Edit View Favorites Tools Help

Age Calculation

# User Age

Enter the husband's age: 47

Enter the wife's age: 37

Display

F:\2009\DoE\AgeOperators.html - Windows Internet Explorer

F:\2009\DoE\AgeOperators.html

File Edit View Favorites Tools Help

F:\2009\DoE\AgeOperators.html

The husband is 47 years old.  
The wife is 37 years old.  
The husband is older than the wife  
Combined they are 84 years old.  
The husbands age multiplied by 2 is 94 years.  
The wifes age divided by 2 is 18.5 years.  
After dividing the husbands age by 3 there are 2 years left.

## String Operators

A string is text that is contained in single ( ' ') or double quotes ( " " ), for example:

```
eldest = "The husband is older than the wife" :
```



Operator Name	Operator	Description
<b>Concatenation</b>	+	Combines two strings var name = "Piet"; var surname = "Poggempoel"; name = name + surname;
<b>Compound assignment</b>	+=	Combines two strings var city = "Johannesburg" city += " is in Gauteng".

## Escape Characters and Sequences

Some times a word must have an apostrophe, for example "it's" instead of "it is". Putting the apostrophe in the word will confuse the interpreter as it will see it as a single quote. To ensure the apostrophe is seen for what it is you need to add an escape character, which is the \ (backslash) for JavaScript. When the escape character is combined with another character it is called an escape sequence.

Escape Sequence	Character that is printed
\\	\ - Backslash
\"	" - Double quotation mark
\'	' - Single quotation mark
\0	Null character
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\XXX	Latin-1 character specified by the XX characters, which represents two hexadecimal digits
\XXXXX	Unicode character specified by the XXXXX characters, which represents four hexadecimal digits

### Step-by-Step create the JavaScript Program

*Develop a simple Web page in that prints the following information using the escape sequences indicated for each line.*

*Double and single quotation – My name is "Donald Mc'Ronald".*

*Backslash – I save my files on c:\My Documents*

*New line – The numbers 167 represents the unicode character |167*

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Escape Sequence</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
</head>
<body>
<h1>This page demonstrates different escape sequences</h1><hr />
<script type="text/javascript">
/*  */
    document.write("&lt;p&gt;My name is \"Donald Mc'Ronald\".&lt;/p&gt;");</pre>
</div>
<div data-bbox="848 886 886 936" data-label="Page-Footer">
<p>110</p>
</div>
```

**Comment 1:** Use the escape sequence \" and \' to print the double and single quotes.

```
document.write("<p>I save my files on c:\\My Documents</p>");
```

**Comment 2:** Use the escape sequence \\ to print the backslash.

```
document.write("<p>The numbers 167 represents the unicode character \\167</p>.</p>");
```

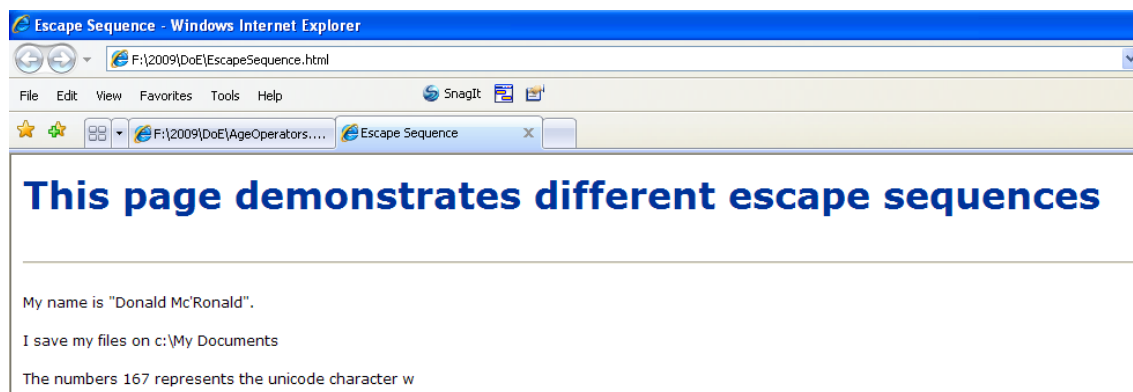
**Comment 3:** Use the escape sequence \\167 to print the Unicode character "w".

```
/* ]]> */  
</script>  
</body>
```

**Step 2:** Save the code you entered as EscapeSequence.html.

**Step 3:** Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.

**Step 4:** Once the code has been validated open the browser and view your code.



## Special Operators

The table below contains special operators that is available in JavaScript but that does not fit anywhere else. Some of these operators have already been used in the programs you have done thus far.

Name	Operator	Description
Property access	.	Appends an object, method, or property to another object
Array index	[]	Accesses an element of an array
Function call	()	Calls a functions or changes the order in which individual operations in an expression is executed
Comma	,	Allows multiple expressions in the same statement
Delete	delete	Deletes array elements, variables created without the var keyword, and properties of custom objects
Property exists	in	Retuns value of true if a specified property is contained without an object
Object type	instanceof	Returns true if an object is of a specified object type
New object	new	Creates a new instance of a user-defined object type or a predefined JavaScript object type
Data type	typeof	Determines data type of a variable
Void	void	Evaluates an expression without returning a result

## Operator Precedence

The operator precedence is the order in which the operations are executed in an expression. Calculations are not all done simultaneously; they are done one after the other in a specific mathematical order. Operators with a higher precedence are executed before those with a lower precedence

Certain operators are of equal precedence are executed associatively, that is they are evaluated from left-to-right or from right-to-left, depending on the operator.

Operators	Description	Precedence Level	Associativity
.	Objects	1	Left to right
[]	Array elements	1	Left to right
()	Functions	1	Left to right
new	New object	1	Right to left
!	Not	1	Right to left
-	Unary negation	1	Right to left
++	Increment	1	Right to left
--	Decrement	1	Right to left
typeof	Data type	1	Right to left
void	Void	1	Right to left
delete	Delete object	1	Right to left
* / %	Multiply / division / modulus	2	Left to right
+ -	Addition / subtraction / concatenation	3	Left to right
< <= > >=	Comparison	4	Left to right
instanceof	Object type	5	Left to right
in	Object property	5	Left to right
== != === !==	Equality	6	Left to right
&&	Logical and	7	Left to right
	Logical or	7	Left to right
?:	Conditional	8	Right to left
= += -= *= /= %=	Compound assignment	8	Right to left
,	Comma	9	Left to right

### Example

$A = 4 + 6 / 2 * 5$	$A = (4 + 6) / 2 * 5$	$A = (4 + 6) / (2 * 5)$
$A = 4 + 3 * 5$	$A = 10 / 2 * 5$	$A = 10 / 10$
$A = 4 + 15$	$A = 5 * 5$	$A = 1$
$A = 19$	$A = 25$	

### Step-by-Step create the JavaScript Program

*Develop a simple Web page that requests the user to enter temperature in Fahrenheit. The program must then convert the temperature into Celsius. Remember that Celsius = Fahrenheit – 32 \* 5 / 9.*

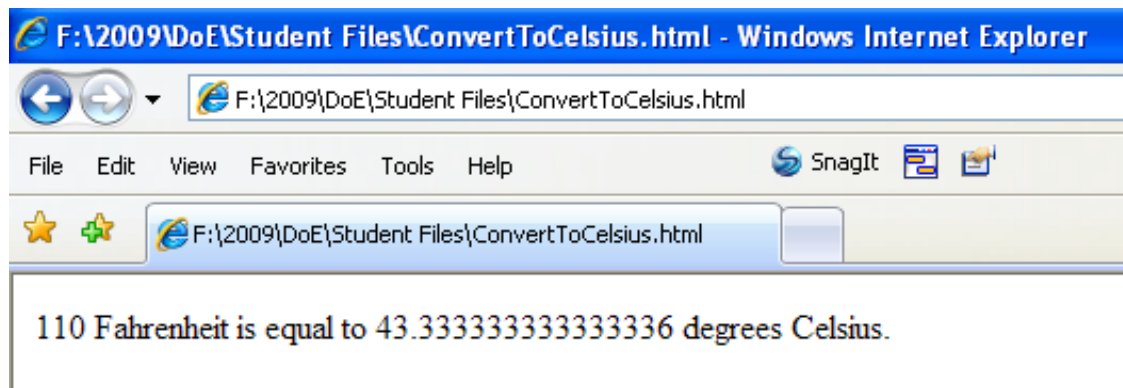
**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Convert to Celcius</title>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
<script type="text/javascript">
```

```

/*  */
function convert() {
    var fTemp = parseInt(celcius.f_temp.value);
</pre>
</div>
<div data-bbox="147 128 743 144" data-label="Text">
<p><b>Comment 1:</b> Use the <i>parseInt()</i> function to ensure the value is converted to integer.</p>
</div>
<div data-bbox="202 152 466 165" data-label="Text">
<pre>
    var cTemp = (fTemp - 32) * (5 / 9);
</pre>
</div>
<div data-bbox="147 175 850 205" data-label="Text">
<p><b>Comment 2:</b> The Fahrenheit degrees are converted into Celsius degrees, the calculations inside the brackets will be done first and only then will the division be done.</p>
</div>
<div data-bbox="147 214 849 375" data-label="Text">
<pre>
    document.write("&lt;p&gt;" + fTemp + " Fahrenheit is equal to " + cTemp + " degrees
Celsius.&lt;/p&gt;");
}
/* ]]&gt; */
&lt;/script&gt;

&lt;/head&gt;
&lt;body&gt;
&lt;h1&gt;Convert Fahrenheit to Celcius&lt;/h1&gt;
&lt;form name = "celcius" action = ""&gt;
    Enter the Fahrenheit degrees: &lt;input type = "text" name = "f_temp" /&gt;&lt;br /&gt;
    &lt;p&gt;&lt;input type = "button" value = "Display" onclick = "convert()" /&gt;&lt;/p&gt;
&lt;/form&gt;
&lt;/body&gt;
&lt;/html&gt;
</pre>
</div>
<div data-bbox="147 386 580 401" data-label="Text">
<p><b>Step 2:</b> Save the code you entered as ConvertToCelsius.html.</p>
</div>
<div data-bbox="147 413 813 443" data-label="Text">
<p><b>Step 3:</b> Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.</p>
</div>
<div data-bbox="147 456 713 473" data-label="Text">
<p><b>Step 4:</b> Once the code has been validated open the browser and view your code.</p>
</div>
<div data-bbox="151 483 835 784" data-label="Image">
<img alt="Screenshot of a Windows Internet Explorer browser window displaying a web page titled 'Convert to Celcius'. The address bar shows the file path 'F:\2009\DoE\Student Files\ConvertToCelsius.html'. The page content includes a heading 'Convert Fahrenheit to Celcius', a text input field with the value '110', and a 'Display' button."/>
<p>The screenshot shows a Windows Internet Explorer browser window. The title bar reads 'Convert to Celcius - Windows Internet Explorer'. The address bar shows the file path 'F:\2009\DoE\Student Files\ConvertToCelsius.html'. The menu bar includes 'File', 'Edit', 'View', 'Favorites', 'Tools', and 'Help'. The toolbar shows a star icon, a plus icon, and a search icon. The main content area displays a heading 'Convert Fahrenheit to Celcius' in a large, bold, black serif font. Below the heading is a text input field with the value '110' and a 'Display' button.</p>
</div>
<div data-bbox="848 885 889 935" data-label="Page-Footer">
<p>113</p>
</div>
```



### Writing mathematical equations the computers way

Unfortunately a computer cannot understand mathematical equations when they are written in the scientific way. We therefore need to "translate" them into a format the computer can understand. Mathematical equations must be written in a single line for the computer to apply the order of precedence to them and to calculate the result.

#### Example

$$X = A^3 + 2B + \frac{2(C + D)}{2}$$

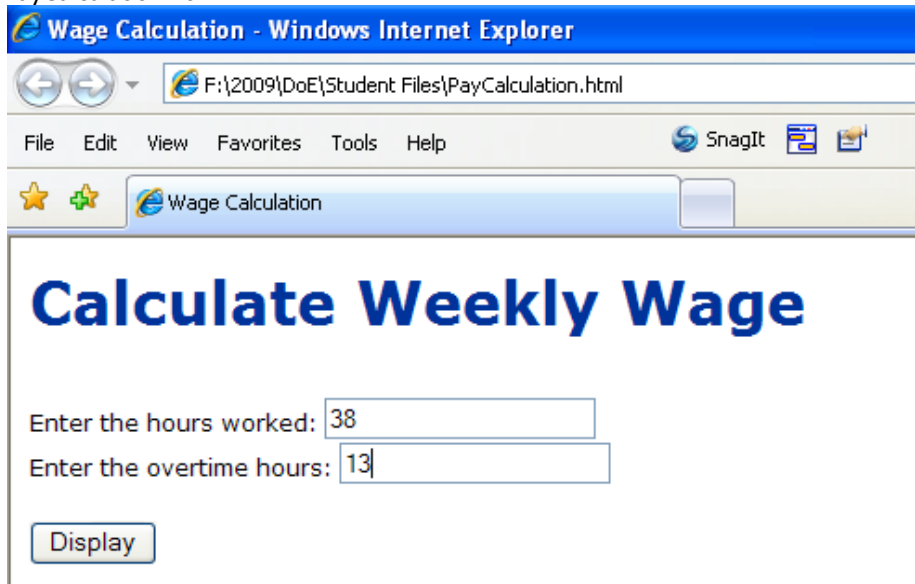
becomes  $X = A^3 - 2 * B(2(C + D)) / 2$

$$17BC - B^{(E - A)}$$

becomes  $- 17 * B * C - B^{(E - A)}$

## Exercise

Themba works as a junior security office at his local supermarket. He is paid R45 per hour for a normal shift of 8 hours a day, days per week. He is allowed to work 1 hour per day over time as well as 8 hours on a Saturday or Sunday. Overtime pay is time and a half of his normal hourly rate. 10% of Themba's pay is deducted for taxes and R15 per week is deducted for the unemployment fund. Develop a simple website that will calculate how much Themba earns weekly. Save the code as PayCalculation.html.



The screenshot shows a Windows Internet Explorer window titled "Wage Calculation - Windows Internet Explorer". The address bar shows the file path "F:\2009\DoE\Student Files\PayCalculation.html". The menu bar includes File, Edit, View, Favorites, Tools, and Help. The toolbar shows the SnagIt icon. The page content features a large blue heading "Calculate Weekly Wage". Below the heading, there are two input fields: "Enter the hours worked:" with the value "38" and "Enter the overtime hours:" with the value "13". A "Display" button is located below the input fields.

Wage Calculation - Windows Internet Explorer

F:\2009\DoE\Student Files\PayCalculation.html

File Edit View Favorites Tools Help

SnagIt

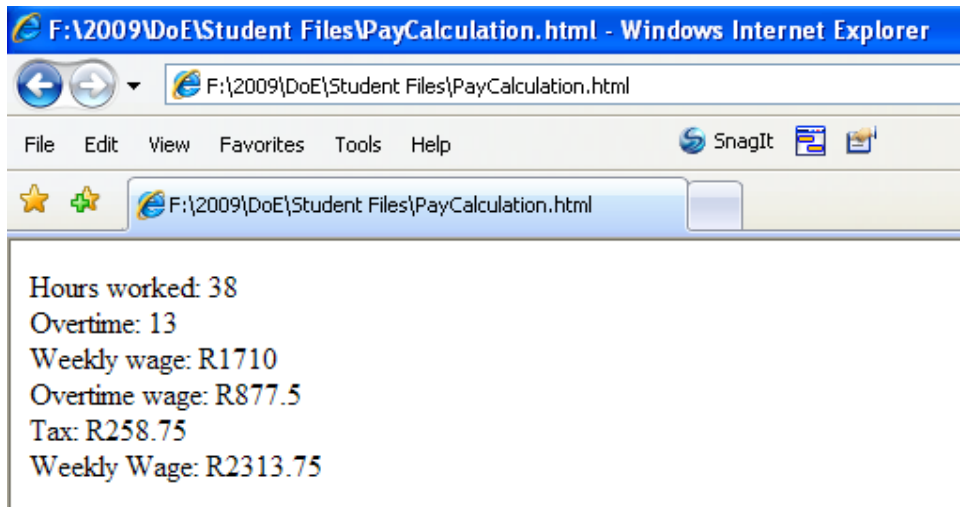
Wage Calculation

## Calculate Weekly Wage

Enter the hours worked: 38

Enter the overtime hours: 13

Display



The screenshot shows the same Windows Internet Explorer window, but now displaying the calculated results. The address bar and menu bar are the same. The page content shows the following results:

Hours worked: 38  
Overtime: 13  
Weekly wage: R1710  
Overtime wage: R877.5  
Tax: R258.75  
Weekly Wage: R2313.75

F:\2009\DoE\Student Files\PayCalculation.html - Windows Internet Explorer

F:\2009\DoE\Student Files\PayCalculation.html

File Edit View Favorites Tools Help

SnagIt

F:\2009\DoE\Student Files\PayCalculation.html

Hours worked: 38  
Overtime: 13  
Weekly wage: R1710  
Overtime wage: R877.5  
Tax: R258.75  
Weekly Wage: R2313.75

# Functions

## Introduction

A function is a group of statements that can be executed as a single unit. Most of the programs you have entered so far make use of a function to enable interactivity between the user and the Web page.

## Built in Functions

JavaScript has numerous built in functions that can assist you in various ways. One of the built in functions we have been using thus far has been the *parseInt()* function which converts a string entered by the user into an integer.

Three more such built in functions that are used frequently give you a way to request or display information from or to the user using a dialog box. A dialog box is a small window that opens in the main window.

### Alert dialog box

The alert function displays information in a dialog box and is used if you do not want to update the web page itself. The alert dialog box is used most often to display error messages. This dialog box displays an OK button which needs to be clicked in order for the JavaScript to continue executing

### Step-by-Step create the JavaScript Program

*Design a simple website that prompts the user to enter 4 integers. Using the information entered by the user determine the sum and the average of the 4 numbers. Display the sum and the average of the 4 integers in a dialog box.*

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Number Calculations</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

  <script type="text/javascript">
    /*  */

    function calculate() {</pre></div><div data-bbox="148 725 552 740" data-label="Text"><p><b>Comment 1:</b> Definition of the function named calculate()</p></div><div data-bbox="148 750 732 772" data-label="Text"><pre>      var sum = parseInt(calc.number1.value) + parseInt(calc.number2.value) +
      parseInt(calc.number3.value) + parseInt(calc.number4.value);</pre></div><div data-bbox="148 782 649 797" data-label="Text"><p><b>Comment 2:</b> parseInt() function ensures value is converted to integer.</p></div><div data-bbox="202 810 369 821" data-label="Text"><pre>      var average = sum / 4;</pre></div><div data-bbox="148 832 666 847" data-label="Text"><p><b>Comment 3:</b> Calculate the average of the four numbers the user entered.</p></div><div data-bbox="202 859 517 871" data-label="Text"><pre>      alert("Sum of the four numbers = " + sum);</pre></div><div data-bbox="148 882 534 897" data-label="Text"><p><b>Comment 4:</b> Display the output in an alert dialog box</p></div><div data-bbox="849 887 886 936" data-label="Page-Footer"><p>116</p></div>
```

```
    alert("Average of the four numbers = " + average);
```

**Comment 5:** Display the output in an alert dialog box

```
}  
  
/* ]]> */  
</script>  
</head>  
  
<body>  
    <form name = "calc" action = "">
```

**Comment 6:** Create the form the user must use to enter the data

```
    Enter number 1: <input type = "text" name = "number1" /><br />  
    Enter number 2: <input type = "text" name = "number2" /><br />  
    Enter number 3: <input type = "text" name = "number3" /><br />  
    Enter number 4: <input type = "text" name = "number4" />
```

**Comment 7:** Create the fields into which the user enter the numbers

```
    <p><input type = "button" value = "Calculate" onclick = "calculate()" /></p>
```

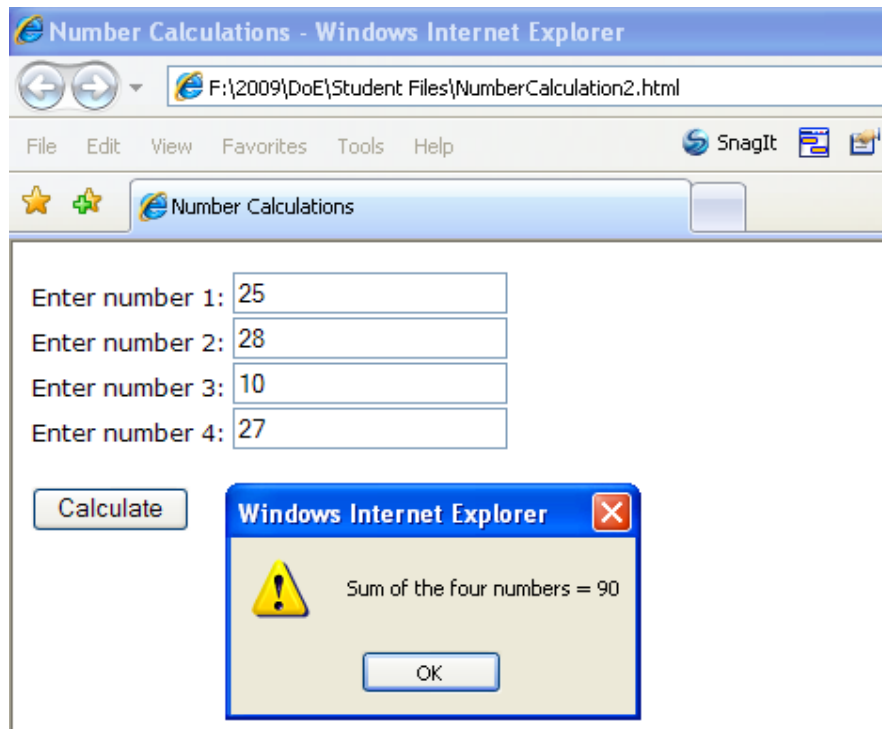
**Comment 8:** Create the button which calls the calculate function in the script

```
    </form>  
</body>  
  
</html>
```

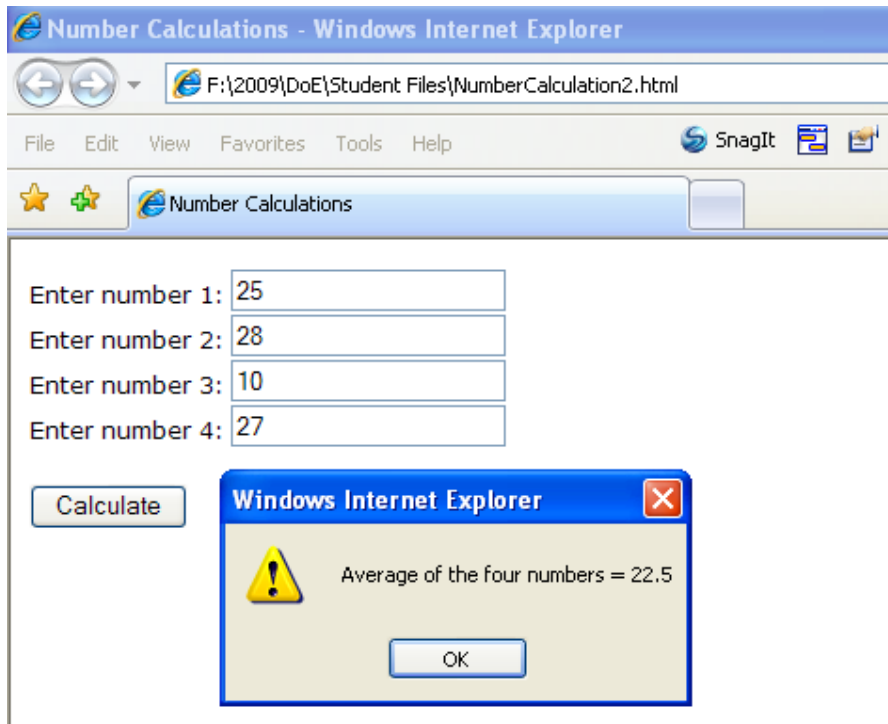
**Step 2:** Save the code you entered as NumberCalculation2.html.

**Step 3:** Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.

**Step 4:** Once the code has been validated open the browser and view your code.







### Confirm dialog box

The confirm dialog box displays a message and the function returns a value of either true or false, depending on the choice of the user. Two buttons are displayed at the bottom of the dialog box, the OK button returns a value of true and the Cancel button returns a value of false.

### Step-by-Step create the JavaScript Program

*Develop a simple Web page in which you request the user to enter his or her age. The program should then calculate what the user's age was 10 year ago as well as what age the user would be in another 10 years. Add a confirm dialog box that asks the users if they are sure they want to continue, if the user selects OK display the calculations in an alert dialog box, if the user selects cancel show the message "Chicken".*

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Age Calculation</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

  <script type="text/javascript">
    /*  */

    function display() {
      var current_age = parseInt(age.currentAge.value);
      var younger_age = current_age - 10;
      var older_age = current_age + 10;</pre>
</div>
<div data-bbox="147 851 743 867" data-label="Text">
<p><b>Comment 1:</b> Use the <i>parseInt()</i> function to ensure the value is converted to integer.</p>
</div>
<div data-bbox="202 875 583 888" data-label="Text">
<pre>    if (confirm("Are you sure you want to do this?")) {</pre>
</div>
<div data-bbox="849 886 886 936" data-label="Page-Footer">
<p>118</p>
</div>
```

**Comment 2:** Add the confirm dialog box function and ask the user if they are sure they want to continue. If the user clicks on the OK button the following code will execute:

```
        alert("Younger age: " + younger_age + "    Current age: " + current_age + "
Older age: " + older_age);
```

**Comment 3:** The results of the calculations made are shown in an alert dialog box.

```
    }
    else
        alert("Chicken!");
```

**Comment 4:** If the user clicks on the Cancel button the message above is displayed.

```
    }
    /* ]]> */
</script>
</head>

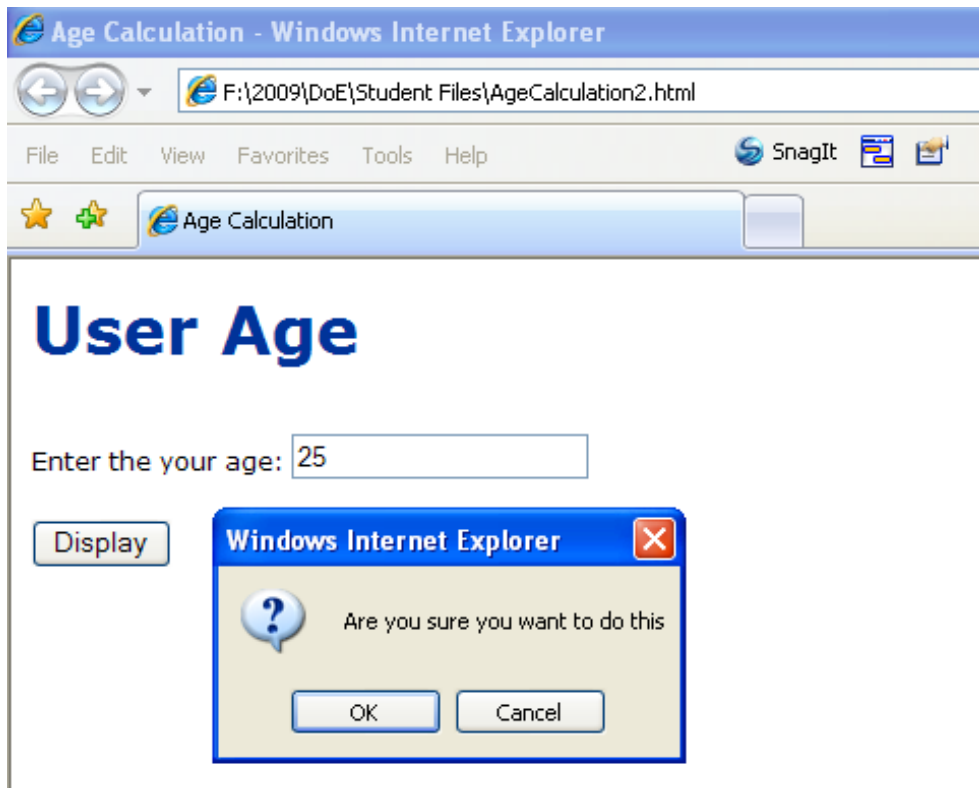
<body>
<h1>User Age</h1>
    <form name = "age" action = ">
        Enter the your age: <input type = "text" name = "currentAge" /><br />
        <p><input type = "button" value = "Display" onclick = "display()" /></p>
    </form>
</body>

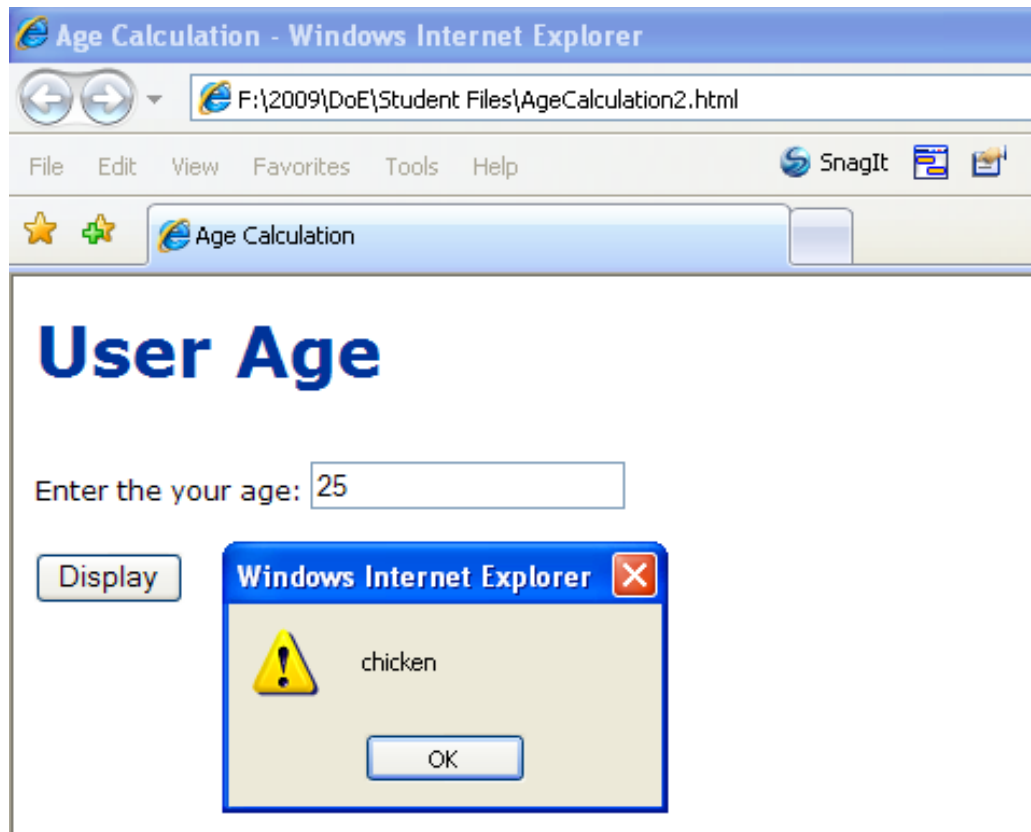
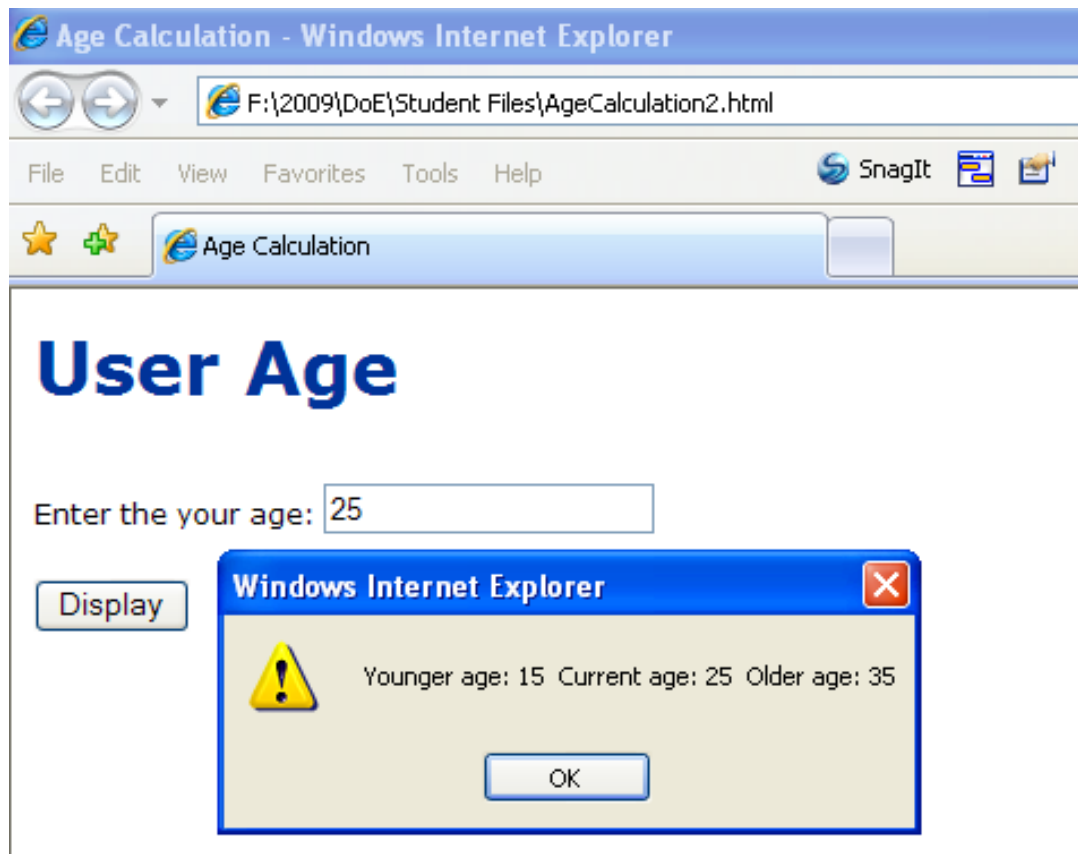
</html>
```

**Step 2:** Save the code you entered as AgeCalculation2.html.

**Step 3:** Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.

**Step 4:** Once the code has been validated open the browser and view your code.





## Prompt dialog box

The prompt dialog box can be used to get information from the user, thus the user must enter some sort of information into the text box provided by the dialog box. This information is returned as a string when the user clicks the OK button on the dialog box, the value *null* is returned when the user click the Cancel button.

## Step-by-Step create the JavaScript Program

*Develop a simple Web page in which you request the user to enter his or her age into a prompt dialog box. The program should then calculate what the user's age was 10 year ago as well as what age the use would be in another 10 years.*

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

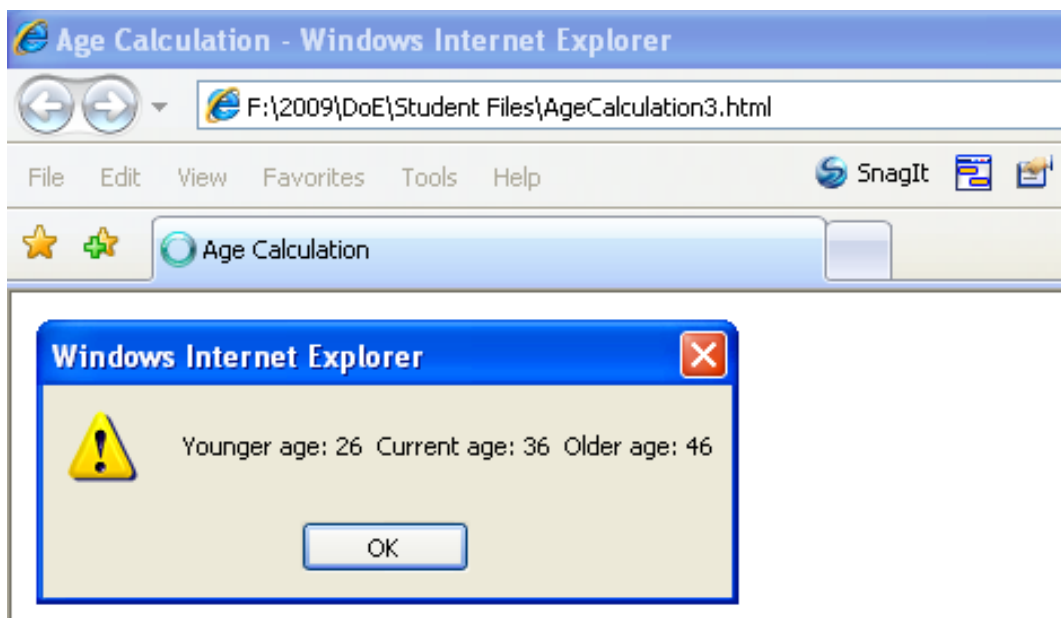
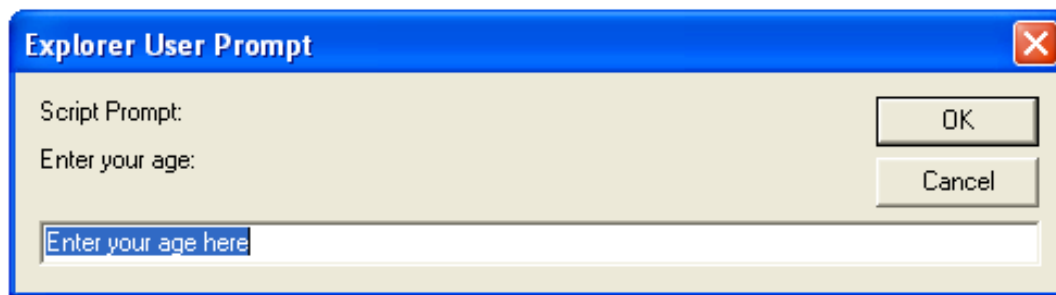
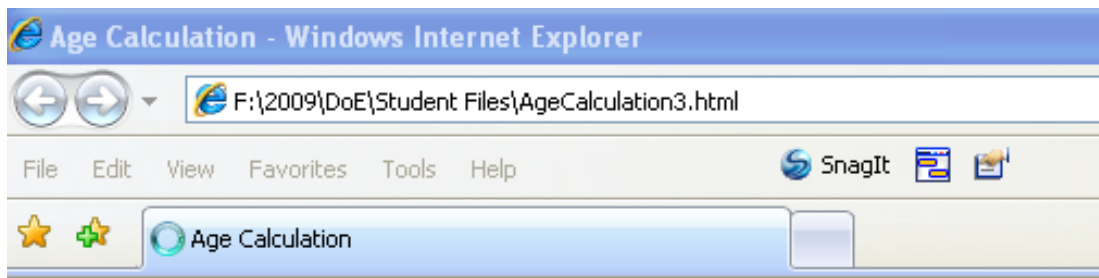
<head>
  <title>Age Calculation</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

  <script type="text/javascript">
    /*  */

      var currentAge = prompt("Enter your age:", "Enter your age here");</pre></div><div data-bbox="147 449 850 480" data-label="Text"><p><b>Comment 1:</b> Declare the variable to accept the users age, add the prompt dialog box and ask the user to enter their age.</p></div><div data-bbox="202 488 494 501" data-label="Text"><pre>      var current_age = parseInt(currentAge);</pre></div><div data-bbox="147 510 743 526" data-label="Text"><p><b>Comment 2:</b> Use the <i>parseInt()</i> function to ensure the value is converted to integer.</p></div><div data-bbox="202 535 466 558" data-label="Text"><pre>      var younger_age = current_age - 10;
      var older_age = current_age + 10;</pre></div><div data-bbox="147 567 553 583" data-label="Text"><p><b>Comment 3:</b> Calculate the user younger and older ages.</p></div><div data-bbox="147 592 847 615" data-label="Text"><pre>      alert("Younger age: " + younger_age + " Current age: " + current_age + " Older age: " + older_age);</pre></div><div data-bbox="147 625 734 640" data-label="Text"><p><b>Comment 4:</b> The results of the calculations made are shown in an alert dialog box.</p></div><div data-bbox="147 653 220 740" data-label="Text"><pre>    /* ]]&gt; */
  &lt;/script&gt;
&lt;/head&gt;

&lt;body&gt;
&lt;/body&gt;

&lt;/html&gt;</pre></div><div data-bbox="147 753 570 769" data-label="Text"><p><b>Step 2:</b> Save the code you entered as AgeCalculation3.html.</p></div><div data-bbox="147 781 812 811" data-label="Text"><p><b>Step 3:</b> Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.</p></div><div data-bbox="147 824 713 841" data-label="Text"><p><b>Step 4:</b> Once the code has been validated open the browser and view your code.</p></div><div data-bbox="849 886 886 936" data-label="Page-Footer"><p>121</p></div>
```



Below is a short list of other built in JavaScript functions that are used often.

Build-in JavaScript Function	Description
<b>decodeURI(string)</b>	Decodes text strings encoded with encodeURIComponent()
<b>decodeURIComponent(string)</b>	Decodes text strings encoded with encodeURIComponent()
<b>encodeURIComponent(string)</b>	Encodes a text string so that it becomes a valid URI
<b>encodeURIComponent(string)</b>	Encodes a text string so it becomes a valid URI component
<b>eval(string)</b>	Evaluates expressions contained within strings
<b>isFinite(number)</b>	Determines whether a number is finite
<b>isNaN(number)</b>	Determines whether a value is Not a Number
<b>parseFloat(string)</b>	Converts string literals to floating-point numbers
<b>parseInt(string)</b>	Converts string literals to integers

## User defined Functions

Recall from the first JavaScript chapter that an object is programming code and data that can be treated as an individual unit or component. Methods are procedures associated with an object and properties are piece of data associated with an object. Procedures are individual statements used in a computer program grouped into logical units, used to perform specific tasks. When programmers write their own procedures they are called functions.

Writing functions saves a lot of time as functions can be re-used, for example, writing a function that checks a person ID number for validity can be used on several different programs.

The code inside a function will only be executed if it is called from a statement within the program. The function can be placed in either the *<head>* or *<body>* section of the program, but to make sure the code is read by the browser before it is called it should be placed in the *<head>* section.

### Defining functions

The function definition looks like the following:

```
function nameOfFunction (parameters) {  
    statements;  
}
```

Declaring the function starts with the key word *function* in lower case. Key words are reserved word which, if used out of context, will cause compilation errors when the script is first loaded by the browser.

Function names may only consist of alphanumeric characters (alphabet and digits) and the \_ (underscore). The name may not begin with a number or a special character. Function names are case sensitive, calculate\_age and Calculate\_age will be two different names.

#### Some valid function names

display\_user\_name  
calculate  
\_calculate\_age  
average1

#### Some invalid function names

@\_user\_email      (invalid character)  
123\_user            (function name cannot begin with a numeral)

Parameters are variables used within a function. Adding the parameter in the brackets next to the function name is the same as declaring a new variable, more than one variable can be declared in this way. The parameter declared in this manner receives its values when the function is called from somewhere else in the program. Parameters are used if the values of variables will be passed from one function to another, if the values of the variables will only be used in a specific function you do not need to include parameters. If no parameters is included the statement must still included the () parenthesis after the function name, for example:

```
function nameOfFunction () {  
    statements;  
}
```

The statements that perform the actual work in the function is contained in a pair of opening and closing curly braces { } called function braces.

## Example

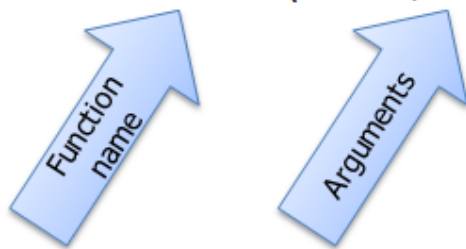
```
function printCourseCodes (code1, code2, code3) {  
    document.write("<p>" + code1 + "</p>");  
    document.write("<p>" + code2 + "</p>");  
    document.write("<p>" + code3 + "</p>");  
}
```

Take note of the structure of the function, although the layout is not a strict requirement it is recommended that this layout always be followed. It makes the reading of the code that much easier.

## Calling Functions

Function definitions do not execute automatically, they need to be called from elsewhere in the program. The function call looks as follows:

**printCourseCode(code1, code2, code3);**



The arguments are variables or values which are passed to the parameters of the calling function.

## Return Statement

The return statement specifies the value that must be returned to the statement that called the function.

```
function printAverageOfNumbers(number1, number2, number) {  
    var sumOfNumbers = number1 + number2 + number3;  
    var result = sumOfNumbers / 3;  
    return result;  
}
```

## Step-by-Step develop the JavaScript program

Develop a simple Web page which calls a function that displays a user name that is passed to the function as an argument. The second function should return the users surname from the function to be printed.

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
  
<head>  
    <title>Print User Name using Functions</title>  
    <link rel="stylesheet" href="styles.css" type="text/css" />  
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />  
  
<script type="text/javascript">  
/* <![CDATA[ */  
  
function printUserName(user_name) {  
    document.write("<p>User name: " + user_name + "</p>");  
}
```

**Comment 1:** The function writes a message on the screen using an argument which is passed from the calling statement

```
function return_name() {  
    return ("<p>User surname: Piper.</p>");  
}
```

**Comment 2:** The message is defined in the function, the purpose of the function is to return the string to the calling statement.

```
/* ]]> */  
</script>  
</head>  
  
<body>  
<h1>User Name</h1>  
<script type="text/javascript">  
/* <![CDATA[ */  
printUserName("Peter");
```

**Comment 3:** This statement calls the function printUserName

```
var return_value = return_name();
```

**Comment 4:** This statement assigns the function call to the variable return\_value, it does not send arguments to the function

```
document.write(return_value);
```

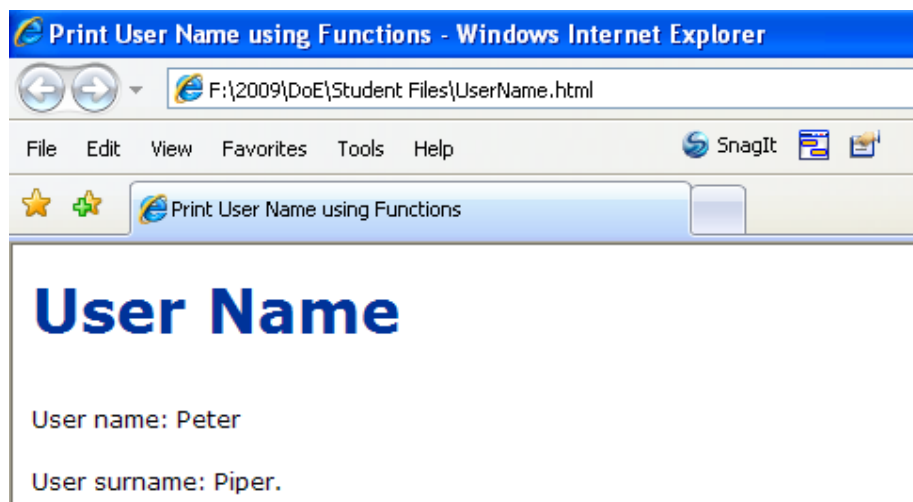
**Comment 5:** The statement writes the value of the variable, which was returned, to the screen

```
/* ]]> */  
</script>  
</body>  
  
</html>
```

**Step 2:** Save the code you entered as UserName.html.

**Step 3:** Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.

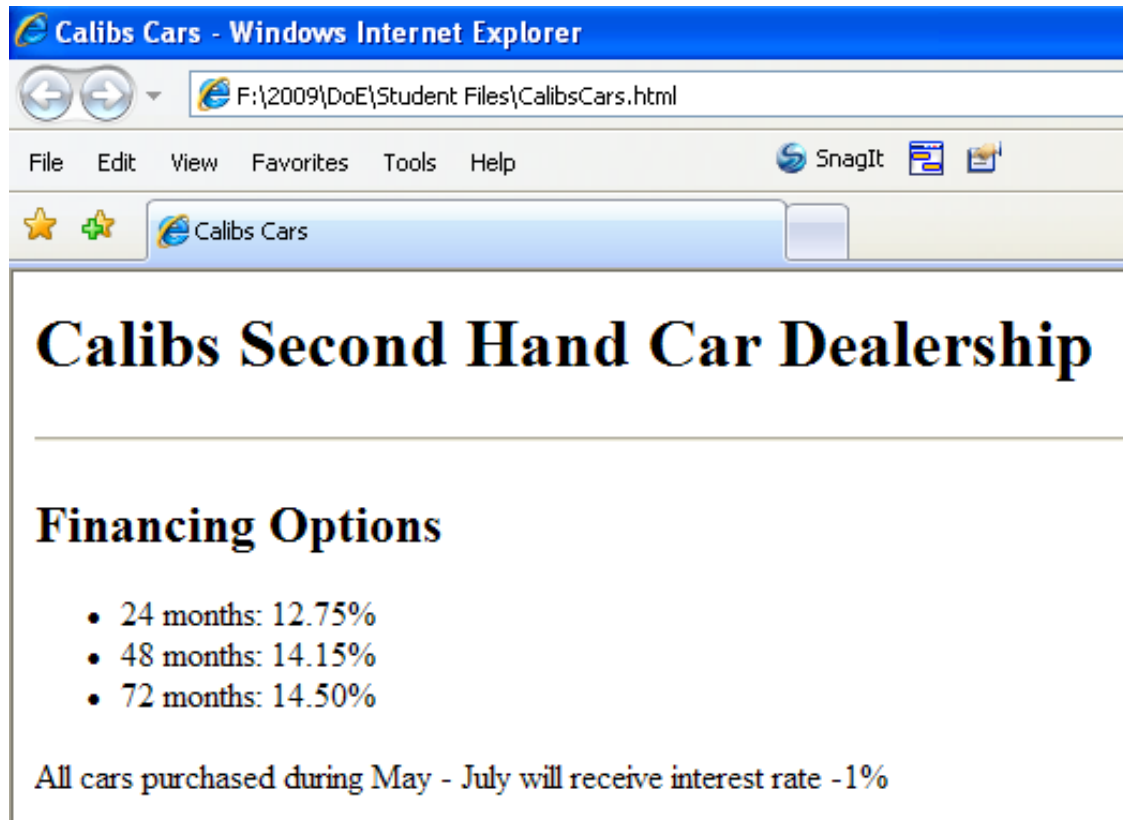
**Step 4:** Once the code has been validated open the browser and view your code.





### Exercise

Develop a simple website which uses a function to print financing options for Calibs Second Hand Car Dealership. A second function should print the special financing offer, all car sold during May – June will receive 1% off on their interest rate. Save the code as CalibsCars.html



## Events

### Introduction

An event is a specific action that happens which can be monitored by JavaScript. When such an event takes place the script can respond in a certain way. The events are most often a result of some action taken by the user.

### Events

Below is a short list of the most common JavaScript events, what they are triggered by and "whom" they are triggered by.

Event	Triggered	By
<b>abort</b>	The loading of an image is interrupted	Script
<b>blur</b>	An element, such as radio button, becomes inactive	Script
<b>click</b>	The user clicks an element once	User
<b>change</b>	The value of an element, such as text box, changes	User
<b>error</b>	An error occurs when a document or image is being loaded	Script
<b>focus</b>	An element, such as a command button, becomes active	Script
<b>load</b>	A document or image loads	Script
<b>mouseout</b>	The mouse moves off an element	User
<b>mouseover</b>	The mouse moves over an element	User
<b>reset</b>	A form's fields are reset to its default values	Script / user
<b>select</b>	A user selects a field in a form	User
<b>submit</b>	A user submits a form	User
<b>unload</b>	A document unloads	Script

### Event Handlers

Event handlers are specific code that is executed when an event occurs. The event handler is included as an attribute to the element that initiates the event. Recall that an element is XHTML data contained in a tag pair, for example `<button></button>`. The code for the event handler looks as follows:

```
<element even.handler = "JavaScript code">
```

Events and their event handlers have the same name; the difference is that the event handler has a prefix of "on". This can be seen as "on" the event happening the handler is initiated. Event handlers are case sensitive and can only be written in all lower case letters.

Listed below is a few of event handlers and their associated XHTML elements.

Event handler	Applies to:	Triggered when:
<b>onAbort</b>	Image	The loading of the image is cancelled.
<b>onBlur</b>	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question loses focus (e.g. by clicking outside it or pressing the <b>TAB</b> key).
<b>onChange</b>	FileUpload, Select, Text, TextArea	The data in the form element is changed by the user.
<b>onClick</b>	Button, Document, Checkbox, Link, Radio, Reset, Submit	The object is clicked on.

<b>onDbClick</b>	Document, Link	The object is double-clicked on.
<b>onDragDrop</b>	Window	An icon is dragged and dropped into the browser.
<b>onError</b>	Image, Window	A JavaScript error occurs.
<b>onFocus</b>	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question gains focus (e.g. by clicking on it or pressing the TAB key).
<b>onKeyDown</b>	Document, Image, Link, TextArea	The user presses a key.
<b>onKeyPress</b>	Document, Image, Link, TextArea	The user presses or holds down a key.
<b>onKeyUp</b>	Document, Image, Link, TextArea	The user releases a key.
<b>onLoad</b>	Image, Window	The whole page has finished loading.
<b>onMouseDown</b>	Button, Document, Link	The user presses a mouse button.
<b>onMouseMove</b>	None	The user moves the mouse.
<b>onMouseOut</b>	Image, Link	The user moves the mouse away from the object.
<b>onMouseOver</b>	Image, Link	The user moves the mouse over the object.
<b>onMouseUp</b>	Button, Document, Link	The user releases a mouse button.
<b>onMove</b>	Window	The user moves the browser window or frame.
<b>onReset</b>	Form	The user clicks the form's Reset button.
<b>onResize</b>	Window	The user resizes the browser window or frame.
<b>onSelect</b>	Text, Textarea	The user selects text within the field.
<b>onSubmit</b>	Form	The user clicks the form's Submit button.
<b>onUnload</b>	Window	The user leaves the page

## Some Common Event Handlers

### onClick Event

The onClick event handler is one of the most popular event handlers as it can be used from buttons to checkboxes and link. The onClick event executes when the user click with the mouse on an object.

### Step-by-step develop the JavaScript Program

Develop a simple Web page that displays three radio buttons. The background colour of the Web page should be yellow, when the user clicks on the radio button it should change to the colour next to the radio button.

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
    <title>Change Screen Color</title>

    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
</head>

<body bgcolor="yellow">
```

**Comment 1:** Assign the colour yellow to the background of the Web page

```
<form method="POST">
  <p><input type="radio" name="radio1" value="r1"
onclick="document.bgColor='lightblue'">Light Blue</p>
```

**Comment 2:** When the user clicks on the first radio button the background color of the Web page changes to light blue

```
  <p><input type="radio" name="radio1" value="r2"
onclick="document.bgColor='lightpink'">Light Pink</p>
```

**Comment 3:** When the user clicks on the second radio button the background color of the Web page changes to light pink

```
  <p><input type="radio" name="radio1" value="r3"
onclick="document.bgColor='lightgreen'">Light Green</p>
```

**Comment 4:** When the user clicks on the third radio button the background color of the Web page changes to light green

```
</form>
```

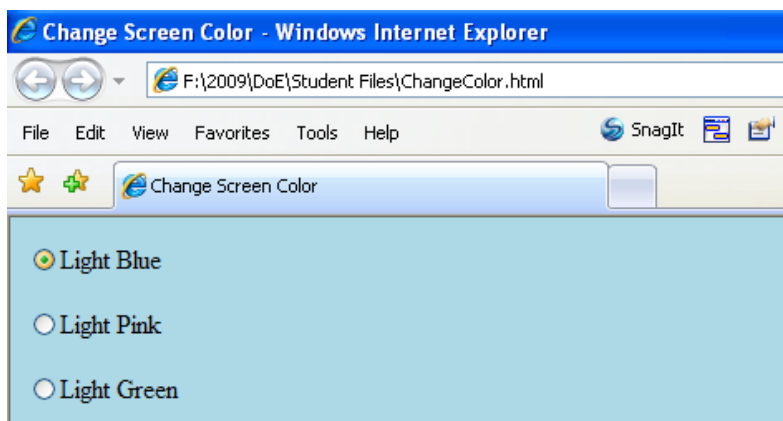
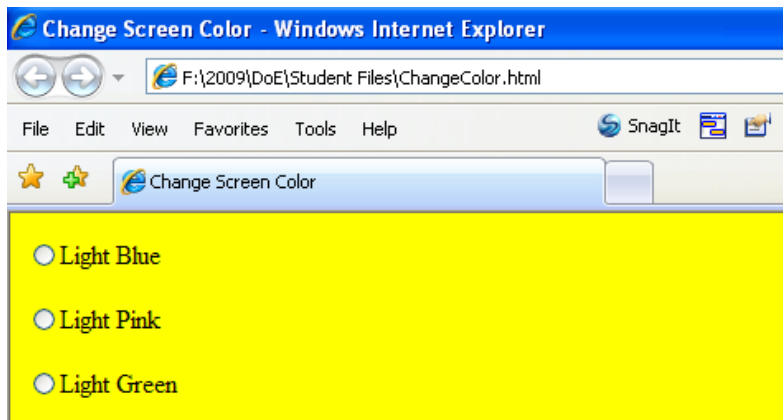
```
</body>
```

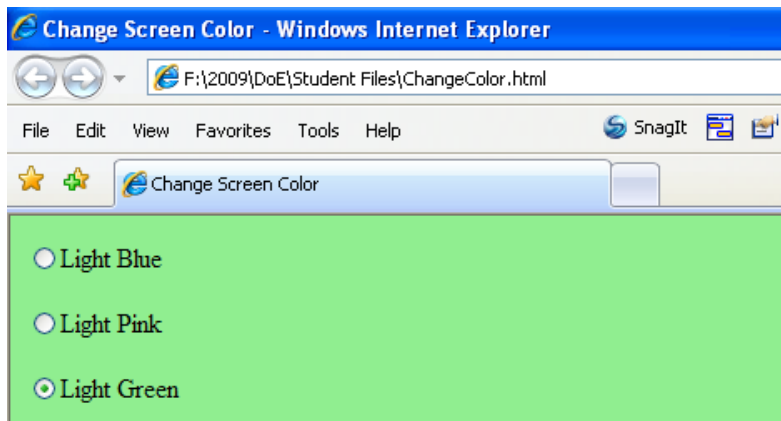
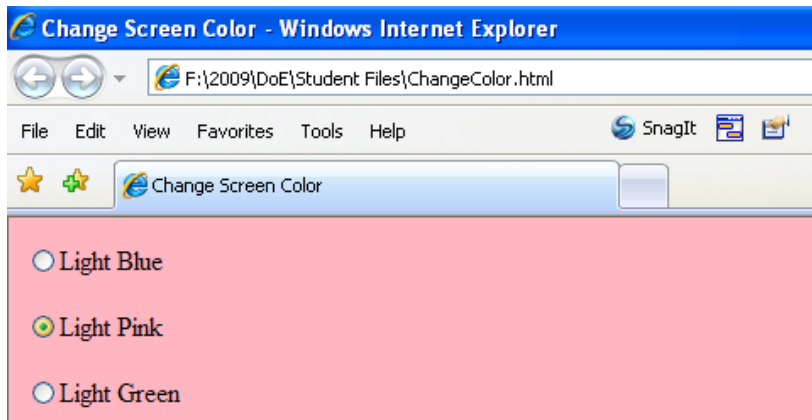
```
</html>
```

**Step 2:** Save the code you entered as changeColor.html.

**Step 3:** Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.

**Step 4:** Once the code has been validated open the browser and view your code.





### onMouseOut and onMouseOver

These are also very popular event handlers and are mostly used with images. When the user passes the mouse over or out of an image it may change the image.

### Step-by-step develop the JavaScript Program

Develop a simple Web page that displays a word, when the mouse is moved over the word the background colour of the Web page should change from yellow, to pink and when the mouse is moved away from the word the colour should change to blue.

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Change Screen Color</title>

  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
</head>

<body bgcolor="yellow">
```

**Comment 1:** Assign the colour yellow to the background of the Web page

```
<form method="POST">
```

```
<p><a href="" onmouseover="document.bgColor='lightpink'"
```

**Comment 2:** When the user moves the mouse across the words "Colour me Pink" the background colour changes to light pink

```
onmouseout="document.bgColor='lightblue'">
```

**Comment 3:** When the user moves the mouse away from the words "Colour me Pink" the background colour changes to light blue

```
Colour me Pink!</a></p>
```

```
</form>
```

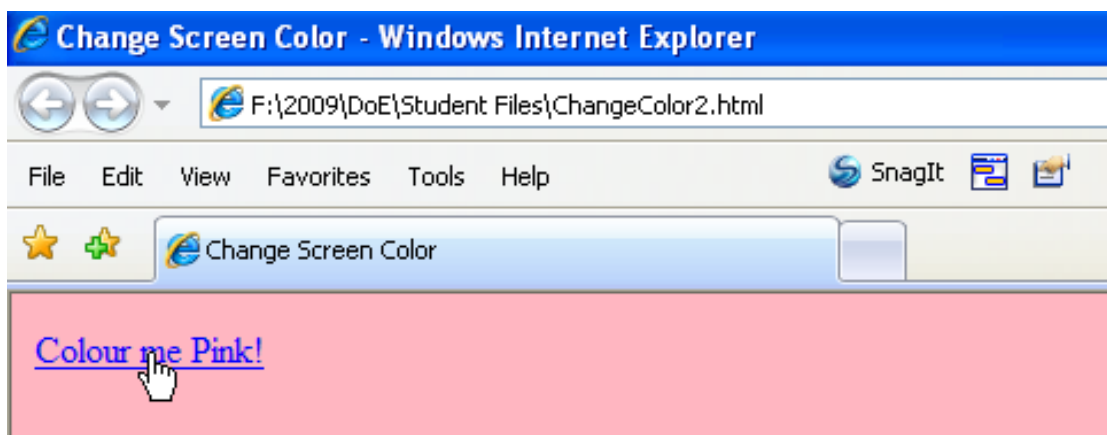
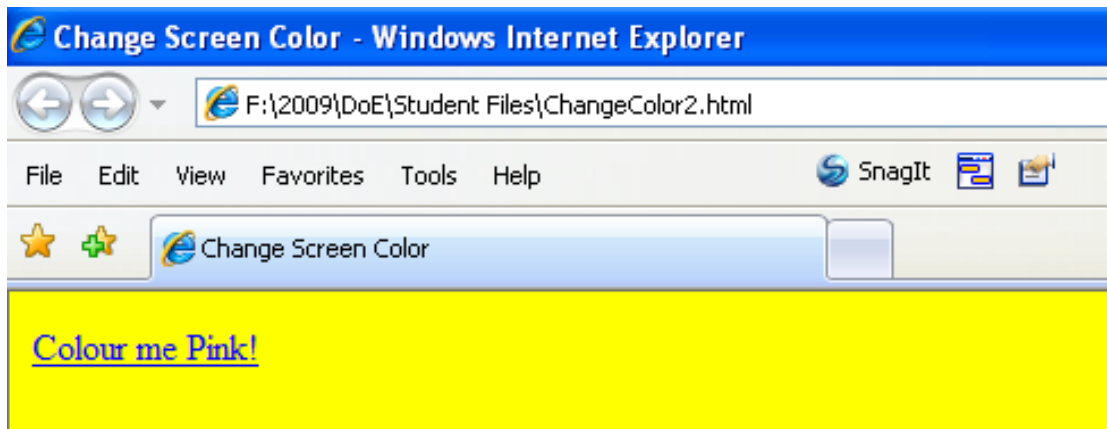
```
</body>
```

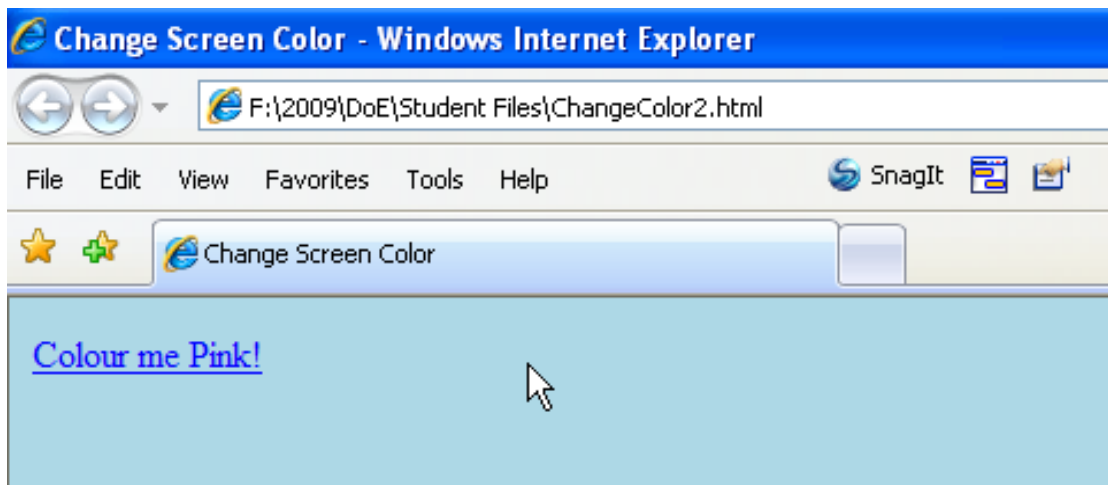
```
</html>
```

**Step 2:** Save the code you entered as changeColor2.html.

**Step 3:** Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.

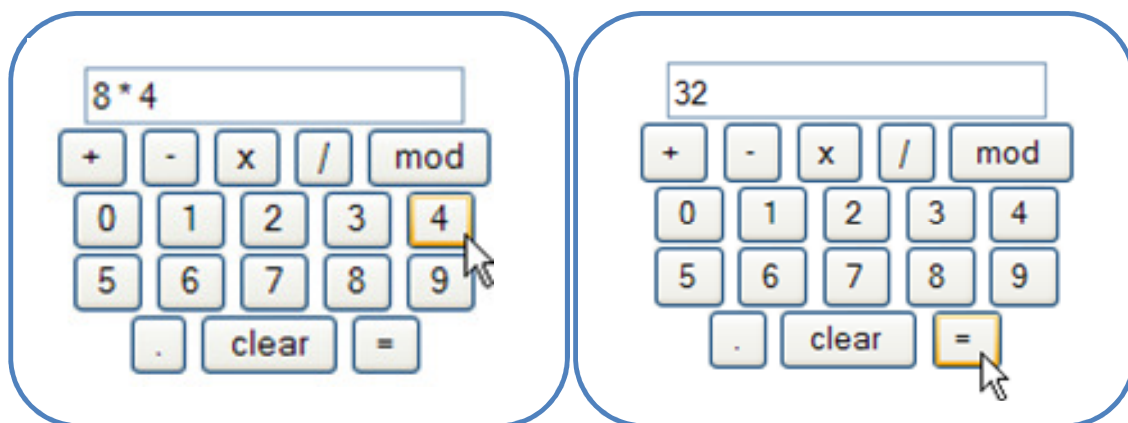
**Step 4:** Once the code has been validated open the browser and view your code.





### Exercise

Develop a simple Web page that displays a calculator. When the user clicks on the button the onclick event handler should be used to send the value of the button to function. The function should then do the calculation as requested by the user. Save the code as Calculator.html



## Control Structures

### Introduction

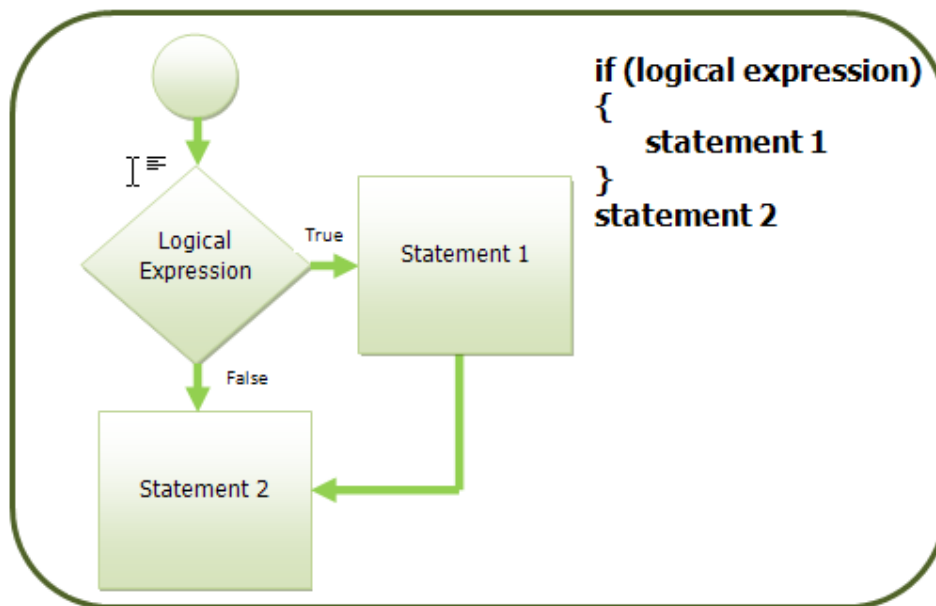
Almost every program you design and write will need some type of control structure. These statements give you the ability to direct the flow of execution through your program based on a condition.

### Logical IF Structure

This is the simplest form of the IF statement. The logical IF statement is used when an action needs to be taken for a single statement.

The logical expression is a Boolean variable, constant, or expression that evaluates to TRUE, FALSE, or NULL. If the logical expression is TRUE, the statement following the logical expression is executed. If the logical expression is FALSE or NULL, the statements are not executed.

In order to make the readability of the IF structure easier, the statement which should be performed in the IF structure is indented. Should there be more than one statement that needs to be executed the statements must be enclosed in curly brackets { }.



### Step-by-Step create the JavaScript Program

*Develop a simple Web page in which you request the user to enter his or her age. The program should show an alert message if the user is older than 30.*

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Test User Age</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
```



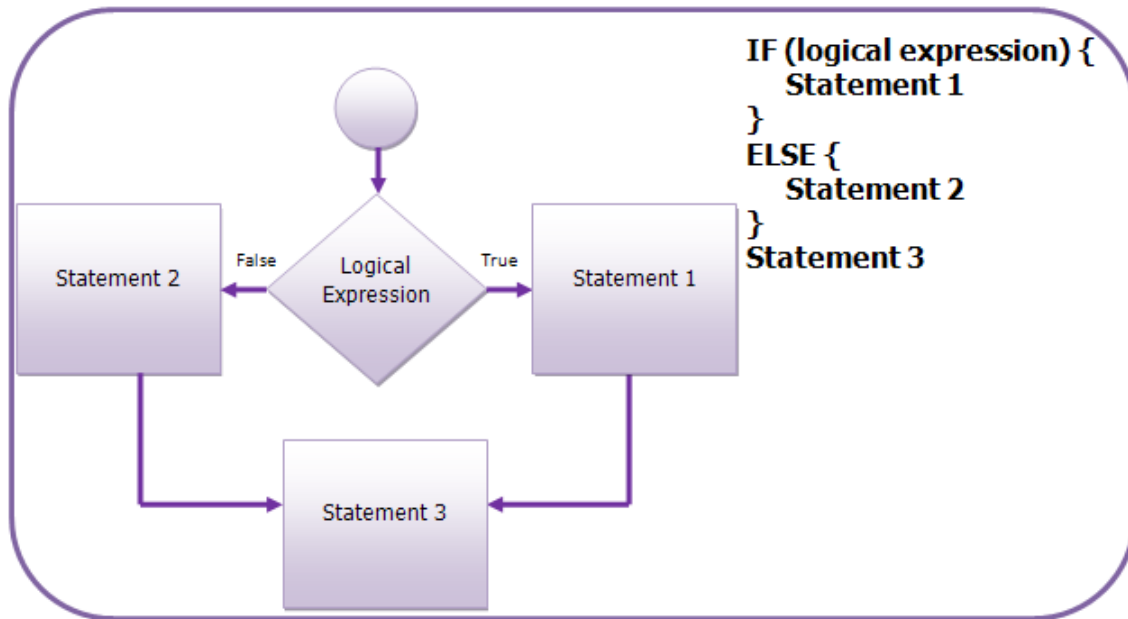
```
<script type="text/javascript">
/*  */</pre>
</div>
<div data-bbox="147 116 540 140" data-label="Text">
<pre>function display() {
    var current_age = parseInt(currentAge.value);</pre>
</div>
<div data-bbox="147 149 744 166" data-label="Text">
<p><b>Comment 1:</b> Use the <i>parseInt()</i> function to ensure the value is converted to integer.</p>
</div>
<div data-bbox="202 174 362 187" data-label="Text">
<pre>    if (current_age &gt; 30)</pre>
</div>
<div data-bbox="147 196 577 212" data-label="Text">
<p><b>Comment 2:</b> Test the condition, if the user is older than 30.</p>
</div>
<div data-bbox="257 220 550 233" data-label="Text">
<pre>        window.alert("You are older than 30!");</pre>
</div>
<div data-bbox="147 242 537 259" data-label="Text">
<p><b>Comment 3:</b> Execute the statement, display the alert.</p>
</div>
<div data-bbox="147 268 818 397" data-label="Text">
<pre>    }
/* ]]&gt; */
&lt;/script&gt;
&lt;/head&gt;

&lt;body&gt;
&lt;h1&gt;User Age&lt;/h1&gt;
    Enter the your age: &lt;input type = "text" name = "currentAge" /&gt;&lt;br /&gt;
    &lt;p&gt;&lt;input type = "button" value = "Display" onclick = "display(currentAge)" /&gt;&lt;/p&gt;
&lt;/body&gt;

&lt;/html&gt;</pre>
</div>
<div data-bbox="147 409 528 426" data-label="Text">
<p><b>Step 2:</b> Save the code you entered as testThirty.html.</p>
</div>
<div data-bbox="147 438 813 468" data-label="Text">
<p><b>Step 3:</b> Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.</p>
</div>
<div data-bbox="147 481 713 500" data-label="Text">
<p><b>Step 4:</b> Once the code has been validated open the browser and view your code.</p>
</div>
<div data-bbox="151 509 770 896" data-label="Image">
<img alt="Screenshot of a Windows Internet Explorer browser window displaying a web page titled 'User Age'. The page has a blue header with the title 'User Age' in large blue font. Below the header, there is a text input field with the placeholder text 'Enter the your age:' and the value '37'. To the left of the input field is a 'Display' button. An alert dialog box is overlaid on the page, titled 'Windows Internet Explorer', with a yellow warning icon and the message 'You are older than 30!'. The dialog box has an 'OK' button at the bottom."/>
</div>
<div data-bbox="848 886 888 936" data-label="Page-Footer">
<p>134</p>
</div>
```

## IF-THEN-ELSE Structure

This structure performs a single or group of statements if the logical expression is true and a different statement or group of statements when the logical expression is false. Should there be more than one statement that needs to be executed the statements must be enclosed in curly brackets { }.



### Step-by-Step create the JavaScript Program

*Develop a simple Web page in which you request the user to enter his or her age. The program should show an alert message "You are older than 30!" if the user is older than 30 and a message "You are younger than 30!" if the user is younger than 30.*

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
  
<head>  
    <title>Test User Age</title>  
    <link rel="stylesheet" href="styles.css" type="text/css" />  
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />  
  
<script type="text/javascript">  
/*  */<br/><br/>function display() {<br/>    var current_age = parseInt(currentAge.value);<br/></pre></div><div data-bbox="147 754 743 770" data-label="Text"><p><b>Comment 1:</b> Use the <code>parseInt()</code> function to ensure the value is converted to integer.</p></div><div data-bbox="202 778 362 791" data-label="Text"><pre>    if (current_age &gt; 30)</pre></div><div data-bbox="147 800 577 817" data-label="Text"><p><b>Comment 2:</b> Test the condition, if the user is older than 30.</p></div><div data-bbox="257 824 550 838" data-label="Text"><pre>        window.alert("You are older than 30!");</pre></div><div data-bbox="147 847 536 863" data-label="Text"><p><b>Comment 3:</b> Execute the statement, display the alert.</p></div><div data-bbox="202 875 566 899" data-label="Text"><pre>    else<br/>        window.alert("You are younger than 30!");</pre></div><div data-bbox="848 886 886 935" data-label="Page-Footer"><p>135</p></div>
```

**Comment 4:** Execute the statement if the test condition is false, display the alert.

```
}
/* ]]> */
</script>
</head>

<body>
<h1>User Age</h1>
    Enter the your age: <input type = "text" name = "currentAge" /><br />
    <p><input type = "button" value = "Display" onclick = "display(currentAge)" /></p>
</body>

</html>
```

**Step 2:** Save the code you entered as testThirtyYounger.html.

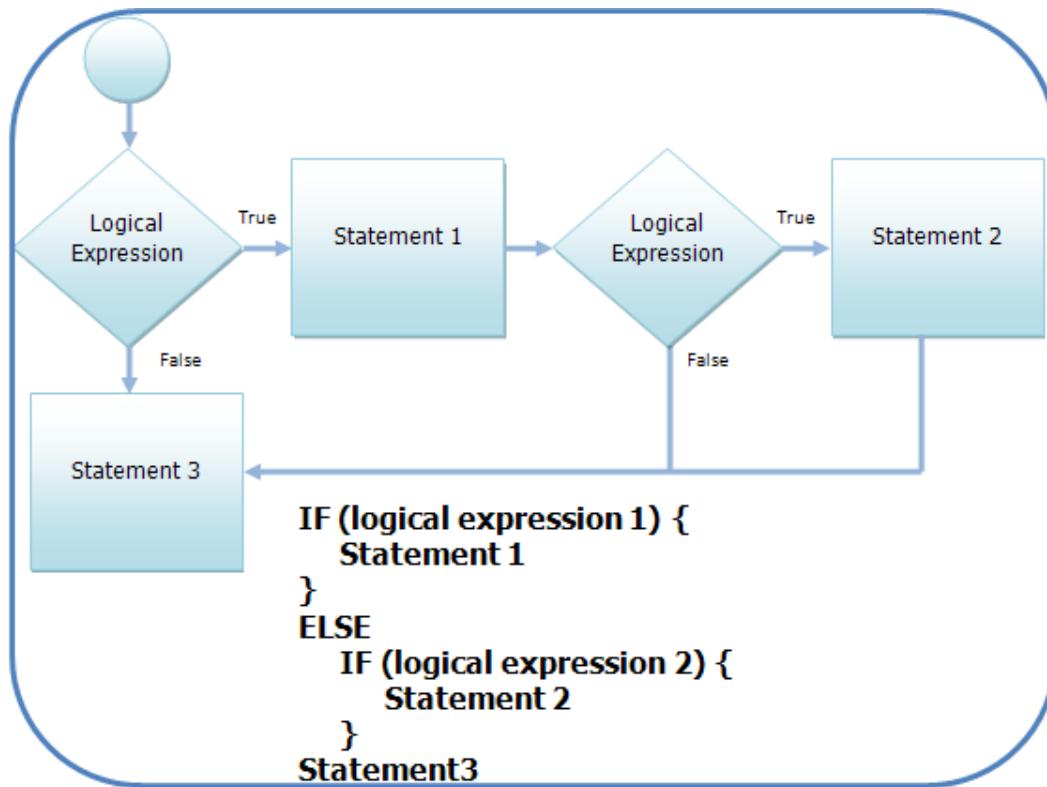
**Step 3:** Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.

**Step 4:** Once the code has been validated open the browser and view your code.



### Nested IF structure

This is the last and most complicated form of the IF statement, it contains an IF-THEN-ELSE or an IF structure within an IF-THEN-ELSE or an IF structure. The feature gives you the ability to handle multiple conditions.



### Step-by-Step create the JavaScript Program

Develop a simple Web page in which you request the user to enter his or her age as well as their gender. The program should show an alert message "You are older than 30 and female!" if the user is older than 30 and a female and a message "You are younger than 30 and female!" if the user is younger than 30 and a female. The same should be done if the user is a male with the relevant change to the message.

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
    <title>Test User Age</title>
    <link rel="stylesheet" href="styles.css" type="text/css" />
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

<script type="text/javascript">
/*  */

function display() {
    var current_age = parseInt(age.currentAge.value);
</pre>
</div>
<div data-bbox="147 771 743 788" data-label="Text">
<p><b>Comment 1:</b> Use the <i>parseInt()</i> function to ensure the value is converted to integer.</p>
</div>
<div data-bbox="202 796 371 810" data-label="Text">
<pre>    if (current_age &gt; 30){</pre>
</div>
<div data-bbox="147 821 577 838" data-label="Text">
<p><b>Comment 2:</b> Test the condition, if the user is older than 30.</p>
</div>
<div data-bbox="257 846 469 859" data-label="Text">
<pre>        if (age.gender.value == "M")</pre>
</div>
<div data-bbox="147 868 532 885" data-label="Text">
<p><b>Comment 3:</b> Test the condition, if the user is a male.</p>
</div>
<div data-bbox="311 892 687 906" data-label="Text">
<pre>            window.alert("You are older than 30 and a male!");
</pre>
</div>
<div data-bbox="848 886 886 935" data-label="Page-Footer">
<p>137</p>
</div>
```

**Comment 4:** Execute the statement, display the alert for male older than 30.

```
else
    window.alert("You are older than 30 and a female!");
```

**Comment 5:** Execute the statement, display the alert for female older than 30.

```
}
else{
```

**Comment 6:** Test condition false, user younger than 30.

```
if (age.gender.value == "M")
```

**Comment 7:** Test the condition, if the user is a male.

```
    window.alert("You are younger than 30 and a male!");
```

**Comment 8:** Execute the statement, display the alert for male younger than 30.

```
else
    window.alert("You are younger than 30 and a female!");
```

**Comment 9:** Execute the statement, display the alert for female younger than 30.

```
    }
}

/* ]]> */
</script>
</head>

<body>
<h1>User Age</h1>
    Enter the your age: <input type = "text" name = "currentAge" /><br />
    <p><input type = "button" value = "Display" onclick = "display(currentAge)" /></p>
</body>

</html>
```

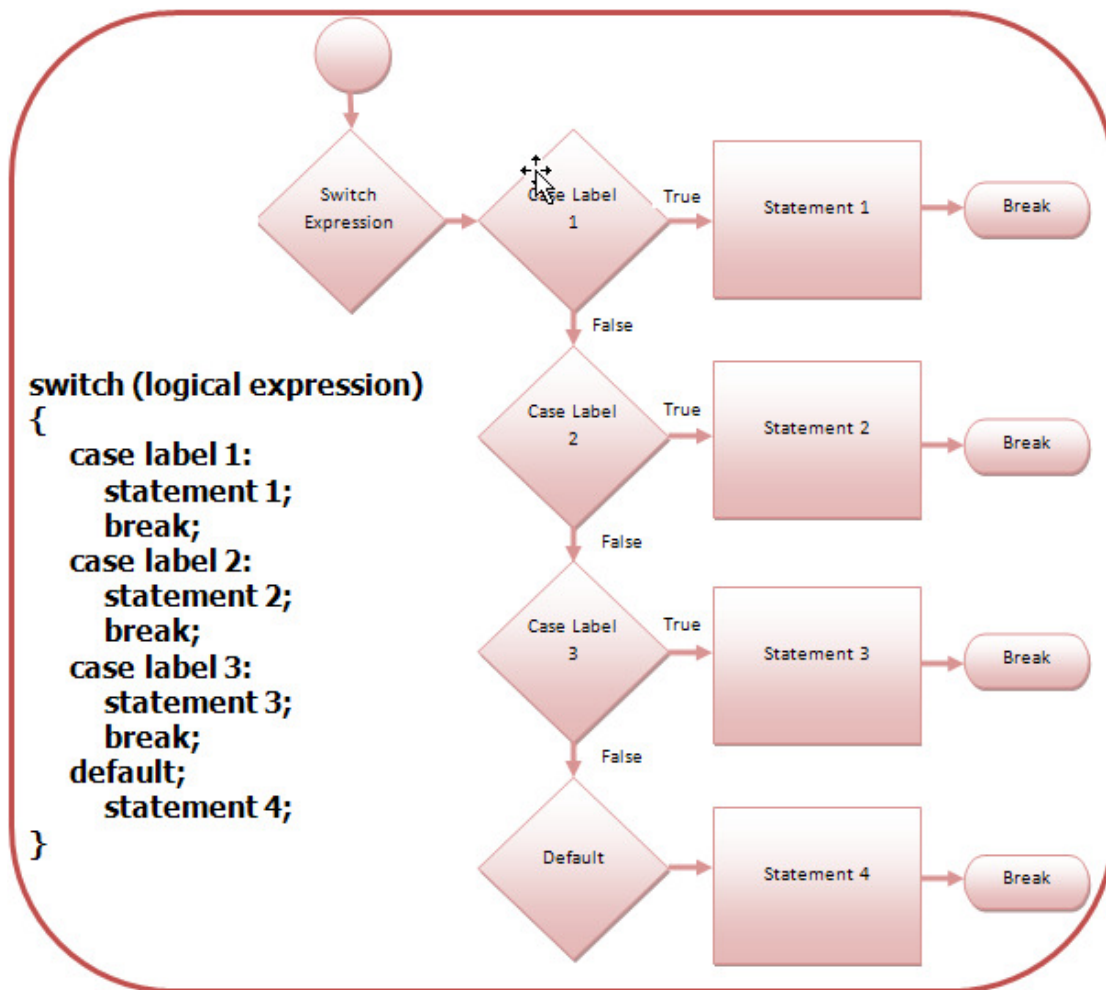
**Step 2:** Save the code you entered as testThirtyGender.html.

**Step 3:** Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.

**Step 4:** Once the code has been validated open the browser and view your code.

The screenshot shows a web browser window with a title bar. The page content includes a heading 'User Age' in blue. Below it, there is a form with two input fields: 'Enter the your age:' with the value '27' and 'Enter your gender:' with the value 'M'. A 'Display' button is located below the form. An alert dialog box from 'Windows Internet Explorer' is overlaid on the page, displaying a yellow warning triangle icon and the message 'You are younger than 30 and a male!'. The dialog has an 'OK' button at the bottom.

## Switch Statement



When the value of the switch statement's expression matches that of the value in the case label, the statements inside the case label is executed. The default label is executed when there are no case labels that match the switch statement. A break statement must be included to end the switch statement once it has performed all the tasks required. When a nested *if ... else* statement contains more than 3 levels it is better to use a switch statement.

### Step-by-Step create the JavaScript Program

Develop a simple Web page in which you request the user to enter his or her age. If the user falls between a certain age group the program should show the message indicated in the table below.

Age	Message
0 – 10	You are still a youngster
10 – 20	Those teenage years
20 – 30	You are a young adult
30 – 40	The age to settle down
40 – 50	Life is good!
50 – 60	Almost finished working!
60 – 70	Retirement is fun!
70+	You are growing old now...

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

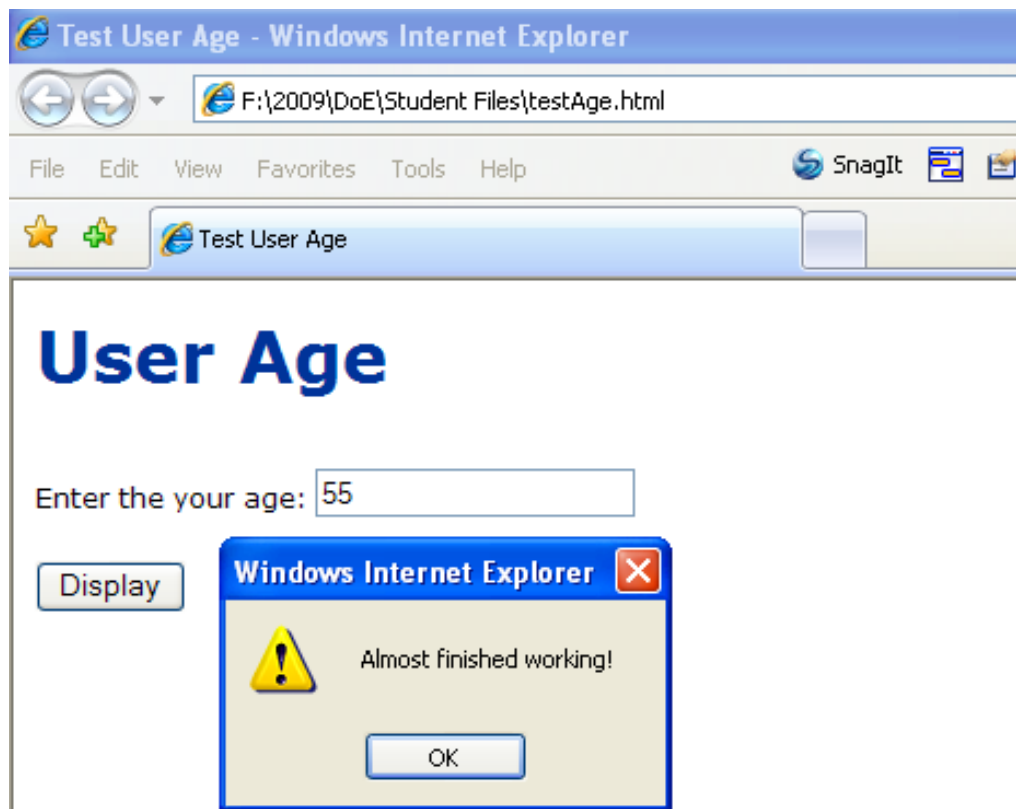
<head>
  <title>Test User Age</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

<script type="text/javascript">
/*  */

function display() {
  var current_age = parseInt(currentAge.value);</pre></div><div data-bbox="147 273 743 290" data-label="Text"><p><b>Comment 1:</b> Use the <i>parseInt()</i> function to ensure the value is converted to integer.</p></div><div data-bbox="203 298 319 310" data-label="Text"><pre>  switch (true) {</pre></div><div data-bbox="147 319 849 349" data-label="Text"><p><b>Comment 2:</b> The conditions inside the case label returns a Boolean value, therefore the switch needs to test which one of the Boolean values returns true.</p></div><div data-bbox="257 358 634 392" data-label="Text"><pre>    case (current_age&gt;=0 &amp;&amp; current_age&lt; 10):
      window.alert("You are still a youngster.");
      break;</pre></div><div data-bbox="147 401 849 432" data-label="Text"><p><b>Comment 3:</b> Each case tests the age entered by the user and returns either a value of true or false, if the case returns the value of true the alert message will be displayed on screen.</p></div><div data-bbox="147 441 743 807" data-label="Text"><pre>    case (current_age&gt;= 10 &amp;&amp; current_age&lt; 20):
      window.alert("Those teenage years.");
      break;
    case (current_age&gt;= 20 &amp;&amp; current_age&lt; 30):
      window.alert("You are a young adult.");
      break;
    case (current_age&gt;= 30 &amp;&amp; current_age&lt; 40):
      window.alert("The age to settle down.");
      break;
    case (current_age&gt;= 40 &amp;&amp; current_age&lt; 50):
      window.alert("Life is good!");
      break;
    case (current_age&gt;= 50 &amp;&amp; current_age&lt; 60):
      window.alert("Almost finished working!");
      break;
    case (current_age&gt;= 60 &amp;&amp; current_age&lt; 70):
      window.alert("Retirement is fun!");
      break;
    case (current_age&gt;= 70):
      window.alert("You are growing old now...");
  }
}
/* ]]&gt; */
&lt;/script&gt;
&lt;/head&gt;

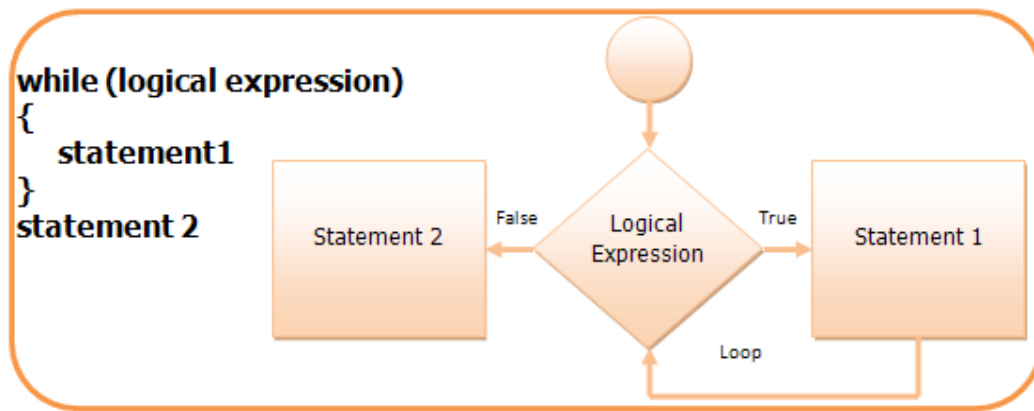
&lt;body&gt;
&lt;h1&gt;User Age&lt;/h1&gt;
  Enter the your age: &lt;input type = "text" name = "currentAge" /&gt;&lt;br /&gt;
  &lt;p&gt;&lt;input type = "button" value = "Display" onclick = "display()" /&gt;&lt;/p&gt;
&lt;/body&gt;

&lt;/html&gt;</pre></div><div data-bbox="147 820 514 837" data-label="Text"><p><b>Step 2:</b> Save the code you entered as testAge.html.</p></div><div data-bbox="147 849 812 879" data-label="Text"><p><b>Step 3:</b> Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.</p></div><div data-bbox="147 891 713 908" data-label="Text"><p><b>Step 4:</b> Once the code has been validated open the browser and view your code.</p></div><div data-bbox="849 886 886 936" data-label="Page-Footer"><p>140</p></div>
```





## While Statement



A while loop is a programming technique that allows you to repeat certain code while a logical expression is true. When the while loop begins the logical expression is checked, if it is true the code between the curly brackets {} is executed. After the last statement is executed, the while loops back to the logical expression and begins again. When the logical expression becomes false the loop is exited without executing the code in the brackets.

### Step-by-Step create the JavaScript Program

*Develop a simple Web page that computes the 12 times table and shows it in an alert box.*

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
    <title>12 Times Table</title>
    <link rel="stylesheet" href="styles.css" type="text/css" />
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

<script type="text/javascript">
/* <![CDATA[ */
var display = "";
```

**Comment 1:** Declare a variable that will hold the output.

```
var changing_variable = 1;
```

**Comment 2:** Declare the variable that will serve as the counter inside the while loop.

```
var answer = 0;
```

**Comment 3:** Declare the variable that will hold the answer for the multiplication.

```
while (changing_variable <= 10)
```

**Comment 4:** The while loop must be executed while the counter variable is less than or equal to 10, thus the moment it becomes 11 the loop will no longer be executed.

```
{
    answer = 12 * changing_variable;
```

**Comment 5:** Multiply the counter with 12.

```
display = display + "12 X " + changing_variable + " = " + answer + "\n";
```

**Comment 6:** Add the counter and the answer to the text that must be displayed.

```
changing_variable++;
```

**Comment 7:** Change the value of the counter by incrementing it by 1.

```
    }  
    alert(display)
```

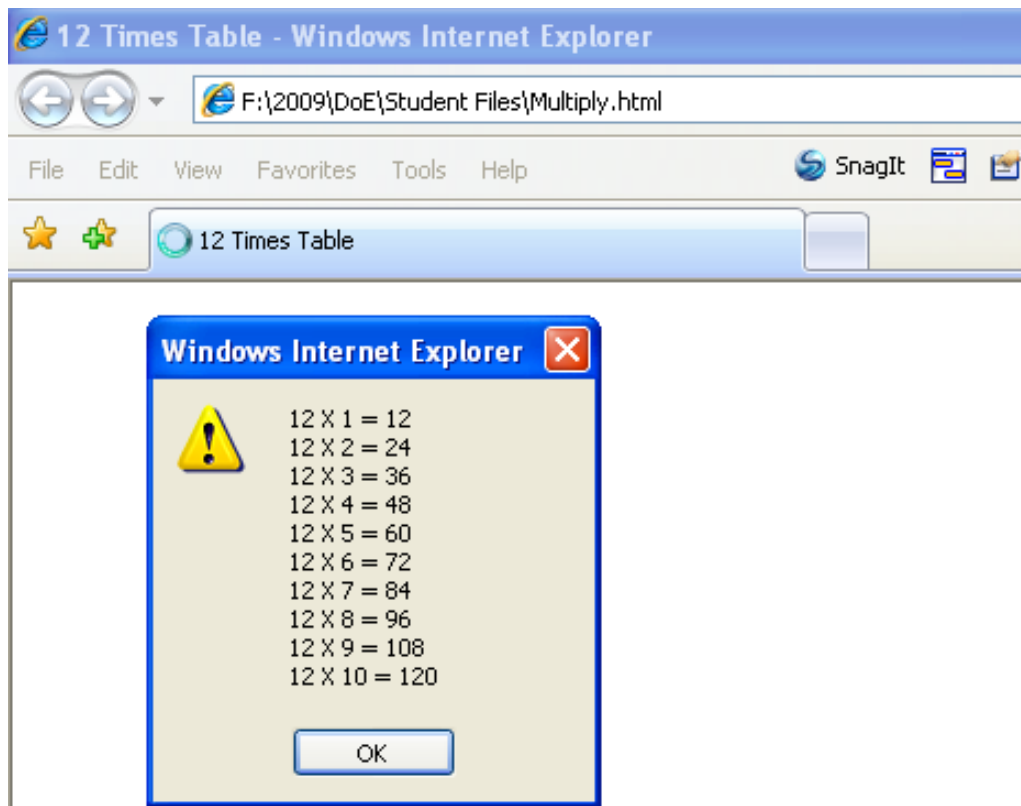
**Comment 8:** Display the output in the alert box.

```
/* ]]> */  
</script>  
</head>  
  
<body>  
  
</html>  
    switch (true) {
```

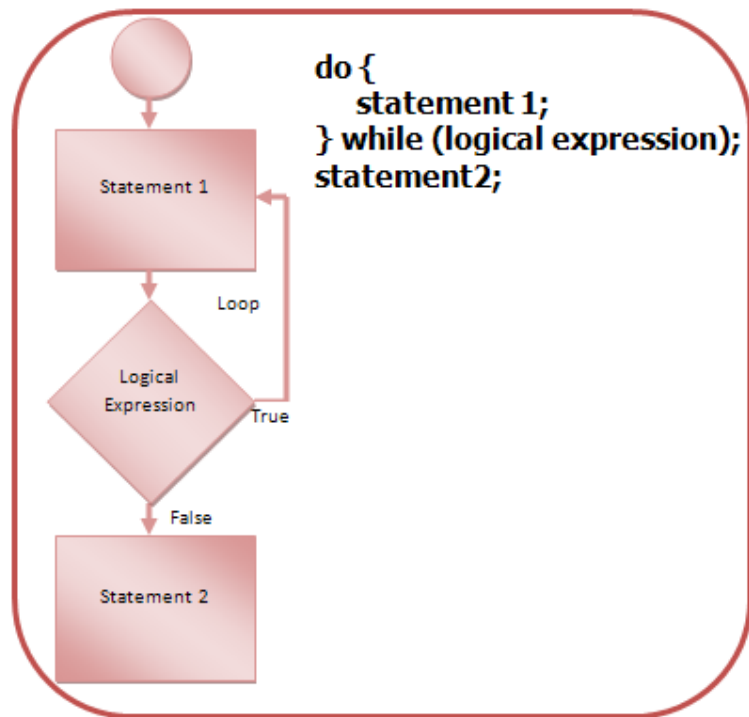
**Step 2:** Save the code you entered as Multiply.html.

**Step 3:** Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.

**Step 4:** Once the code has been validated open the browser and view your code.



## Do...While Statement



The do while loop is slightly different from the while loop as the statement is executed at least once before the logical expression is checked. If the logical condition is true the loop is repeated, but if it is false the execution stops and control is passed to the next statement.

### Step-by-Step create the JavaScript Program

*Develop a simple Web page where the user can play a number guessing games. The user must be prompted to enter any number between 1 and 10, if the number is not equal to the number specified the user must enter the number again. If the user guessed correctly a congratulations message must be shown.*

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Lucky Numbers</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

  <script type="text/javascript">
    /*  */
    do
    {
      var userResponse;</pre></div><div data-bbox="147 813 579 830" data-label="Text"><p><b>Comment 1:</b> Declare a variable that will hold the user input.</p></div><div data-bbox="257 838 720 852" data-label="Text"><pre>      userResponse = prompt ("Enter a number between 1 and 10", "");</pre></div><div data-bbox="147 860 452 877" data-label="Text"><p><b>Comment 2:</b> Get the input from the user.</p></div><div data-bbox="202 885 420 898" data-label="Text"><pre>    } while (userResponse != "8")</pre></div><div data-bbox="849 886 886 936" data-label="Page-Footer"><p>144</p></div>
```

**Comment 3:** Compare the user input to that of the specified number, if it is not the same the user will be requested to enter the number again.

```
alert("Congratulations!")
```

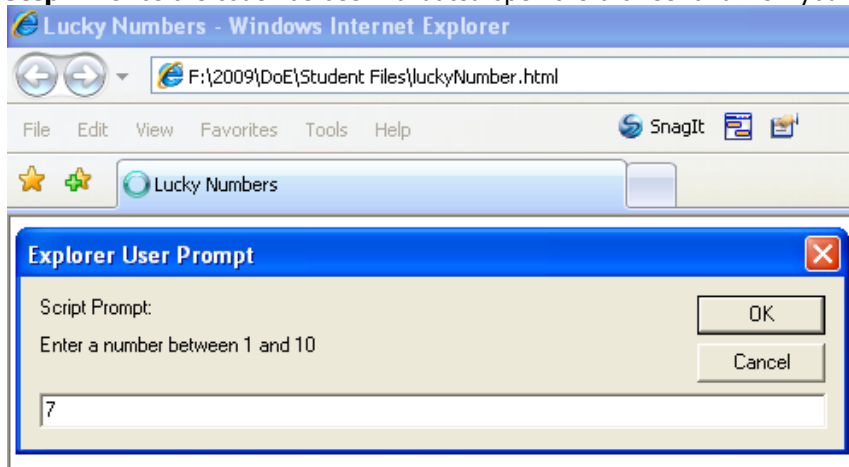
**Comment 4:** If the user entered the correct number the alert box will be displayed.

```
/* ]]> */  
</script>  
</head>  
  
<body>  
  
</html>
```

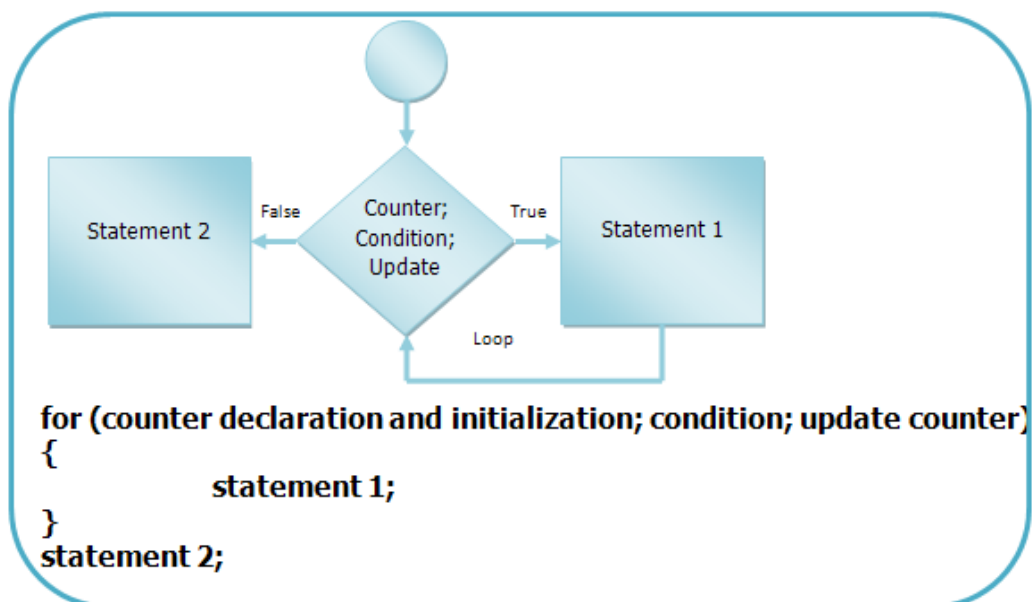
**Step 2:** Save the code you entered as luckyNumber.html.

**Step 3:** Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.

**Step 4:** Once the code has been validated open the browser and view your code.



## For Statement



The for loop is used if you know exactly how many times the script needs to be looped. The counter is declared and initialized, then the condition is checked, if the condition is true the statements inside the curly brackets {} are executed. Once all the statements were executed the counter is updated and the condition is checked again, this is repeated until the condition becomes false.

### Step-by-Step create the JavaScript Program

*Develop a simple Web page that computes the 8 times table and shows it in an alert box.*

**Step 1:** Open Windows Notepad or your text editor and enter the following code.

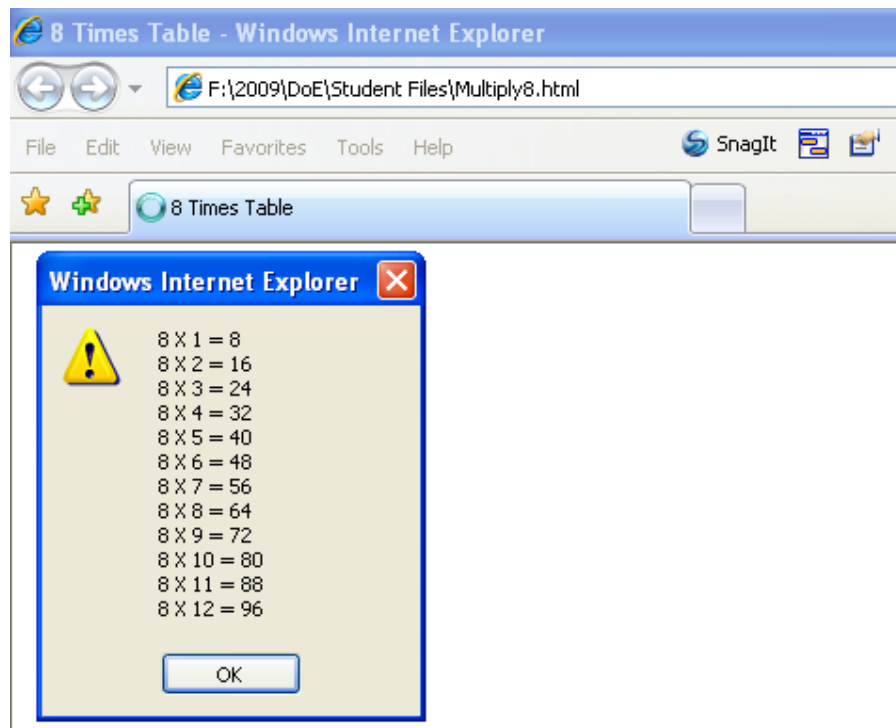
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
    <title>8 Times Table</title>
    <link rel="stylesheet" href="styles.css" type="text/css" />
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

    <script type="text/javascript">
        /*  */
        var display = "";</pre></div><div data-bbox="147 392 555 407" data-label="Text"><p><b>Comment 1:</b> Declare a variable that will hold the output.</p></div><div data-bbox="147 416 262 428" data-label="Text"><pre>var answer = 0;</pre></div><div data-bbox="147 438 718 454" data-label="Text"><p><b>Comment 2:</b> Declare the variable that will hold the answer for the multiplication.</p></div><div data-bbox="147 463 719 476" data-label="Text"><pre>for (var changing_variable = 1; changing_variable &lt;= 12; changing_variable++)</pre></div><div data-bbox="147 485 849 515" data-label="Text"><p><b>Comment 3:</b> Start the for loop by declaring the counter, checking whether it is still less than 12 and incrementing it by 1.</p></div><div data-bbox="170 525 403 547" data-label="Text"><pre>{
    answer = 8 * changing_variable;</pre></div><div data-bbox="147 557 447 572" data-label="Text"><p><b>Comment 4:</b> Multiply the counter with 8.</p></div><div data-bbox="170 581 697 594" data-label="Text"><pre>display = display + "8 X " + changing_variable + " = " + answer + "\n";</pre></div><div data-bbox="147 603 719 619" data-label="Text"><p><b>Comment 5:</b> Add the counter and the answer to the text that must be displayed.</p></div><div data-bbox="147 629 255 651" data-label="Text"><pre>}
alert(display)</pre></div><div data-bbox="147 660 498 676" data-label="Text"><p><b>Comment 6:</b> Display the output in the alert box.</p></div><div data-bbox="147 685 319 772" data-label="Text"><pre>/* ]]&gt; */
&lt;/script&gt;
&lt;/head&gt;

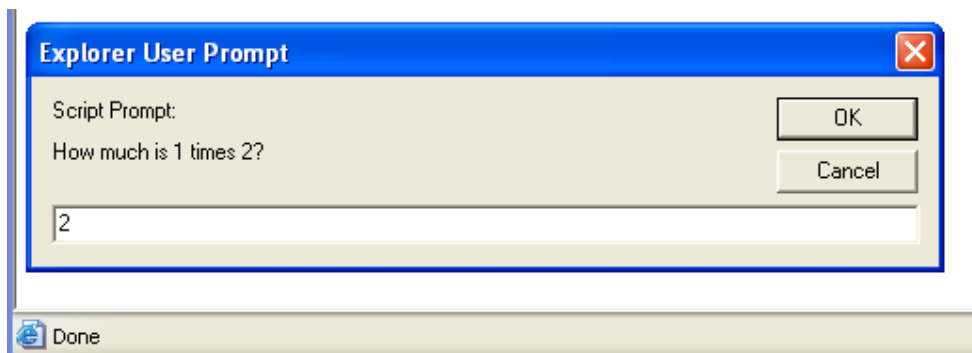
&lt;body&gt;

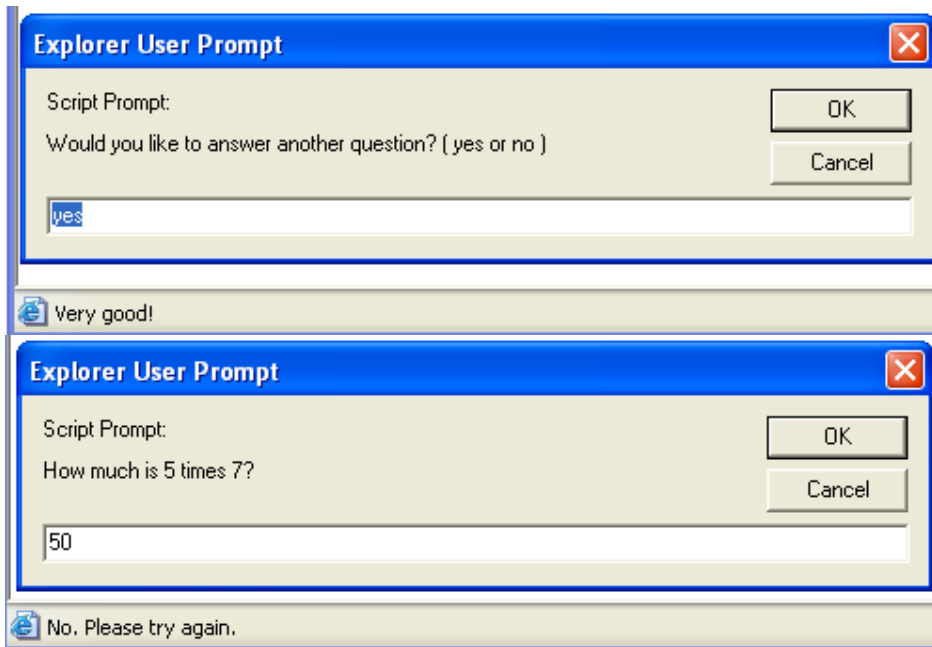
&lt;/html&gt;
    switch (true) {</pre></div><div data-bbox="147 785 523 801" data-label="Text"><p><b>Step 2:</b> Save the code you entered as Multiply8.html.</p></div><div data-bbox="147 812 812 843" data-label="Text"><p><b>Step 3:</b> Validate the code using the Markup Validation Service provided by the World Wide Web Consortium.</p></div><div data-bbox="147 855 713 873" data-label="Text"><p><b>Step 4:</b> Once the code has been validated open the browser and view your code.</p></div><div data-bbox="849 886 886 936" data-label="Page-Footer"><p>146</p></div>
```



### Exercise

Develop a simple Web site where users can play a math game. The user should be asked a questions such as "How much is  $5 \times 8$ ?", the user should then type the answer into a text field. This can be done using the prompt dialog box. Use *Math.random* method to produce two positive one-digit integers. If the user answered the question correctly the message "Very good!" must be displayed in the browser status bar. If the user answers incorrectly the message "No, please try again!" must be displayed in the browser's status bar, the question must be repeated until the user gets it right. Save the code as Math.html.





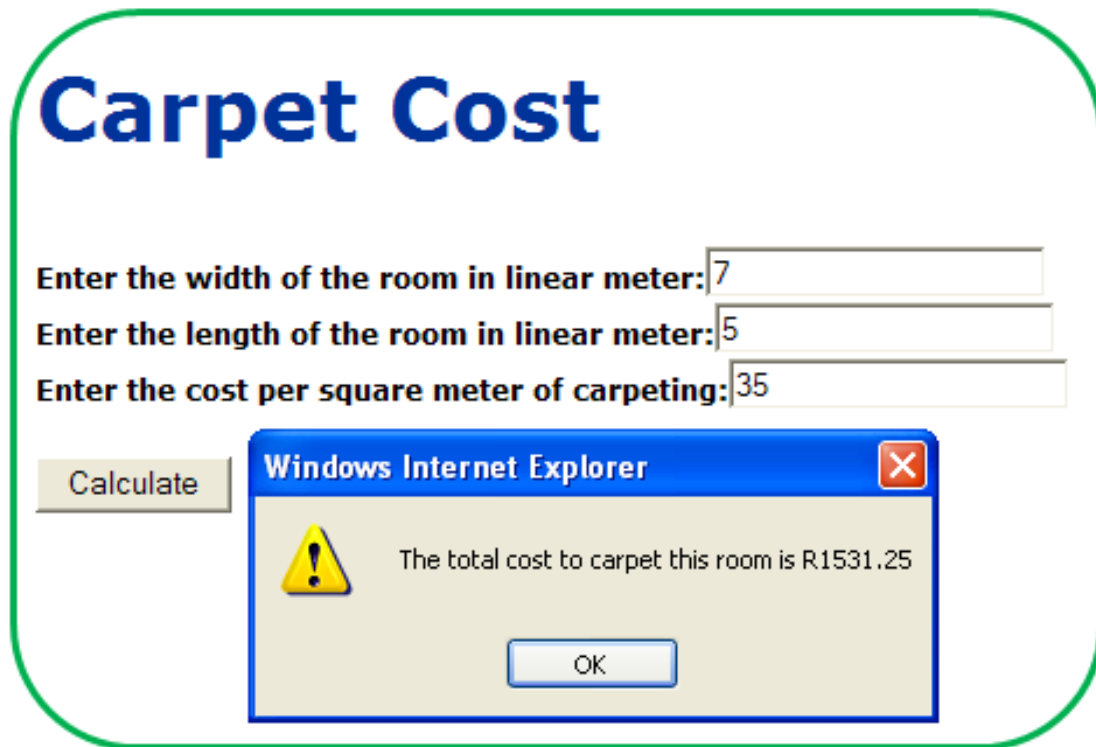
## Practical Applications

### Introduction

This chapter will give you many examples of questions, their solutions and explanations of what each piece of code does. The best way to study any programming language is by practically applying the theory.

### Application 1

Create a simple program that calculates the square meter of carpet required to carpet a room. Include three text boxes. Create one text box for the width of the room in meters and another for the length of the room in meters. Also create a text box for the cost per square meter of carpeting. When you calculate the cost, add 25% to the total number of square meters to account for cupboards and other features in the room. Display the total cost in an alert dialog box. Save the document as CarpetCost.html.



### Step-by-Step Suggested Solution

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Carpet Cost</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

  <script type="text/javascript">
    /*  */

    function calculate() {
      var squareMeter = 0;</pre></div><div data-bbox="849 886 886 935" data-label="Page-Footer"><p>149</p></div>
```



**Comment 1:** Declare and initialize the variable in which the square meter of the room will be calculated.

```
var totalCost = 0;
```

**Comment 2:** Declare and initialize the variable in which the cost of the carpet will be calculated.

```
squareMeter = document.Estimador.width.value * document.Estimador.length.value;
```

**Comment 3:** Calculate the square meter of the room by multiplying the width of the room by the length of the room. The measurements is obtained by referencing the document object, form named Estimator, field named width/length, value of the field.

```
totalCost = squareMeter * document.Estimador.cost_per_meters.value;
```

**Comment 3:** Calculate the cost of the carpets by multiplying the square meters of the room by the price of the carpet. The price is obtained by referencing the document object, form named Estimator, field named cost\_per\_meter, value of the field.

```
totalCost *= 1.25;
```

**Comment 4:** Calculate the final cost of the carpet by adding 25% to the cost, this is accomplished by multiplying the cost by 1.25 using the compound multiplication assignment operator.

```
window.alert("The total cost to carpet this room is R" + totalCost);
```

**Comment 5:** Display the output in an alert box.

```
}  
/* ]]> */  
</script>  
</head>  
  
<body>  
<h1>Carpet Cost</h1>
```

**Comment 6:** Print the heading to the screen.

```
<form name="Estimator">
```

**Comment 7:** Declare the form with name "Estimator".

```
<p><b>Enter the width of the room in linear meter:</b><input type="text" name="width" /><br />  
<b>Enter the length of the room in linear meter:</b><input type="text" name="length" /><br />  
<b>Enter the cost per square meter of carpeting:</b><input type="text" name="cost_per_meters" />
```

**Comment 8:** Declare the input fields.

```
</p>  
<p><input type = "button" value = "Calculate" onclick = "calculate()" /></p>
```

**Comment 9:** Declare the button which calls the "calculate" function when it is clicked.

```
</form>  
</body>  
  
</html>
```

## Application 2

Develop a simple JavaScript program that displays the current time as hour, minute and second in a text box. Save the program as Time.html

**The current time is:**

11:06:36

### Step-By-Step Suggested Solution

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
    <title>Time</title>
    <link rel="stylesheet" href="styles.css" type="text/css" />
    <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

<script type="text/javascript">
/*  */
var timer = null;</pre></div><div data-bbox="147 371 849 401" data-label="Text"><p><b>Comment 1:</b> Declare the variable that will be used as the "counter" for the <code>setTimeout()</code> and <code>clearTimeout()</code> methods.</p></div><div data-bbox="147 414 354 436" data-label="Text"><pre>function stop() {
    clearTimeout(timer);</pre></div><div data-bbox="147 446 849 490" data-label="Text"><p><b>Comment 2:</b> The <code>clearTimeout()</code> method is used to cancel a <code>setTimeout()</code> method before its code can be executed, it receives a single argument (<code>timer</code>), which is the variable that represents a <code>setTimeout()</code> method call.</p></div><div data-bbox="147 504 161 515" data-label="Text"><pre>}</pre></div><div data-bbox="147 525 369 547" data-label="Text"><pre>function start() {
    var time = new Date();</pre></div><div data-bbox="147 558 849 587" data-label="Text"><p><b>Comment 3:</b> Declare a variable that creates a <code>Date</code> object that contains the current date and time from the local computer.</p></div><div data-bbox="202 596 413 608" data-label="Text"><pre>    var hours = time.getHours();</pre></div><div data-bbox="147 617 849 647" data-label="Text"><p><b>Comment 4:</b> Once the new <code>Date</code> object is created you can manipulate the date and time in the variable using the methods of the <code>Date</code> class. <code>getHours()</code> returns the hour of the <code>Date</code> object.</p></div><div data-bbox="202 660 443 672" data-label="Text"><pre>    var minutes = time.getMinutes();</pre></div><div data-bbox="147 682 669 697" data-label="Text"><p><b>Comment 5:</b> <code>getMinutes()</code> method returns the minutes of the <code>Date</code> object.</p></div><div data-bbox="202 710 546 722" data-label="Text"><pre>    minutes=(minutes &lt; 10) ? "0" : "" + minutes;</pre></div><div data-bbox="147 732 849 762" data-label="Text"><p><b>Comment 6:</b> If the current minutes are less than 10 a zero needs to be added in front of the minute for display purposes.</p></div><div data-bbox="202 774 443 786" data-label="Text"><pre>    var seconds = time.getSeconds();</pre></div><div data-bbox="147 796 674 812" data-label="Text"><p><b>Comment 7:</b> <code>getSeconds()</code> method returns the minutes of the <code>Date</code> object.</p></div><div data-bbox="202 824 546 836" data-label="Text"><pre>    seconds=((seconds &lt; 10) ? "0" : "") + seconds;</pre></div><div data-bbox="147 846 849 877" data-label="Text"><p><b>Comment 8:</b> If the current second are less than 10 a zero needs to be added in front of the minute for display purposes.</p></div><div data-bbox="202 888 576 900" data-label="Text"><pre>    var clock = hours + ":" + minutes + ":" + seconds;</pre></div><div data-bbox="849 886 889 935" data-label="Page-Footer"><p>151</p></div>
```

**Comment 9:** Declare the variable that will display the current time, initialize it by adding the hours, minutes and seconds.

```
document.clock.display.value = clock;
```

**Comment 10:** Assign the current time by referencing the document object, form named clock, field display, value of the field.

```
timer = setTimeout("start()",1000);
```

**Comment 11:** The setTimeout() method is used to execute the start() function every 1000 milliseconds (1 second).

```
}  
/* ]]> */  
</script>  
</head>
```

```
<body onload="start()" onunload="stop()">
```

**Comment 12:** The start() function is called the moment the html document is loaded in the browser and the stop() function is called when the document is closed.

```
<form name="clock">
```

**Comment 13:** Declare the form with the name "clock".

```
<h3>The current time is: </h3>
```

**Comment 14:** Write the heading to the screen.

```
<input type="text" name="display" size="20">
```

**Comment 15:** Declare the text field with the name "display" in which the changing time will be printed on screen.

```
</form>  
</body>  
</html>
```

### Application 3

Develop as simple Web page to accept the current day of the week and display a message for each day, use the switch statement. Save the program as WeekDays.html

Sunday – Peace and quiet

Monday – Blue as can be

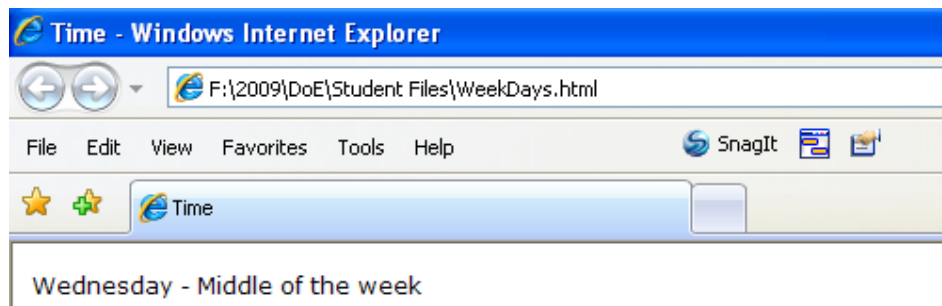
Tuesday – Getting into things

Wednesday – Middle of the week

Thursday – Almost there

Friday – Finally Friday

Saturday – Party on!



## Step-By-Step Suggested Solution

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
  <title>Week Days</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
```

```
<script type="text/javascript">
/* <![CDATA[ */
var d = new Date()
```

**Comment 1:** Declare a variable that creates a Date object that contains the current date and time from the local computer.

```
theDay=d.getDay()
```

**Comment 2:** Once the new Date object is created you can manipulate the date and time in the variable using the methods of the Date class. getDay returns the day of the Date object.

```
switch (theDay)
```

**Comment 3:** The days of the week and months of the year are stored in a Date object using numeric representations, starting with zero, similar to an array.

```
{
  case 0:
    document.write ("Sunday - Peace and quite")
  break
  case 1:
    document.write("Monday - Blue as can be")
  break
  case 2:
    document.write("Tuesday - Getting into things")
  break
  case 3:
    document.write("Wednesday - Middle of the week")
  break
  case 4:
    document.write("Thursday - Almost there")
  break
  case 5:
    document.write("Finally Friday")
  break
  default:
    document.write("Saturday - Party on!")
}
```

**Comment 4:** Each case tests the day and returns either a value of true or false, if the case returns the value of true the alert message will be displayed on screen.

```
}
/* ]]> */
</script>
</head>
```

```
<body>
</body>

</html>
```

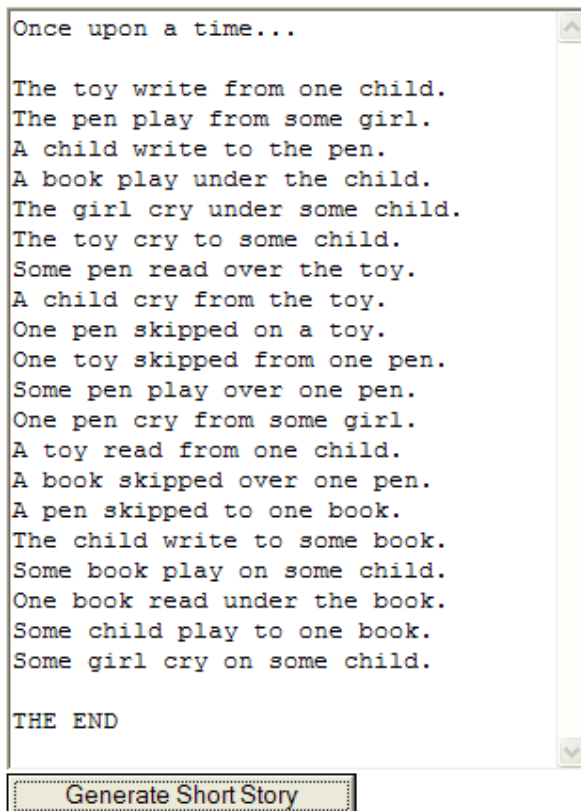
## Application 4

Develop a simple Web page that uses random number generation to create sentences. Use four arrays of strings called *article*, *noun*, *verb* and *preposition*. Create a sentence by selecting a word at random from each array in the following order: *article*, *noun*, *verb*, *preposition*, *article* and *noun*. As each word is picked, concatenate it to the previous words in the sentence. The words should be separated by spaces. When the final sentence is output, it should start with a capital letter and end with a period (full stop).

The arrays should be filled as follows:

- the *article* array should contain the articles "the", "a", "one", "some" and "any"
- the *noun* array should contain the nouns "child", "book", "toy", "pen", "girl"
- the *verb* array should contain the verbs "play", "cry", "write", "read", "skipped"
- the *preposition* array should contain the prepositions "to", "from", "over", "under" and "on".

The program should generate 20 sentences to form a short story and output the result to an HTML textarea. The story should begin with a line reading "Once upon a time ..." and end with a line reading "THE END". Save the program as Story.html



Once upon a time...

The toy write from one child.  
The pen play from some girl.  
A child write to the pen.  
A book play under the child.  
The girl cry under some child.  
The toy cry to some child.  
Some pen read over the toy.  
A child cry from the toy.  
One pen skipped on a toy.  
One toy skipped from one pen.  
Some pen play over one pen.  
One pen cry from some girl.  
A toy read from one child.  
A book skipped over one pen.  
A pen skipped to one book.  
The child write to some book.  
Some book play on some child.  
One book read under the book.  
Some child play to one book.  
Some girl cry on some child.

THE END

Generate Short Story

## Step-By-Step Suggested Solution

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Short Story</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

  <script type="text/javascript">
    /*  */
      var article = [ "the", "a", "one", "some", "some"];
      var noun = [ "child", "book", "toy", "pen", "girl" ];
      var verb = [ "play", "cry", "write", "read", "skipped" ];
      var preposition = [ "to", "from", "over", "under", "on" ];</pre></div><div data-bbox="147 292 613 308" data-label="Text"><p><b>Comment 1:</b> Declare the variables containing the arrays of words.</p></div><div data-bbox="214 316 403 329" data-label="Text"><pre>function randomSentence()</pre></div><div data-bbox="147 338 849 369" data-label="Text"><p><b>Comment 2:</b> The function generates a random sentence in the form of an "article noun verb preposition article noun."</p></div><div data-bbox="214 378 336 400" data-label="Text"><pre>{
  var sentence;</pre></div><div data-bbox="147 410 642 426" data-label="Text"><p><b>Comment 3:</b> Declare a variable that will hold the completed sentence.</p></div><div data-bbox="234 436 359 447" data-label="Text"><pre>var firstLetter;</pre></div><div data-bbox="147 457 797 472" data-label="Text"><p><b>Comment 4:</b> Declare a variable that will hold the first letter of the first word of the sentence.</p></div><div data-bbox="234 481 639 494" data-label="Text"><pre>sentence = article[ Math.floor( Math.random() * 5 ) ];</pre></div><div data-bbox="147 503 849 533" data-label="Text"><p><b>Comment 5:</b> Use the Math.random class of the Math.floor object to randomly select an article from the var article array to be used as the first word of the sentence.</p></div><div data-bbox="234 542 520 555" data-label="Text"><pre>firstLetter = sentence.substr( 0, 1 );</pre></div><div data-bbox="147 564 662 580" data-label="Text"><p><b>Comment 6:</b> Use the substr() method to break the first word into letters.</p></div><div data-bbox="234 589 536 601" data-label="Text"><pre>firstLetter = firstLetter.toUpperCase();</pre></div><div data-bbox="147 610 734 626" data-label="Text"><p><b>Comment 7:</b> Capitalize the first letter of the word using the toUpperCase() method.</p></div><div data-bbox="234 635 580 647" data-label="Text"><pre>sentence = firstLetter + sentence.substr( 1 );</pre></div><div data-bbox="147 656 849 686" data-label="Text"><p><b>Comment 8:</b> Combine the first letter with the rest of the word in the sentence using the substr() method.</p></div><div data-bbox="234 695 720 751" data-label="Text"><pre>sentence += " " + noun[ Math.floor( Math.random() * 5 ) ];
sentence += " " + verb[ Math.floor( Math.random() * 5 ) ];
sentence += " " + preposition[ Math.floor( Math.random() * 5 ) ];
sentence += " " + article[ Math.floor( Math.random() * 5 ) ];
sentence += " " + noun[ Math.floor( Math.random() * 5 ) ];</pre></div><div data-bbox="147 761 849 791" data-label="Text"><p><b>Comment 9:</b> Use the Math.random class of the Math.floor object to randomly select words from the noun, verb, preposition, article and noun arrays complete the sentence.</p></div><div data-bbox="234 800 359 812" data-label="Text"><pre>sentence += ".";</pre></div><div data-bbox="147 821 576 838" data-label="Text"><p><b>Comment 10:</b> Add a full stop (.) to the end of the sentence.</p></div><div data-bbox="234 846 359 858" data-label="Text"><pre>return sentence;</pre></div><div data-bbox="147 867 688 884" data-label="Text"><p><b>Comment 11:</b> Send the completed sentence to the generateStory() function.</p></div><div data-bbox="214 892 227 904" data-label="Text"><pre>}</pre></div><div data-bbox="849 886 886 936" data-label="Page-Footer"><p>155</p></div>
```

```
function generateStory()
```

**Comment 12:** The function generates a story of 20 random sentences and displays them in the text area of the form.

```
{
    document.myForm.output.value = "Once upon a time...\n\n";
```

**Comment 13:** Assign the first sentence to the text field by referencing the document object, form named myForm, field output, value of the field.

```
for ( var i = 1; i <= 20; i++ ) {
```

**Comment 14:** Start the for loop by declaring the counter (j), checking whether it is less than or equal to 20 and incrementing it by 1.

```
document.myForm.output.value += randomSentence() + "\n";
```

**Comment 15:** Add the sentences to the text field, the randomSentence() function is called to create each sentence.

```
}

document.myForm.output.value += "\nTHE END";
```

**Comment 13:** Assign the last sentence to the text field by referencing the document object, form named myForm, field output, value of the field.

```
}

/* ]]> */
</script>
</head>

<body>
    <form name = "myForm">
```

**Comment 14:** Declare the form with the name "myForm".

```
<textarea name = "output" rows = "25" cols = "36"></textarea>
```

**Comment 15:** Declare the text area with the name "output", make the size of the text area 25 rows and 36 columns.

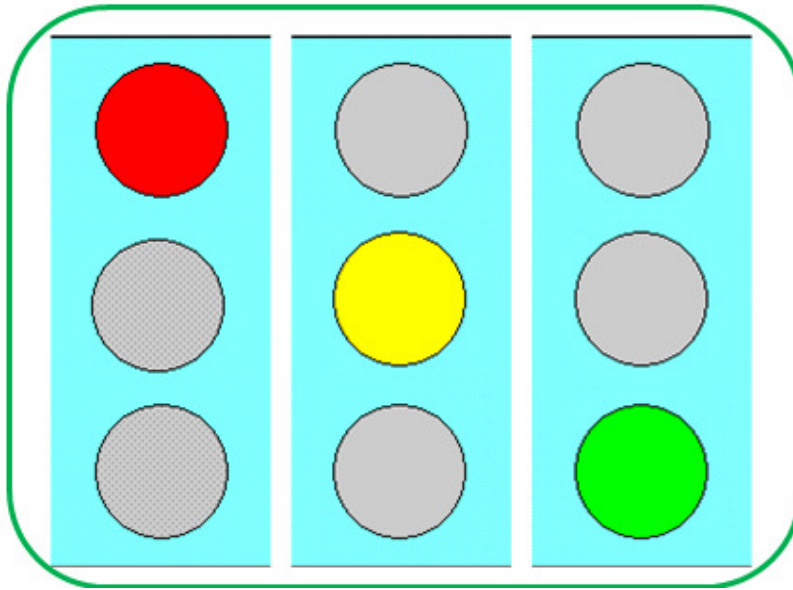
```
<br />
    <input type = "button" value = "Generate Short Story" onclick = "generateStory()"
/><br />
```

**Comment 16:** Declare the button which calls the "generateStory()" function when it is clicked.

```
</form>
</body>
</html>
```

## Application 5

Use Paint or another graphics program to create three pictures of a traffic light. One image should have the green light illuminated, one the yellow light and one the red light. Use buttons to control the animation and JavaScript code to animate the images. Save the program as TrafficLight.html



### Step-By-Step Suggested Solution

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Traffic Light</title>
  <link rel="stylesheet" href="styles.css" type="text/css" />
  <meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

  <script type="text/javascript">
    /*  */
    var stoplight = new Array(3);</pre>
</div>
<div data-bbox="147 576 850 606" data-label="Text">
<p><b>Comment 1:</b> Declare the array that will contain the three pictures of the traffic light (red, yellow and green).</p>
</div>
<div data-bbox="147 618 306 631" data-label="Text">
<pre>var curStoptlight = 0;</pre>
</div>
<div data-bbox="147 641 743 656" data-label="Text">
<p><b>Comment 2:</b> Declare the variable that will determine which light is currently flashing.</p>
</div>
<div data-bbox="147 669 226 681" data-label="Text">
<pre>var begin;</pre>
</div>
<div data-bbox="147 690 850 721" data-label="Text">
<p><b>Comment 3:</b> Declare the variable that determines which light to start with if the flashing cycle was stopped.</p>
</div>
<div data-bbox="147 733 389 799" data-label="Text">
<pre>stoplight[0] = new Image();
stoplight[0].src = "red.gif";
stoplight[1] = new Image();
stoplight[1].src = "yellow.gif";
stoplight[2] = new Image();
stoplight[2].src = "green.gif";</pre>
</div>
<div data-bbox="147 808 457 825" data-label="Text">
<p><b>Comment 3:</b> Add the pictures to the array.</p>
</div>
<div data-bbox="147 836 396 871" data-label="Text">
<pre>function cycle() {
  if (curStoptlight == 2)
    curStoptlight = 0;</pre>
</div>
<div data-bbox="147 879 850 910" data-label="Text">
<p><b>Comment 4:</b> If the picture that is currently showing is the 3<sup>rd</sup> element (2) in the array, thus the green light, it must be changed to the 1<sup>st</sup> element (0), thus the red light.</p>
</div>
<div data-bbox="849 886 886 935" data-label="Page-Footer">
<p>157</p>
</div>
```



```

else
    ++curStoplight;

```

**Comment 5:** If it is the 1<sup>st</sup> element (0) or the 2<sup>nd</sup> element (1) then the element must be incremented by 1 to show the next element in the array.

```

document.images[0].src = stoplight[curStoplight].src;

```

**Comment 6:** Print the picture in the array to the screen by referencing the document object, image source.

```

}

function startCycling() {
    if (begin)
        clearInterval(begin);

```

**Comment 7:** The clearInterval() method is used to cancel the setInterval() method before its code executes, it receives a single argument(begin), which is the variable that represents a setInterval() method call.

```

begin = setInterval("cycle()",400);

```

**Comment 8:** The setInterval() method repeatedly executes code after being called only once. The setInterval method is set to execute after 400 milliseconds (0.04 seconds) have elapsed.

```

}
/* ]]> */
</script>
</head>
<body>
<p></p>

```

**Comment 9:** Displays the first image, red light, of the traffic light on the screen.

```

<form action="" ><p><input type="button" value=" Cycle" onclick="startCycling();" />

```

**Comment 10:** Declare the button which calls the "startCycling()" function when it is clicked.

```

<input type="button" value=" Stop" onclick="clearInterval(begin);" /></p></form>

```

**Comment 11:** Declare the button which calls the "clearInterval()" method when it is clicked.

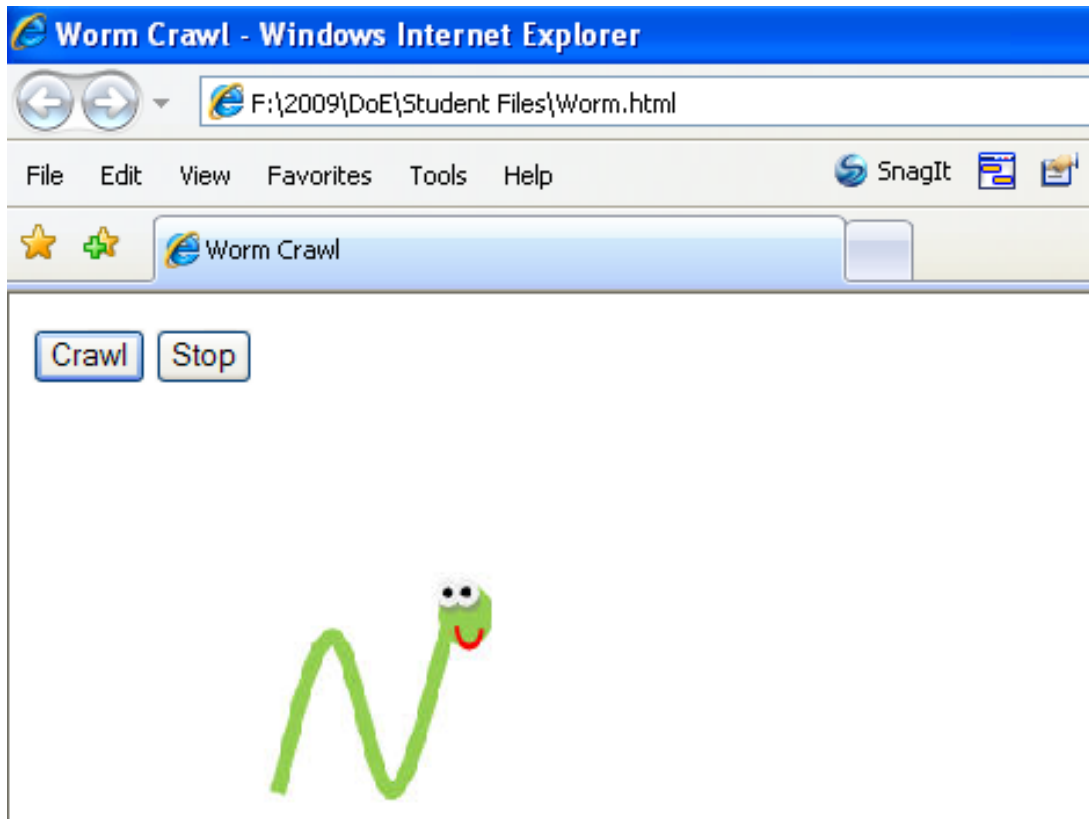
```

</body>
</html>

```

## Application 6

Use Paint or another graphics program to draw a picture of a worm. Develop a simple website that animates the image so that the worm appears to crawl across the screen. Save the program as Worm.html.



### Step-By-Step Suggested Solution

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Worm Crawl</title>

<script type="text/javascript">
<!-- HIDE FROM INCOMPATIBLE BROWSERS
var up_or_down;
```

**Comment 1:** Declare a variable that will determine if the picture of the worm must move up or down.

```
var begin;
```

**Comment 2:** Declare the variable that determines where the worm stopped in the cycle when the stop button is clicked.

```
position = 0;
```

**Comment 3:** Declare the variable that indicates the start position for the worm on the screen.

```
horizontal = new Array(10);
horizontal[0]=0;
horizontal[1]=50;
horizontal[2]=100;
horizontal[3]=150;
horizontal[4]=200;
horizontal[5]=250;
horizontal[6]=300;
horizontal[7]=350;
horizontal[8]=400;
horizontal[9]=450;
horizontal[10]=500;
```

```
horizontal[11]=550;
horizontal[12]=600;
horizontal[13]=650;
horizontal[14]=700;
horizontal[15]=750;
```

**Comment 4:** Declare an array that gives the position on the screen where the picture of the worm will next be displayed horizontally.

```
function crawl() {
    if (up_or_down == "up") {
        document.getElementById("worm").style.top = 60 + "px";
```

**Comment 4:** If the up\_or\_down variable has the value of "up" the picture of the worm must be moved vertically 60px to create the illusion of the upwards movement of the worm.

```
        up_or_down = "down";
```

**Comment 5:** Change the value of the up\_down\_variable to "down".

```
    }
    else {
        document.getElementById("worm").style.top = 120 + "px";
```

**Comment 6:** If the up\_or\_down variable has the value of "down" the picture of the worm must be moved vertically 120px to create the illusion of the downwards movement of the worm.

```
        up_or_down = "up";
```

**Comment 7:** Change the value of the up\_down\_variable to "down".

```
    }
    document.getElementById("worm").style.left = horizontal[position] + "px";
```

**Comment 8:** The horizontal placement of the worm is determined by the element in the array.

```
    ++position;
```

**Comment 9:** Add 1 to the counter of the array.

```
    if (position == 16)
        position = 0;
```

**Comment 10:** When the last element in the array has been reached (then end of the screen) the worm must be moved back to the beginning of the screen.

```
    }

    function startcrawlping() {
        if (begin)
            clearInterval(begin);
```

**Comment 11:** The clearInterval() method is used to cancel the setInterval() method before its code executes, it receives a single argument(begin), which is the variable that represents a setInterval() method call.

```
        begin = setInterval("crawl()",100);
```

**Comment 12:** The setInterval() method repeatedly executes code after being called only once. The setInterval method is set to execute after 100 milliseconds (0.01 seconds) have elapsed.

```
    }
    // STOP HIDING FROM INCOMPATIBLE BROWSERS -->
</script>
</head>
<body>
<form action="">
<p><input type="button" value="Crawl" onclick="startcrawlping();" />
```

**Comment 13:** Declare the button which calls the "startCrawling()" function when it is clicked.

```
<input type="button" name="stop" value="Stop" onclick="clearInterval(begin);" /></p></form>
```

**Comment 14:** Declare the button which calls the "clearInterval()" method.

```
<p><span id="worm" style="position:absolute; left:10px; top:120px">
</span></p>
```

**Comment 15:** Add the picture of the worm to the Web page.

```
</body>
</html>
```

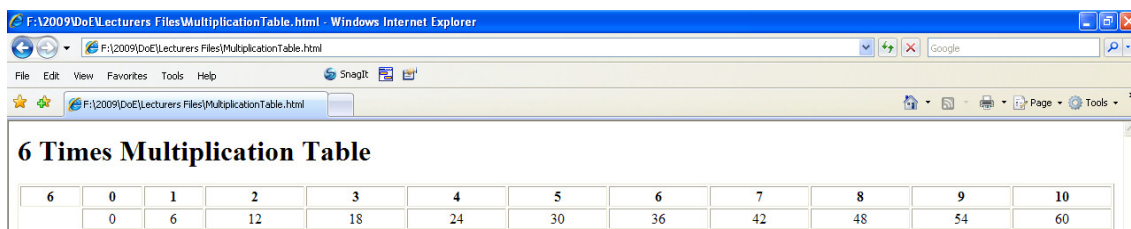
## Additional Exercises

### Introduction

This chapter will give you some additional exercises that will enable you to practice your programming skills.

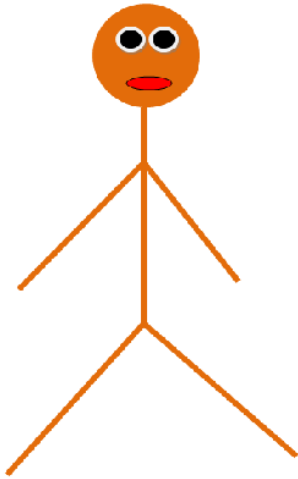
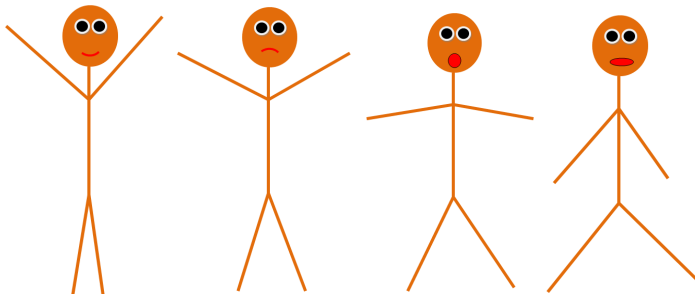
### Exercise 1

Create a web page that generates multiplication tables for the values zero through ten; depending on the number the user enters. Save the code as MultiplicationTable.



### Exercise 2

Use Paint or another graphics program to draw 4 pictures of a stick figure, the stick figures should be doing jumping jack exercises. Develop a simple website that animates the images so that the stick figure appears to be jumping up and down. Save the program as JumpingJack.html.



Jump Stop

### Exercise 3

Develop a simple Web page where a user can request a quote on soccer paraphernalia, the price should show the moment the quantity is entered. When the user clicks on the calculate button it must work out 14% value added tax and show the total amount. Save the program as OrderForm.html

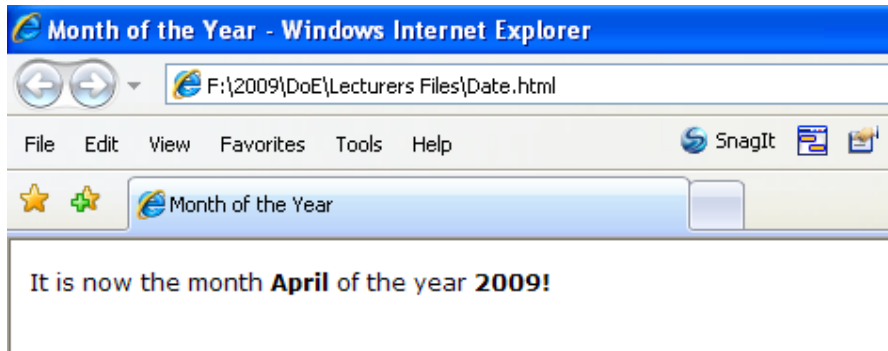
Part Number	Description	Price	Quantity	Item Total
12345	Soccer ball	R219	<input type="text" value="7"/>	<input type="text" value="1533"/>
23456	Bafana Bafana T-Shirt	R329	<input type="text" value="9"/>	<input type="text" value="2961"/>
34567	Vuvuzela	R139	<input type="text" value="3"/>	<input type="text" value="417"/>

VAT	<input type="text" value="687"/>
Total	<input type="text" value="5598"/>

Calculate

## Exercise 4

Develop a simple Web page to accept the current date from the computer. Display the month and the year in the following format "It is now the month **January** of the year **2002**" on the Screen, use the switch statement. Save the program as Date.html.



# Introduction to OOP in VB.NET



Bertie Buitendag  
© 2009

# Object Oriented Programming

## Introduction to OOP in VB.NET

Notes compiled by: Bertie Buitendag

### Introduction

OOP (Object Oriented Programming) is a programming paradigm that was introduced to the IT world in the late 1960's. It is said that the first OO language was SIMULA I and SIMULA 67. Other pioneering OOP languages were: SmallTalk (1970's), and C++ (1980's) of which the latter is still very popular today. In contrast to procedural programming which was and still is very popular today OOP requires a total new way of thinking especially for those who was and are familiar with procedural programming.

Within the OOP milieu one would often hear various terms. The next figure lists some of these concepts and terms.



Figure 1 OOP Concepts and Terms

### What is OOP

There are various definitions for OOP, just go to Google and type, *Define: OOP* and an endless list of links with definitions will appear.

### Object Orientation

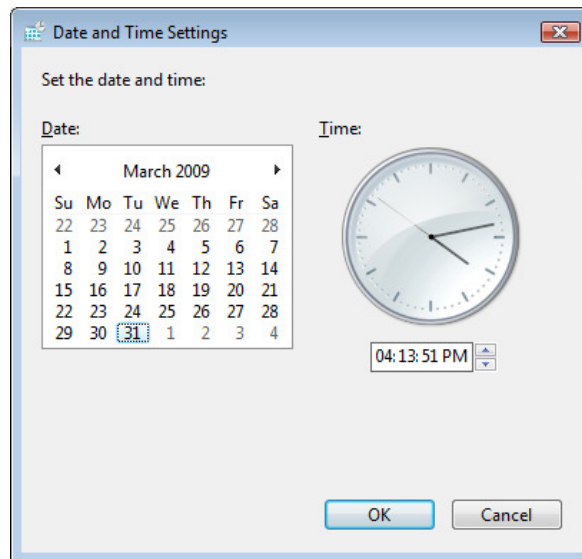
OO or Object Orientation tries to model real world objects (in a software format) in terms of its actions and its attributes.

Most visual software applications aim to model processes on real world objects e.g. performing a sale of Items at a Point of Sale terminal or Making a booking for a concert etc.

The control panel of your operating system is nothing more than an object that was modeled by software in order to help you control and manage computer. E.g. the Date and Time interface provided by the operating system allows a user to interact with the computers, date and time. Not



only does the interface allow you to set the date and time it also allows the user to read and obtain the current date and time.



**Figure 2 Date and Time Settings**

## Object Oriented Programming

Wikipedia (2008)<sup>1</sup> defines OOP by stating that: Object-oriented programming (OOP) is a programming paradigm that uses "objects" and their interactions to design applications and computer programs. Programming techniques may include features such as information hiding, data abstraction, encapsulation, modularity, polymorphism, and inheritance. It was not commonly used in mainstream software application development until the early 1990s. Many modern programming languages now support OOP.

In other words OOP aims to create software that models or simulate real world objects. OOP aims to simulate the object in terms of its attributes (the characteristics of the object) and its behavior (what the object can do).

Many programming languages which support EDP (Event Driven Programming) are based on OO. Users create computer programs which models real world scenarios. The program interface, normally a GUI is composed of various components which reacts to user events such as clicking on a button opening a form etc.

In the screenshot above which models a watch and a calendar (both objects) via a GUI the user are allowed to either obtain (read the time) or set a new time, the same applies to the date. The watch object models the time for the user while the calendar models the date.

<sup>1</sup> Wikipedia (2009) OOP Defined. [Online] Available At: [en.wikipedia.org/wiki/Object\\_oriented](http://en.wikipedia.org/wiki/Object_oriented) Accessed 2009/04/02

## OOP in terms of the concepts involved

Figure 1 depicted some of the concepts normally associated with OOP the following section would provide an overview of each of the concepts.

### ADT (Abstract Data Type)

VB.NET has a lot of common built in data types which allows a user to easily define a variable to represent and store a value for a certain item or descriptor, e.g. Integer to declare variables to store whole numbers, Decimal for floating point values and Boolean for flags.

Sometimes however more than one simple descriptor are needed to define/declare an object.

A concept such as a rectangle needs to be defined in terms of its length and its width both of which could in turn be stored as decimals.

So Simple value variables are normally described by a single descriptor:

e.g.   Length  
          Colour  
          Distance  
          Name

Complex objects are described by using more than one simple attribute.

e.g.   Motorcar  
          Engine  
          Doors  
          Make  
          Model  
          Year  
          Transmission type  
          etc.

A Motorcar is an abstract concept since more than one simple descriptor is needed to describe the car.

ADT (Abstract Data type) can therefore be seen as: a user- defined data type consisting of (being described by) several simple data types.

E.g.   Brick (length, width, height)  
          Date (year, month, day)  
          Time (hours, minutes, seconds)

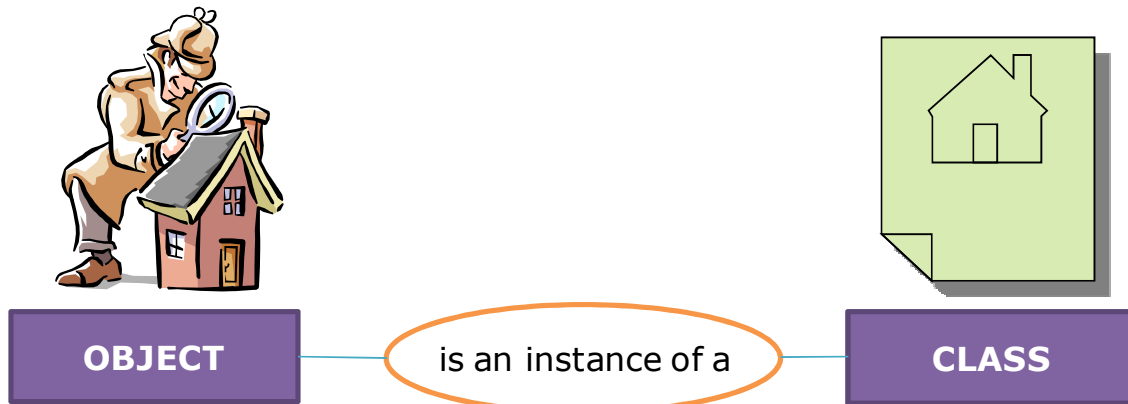
An Abstract Data Type is used to declare objects that cannot be defined by using concrete / simple data types.

### Classes

Classes are the design plans for building and describing an object. A class describes an object in terms of what the object can do (its behavior) as well as what the object is (its attributes).

A Building plan is an example of a class, with one building plan a builder will be able to build and construct many buildings, all with the same attributes.




Many textbooks state that classes are the blueprints for defining objects, in other words a class allows someone or something to construct an object. Robots are programmed with "classes" to construct cars, while a watch factory's robots construct watches from a class. A carpenter may build a wooden box from a plan which could be seen as a class.



The design specification as set aside in the class ensures that the object created conforms to the plan. E.g. if a rectangle class are to describe a rectangle in terms of its length and its width, the rectangle class would ensure that every subsequent rectangle has a length and a width.

## Objects

Objects are real world things which have been created at some or other stage; objects may be tangible or non-tangible. Most real world objects are tangible while software objects are predominantly non-tangible.

Watch	Car	Button
A Watch is an example of an object; the watch has various attributes; color strap-size, shape, time, buttons etc.	A Car is an example of an object it is described in terms of its various attributes (the things that make the car what it is) and its behavior .(what the car can do)	A Button is an example of a software object; it is described in terms of its various properties and behavior.
		

Objects are normally described in terms of attributes and behavior.

The **attributes** of an object describe what the object is in the example of the watch a watch could be described in terms of:

- FaceShape
- Colour
- Straptype
- NrOfButtons

WatchShape  
Time  
DigitalOrAnalog  
Etc

The behavior of the watch describes what the watch can do:

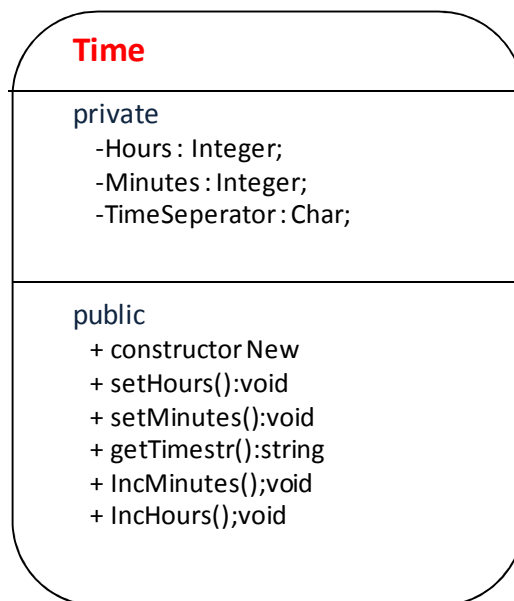
Set the Hours  
Set the Minutes  
Read the Time  
Set the Alarm  
Sound the Alarm etc

The behavior of the watch describes the functionality of the object.

### ***UML Class Diagrams***

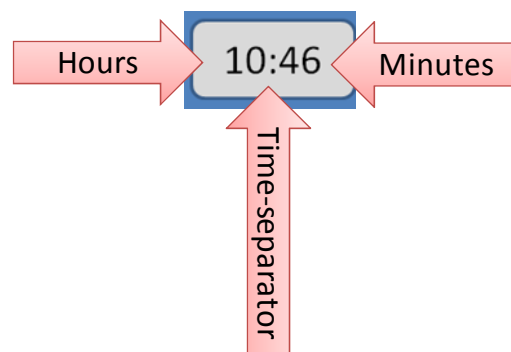
The Unified Modeling Language uses class diagrams to allow programmers to model classes diagrammatically.

The following UML class diagram represents a Time class:



The Time class has 3 attributes as indicated by the diagram. The attributes are:

Hours  
Minutes  
TimeSeperator



Each of these attributes are represented and stored by using a concrete data type

Hours is stored as an *Integer*

Minutes is stored as an *Integer*

TimeSeperator is stored as a *Char*

These three attributes are used to represent the "data" of the time object.

**TAKE NOTE:** The UML class diagram only represents the TIME object that is part of the watch.

Within the watch the actual time is hidden away from the user, and the time is only displayed as a set of characters (a string) on the screen of the digital watch. The user cannot "touch" the actual time inside of the watch but it watch allows the user to change the time by using buttons on the watch.

Objects normally have an interface that allows a user to communicate with the values and the attributes of the object. The interface of the watch allows the user to:

Set the hours

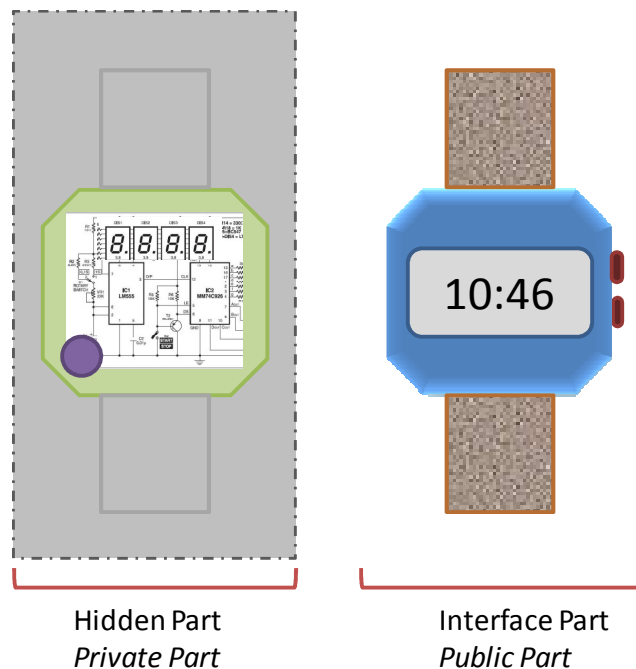
Set the minutes

Increment the hours with 1

Increment the minutes with 1

Read the time as a string

Objects are commonly described in two parts the visible interface part which the user is allowed to interact with and the non visible hidden part. The visible parts normally interact with the non visible parts.



The attributes indicated with a preceding – in the class diagram is indicated to be private or hidden from the user. The methods or attributes preceded with a + indicates that it is visible or public to the user.

A user of an object only has access to the public's of the object.

The time class is therefore modeled in terms of its three attributes as well as its behavior what it can do. A class diagram indicates to the programmer which of the attributes are visible to the user and which of the attributes are hidden away.

Most methods of an object are normally accessible to the user of the objects. Methods are functions and subroutines used to code some functionality pertaining to the object.

The concept of hiding some of the functionality and attributes of an object is called encapsulation. A watch is encapsulated in order to hide and protect its inner parts.

### Attributes

We have seen that attributes are the descriptors of a class which is to represent an object. Attributes are normally defined using built in data types. An attribute that is used to describe a class may also be an object by itself.

E.g: a Watch may contain a time object; the same time object may also be used in a DVD player, as well as a microwave. The reusability of objects in programming is what makes OOP such a powerful software development concept. A programmer does not need to write the code for the Time class instead the programmer may decide to incorporate a current available class into his solution. EDP utilizes this concept fully, all components placed on a form is actually created from a classes.

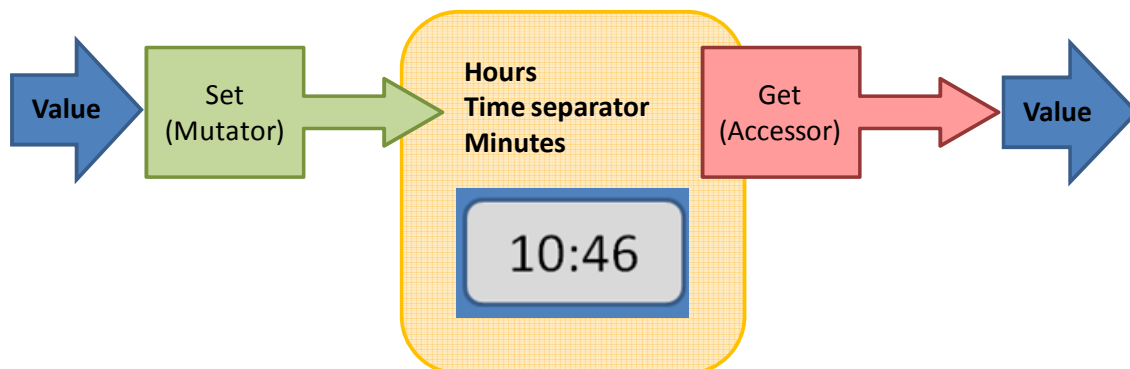
Using one object as part of a new object is called composition. Another good example is that of a car a car comprises of various other objects. E.g. motor, gearbox, seats, wheels, etc.

### Methods

The methods of the class represent what the class can do. Methods in program code are presented as subroutines which may be functions or procedures. There are various types of methods, some methods are used to obtain a value from a user and to set the value of an attribute. Other methods are used to retrieve the value of an attribute and to send it to the user, while other methods are used to perform certain functionality of the object.

**TAKE NOTE:** It is advisable at this stage to do a thorough review of subroutines, functions and procedures.

- Accessor methods are used to retrieve the value of an attribute. Accessor methods are sometimes preceded with the word get. E.g getTime
- Mutator methods are used to assign a value from the user to an attribute. Mutator methods are sometimes preceded with the word set. E.g. setHours



- Constructors are methods that are used to set the default value for objects when they are created. When a bunch of watches are switched on that is of the same type and same manufacturer the watch would have been preprogrammed with a default time.

In VB.NET methods are defined to classes by means of subroutines and functions.

## Properties

Properties are special attributes of a class that is usually only read and assigned a value. The text attribute of a Button or a Label for example in VB.NET is considered a property. The user may set the value to the property or get the value for a calculation.

## Encapsulation

Encapsulation is the term that is used to describe how an object's attributes are hidden. Information hiding is one of the benefits of OOP where the internal data is hidden from the actual functioning of the object. Encapsulation hides the internal attributes of a class to protect the data and to separate the class logic. Encapsulation is the method of dividing and defining an object into a visible and non-visible part covering up the data and the attributes as well as the function behaviors into a class.

Just as a capsule hides and protects the inner parts of the capsule just so does encapsulation.

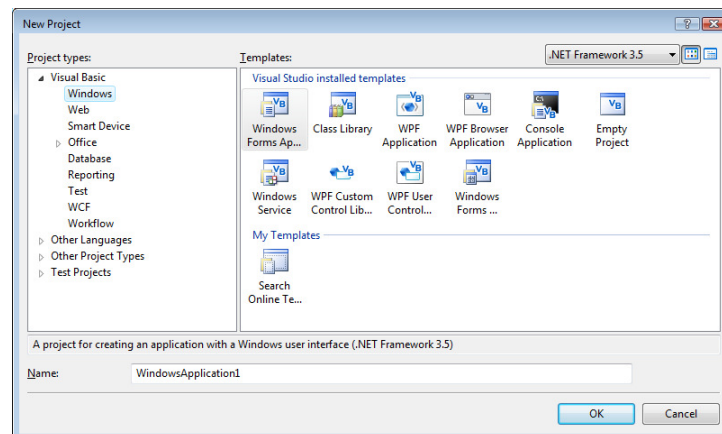
A Microwave object is encapsulated into two parts (a private inner part of which you don't want to interfere with) and an outside interface for the user to interact with. The same applies for the watch object.

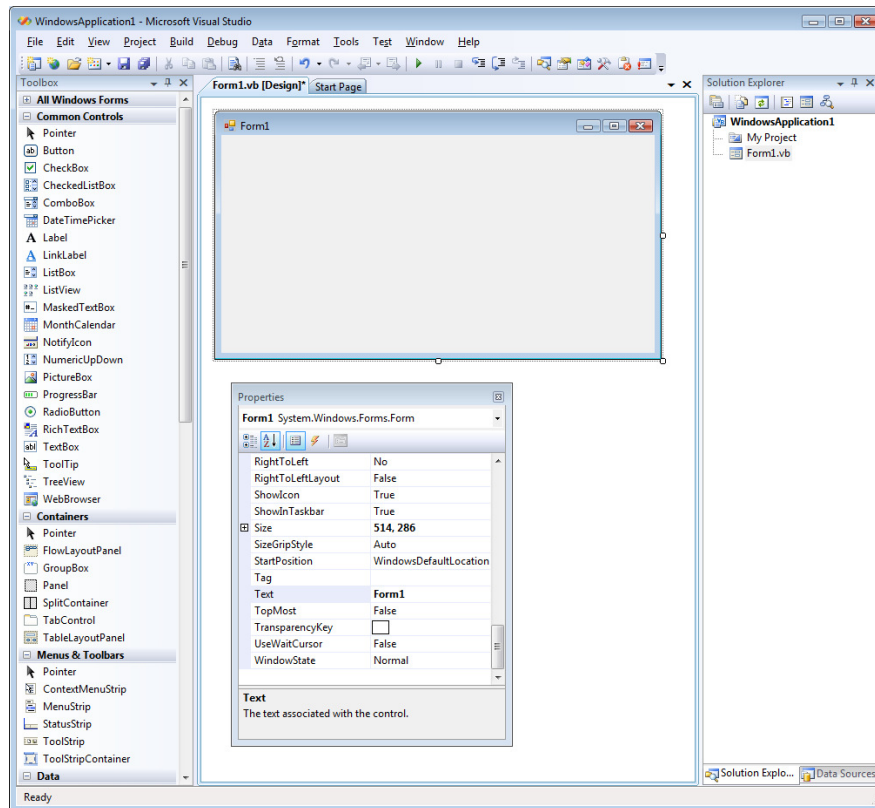
## Instantiation

When an object is made it is said that the object is instantiated. We have learned that an object is an instance of a class, so making the object is instantiating it.

## VB.NET as an OOP Language

VB.NET is regarded as a true OOP language since it allows programmers to create and use classes, inherits from classes, and allows for polymorphism. When a new Visual Studio VB.NET project are to be started the developer are supplied with options to state which type of programming project he/she wants to create. When selecting a Windows Form Application such as indicated in the figure below the IDE would automatically create the first Windows form for the user on which the programmer may add components to model a solution to a problem. The form which is created is an object. The properties window lists all the properties of the newly created form object.





## Working with a VB.NET GUI Application

With all prior applications that you may have developed, you would have implemented OOP without you even knowing it. If you study the code for a simple VB GUI application you would notice that you have added objects and used attributes in your form. You have even created event handlers and passed messages to and from created objects on your form.

Study the following GUI (which you may have encountered in a prior course) and the solution code.



## Keep Fit Gym – Solution code

```
Option Explicit On
Option Strict On
Public Class frmGym

    Private AppDate As Date

    Private Sub btnEntranceFee_Click(. . .) Handles btnEntranceFee.Click
        Dim strName, strMessage As String
        Dim decEntranceFee As Decimal
        Dim decDiscount As Decimal
        strName = txtName.Text
        If chkMember.Checked Then
            decEntranceFee = 8D
        Else
            decEntranceFee = 12D
        End If
        If chkStudent.Checked Then
            decDiscount = decEntranceFee * 0.1D
        Else
            decDiscount = 0
        End If
        decEntranceFee = decEntranceFee - decDiscount
        strMessage = " you must pay R" & decEntranceFee.ToString("N2") & " entrance fee"
        lblOutput.Text = strName & strMessage
    End Sub

    Private Sub btnClear_Click(. . .) Handles btnClear.Click
        txtName.Text = ""
        chkStudent.Checked = False
        chkMember.Checked = False
        lblOutput.Text = ""
        txtName.Focus()
    End Sub

    Private Sub btnExit_Click(. . .) Handles btnExit.Click
        Close()
    End Sub

    Private Sub frmGym_Load(. . .) Handles MyBase.Load
        AppDate = New Date
        lblDate.Text = ""
        AppDate = Now
        lblDate.Text = AppDate.Day.ToString + "/" + AppDate.Month.ToString + "/" +
        AppDate.Year.ToString
    End Sub
End Class
```

## Sample run screen

Entrance Fee

KEEP FIT GYM

Please provide your name: Koos

Please select if applicable

☒ Student ☐ Member of the Gym

Koos you must pay R10.80 entrance fee

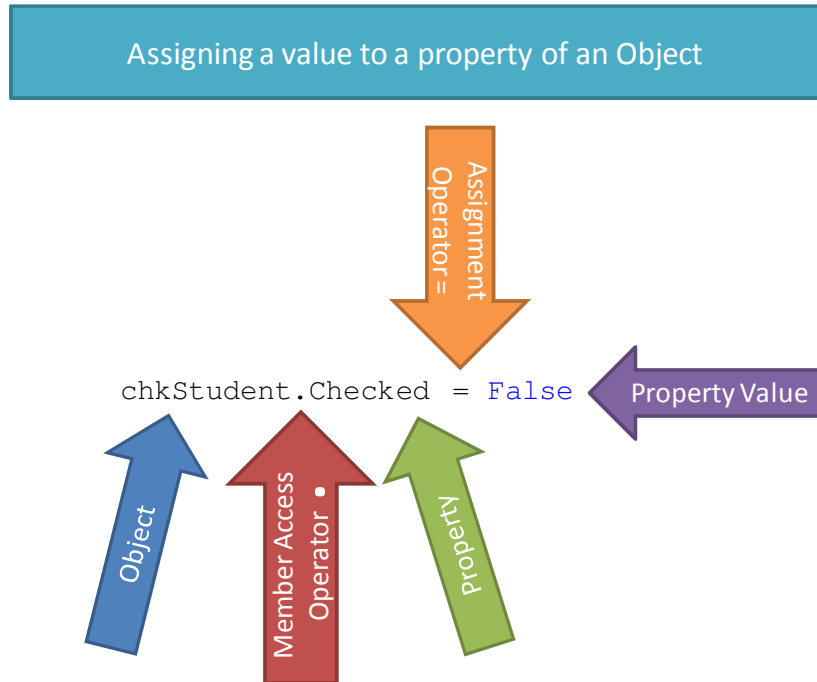
Entrance Fee Clear Exit

1/5/2009

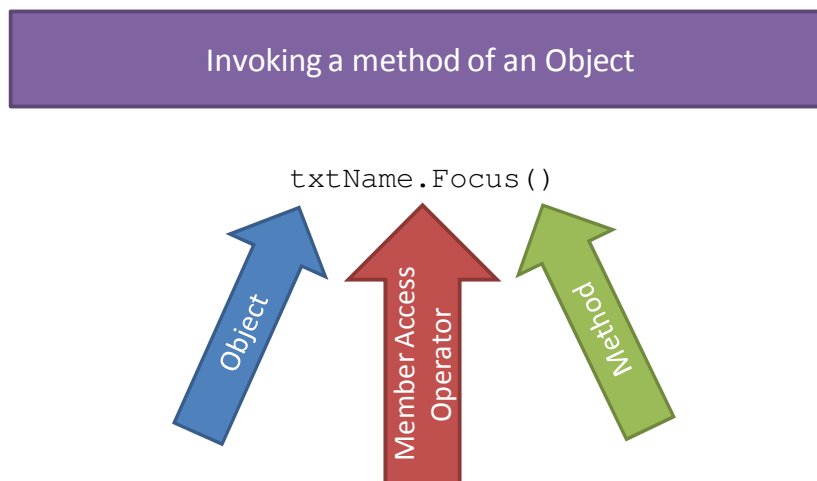
## NOTES on the program

This program implements a Data object, the date is displayed on the Date label named lblDate. Various objects are implemented e.g. Checkboxes, labels etc. In fact all the components placed in the form as well as the form itself may be seen as objects.

The checked property of the Checkboxes is implemented to determine the entrancefee.



The Focus method of the Name textbox is used to place the Cursorfocus in the textbox.



## Creating and using a class in a VB.NET application

The Visual Studio.NET IDE is fully developed using OOP principles. The IDE allows a user to develop amongst others solutions with programmer defined classes. Read through the following example and follow the steps.

### Example 1 – Implementing the Time Class in a GUI Application

Develop a class named, MTime. The class comprises of two attributes Hours and Minutes and a property Timeseparator. The timeseparator property is used to set the character to be used for display purposes of the time, it should default to a colon :.

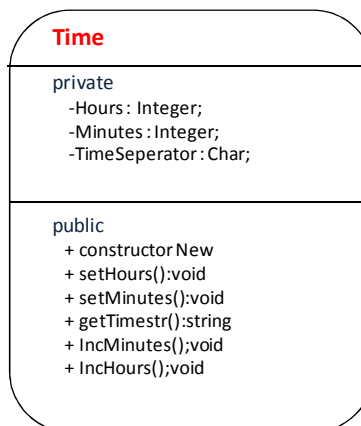
The class should include methods to allow the user to set the hours and the minutes and to retrieve the time as a string in the format HH:MM. The time should be padded with leading 0's where applicable.

Five minutes past eight should be returned as 08:05. All newly created Time objects should default to 12:00.

The class should also include a method that will increment the hours with one hour and a method that will increment the minutes with one minute. These two methods should ensure that the data manipulated stay valid. When invalid data is sent to the SetHours and SetMinutes methods respectively the data values of the corresponding attributes should not be changed.

Define your class in a separate unit. Develop an application to test your class.

*Step 1: Plan your class, using a UML class diagram.*



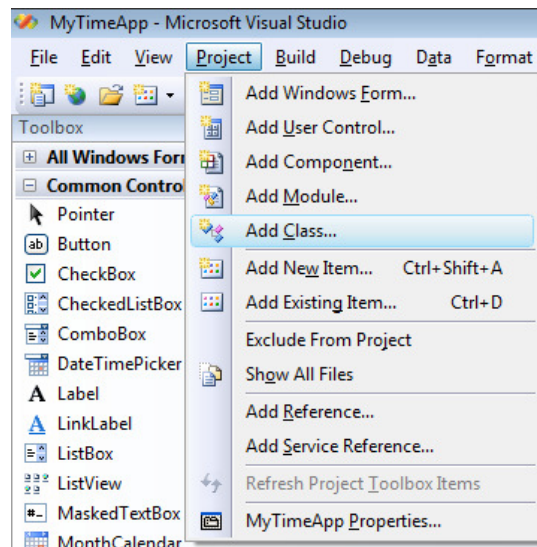
*Step 2: Design and plan the functionality of each method*

Design and plan the logic and functioning of each of the methods.

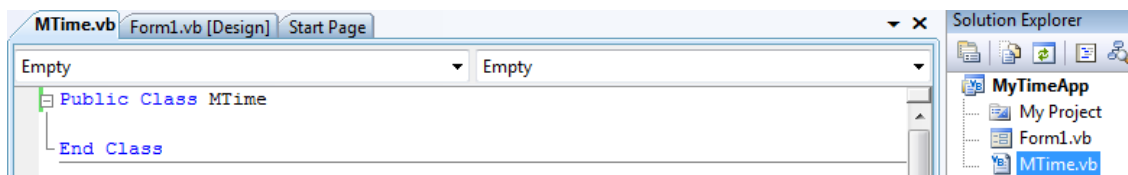
A function / planning chart could be implemented at this stage. The developer needs to plan how each of the methods of the class are to be implemented with regard to the parameters and their types etc.

*Step 3: Start VB and create a new Project and create a new windows form application named: MyTimeApp and save the application.*

*Step 4: Click on Project and Add Class*



Step 5: Name your class MTime and click on the Add button. (Click on Save All.) The IDE will automatically generate the following code:



Take Note: Your class now forms part of your solution.

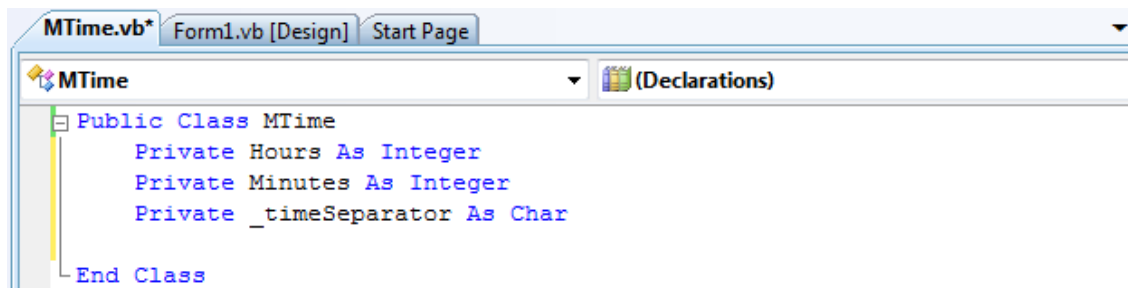
Step 6: Define your class.

Step 6.1: Add the private attributes

Step 6.2: Add the property attributes

Take note: An underscore \_ is added as a prefix to properties

Code should appear as follow:



Step 7: Write the method code

Step 7.1: Add the property code

The design of the class indicated that the timeSeparator attribute of the class should be written as a property. As you can recall a property is a specialized attribute which will allow the user to automatically assign or read a value from the object.

Add the following line of code: `Public Property TimeSeparator() As Char`

When the developer presses enter after typing the line the following code is automatically added by the compiler.

```
Public Property TimeSeparator() As Char
    Get

    End Get
    Set(ByVal value As Char)

    End Set
End Property
```

Take note that the property allows the user to add the code to return the value of the property and to assign the value of the property.

Add the following code to the Get section:

```
Get
    Return _timeSeparator
End Get
```

The Get section of the property indicates what value should be sent back or returned for the property. Take note that the name of the Property is TimeSeparator while the name of the private attribute is \_timeSeparator.

Add the following code to the Set section:

```
Set(ByVal value As Char)
    _timeSeparator = value
End Set
```

The Set part indicates that a by value parameter named value is received, the value stored in the value parameter is assigned to the private \_timeSeparator. Remember that only the Public methods are allowed to access the private methods or attributes. Our property must therefore be defined as Public as the user would have to have access to the property. Properties are by default and design not supposed to be hidden to the user.

Step 7.2 Add the code for the constructor

The constructor is a special method that is invoked and called only once, when the object is made/created or instantiated. As said previously the constructor is normally used to set default values to the attributes of the object that is created. In our case the time should be set to 12:00 for all new time objects.

VB.NET has a special name to be used for the constructor which is called new. The new method indicates that the method is to be invoked when a new object is to be created.

```
Public Sub New()
    Hours = 12
    Minutes = 0
    _timeSeparator = ":"
End Sub
```

### Step 7.3: Add the code for each of the methods

Add the following Public Subroutines to your class; these two subroutines are used to set the attribute values of the Hours and the Minutes respectively. Take Note the set methods receives and input parameter by value, this value is assigned to the private attribute in each case only if the values are valid. Invalid data is not set.

```
Public Sub setHours(ByVal HH As Integer)
    If (HH < 24) Then
        Hours = HH
    End If
End Sub

Public Sub setMinutes(ByVal MM As Integer)
    If (MM < 260) Then
        Minutes = MM
    End If
End Sub
```

Add the following Public Functions to your class: these two functions are used to return the values of the Hours and the Minute attributes respectively. Public functions are implemented for this task, the return types of each of these an Integer is used as the Hours and Minutes attributes are of the same type.

```
Public Function getMinutes() As Integer
    Return Minutes
End Function

Public Function getHours() As Integer
    Return Hours
End Function
```

The IncHours method should increase the value of the hours with one hour. A public subroutine is used to increment the Hours attribute with one. In order to perform a valid operation the hours may not exceed 23, in the case of the time being 23:25 and the IncHour method is invoked then the time should be 00:25. Add the following subroutine as part of your class.

```
Public Sub IncHours()
    If (Hours = 23) Then
        Hours = 0
    Else
        Hours = Hours + 1
    End If
End Sub
```

The IncMinutes subroutine performs the same function but to increment the Minutes attribute. In this case the minutes may not exceed 59. If the minutes exceed 59 then one Hour should be added to the Hours attribute E.g. 10:59 should be 11:00. The IncMinutes method calls the IncHours method when the minutes are equal to 59 and sets the minutes back to 0. Add the following subroutine to your class. **Take Note:** Because the IncHours method is part of your class other methods are allowed to call and use it. Methods in object may be called and used in other methods of the same class.

```
Public Sub IncMin()
    If (Minutes = 59) Then
        Minutes = 0
        IncHours()
    Else
        Minutes = Minutes + 1
    End If
End Sub
```

Many classes in VB.NET have a method called *toString* this method is typically used to return numerical values as a string. Our getTimeStr method is to perform such an operation. If the current attributes hold the values (Hours = 8) and (Minutes = 9) then the time is 9 past 8. This time should then be returned as 08:09 in a string format. The method should first determine if either the Hours or

the Minutes are less than 10 in such a case a 0 should be prefixed to the Hours or the Minutes as the case may be.

Study the following code of the getTimeStr method and add it to your class. Take note that the TimeSeparator attribute is appended between the Hours and the Minutes. For this method an function scope variable named TempTimeStr is used, which is only available for use within the method.

```
Public Function getTimeStr() As String
    Dim TempTimeStr As String
    If (Hours < 10) Then
        TempTimeStr = "0" + Hours.ToString
    Else
        TempTimeStr = Hours.ToString
    End If

    TempTimeStr = TempTimeStr + TimeSeparator

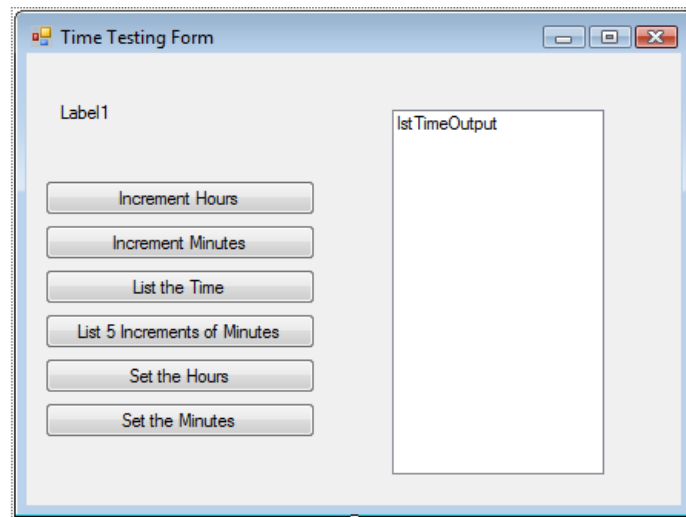
    If (Minutes < 10) Then
        TempTimeStr = TempTimeStr + "0" + Minutes.ToString
    Else
        TempTimeStr = TempTimeStr + Minutes.ToString
    End If

    Return TempTimeStr
End Function
```

TAKE NOTE: If you have followed the following steps, now would be a good time to save all your files.

#### Step 7: Develop the testing application

Our class is now completed, and at this stage we may continue to develop the driver / testing program. To test our class the following GUI was developed.



The functioning of the application is envisioned as follows: One time object is to be instantiated when at the same time that the form is loaded / created.

The Increment Hours and Increment Minutes button invokes / calls the corresponding methods.

The List the Time button lists the time as a string in the lstTimeOutput listbox.

When the user clicks on the Set the Hours or Set the Minutes button's respectively then an InputBox is used to obtain the new minutes and hours from the user and the corresponding method is called.

Every time that one of the buttons is invoked the current time is displayed on the Label.

Step 8: Define the object / declare an object

VB.NET provides a programmer with two ways of instantiating a variable or an object it is important to state that when dealing with objects the following two steps must be complied to:

- Firstly: The compiler must be notified that an object of a certain type is to be included as part of the solution. In other words the object needs to be declared for the same purposes as each variable are declared within a solution.
- Secondly the object declared must be created.

It is possible in VB.NET to perform these two steps in one line of code. For the purposes of this example the declaration and creation of the object would be illustrated in two steps.

```
Public Class frmTimeTester
```

```
Private MyTime As MTime
```

Declares the Object MyTime from the class MTime. NOTE the use of the keyword AS

```
Private Sub frmTimeTester_Load(. . .) Handles MyBase.Load
```

```
MyTime = New MTime()
```

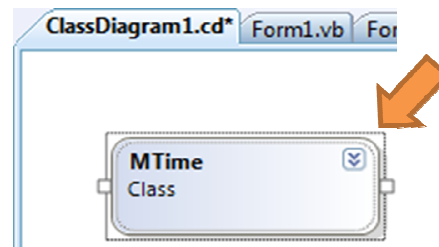
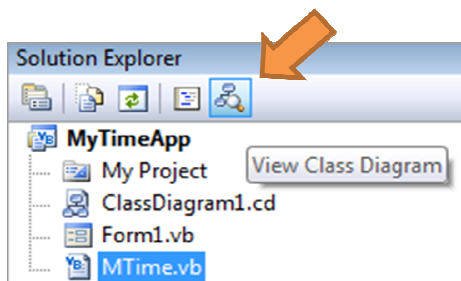
This line of code Instantiates the object. It invokes the default constructor and creates the object in the memory.

```
End Sub  
End Class
```

As you probably have noticed the MyTime object forms part of the frmTimeTester class; this means that the MyTime object is defined as an attribute of the form class. Objects may be included as part of other objects.

TAKE NOTE: An object may only be used once it has been defined and instantiated.

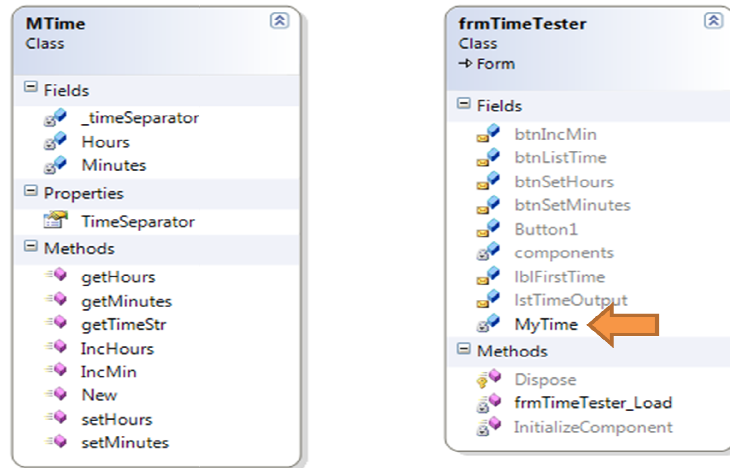
At this stage it is advisable to study the solution explorer window. You will notice that your class file is part of your solution. If you select the MTime.vb file and click on the View Class diagram button, the IDE will insert a new tab in the Code Editor window.



Clicking on the expand button in the Class Diagram would display a complete diagram indicating the various attributes, properties and methods of your class.

As your form (frmTimeTester) is also an object instantiated from the class Form it is also possible to show the class diagram for your form.





TAKE NOTE: The MyTime user defined object is one of the attributes (fields) of the frmTimeTester class.

Step 9: Implement your object as part of your program solution.

We have declared an MyTime object as part of our program solution. The MyTime object is said to have class scope, which means that the programmer may use the object after it has been instantiated anywhere within the methods of the Form class. The MyTime object gets created the same time that the Form object is displayed.

Create the correct event handlers and add the following code as part of your form class.

```
Public Class frmTimeTester

    Private MyTime As MTime

    Private Sub frmTimeTester_Load(. . .) Handles MyBase.Load
        MyTime = New MTime()
        lblFirstTime.Text = MyTime.getTimeStr()
        lstTimeOutput.Items.Clear()
    End Sub

    Private Sub btnIncHours_Click(. . .) Handles btnIncHours.Click
        MyTime.IncHours()
        lblFirstTime.Text = MyTime.getTimeStr()
    End Sub

    Private Sub btnIncMin_Click(. . .) Handles btnIncMin.Click
        MyTime.IncMin()
        lblFirstTime.Text = MyTime.getTimeStr()
    End Sub

    Private Sub btnListTime_Click(. . .) Handles btnListTime.Click
        lstTimeOutput.Items.Add(MyTime.getTimeStr())
    End Sub

    Private Sub btnSetHours_Click(. . .) Handles btnSetHours.Click
        Dim sHrs As String
        sHrs = InputBox("Enter the Hours", "Hours", "")
        MyTime.setHours(Integer.Parse(sHrs))
        lblFirstTime.Text = MyTime.getTimeStr()
    End Sub

    Private Sub btnSetMinutes_Click(. . .) Handles btnSetMinutes.Click
        Dim sMins As String
        sMins = InputBox("Enter the Minutes", "Minutes", "")
        MyTime.setMinutes(Integer.Parse(sMins))
        lblFirstTime.Text = MyTime.getTimeStr()
    End Sub

End Class
```

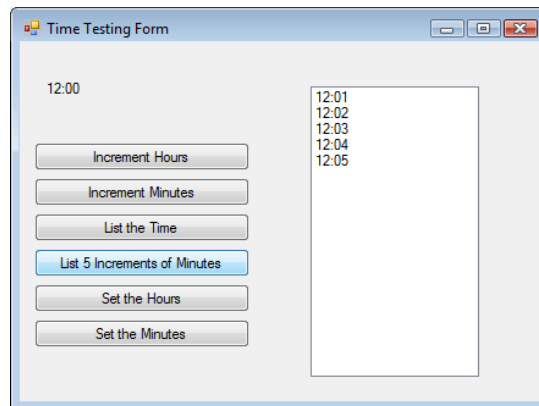
```

Private Sub btnListFiveInc_Click (. . .) Handles btnListFiveInc.Click
    lstTimeOutput.Items.Clear()
    For K As Integer = 1 To 5
        MyTime.IncMin()
        lstTimeOutput.Items.Add(MyTime.getTimeStr)
    Next K
End Sub
End Class

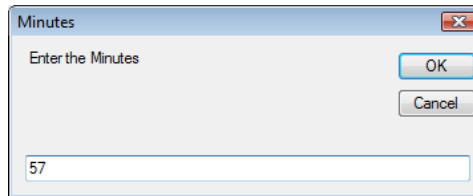
```

Step 10: Run and test your solution.

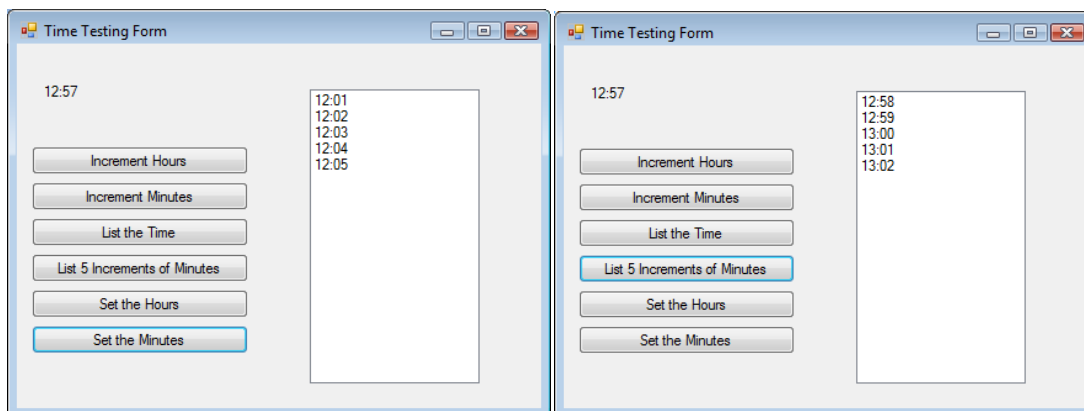
The following screenshot is displayed if the user runs the application and clicked on the [List 5 Increments of Minutes] button.



If the user clicks on the Set the minutes button the following inputbox will allow the user to enter a new value for the minutes.



The new time is updated on the label and when the user clicks on the [List 5 Increments of Minutes] button again the time is correctly incremented.



## Example 2 – Phone class & Overloaded constructor

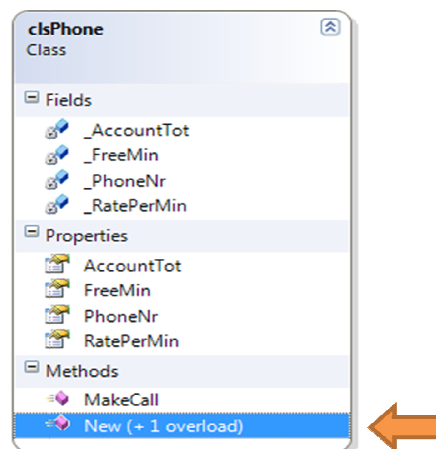
Develop a class named clsPhone. The class is to simulate the operations on a cellphone. The class should be designed to incorporate four private attributes (fields). The \_AccountTot field should store the value of the current call account. The \_FreeMin field should store the number of free minutes. The \_PhoneNr field should store the phone number as a string and the \_RatePerMinute field should store the rate per minute.

Create equivalent properties to set and get the values of each of the fields.

The class should contain two constructors a default constructor and a parameterized constructor. The default constructor should set the AccountTotal to 0, and allocate 15 free minutes to the object. The default rate per minute should be set to R2.50 and the PhoneNr should be set to an empty string.

Add a method called MakeCall that will receive the duration of the call in minutes. The method should include logic to calculate and update the AccountTotal attribute. Take note that the caller is only charged once all his / her free minutes have been used. It may thus happen that a user may be charged for a part of a call. E.g. The user has 5 free minutes left and a call is made for 7 minutes then only 2 minutes are charged and added to the AccountTot.

Define your class in a separate unit. Develop an application to test your class.



The clsPhone class. The following class has been developed by the developer which conforms to steps 1 to 6 of the previous example. Observe the declaration of the four private fields and the definition of the four corresponding properties.

```
Public Class clsPhone
    Private _PhoneNr As String
    Private _FreeMin As Integer
    Private _RatePerMin As Decimal
    Private _AccountTot As Decimal

    Public Property PhoneNr() As String
        Get
            Return _PhoneNr
        End Get
        Set(ByVal value As String)
            _PhoneNr = value
        End Set
    End Property

    Public Property FreeMin() As Integer
        Get
            Return _FreeMin
        End Get
        Set(ByVal value As Integer)
```

```

        _FreeMin = value
    End Set
End Property
Public Property RatePerMin() As Integer
    Get
        Return _RatePerMin
    End Get
    Set(ByVal value As Integer)
        _RatePerMin = value
    End Set
End Property
Public Property AccountTot() As Integer
    Get
        Return _AccountTot
    End Get
    Set(ByVal value As Integer)
        _AccountTot = value
    End Set
End Property

```

### **Method Overloading.**

Most OOP languages allow a developer to create methods with the same name provided that the methods signature is different. The signature of a method is determined by the various types of the parameters that is sent. Creating methods with the same name but with different parameters is called method overloading. A good example of where method overloading is often used is in the constructor of a class.

Study the following two overloaded constructors for the clsPhone class.

```

Public Sub New()
    _AccountTot = 0
    _FreeMin = 15
    _PhoneNr = String.Empty
    _RatePerMin = 2.5
End Sub

Public Sub New(ByVal pNr As String, ByVal fMin As Integer, ByVal rate As Decimal, _
ByVal accTot As Decimal)
    PhoneNr = pNr
    FreeMin = fMin
    RatePerMin = rate
    AccountTot = accTot
End Sub

```

The first constructor is called the default constructor. This constructor will be called when the user instantiates a MyPhone Object by writing the following line of code:

```
MyPhone = New clsPhone()
```

The second constructor will allow the user to pass default values for assignment to the attributes by the implementation of parameters hence the second constructor is referred to as a parameterized constructor. The values of passed parameters are assigned to the attributes of the object. In such a case the developer may create a MyPhone object with default values in the following way:

```
MyPhone = New clsPhone("0832214321", 25, 1.85, 0)
```

The compiler would know which constructor to call because the latter one passes values via parameters.

The class inspector indicates how the various methods are built and which parameters they implement.

Class Details - clsPhone		
Name	Type	Modifier
<b>Methods</b>		
MakeCall		Public
( Duration	Integer	ByVal
) <add parameter>		
New		Public
() <add parameter>		
New		Public
( pNr	String	ByVal
, fMin	Integer	ByVal
, rate	Decimal	ByVal
, accTot	Decimal	ByVal
) <add parameter>		

### The MakeCall method

The following code is used to perform the functional logic for the MakeCall method as described in the scenario. Take note that the method uses the properties of the class instead of the private attributes. The programmer could also have implemented the private attributes.

```
Public Sub MakeCall(ByVal Duration As Integer)
    Dim LeftOver As Integer

    If FreeMin > 0 Then
        LeftOver = FreeMin - Duration
        If LeftOver >= 0 Then
            FreeMin = FreeMin - Duration
        Else
            Duration = Duration - FreeMin
            AccountTot = AccountTot + (RatePerMin * Duration)
            FreeMin = 0
        End If
    Else
        AccountTot = AccountTot + (RatePerMin * Duration)
    End If
End Sub
```

### The implementation application

The following form was designed to test the implementation of the class.

The create phone button is used to create a Phone object.  
The make call button is used to allow the user to enter the duration of the call made via an Inputbox and the display account button is used to display the amount due on the corresponding label.  
The user enters the phone number of the phone object that was created.

### ***The program code:***

```
Public Class frmPhoneClassImpForm

    Dim MyPhoneObj As clsPhone

    Private Sub btnCreatePhone_Click(. . .) Handles btnCreatePhone.Click
        MyPhoneObj = New clsPhone (txtPhoneNr.Text, 15, 2.5, 0)
        lblFreeMins.Text = "FreeMins " + MyPhoneObj.FreeMin.ToString
        MessageBox.Show(MyPhoneObj.PhoneNr + " - Phone was succesfully created", "IenFo", _
        MessageBoxButtons.OK, MessageBoxIcon.Information)
        btnDisplayAccount.Enabled = True
        btnMakeCall.Enabled = True
    End Sub

    Private Sub frmPhoneClassImpForm_Load (. . .) Handles MyBase.Load
        btnDisplayAccount.Enabled = False
        btnMakeCall.Enabled = False
    End Sub

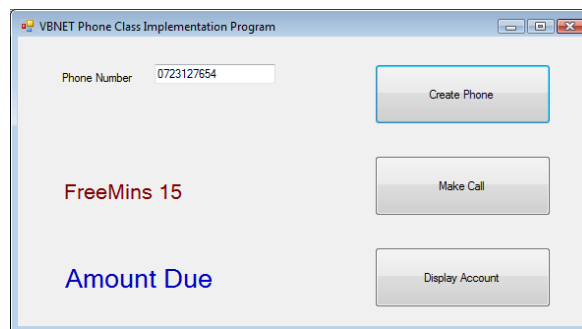
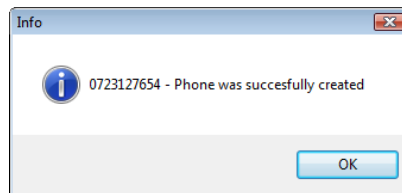
    Private Sub btnMakeCall_Click(. . .) Handles btnMakeCall.Click
        Dim CallDur As Integer
        Dim sCallDur As String

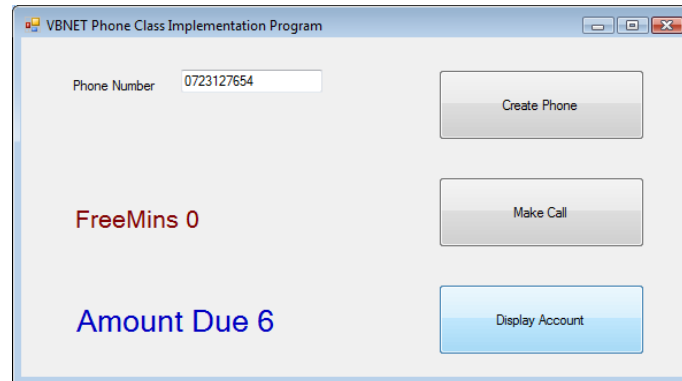
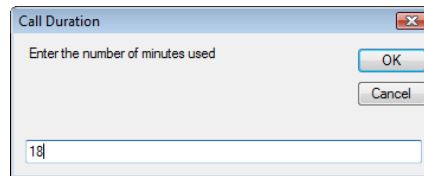
        sCallDur = InputBox("Enter the number of minutes used", "Call Duration")
        CallDur = Convert.ToInt32(sCallDur)
        MyPhoneObj.MakeCall(CallDur)
        lblFreeMins.Text = "FreeMins " + MyPhoneObj.FreeMin.ToString
    End Sub

    Private Sub btnDisplayAccount_Click (. . .) Handles btnDisplayAccount.Click
        lblAmountDue.Text = "Amount Due " + MyPhoneObj.AccountTot.ToString
    End Sub
End Class
```

### ***Notes on the code:***

When a user clicks on the create new Phone button the object is created by invoking a parameterized constructor. This constructor copies the values of the phonenumber from the Phonenumber textbox, the free minutes is set to 15, the rate per minute to R2.50 and the account total to 0.





## Exercises

- 1) Create a class called `clsLottoNrs` the class should include one integer array attribute named `LotArrValues` that could store the values of 6 Lotto numbers from 1 to 149. The method `Draw` should fill the array with 6 random values from 1 to 49 ensuring that a random value is only placed in the array once.

The default constructor should set the values of the array to 0's. A method named `getvalue` should return the value of a corresponding element (number) if the index is passed as a parameter.

Implement your class in an application illustrating that your class works correctly. You may implement a listbox to display the values.

- 2) Design a class named `clsRectangle`, the class should store the length and the width of a rectangle as well as the Color. Include `Get` and `Set` methods for the various attributes. (You may create corresponding properties for each of the attributes)

Provide two constructor methods, a default constructor which will set the length and the width of the rectangle to 1 and 1 respectively and the color to Red. The copy constructor should set the length and the width to the according to the passed parameter values and the color to Blue.

Add a method `calcArea` that will return the area covered by the Rectangle  
Create a test application that illustrates the use of your class.

- 3) Modify the `MTime` class and include attributes for the Seconds and Milliseconds as well. You must add methods to increase the seconds and milliseconds. Take note the milliseconds values range from 0 to 99. Adapt the main solution to illustrate that your new class works correctly.
- 4) Create a program to calculate a quote the amount of carpet block boxes needed to cover the designated area for the "Carpet Man" company

The program contain the following methods:

- A method to calculate the number of carpet boxes needed to cover the designated room according to the length and width
- A method to calculate the labour cost
- A method to calculate the total cost, Labour + carpet price
- A default constructor, area = 0, cover area = 2.5, box price = 50, labour = 100
- And the required set and get methods

You must cater so that the user is able to enter the "Labour cost" and the "carpet box price" and also the amount each "box carpets cover" the floor. The quote can only contain boxes, in other words to buy one piece of the carpet you will have to buy the whole box.

You must also write code for error handling in the program.

The quote must be displayed in a label in the format shown.

[This question was originally part of notes for OOP in Delphi, developed by Mrs U Wassermann, which gave kind permission to have the question adapted for use in VB.NET]

**Carpet Man - Quote**

**Carpet Man**

Length in Meter: 4.3

Width in Meter: 4.8

Total floor size: 20.64  
 Total Boxes needed: 9  
 Total Labour: R2064  
 Total Cost: R2514

Process Quote Exit

Area Labour Price



### Example 3 – BasicBankAccount Class

The following example will illustrate the following concepts:

- Using overloaded methods
- Shared class objects

Create a class named `clsBasicBankAccount` with the following private fields:

- `AccountNr`
- `Balance`
- Interest rate per annum (Shared member) stored as a percentage.

Add the following methods

- A default constructor and a parameterized constructor
- A method `getBalance` that would return the current balance
- A method `deposit` that would add a deposit value the current balance
- A Boolean method `withdrawal` that will subtract a withdrawal amount from the balance; The method should return a true if there is sufficient funds in the balance and a false if the withdrawal amount is bigger than the balance. In such a case the balance should not be reduced.
- A `toString` method to return the class data as a string
- A `addInterest` method this method should calculate the interest for one month and add the interest to the `Balance`

Create a VB.NET GUI application and instantiate 3 `BasicBankAccount` objects. Show the implementation of the various methods. (You may hardcode some of the method calls.) Display the data for each account in a `Rich textbox`.

The following code has been created for the `clsBasicBankAccount`

```
Public Class clsBasicBankAccount
    Private _AccountNr As String
    Private _Balance As Decimal
    Private Shared _IntRate As Decimal

    Public Sub New()
        _AccountNr = ""
        _Balance = 0
        _IntRate = 0.1
    End Sub

    Public Sub New(ByVal AccNr As String, ByVal Bal As Decimal)
        _AccountNr = AccNr
        _Balance = Bal
        _IntRate = 0.1
    End Sub

    Public Shared ReadOnly Property iRate() As Decimal
        Get
            Return _IntRate
        End Get
    End Property

    Public Shared Sub setInterestRate(ByVal IR As Decimal)
        If (IR < 1) Then
            _IntRate = IR
        End If
    End Sub

    Public Function getBalance() As Decimal
        Return _Balance
    End Function

    Public Sub deposit(ByVal Amount As Decimal)
        _Balance = _Balance + Amount
    End Sub

    Public Function withdrawal(ByVal Amount As Decimal) As Boolean
        If (Amount < _Balance) Then
            _Balance = _Balance - Amount
        End If
    End Function
End Class
```

```

        Return True
    Else
        Return False
    End If
End Function

Public Overrides Function toString() As String
    Return "Account: " + _AccountNr + " Interest rate: " + _
        + (iRate * 100).ToString + "% Balance: " + _Balance.ToString
End Function

Public Sub addInterest()
    Dim Interest As Decimal
    Interest = _Balance * (_IntRate / 12)
    _Balance = _Balance + Interest
End Sub

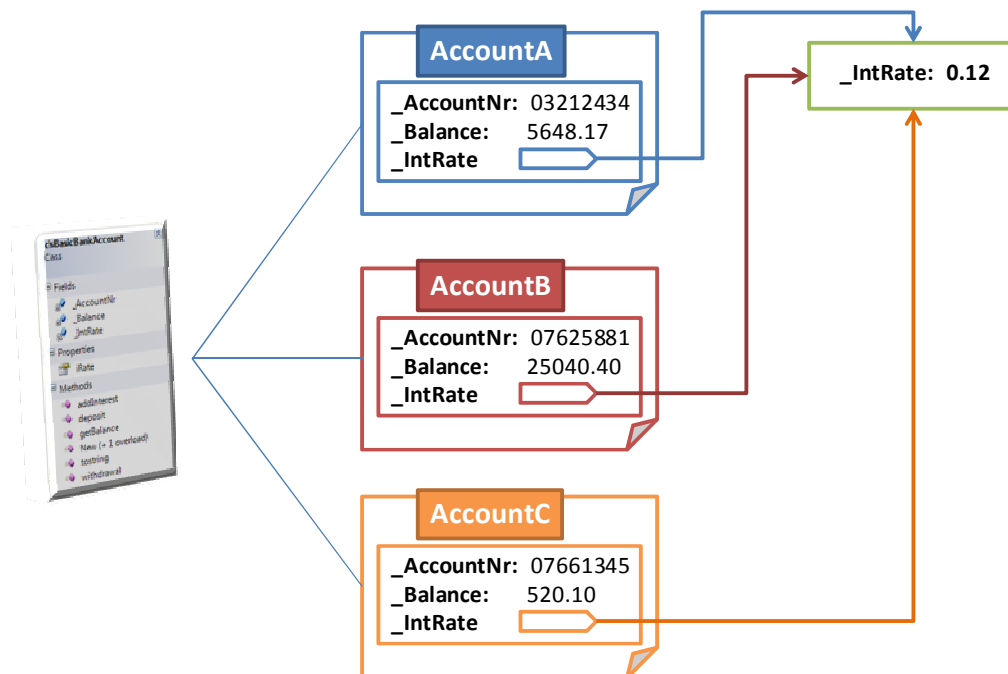
End Class

```

This example introduces two new VB.NET keywords i.e.: `Shared`, `ReadOnly` and `Overrides`

### ***Shared data members – and the Shared keyword***

When an object is instantiated from a class, each object contains its own value copy of each of the attributes. Thus changing the value of one attribute from Object A would not influence the same corresponding attribute of Object B. Sometimes however a developer may want to have one or more of the attributes shared by all objects instantiated of the class. The example above is a typical scenario where such an arrangement would deem proper. A bank would normally have all accounts to have the same interest rate. If the interest rate is to be changed for one account then the interest rate should be the same for all the other accounts as well. A Shared attribute has the same value for all objects instantiated of the class. Changing the value for one “object” will “change” the value for the other objects as well.



The figure depicts that: if three BasicBankAccount Objects are to be created, then all three objects would share the same memory location for the `_IntRate` attribute. If one “Object” changes the `_IntRate` the value would be “changed” for all objects. Shared attributes are sometimes referred to as class wide attributes.

**TAKE NOTE:** When a programmer wants to set a new value for a shared attribute in the implementation program the name of the class instead of the object is used, in conjunction with the shared set Method E.g:

```
clsBasicBankAccount.setInterestRate(0.15)
```

Using the name of an object will result in the compiler giving the following warning:

```
Private Sub btnAccOperations_Click(ByVal sender As
    AccountA.deposit(1000)
    AccountB.deposit(200)
    AccountC.withdrawal(300)

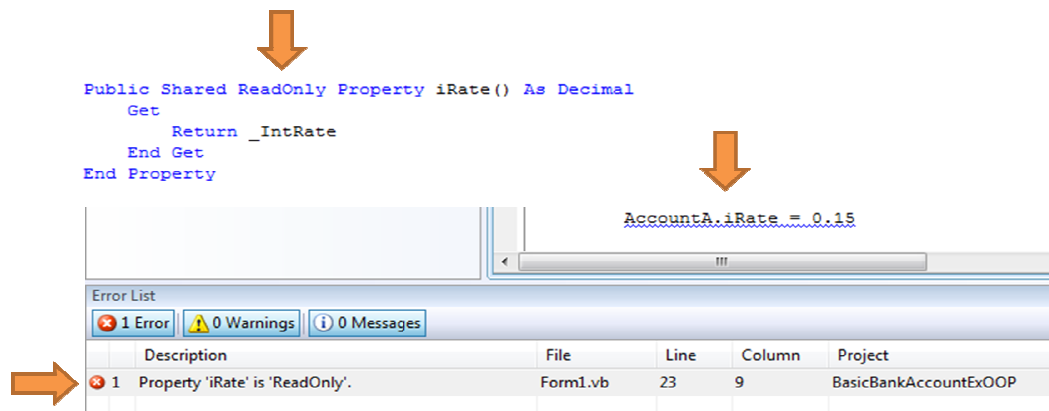
    AccountA.iRate = 0.15
```

Error List					
0 Errors 1 Warning 0 Messages					
	Description	File	Line	Column	Project
1	Access of shared member, constant member, enum member or nested type through an instance; qualifying expression will not be evaluated.	Form1.vb	23	9	BasicBankAccountExOOP

So in order to set and change the value of a shared attribute the name of the class instead of the name of an object is used.

### ***The IRate property – and the ReadOnly keyword***

If you examine the code for the IRate property, you will note that the keyword `ReadOnly` is used. This keyword when used with a property declaration indicates that the property is only to be read, and that the user will not be allowed to set or assign a value to the property. Attempting to assign a value to the property will result in an error.



The diagram illustrates the concept of a `ReadOnly` property. It shows a code snippet for a `Public Shared ReadOnly Property iRate() As Decimal` with a `Get` block returning `_IntRate`. An orange arrow points from this property declaration to another code snippet where `AccountA.iRate = 0.15` is being assigned. A second orange arrow points from this assignment line to an `Error List` window. The error list shows a single error: "Property 'iRate' is 'ReadOnly'." with details for File: Form1.vb, Line: 23, Column: 9, and Project: BasicBankAccountExOOP.

The developer of a class would deem it necessary to only give the user of the class permission only to read the value of the attribute.

For this example the programmer decided to create a separate set method to set the value of the shared `_IntRate` attribute.

```
Public Shared Sub setInterestRate(ByVal IR As Decimal)
    If (IR < 1) Then
        _IntRate = IR
    End If
End Sub
```

TAKE NOTE: That the setInterestRate subroutine was declared as a shared routine. The compiler would generate an error if the Subroutine tried to access a shared member without having been indicated as a shared method. ONLY Shared methods can access shared attributes.

### ***The toString method - and the Overrides keyword***

VB.NET is build upon the implementation of strong OO principles. The developers of VB.NET and the .NET framework decided that all classes that is created or defined by any developer are to inherit from a parent class named Object automatically. This implies that all classes user defined and built-in classes are related to one another.

The toString method is a method common to many classes in VB.NET. The overrides keyword tells the compiler that the method that you are writing as part of your class should redefine the current toString method.

```
Public Overrides Function toString() As String
    Return "Account: " + _AccountNr + " Interest rate: " + _
        + (iRate * 100).ToString + "% Balance: " + _Balance.ToString
End Function
```

As part of the example the developer decided that the clsBasicBankAccount should have its own toString method in order to return the data pertaining to one account object. In order to avoid the compiler to not confuse the other toString methods the keyword Overrides is used in front of the method declaration.

Omitting the keyword overrides would result in the compiler giving the following warning:

Error List					
<div> <div>0 Errors</div> <div>2 Warnings</div> <div>0 Messages</div> </div>					
	Description	File	Line	Column	Project
1	function 'toString' shadows an overridable method in the base class 'Object'. To override the base method, this method must be declared 'Overrides'.	clsBasicBankAccount	44	21	BasicBankAccountEx OOP
2	Function 'toString' doesn't return a value on all code paths. A null reference exception could occur at run time when the result is used.	clsBasicBankAccount	46	5	BasicBankAccountEx OOP

### **Writing the implementation code**

During the coding of the program the developer declared three BasicBankAccount objects. The IDE shows the developer that there are two possible ways to instantiate an Account object due to the fact that the constructor has been overloaded. Refer to the partial screenshot below:

```
Public Class frmBankAccount

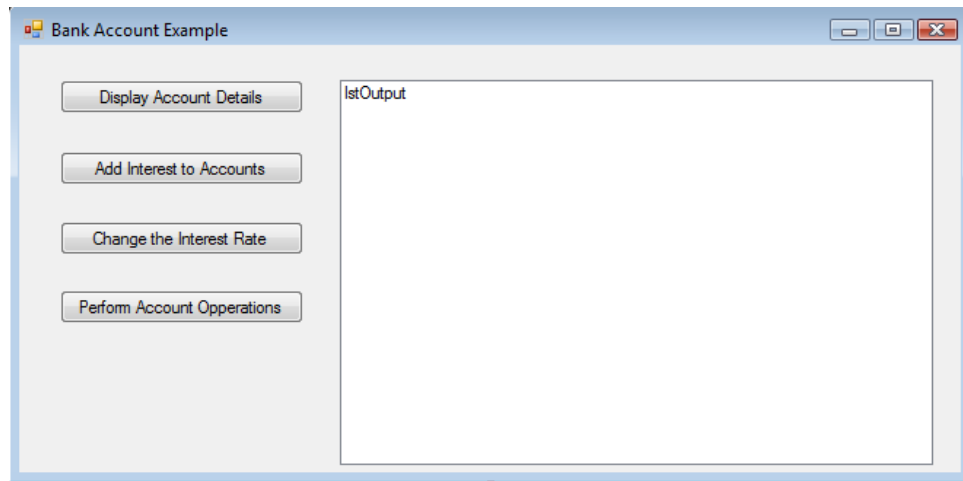
    Private AccountA As clsBasicBankAccount
    Private AccountB As clsBasicBankAccount
    Private AccountC As clsBasicBankAccount

    Private Sub btnDisplayAccounts_Click(ByVal sender As Object, ByVal e As EventArgs)
        '
    End Sub

    Private Sub frmBankAccount_Load(ByVal sender As Object, ByVal e As EventArgs)
        AccountA = New clsBasicBankAccount(
            1 of 2 New 0
        )
    End Sub
End Class
```

## The user interface

The following simple user interface was designed to illustrate the implementation of the class with three Account objects A – C.



## Changing the value of a shared attribute

In order to change the value of a shared Attribute one of two ways could be implemented.

- Firstly the developer could have decided to write and define a full shared property for the shared attribute or;
- the developer could create separate shared set and get methods to allow the user to assign and read a value from the shared attribute.

### ***Version 1 – Shared property***

The class definition code could also have been written as follows:

```
' VERSION 1
Public Class clsBasicBankAccount
    Private _AccountNr As String
    Private _Balance As Decimal
    Private Shared _IntRate As Decimal

    Public Sub New()
        _AccountNr = ""
        _Balance = 0
        _IntRate = 0.1
    End Sub

    Public Sub New(ByVal AccNr As String, ByVal Bal As Decimal)
        _AccountNr = AccNr
        _Balance = Bal
        _IntRate = 0.1
    End Sub

    Public Shared Property iRate() As Decimal
        Get
            Return _IntRate
        End Get
        Set(ByVal value As Decimal)
            _IntRate = value
        End Set
    End Property

    Public Function getBalance() As Decimal
        Return _Balance
    End Function

    Public Sub deposit(ByVal Amount As Decimal)
        _Balance = _Balance + Amount
    End Sub
```

```

End Sub

Public Function withdraw(ByVal Amount As Decimal) As Boolean
    If (Amount < _Balance) Then
        _Balance = _Balance - Amount
        Return True
    Else
        Return False
    End If
End Function

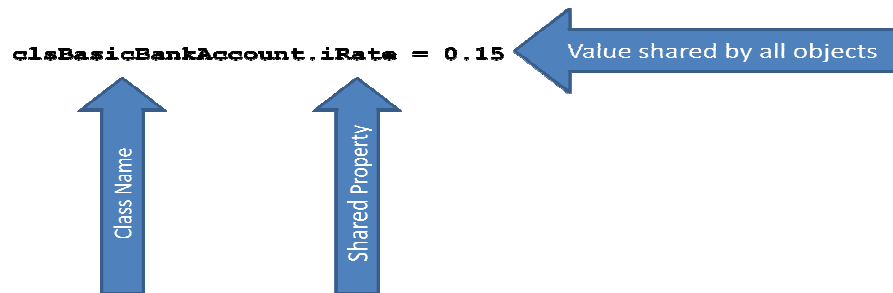
Public Overrides Function toString() As String
    Return "Account: " + _AccountNr + " Interest rate: " + _
        + (iRate * 100).ToString + "% Balance: " + _Balance.ToString
End Function

Public Sub addInterest()
    Dim Interest As Decimal
    Interest = _Balance * (_IntRate / 12)
    _Balance = _Balance + Interest
End Sub

End Class

```

In order to set the value of the shared attribute the programmer would refer to the name of the class and the shared attribute, e.g.



### ***Version 2 – Shared method subroutine and readonly property***

```

' VERSION 2
Public Class clsBasicBankAccount
    Private _AccountNr As String
    Private _Balance As Decimal
    Private Shared _IntRate As Decimal

    Public Sub New()
        _AccountNr = ""
        _Balance = 0
        _IntRate = 0.1
    End Sub

    Public Sub New(ByVal AccNr As String, ByVal Bal As Decimal)
        _AccountNr = AccNr
        _Balance = Bal
        _IntRate = 0.1
    End Sub

    Public Shared ReadOnly Property iRate() As Decimal
        Get
            Return _IntRate
        End Get
    End Property

    Public Shared Sub setInterestRate(ByVal IR As Decimal)
        If (IR < 1) Then
            _IntRate = IR
        End If
    End Sub

    Public Function getBalance() As Decimal
        Return _Balance
    End Function

```

```

Public Sub deposit(ByVal Amount As Decimal)
    _Balance = _Balance + Amount
End Sub

Public Function withdrawal(ByVal Amount As Decimal) As Boolean
    If (Amount < _Balance) Then
        _Balance = _Balance - Amount
        Return True
    Else
        Return False
    End If
End Function

Public Overrides Function toString() As String
    Return "Account: " + _AccountNr + " Interest rate: " + _
        + (iRate * 100).ToString + "% Balance: " + _Balance.ToString
End Function

Public Sub addInterest()
    Dim Interest As Decimal
    Interest = _Balance * (_IntRate / 12)
    _Balance = _Balance + Interest
End Sub
End Class

```

In order to set the value of the shared attribute the programmer would need to implement the shared `setInterestRate` method.

`clsBasicBankAccount.setInterestRate(0.15)`

Value passed to shared set method



Class Name



Shared method

**TAKE NOTE:** The example to follow is based on version 2 of the class.

### ***The program code:***

```

Public Class frmBankAccount
    Private AccountA As clsBasicBankAccount
    Private AccountB As clsBasicBankAccount
    Private AccountC As clsBasicBankAccount

    Private Sub btnDisplayAccounts_Click(...) Handles btnDisplayAccounts.Click
        lstOutput.Items.Add("ACCOUNT DETAILS LISTED")
        lstOutput.Items.Add(AccountA.toString())
        lstOutput.Items.Add(AccountB.toString())
        lstOutput.Items.Add(AccountC.toString())
    End Sub

    Private Sub frmBankAccount_Load(...) Handles MyBase.Load
        AccountA = New clsBasicBankAccount("03212434", 5648.17)
        AccountB = New clsBasicBankAccount("07625881", 25040.4)
        AccountC = New clsBasicBankAccount("07661345", 520.1)
    End Sub

    Private Sub btnAccOperations_Click(...) Handles btnAccOperations.Click
        lstOutput.Items.Add("ACCOUNT OPERATIONS PERFORMED")
        AccountA.deposit(1000)
        AccountB.deposit(200)
        AccountC.withdrawal(300)
    End Sub

    Private Sub btnAddInterest_Click(...) Handles btnAddInterest.Click
        lstOutput.Items.Add("INTEREST ADDED TO ACCOUNTS")
        AccountA.addInterest()
    End Sub
End Class

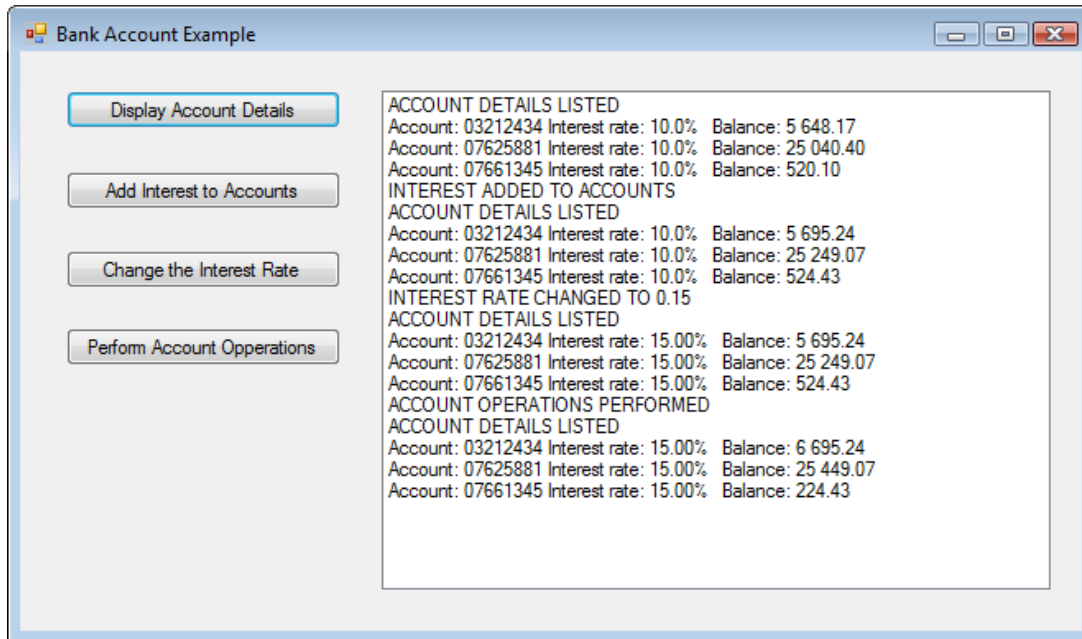
```

```

        AccountB.addInterest()
        AccountC.addInterest()
    End Sub

    Private Sub btnChangeInterest_Click(...)Handles btnChangeInterest.Click
        Dim IR As Decimal
        IR = Decimal.Parse(TextBox("Enter the New Interest Rate", ""))
        clsBasicBankAccount.setInterestRate(IR / 100)
        lstOutput.Items.Add("INTEREST RATE CHANGED TO " &
        clsBasicBankAccount.iRate.ToString("N2"))
    End Sub
End Class

```



### Notes on the program and sample output

When a user performs any click the information pertaining to which button has been clicked is listed in the listbox.

The user enters a new value for the interest rate by means of an inputbox.

TAKE NOTE: All the accounts would receive 15% interest after the change of the shared property by means of the shared method setInterestRate. The change was applicable to all Accounts.



## Creating an array of Objects

It is possible to create an array of objects in VB.NET.

The following example creates an array of 3 BasicAccountClass objects, the [Add new Accounts Object to Array] button dynamically adds a new account object to the array. Study the following code as well as the sample output screen. There are two generally accepted ways of creating object arrays with VB.NET. The first method which could render slow for bigger sized arrays implements the ReDim statement to resize the array, another alternative is to implement an arraylist object. The second method is faster for larger arrays and memory management is handled by the build in methods of the class. The following two examples illustrate both methods.

### Example A – Generating an array of objects using the Dim and ReDim statements.

```
Public Class frmBankAccount
    Dim AccountsArray(2) As clsBasicBankAccount
    Private Sub btnDisplayAccounts_Click(. . .) Handles btnDisplayAccounts.Click
        lstOutput.Items.Add("ACCOUNT DETAILS LISTED")
        Dim n As Integer
        MessageBox.Show(AccountsArray.Length.ToString)
        For n = 0 To AccountsArray.Length - 1
            lstOutput.Items.Add(AccountsArray(n).ToString)
        Next n
    End Sub

    Private Sub frmBankAccount_Load(. . .) Handles MyBase.Load
        AccountsArray(0) = New clsBasicBankAccount("03212434", 5648.17)
        AccountsArray(1) = New clsBasicBankAccount("07625881", 25040.4)
        AccountsArray(2) = New clsBasicBankAccount("07661345", 520.1)
    End Sub

    Private Sub btnAccOperations_Click(. . .) Handles btnAccOperations.Click
        lstOutput.Items.Add("ACCOUNT OPERATIONS PERFORMED")
        AccountsArray(0).deposit(1000)
        AccountsArray(1).deposit(200)
        AccountsArray(2).withdrawal(300)
    End Sub

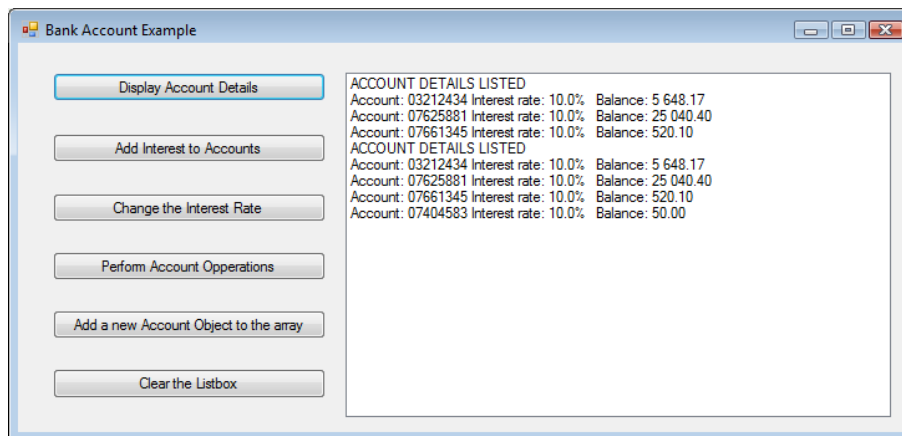
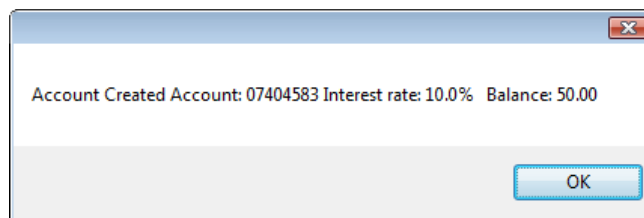
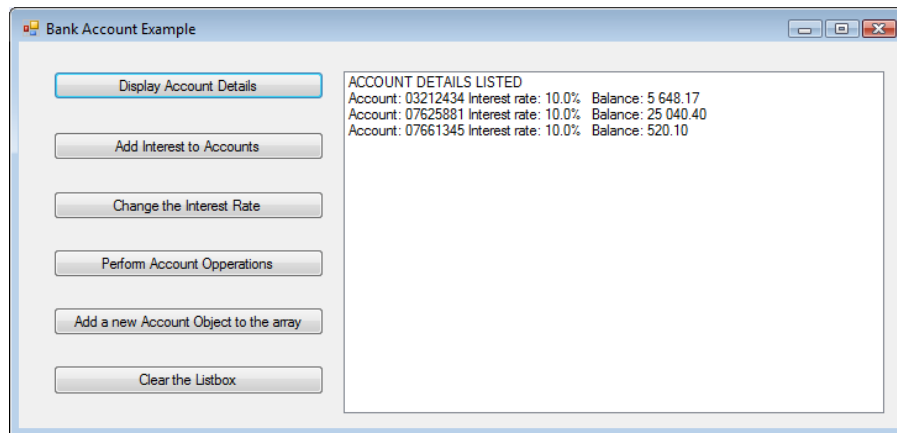
    Private Sub btnAddInterest_Click(. . .) Handles btnAddInterest.Click
        lstOutput.Items.Add("INTEREST ADDED TO ACCOUNTS")
        Dim n As Integer
        For n = 0 To AccountsArray.Length - 1
            AccountsArray(n).addInterest()
        Next n
    End Sub

    Private Sub btnChangeInterest_Click(. . .) Handles btnChangeInterest.Click
        Dim IR As Decimal
        IR = Decimal.Parse(InputBox("Enter the New Interest Rate", ""))
        clsBasicBankAccount.setInterestRate(IR / 100)
        lstOutput.Items.Add("INTEREST RATE CHANGED TO " & _
            clsBasicBankAccount.iRate.ToString("N2"))
    End Sub

    Private Sub btnNewAccObjectToArray_Click(. . .) Handles btnNewAccObjectToArray.Click
        Dim ElementCount As Integer
        Dim RandomAccount As String
        Dim random As New Random()
        RandomAccount = "07"
        RandomAccount = RandomAccount + random.Next(100000, 999999).ToString()
        ElementCount = AccountsArray.Length
        ReDim Preserve AccountsArray(ElementCount)
        AccountsArray(ElementCount) = New clsBasicBankAccount(RandomAccount, 50)
        MessageBox.Show("Account Created " + AccountsArray(ElementCount).ToString)
    End Sub

    Private Sub btnClearListBox_Click(. . .) Handles btnClearListBox.Click
        lstOutput.Items.Clear()
    End Sub
End Class
```

## Sample run screens



### NOTES on the program:

The statement:

```
Dim AccountsArray(2) As clsBasicBankAccount
```

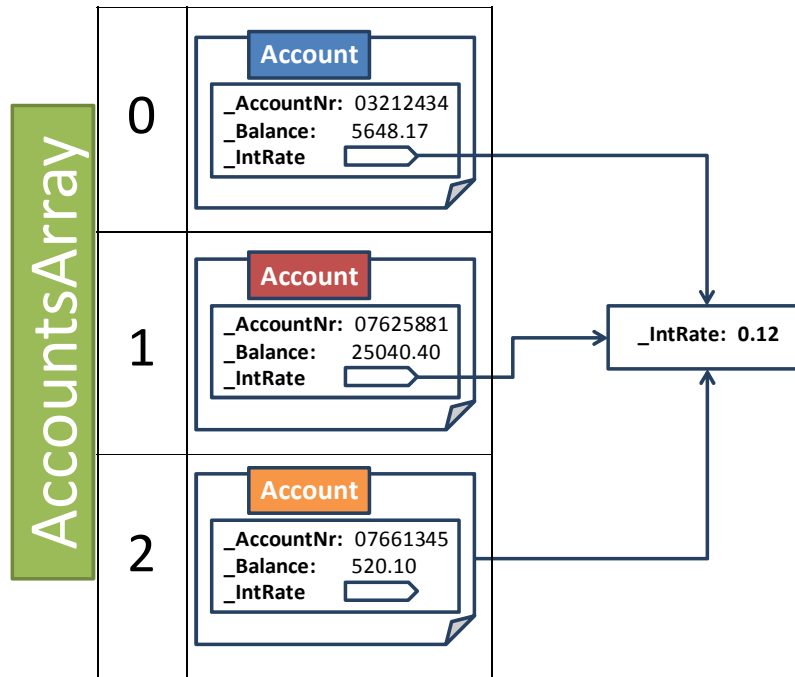
Declares an array that can reference three account objects.

The array is declared with class scope, this implies that the array would be accessible throughout the class.

The programmer could also have declared the array as a private attribute to the form class i.e.

```
Private AccountsArray(2) As clsBasicBankAccount
```

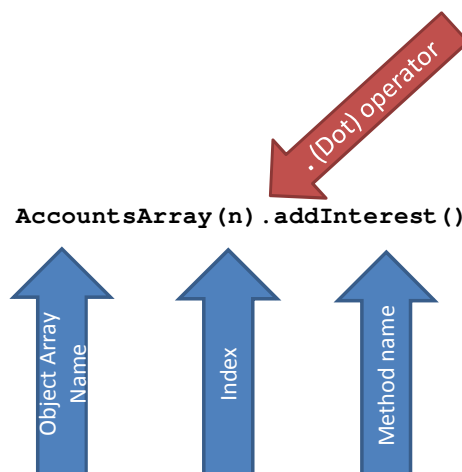
The on from load event creates each object in the array. Diagramaticly the array could be represented as follows:



Remember all arrays in VB.NET are zero based, i.e. the first element of the array is indexed with a zero. For loops are often used to iterate through the elements of an object array.

```
Private Sub btnAddInterest_Click(. . .) Handles btnAddInterest.Click
    lstOutput.Items.Add("INTEREST ADDED TO ACCOUNTS")
    Dim n As Integer
    For n = 0 To AccountsArray.Length - 1
        AccountsArray(n).addInterest()
    Next n
End Sub
```

The AddInterest button event utilizes the .Length method to determine the size of the array. The size of the array is used as the conditional control statement in the for loop.



In the fragment of code above the programmer references each object by using the () operator.

The NewAccObjectToArray event adds a new object to the array by implementing the ReDim Preserve statement. The size of the array is increased with one element by stating the size via the ElementCount variable. Remember the size of the array is indicated by using the upperbound index, therefore if the current size of the array is 3 elements i.e. element 0, 1 and 2. Using the ReDim statement to set the new upper bound to 3 will effectively add a new element reference to the array.

```
Private Sub btnNewAccObjectToArray_Click(. . .) Handles btnNewAccObjectToArray.Click
    Dim ElementCount As Integer
    Dim RandomAccount As String
    Dim random As New Random()
    RandomAccount = "07"
    RandomAccount = RandomAccount + random.Next(100000, 999999).ToString()
    ElementCount = AccountsArray.Length

    ReDim Preserve AccountsArray(ElementCount)

    AccountsArray(ElementCount) = New clsBasicBankAccount(RandomAccount, 50)
    MessageBox.Show("Account Created " + AccountsArray(ElementCount).ToString())

End Sub
```

NOTE: The Preserve keyword instructs the compiler not to wipe the current content of the array.

### Example B – Generating an array of objects by implementing an ArrayList

```
Imports System.Collections

Public Class frmBankAccount

    Dim MyAccountsArrayList = New ArrayList()

    Private Sub btnDisplayAccounts_Click(. . .) Handles btnDisplayAccounts.Click
        lstOutput.Items.Add("ACCOUNT DETAILS LISTED")
        Dim n As Integer
        MessageBox.Show(MyAccountsArrayList.Count.ToString())
        For n = 0 To MyAccountsArrayList.Count - 1
            lstOutput.Items.Add(MyAccountsArrayList(n).ToString())
        Next n
    End Sub

    Private Sub frmBankAccount_Load(. . .) Handles MyBase.Load
        MyAccountsArrayList.Add(New clsBasicBankAccount("03212434", 5648.17))
        MyAccountsArrayList.Add(New clsBasicBankAccount("07625881", 25040.4))
        MyAccountsArrayList.Add(New clsBasicBankAccount("07661345", 520.1))
    End Sub

    Private Sub btnAccOperations_Click(. . .) Handles btnAccOperations.Click
        lstOutput.Items.Add("ACCOUNT OPERATIONS PERFORMED")
        MyAccountsArrayList(0).deposit(1000)
        MyAccountsArrayList(1).deposit(200)
        MyAccountsArrayList(2).withdrawal(300)
    End Sub

    Private Sub btnAddInterest_Click(. . .) Handles btnAddInterest.Click
        lstOutput.Items.Add("INTEREST ADDED TO ACCOUNTS")
        Dim n As Integer
        For n = 0 To MyAccountsArrayList.Count - 1
            MyAccountsArrayList(n).addInterest()
        Next n
    End Sub

    Private Sub btnChangeInterest_Click(. . .) Handles btnChangeInterest.Click
        Dim IR As Decimal
        IR = Decimal.Parse(InputBox("Enter the New Interest Rate", ""))
        clsBasicBankAccount.setInterestRate(IR / 100)
        lstOutput.Items.Add("INTEREST RATE CHANGED TO " &
            clsBasicBankAccount.iRate.ToString("N2"))
    End Sub

    Private Sub btnNewAccObjectToArray_Click(. . .) Handles btnNewAccObjectToArray.Click
        Dim RandomAccount As String
```

```

Dim random As New Random()
Dim ElementCount As Integer
RandomAccount = "07"
RandomAccount = RandomAccount + random.Next(100000, 999999).ToString()

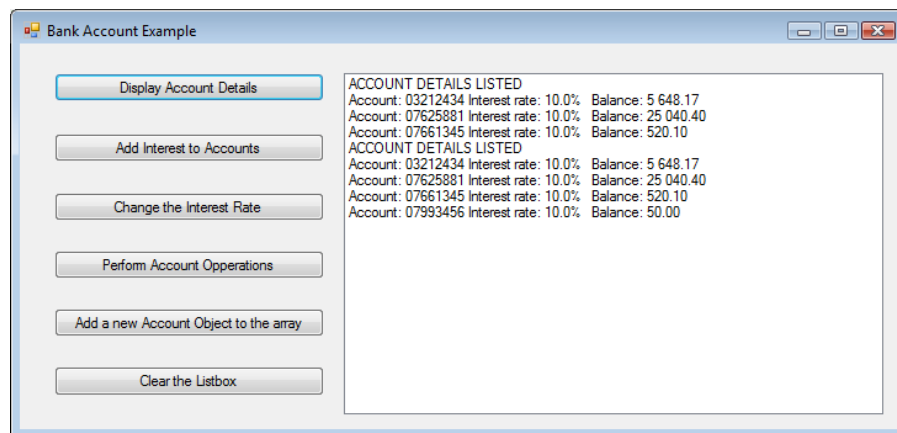
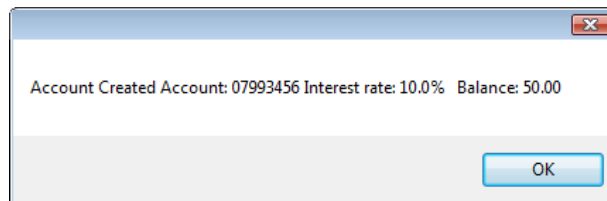
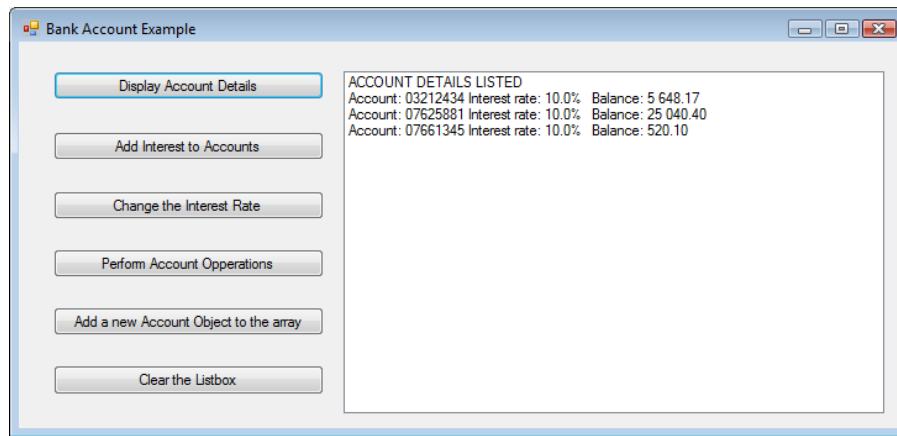
MyAccountsArrayList.Add(New clsBasicBankAccount(RandomAccount, 50))

ElementCount = MyAccountsArrayList.Count
MessageBox.Show("Account Created " + MyAccountsArrayList(ElementCount - 1).ToString)
End Sub

Private Sub btnClearListBox_Click(. . .) Handles btnClearListBox.Click
    lstOutput.Items.Clear()
End Sub
End Class

```

## Sample run screens



## NOTES on the program:

The second example implements an arraylist. In order to declare an arraylist object the programmer must instruct the compiler to import the correct library for the task.

The statement:

```
Imports System.Collections
```

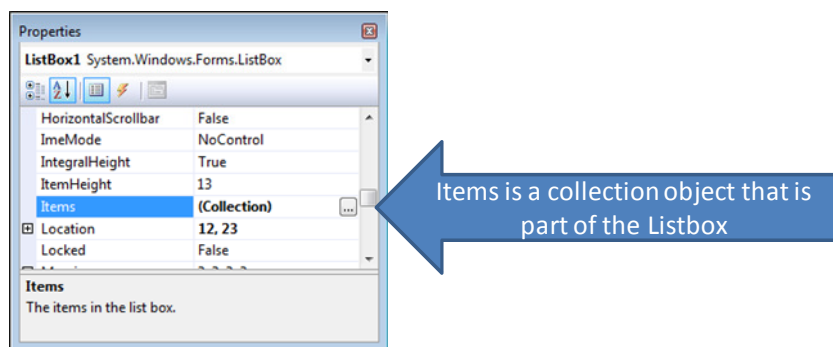
Tells the compiler to import the Systems.Collections namespace "class library", without the statement the compiler would not know where to find the class definition of the ArrayList class.

The statement:

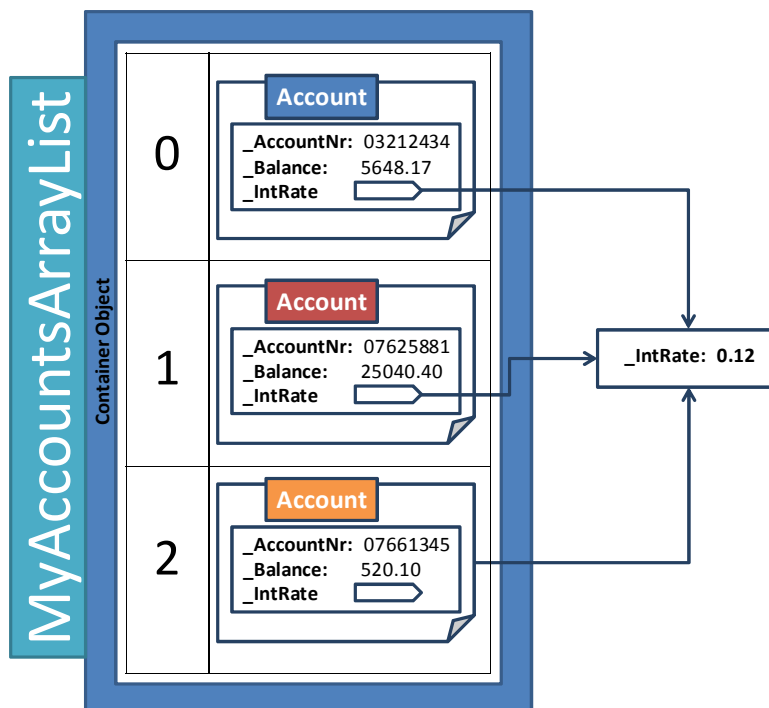
```
Dim MyAccountsArrayList = New ArrayList()
```

Declares and creates a MyAccountsArraylist object for use within your application.

ArrayLists are special objects that act as containers for similar data type objects. The listbox component implements a type of an arraylist, the user may add and delete and search for items in the listbox by utilizing methods that is also available in the arraylist class.



E.g. To add a string to a listbox object the Add method is implemented. Similarly to add an object to an arraylist the Add method is used.



The OnLoad event of the form is used to create and add 3 account objects to the MyAccountsArrayList arraylist.

```
MyAccountsArrayList.Add(New clsBasicBankAccount("03212434", 5648.17))
MyAccountsArrayList.Add(New clsBasicBankAccount("07625881", 25040.4))
MyAccountsArrayList.Add(New clsBasicBankAccount("07661345", 520.1))
```

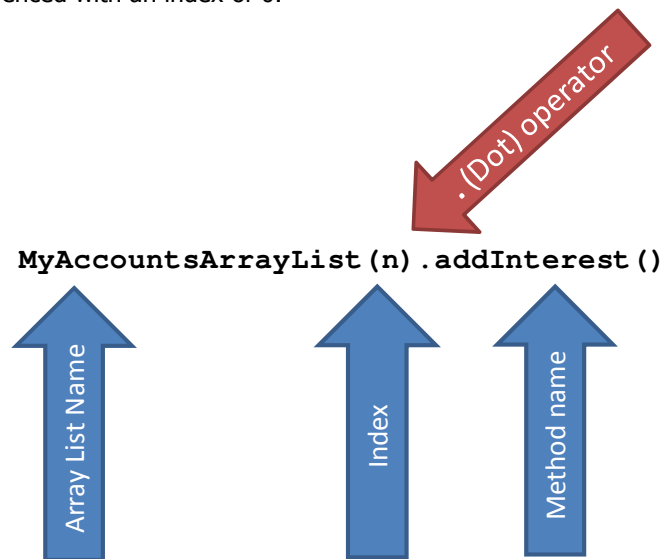
A BackAccount object is first created and added to the ArrayList object.

In order to invoke a method of the object in the arraylist the . (dot) operator is used. The AddInterest event utilizes the . (dot) operator the add the interest to each of the ArrayList object elements.

The Count property of the ArrayList class returns the number of elements in the array.

```
Private Sub btnAddInterest_Click(. . .) Handles btnAddInterest.Click
    lstOutput.Items.Add("INTEREST ADDED TO ACCOUNTS")
    Dim n As Integer
    For n = 0 To MyAccountsArrayList.Count - 1
        MyAccountsArrayList(n).addInterest()
    Next n
End Sub
```

Take note: The elements in the container arraylist object are also zero indexed, which means that the first element is referenced with an index of 0.



## Example 4 – Matla le Boitekanelo Gym (Strong and Fit Gym)

Compiled by: [Leandré Roux](#)

### Introduction

**Matla le Boitekanelo Gym** is Gym that focuses not only on the fitness of its members but also on their wellbeing. The gym offers all the standard equipment as well as the wellbeing centre which offers rest and relaxation services in the form of Jacuzzi's, stream rooms and massage parlors. The club also offers exercising routines that differ from the norm in the form of martial arts, kendo as well as boxing and kick boxing.

Matla le Boitekanelo Gym is looking for a developer to write a small system they can use to create new gym members as well as view existing members and keep count of their visits to the gym.

### The Application

The following example will illustrate the following concepts:

- Using overloaded methods
- Implementing a text file as part of the solution
- Creating an array of object using an ArrayList
- Exception handling
- Shared class attributes

Create a class named **clsClubMemeber**, the class is used to represent a member at the gym. The following data are to be stored for each member object:

- Membership Number (Property of clsClubMember)
- Name (Property of clsClubMember)
- Surname (Property of clsClubMember)
- Membership Type (Enumeration)
  - Bronze (Value 0)
  - Silver (Value 1)
  - Gold (Value 2)
- Basic Monthly Fee (Shared Attribute)
- Gym Visit Count (Property of clsClubMember)

The text file GymFees.txt contains the following data:

```
Bronze,185.45  
Silver,225.65  
Gold,345.50
```

**Note:** The monthly fee is calculated by adding the basic membership fee to the insurance fee. Members are charged the basic monthly fee according to what membership type they have selected. The monthly cost may be retrieved from the GymFees.txt file. In addition, all members pay the monthly insurance fee of R 120.00, this value is a Shared Attribute.

Create two **Dialogs** for the GUI in addition to Form1 which will act as the branching point of the application, i.e. the menu. Name them respectively NewMemberDialog and ViewMemberDialog.





### clsClubMember Requirements

Create two overloaded constructors for the class clsClubMember that meet the following requirements:

- A constructor that receives, as a parameter, a user given membership numbers as a String.
- A constructor that receives, as parameters, a user name and surname as String.

The following Properties that will act as both Get() and Set() methods:

- MembershipNumber As String
- Name As String
- Surname As String
- MembershipType As EnumMembershipType
- BasicMonthlyFee As Decimal
- VisitCount As Integer
- The Shared Function GetInsuranceFee() As Decimal
- A Function, GenerateMembershipNumber As String
  - The membership number is automatically created for each new gym member object. A membership number is only generated if no membership number is supplied via the constructor. Omitting a membership number should generate a membership number automatically. Use the overloaded constructor to perform this task. A membership number is generated by using the first letter of the members name and the first letter of the members surname added as a prefix to a randomly generated value between 1000 and 9999. Eg. DE2343

The following Enumeration:

- EnumMembershipType
  - Bronze = 0
  - Silver
  - Gold

### NewMemberDialog Requirements

- Create a Private Sub **CreateNewClubMember** which will create a new club member and call the correct constructor for clsClubMember. This function should also call all other functions needed to create a new member. This function should also store the created member in an ArrayList.
- Create a Private Sub **SetMembershipType** which will be called by CreateNewClubMember and will utilize a Case to set the selected membership type. (The membership type will be selected from a combobox)

- A Private Sub **AutoCompleteInformation** which will be called by CreateNewMember to complete the rest of the information on the form such as insurance fee's and basic monthly fees as well as display the generated membership number.
- A Private Sub **LoadFees** that will load the fees from the GymFees.txt file.
- A Public Function **GetNewMember** which will return a newly created member.

### ViewMemberDialog Requirements

- Create a **Constructor** that will receive a list of existing members as an ArrayList
- A Private Sub **GetNextMember** that receives an index as a parameter as an Integer, this function will retrieve the next member from the ArrayList
- You must be able to scroll between members using a Next and Back button as well as increment the visit count using a button.
- A Private Sub **UpdateFields** that will display the current member in the ArrayList.
- GetUpdatedData that takes a Referenced parameter of an ArrayList used to update the data if any changes occurred. Returns a Boolean value indicating a change was made.

**View Members**

Member Details

Membership Number : JY9058

Name : James

Surname : Young

Membership Type : Silver

Insurance Fee : 120

Basic Monthly Fee : 225.65

Visit Count : 2

At Gym Back Next

OK Cancel

### Form1 Requirements

- A Private Function GetIndex that takes as parameter a string value and return the Integer equivalent of the key as either 0 for Bronze, 1 for Silver or 2 for Gold.
- Code the buttons for creating the correct dialog and passing the data as needed.

**Matla le Boitekanelo Gym (Strong and Fit)**

New Member

View Members

Exit

MatlaleBoitekaneloGym Class Diagrams

clsClubMember

Class

→ Form

Fields

\_BasicMonthlyFee

\_InsuranceFee

\_MembershipNumber

\_MemType

\_Name

\_Surname

\_VisitCount

Properties

Basic MonthlyFee

MembershipNumber

MembershipType

Name

Surname

VisitCount

Methods

GenerateMembershipNumber

GetInsuranceFee

New (+ 1 overload)

Nested Types

Form1

Class

→ Form

Fields

clubMembers

fileReader

fileWriter

memberData

memberFile

newMemberDialog

savedMembers

updatedData

viewMembershipDialog

Methods

btnNew\_Click

Button1\_Click

Button2\_Click

GetIndex

WriteToFile

NewMemberDialog

Class

→ Form

Fields

clubFees

clubMember

clubMemberArray

fileLocation

fileReader

Methods

AutoCompleteInformation

btnAdd\_Click

Cancel\_Button\_Click

CreateNewClubMember

GetNewMember

LoadFees

OK\_Button\_Click

SetMembershipType

ViewMembersDialog

Class

→ Form

Fields

data

dataHasChanged

member

position

Methods

btnBack\_Click

btnNext\_Click

btnVisit\_Click

Cancel\_Button\_Click

GetNextMember

GetUpdatedData

New

OK\_Button\_Click

UpdateFields

## Matla le Boitekanelo Gym (Strong and Fit Gym): Suggested Solution -

Developed by: [Leandré Roux](#)

### clsClubMember

```
Private _MembershipNumber As String
Private _Name, _Surname As String
Private _BasicMonthlyFee As Decimal
Private _MemType As EnumMembershipType
Private _VisitCount As Integer
Public Shared _InsuranceFee As Decimal = 120.0
```

We start by defining our global variables for the class clsClubMember, these will hold the data for each individual member. Note that `_MemType` is declared as our enumerator type `EnumMembershipType`. Finally we declare our `Public Shared _InsuranceFee As Decimal = 120.0`, this will be our shared attribute between the member objects.

```
Public Sub New(ByVal name As String, ByVal surname As String)
    _Name = name
    _Surname = surname
    _MembershipNumber = GenerateMembershipNumber()
End Sub

Public Sub New(ByVal membershipNumber As String)
    _MembershipNumber = membershipNumber
End Sub
```

Next, we define our overloaded constructors for clsClubMember, we create the first constructor with parameters for name and surname and in the body we set the global variables to these parameters as well as initialize the `_MembershipNumber` parameter by calling the `GenerateMembershipNumber` function, this constructor will be called if we do not specify a default membership number. We need the name and surname of the member before we can successfully execute the `GenerateMembershipNumber` function.

The second constructor takes a single parameter `membershipNumber` and initializes the global variable to this.

Next, we create our Properties that will hold the values for the current member object.

```
Property MembershipNumber() As String
    Get
        Return _MembershipNumber
    End Get
    Set(ByVal value As String)
        _MembershipNumber = value
    End Set
End Property

Property Name() As String
    Get
        Return _Name
    End Get
    Set(ByVal value As String)
        _Name = value
    End Set
End Property

Property Surname() As String
    Get
        Return _Surname
    End Get
    Set(ByVal value As String)
        _Surname = value
    End Set
End Property
```

```

Property MembershipType() As EnumMembershipType
    Get
        Return _MemType
    End Get
    Set(ByVal value As EnumMembershipType)
        _MemType = value
    End Set
End Property

Property BasicMonthlyFee() As Decimal
    Get
        Return _BasicMonthlyFee
    End Get
    Set(ByVal value As Decimal)
        _BasicMonthlyFee = value
    End Set
End Property

Property VisitCount() As Integer
    Get
        Return _VisitCount
    End Get
    Set(ByVal value As Integer)
        _VisitCount = value
    End Set
End Property

```

These Properties provide us a means of retrieving and setting the information stored by them, thus they act as both **accessor** and **mutator** functions. Next, we define our shared attribute:

```

Public Shared Function GetInsuranceFee() As Decimal
    Return _InsuranceFee
End Function

```

This Function will provide the insurance fee as needed, it is shared among all instances of this class.

The GenerateMembershipNumber Function is used to generate a membership number when invoked and will use the Property values of \_Name and \_Surname to generate this number. The code is as follows

```

Private Function GenerateMembershipNumber() As String
    Dim generatedMembershipNumber As String
    Dim initials As String = String.Concat(_Name.Substring(0, 1),
                                           _Surname.Substring(0, 1))

    Dim randomVal As Random = New Random()

    generatedMembershipNumber = String.Concat(initials,
                                              randomVal.Next(1000, 10000).ToString())

    Return generatedMembershipNumber
End Function

```

This Function creates the membership number prefix by using the **substring** functionality of the String class. We substring the value of \_Name to get the first letter of the member's name and substring the value of \_Surname to get the first letter of the surname of the member, finally we use the **concat** function to concatenate them together.

Next, we create a variable of type Random(), this will allow us to generate a random value. We take the concatenated initials and prefix them to a random number between 1000 and 9999. The result of the concatenation is then returned to the calling function.

We also need to create an enumeration for the membership type, this is basically a method for creating an enumerable type, i.e. we represent values by giving them easily interpreted meanings, in this case we want to represent Bronze as the integer 0, Silver as the Integer 1 and Gold as the Integer 2, which is more suitable when working with a combobox. We do not want to burden the programmer with having to remember what Integer values represent what membership types, so we use an enumerator. The programmer may simply refer to the more understandable enumeration value and the compiler will automatically know what Integer value to use.

```
Public Enum EnumMembershipType As Integer
    Bronze = 0
    Silver
    Gold
End Enum
```

As an example, if you wish to reference the Gold membership, you would simply say the following:

```
EnumMembershipType.Gold
```

The compiler would automatically know the value for this would be the Integer 2.

## NewMemberDialog

The NewMemberDialog will be used to add new members to the gym, logically it will always just add one member to the gym at a time, but we will still use an ArrayList to save this member, this is to create a more generic approach.

Start by defining the following global variables:

```
Private clubMember As clsClubMember
Private clubMemberArray As ArrayList
Private clubFees As ArrayList
Private fileReader As StreamReader
Private Const fileLocation As String = "C:\GymFees.txt"
```

We create a variable that will represent the object of clsClubMember, then we define the ArrayList we will use to save this object in as well as another ArrayList that will hold the data representing the Gym Fees.

We define a StreamReader variable that we will use to read the data from the GymFees.txt file and finally we create a Constant fileLocation variable that will point to our GymFees.txt file.

We add the following code to the event listener of our 'Create Member' button, this will call the relevant functions for creating a new member.

```
CreateNewClubMember()
btnAdd.Enabled = False
OK_Button.Enabled = True
```

The code for the CreateNewClubMember sub is as follows:

```
Private Sub CreateNewClubMember()

    If String.IsNullOrEmpty(txtMembershipNumber.Text) Then
        clubMember = New clsClubMember(txtName.Text, txtSurname.Text)
        SetMembershipType()
        AutoCompleteInformation()
    Else
        clubMember = New clsClubMember(txtMembershipNumber.Text)
        clubMember.Name = txtName.Text
        clubMember.Surname = txtSurname.Text
        SetMembershipType()
        AutoCompleteInformation()
    End If

    If (clubMemberArray Is Nothing) Then
        clubMemberArray = New ArrayList
    Else
        clubMemberArray.Clear()
    End If

    clubMemberArray.Add(clubMember)

End Sub
```

We start by checking if the textbox txtMembershipNumber has any text typed into it, if so we call the clsClubMember constructor that takes this membership number as a parameter, we then invoke the other required functions to complete the member creation. We also set the name and surname properties manually.

If the textbox was empty, we need to pass the name and surname parameters to the correct constructor so it will automatically create the membership number. Next we initialize the ArrayList object that will hold the clsClubMember object. Finally we add the newly created club member to our array.

The code for SetMembershipType follows:

```
Private Sub SetMembershipType()
    Select Case cmbMembershipType.SelectedIndex
        Case 0
            clubMember.MembershipType =
                clsClubMember.EnumMembershipType.Bronze
            LoadFees()
        Case 1
            clubMember.MembershipType =
                clsClubMember.EnumMembershipType.Silver
            LoadFees()
        Case 2
            clubMember.MembershipType =
                clsClubMember.EnumMembershipType.Gold
            LoadFees()
    End Select

    clubMember.BasicMonthlyFee =
    Convert.ToDecimal(clubFees(cmbMembershipType.SelectedIndex)(1).ToString())

End Sub
```

The above function is used to set the membership type for the new member, we do this by retrieving the SelectedIndex of the combobox that holds three options, Bronze, Silver and Gold. These indexes correspond to our enumeration type, so we can set the MembershipType property by calling the appropriate enumeration value.

After we have set the MembershipType, we may load the GymFees.txt file to retrieve the associated fee of the selected membership type.

The AutoCompleteInformation Sub is used to fill in the remaining fields that should not be filled in manually, these include the calculated monthly fees.

```
Private Sub AutoCompleteInformation()
    txtBasicMonthlyFee.Text =
    (Convert.ToDecimal(clsClubMember.GetInsuranceFee().ToString()) _ +
    Convert.ToDecimal(clubMember.BasicMonthlyFee.ToString())).ToString()

    txtMembershipNumber.Text = clubMember.MembershipNumber
    txtInsuranceFee.Text = clsClubMember.GetInsuranceFee().ToString()
End Sub
```

In the method depicted above, we calculate the monthly fee by adding the decimal value of the insurance fee to the decimal value of the basic monthly fee. We also set the generated membership number field and also display the insurance fee.

The LoadFees Sub is as follows

```
Private Sub LoadFees()

    If clubFees Is Nothing Then
        clubFees = New ArrayList
    Else
        clubFees.Clear()
    End If

    If (File.Exists(fileLocation)) Then
```



```

Try
    fileReader = New StreamReader(fileLocation)

    While Not fileReader.EndOfStream
        clubFees.Add(fileReader.ReadLine().Split(","))
    End While
Catch ex As Exception
    MessageBox.Show(String.Format("An Exception Occured While
                                   Reading The File. {0}", ex.ToString()), _
                    "Could Not Read File", MessageBoxButtons.OK,
                    MessageBoxIcon.Error)

End Try
End If
End Sub

```

We start off by checking if our container ArrayList has been created, if not we create a new instance, otherwise we simply clear any existing data from it. Before we try to access the file, we first check if it exists, if so we create a new StreamReader object and continue to read a single line at a time until we reach the end of the file.

Each line that is read is split every time the ',' character is found, thus we are creating an array of objects that is similar to a two-dimensional array. We perform all these tasks in a Try-Catch block, this acts as exception handling for in case there is a problem reading the file.

Finally we create a Public Function GetNewMember that will return the newly populated ArrayList to the calling function.

```

Public Function GetNewMember() As ArrayList
    Return clubMemberArray
End Function

```

## ViewMemberDialog

The ViewMemberDialog will display members already part of the gym and will allow you to scroll back and forth between existing members as well as increase the visitations of each member.

We start by defining our global variables, an ArrayList that will hold all the current members, an Integer index that will act as a positioning mechanism in our ArrayList and an instance of the class clsClubMember so we can access the properties of the stored members. We also create a Boolean flag to check if any changes were made to the visit count.

```

Private data As ArrayList
Private position As Integer = 0
Private member As clsClubMember
Private dataHasChanged As Boolean = False

```

We create an overloaded constructor that will receive the ArrayList of members already enrolled at our gym, we create a reference to this data by setting it equal to our local ArrayList. Next, we load the first member from the ArrayList by invoking our GetNextMember function and passing zero as the first index.

Finally we tell the application to fill all the fields with the data from the member object at this index.

```

Public Sub New(ByVal data As ArrayList)
    InitializeComponent()

    Me.data = data

    member = GetNextMember(0)
    UpdateFields()
End Sub

```

The GetNextMember Function is used to retrieve the member at the given index to the calling function.

```

Private Function GetNextMember(ByVal index As Integer) As clsClubMember
    Return data(index)
End Function

```

The UpdateFields Sub is used to retrieve the information from the returned member object and display the information to the end user.

```

Private Sub UpdateFields()
    txtMembershipNumber.Text = member.MembershipNumber.ToString()
    txtName.Text = member.Name.ToString().ToString()
    txtSurname.Text = member.Surname.ToString().ToString()
    txtBasicMonthlyFee.Text =
        member.BasicMonthlyFee.ToString().ToString()
    txtInsuranceFee.Text = clsClubMember.GetInsuranceFee().ToString()
    cmbMembershipType.SelectedIndex = member.MembershipType()
    txtVisitCount.Text = member.VisitCount().ToString()
End Sub

```

The code for the 'Next' button's event listener is as follows:

```

If (position < data.Count - 1) Then
    If position <> data.Count Then
        position += 1
    End If
    member = GetNextMember(position)
    UpdateFields()
End If

```

It first checks to see if we have not already reached the final object, then it checks to see if the current position is not the last item in the array, if not it increments the position with one to move to the next object. Finally it retrieves the member at that index and then forces a refresh of the data being displayed.

The code for the 'Back' button's event listener is as follows, it performs the same functionality as discussed above, but decrements the current index:

```

If (position > -1) Then
    If position <> 0 Then
        position -= 1
    End If
    member = GetNextMember(position)
    UpdateFields()
End If

```

The code for the 'At Gym' button's event listener is as follows

```

member.VisitCount = (Convert.ToInt32(member.VisitCount) + 1)
                    .ToString()

txtVisitCount.Text = member.VisitCount.ToString()
dataHasChanged = True

```

The function simply increments the visit count and sets the flag to true.

The GetUpdatedData passes an ArrayList as a referenced parameter, if any changes were made this ArrayList is updated to reflect the changes. A Boolean is used as a flag.

```

Public Function GetUpdatedData(ByRef memberData As ArrayList) As Boolean
    If dataHasChanged Then
        memberData = data
        Return True
    Else
        Return False
    End If
End Function

```

## Form1

This Form acts as our branching point to all other functionality, it acts as a menu to the other Dialogs. We start by defining our global variables, these include objects of our dialogs as well as our class clsClubMember. We also define a StreamReader and StreamWriter for reading and writing to file. Finally we define a Constant String variable that points to our text file GymMembers.txt which will save our enrolled members for later retrieval.

```
Private newMemberDialog As NewMemberDialog
Private clubMembers, savedMembers, updatedData As ArrayList
Private fileWriter As StreamWriter
Private fileReader As StreamReader
Private memberData As clsClubMember
Private viewMembershipDialog As ViewMembersDialog
Private Const memberFile As String = "C:\GymMembers.txt"
```

The code for the 'New Member' is as follows:

```
NewMemberDialog = New NewMemberDialog()

If (clubMembers Is Nothing) Then
    clubMembers = New ArrayList
Else
    clubMembers.Clear()
End If

NewMemberDialog.ShowDialog()

If NewMemberDialog.DialogResult = Windows.Forms.DialogResult.OK Then
    clubMembers = NewMemberDialog.GetNewMember()

    WriteToFile(clubMembers, True)

End If
```

A new object of the NewMemberDialog is created before we initialize our ArrayList for storing our club members. We call the ShowDialog function to display our dialog to the end user, then we check if the user clicked the OK button before continuing.

If the user clicked on OK, we collect the member ArrayList by invoking the GetNewMember function and finally we invoke our custom function WriteToFile with the first parameter being our ArrayList and the second a flag indicating if the data should be appended.

The WriteToFile Sub is used to write the member information to file for later retrieval during the ViewMemberDialog invocation.

```
Private Sub WriteToFile(ByVal clubMembers As ArrayList, ByVal append As Boolean)
    fileWriter = New StreamWriter(memberFile, append)

    For i As Integer = 0 To clubMembers.Count - 1
        Try

            fileWriter.WriteLine(clubMembers(i).MembershipNumber().ToString() + "," + _
                                clubMembers(i).Name().ToString() + "," + _
                                clubMembers(i).Surname().ToString() + "," + _
                                clubMembers(i).MembershipType().ToString() + "," + _
                                clubMembers(i).BasicMonthlyFee().ToString() + "," + _
                                clubMembers(i).VisitCount().ToString() + "," + _
                                clubMembers(i).GetInsuranceFee().ToString())

        Catch ex As Exception
            MessageBox.Show(String.Format("An Exception Occured While Writing The File. {0}", ex.ToString()), _
                            "Could Not Write File", MessageBoxButtons.OK, _
                            MessageBoxIcon.Error)

        End Try
    Next

    fileWriter.Close()
End Sub
```

We start by creating a new StreamWriter and setting the file location and the flag indicating an append or overwrite of the file. Next we use a loop to write each individual member object in our ArrayList to a comma separated text file.

The GetIndex Function returns the equivalent Integer value representing the SelectedIndex for the combobox during the ViewMemberDialog invocation.

```
Private Function GetIndex(ByVal stringValue As String) As Integer
    If (stringValue.Equals("Bronze")) Then
        Return 0
    ElseIf (stringValue.Equals("Silver")) Then
        Return 1
    ElseIf (stringValue.Equals("Gold")) Then
        Return 2
    End If
End Function
```

The code for the 'View Members' event listener is listed below:

```
If savedMembers Is Nothing Then
    savedMembers = New ArrayList
Else
    savedMembers.Clear()
End If

fileReader = New StreamReader(memberFile)

While Not fileReader.EndOfStream
    Try
        Dim dataArray() = fileReader.ReadLine.Split(",")
        memberData = New clsClubMember(dataArray(0).ToString())
        memberData.Name = dataArray(1).ToString()
        memberData.Surname = dataArray(2).ToString()
        memberData.MembershipType = GetIndex(dataArray(3).ToString())
        memberData.BasicMonthlyFee = dataArray(4).ToString()
        memberData.VisitCount = dataArray(5).ToString()

        savedMembers.Add(memberData)

    Catch ex As Exception
        MessageBox.Show(String.Format("An Exception Occured While
            Reading The File. {0}", ex.ToString()), _
            "Could Not Read File", MessageBoxButtons.OK,
            MessageBoxIcon.Error)

    End Try
End While

fileReader.Close()
viewMembershipDialog = New ViewMembersDialog(savedMembers)
viewMembershipDialog.ShowDialog()

If viewMembershipDialog.DialogResult = Windows.Forms.DialogResult.OK Then
    If (viewMembershipDialog.GetUpdatedData(updatedData)) Then
        WriteToFile(updatedData, False)
    End If
End If
```

This function creates an empty ArrayList for holding the loaded members from file. We start by creating a StreamReader object and passing the file location of our saved members to it. We read each line of the file until we reach the end of the stream, during each read we add the data to a new object of clsClubMember and finally to the ArrayList.

After we have placed all our clsClubMember objects in our ArrayList we create a new instance of ViewMembersDialog and pass the filled ArrayList as parameter, finally we display the dialog for the end user. We check if the user clicked on OK, if so we call the GetUpdatedData function to see if any changes were made, if so we overwrite the file to reflect the changes.

The code for our 'Exit' button simply disposes the application

```
Me.Dispose()
```

## Example 5: Array of objects – Moka Rugby Club

Create a program for a rugby club to check if an applicant is qualifying to be part of this club.

The criteria are as follows:

The player must be older than 20

If the player is a male then the minimum weight is 50, otherwise it is 40

Create a class for the player "RugbyPlayer" with fields for Name, Surname, Age, Gender, Weight, Position (BackLine, FullBack, ThightFive) and the Player number and a Success which data type is Boolean

The Player number is generated as follows: the first character is the letter representing his gender, ex "M" for Male and "F" for Female, the second character is the first letter of his name and the third character is the first character of his surname, then the rest is his age + weight. EX: Peter Smith, weight 52, age 21. The code will be MPS73

Include set and get methods for all the attributes. Include a boolean method to validate if the data that is entered is correct matches the criteria set out for the player to be included in the team

Use a constructor to initialize the variable and the Boolean to false.

Develop a program that will create an array of Rugby players when the data is read from a text file named Applicants.txt

The content of the textfile named Applicants are comma separated, and each line represents an applicant's record. Take note a player number must be automatically generated for each applicant.

```
Paul,White,M,18,50,BackLine,
Aubrey,Zuma,M,20,48,FullBack,
Peter,Shibambo,M,21,66,BackLine,
Bony,Kgosana,F,22,54,FullBack,
Candice,Smith,F,17,45,FullBack,
Chris,Ndobe,M,23,52,Thight-5,
Piet,Xavier,M,19,67,Thight-5,
Precious,van Wyk,F,20,45,Thight-5,
James,Magagula,M,16,42,BackLine,
```

The load button will load the players into an array of players and display it into the list box.

The Process button must generate a Player number, and list the result of the application in a listbox.

Use the following screen to guide you to develop your own application.



## The Rugbyplayer class

```
Option Explicit On
Option Strict On
Public Class RugbyPlayer
    Private PName As String
    Private PSurname As String
    Private PGender As Char
    Private PAge As Integer
    Private PWeight As Integer
    Private PNumber As String
    Private PPosition As String
    Private SuccessAnswer As Boolean

    Public Sub New()
        SuccessAnswer = False
        PName = ""
        PSurname = ""
        PGender = Convert.ToChar("E")
        PAge = 0
        PWeight = 0
        PNumber = "0"
    End Sub

    Public Sub New(ByVal _Name As String, ByVal _Surname As String, ByVal _Age As Integer, _
        ByVal _Weight As Integer, ByVal _Gender As Char)
        SuccessAnswer = False
        PName = _Name
        PSurname = _Surname
        PGender = _Gender
        PAge = _Age
        PWeight = _Weight
        PNumber = "0"
    End Sub

    Public Property Name() As String
        Get
            Return PName
        End Get
        Set(ByVal value As String)
            PName = value
        End Set
    End Property

    Public Property Surname() As String
        Get
            Return PSurname
        End Get
        Set(ByVal value As String)
            PSurname = value
        End Set
    End Property

    Public Property Gender() As Char
        Get
            Return PGender
        End Get
        Set(ByVal value As Char)
            PGender = value
        End Set
    End Property

    Public Property Position() As String
        Get
            Return PPosition
        End Get
        Set(ByVal value As String)
            PPosition = value
        End Set
    End Property

    Public Property Age() As Integer
        Get
```

```

        Return PAge
    End Get
    Set(ByVal value As Integer)
        PAge = value
    End Set
End Property

Public Property Weight() As Integer
    Get
        Return PWeight
    End Get
    Set(ByVal value As Integer)
        PWeight = value
    End Set
End Property

Public Function getPlayerNumber() As String
    Dim temp As Integer

    temp = PAge + PWeight

    If PGender = "M" Then
        PNumber = "M"
    Else
        PNumber = "F"
    End If
    PNumber = PNumber + PName.Substring(0, 1).ToUpper
    PNumber = PNumber + PSurname.Substring(0, 1).ToUpper
    PNumber = PNumber + temp.ToString

    Return PNumber
End Function

Public Function getSuccess() As Boolean
    If PAge <= 21 Then
        Select Case Convert.ToString(PGender)
            Case "M"
                If PWeight < 65 Then
                    SuccessAnswer = True
                End If
            Case "F"
                If PWeight < 55 Then
                    SuccessAnswer = True
                End If
        End Select
    End If

    Return SuccessAnswer
End Function
End Class

```

## The Implementation program

```

Option Strict On
Option Explicit On
Public Class frmMokaRugbyPartA
    Private InputFileReader As System.IO.StreamReader
    Private OneFileLine As String
    Private SpacePos As Integer
    Private Player() As RugbyPlayer
    Private ArrayCounter As Integer = 0
    Private Sub btnProcess_Click(. . .) Handles btnProcess.Click
        Me.lstDisplay.Items.Clear()
        Dim Result As String
        For J As Integer = 0 To ArrayCounter - 1 Step 1
            If Player(J).getSuccess Then
                Result = "Successful"
            Else
                Result = "NOT Successful"
            End If
            Me.lstDisplay.Items.Add(Player(J).Name & ControlChars.Tab & Player(J).Surname &
            ControlChars.Tab & _
            Player(J).Age & ControlChars.Tab & Player(J).Gender & ControlChars.Tab & _
            Player(J).Weight & ControlChars.Tab & Player(J).Position & ControlChars.Tab & _
            Player(J).getPlayerNumber & ControlChars.Tab & Result)
        Next
    End Sub
End Class

```

```

End Sub

Private Sub btnClose_Click(. . .) Handles btnClose.Click
    Me.Close()
End Sub

Private Sub btnLoad_Click(. . .) Handles btnLoad.Click
    If System.IO.File.Exists("C:\applicants.txt") Then
        InputFileReader = System.IO.File.OpenText("C:\applicants.txt")
        OneFileLine = InputFileReader.ReadLine()
        Dim EndOfFile As Boolean = False

        ' Do While InputFileReader.Peek <> -1
        Do While EndOfFile <> True
            ReDim Preserve Player(ArrayCounter)
            Player(ArrayCounter) = New RugbyPlayer

            If InputFileReader.Peek = -1 Then
                EndOfFile = True
            End If

            SpacePos = OneFileLine.IndexOf(",")
            Player(ArrayCounter).Name = OneFileLine.Substring(0, SpacePos)
            OneFileLine = OneFileLine.Remove(0, SpacePos + 1)
            SpacePos = OneFileLine.IndexOf(",")
            Player(ArrayCounter).Surname = OneFileLine.Substring(0, SpacePos)
            OneFileLine = OneFileLine.Remove(0, SpacePos + 1)
            SpacePos = OneFileLine.IndexOf(",")
            Player(ArrayCounter).Gender = Convert.ToChar(OneFileLine.Substring(0, SpacePos))
            OneFileLine = OneFileLine.Remove(0, SpacePos + 1)
            SpacePos = OneFileLine.IndexOf(",")
            Player(ArrayCounter).Age = Convert.ToInt32(OneFileLine.Substring(0, SpacePos))
            OneFileLine = OneFileLine.Remove(0, SpacePos + 1)
            SpacePos = OneFileLine.IndexOf(",")
            Player(ArrayCounter).Weight = Convert.ToInt32(OneFileLine.Substring(0, SpacePos))
            OneFileLine = OneFileLine.Remove(0, SpacePos + 1)
            SpacePos = OneFileLine.IndexOf(",")
            Player(ArrayCounter).Position = OneFileLine.Substring(0, SpacePos)

            OneFileLine = InputFileReader.ReadLine()

            Me.lstDisplay.Items.Add(Player(ArrayCounter).Name & ControlChars.Tab &
                Player(ArrayCounter).Surname & ControlChars.Tab & _
                Player(ArrayCounter).Age & ControlChars.Tab & Player(ArrayCounter).Gender &
                ControlChars.Tab & _ Player(ArrayCounter).Weight & ControlChars.Tab &
                Player(ArrayCounter).Position)

            ArrayCounter += 1
        Loop
        InputFileReader.Close()
    Else
        MessageBox.Show("File could not be opened")
    End If
End Sub
End Class

```



## Exercises

- 1) Define a class named ProductItem. Include private fields that hold an item stock number, price quantity in stock, and total value.

Write public methods that each set the stock number, the price and quantity in stock, implement properties for some of the attributes if you want to.

The method that sets the stock number requires the user to enter five digits.

Include a private method that calculates the total value of the product item (price \* quantity in stock) and call it within the same method the accepts the quantity in stock input

A constructor which will initialize all the fields can also be included. Because all fields are private, include get methods for each field to be able to retrieve data from the class

- a. Write a program that declares a PRODUCT item object. The values to the TPRODUCT item fields should be entered using edit boxes and the program should call the appropriate set function giving it the correct argument. Before calling the function which set the stock number the user program must check if the number contains five digits, if not ,show an error message and require the user to enter the stock number again . When the display button is checked the object fields are displayed on a label. Add another button to clear the fields of the object to zero the clear button should also clear the necessary controls on the form
  - b. Create a new program which uses the same TPRODUCT class. Declare an array of PRODUCTS objects. The program should be able to go through all the array objects, set and display them using one set of controls, this can be made possible adding two buttons to move up and down the array. In addition, the clear button should be paralleled with a radio group with two options to select if only one or all items should be cleared.
- 2) You are required to define a class called FBPHomes to keep information on a block of homes. Each block has a number of homes, and a certain number of them are occupied .The rent per home and the levy is also included. The set values for blocksName , homesNumbers , occupiedHomes , rent and levy properties must be set after the classes has been instantiated(messages blocks). The user wants to use the class to keep record of the number of occupied homes in a block. He needs the following methods:

A SET method(setBlockName) to assign the name of the blocks

A GET method(getBlocksName) to access the name of the blocks

A SET method(setHomesNumber) to assign the number of homes in a block

A GET method(getBlocksNumber) to access the number of homes in a block

A SET method(setHomesRent) to assign the rent per home

A GET method(getHomesRent) to access the rent per home

A SET method(setHomesLevy) to assign the levy per home

A GET method(getHomesLevy) to access the levy per home

A SET method(setOccupiedHomesIn) to assign the moving in of occupied homes

A SET method(setOccupiedHomesOut) to assign the moving out of occupied homes

A Function method(SetgetTotalRent) to get the total Rent of all homes

A GET method(getOccupiedTotal) to access the occupied total of the homes

A method (calcPer) to calculate the percentage of the homes available

Develop a VB.NET application that will create multiple homes Objects which you can name on your own.

The application must keep information for the number of homes in a block as well as the rent payable the levy and the percentage occupied homes

Form1

## Block 2 - Happy Days

**Specify Block**

Which Block:  
Block 2 - Happy Day ▼

**General Information**

Occupied Homes 34

% of Homes occupied 34%

Total Homes income R 26350

**Submit Information**

Number of Homes:  
100

Payable Rent:  
650

Payable Levy:  
125

Set Values

**Block Information**

Available Homes:  
66

Payable Rent:  
650

Payable Levy:  
125

Get Values

**Update Block Information**

Occupy Home:  
34

Move Into Home

Move Out of Home

Close

## Inheritance

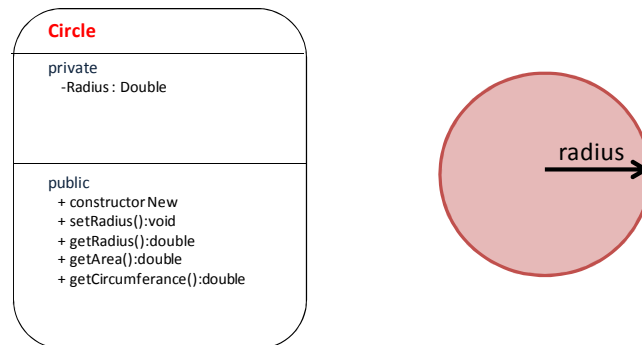
Inheritance is one of the cornerstone concepts of Object Orientation (OO). If a programming language does not support inheritance then it is not considered a true OO language.

Inheritance is a functionality of OOP that allows a programmer to develop a new class using an existing class as a startup. It could be seen in the same manner that a builder would use a foundation and build a house upon it. The builder adds new attributes to the foundation without changing the foundation itself.

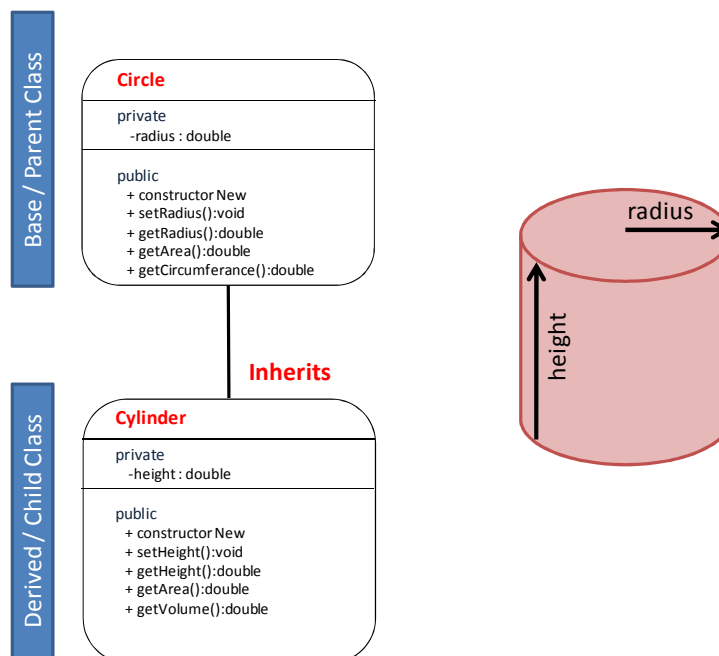
In other words: Inheritance is a feature of OOP that allows a developer to create new classes from existing classes. The new class extends the capacity/functionality of the original class.

### Example 6 – Inheritance: Circle -> Cylinder

Consider the class Circle with its various methods. It is possible for the class developer to derive a new class named Cylinder from the circle class. The class cylinder could use the Circle class as a base (or parent).



The derived or child class (Cylinder) inherits all the attributes and behavior of the parent class.

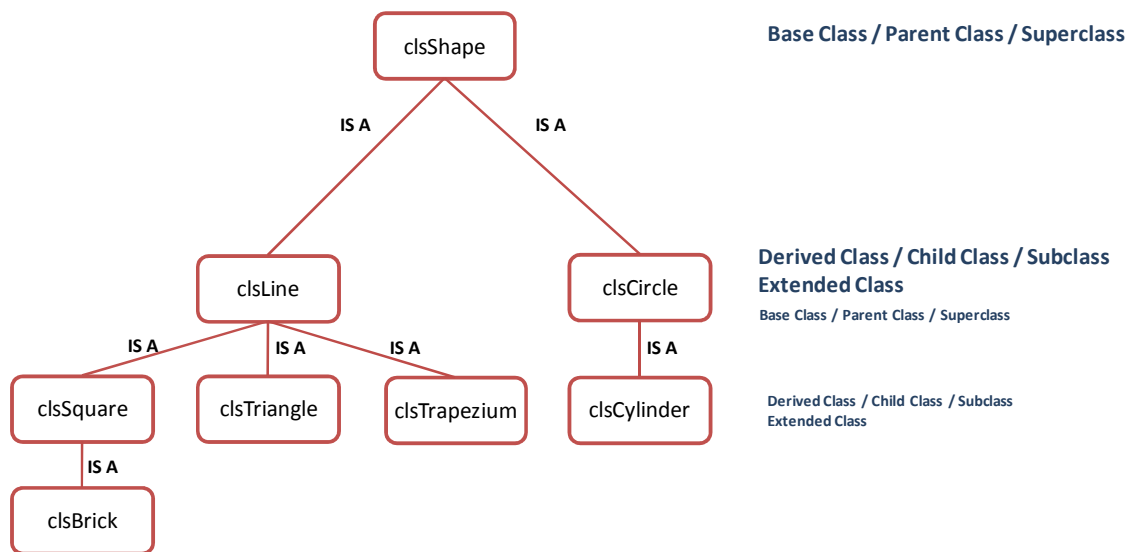


The class cylinder added new attributes and methods to the parent class Circle. All the public attributes and methods of the parent class circle are available for use as part of the child class. A child class only has access to the protected and the public attributes and methods of the parent class.

TAKE NOTE: Certain programming languages refer to the parent class as the super class and the child class as the sub class.

There is also a general rule to inheritance, it should always be possible to say that the child class: IS A parent.

A cylinder **IS A** circle, if it is not possible to state that the child class IS A of the parent then it is generally not considered to be inheritance.



## Example 7: Inheritance example using Circle class as the base class for the Cylinder class.

Study the following code which lists the class definition of a class named circle, and a derived class named cylinder which inherits from circle. The main implementation program instantiates 2 objects, a circle and a cylinder. The main application utilizes the objects to display data about each object.

The circle class

```

Public Class clsCircle
    Private _radius As Double

    Public Property Radius() As Double
        Get
            Return _radius
        End Get
        Set(ByVal value As Double)
            _radius = value
        End Set
    End Property

    Public Sub New()
        _radius = 0.1
    End Sub

```

```

Public Sub New(ByVal R As Double)
    _radius = R
End Sub

Public Function getArea() As Double
    Return Math.PI * _radius * _radius
End Function

Public Function getCircumference() As Double
    Return 2 * Math.PI * _radius
End Function
End Class

```

## The cylinder class

```

Public Class clsCylinder
    Inherits clsCircle

    Private _height As Double

    Public Property Height() As Double
        Get
            Return _height
        End Get
        Set(ByVal value As Double)
            _height = value
        End Set
    End Property

    Public Sub New()
        MyBase.New()
        _height = 1.0
    End Sub

    Public Sub New(ByVal R As Double, ByVal H As Double)
        MyBase.New(R)
        _height = H
    End Sub

    Public Function getVolume() As Double
        Return MyBase.getArea * _height
    End Function

    Public Overloads Function getArea() As Double
        Return (MyBase.getArea * 2) + (MyBase.getCircumference * _height)
    End Function
End Class

```

## The main implementation program

```

Public Class frmInheritance

    Private MyCircle As clsCircle
    Private MyCylinder As clsCylinder
    Private Sub frmInheritance_Load(. . .) Handles MyBase.Load
        lstData.Items.Clear()
    End Sub

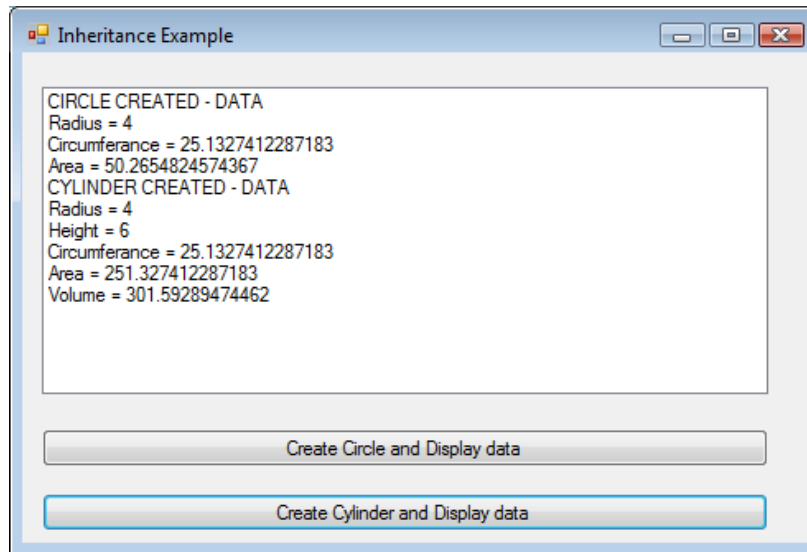
    Private Sub btnCircle_Click(. . .) Handles btnCircle.Click
        MyCircle = New clsCircle(4)
        lstData.Items.Add("CIRCLE CREATED - DATA")
        lstData.Items.Add("Radius = " + MyCircle.Radius.ToString)
        lstData.Items.Add("Circumference = " + MyCircle.getCircumference.ToString)
        lstData.Items.Add("Area = " + MyCircle.getArea.ToString)
    End Sub

    Private Sub btnCylinder_Click(. . .) Handles btnCylinder.Click
        MyCylinder = New clsCylinder(4, 6)
        lstData.Items.Add("CYLINDER CREATED - DATA")
        lstData.Items.Add("Radius = " + MyCylinder.Radius.ToString)
        lstData.Items.Add("Height = " + MyCylinder.Height.ToString)
        lstData.Items.Add("Circumference = " + MyCylinder.getCircumference.ToString)
        lstData.Items.Add("Area = " + MyCylinder.getArea.ToString)
        lstData.Items.Add("Volume = " + MyCylinder.getVolume.ToString)
    End Sub
End Class

```

```
End Sub
End Class
```

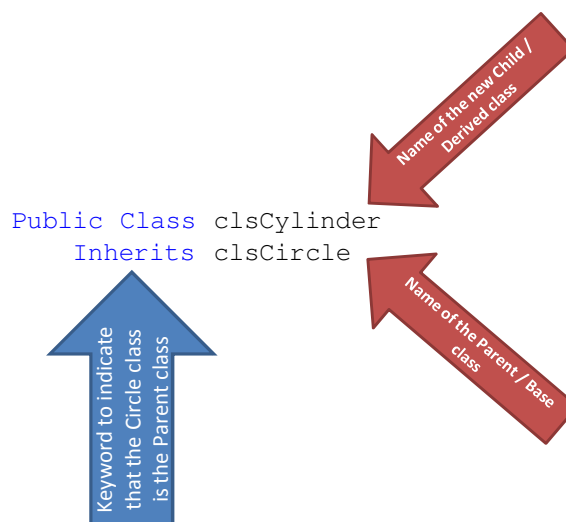
Sample run screen



### NOTES on the program

The definition of the class Circle is straightforward. Note the use of the Math.PI object to obtain the value of  $\pi$  (Pi).

The Inherits keyword which is used as part of the class definition of the Cylinder class indicates that the cylinder class is to inherit from the Circle class.



The constructors of the cylinder class calls the constructors of the parent class by using the MyBase keyword. The MyBase keyword indicates to the compiler to call the method from the Base or parent class.

```
Public Sub New()
    MyBase.New()
    _height = 1.0
End Sub
```

```
Public Sub New(ByVal R As Double, ByVal H As Double)
    MyBase.New(R)
    _height = H
End Sub
```

Calls the parameterized constructor from the Parent / Base class

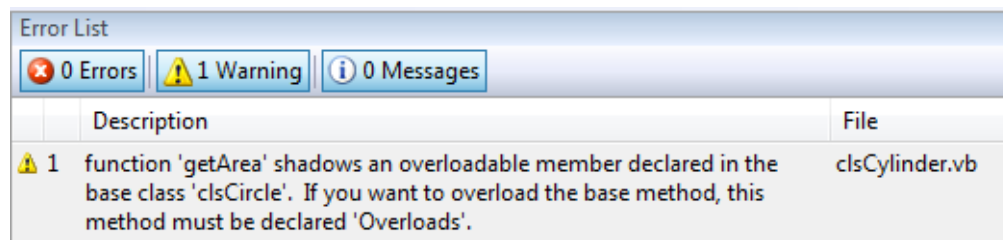
The MyBase keyword is often used in derived class methods. The getVolume and getArea methods in the Cylinder class call the getArea method of the parent class to calculate the Volume and the area of the Cylinder respectively.

```
Public Function getVolume() As Double
    Return MyBase.getArea * _height
End Function

Public Overloads Function getArea() As Double
    Return (MyBase.getArea * 2) + (MyBase.getCircumference * _height)
End Function
```

Take Note: The overloads keyword is added to the getArea method, this is to indicate to the compiler that the getArea of the Cylinder is implemented and calculated differently than the getArea of the Circle class.

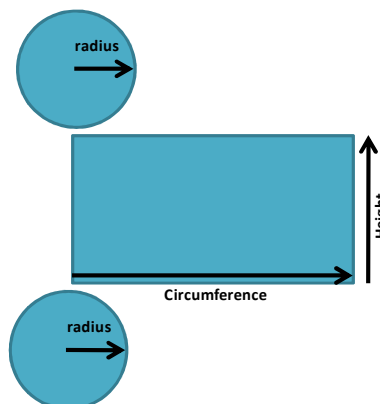
If the use programmer omitted the overloads keyword the compiler will give the following warning.



The keyword overloads indicates that functionality is added to the methods, where override indicates that the new method completely overwrites any previous definitions.

The Area of a Cylinder is calculated as:

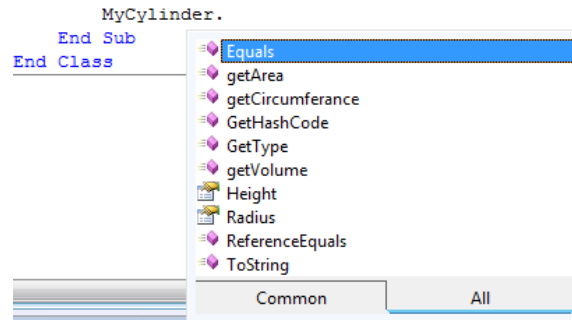
Twice the area of each circle + The area of the rectangular area.



The main implementation program instantiates a Circle and a Cylinder object. A Cylinder object named MyCylinder is instantiated with a radius value of 4 and a height of 6.

```
MyCylinder = New clsCylinder(4, 6)
```

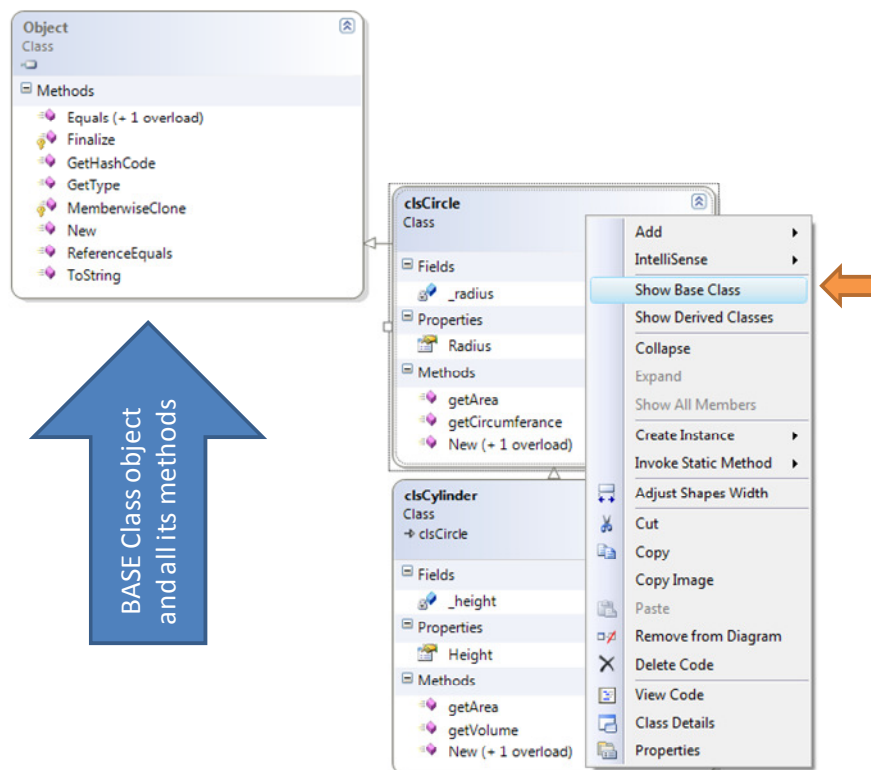
Observe which properties and methods are available to the user:



Take note that all the public methods and attributes of the parent class Circle is accessible to the programmer for use. You will also note that there are 4 additional methods which was not included in any of the definitions of either the Circle or Cylinder classes, they are:

- Equals
- GetHashCode
- Finalize
- ToString

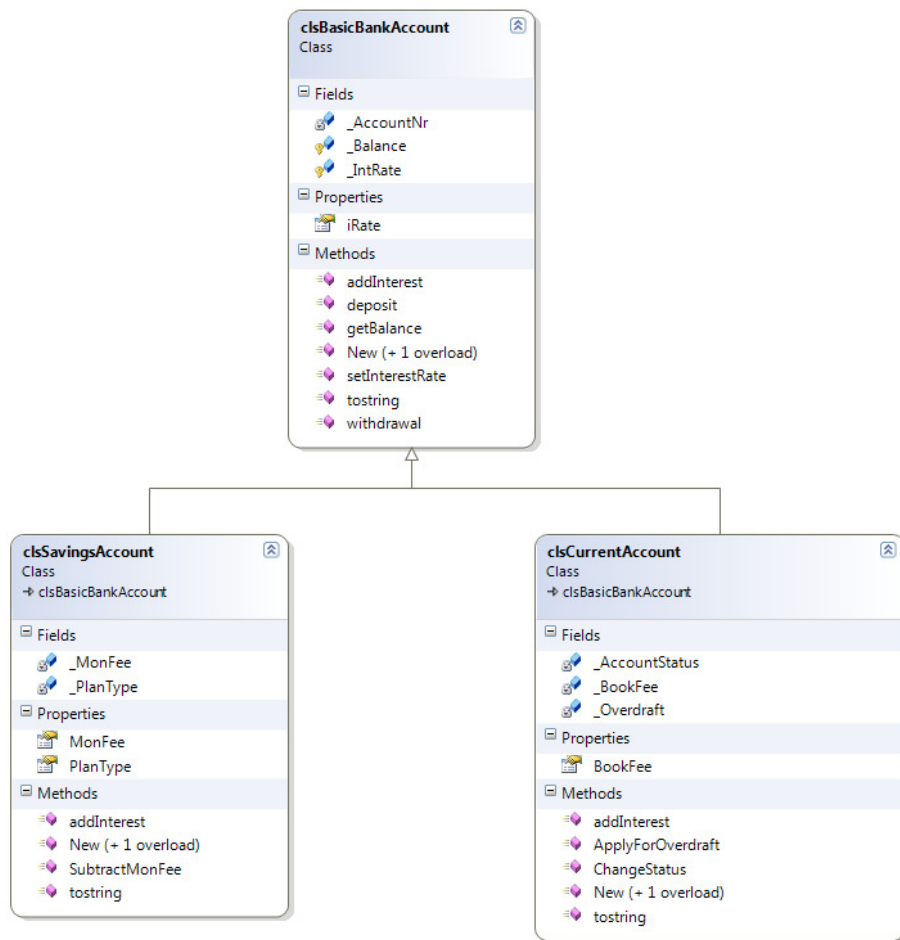
These four methods are actually methods that form part of the parent class of all objects, either user defined or build-in within the VB.NET framework. All new classes defined by a programmer automatically inherit from the base class Object.





## Example 8 Inheritance Class BankAccount

The following example uses the clsBasicBankAccount as a base class from which two subclasses (child classes) are derived. The following class diagram is supplied.



The SavingsAccount class adds a `_MonFee` and a `_PlanType` attribute which represents the monthly service fee and the type of the Account as a char, where  
B represents a Basic Save Savings Account and;  
C represents a Classic Save Savings Account.

The `addInterest` method is overloaded to subtract the monthly fee after the interest has been added. The overridden `toString` method returns the a string representing the current data of the object. The `SubtractMonFee` method subtracts the monthly account fee from the balance.

The CurrentAccount class adds three new attributes i.e.  
An account status field which indicates the status of the account where  
A represents a new account  
B an account that has a good track and payment record  
C an account that has an excellent track and payment record

A booking fee which indicates the cost of a chequebook to the issued, and  
Overdraft attribute which indicates the overdraft amount available in the account. The `addInterest` method calculates the monthly fee depending on the account type, and subtracts the value after the interest has been added. C type account holders receive R45.00 discount on their monthly fee where B type account holders receive R25.00.

The apply for overdraft method determines if the account holder qualifies for an overdraft C type account holders may have up to a maximum of R10000.00 overdraft where B type account holders may have up to a maximum of R5000.00. The status of the application is also returned.

The `_Balance` and `_IntRate` attributes of the parent class access specifier has been changed from private to protected, this is to allow the child classes to have access to the attributes as well.

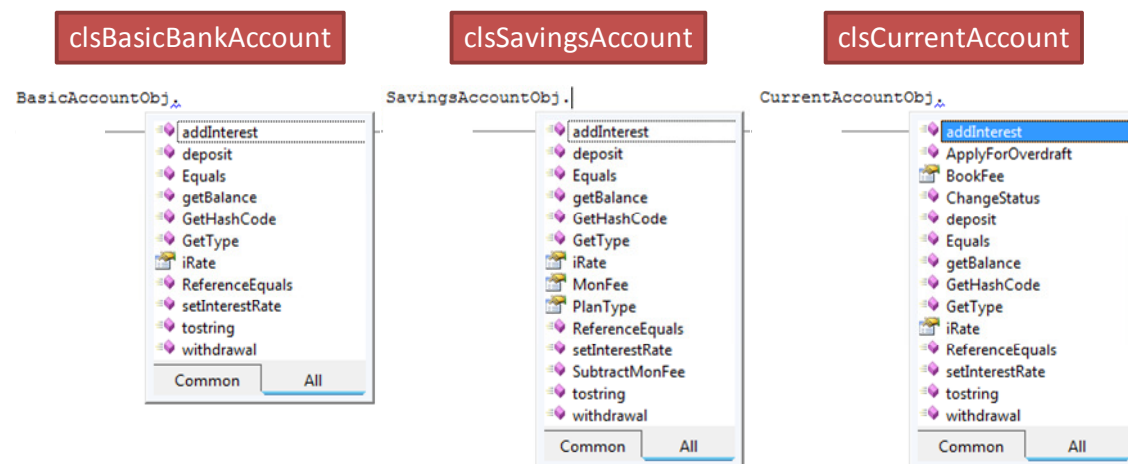
## Class Access Specifiers

Class Access Specifiers control what type of “visibility” an attribute, property or method has. We have seen that the privates of a class are only accessible to the current members of the class. VB.NET has other specifiers as well; the 3 most common ones are listed in the table below. Other access specifiers include, Friend and ProtectedFriend which is not part of the current content scope of the current discussion.

Access Specifier	Description
<b>Public</b>	This specifier indicates that the attribute, property or method is visible to all and places no restrictions on the accessibility. Attributes, properties, and methods are visible inside as well as on the outside of the class.
<b>Protected</b>	Protected indicates that the attribute, property or method is private to the outside of the class but visible to the children classes.
<b>Private</b>	All attributes, methods and properties marked with this spicier are only visible inside the current class. Private attributes stay hidden in derived classes, and are inaccessible to them.

TAKE NOTE: The children of a class only have access to the public’s and the protected attributes of the parent, the privates are inaccessible. All the protected members of the parent class are then treated as privates in the children class. All public attributes stay public. Classes can also be declared as private, public or protected but is functionality is outside of the scope of the current discussion.

The diagram below indicates the access that each of the objects have outside the class scope, i.e. the visibility to the user of the object.



Visible attributes, properties and methods accessible by the user/programmer from the Account class inheritance hierarchy.

Study the following code.

### Basic account class

```
Public Class clsBasicBankAccount
    Private _AccountNr As String
    Protected _Balance As Decimal
    Protected Shared _IntRate As Decimal

    Public Sub New()
        _AccountNr = ""
        _Balance = 0
        _IntRate = 0.1
    End Sub

    Public Sub New(ByVal AccNr As String, ByVal Bal As Decimal)
        _AccountNr = AccNr
        _Balance = Bal
        _IntRate = 0.1
    End Sub

    Public Shared ReadOnly Property iRate() As Decimal
        Get
            Return _IntRate
        End Get
    End Property

    Public Shared Sub setInterestRate(ByVal IR As Decimal)
        If (IR < 1) Then
            _IntRate = IR
        End If
    End Sub

    Public Function getBalance() As Decimal
        Return _Balance
    End Function

    Public Sub deposit(ByVal Amount As Decimal)
        _Balance = _Balance + Amount
    End Sub

    Public Function withdrawal(ByVal Amount As Decimal) As Boolean
        If (Amount < _Balance) Then
            _Balance = _Balance - Amount
            Return True
        Else
            Return False
        End If
    End Function

    Public Overrides Function toString() As String
        Return "Account: " + _AccountNr + " Interest rate: " + (iRate * 100).ToString
        + "% Balance: " + _Balance.ToString("N2")
    End Function

    Public Sub addInterest()
        Dim Interest As Decimal
        Interest = _Balance * (_IntRate / 12)
        _Balance = _Balance + Interest
    End Sub
End Class
```

### Savings Account class

```
Public Class clsSavingsAccount
    Inherits clsBasicBankAccount
    Private _MonFee As Double
    Private _PlanType As Char

    Public Property MonFee() As Double
        Get
            Return _MonFee
        End Get
    End Property
End Class
```

```

        Set(ByVal value As Double)
            _MonFee = value
        End Set
    End Property

    Public Property PlanType() As Char
        Get
            Return _PlanType
        End Get
        Set(ByVal value As Char)
            _PlanType = value
        End Set
    End Property

    Public Sub SubtractMonFee()
        _Balance = _Balance - _MonFee
    End Sub

    Public Sub New()
        MyBase.New()
        _MonFee = 75
        _PlanType = "B"
    End Sub

    Public Sub New(ByVal AccNr As String, ByVal Bal As Decimal, ByVal MFee As Double, ByVal
        PType As Char)
        MyBase.New(AccNr, Bal)
        _MonFee = MFee
        _PlanType = PType
    End Sub

    Public Overrides Function toString() As String
        Dim PlanName As String
        If _PlanType = "B" Then
            PlanName = "Basic Save "
        ElseIf _PlanType = "C" Then
            PlanName = "Classic Save "
        End If
        Return PlanName + MyBase.toString() + " Service Fee PM = " + _MonFee.ToString("N2")
    End Function

    Public Overloads Sub addInterest()
        Dim Interest As Decimal
        Interest = _Balance * (_IntRate / 12)
        _Balance = (_Balance + Interest)
        SubtractMonFee()
    End Sub
End Class

```

## Current Account class

```

Public Class clsCurrentAccount
    Inherits clsBasicBankAccount

    Private _BookFee As Decimal
    Private _Overdraft As Double
    Private _AccountStatus As Char

    Property BookFee() As Double
        Get
            Return _BookFee
        End Get
        Set(ByVal value As Double)
            _BookFee = value
        End Set
    End Property

    Public Sub New()
        MyBase.New()
        _BookFee = 100
        _Overdraft = 0
        _AccountStatus = "A"
    End Sub

```

```

Public Sub New(ByVal AccNr As String, ByVal Bal As Decimal, ByVal BFee As Double, ByVal
    OverDr As Double, ByVal AccStatus As Char)
    MyBase.New(AccNr, Bal)
    _BookFee = BFee
    _Overdraft = OverDr
    _AccountStatus = AccStatus
End Sub

Public Function ApplyForOverdraft(ByVal OverdaftApp As Double) As Boolean
    Dim Status As Boolean
    Status = False

    If _AccountStatus = "C" And (_Overdraft + OverdaftApp) <= 10000 Then
        _Overdraft = OverdaftApp
        Status = True
    End If
    If _AccountStatus = "B" And (_Overdraft + OverdaftApp) <= 5000 Then
        _Overdraft = OverdaftApp
        Status = True
    End If
    Return Status
End Function

Public Sub ChangeStatus(ByVal Status As Char)
    _AccountStatus = Status
End Sub

Public Overrides Function toString() As String
    Return "Current Account " + MyBase.toString() + "Overdraft " +
        _Overdraft.ToString("N2") + " Book Fee PM = " + _BookFee.ToString("N2")
End Function

Public Overloads Sub addInterest()
    Dim Interest As Decimal
    Dim AccFeeDiscount As Decimal
    Dim MFee As Decimal

    If _AccountStatus = "C" Then
        AccFeeDiscount = 45
    End If
    If _AccountStatus = "B" Then
        AccFeeDiscount = 25
    End If

    MFee = _BookFee - AccFeeDiscount
    Interest = _Balance * (_IntRate / 12)
    _Balance = (_Balance + Interest) - MFee

End Sub
End Class

```

## Implementation program

```

Public Class frmBankAccount

    Private BasicAccountObj As clsBasicBankAccount
    Private SavingsAccountObj As clsSavingsAccount
    Private CurrentAccountObj As clsCurrentAccount

    Private Sub btnDisplayAccounts_Click(. . .) Handles btnDisplayAccounts.Click
        lstOutput.Items.Add("ACCOUNT DETAILS LISTED")
        lstOutput.Items.Add(BasicAccountObj.toString())
        lstOutput.Items.Add(SavingsAccountObj.toString())
        lstOutput.Items.Add(CurrentAccountObj.toString())
    End Sub

    Private Sub frmBankAccount_Load(. . .) Handles MyBase.Load
        BasicAccountObj = New clsBasicBankAccount("03212434", 5648.17)
        SavingsAccountObj = New clsSavingsAccount("07625881", 25040.4, 25, "C")
        CurrentAccountObj = New clsCurrentAccount("07661345", 520.1, 15.5, 0, "A")
    End Sub

    Private Sub btnAccOperations_Click(. . .) Handles btnAccOperations.Click
        lstOutput.Items.Add("ACCOUNT OPERATIONS PERFORMED")
    End Sub
End Class

```

```

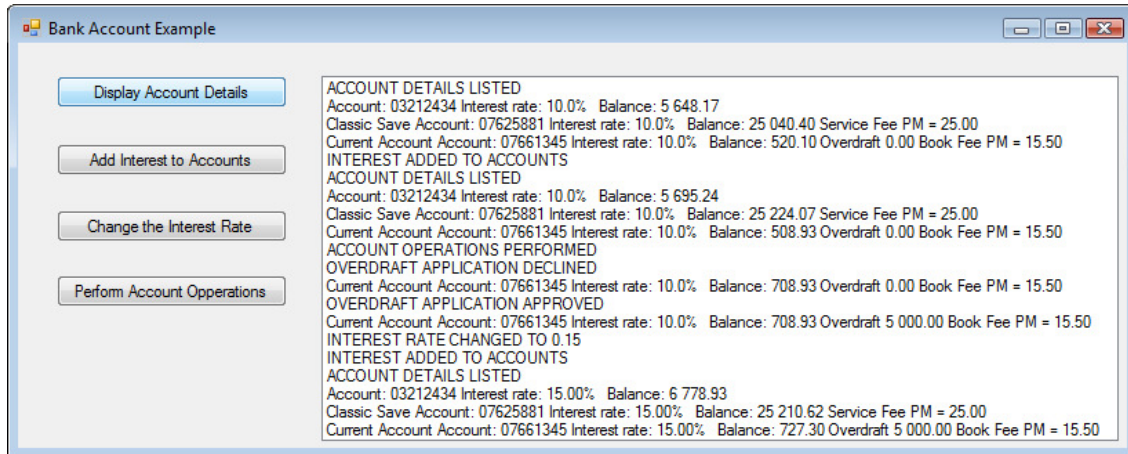
BasicAccountObj.deposit(1000)
CurrentAccountObj.deposit(200)
SavingsAccountObj.withdrawal(300)
If CurrentAccountObj.ApplyForOverdraft(5000) = True Then
    lstOutput.Items.Add("OVERDRAFT APPLICATION APPROVED")
    lstOutput.Items.Add(CurrentAccountObj.ToString())
Else
    lstOutput.Items.Add("OVERDRAFT APPLICATION DECLINED")
    lstOutput.Items.Add(CurrentAccountObj.ToString())
End If
CurrentAccountObj.ChangeStatus("B")
If CurrentAccountObj.ApplyForOverdraft(5000) = True Then
    lstOutput.Items.Add("OVERDRAFT APPLICATION APPROVED")
    lstOutput.Items.Add(CurrentAccountObj.ToString())
Else
    lstOutput.Items.Add("OVERDRAFT APPLICATION DECLINED")
    lstOutput.Items.Add(CurrentAccountObj.ToString())
End If

End Sub

Private Sub btnAddInterest_Click(. . .) Handles btnAddInterest.Click
    lstOutput.Items.Add("INTEREST ADDED TO ACCOUNTS")
    BasicAccountObj.addInterest()
    CurrentAccountObj.addInterest()
    SavingsAccountObj.addInterest()
End Sub

Private Sub btnChangeInterest_Click(. . .) Handles btnChangeInterest.Click
    Dim IR As Decimal
    IR = Decimal.Parse(InputBox("Enter the New Interest Rate", ""))
    clsBasicBankAccount.setInterestRate(IR / 100)
    lstOutput.Items.Add("INTEREST RATE CHANGED TO " &
        clsBasicBankAccount.iRate.ToString("N2"))
End Sub
End Class

```



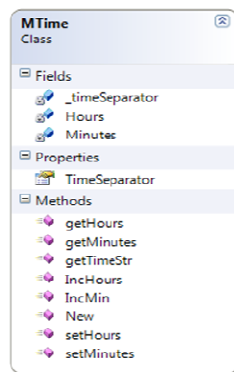
## Exercises

- 1) Using the MTime class as a parent class derive a class named ADVTime and include the following attributes and functionality:  
Add a second attribute and corresponding methods to set and get the seconds.  
The method IncSec should increase the value of the seconds by 1, adapting the minutes if applicable. Add a default and a parameterized constructor.

A Boolean method validateTime should validate if the time is valid (Take note our time object represents the time in military format)

Include an overridden toString method that will return the time in the format:

HH:MM:SS AM/PM      Where HH represents the time as (00 to 11)



Create an implementation application and show the functionality of your class.

- 2) Using the BasicBankAccount class hierarchy, include a derived class named clsCreditCardAccount. The creditcard account adds the following attributes and methods: An overloaded withdraw method which charges a fixed transaction fee of R2.10 plus 5% of the withdraw amount. This amount is added to the outstanding balance. The widthdraw method should validate that the widthdrawel amount is bigger than 0 and return a true, else a false should be returned. A withdrawel constitutes a transaction. The transaction fee is stored as one of the attributes of the class. An attribute should keep count of how many transactions have been done during the month. Depending on the number of transactions, the following discount is given on the monthly fee, which defaults to R42.50 per month. A method payment should receive a payment parameter which are to be de subtracted from the credit card balance.

0 . . 5 transactions	No discount
6 . . 19 transactions	5% discount on the monthly fee
20 . . 40 transactions	10% discount fee
40 or more transactions	15% discount fee

The monthly fee is added to the balance before the interest is added, and the numbers of transactions are set to 0. Interest for a credit card account amounts to 18% pa. Override the toString method for the class as well.

Create an implementation application and show the functionality of your class.

# Polymorphism

Is a feature of OOP that describes how various objects in the same object hierarchy respond to the same message. A true OO programming language should allow for polymorphism. Polymorphism can only "occur" with objects within the same inheritance hierarchy. The word polymorphism means many forms, or many "changeabilities".

Birds and Lions are objects from the same hierarchy (i.e. animal) – The message move would be interpreted differently by each type of animal.

Polymorphism is enabled by the concept of binding. Binding is the term programmers use to describe how the compiler knows about methods of objects, and where and when to call them. When a method call is made in an application the call must be associated to the method definition of the class.

*Early Binding (Static). [During compile time]*

With early binding each method call of an object is associated to the corresponding address of the class method in memory, when the application is linked and compiled.

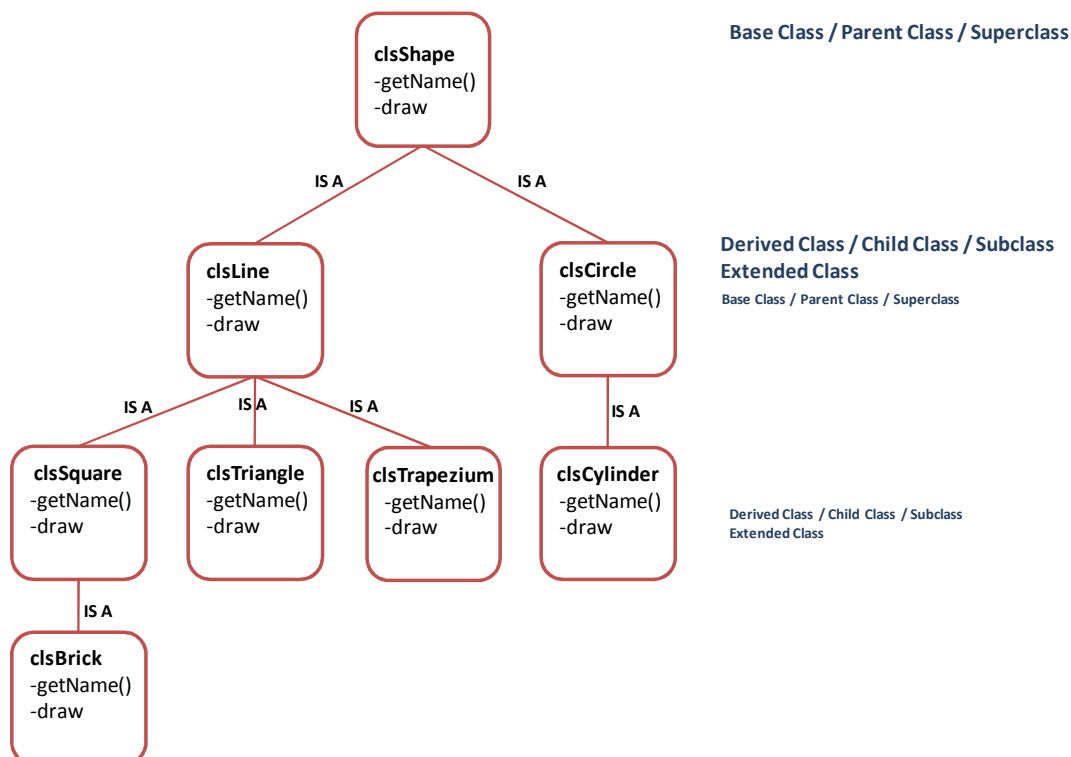
*Late Binding (Dynamic). [During run time]*

The link to the correct address of the method is only assigned during run time. This is because the object type can only be determined during runtime

Late Binding allows for polymorphism.

'The application developer can write method calls where the method call address can only be determined during runtime.'

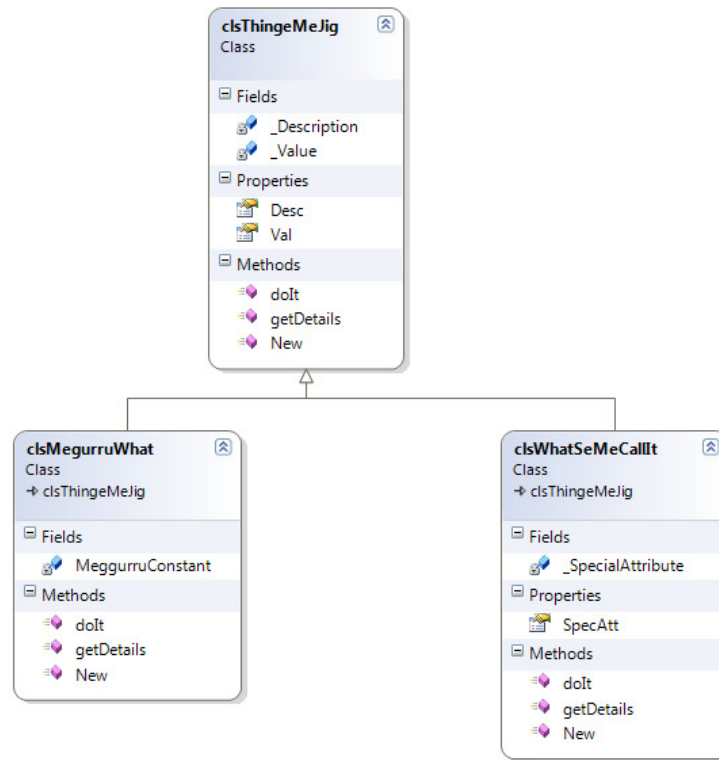
E.g. If two classes clsLine and clsCircle are derived from a parent class clsShape, and both classes has for example its own draw method, Treating a derived object as an object of the parent class will automatically (because of the address assigned to the object when it was created during runtime) call the correct draw method.





## Example 9 Polymorphism Class ThingeMeJig

Study the following class diagram and code provided.



Two child classes' `clsMegurruWhat` and `clsWhatSeMeCallIt` are derived from a parent class named `clsThingemejig`. Each child class adds one additional private attribute.

### Parent class `clsThingeMeJig`

```
Public Class clsThingeMeJig
    Private _Description As String
    Private _Value As Double

    Public Property Desc() As String
        Get
            Return _Description
        End Get
        Set(ByVal value As String)
            _Description = value
        End Set
    End Property

    Public Property Val() As Double
        Get
            Return _Value
        End Get
        Set(ByVal value As Double)
            _Value = value
        End Set
    End Property

    Public Sub New()
        _Description = "ThigeMeJig Parent"
        _Value = 10
    End Sub

    Public Overridable Function getDetails() As String
        Return _Description + " " + _Value.ToString("N2")
    End Function
End Class
```

```

        Public Overridable Function doIt() As Double
            Return _Value
        End Function
    End Class

```

### Derived class clsWhatSeMeCallIt

```

Public Class clsWhatSeMeCallIt
    Inherits clsThingMeJig
    Private _SpecialAttribute As String

    Public Property SpecAtt() As String
        Get
            Return _SpecialAttribute
        End Get
        Set(ByVal value As String)
            _SpecialAttribute = value
        End Set
    End Property

    Public Sub New()
        MyBase.Desc = "Whatsemecallit"
        MyBase.Val = 10
        _SpecialAttribute = " Special Attribute Value XYZ"
    End Sub

    Public Overrides Function getDetails() As String
        Return "WHATSEMECALLIT GETDAILS " + MyBase.getDetails + " " + _SpecialAttribute
    End Function

    Public Overrides Function doIt() As Double
        Return Val * 3
    End Function
End Class

```

### Derived class clsMegurruWhat

```

Public Class clsMegurruWhat
    Inherits clsThingMeJig
    Private MeggurruConstant As Integer

    Public Sub New()
        MyBase.Desc = "Megurruwhat"
        MyBase.Val = 4
        MeggurruConstant = 1234
    End Sub

    Public Overrides Function getDetails() As String
        Return "MEGURRUWHAT GETDAILS " + MyBase.getDetails + " " + MeggurruConstant.ToString
    End Function

    Public Overrides Function doIt() As Double
        Return Val * MeggurruConstant
    End Function
End Class

```

### Main implementation program

```

Public Class frmPolyExample

    Public MySelectedObject As clsThingMeJig

    Private Sub btnCreateObject_Click(. . .) Handles btnCreateObject.Click
        If ThingMeJigRadioButton.Checked Then
            MySelectedObject = New clsThingMeJig
            pnlData.Visible = True
        ElseIf MegurruWhatRadioButton.Checked Then
            MySelectedObject = New clsMegurruWhat
            pnlData.Visible = True
        ElseIf WhatSeMeCallItRadioButton.Checked Then
            MySelectedObject = New clsWhatSeMeCallIt
            pnlData.Visible = True
        End If
    End Sub

```

```

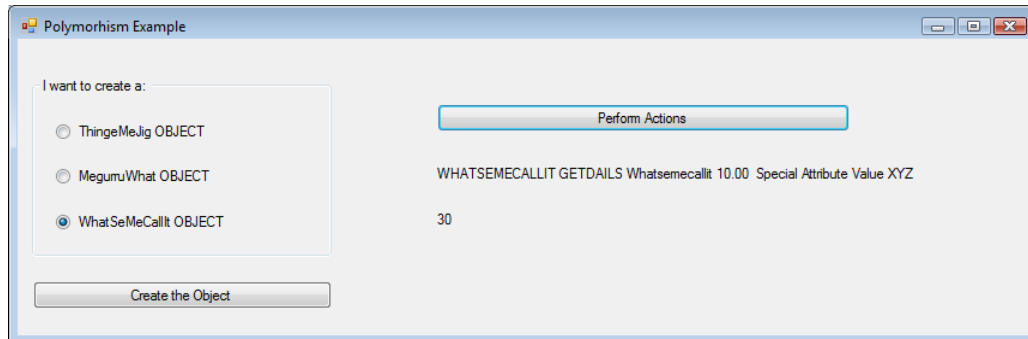
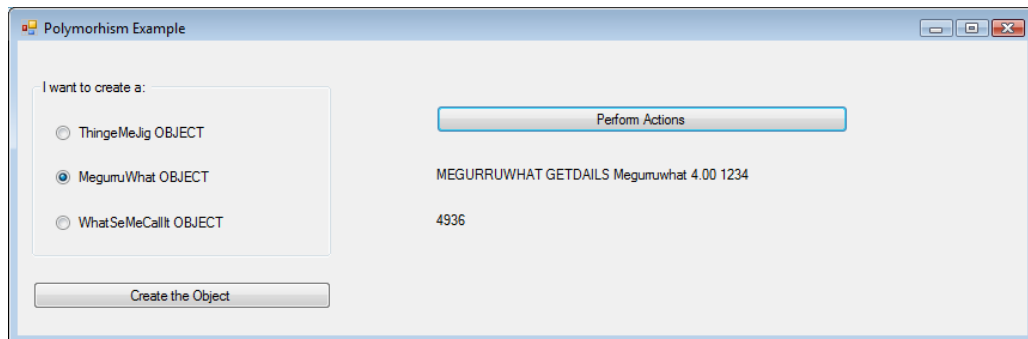
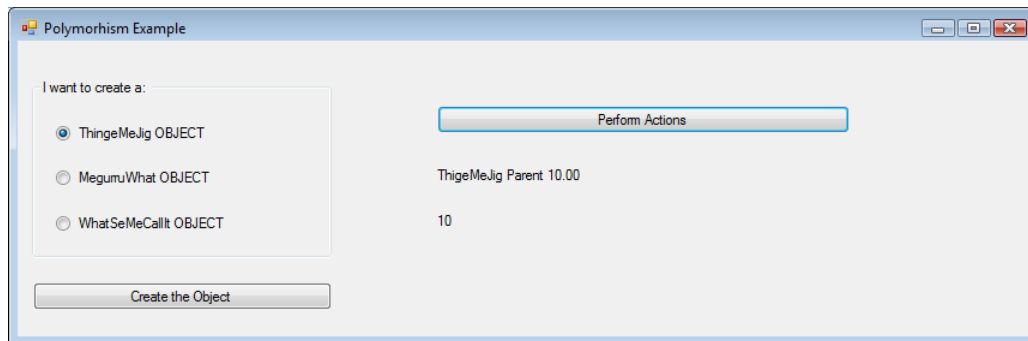
End Sub

Private Sub frmPolyExample_Load(. . .) Handles MyBase.Load
    pnlData.Visible = False
    lblDetails.Text = ""
    lblDoItValue.Text = ""
End Sub

Private Sub btnPerformActions_Click(. . .) Handles Button1.Click
    lblDetails.Text = MySelectedObject.getDetails
    lblDoItValue.Text = MySelectedObject.doIt.ToString
End Sub
End Class

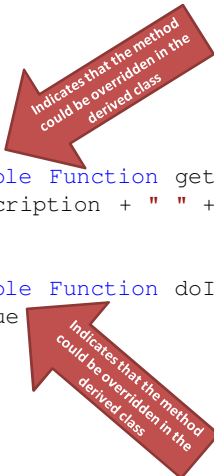
```

## Sample run screens



## NOTES on the program

The base class implements the `Overridable` keyword for both the `getDetails()` as well as the `doIt()` methods. The `Overridable` keyword indicates that the method could be overridden in the derived class. This implies that the developer of the child class may write a totally new implementation for the method with the same name.



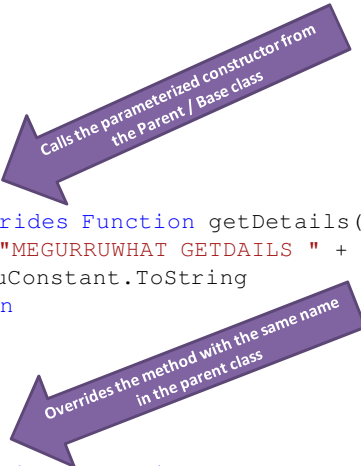
```
Public Overridable Function getDetails() As String
    Return _Description + " " + _Value.ToString("N2")
End Function

Public Overridable Function doIt() As Double
    Return _Value
End Function
```

Both the derived classes defined overridden `getDetails()` and `doIt()` methods, because the implementation of the methods are to differ.

Within the `clsWhatSeMcAllIt` class as well as the `clsMegurruWhat` class the `getDetails()` and `doIt()` has been overridden.

### Overridden Methods in the `clsMegurruWhat` child class



```
Public Overrides Function getDetails() As String
    Return "MEGURRUWHAT GETDAILS " + MyBase.getDetails + " " +
        MeggurruConstant.ToString
End Function

Public Overrides Function doIt() As Double
    Return Val * MeggurruConstant
End Function
```

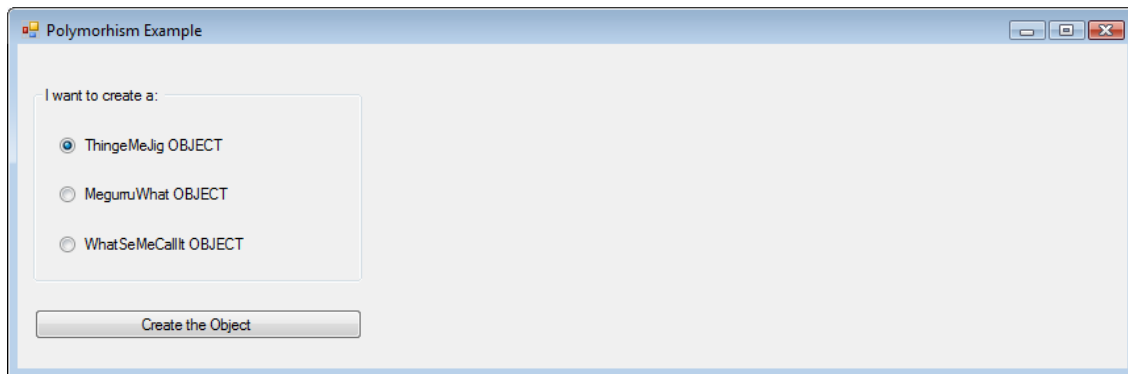
**TAKE NOTE:** If a method is marked with the `overloads` directive then, it implies that the overloaded function adds or extends the functionality of the base class method, whereas `overrides` completely redefine the method. Methods with the same name but different signatures are normally overloaded.

When the application is run the user is given the option to select which object he or she wants to instantiate. If you study the program carefully you will however note that only one object reference has been declared. i.e.:

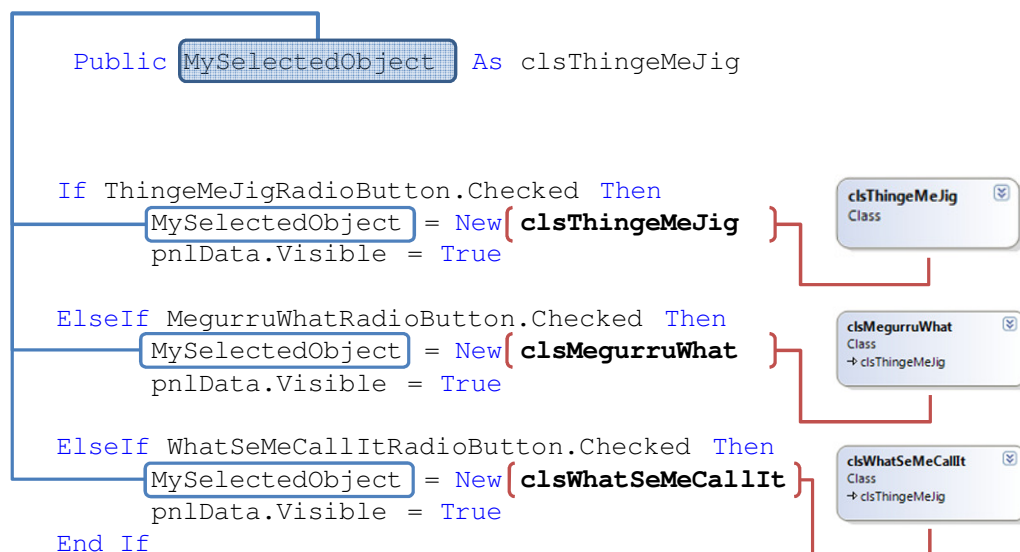
```
Public MySelectedObject As clsThingeMeJig
```

One of the features of polymorphism is that it allows a programmer to treat a derived class object as a parent class object, this is because of the inherent rule of inheritance which states that a child class should always be defined as an extension of the parent. In other words it should always be possible to state that a ChildClass object **IS A** ParentClass object as well.

On startup the initial application allows the user to choose which object he/she wants to create, the user indicates his/her choice by means of radiobuttons.



The [Create the Object] button event then instantiates the corresponding object. Observe the following code:



Depending on which option the user choose a corresponding object is created and assigned to an object reference variable of the parent class. Polymorphism allows it! In each case the MySelectedObject will reference a different object in the memory, this is because the address of where the chosen object is made is assigned to the object reference. i.e. Late Binding

After the object has been instantiated the user has the option to perform the various actions on the object. TAKE NOTE, each of the `doIt()` and `getDetails()` methods have their own implementation which differs from one another.

The `MySelectedObject` will either contain the address of a `ThingeMejig`, a `WhatSeMeCallIt` or a `MegurruWhat` Object. The subsequent methods that is invoked would be the correct method belonging to the actual object created.

```
Private Sub btnPerformActions_Click(...) Handles Button1.Click
```

```
lblDetails.Text = MySelectedObject.getDetails
```

The correct object reference address will refer to the user-selected object created

The correct corresponding method call will be invoked due to the late binding of the method address.

```
lblDoItValue.Text = MySelectedObject.doIt.ToString
```

```
End Sub
```

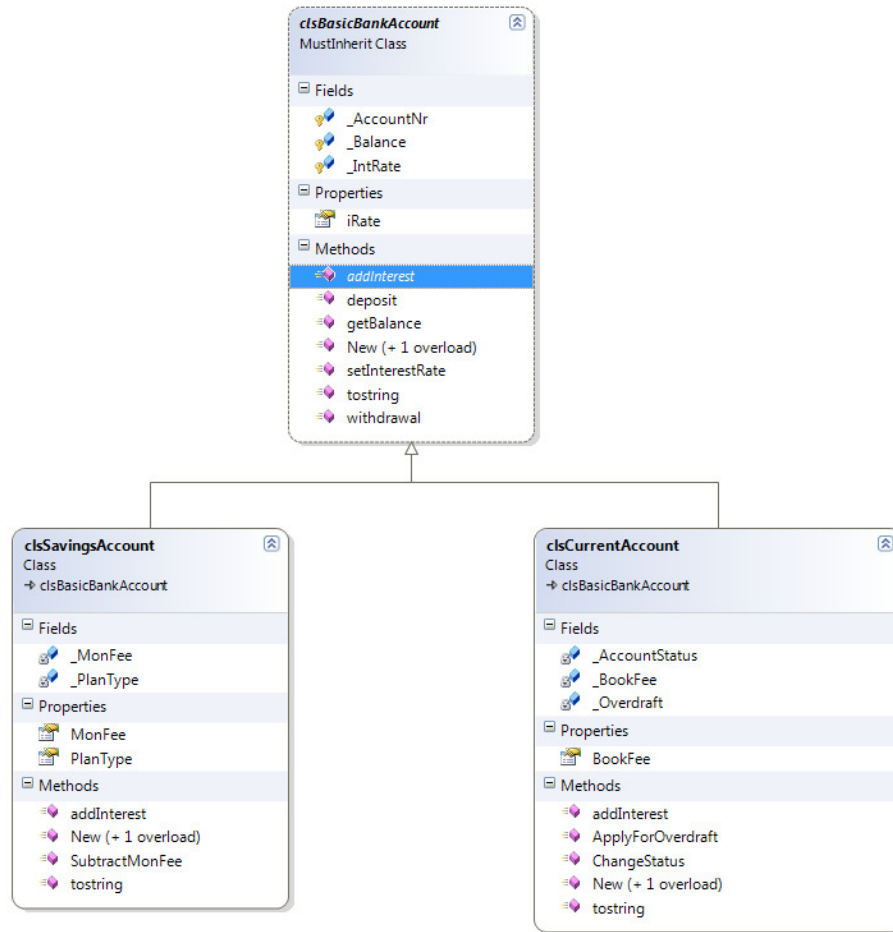
The correct object reference address will refer to the user-selected object created

The correct corresponding method call will be invoked due to the late binding of the method address.

The address of the correct method call is only known during runtime and the correct method is invoked with the correct object reference, i.e. POLYMORPHISM -> One Message many forms / "changeabilities"

## Example 10 Polymorphism – Class BankAccount

The second example is based on our BankingAccount Inheritance hierarchy.



For this example the `addInterest` and the `toString` methods of the parent class have been adapted to include polymorphic capabilities. This example introduces new VB.NET keywords; study the following code and sample run form and carefully read the explanations that follow.

### **BasicAccount Class – Parent Class**

```
Public MustInherit Class clsBasicBankAccount
    Protected _AccountNr As String
    Protected _Balance As Decimal
    Protected Shared _IntRate As Decimal

    Public Sub New()
        _AccountNr = ""
        _Balance = 0
        _IntRate = 0.1
    End Sub

    Public Sub New(ByVal AccNr As String, ByVal Bal As Decimal)
        _AccountNr = AccNr
        _Balance = Bal
        _IntRate = 0.1
    End Sub

    Public Shared ReadOnly Property iRate() As Decimal
        Get
            Return _IntRate
        End Get
    End Property
```

```

End Property

Public Shared Sub setInterestRate(ByVal IR As Decimal)
    If (IR < 1) Then
        _IntRate = IR
    End If
End Sub

Public Function getBalance() As Decimal
    Return _Balance
End Function

Public Sub deposit(ByVal Amount As Decimal)
    _Balance = _Balance + Amount
End Sub

Public Function withdrawal(ByVal Amount As Decimal) As Boolean
    If (Amount < _Balance) Then
        _Balance = _Balance - Amount
        Return True
    Else
        Return False
    End If
End Function

Public MustOverride Overrides Function toString() As String ' Abstract Method

Public MustOverride Sub addInterest() ' Abstract Method

End Class

```

### ***Savings Account Class***

```

Public Class clsSavingsAccount
    Inherits clsBasicBankAccount
    Private _MonFee As Double
    Private _PlanType As Char

    Public Property MonFee() As Double
        Get
            Return _MonFee
        End Get
        Set(ByVal value As Double)
            _MonFee = value
        End Set
    End Property

    Public Property PlanType() As Char
        Get
            Return _PlanType
        End Get
        Set(ByVal value As Char)
            _PlanType = value
        End Set
    End Property

    Public Sub SubtractMonFee()
        _Balance = _Balance - _MonFee
    End Sub

    Public Sub New()
        MyBase.New()
        _MonFee = 75
        _PlanType = "B"
    End Sub

    Public Sub New(ByVal AccNr As String, ByVal Bal As Decimal, ByVal MFee As Double, ByVal
        PType As Char)
        MyBase.New(AccNr, Bal)
        _MonFee = MFee
        _PlanType = PType
    End Sub

```



```

Public Overrides Function toString() As String
    Dim PlanName As String
    If _PlanType = "B" Then
        PlanName = "Basic Save "
    ElseIf PlanType = "C" Then
        PlanName = "Classic Save "
    End If
    Return PlanName + "Account: " + _AccountNr + " Interest rate: " + (iRate *
100).ToString + "% Balance: " + _Balance.ToString("N2") + " Service Fee PM = " +
_MonFee.ToString("N2")
End Function

Public Overrides Sub addInterest()
    Dim Interest As Decimal
    Interest = _Balance * (_IntRate / 12)
    _Balance = (_Balance + Interest)
    SubtractMonFee()
End Sub

End Class

Current Account Class

Public Class clsCurrentAccount
    Inherits clsBasicBankAccount

    Private _BookFee As Decimal
    Private _Overdraft As Double
    Private _AccountStatus As Char

    Property BookFee() As Double
        Get
            Return _BookFee
        End Get
        Set(ByVal value As Double)
            _BookFee = value
        End Set
    End Property

    Public Sub New()
        MyBase.New()
        _BookFee = 100
        _Overdraft = 0
        _AccountStatus = "A"
    End Sub

    Public Sub New(ByVal AccNr As String, ByVal Bal As Decimal, ByVal BFee As Double, ByVal
OverDr As Double, ByVal AccStatus As Char)
        MyBase.New(AccNr, Bal)
        _BookFee = BFee
        _Overdraft = OverDr
        _AccountStatus = AccStatus
    End Sub

    Public Function ApplyForOverdraft(ByVal OverdaftApp As Double) As Boolean
        Dim Status As Boolean
        Status = False

        If _AccountStatus = "C" And (_Overdraft + OverdaftApp) <= 10000 Then
            _Overdraft = OverdaftApp
            Status = True
        End If
        If _AccountStatus = "B" And (_Overdraft + OverdaftApp) <= 5000 Then
            _Overdraft = OverdaftApp
            Status = True
        End If
        Return Status
    End Function

    Public Sub ChangeStatus(ByVal Status As Char)
        _AccountStatus = Status
    End Sub

```

```

Public Overrides Function toString() As String
    Return "Current Account " + _AccountNr + " Interest rate: " + (iRate*100).ToString
    + "% Balance: " + _Balance.ToString("N2") + " Overdraft " +
    _Overdraft.ToString("N2") + " Book Fee PM = " + _BookFee.ToString("N2")
End Function

Public Overrides Sub addInterest()
    Dim Interest As Decimal
    Dim AccFeeDiscount As Decimal
    Dim MFee As Decimal

    If _AccountStatus = "C" Then
        AccFeeDiscount = 45
    End If
    If _AccountStatus = "B" Then
        AccFeeDiscount = 25
    End If

    MFee = _BookFee - AccFeeDiscount

    Interest = _Balance * (_IntRate / 12)
    _Balance = (_Balance + Interest) - MFee

End Sub
End Class

```

### ***Implementation Application***

```

Public Class frmBankAccount

    Dim MyAccountsArrayList = New ArrayList()

    Private Sub btnDisplayAccounts_Click(. . .) Handles btnDisplayAccounts.Click
        lstOutput.Items.Add("ACCOUNT DETAILS LISTED")
        Dim n As Integer

        For n = 0 To MyAccountsArrayList.Count - 1
            lstOutput.Items.Add(MyAccountsArrayList(n).ToString)
        Next n
    End Sub

    Private Sub frmBankAccount_Load(. . .) Handles MyBase.Load
        MyAccountsArrayList.Add(New clsCurrentAccount("03212434", 5648.17, 50, 0, "A"))
        MyAccountsArrayList.Add(New clsCurrentAccount("07625881", 25040.4, 50, 1000, "B"))
        MyAccountsArrayList.Add(New clsSavingsAccount("07661345", 520.1, 75, "C"))
        MyAccountsArrayList.Add(New clsSavingsAccount("07866335", 12540, 75, "B"))
    End Sub

    Private Sub btnAccOperations_Click(. . .) Handles btnAccOperations.Click
        lstOutput.Items.Add("ACCOUNT OPERATIONS PERFORMED")
        MyAccountsArrayList(0).deposit(1000)
        MyAccountsArrayList(1).deposit(200)
        MyAccountsArrayList(2).withdrawal(300)
    End Sub

    Private Sub btnAddInterest_Click(. . .) Handles btnAddInterest.Click
        lstOutput.Items.Add("INTEREST ADDED TO ACCOUNTS")
        Dim n As Integer
        For n = 0 To MyAccountsArrayList.Count - 1
            MyAccountsArrayList(n).addInterest()
        Next n
    End Sub

    Private Sub btnChangeInterest_Click(. . .) Handles btnChangeInterest.Click
        Dim IR As Decimal
        IR = Decimal.Parse(InputBox("Enter the New Interest Rate", ""))
        clsBasicBankAccount.setInterestRate(IR / 100)
        lstOutput.Items.Add("INTEREST RATE CHANGED TO " &
        clsBasicBankAccount.iRate.ToString("N2"))
    End Sub

```

```

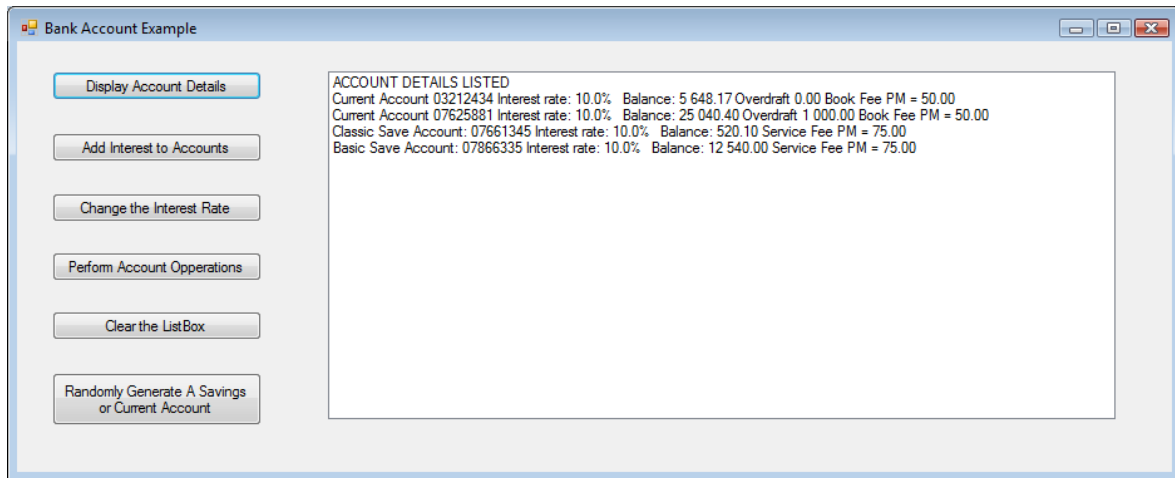
Private Sub btnNewAccObjectToArray_Click(. . .) btnNewAccObjectToArray.Click
    Dim RandomAccount As String
    Dim RandomType As Integer
    Dim random As New Random()
    Dim ElementCount As Integer
    RandomAccount = "07"
    RandomAccount = RandomAccount + random.Next(100000, 999999).ToString()
    RandomType = random.Next(1, 101)
    If RandomType <= 50 Then
        MyAccountsArrayList.Add(New clsCurrentAccount(RandomAccount, 1500, 50, 0, "A"))
    Else
        MyAccountsArrayList.Add(New clsSavingsAccount(RandomAccount, 100, 75, "B"))
    End If

    ElementCount = MyAccountsArrayList.Count
    MessageBox.Show("ACCOUNT CREATED : " + MyAccountsArrayList(ElementCount - 1).ToString)
End Sub

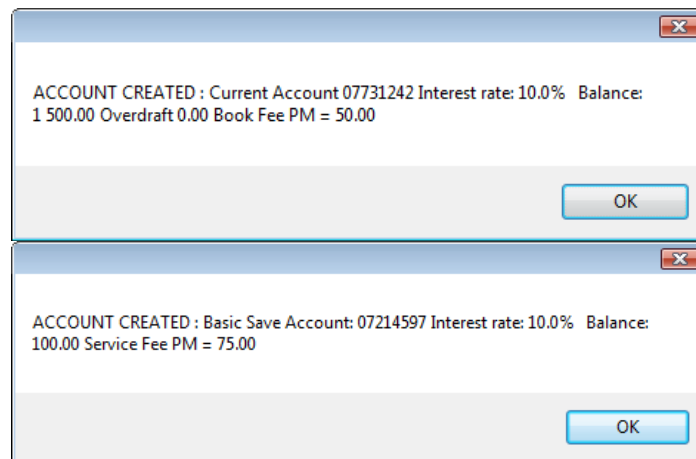
Private Sub btnClearListBox_Click_1(. . .) Handles btnClearListBox.Click
    lstOutput.Items.Clear()
End Sub
End Class

```

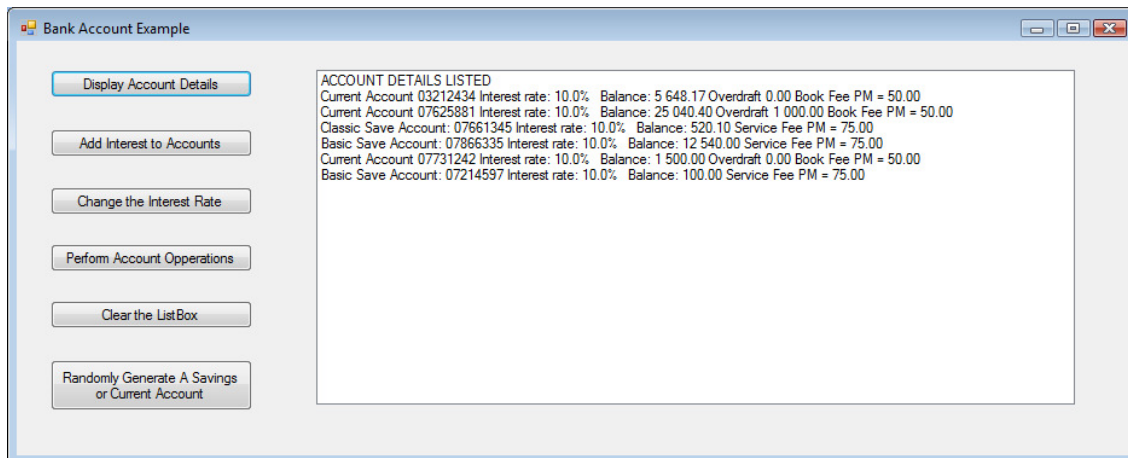
### Sample Run



### Generating 2 Random Accounts



### Displaying all the account details after the listbox has been cleared



### NOTES on the program

The basicbankaccount class introduces two new keywords, i.e.:

MustInherit  
MustOverride

These two keywords work hand in hand. The MustInherit Keyword indicates that a programmer must create a derived class from this class in order to use the functionality of the class.

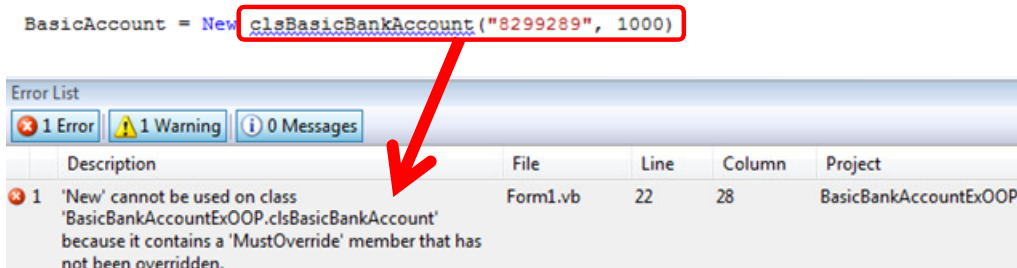
The MustOverride keyword is used to define an abstract method as part of a VB.NET class. An abstract method is one that does not have any implementation in the class where it is first included but it forces the developer of the derived class to include the implementation code of the specific method.

TAKE note: It is not possible to instantiate objects from a class that has been marked with the MustInherit directive.

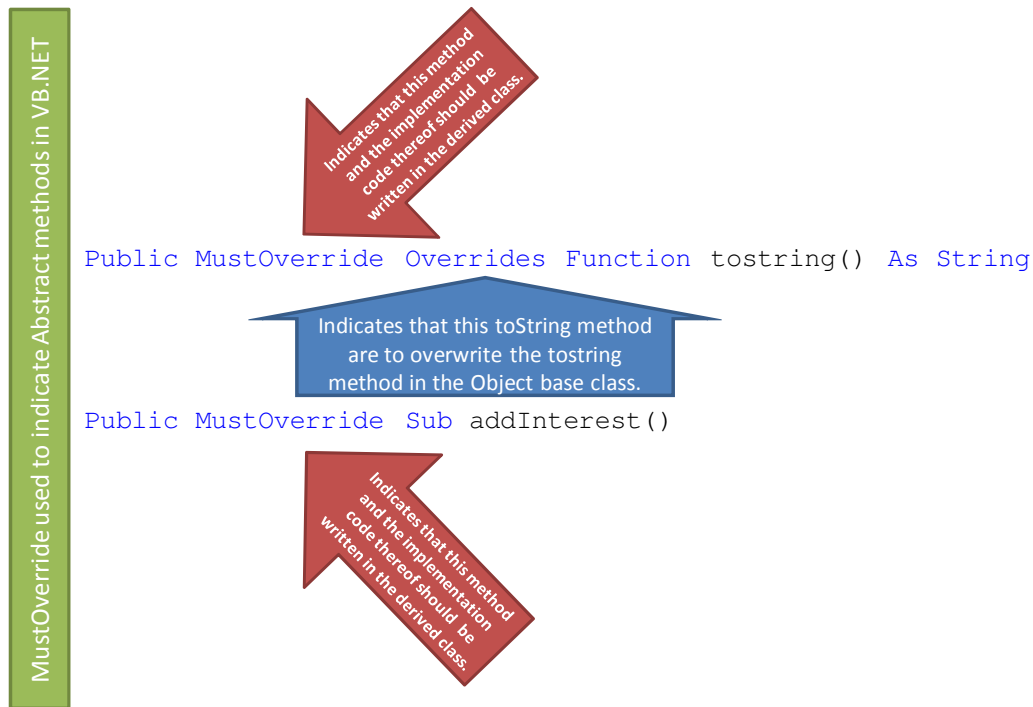
```
Public MustInherit Class clsBasicBankAccount
```

A child class must be derived from this class! No instances of a clsBasicBankAccount object may be created!

Attempting to instantiate a BasicAccount object will result in an error



The two abstract methods toString() and addInterest() contains no implementation code in the BasicAccount parent class.



TAKE note of the use of the overrides keyword implemented in both the method implementations of the addInterest and toString methods in the derived classes.

The keywords Overrides and MustOverride and Overridable are, used in Inheritance hierarchies to indicate methods with polymorphic capabilities.

The main implementation program declares an ArrayList container object to represent an array of account objects to be created.

```
Dim MyAccountsArrayList = New ArrayList()
```

The form load event is used to create 4 account objects which are added to the MyAccountsArrayList object.

```
MyAccountsArrayList.Add(New clsCurrentAccount("03212434", 5648.17, 50, 0, "A"))
MyAccountsArrayList.Add(New clsCurrentAccount("07625881", 25040.4, 50, 1000, "B"))
MyAccountsArrayList.Add(New clsSavingsAccount("07661345", 520.1, 75, "C"))
MyAccountsArrayList.Add(New clsSavingsAccount("07866335", 12540, 75, "B"))
```

The [Randomly Generate a Savings or Current Account] button generates either a Current account object or a Savings account object to be stored in the ArrayList container object. There is a 50 / 50 change of the code generating either a Savings or a Current account. Study the code below:

```
Dim RandomAccount As String
Dim RandomType As Integer
Dim random As New Random()
Dim ElementCount As Integer
RandomAccount = "07"
RandomAccount = RandomAccount + random.Next(100000, 999999).ToString()
RandomType = random.Next(1, 101)
If RandomType <= 50 Then
    MyAccountsArrayList.Add(New clsCurrentAccount(RandomAccount, 1500, 50, 0, "A"))
```

```

Else
    MyAccountsArrayList.Add(New clsSavingsAccount(RandomAccount, 100, 75, "B"))
End If

ElementCount = MyAccountsArrayList.Count
MessageBox.Show("ACCOUNT CREATED : " + MyAccountsArrayList(ElementCount - 1).ToString)

```

## BUT WHERE is the Polymorphism?

The polymorphic capabilities of this example resides in the fact that the type of the object that is created is determined during run-time. Invoking the overridden `toString` or `addInterest` methods for an account within the `AccountsArrayList` will automatically call the correct method depending on the type of the object created.

The inherent feature of polymorphism becomes more evident if the `ArrayList` object are changed to use a reference array of `BasicAccount` Objects.

## Adapted Example Using an Array of Objects

```

Public Class frmBankAccount

    Private AccountsArray(3) As clsBasicBankAccount

    Private Sub btnDisplayAccounts_Click(. . .) Handles btnDisplayAccounts.Click
        lstOutput.Items.Add("ACCOUNT DETAILS LISTED")
        Dim n As Integer

        For n = 0 To AccountsArray.Length - 1
            lstOutput.Items.Add(AccountsArray(n).toString)
        Next n
    End Sub

    Private Sub frmBankAccount_Load(. . .) Handles MyBase.Load
        AccountsArray(0) = New clsCurrentAccount("03212434", 5648.17, 50, 0, "A")
        AccountsArray(1) = New clsCurrentAccount("07625881", 25040.4, 50, 1000, "B")
        AccountsArray(2) = New clsSavingsAccount("07661345", 520.1, 75, "C")
        AccountsArray(3) = New clsSavingsAccount("07866335", 12540, 75, "B")
    End Sub

    Private Sub btnAccOperations_Click(. . .) Handles btnAccOperations.Click
        lstOutput.Items.Add("ACCOUNT OPERATIONS PERFORMED")
        AccountsArray(0).deposit(1000)
        AccountsArray(1).deposit(200)
        AccountsArray(2).withdrawal(300)
    End Sub

    Private Sub btnAddInterest_Click(. . .) Handles btnAddInterest.Click
        lstOutput.Items.Add("INTEREST ADDED TO ACCOUNTS")
        Dim n As Integer
        For n = 0 To AccountsArray.Length - 1
            AccountsArray(n).addInterest()
        Next n
    End Sub

    Private Sub btnChangeInterest_Click(. . .) Handles btnChangeInterest.Click
        Dim IR As Decimal
        IR = Decimal.Parse(InputBox("Enter the New Interest Rate", ""))
        clsBasicBankAccount.setInterestRate(IR / 100)
        lstOutput.Items.Add("INTEREST RATE CHANGED TO " &
            clsBasicBankAccount.iRate.ToString("N2"))
    End Sub

    Private Sub btnNewAccObjectToArray_Click(. . .) Handles btnNewAccObjectToArray.Click
        Dim RandomAccount As String
        Dim RandomType As Integer
        Dim random As New Random()
        Dim ElementCount As Integer
        RandomAccount = "07"
        RandomAccount = RandomAccount + random.Next(100000, 999999).ToString()
    End Sub
End Class

```

```

RandomType = random.Next(1, 101)
ElementCount = AccountsArray.Length
ReDim Preserve AccountsArray(ElementCount)

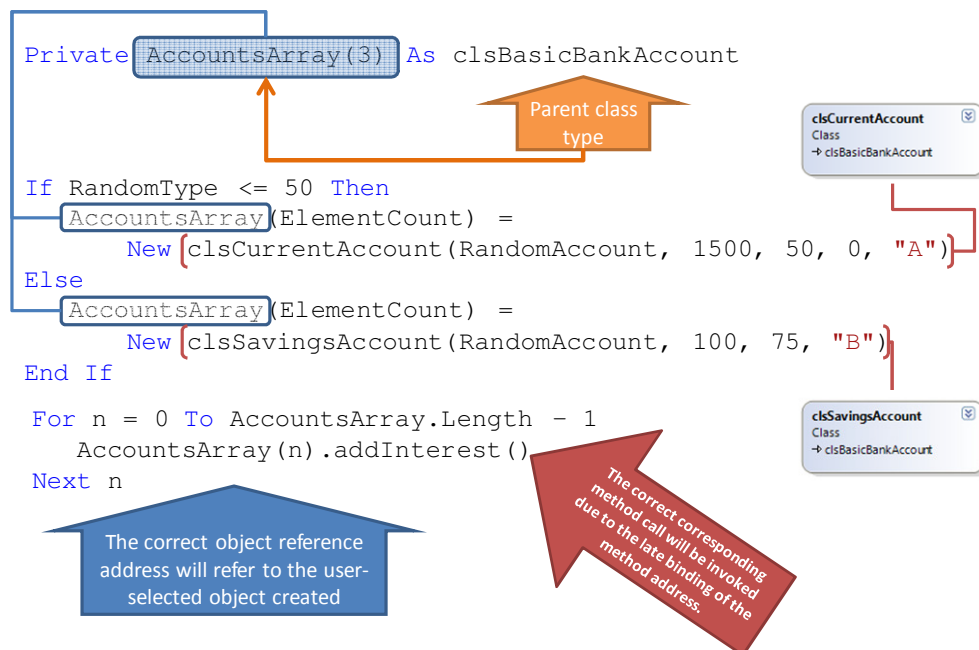
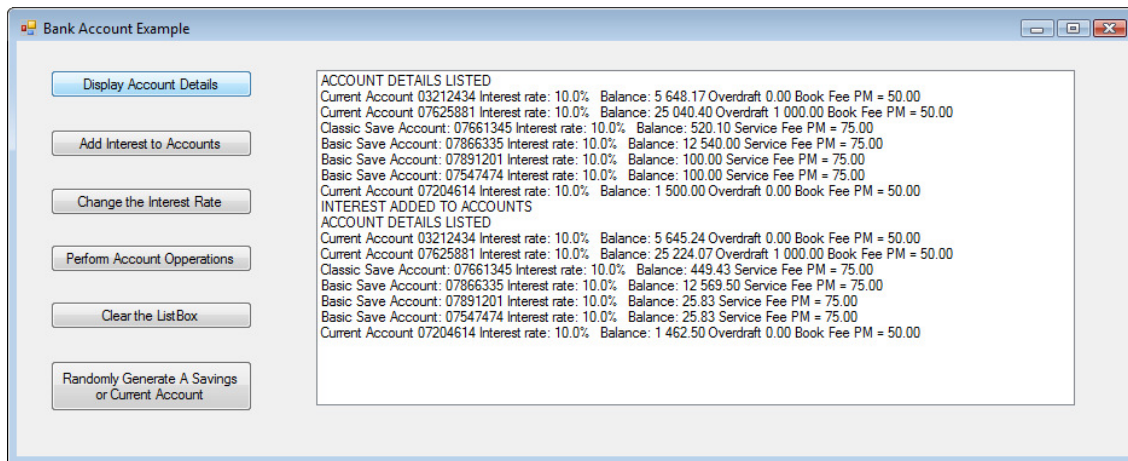
If RandomType <= 50 Then
    AccountsArray(ElementCount) = New clsCurrentAccount(RandomAccount, 1500, 50, 0, "A")
Else
    AccountsArray(ElementCount) = New clsSavingsAccount(RandomAccount, 100, 75, "B")
End If

MessageBox.Show("ACCOUNT CREATED : " + AccountsArray(ElementCount).ToString())

End Sub

Private Sub btnClearListBox_Click_1(. . .) Handles btnClearListBox.Click
    lstOutput.Items.Clear()
End Sub
End Class

```



An object reference of the created object is stored in an array of the parent type, which is permissible in a true inheritance hierarchy. When the user invokes a method the correct corresponding method of the created class is invoked. The program automatically "knows" which method to call due to the Late Binding of the object address and corresponding method call addresses. This is Polymorphism!

## Polymorphism (and Inheritance) Comprehensive Exercise

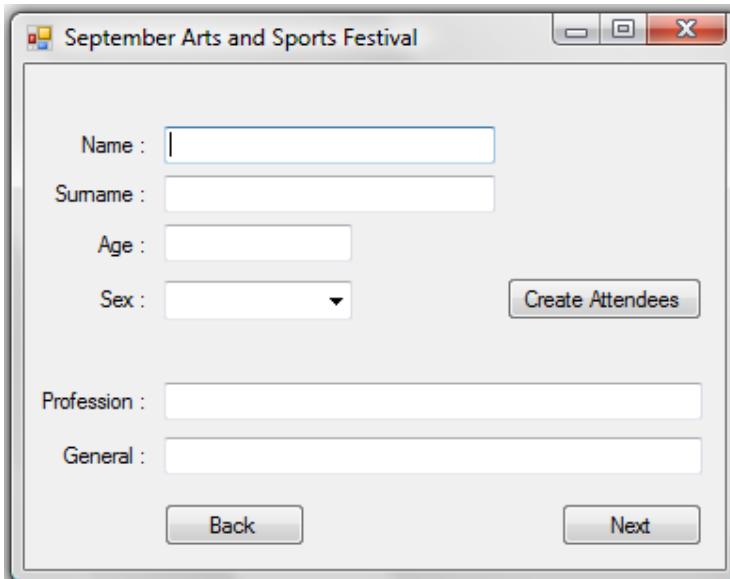
### Background

The September Sports And Arts Festival requires a small application they can use to view attendees to the festival as well as their claim to fame. The festival is held each year and focuses mainly on fund raising for the less privileged communities and schools.

You are required to write an application that will allow the hosts of the festival to browse through the attendees. You may design the form to your liking, but it must include the following controls:

- Textbox that will display the Name of the attendee.
- Textbox that will display the Surname of the attendee.
- Textbox that will display the Age of the attendee.
- Combobox that will display the sex of the attendee.
- Textbox that will display the profession of the attendee
- Textbox that will display general information about the attendee.
- Three (3) Button controls for the following functions:
  - Creating a new list of attendees (array)
  - Next and Back button to traverse the attendee array.

Below is an example what a form might look like:



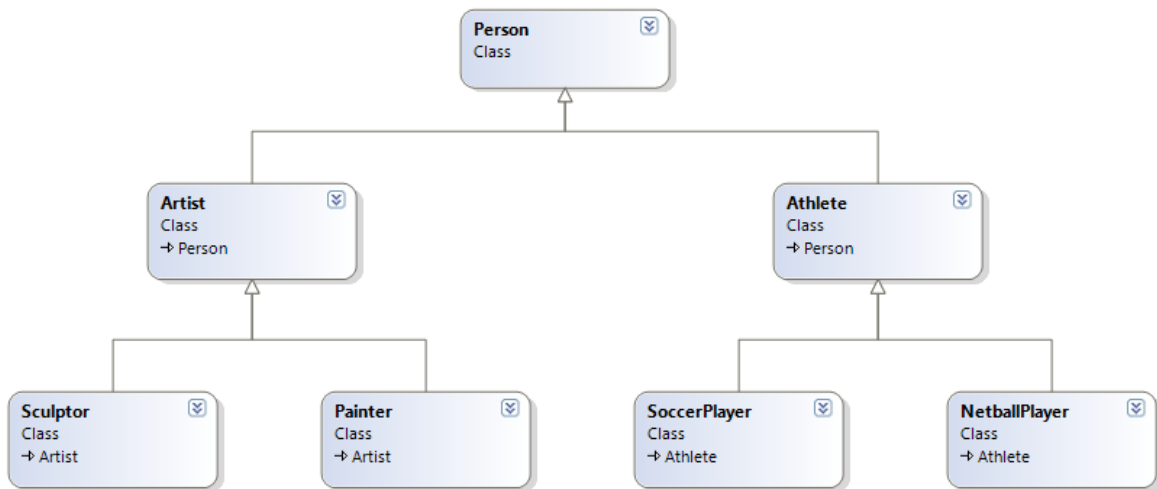
The screenshot shows a Windows application window titled "September Arts and Sports Festival". The window contains a form with the following controls:

- Name :** A text input field.
- Surname :** A text input field.
- Age :** A text input field.
- Sex :** A combobox (dropdown menu).
- Profession :** A text input field.
- General :** A text input field.
- Create Attendees**: A button located to the right of the Sex combobox.
- Back**: A button at the bottom left.
- Next**: A button at the bottom right.

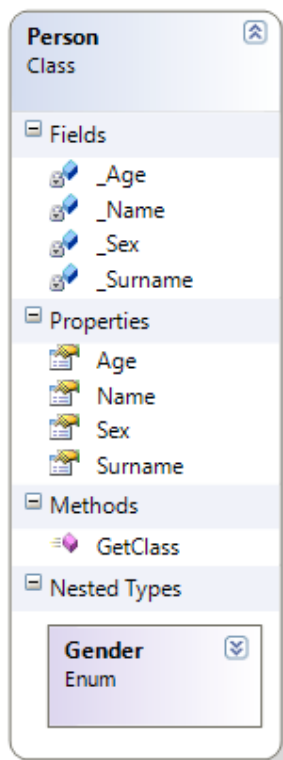


## Class Design

This solution requires that you design the classes to meet the following hierarchy:



- Design a class named **Person** that has the following Public Properties and Enumerations:



- A Public Property, `Name`, that returns a String
- A Public Property, `Surname`, that returns a String
- A Public Property, `Sex` of enumeration type `Gender`
- A Public Property, `Age`, that returns a Integer

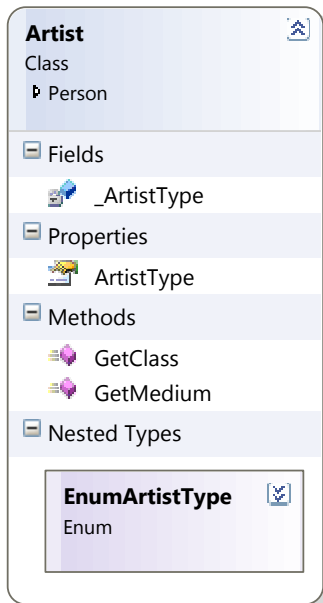
Use properties to eliminate the use of set and get methods when designing the application.

Create a Public Overridable Function `GetClass()` that return a string representation of the class name.

Create the following enumeration for `Gender`:

```
Public Enum Gender As Integer
    Male
    Female
End Enum
```

- Design a class named Artist which Inherits Person as its parent class. The following Functions, Properties and Enumerations are required:

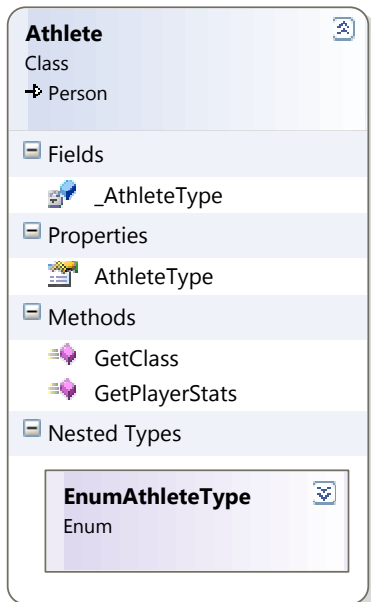


- Create a Public Property of enumeration type EnumArtistType called ArtistType
- Override the GetClass() function to display the current class.
- A Public Overridable Function GetMedium which returns a String

Create the following enumeration:

```
Public Enum EnumArtistType As Integer
    Painter
    Sculptor
End Enum
```

- Design a class named Athlete which Inherits Person as its parent class. The following Functions, Properties and Enumerations are required:

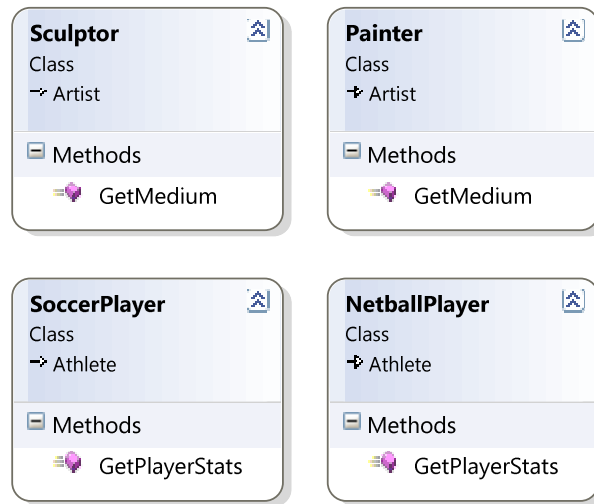


- Create a Public Property of enumeration type EnumAthleteType called AthleteType
- Override the GetClass() function to display the current class.
- A Public Overridable Function GetPlayerStats which returns a String

Create the following enumeration:

```
Public Enum EnumAthleteType As Integer
    Soccer
    Netball
End Enum
```

- Create the classes Sculptor, Painter, NetballPlayer and SoccerPlayer. These classes inherit from the relevant parent classes Artist or Athlete. See the hierarchy if you are uncertain.



- Sculptor and Painter overrides GetMedium and SoccerPlayer and NetballPlayer overrides GetPlayerStats.
  - As an example, the medium for a sculptor might be clay, a painter uses oil paint.
  - A soccer player might have goals scored in career as a statistic, where a similarly suitable statistic might be available for a netball player.

## Putting It Together

- In your main form, create objects of the class Sculptor, Painter, SoccerPlayer and NetballPlayer.
- Create an array of type Person with a size of 4 elements.
- Create two sub's (CreateArtists(), CreateAthletes()) that will provide values for the properties suitable to each object, as an example, consider the following code that creates a Painter:

```

painter = New Painter()
painter.Age = 23
painter.Name = "Louise"
painter.Surname = "Franc"

painter.Sex = InheritanceAndPolymorphism.Person.Gender.Female
painter.ArtistType = Artist.EnumArtistType.Sculptor
  
```

NOTE : In this code sample, InheritanceAndPolymorphism refers to the Project name.

- Add each object to the Person array.
- Write code that will allow you to traverse the Person Array, below is a skeleton function you may use:

```
Private Sub DisplayArtistsAndAthletes(ByVal index As Integer)
```

- Add code to the Next and Back button's event listeners to traverse the Person array. Call the skeleton function from the event listeners passing it the current position. Consider the code fragment below for a general idea of decreasing the position:

```
If (position > -1) Then
```

```

        If position <> 0 Then
            position -= 1
        End If
        DisplayArtistsAndAthletes(position)
    End If

```

- Write the code body for the DisplayArtistAndAthletes sub. This will require you to use Polymorphism to retrieve the information of each object and display it in the textboxes. The combobox may be prepopulated with the values Male and Female and set via SelectedIndex.

For you to successfully retrieve the information of each object you will have to take in consideration that some Properties and attributes are only accessible for certain class objects. As an example GetMedium is only accessible to Athlete objects or inherited objects and cannot be seen by Artist classes.

You will have to use Type checks and Type casting to retrieve some information.

To check whether an object is a type of a certain class you can use the following code:

```

If TypeOf <variable> Is <class> Then

```

An example:

```

If TypeOf person(index) Is Artist Then

```

Finally, to access the properties of these classes you will first have to Cast it to the relevant type. To cast you can use the following code:

```

CType(<variable>), <type_to_cast_to>)

```

An example:

```

txtSpeciality.Text = CType(person(index), Artist).GetMedium().ToString()

```

# Introduction to ADO Databases in VB.NET



**JOHAN KIRSTEN © 2009**

# Database application Development

## Using Microsoft Visual Basic 2008 Express Edition

Notes compiled by: [Johan Kirsten](#)

### Introduction

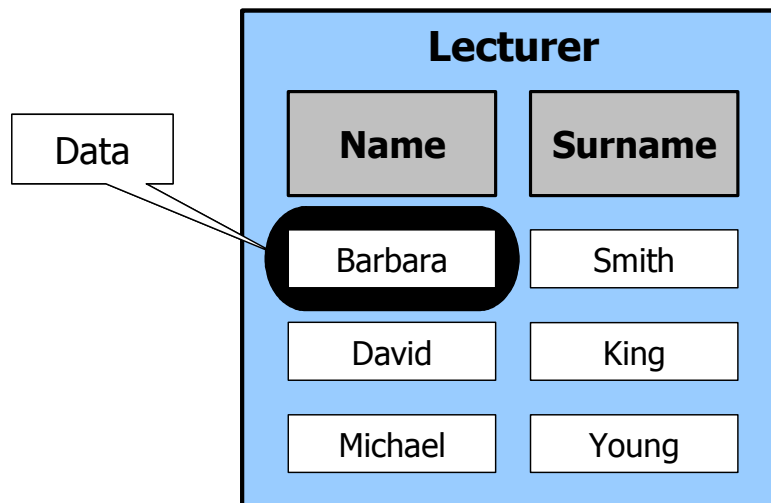
Welcome to the database applications chapter where we will discuss the exciting topic of building data-driven software applications. A database application is a powerful business tool used for day-to-day operations or to project important business decisions by analyzing data stored within a database. During this chapter, we will discuss topics related to databases and software applications implementing the features of databases. Amongst these are the following:

- Database concepts and terminology common to database programming.
- Using Microsoft ADO.NET to connect to databases and utilize the data.
- Implementing error handling and flow control into a database application.
- Validating data in order to ensure data accuracy and integrity
- Using other programming techniques with data driven application development.

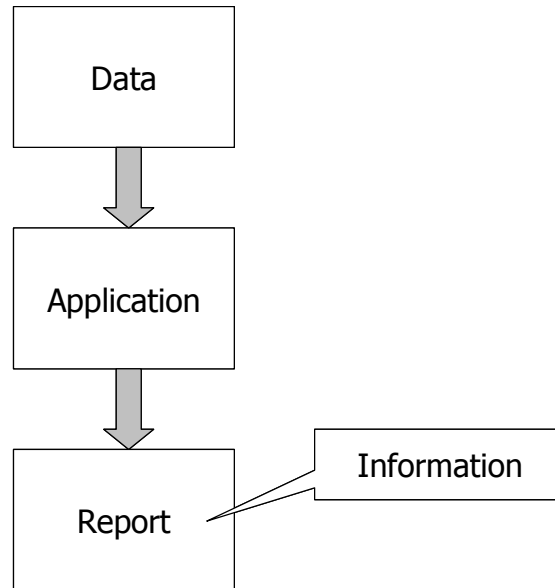
### Database concepts

In order to successfully build data driven applications, it is very important to be acquainted with the basic terminology used in data driven applications. During this section, we will look at some terminology used in this chapter. It is very important to understand these terms as these are the basic building blocks for a successful database application developer. Please take the time to review the terminology and feel free to reference back to this section in the future.

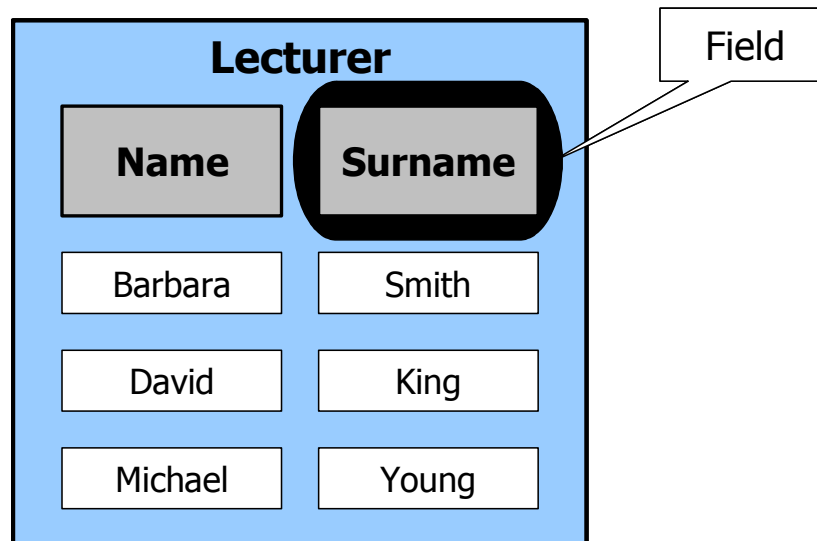
- 1) Data – are raw facts that do not mean anything by itself, but with the proper manipulation it is a valuable business asset. For instance, your name is a piece of data, but it does not hold any value by itself. If you combine your name and surname together, you are able to identify yourself properly to somebody else, giving those two pieces of data much more purpose than being on their own.



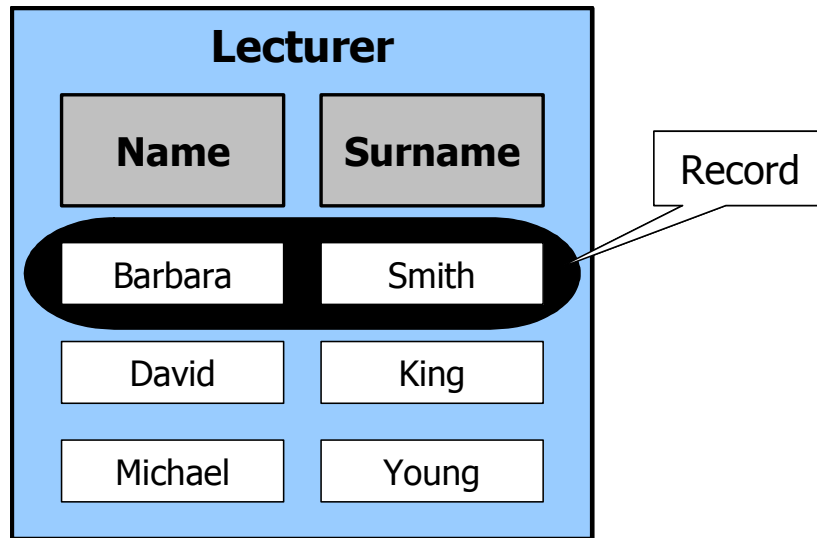
- 2) Information – is data in a processed form. Information is the strategic combination and manipulation of data so that it has value to the stakeholder in possession of the particular data. An example of information is the financial statements of a company. The content of the financial statements is derived from the data gathered throughout the year and thus forms a useful report which may be used to determine the financial stance of the company.



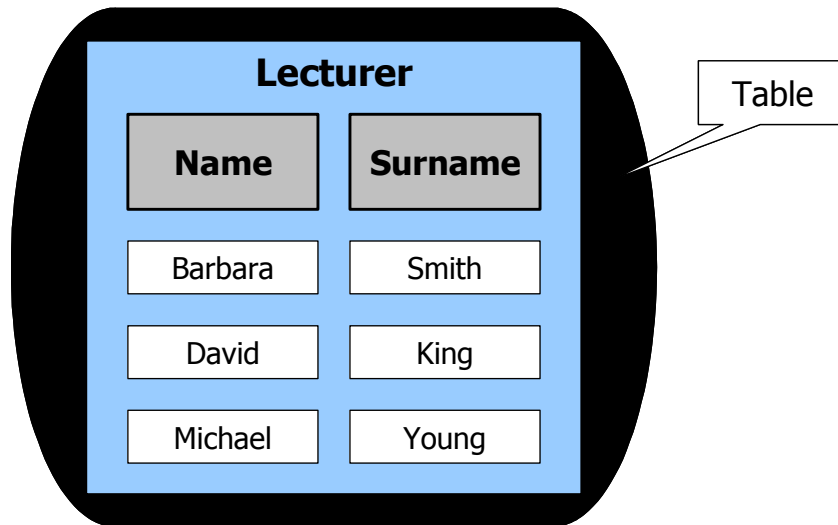
- 3) Field – within a database system, a piece of data is referred to as a field. For instance, the "Surname" is called a field and stores the data which should be something like the following, "Smith". In other words, a piece of data is labelled so that we may identify it, and we call this type of identification a field.



- 4) Record – is the combination of more than one field. By carefully combining certain fields, it is possible to describe an entity with the data available. For instance, if the record contains the "Name" and "Surname", a lecturer can easily be described and identified using the data.

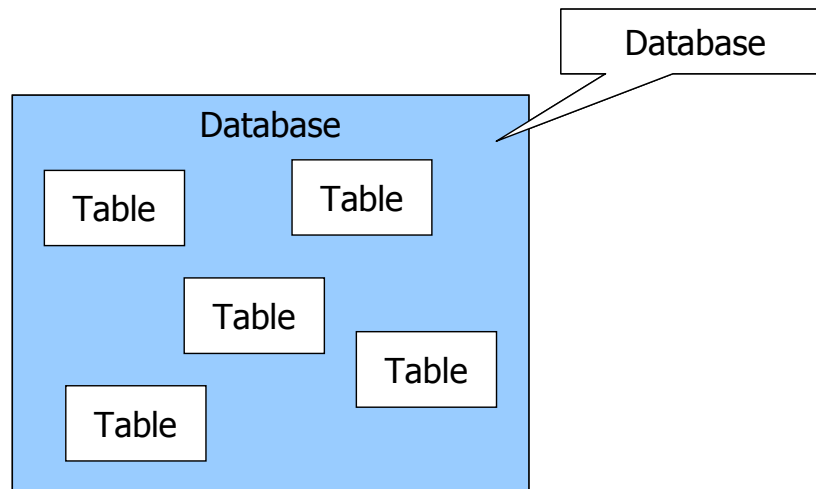


- 5) Table – a table is a collection of related records. A list of lecturers' mutual pieces of data arranged into records can be grouped together to form a table.

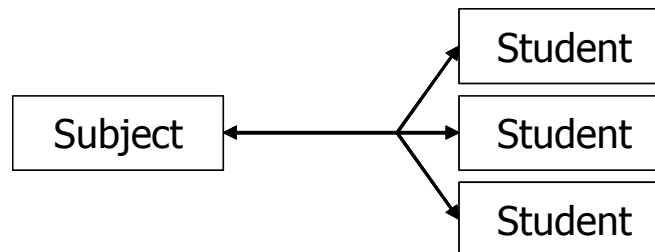


- 6) Database – a database is a collection of tables and various other parts of data driven functions, but for our purposes it is good enough to look at only the tables. A collection of tables describing related entities forms a database of raw facts that may be used to perform business processes.

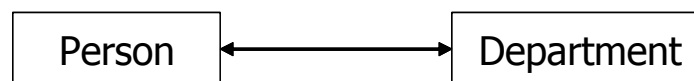




- 7) Database Management System – is a software system providing the functionality to manage the database content such as tables, record, fields and data stored within the database.
- 8) Database relationships – provides an association between two different, but related entities. For instance, a lecturer schools many students. This is an example of a database relationship. It provides a link between the "Lecturer" and "Student" entities and ensures data integrity between the two entities. For the purpose of this text, we will only look into two types of database relationships:
  - a. One-to-many (1 : M) – this type of database relationship, indicates that for every one entity on the left, there is more than one entity on the right. For instance, for every one "Subject" a group of "Students" may enrol.



- b. One-to-one (1 : 1) – this type of database relationship, indicates that for every one entity on the left, there may only be one entity on the right. For instance, a department within the college may only have one head of department, and a person may only be head of department over one department at a time. This means that at any given time, a "Person" will be linked to only one "Department".



- 9) ERD – Entity Relationship Diagram – this diagram is used to draw the database components and the relationships between them in a graphical manner. Typically the ERD will contain tables, relationships between the tables, the definition or list of fields within the table and also the types of relationships between the different tables.
- 10) SQL – Structured Query Language – provides a manner to communicate in a standardized way to the contents within a database. SQL uses specific syntax and keywords to perform

duties. Please refer to the appendix for a detailed walkthrough on understanding SQL and the usage thereof.

- 11) Class – is the collection of related properties or fields within a database table describing a specific group of related entities. The class provides behavioural functionality called methods or functions to manipulate data and perform specific actions. Please refer to the appendix for a detailed walkthrough on classes and the usage thereof.

## **Microsoft ADO.NET**

Microsoft ADO.NET is a collection of classes provided with the Microsoft .NET Framework for software developers to build data driven software applications. ADO.NET consists of data access and manipulation classes. Data access classes provide the necessary functionality to connect to a data source and communicate with the data source. Data manipulation classes provide the functionality to utilize the data retrieved from the data source within a software application. Please study the following section to gain a better understanding on how Microsoft ADO.NET implements the aforementioned classes.

### **Data access**

To access the data stored within a database file or database system, we need to connect to the database file or database system programmatically. Microsoft provides various options to connect to these types of database files or database systems. During this section, we will look at some of the options which are:

1. ODBC .NET Data Provider
2. OLE DB .NET Data Provider
3. Microsoft SQL Server .NET Data Provider
- 4.

#### **ODBC .NET Data Provider**

The ODBC .NET Data Provider is an implementation of the ODBC standard developed in earlier years to connect to data sources such as Microsoft Access, Microsoft Excel and even plain text files using a standardized set of software libraries. ODBC is useful for connecting to many types of databases and is still in wide usage today.

#### **OLE DB .NET Data Provider**

The OLE DB .NET Data Provider is an upgraded and improved version of the ODBC standard developed by Microsoft. This data provider harnesses the power and versatility of the Microsoft technologies and software libraries to build more efficient and better software applications. OLE DB provides better support for a wider range of data sources which are not supported or implemented in the ODBC standard.

#### **Microsoft SQL Server .NET Data Provider**

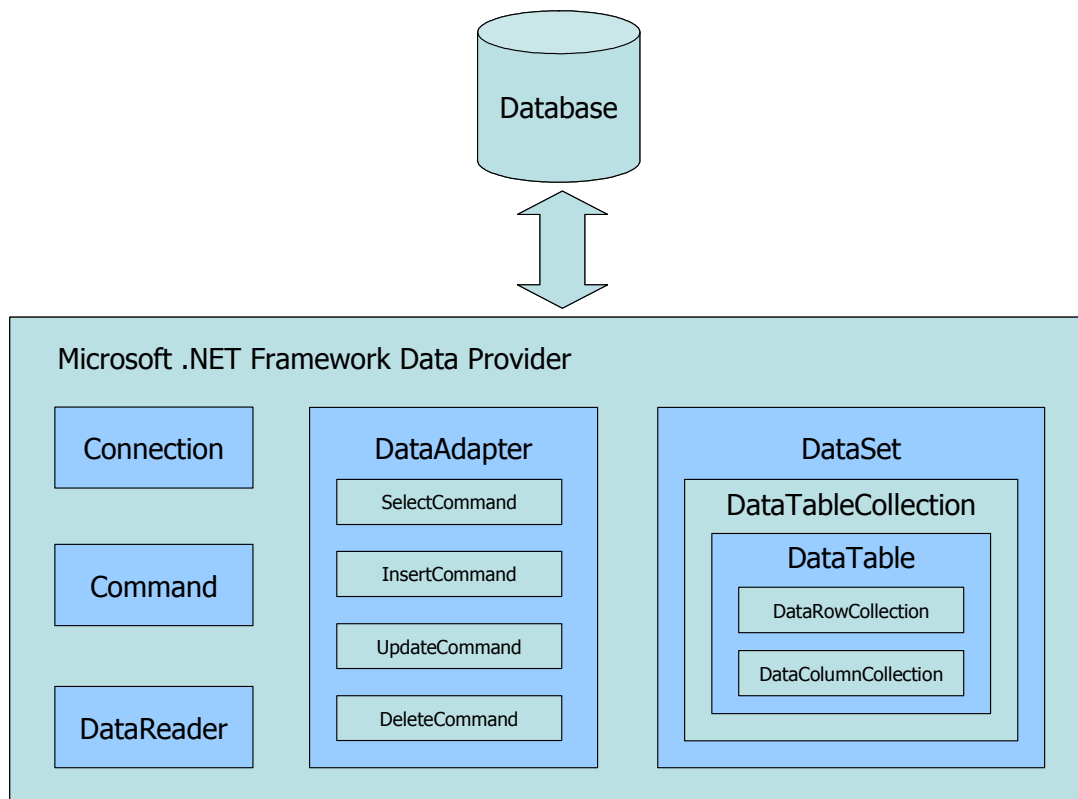
The Microsoft SQL Server .NET Data Provider is specially built for connecting to the Microsoft SQL Server systems. Microsoft SQL Server is an advanced database management system used for providing multiple access and management of the data contained within the database. The Microsoft SQL Server .NET Data Provider caters for the requirements of Microsoft SQL Server data sources exclusively.

For the purposes of this text, the ODBC and OLE DB data providers are suitable. The Microsoft SQL Server .NET Data Provider is an advanced topic which won't be discussed in this text. It's just worth mentioning because you will most definitely come across this type of data provider in the

industry. More specifically, the OLE DB .NET Data Provider will be used in the code samples, examples and exercises. The reason for using the OLE DB .NET Data Provider is because it's an updated Microsoft technology in widespread usage providing all the functionality for database programming.

## Data manipulation

A data provider performs the duties of transferring data between the software application and the data source. To do something with the data, we need a component to retrieve the data from a data provider and do something with that data in order for the data to mean something, in other words for the data to become information. The Microsoft .NET Framework provides powerful and easy to use libraries called the ADO.NET data access and manipulation libraries. During the next section we will look at a diagram explaining how the Microsoft ADO.NET component works:



The above diagram demonstrates how the Microsoft ADO.NET classes fit together. Supplementary to the diagram is a list of the various classes to discuss the purpose within the ADO.NET technology. Please study the table of descriptions to gain a better understanding as to where does each class fit into.

Class	Description
<b>Database</b>	The data source containing the data. This may be a database server system such as Microsoft SQL Server, a standalone database file such as a Microsoft Access file, or even a text file.
<b>Connection</b>	The Connection class provides a means of connecting to the data source using a connection string. The connection string is a collection of properties identifying the data source and the method to connect to the data source.
<b>Command</b>	The Command class provides functionality to retrieve and alter data within the database through the Connection class. SELECT, INSERT, UPDATE, and DELETE SQL statements are executed using the Command class.
<b>DataReader</b>	The DataReader class is used to retrieve specific data from the data source using the Connection class. The DataReader class uses the Command class and provides a useable result set for data retrieval.
<b>DataAdapter</b>	The DataAdapter class provides more advanced reading and writing functionality provided by the Command and DataReader classes.
<b>DataSet</b>	The DataSet class provides the container for data to be stored in within the software application. The DataSet is a collection of other classes and it is important to understand that it is a container class for other classes.
<b>DataTableCollection</b>	The DataTableCollection class is a collection of DataTables stored within the DataSet. The DataTableCollection class stores multiple result sets retrieved from a data source.
<b>DataTable</b>	The DataTable class is a container to store the result set of one table as perceived in the database.
<b>DataRowCollection</b>	The DataRowCollection class is a collection of DataRows each representing a single record retrieved from the data source.
<b>DataColumnCollection</b>	The DataColumnCollection class is a collection of DataColumns representing each column within the DataRow.

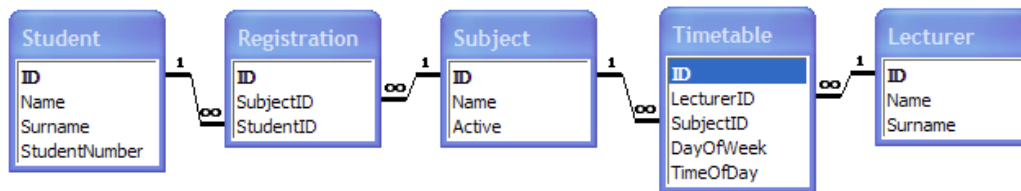
The table and diagram contains classes of ADO.NET components not used in this text. This text will focus on the Connection, Command, DataReader, and DataTable classes. The usage of these classes is discussed in detail in the subsequent examples of this text.

## Case Studies

During this section we will look at the Case Studies used within this text. Three Case Studies will be discussed and used in examples as well as exercises for this chapter. Please study and reference back to these Case Studies on a regular basis for the examples and exercises to follow in this chapter. The Case Studies refers to a fictional company called Software Solutions, which is the company who “employed” you as a software developer. Additionally, examples will give extra information regarding the Case Study used for the example within the specific context and immediate requirement.

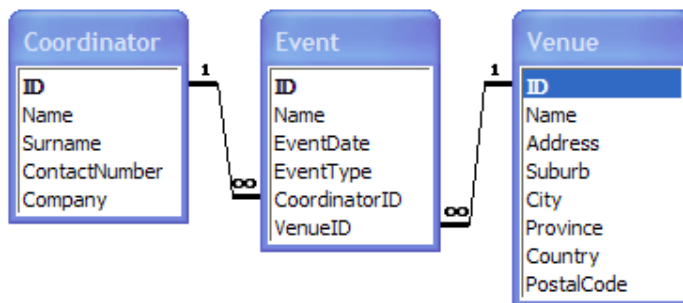
## Case Study: College

Your company called Software Solutions has been employed for the development of a software system to handle basic registration and timetable functions within Crystal Creek College. The college currently uses a manual paper system and feels that the need for a computerized system is critical for the efficient handling of their student affairs. Their requirement is that Software Solutions will provide a reliable and easy to use system for the recording of data such as lecturers, students, student registrations and timetables. The data should be stored within a database and the software system provides the functionality to draw and manipulate the data stored within the database for the day-to-day operations of the college. Additionally, the system should also provide the headroom to generate managerial reports such as student counts. Below is the table structure for the CollegeDB Microsoft Access database file. Please study the diagram, in specific the relationships between the tables indicated by the links between each table. For instance, the "Student" and "Registration" tables are linked together by the "StudentID" field. The "StudentID" field in the "Registration" table links to the "ID" field in the "Student" table. We call the "ID" field in the "Student" table, a primary key and the "StudentID" field in the "Registration" table a foreign key. Please study the rest of the table structure and identify the remaining relationships.



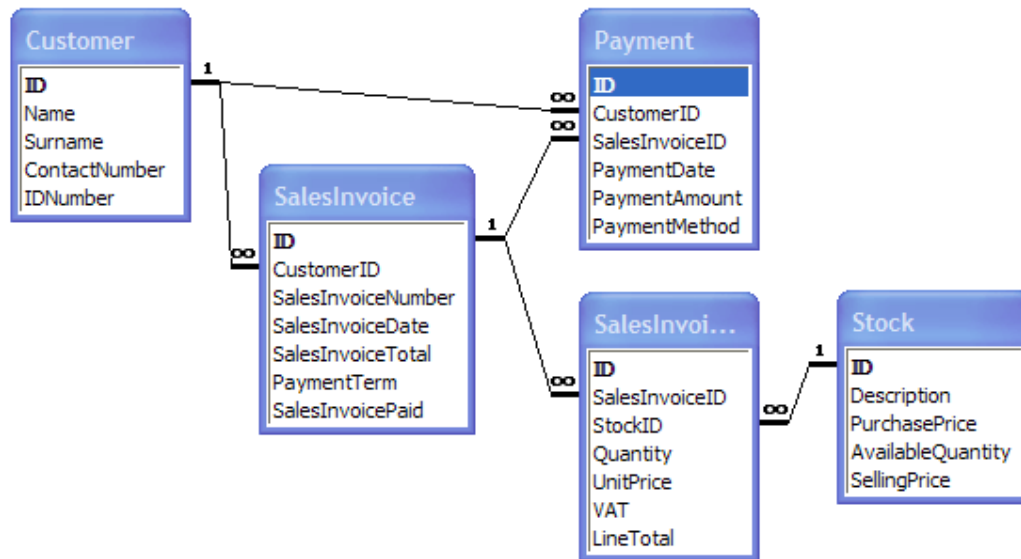
## Case Study: Fascination Productions & Events

Fascination Production & Events contacted Software Solutions to build a software application to handle the scheduling of events. Fascination Production & Events is a dynamic company specializing in the organization of important and exciting events within the entertainment industry. Amongst these events are road shows, live performances of bands, comedian events, sporting events, and theatre performances. The company employs coordinators who handle the organization of a specific event which is held at a specific venue. Software Solutions' duty is to provide a software system that will keep track of the coordinators, venues, and events. Please study the relationships between the tables for the subsequent exercises at the end of the chapter.



## Case Study: KC's Trading

A local distribution store called KC's Trading is using an outdated and unsupported point of sales system. A lot of problems have arisen since the system is no longer supported or actively developed. KC's Trading has requested Software Solutions to develop a new and exciting point of sales system catering for the specific needs of KC's Trading. The point of sales system should keep track of what stock items are sold by KC's Trading as well as the stock levels. Secondly, the system should keep track of customers, sales and payments. KC's Trading also requires that the system provides the facility for the development of managerial tools such as management reports with which the company directors may make important decisions regarding the operations of the business. The first and foremost requirement of the system is to replace the legacy point of sales system with as little interruptions as possible. Please study the relationships between the tables for the subsequent exercises at the end of the chapter.



### Data Transactions

Data transactions are the actions taken on the database to fetch, manipulate and send data between the database and the application. The application uses software libraries to perform these data transactions and in our case, we're using the ADO.NET technology to perform these data transactions. There are four data transactions of concern and these are:

1. SELECT
2. INSERT
3. UPDATE
4. DELETE

SELECT statements are used to fetch data from the database. A SELECT statement consists of a SQL query in the following format:

```
SELECT [ColumnName], [ColumnName]
FROM [TableName]
WHERE [ColumnName] = [Condition]
ORDER BY [ColumnName]
```

This SQL query does not contain all the elements for a SELECT statement, it's merely an example and most of the common elements are present in this query. Please refer to the SQL appendix for further information regarding SQL queries. What is important though, is to understand that the SQL standard is implemented into the ADO.NET technology. A SQL query is used in the Command class as the CommandText property. Along with the Parameters collection of the Command class, we are also able to create and execute parameterized queries. Parameterized queries are discussed in a subsequent section including an example application. SELECT statements may be used to populate controls such as ListBoxes and DataGridViews in the application.

INSERT statements are used for data insertion into the database. Below is an example of the INSERT statement's basic format and structure:

```
INSERT INTO [TableName] ([ColumnName], [ColumnName])  
VALUES ([ColumnValue], [ColumnValue])
```

UPDATE statements are used for the updating of data already present in the database. Below is an example of the basic format and structure of an UPDATE statement:

```
UPDATE [TableName] SET  
    [ColumnName] = [ColumnValue],  
    [ColumnName] = [ColumnValue]  
WHERE [ColumnName] = [Condition]
```

DELETE statements are used for the deletion of existing data in the database. The basic format and structure of the DELETE statement is as follows:

```
DELETE FROM [TableName]  
WHERE [ColumnName] = [Condition]
```

All of these transactions are executed using the Command class of the ADO.NET technology and is used as a string in the CommandText property. Please refer to the examples to follow for the use of the CommandText property.

## Flow Control

Flow control is the programmatic provision of a failover mechanism for when a critical error has occurred. This is useful for when the application has failed at a specific point but you don't want the application to halt entirely. For our purposes, we will look at the Try-Catch block as a flow control mechanism. The Try-Catch block consists of two parts, namely the Try clause and the Catch clause. Below is an example of what the Try-Catch block looks like:

```
Try  
  
Catch ex As Exception  
  
End Try
```

Both the Try and Catch clauses may contain code, but it is not definite that all of the code will execute. A Try-Catch block executes the code contained in the Try clause and if the code in the Try clause fails, the Catch clause will take over to complete whatever code needs to execute for the successful continuation of the application. Typically, the Try clause will contain the code to be executed for the specific function and the Catch clause will contain the code to raise the error to the user, or log it in a file, or finalizing code that needs to run after the failure of the code in the Try clause to do cleanup.

Database applications rely on the use of Try-Catch blocks to allow the application to continue with execution even if the application failed to read or write a record(s). Hence, the entire database transaction is usually contained within a Try-Catch block in order for the application to remain stable after the code has failed to execute properly. Usually we need to do something with the error and maybe even perform cleanup on the database. For the purposes of this text, we will only raise the error to the interface so that we know of the code failure.

## Examples

### How to use the examples

Included with this text, is a CD-ROM with each example's source code. Each example has two sets of source code, namely "Incomplete" and "Complete". The "Incomplete" source code is the source code to be used for working through each example, and the "Complete" source code is the completed version which you may use as a reference point. For instance, you will find two folders namely "Complete" and "Incomplete" in the "Examples" folder on the CD-ROM. Within each of the two folders, you'll find the appropriate source code for the example. To work through the examples, use the "Incomplete" source code and to check the end result of an example, use the "Complete" folder's example source code. You need to copy the source code to a writeable media though. Such as a flash drive or your computer's hard drive should be sufficient.

### Example 1: Connecting to a database using ADO.NET

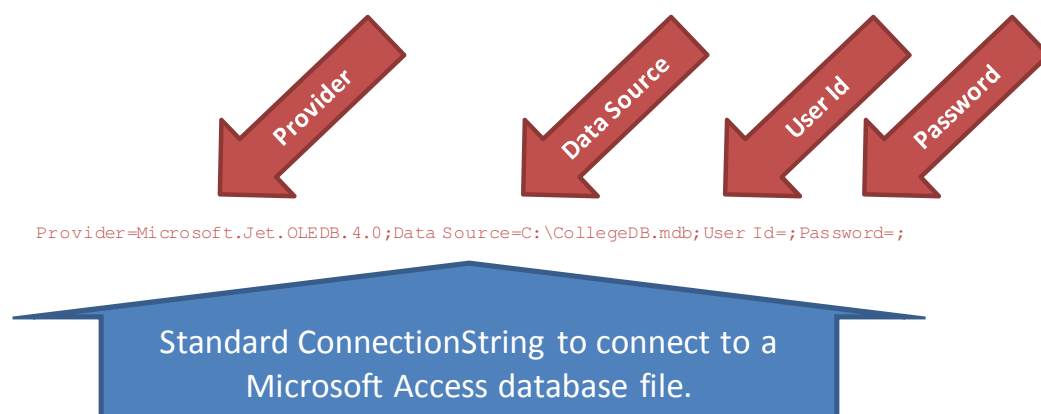
This example will guide you in connecting to a Microsoft Access database using Microsoft Visual Studio 2008 Express and the Microsoft ADO.NET classes. Before we start with the code, it is important to identify two of the components required for this exercise which are:

#### The data source and its location on the hard disk drive.

To determine the location, use Windows Explorer to browse to the location of the source code you've copied from the supplied CD-ROM. Hereby, you will be able to determine the complete path of the location.

#### A connection string with the relevant properties:

The connection string is collection of properties indicating where the data source is located, how it should be accessed, and with what parameters it should be accessed. Let's have a closer look at the connection string. Below is an example connection string:



As you can see, the connection string is a semi-colon delimited list of properties indicating where the data source is located, with what data provider it should be access, and with what credentials (username and password) it should be accessed.


For our purposes, we will always use the connection string in this format and the only property that we need to change whilst programming is the "Data Source" property. The "Data Source" property points your program to the particular database file to be used. For explanatory purposes, here is a list of the properties' purposes:



- 1) Provider: indicates what type of access method should be used, e.g. OLE DB, ODBC, or SqlClient.
- 2) Data Source: indicates the database file to be used in a full path- and filename format.
- 3) User Id: this property is required for the standard syntax of the connection string but we will not be using secure access to our data sources, but if you had controlled access to a data source, you would use this property to specify the username.
- 4) Password: this property is required for the standard syntax of the connection string but we will not be using secure access to our data sources, but if you had controlled access to a data source, you would use this property to specify the password.

## NB SETTING THE DATASOURCE PATH NB

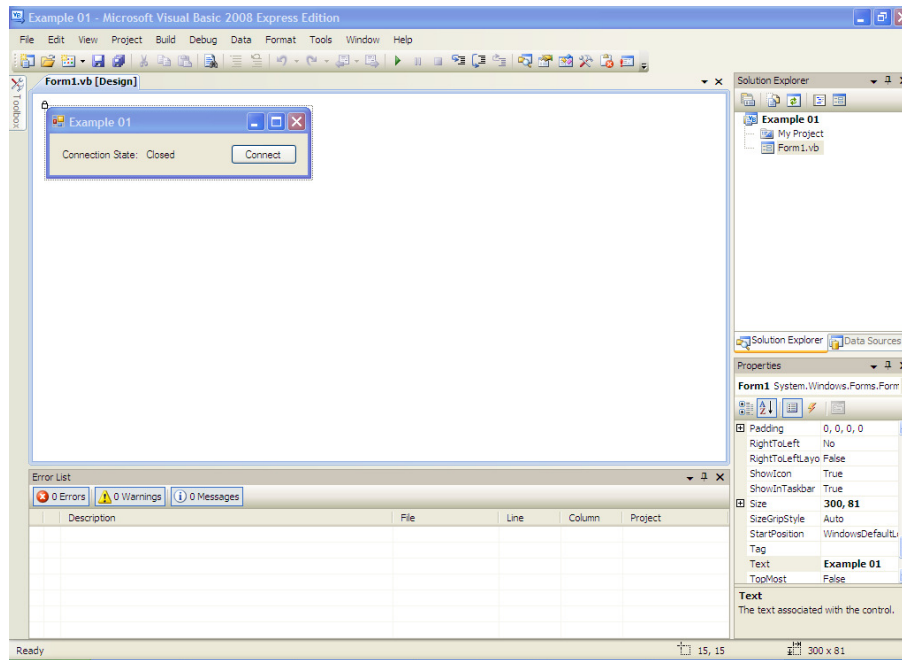
```
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\CollegeDB.mdb User Id=; Password=;
```



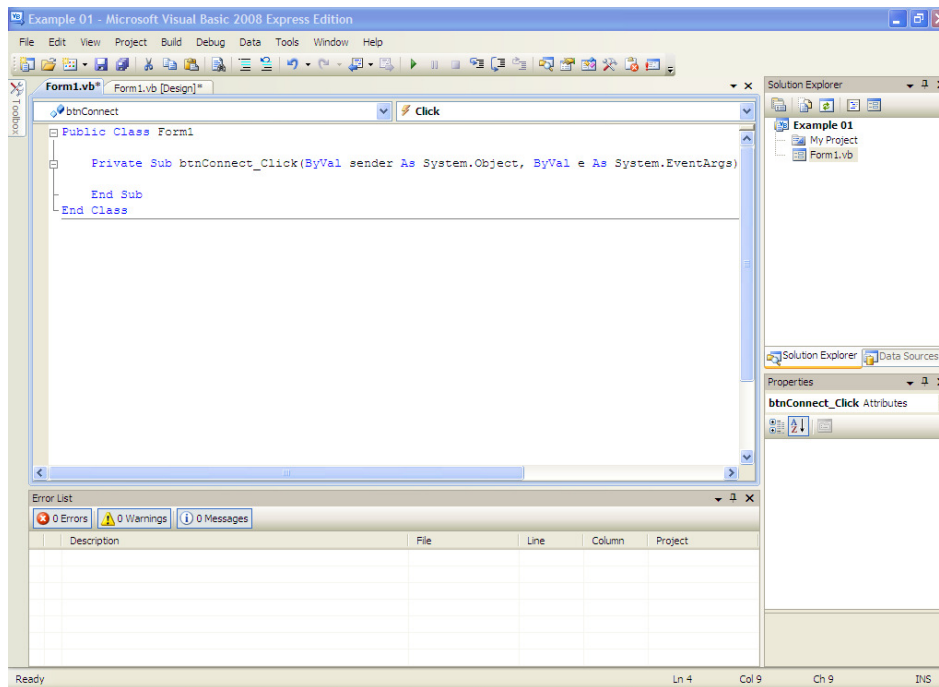
NB! Ensure that the DataSource PATH points to the folder that contain the Access Database.  
TAKE NOTE: This path may be different on your computer.

So let's dig in on how to connect to a database file. Make sure that you have copied the "Incomplete" source code from the CD-ROM unto your hard disk drive or flash drive.

- Open the solution file and double click on Form1.vb in the "Object Explorer". Please take a look at the following screenshot. The form is already visually prepared for this example; we will just fill in the source code to complete the example:



- On the form, we have two labels. One label is a descriptive label e.g. "Connection State" and the second one will be altered programmatically to indicate the state of the connection we are going to make to the database file. Please note that the default value for the second label is "Closed". Additionally, we have a button on the form called "Connect" and this button is used to initiate the database connection, keep it alive for 5 seconds and thereafter disconnect from the data source.
- Double click the "Connect" button to create an event handler for it. You should now have the following screen in front of you.



- The event handler will contain the logic to connect to the database file.

- Create a Try-Catch block for flow control. Flow control is the programmatic measurement taken to prevent an application from falling over because of exceptions or coding errors. Database related transactions should always be executed within a Try-Catch block. Please look at the structure of the Try-Catch block below. This is the standard format of the Try-Catch block and may be used for purposes other than database connectivity as well. Flow control will be discussed in a later section, but for now it's important only to know that the database related code needs to be contained in a Try-Catch code block.

```
Try
Catch ex As Exception
End Try
```

- The Try clause will execute the code the programmer wants executed. The Catch clause will take over from the Try clause whenever there was an error or exception in the code written in the Try clause. A Catch block will only be executed when there was an error or exception in the Try clause. For database connectivity, we will be using Try clause for executing the code to connect, fetch and alter the data within a data source and the Catch block to raise the exception to the user using the MessageBox.Show() method.
- Declare a connection object using the following statement in the Try clause:

```
' Declare the connection object.
Dim connection As New OleDb.OleDbConnection
```

- Set the connection string property of the connection object accordingly. For example, the source code for this tutorial was stored on my C drive, thus my connection string will look like this:

```
' Initialize the connection object's connection string.
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Complete\Example 01\CollegeDB.mdb;User Id=;Password=;"
```



**REMEMBER** to determine your database file's path and name and use that particular name in the Data Source property of the connection string.

- Open the database connection using the Open() method:

```
' Open the connection.
connection.Open()
```

Now we have an active and open database connection that we may use to interact with the database.

- For the purpose of this example, we just want to create a connection, open it, and close it afterwards. For that, we have a label on the form called lblConnectionState indicating the current state of the connection object. The default value specified in the Text property of the label is "Closed". We will programmatically alter this value to "Open" and back to "Closed" whilst executing the database connection code. After opening the database connection, alter the label's text property to "Open" and disable the "Connect" button in order to prevent the user from clicking the button again while the connection is active:

```
' Update the text of the label to "Open" and disable the "Connect" button whilst in
the connected state.
lblConnectionState.Text = "Open"
btnConnect.Enabled = False
```

Refresh the form in order to make the label's change visible.

```
' Update the form's content.  
Me.Refresh()
```

In order for us to see the connection object opening and closing the connection to the database file we need to stop the application execution for a small period. Pause the application execution using the Sleep() method in the Thread class:

```
' Stop the application's execution for 5 seconds.  
System.Threading.Thread.Sleep(5000)
```

The Sleep() method takes the delay time in milliseconds as an argument. Thus 5000 milliseconds will cause the application to sleep for 5 seconds, making it possible for us to see the change in the connection's state.

- Lastly, we need to close the connection to the database file. Use the Close() method of the connection object to close the database connection.

```
' Close the connection.  
connection.Close()
```

- Change the label's text property to "Closed", enable the "Connect" button again and update the form accordingly:

```
' Update the text of the label to "Closed" and enable the "Connect" button.  
lblConnectionState.Text = "Closed"  
btnConnect.Enabled = True
```

```
' Update the form's content.  
Me.Refresh()
```

- Add a prompt to the Catch clause in order to raise the exception to the user. Raising the error to the user notifies him / her that there was a problem with the execution of the application. This manner of exception handling is not very elegant but sufficient for us to determine the nature of the failure of the code.

```
' Prompt the user with the exception.  
MessageBox.Show(ex.ToString())
```

- Let's summarize the code up until now, so that we have a better view of it. In the event handler of the "Connect" button you should have the following code:

```
Private Sub btnConnect_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles btnConnect.Click  
    Try  
        ' Declare the connection object.  
        Dim connection As New OleDb.OleDbConnection  
  
        ' Initialize the connection object's connection string.  
        connection.ConnectionString =  
            "Provider=Microsoft.Jet.OLEDB.4.0;Data  
            Source=C:\Examples\Incomplete\Example 01\CollegeDB.mdb;User  
            Id=;Password=;"  
  
        ' Open the connection.  
        connection.Open()  
  
        ' Update the text of the label to "Open" and disable the  
        "Connect" button whilst in the connected state.  
        lblConnectionState.Text = "Open"  
        btnConnect.Enabled = False  
  
        ' Update the form's content.  
        Me.Refresh()  
  
        ' Stop the application's execution for 5 seconds.
```

```

        System.Threading.Thread.Sleep(5000)

        ' Close the connection.
        connection.Close()

        ' Update the text of the label to "Closed" and enable the
        "Connect" button.
        lblConnectionState.Text = "Closed"
        btnConnect.Enabled = True

        ' Update the form's content.
        Me.Refresh()
    Catch ex As Exception
        ' Prompt the user with the exception.
        MessageBox.Show(ex.ToString())
    End Try
End Sub

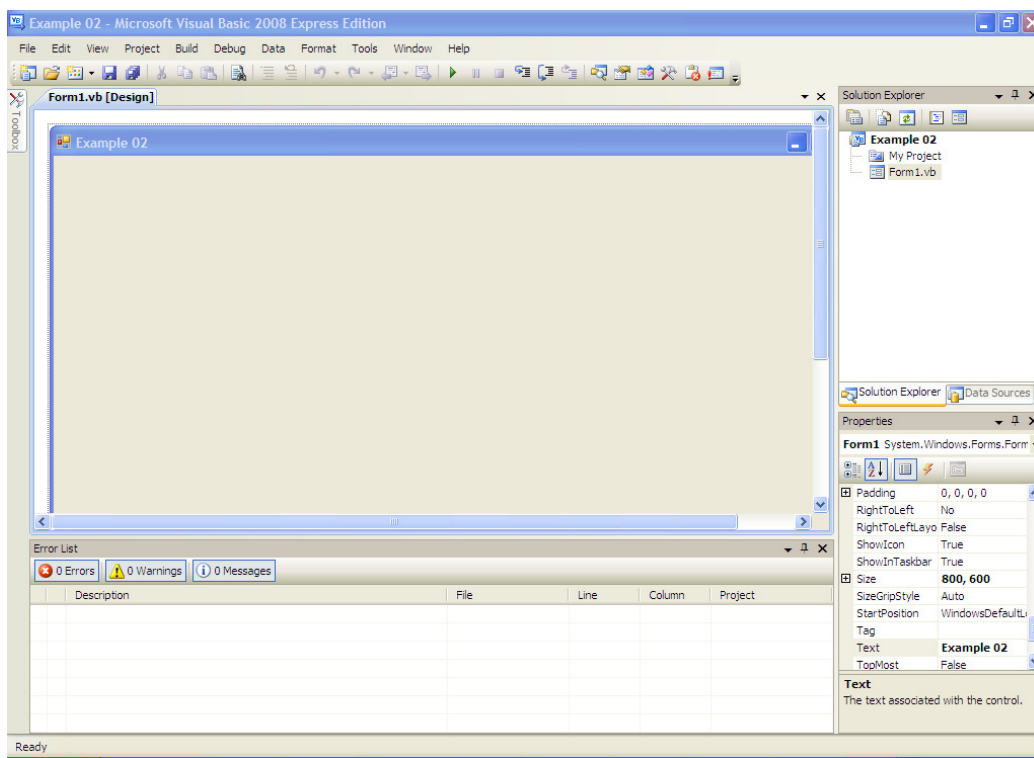
```

- Run the application by pressing F5 or clicking on Debug > Start Debugging.
- Click on the "Connect" button. The label should change from "Closed" to "Open" and remain like that for 5 seconds. Afterwards, it should change back to "Closed". Additionally, the "Connect" button should also become disabled in order to prevent another click event while the current click event is in progress.

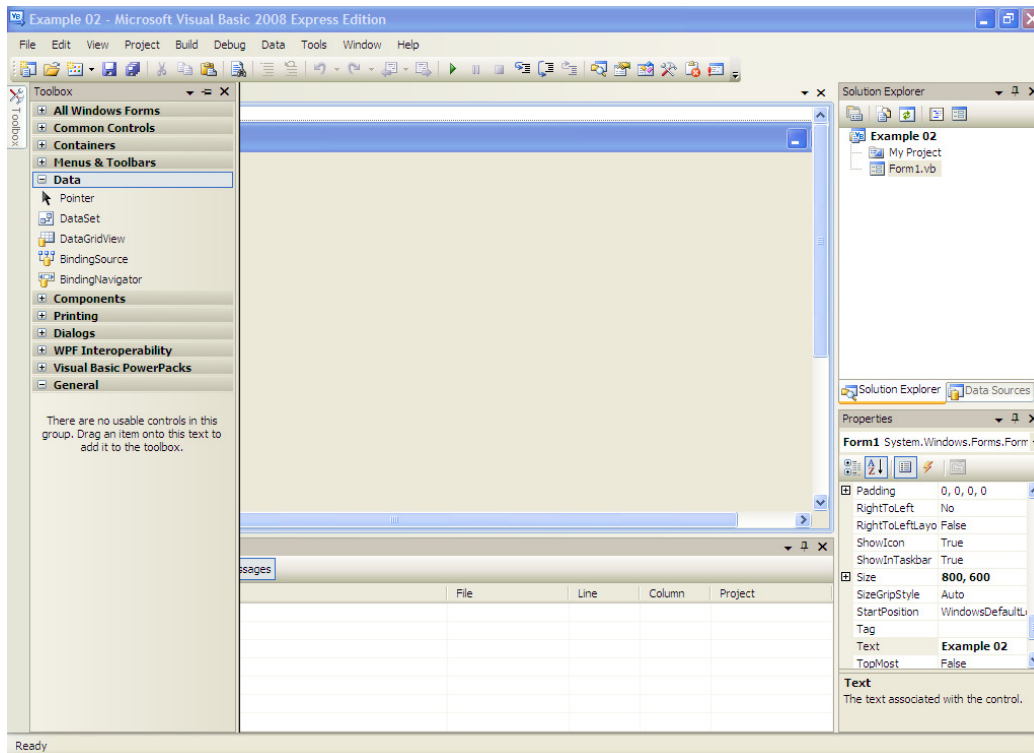
## Example 2: Retrieving and displaying data in a DataGridView control

During this example, we will learn how to retrieve data from a database file and display the result set in a DataGridView control. The DataGridView control is a Microsoft .NET control and makes the display of data in a tabular format very easy and hassle free. DataGridViews can also perform many other duties but for the purpose of this text, we will use it for display purposes only. This tutorial will use the OleDbConnection, OleDbCommand and DataTable classes to retrieve the data from a database file using a SQL query and display the result set in a DataGridView.

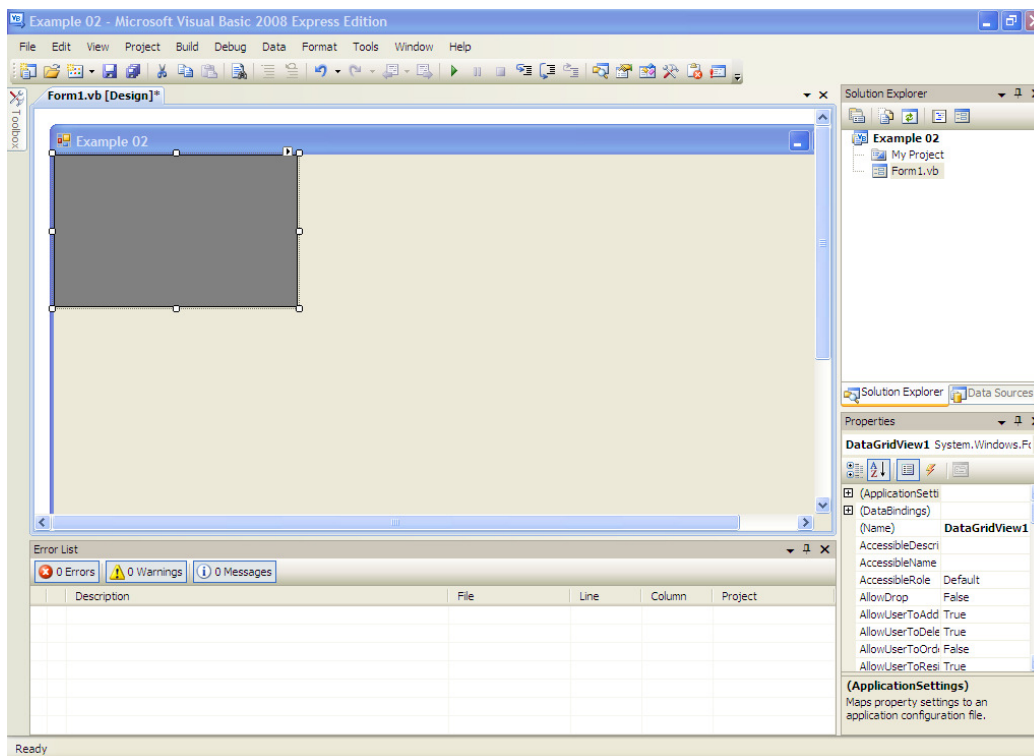
- Copy the source code for Example 02 to the hard disk drive or flash drive and open the solution with Microsoft Visual Basic 2008 Express Edition.
- Double click the Form1.vb item in the Object Explorer. You should now have the following screen in front of you:



- The form we are going to use does not have anything on it yet. We will be building the application from the start. Firstly, we need to add a DataGridView control to the form. Expand the Toolbox pane by hovering the mouse pointer over the Toolbox pane on the left.



- Expand the “Data” category on the Toolbox and double click on the DataGridView item in the list to add the control to the form. Your screen should now look like the one below:

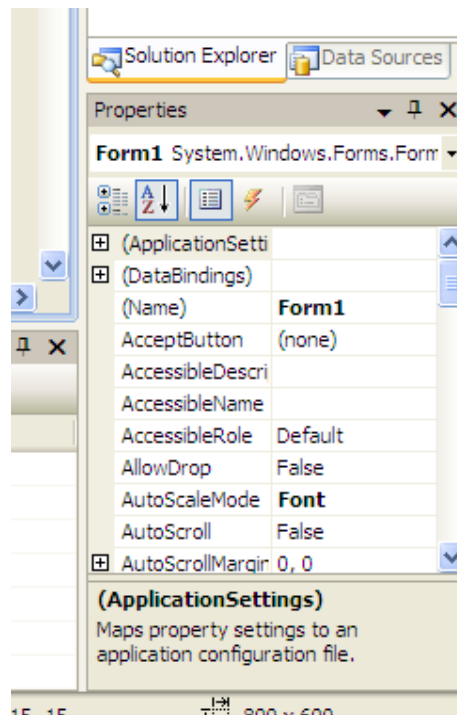


- You are welcome to style the DataGridView to your taste by resizing or docking it. Just make sure to properly name the DataGridView e.g. grvStudents is a proper name for the purpose of this tutorial as we are going to populate the DataGridView with records from the Student table in the Microsoft Access database.
- Additionally, change the following properties of the DataGridView control accordingly:

<b>AllowUserToAddRows</b>	<b>False</b>	<b>Prevents user from adding new items in the DataGridView.</b>
<b>AllowUserToDeleteRows</b>	<b>False</b>	<b>Prevent user from removing existing items in the DataGridView.</b>
<b>AutoSizeColumnsMode</b>	<b>Fill</b>	<b>Formats the columns to fit the width of the DataGridView. This property is purely aesthetic and may be void if need be.</b>
<b>MultiSelect</b>	<b>False</b>	<b>Prevents the user from selecting more than one row at a time.</b>
<b>ReadOnly</b>	<b>True</b>	<b>Prevents the user from editing existing items in the DataGridView.</b>

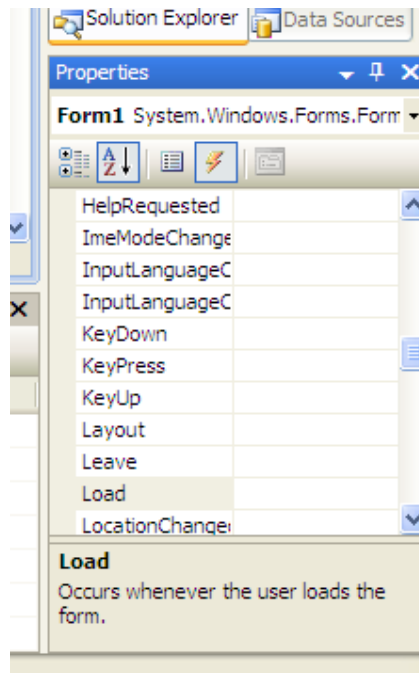
This list of properties with their values is to be used for all DataGridView controls for the examples to follow. Please make sure that you configure all of your DataGridViews according to this table.

- In the Properties pane on the right, click on the dropdown list and select Form1.





- Switch the Properties pane to the Events section by clicking on the little orange thunderbolt. Scroll down to the Load item in the list and double click on it. Microsoft Visual Basic 2008 Express Edition should now switch you to a source code file with an event handler for the Load event of the form. This is where we will add all of the logic for this example.



- Add a Try-Catch block to the event handler.

```
Try
Catch ex As Exception
End Try
```

- Within the Try clause, declare the connection, command and data table objects using the following lines of code.

```
' Declare the connection object.
Dim connection As New OleDb.OleDbConnection

' Declare the command object.
Dim command As New OleDb.OleDbCommand

' Declare the table object.
Dim table As New Data.DataTable
```

Two new classes are introduced in this example, namely DataTable and OleDbCommand. The OleDbCommand uses an object instance of the OleDbConnection class to connect to a database file. And with a SQL query the command object fills the DataTable object with the result set retrieved from the database.

- Initialize the connection object with the necessary connection string and open the connection to the database file.

```
' Initialize the ConnectionString property.
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Complete\Example 02\CollegeDB.mdb;User Id=;Password=;"

' Open the database connection.
connection.Open()
```

- Set the connection property of the command object.

```
' Set the Connection property of the command object.
command.Connection = connection
```

- Set the SQL query for the command object.

```
' Set the CommandText property of the command object.
command.CommandText = "SELECT * FROM Student"
```

The CommandText property is used for all of the types of SQL transactions including SELECT, INSERT, UPDATE, and DELETE statements.

- The OleDbCommand class provides two-way communication to the database file. To retrieve a result from the database, the ExecuteReader() method is used. To send a query such as an INSERT, UPDATE or DELETE statement to the database, the ExecuteNonQuery() is used. The ExecuteNonQuery() method is introduced at a later stage. The DataTable class contains a Load() method that takes a DataReader object. To populate the table object, we are going to execute two statements at once, because the one method's return value is the input value for the other method. To populate the table object, add the following code.

```
' Execute the command and load the data into the table.
table.Load(command.ExecuteReader())
```

- Close the connection to the database.

```
' Close the database connection.
connection.Close()
```

- After populating the table object, we need to tell the DataGridView to use the table object as its data source.

```
' Set the DataSource property of the grvStudents DataGridView control.
grvStudents.DataSource = table
```

- Add a prompt to the Catch clause in order to raise the exception to the user.

```
' Prompt the user with the exception.
MessageBox.Show(ex.ToString())
```

- Let's summarize the code block for the Load event handler.

```
Private Sub Form1_Load(...)
    Try
        ' Declare the connection object.
        Dim connection As New OleDb.OleDbConnection

        ' Declare the command object.
        Dim command As New OleDb.OleDbCommand

        ' Declare the table object.
        Dim table As New Data.DataTable

        ' Initialize the ConnectionString property.
        connection.ConnectionString =
            "Provider=Microsoft.Jet.OLEDB.4.0;Data
            Source=C:\Examples\Incomplete\Example 02\CollegeDB.mdb;User
            Id=;Password=;"

        ' Open the database connection.
        connection.Open()

        ' Set the Connection property of the command object.
        command.Connection = connection
```

```

' Set the CommandText property of the command object.
command.CommandText = "SELECT * FROM Student"

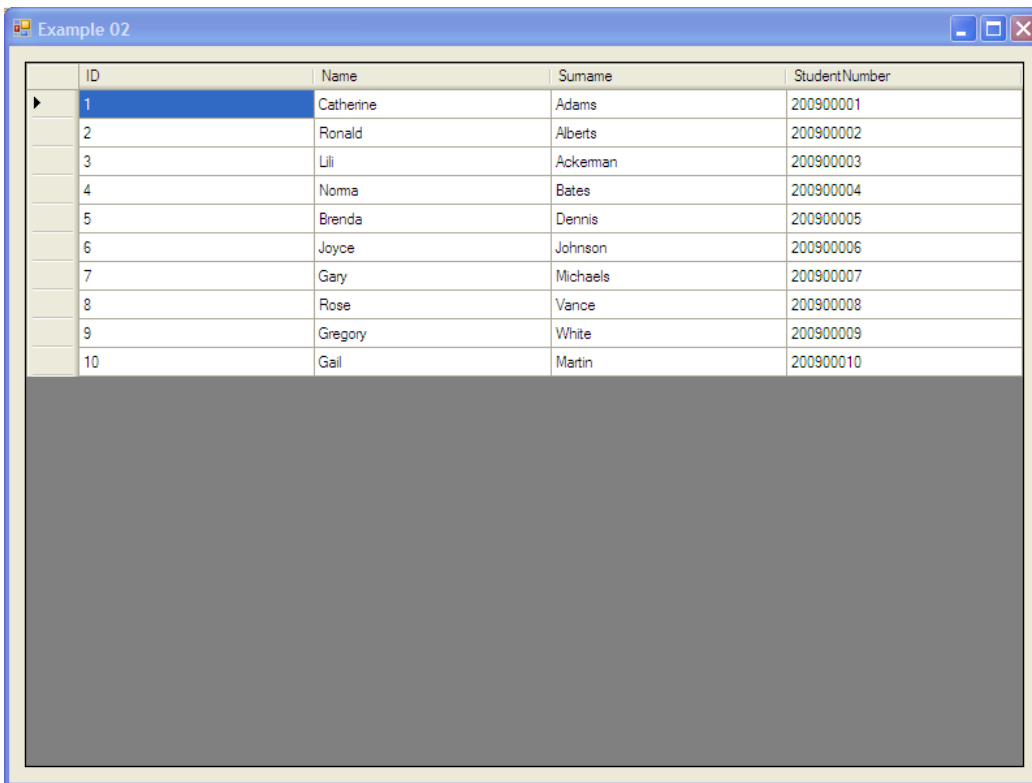
' Execute the command and load the data into the table.
table.Load(command.ExecuteReader())

' Close the database connection.
connection.Close()

' Set the DataSource property of the grvStudents DataGridView control.
grvStudents.DataSource = table
Catch ex As Exception
' Prompt the user with the exception.
MessageBox.Show(ex.ToString())
End Try
End Sub

```

- Run the application by pressing F5 or clicking on Debug > Start Debugging and check the results in the window. The result should look like this:



ID	Name	Surname	StudentNumber
1	Catherine	Adams	200900001
2	Ronald	Alberts	200900002
3	Lili	Ackeman	200900003
4	Noma	Bates	200900004
5	Brenda	Dennis	200900005
6	Joyce	Johnson	200900006
7	Gary	Michaels	200900007
8	Rose	Vance	200900008
9	Gregory	White	200900009
10	Gail	Martin	200900010

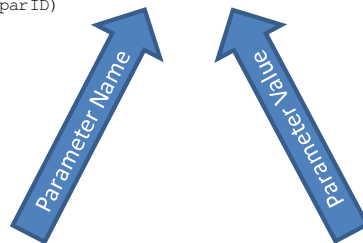
## Parameterized Queries

When doing queries to the database, be it database reading or writing we sometimes require a level of scalability in terms of data input when preparing the queries and sending it to the database. The data input could be the WHERE clause's values or the VALUES list in an INSERT statement. SQL Parameters provide us the functionality to specify the values programmatically and we call these parameterized queries.

ADO.NET implements parameterized queries by providing a Parameters property in the Command class. Let's have a closer look at the following piece of code illustrating how the parameters are implemented.

The parameter below is a Name-Value pair consisting of the placeholder @ID and pID variable in the code snippet below.

```
command.CommandText = "SELECT * FROM Student WHERE ID = @ID"  
Dim parID = New OleDb.OleDbParameter("@ID", pID)  
command.Parameters.Add(parID)
```



- Firstly, the query specified in the CommandText property of the Command object is a little different from the queries we've seen in the preceding examples. We now have a "@ID" specified in the query. This is called a Parameter.
- Secondly, we need to create a parameter instance along with the value we want to send to the database. The OleDb.OleDbParameter constructor takes two arguments, namely the name of the parameter and the value for the parameter. The name must be identical to the parameter name specified in the CommandText property, and the value may be an appropriate value as expected by the database.
- Lastly, we need to add the parameter to the list of parameters for the Command object. We do this by specifying the Parameters property of the Command class and then by calling the Add() method along with the parameter we've created in the previous step to add the parameter to the list.

Important points to remember with parameters:

1. Parameters should be specified using the "@" sign prefixed to the name. For instance, if we want to specify a "Name" parameter, it would look something like this: "@Name".
2. Parameters should be added to the Command object in the order as they are found in the CommandText property. For instance, if we have the following CommandText:

```
INSERT INTO Student (Name, Surname, StudentNumber)  
VALUES (@Name, @Surname, @StudentNumber)
```

The parameters should be added to the Command object in the following order:

```
Dim parName = New OleDb.OleDbParameter("@Name", txtName.Text)
```

```
Dim parSurname = New OleDb.OleDbParameter("@Surname", txtSurname.Text)
Dim parStudentNumber = New OleDb.OleDbParameter("@StudentNumber",
txtStudentNumber.Text)
```

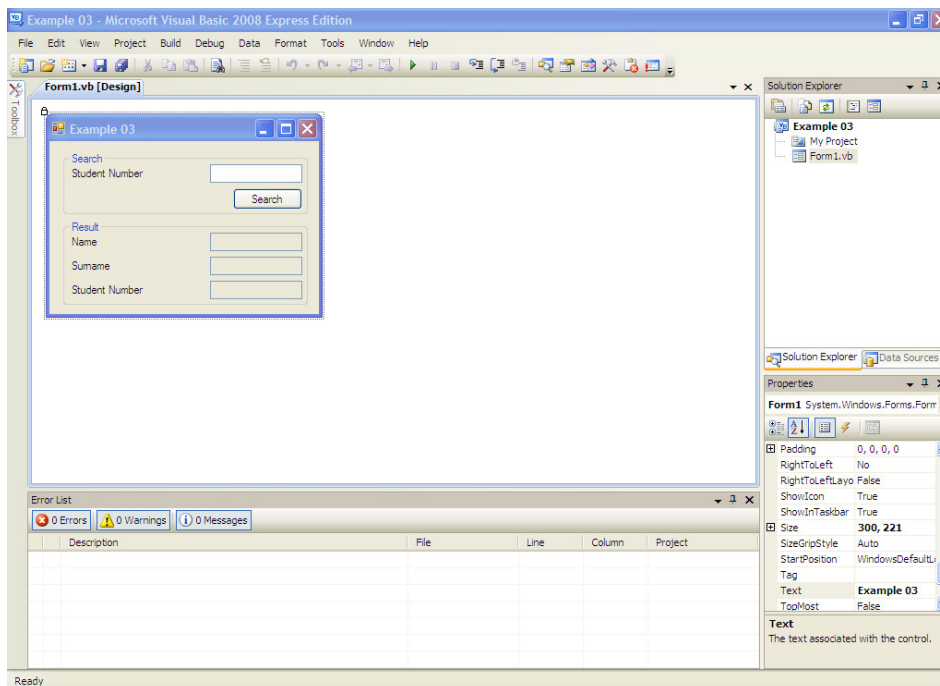
3. Parameters' values should correspond to the data types as expected by the database. For instance, if you want to query the student table with a student number, the value sent to the database should be in the correct format for the query to succeed, otherwise the execution of the query will fail.
4. Parameters should be complete. If five parameters are specified in the CommandText property, five corresponding parameters should be added to the list of parameters.

The remaining examples of this chapter rely heavily on the use of parameterized queries. Please study the following four examples on how to use parameterized queries for the different types of data transactions, e.g. SELECT, INSERT, UPDATE, and DELETE.

### Example 3: Retrieving a single record using a parameterized SQL query

Retrieving chunks of data from a data source without any conditions is mostly pointless, hence we should be able to specify certain conditions as to what data we want in the result set. In SQL terms this is called the "WHERE" clause or "HAVING" clause. The Command class contains a property called Parameters. This Parameters property allows the programmer to specify what values should be passed through as conditions. For this tutorial, we will look at how to retrieve a single record using a parameter on the "StudentNumber" field in the "Student" table.

- Copy the source code for Example 03 to your hard disk drive or flash drive and open the solution with Microsoft Visual Basic 2008 Express Edition.
- Double click on the Form1.vb item in the Object Explorer. Note that the visual part of the form is already built and that we only need to add the code for this application to be functional.



- Double click on the Search button to create an event handler for it.

- Add a Try-Catch code block.

```
Try
Catch ex As Exception
End Try
```

- Within the Try clause, declare connection, command and table objects:

```
' Declare the OleDbConnection, OleDbCommand and DataTable objects.
Dim connection As New OleDb.OleDbConnection
Dim command As New OleDb.OleDbCommand
Dim table As New Data.DataTable
```

- Initialize the ConnectionString property of the connection and open the connection.

```
' Set the ConnectionString property of the connection and open the database
connection.
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Incomplete\Example 03\CollegeDB.mdb;User Id=;Password=;"
connection.Open()
```

- Set the Connection and CommandText properties of the command.

```
' Set the Connection and CommandText properties of the command object.
command.Connection = connection
command.CommandText = "SELECT * FROM Student WHERE StudentNumber = @StudentNumber"
```

Note that the SQL query looks somewhat different from before. The SQL query now includes a WHERE clause specifying which column to filter results by along with a placeholder called a parameter. The parameter may be identified by looking for the "@" prefix.

- Next, we need to add the parameter value from the form to the command object so that we may receive the appropriate results. The OleDbParameter requires a name and value pair. The name is used to match the parameter name in the SQL query and the value is used for the filtering of the results. Declare a parameter using the following line of code:

```
' Declare and initialize the "StudentNumber" parameter.
Dim parStudentNumber = New OleDb.OleDbParameter("@StudentNumber",
txtStudentNumberSearch.Text)
```

- Note that on the declaration of the parameter, the value entered by the user is retrieved from the textbox on the form. Use the following line of code to add the parameter to the command object:

```
' Add the parameter to the command's parameter list.
command.Parameters.Add(parStudentNumber)
```

When the command executes the query, it will return only the data matching the criteria specified in the parameter.

- Retrieve the result set from the DataReader and store it into the table object.

```
' Execute the command and load the result set into the table object.
table.Load(command.ExecuteReader())
```

- Close the database connection.

```
' Close the database connection.
connection.Close()
```

- The remaining part of this tutorial relies on logic to check and populate the controls on the form according to the result set stored in the table object. First we need to check whether

there were any results at all and if there was no results returned, prompt the user to inform him or her.

```
If table.Rows.Count = 0 Then ' Check whether the row count of the table is zero, and
    prompt the user that no student was found.
    MessageBox.Show("No student found, please try again")
ElseIf table.Rows.Count = 1 Then ' Else load the student's details into the textbox
    controls on the form.
    txtName.Text = table.Rows(0)("Name")
    txtSurname.Text = table.Rows(0)("Surname")
    txtStudentNumber.Text = table.Rows(0)("StudentNumber")
End If
```

- Secondly, we need to handle the result set if there was indeed a record returned. We are expecting a single record from the database, thus it is safe to assume for the purpose of this example that if there is a result set, it will have only one record. A table consists of a list of rows, and a row consists of a list of columns. To access a specific row in a table, we need to specify the row and we do this in the following manner.

```
table.Rows(0)
```

The above statement will retrieve the first row in the table. To access the columns, we can either use the column name or the column index. If you know the column name, it is easier and better to use than the column indexes, because the column index method is somewhat ambiguous. To access a specific column in a row, use the following method to specify it.

```
table.Rows(0)("Name")
```

The statement above will return the value for the Surname column for the first record in the result set stored in the table object. To populate the controls on the form, we need to fetch the appropriate values from the table. The Else part of the If statement will look like this:

```
ElseIf table.Rows.Count = 1 Then ' Else load the student's details into the textbox
    controls on the form.
    txtName.Text = table.Rows(0)("Name")
    txtSurname.Text = table.Rows(0)("Surname")
    txtStudentNumber.Text = table.Rows(0)("StudentNumber")
End If
```

The complete If-Else statement should look like this:

```
If table.Rows.Count = 0 Then ' Check whether the row count of the table is zero, and
    prompt the user that no student was found.
    MessageBox.Show("No student found, please try again")
ElseIf table.Rows.Count = 1 Then ' Else load the student's details into the textbox
    controls on the form.
    txtName.Text = table.Rows(0)("Name")
    txtSurname.Text = table.Rows(0)("Surname")
    txtStudentNumber.Text = table.Rows(0)("StudentNumber")
End If
```

- Let's summarize the purpose of this If-Else statement. If the table contains no rows at all, notify the user. Else if the table contains a row, retrieve the specific values stored in the table and set the text properties of the textboxes on the form.
- Add a prompt to notify the user if there was an exception.

```
' Prompt the user with the exception.
MessageBox.Show(ex.ToString())
```

- The complete source code for this event handler should look like the code block below.

```
Private Sub btnSearch_Click(...)
    Try
        ' Declare the OleDbConnection, OleDbCommand and DataTable objects.
        Dim connection As New OleDb.OleDbConnection
        Dim command As New OleDb.OleDbCommand
```

```

Dim table As New Data.DataTable

' Set the ConnectionString property of the connection and open the
database connection.
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Complete\Example 03\CollegeDB.mdb;User
Id=;Password=;"
connection.Open()

' Set the Connection and CommandText properties of the command object.
command.Connection = connection
command.CommandText = "SELECT * FROM Student WHERE StudentNumber =
@StudentNumber"

' Declare and initialize the "StudentNumber" parameter.
Dim parStudentNumber = New OleDb.OleDbParameter("@StudentNumber",
txtStudentNumberSearch.Text)

' Add the parameter to the command's parameter list.
command.Parameters.Add(parStudentNumber)

' Execute the command and load the result set into the table object.
table.Load(command.ExecuteReader())

' Close the database connection.
connection.Close()

If table.Rows.Count = 0 Then ' Check whether the row count of the table
is zero, and prompt the user that no student was found.
    MessageBox.Show("No student found, please try again")
ElseIf table.Rows.Count = 1 Then ' Else load the student's details into
the textbox controls on the form.
    txtName.Text = table.Rows(0)("Name")
    txtSurname.Text = table.Rows(0)("Surname")
    txtStudentNumber.Text = table.Rows(0)("StudentNumber")
End If
Catch ex As Exception
    ' Prompt the user with the exception.
    MessageBox.Show(ex.ToString())
End Try
End Sub

```

- Run the application by pressing F5 or clicking Debug > Start Debugging.

- Test the application by typing a Student Number in the Search section of the form and click on Search. If you typed a Student Number found in the database, the Result block should be populated with the record returned from the database.



The screenshot shows a window titled "Example 03". It has two main sections: "Search" and "Result". In the "Search" section, there is a label "Student Number" followed by a text box containing "200900001" and a "Search" button. In the "Result" section, there are three labels: "Name", "Surname", and "Student Number". Each label is followed by a text box containing the corresponding result: "Catherine", "Adams", and "200900001".

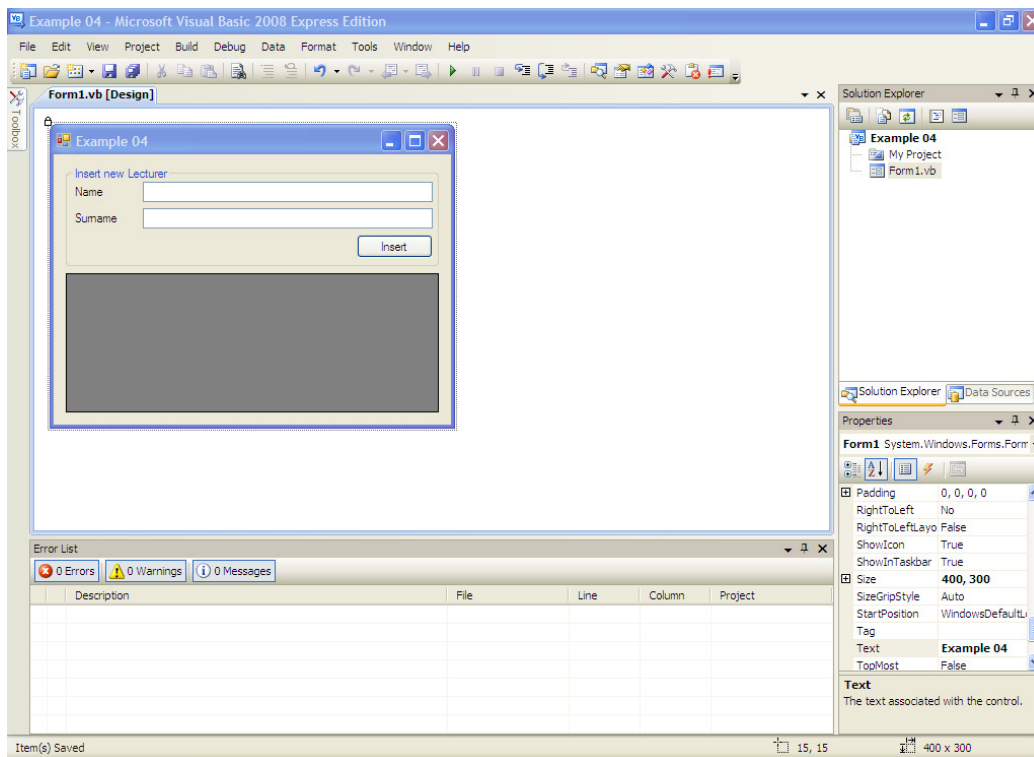
- If no student was found, the application should show a notification:

The screenshot shows the same "Example 03" window as before, but with a modal dialog box overlaid in the center. The dialog box has a title bar with a close button (X). The text inside the dialog says "No student found, please try again". Below the text is an "OK" button. The background window is partially obscured by the dialog.

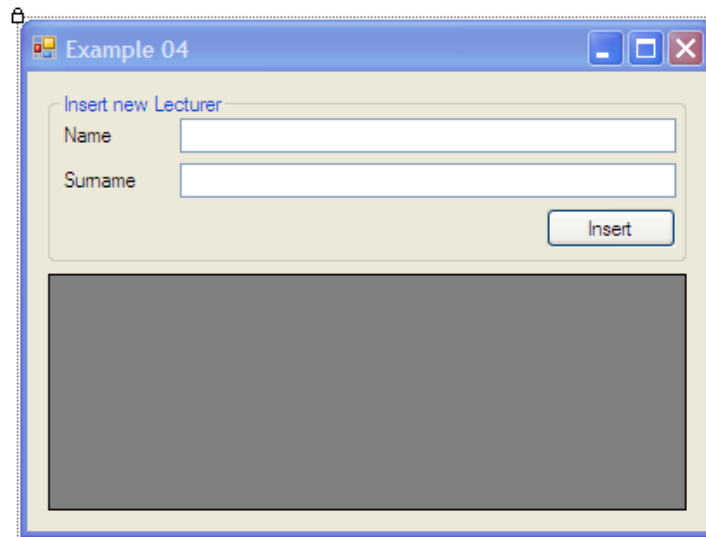
## Example 4: Inserting a new record

In this example, we will insert a new record in the database. To perform data insertion, we will use a parameterized query such as in example 3. To insert a new record, we need to use instances of an `OleDbConnection` and `OleDbCommand` classes. The connection is initialized in the same manner as for data retrieval. The command is also initialized in the same manner but a different method is used to perform the action and the `CommandText` for an `INSERT` statement is used instead. For data retrieval, we used the `ExecuteReader()` method to fetch the data from the database. But for sending data to the database, we need to use the `ExecuteNonQuery()` method in the `OleDbCommand` class. Please note that inserting, updating and deleting actions are all regarded as sending data to the database and for any of those actions, we need to use the `ExecuteNonQuery()` method. This example will guide you to insert a new Lecturer in the CollegeDB database.

- Copy the source code for Example 04 to your hard disk drive or flash drive and open the solution with Microsoft Visual Studio 2008 Express Edition.
- Open the Form1 item in the Object Explorer by double clicking on it. Note that the form is already built and we will only add the program logic to it. The screen you should have in front of you should look like the following.



- Let's have a closer look at the layout the screen.



This screen consists of two sections; the first section is the "Insert new Lecturer" section, and the second being the DataGridView which purpose is to display the lecturers in the database. The first section harvests the data input from the user and will perform an insert transaction into the database. Thereafter, we will get the new data from the database for the Lecturer table and display it in the DataGridView. The data retrieval for the DataGridView is already done and we only need to do the insert transaction in the "Insert" button's event handler.

- All of the functionality for inserting a new lecturer is contained in the "Insert" button's event handler. Double click on the "Insert" button to open its event handler.
- Add a Try-Catch block for the database transaction.

```
Try
Catch ex As Exception
End Try
```

- Declare the connection and command objects.

```
' Declare the connection and command objects.
Dim connection As New OleDb.OleDbConnection
Dim command As New OleDb.OleDbCommand
```

- Initialize theConnectionString of the connection and open the database connection.

```
' Initialize the ConnectionString property of the connection and open the database
connection.
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Incomplete\Example 04\CollegeDB.mdb;User Id=;Password=;"
connection.Open()
```

- Initialize the command by setting its Connection and CommandText properties.

```
' Set the Connection and CommandText properties of the command object.
command.Connection = connection
command.CommandText = "INSERT INTO Lecturer (Name, Surname) VALUES (@Name, @Surname)"
```

Note that we are now using an INSERT SQL statement, for the proper use of SQL INSERT statements; please refer to the SQL appendix. The INSERT statement is written as a parameterized query same as we've done on the parameterized SELECT query.

- Fetch the values for the parameters and add it to the command.

```
' Declare and initialize the parameters.
Dim parName = New OleDb.OleDbParameter("@Name", txtName.Text)
Dim parSurname = New OleDb.OleDbParameter("@Surname", txtSurname.Text)

' Add the parameters to the command object.
command.Parameters.Add(parName)
command.Parameters.Add(parSurname)
```

- Perform the insertion transaction to the database file.

```
' Execute the command.
command.ExecuteNonQuery()
```

- Close the connection to the database.

```
' Close the database connection.
connection.Close()
```

- In order to visually see the results of the insert transaction, we need to retrieve the lecturers from the database and populate the DataGridView with the result set. This example's source code includes a GetLecturers() method already written to retrieve the data. The only change you would need to make is on the ConnectionString of the connection object.

```
' Fetch the new result set for the Lecturer table.
GetLecturers()
```

- Add a prompt to the Catch clause in the case of an exception.

```
' Prompt the user with the exception.
MessageBox.Show(ex.ToString())
```

- Let's summarize the code for the "Insert" button's event handler:

```
Private Sub btnInsert_Click(...)
    Try
        ' Declare the connection and command objects.
        Dim connection As New OleDb.OleDbConnection
        Dim command As New OleDb.OleDbCommand

        ' Initialize the ConnectionString property of the connection and open
        the database connection.
        connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
        Source=C:\Examples\Incomplete\Example 04\CollegeDB.mdb;User
        Id=;Password=;"
        connection.Open()

        ' Set the Connection and CommandText properties of the command object.
        command.Connection = connection
        command.CommandText = "INSERT INTO Lecturer (Name, Surname) VALUES
        (@Name, @Surname)"

        ' Declare and initialize the parameters.
        Dim parName = New OleDb.OleDbParameter("@Name", txtName.Text)
        Dim parSurname = New OleDb.OleDbParameter("@Surname", txtSurname.Text)

        ' Add the parameters to the command object.
        command.Parameters.Add(parName)
        command.Parameters.Add(parSurname)

        ' Execute the command.
        command.ExecuteNonQuery()

        ' Close the database connection.
```

```

        connection.Close()

        ' Fetch the new result set for the Lecturer table.
        GetLecturers()
    Catch ex As Exception
        ' Prompt the user with the exception.
        MessageBox.Show(ex.ToString())
    End Try
End Sub

```

- Run the application by pressing F5 or clicking on Debug > Start Debugging. You should now have the following screen in front of you.

ID	Name	Surname
1	Barbara	Smith
2	David	King
3	Michael	Young

Note that the DataGridView is already populated with the data from the Lecturer table in the database file. This is done on the Load event of the Windows Form using the GetLecturers() method.

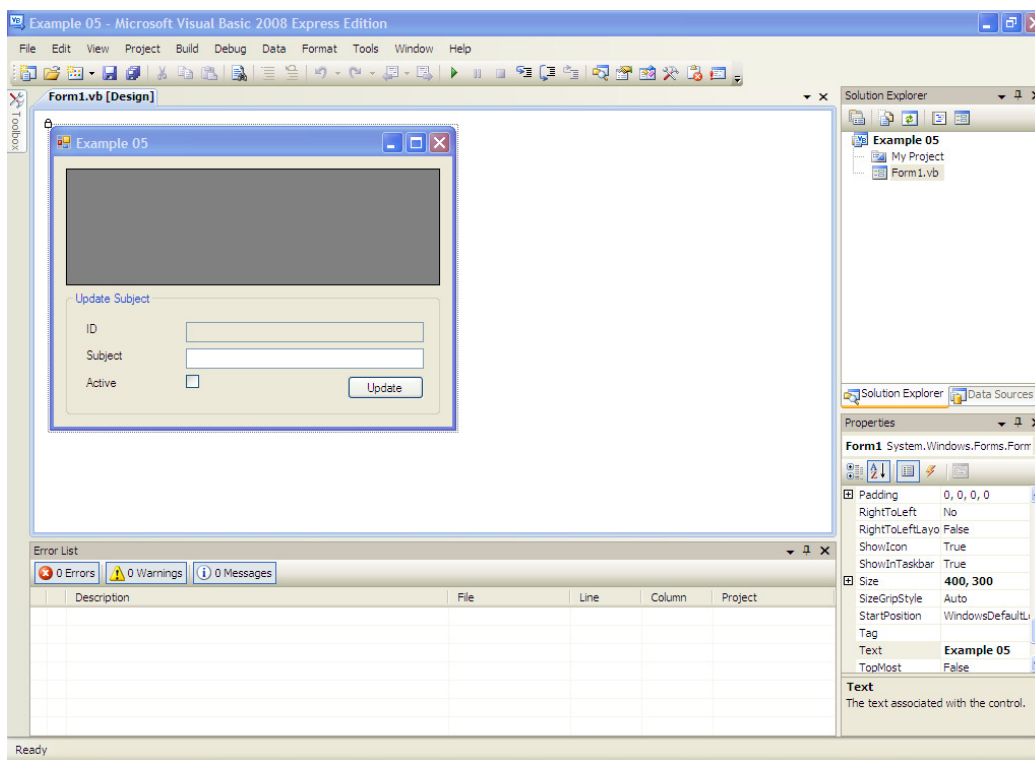
- Test the application by adding a "Name" and "Surname" for a new lecturer. Click on the "Insert" button and check whether the new lecturer is inserted and displayed in the DataGridView control.

ID	Name	Surname
1	Barbara	Smith
2	David	King
3	Michael	Young
4	John	Smith

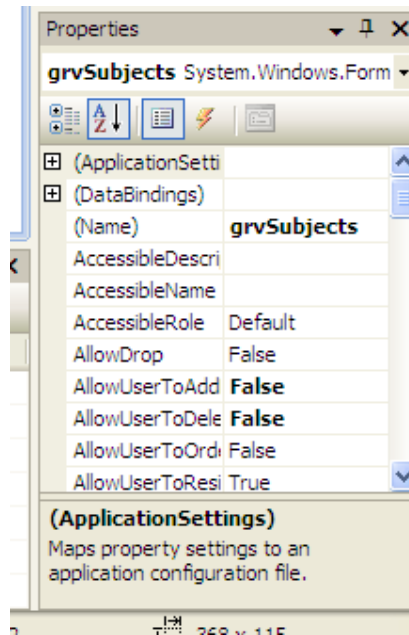
## Example 5: Updating an existing record

During this example, we will look at how to update an existing record. This example will make use of a DataGridView control to handle the selection of the record which we want to update, and secondly a set of input control to accept the changes we want to make to the selected record. We will be using the "Subject" table for this example. Same as with data insertion, we need to use the `ExecuteNonQuery()` method of the `OleDbCommand` class to perform the update transaction.

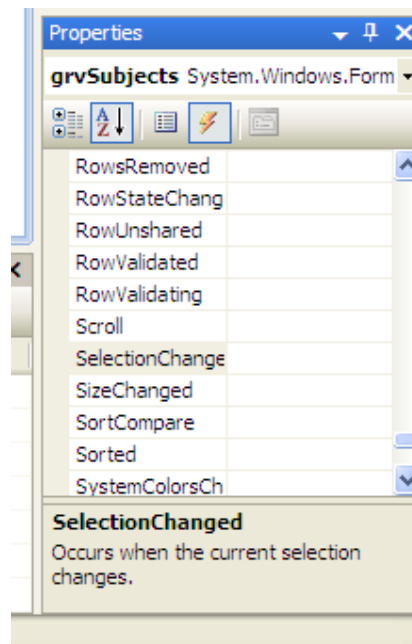
- Copy the source code of Example 05 to the hard disk drive or flash drive and open the solution with Microsoft Visual Basic 2008 Express Edition.
- Part of the screen for this application is already built. The DataGridView is already partially prepared for this tutorial, and we will only add the functionality of the "Update" button and selection changed event on the DataGridView. You should have the following screen in front of you.



- Add an event handler for the SelectionChanged event on the grvSubjects DataGridView control. To do this, click on the grvSubjects DataGridView control, move your mouse pointer to the "Properties" window.



- Switch to the “Events” section and double-click on the SelectionChanged event.



- The purpose of the SelectionChanged event handler is to update the input controls on the screen with the appropriate values contained in the existing record. On each SelectionChanged event, this event handler will fire and the correct data will be fetched from the database file and updated in the input controls.
- Add a Try-Catch block to the SelectionChanged event handler.

```
Try
Catch ex As Exception
End Try
```

- Firstly, we need to determine the ID value of the record that we are dealing with. The ID field is a unique value on each table and serves as the primary key of the table. Add the following line of code in the SelectionChanged event handler to determine the ID primary key value of the selected record:

```
' Fetch the selected ID from the DataGridView.
Dim ID = grvSubjects.CurrentRow.Cells("ID").Value
```

To fetch the ID field in the DataGridView, we use a combination of properties on the DataGridView. Let's look at these up close. The first property of concern is the "CurrentRow" property. This property returns the selected row of the DataGridView. The second property is the "Cells" property which returns a collection of cells for the particular row. We identify the "Cell" of concern by the name of the column as returned from the SQL query. For instance, if the column name from the database was called "ID", we use that caption to identify the cell in the DataGridView. Please note that if there is more than one column returned from the SQL query with the same name, the DataGridView will prefix the cell name with the table name. Lastly, we use the "Value" property to fetch the value of the cell that we've identified.

- Declare the connection, command and table objects.

```
' Declare the connection, command and table objects.
Dim connection As New OleDb.OleDbConnection
Dim command As New OleDb.OleDbCommand
Dim table As New Data.DataTable
```

- Initialize theConnectionString of the connection and open the database connection.

```
' Set the ConnectionString property of the connection and open the database
connection.
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Incomplete\Example 05\CollegeDB.mdb;User Id=;Password=;"
connection.Open()
```

- Initialize the command's Connection and CommandText properties.

```
' Set the Connection and CommandText properties of the command.
command.Connection = connection
command.CommandText = "SELECT * FROM Subject WHERE ID = @ID"
```

- Prepare and add the ID parameter for the SQL query. This is the value that we have fetched earlier in this code block.

```
' Declare and initialize the parameter for the command.
Dim parID = New OleDb.OleDbParameter("@ID", ID)

' Add the parameter to the command.
command.Parameters.Add(parID)
```

- Fetch the data from the database and store the result set in the table object.

```
' Execute the command and load the result set into the table.
table.Load(command.ExecuteReader())
```

- Close the database connection.

```
' Close the database connection.
connection.Close()
```

- Use an If statement to check whether a result set was returned. If a result set was returned, use the values to populate the input controls on the form, else initialize all the input controls to blank.

```
If table.Rows.Count > 0 Then ' Check whether a record was returned and load the
    results into the data input controls.
```



```

txtID.Text = table(0)("ID").ToString()
txtSubject.Text = table(0)("Name").ToString()
chkActive.Checked = table(0)("Active").ToString()
Else ' Else make all the input fields blank.
txtID.Text = ""
txtSubject.Text = ""
chkActive.Checked = False
End If

```

Note that we're using the first record in the result set. To identify the columns in the result set, we use the names of the columns as defined in the database file. For instance, if the name of the column in the database table is "Active", we may use that same name to identify the column in the result set if the column name was not explicitly named as something else in the SQL query.

- Add a prompt to the Catch clause to notify the user in the case of an exception:

```

' Prompt the user with the exception.
MessageBox.Show(ex.ToString())

```

- Let's summarize the code for the SelectionChanged event handler.

```

Private Sub grvSubjects_SelectionChanged(...)
Try
    ' Fetch the selected ID from the DataGridView.
    Dim ID = grvSubjects.CurrentRow.Cells("ID").Value

    ' Declare the connection, command and table objects.
    Dim connection As New OleDb.OleDbConnection
    Dim command As New OleDb.OleDbCommand
    Dim table As New Data.DataTable

    ' Set theConnectionString property of the connection and open the
    database connection.
    connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=C:\Examples\Incomplete\Example 05\CollegeDB.mdb;User
    Id=;Password=;"
    connection.Open()

    ' Set the Connection and CommandText properties of the command.
    command.Connection = connection
    command.CommandText = "SELECT * FROM Subject WHERE ID = @ID"

    ' Declare and initialize the parameter for the command.
    Dim parID = New OleDb.OleDbParameter("@ID", ID)

    ' Add the parameter to the command.
    command.Parameters.Add(parID)

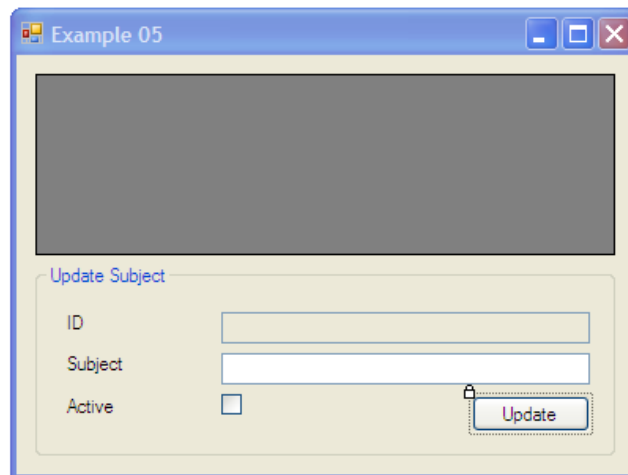
    ' Execute the command and load the result set into the table.
    table.Load(command.ExecuteReader())

    ' Close the database connection.
    connection.Close()

    If table.Rows.Count > 0 Then ' Check whether a record was returned and
    load the results into the data input controls.
        txtID.Text = table(0)("ID").ToString()
        txtSubject.Text = table(0)("Name").ToString()
        chkActive.Checked = table(0)("Active").ToString()
    Else ' Else make all the input fields blank.
        txtID.Text = ""
        txtSubject.Text = ""
        chkActive.Checked = False
    End If
Catch ex As Exception
    ' Prompt the user with the exception.
    MessageBox.Show(ex.ToString())
End Try
End Sub

```

- Next, we need to add the code for the "Update" button event handler. Switch back to the Design View of the screen.



- Add an event handler for the “Update” button by double-clicking on the “Update” button.
- Declare an Integer variable for use in this event handler. This Integer will be used for the validation and execution of the ID field in the database table.

```
Dim ID As Integer
```

- Next, we need to do a little bit of validation before executing the code. We are going to check whether a valid record has been selected. To do this check, we are going to use the Integer.TryParse() method. This method takes an input value, in our case the ID TextBox’s value, and an Integer variable to store the parsed value into. The Integer.TryParse() method returns a Boolean value to indicate whether the input value was indeed a valid Integer value, and if the input value was valid it is stored into the Integer variable provided. The statement to perform this check will look like the following.

```
If Integer.TryParse(txtID.Text, ID) Then ' Check whether a valid record is selected
    for updating.
    ' Code goes here.
End If
```

- The rest of the code for this event handler will be written within this If-statement.
- Add a Try-Catch block for the database transaction within the If-statement.

```
Try
Catch ex As Exception
End Try
```

- Declare the connection and command objects.

```
' Declare the connection and command objects.
Dim connection As New OleDb.OleDbConnection
Dim command As New OleDb.OleDbCommand
```

- Initialize theConnectionString of the connection and open the database connection.

```
' Set the ConnectionString property and open the database connection.
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Incomplete\Example 05\CollegeDB.mdb;User Id=;Password=;"
connection.Open()
```

- Set the Connection and CommandText properties of the command object.

```
' Set the Connection and CommandText properties of the command.
command.Connection = connection
command.CommandText = "UPDATE Subject SET Name = @Name, Active = @Active WHERE ID =
@ID"
```

Note that we have multiple parameters defined in the CommandText property. This is to specify all of the input values as well as to identify the particular record that we want to update.

- Prepare and add the parameters to the command object.

```
' Declare and initialize the parameters required for the command.
Dim parName = New OleDb.OleDbParameter("@Name", txtSubject.Text)
Dim parActive = New OleDb.OleDbParameter("@Active", chkActive.Checked)
Dim parID = New OleDb.OleDbParameter("@ID", txtID.Text)

' Add the parameters to the command object.
command.Parameters.Add(parName)
command.Parameters.Add(parActive)
command.Parameters.Add(parID)
```

Please note that it is very important to add the parameters in the order as they are specified in the SQL query on the CommandText property of the command object. For instance, this example requires that we add the "@Name" parameter first because it is the first parameter found in the SQL query. The same order goes for the remaining parameters.

- Execute the SQL query.

```
' Execute the command.
command.ExecuteNonQuery()
```

Remember that we are sending data to the database and we need to use the ExecuteNonQuery() method for this purpose.

- Close the database connection:

```
' Close the database connection.
connection.Close()
```

- Call the GetSubjects() method to repopulate the DataGridView with the updated data.

```
' Fetch the updated list of Subjects.
GetSubjects()
```

- Add a prompt in the Catch clause to notify the user of any exceptions.

```
' Prompt the user with the exception.
MessageBox.Show(ex.ToString())
```

- Let's summarize the code for the "Update" button event handler.

```
Private Sub btnUpdate_Click(...)
    Dim ID As Integer

    If Integer.TryParse(txtID.Text, ID) Then ' Check whether a valid record is
        selected for updating.
        Try
            ' Declare the connection and command objects.
            Dim connection As New OleDb.OleDbConnection
            Dim command As New OleDb.OleDbCommand

            ' Set the ConnectionString property and open the database
            connection.
            connection.ConnectionString =
                "Provider=Microsoft.Jet.OLEDB.4.0;Data
                Source=C:\Examples\Incomplete\Example
                05\CollegeDB.mdb;User Id=;Password=;"
            connection.Open()

            ' Set the Connection and CommandText properties of the command.
            command.Connection = connection
            command.CommandText = "UPDATE Subject SET Name = @Name, Active =
                @Active WHERE ID = @ID"

            ' Declare and initialize the parameters required for the
            command.
            Dim parName = New OleDb.OleDbParameter("@Name", txtSubject.Text)
            Dim parActive = New OleDb.OleDbParameter("@Active",
                chkActive.Checked)
            Dim parID = New OleDb.OleDbParameter("@ID", txtID.Text)

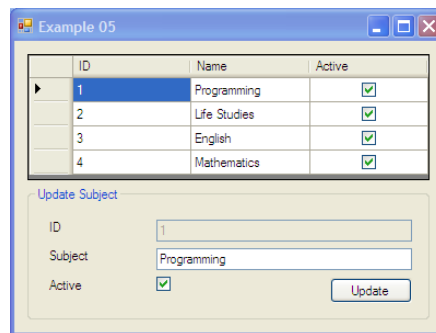
            ' Add the parameters to the command object.
            command.Parameters.Add(parName)
            command.Parameters.Add(parActive)
            command.Parameters.Add(parID)

            ' Execute the command.
            command.ExecuteNonQuery()

            ' Close the database connection.
            connection.Close()

            ' Fetch the updated list of Subjects.
            GetSubjects()
        Catch ex As Exception
            ' Prompt the user with the exception.
            MessageBox.Show(ex.ToString())
        End Try
    End If
End Sub
```

- Run the application by pressing F5 or clicking Debug > Start Debugging. You should now have the following screen in front of you.

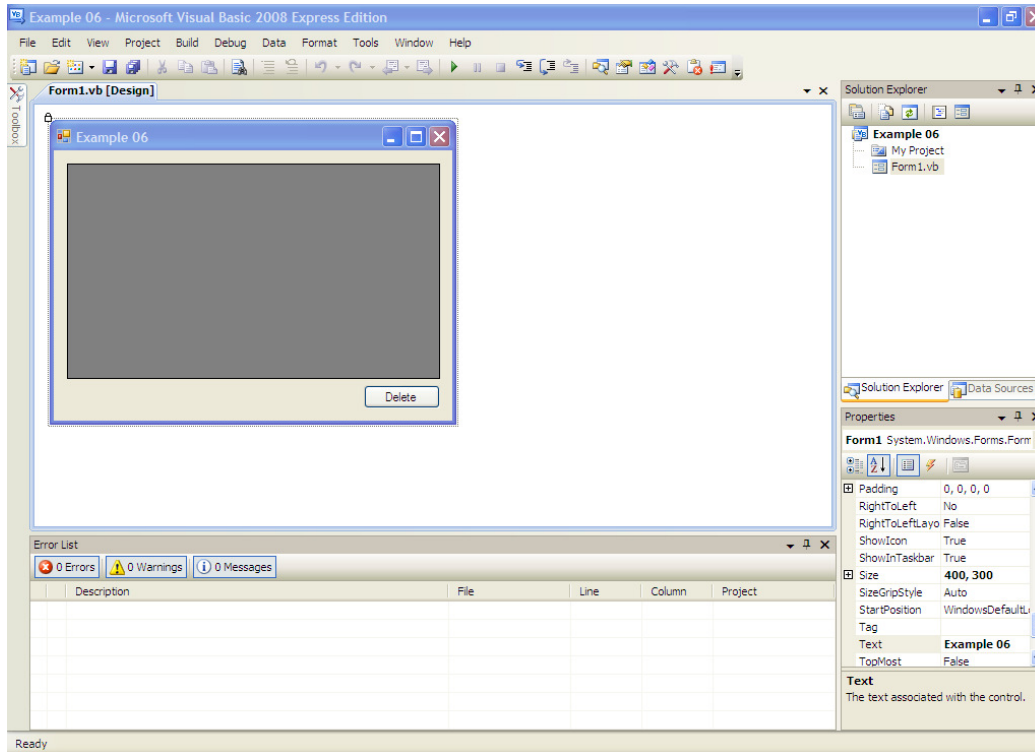


- Test the application by selecting a record in the DataGridView, edit the data in the "Subject" and "Active" input fields and click on the "Update" button to perform the changes. The DataGridView should also reflect the changes accordingly.

## Example 6: Deleting an existing record

During this example, we will look at how to delete an existing record from the database. We will use the "Registration" table in the College database for this example. To perform delete statements programmatically, is the same as doing an insert or update statement, the only difference is the SQL query used for the database transaction in the CommandText property of the Command object.

- Copy the source code for Example 06 to your hard disk drive or flash drive and open the solution with Microsoft Visual Basic 2008 Express Edition.
- You should now have the following screen in front of you.



The screen for this tutorial is very basic. It contains only a DataGridView and "Delete" button. The DataGridView will display the "Registration" records contained in the database and the "Delete" button allows us to delete the selected record.

- The data retrieval of the DataGridView is already written and we only need to add the code for the "Delete" button.
- Double-click on the Form1.vb item in the Object Explorer.
- Double-click on the "Delete" button to add an event handler for the button.
- In the "Delete" button's event handler, add a Try-Catch code block.

```
Try
Catch ex As Exception
End Try
```

- Declare the connection and command objects.

```
' Declare the connection and command objects.
Dim connection As New OleDb.OleDbConnection
Dim command As New OleDb.OleDbCommand
```

- Initialize the ConnectionString of the connection and open the database connection.

```
' Set the ConnectionString property and open the database connection.
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Incomplete\Example 06\CollegeDB.mdb;User Id=;Password="
connection.Open()
```

- Initialize the Connection and CommandText properties of the command.

```
' Set the Connection and CommandText properties.
command.Connection = connection
command.CommandText = "DELETE FROM Registration WHERE ID = @ID"
```

- Prepare and add the ID parameter to the command object.

```
' Declare and initialize the parameter required by the command.
Dim parID = New OleDb.OleDbParameter("@ID",
grvRegistrations.CurrentRow.Cells("ID").Value)
```

```
' Add the parameter to the command.
command.Parameters.Add(parID)
```

Note that we are using the same method of retrieving the ID parameter from the DataGridView as we've done in Example 05.

- Execute the DELETE transaction.

```
' Execute the command.
command.ExecuteNonQuery()
```

- Close the database connection.

```
' Close the database connection.
connection.Close()
```

- Add a call to the GetRegistrations() method to update the DataGridView with the new table data.

```
' Fetch the new result set of Registration records.
GetRegistrations()
```

- Add a prompt to the Catch clause to notify the user in the case of an exception.

```
' Prompt the user with the exception.
MessageBox.Show(ex.ToString())
```

- Let's summarize the event handler's code for the "Delete" button.

```
Private Sub btnDelete_Click(...)
    Try
        ' Declare the connection and command objects.
        Dim connection As New OleDb.OleDbConnection
        Dim command As New OleDb.OleDbCommand

        ' Set the ConnectionString property and open the database connection.
        connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Incomplete\Example 06\CollegeDB.mdb;User
Id=;Password="
        connection.Open()

        ' Set the Connection and CommandText properties.
        command.Connection = connection
```

```

command.CommandText = "DELETE FROM Registration WHERE ID = @ID"

' Declare and initialize the parameter required by the command.
Dim parID = New OleDb.OleDbParameter("@ID",
grvRegistrations.CurrentRow.Cells("ID").Value)

' Add the parameter to the command.
command.Parameters.Add(parID)

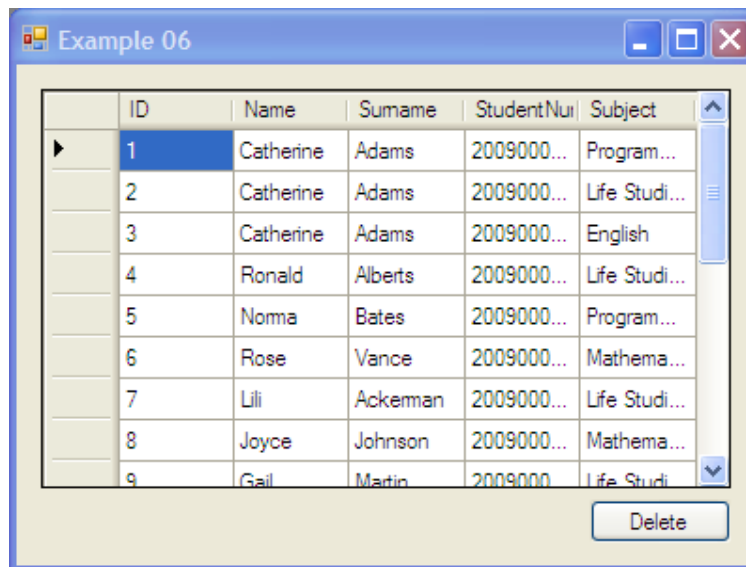
' Execute the command.
command.ExecuteNonQuery()

' Close the database connection.
connection.Close()

' Fetch the new result set of Registration records.
GetRegistrations()
Catch ex As Exception
' Prompt the user with the exception.
MessageBox.Show(ex.ToString())
End Try
End Sub

```

- Run the application by pressing F5 or clicking on Debug > Start Debugging. You should now have the following screen in front of you.



- Test the application by selecting a record and click on the "Delete" button. The record should be removed from the list of records displayed in the DataGridView.

## Validations

Validations are a very important part of a program, especially when data is involved. Imagine the problem that could surface if no data is tested and verified in a database application. Student details could be missing or incorrect, or the application could fail because of incorrect input such as string input to a number field could lead to the application failure. Validating data is a very specific duty and a programmer would need to identify which fields require validation and also the type of validation to apply. Below is a list of different types of validations commonly found in a database application.

1. **Required:** This means that a field is required in order to proceed with the program flow. For instance, the name and surname of a student would be required fields because we use those two fields to identify the student when the student number is not available.
2. **Type:** Data type validation is the practice of checking the input value against the required data type. For instance, an outstanding balance field stores the financial outstanding balance of a student which is of decimal data type. If a text input is given, the application could fail by sending an invalid value to the database.
3. **Length:** Validating the length of an input is also important for fields that have a maximum storage capacity in the database. For instance, to make the application more user-friendly and easy to use, we can put a length validation on the subject name field. This validation would stop the user from entering a name that is too long for the database field or otherwise ease the readability of the name for use in the system.

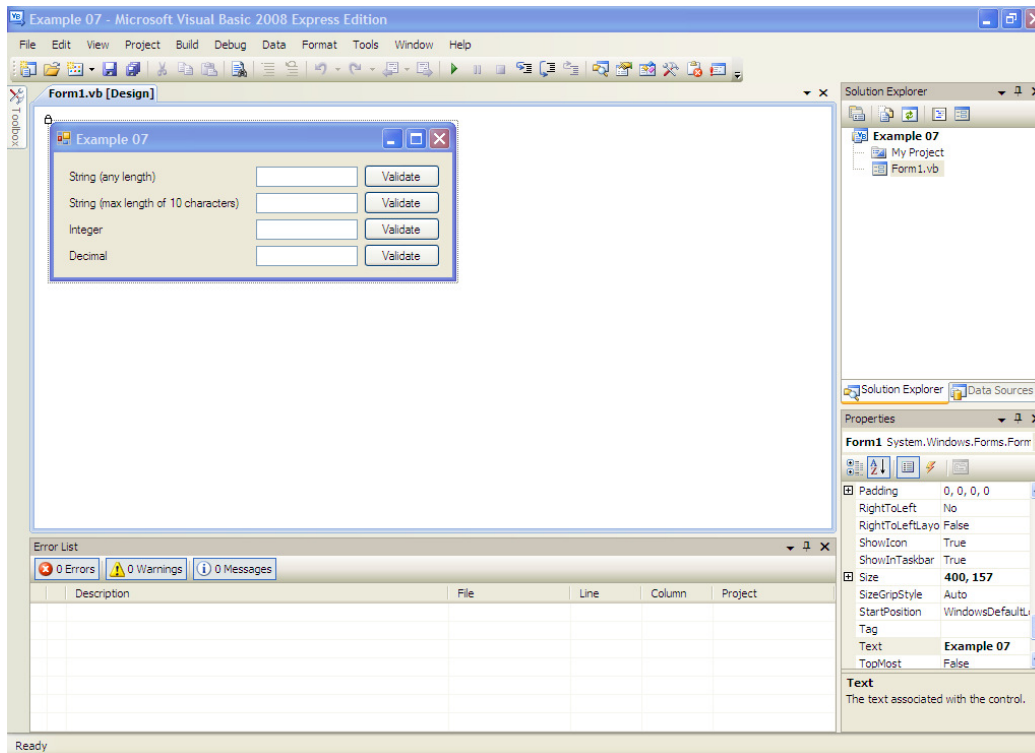
In the next example, we will look at how to implement the abovementioned validations. It is considered good programming practice to do the proper validations rather than relying on the failover mechanism of the programming language because the failover mechanisms such as Try-Catch blocks are to be used for more complex situations rather than checking for data integrity and validity.

### Example 7: Validating input.

During this example, we will look at how to implement basic validations to our code in order for us to build more robust database applications.

- Copy the source code for Example 07 to your hard disk drive or flash drive and open the solution using Microsoft Visual Basic 2008 Express Edition.
- Double-click on the Form1.vb item in the Solution Explorer. You should now have the following screen in front of you.





- The visual part of the form is already built and we only need to add the necessary code.
- On the form, you'll find four "Validate" buttons. Each button has its own "validation" purpose. We will add code for each button individually. Double-click on the first "Validate" button to add an event handler for it. You should now have the code file in front of you.
- The String class has a method called `IsNullOrEmpty()` which takes a string argument and checks whether the input string is Null or Empty. For this event handler, we'll be using this method along with an If statement to check the validity of the input string and prompt the user with a MessageBox accordingly.

```
If String.IsNullOrEmpty(txtStringAnyLength.Text) Then ' Check whether the input string
is empty and prompt the user accordingly.
    MessageBox.Show("Input string is invalid")
Else ' Else prompt the user that the input was valid.
    MessageBox.Show("Input is valid")
End If
```

- This piece of code will take the first TextBox on the form and test its input's validity.
- Switch back to the design view of the form. Double-click on the second "Validate" button to create an event handler for it.
- For the next validation, we'll be doing two validations. The first being whether the input string is empty or null using the previous validation method and also to check the maximum number of characters. We are going to use an If statement with an additional ElseIf clause for the second validation.

```
If String.IsNullOrEmpty(txtStringMaxLength.Text) Then ' Check whether the input
string is empty and prompt the user accordingly.
    MessageBox.Show("Input string is invalid")
Elseif txtStringMaxLength.Text.Length > 10 Then ' Check whether the input string is
longer than 10 characters and prompt the user accordingly.
    MessageBox.Show("Input string is too long, maximum of 10 characters is
allowed.")
```

```
Else ' Else prompt the user that the input was valid.
    MessageBox.Show("Input is valid")
End If
```

- This validation will use the result from the `String.IsNullOrEmpty()` method and if the input string is not empty or null, the second part of the If statement will execute. The next part will check whether the input string is within the boundary of 10 characters. If both validations succeed, the user will receive a confirmation on the screen that the input string is correct. In the case of a validation not succeeding, the user will receive a notification on the interface with the error.
- Next, we are going to validate for a valid Integer value. Switch back to the design view of the form and double-click on the third "Validate" button to create an event handler for it.
- In the event handler, declare an Integer variable for use in the validation.

```
' Declare an Integer type variable for use in the validation.
Dim int As Integer
```

- The Integer class contains a method `Integer.TryParse()` which takes an input string and tries to convert it to an Integer value. The method takes two arguments, namely the input string and the output value stored in a variable. This is where the Integer variable we declared in the previous step comes into play. If the method succeeds in parsing the input string, the parsed value is stored into the output variable and a "True" state is returned by the method, else a "False" state is returned. The result is then used in the If statement to determine the validity based on the state returned from the `Integer.TryParse()` method.

```
If Not Integer.TryParse(txtInteger.Text, int) Then ' Use the Integer.TryParse() method
    to check whether the input was an invalid Integer and prompt the user
    accordingly.
    MessageBox.Show("Input is not in the correct format, only numerals are allowed
    '0 - 9'")
Else ' Else prompt the user that the input was valid.
    MessageBox.Show("Input is valid")
End If
```

- When the If statement has completed its evaluation, the user will receive a prompt indicating whether the input value is valid or not valid.
- Lastly, we'll validate the input value as a Decimal. Validating a Decimal and Integer is basically the same except that we'll use the Decimal class' `TryParse()` method instead of the Integer class.
- Switch back to the design view of the form and double-click on the last "Validate" button on the form to create an event handler for it.
- Basically, this event handler is a duplication of the previous validation except that we are using the Decimal class instead.

```
' Declare an Decimal type variable for use in the validation.
Dim dec As Decimal

If Not Decimal.TryParse(txtDecimal.Text, dec) Then ' Use the Decimal.TryParse() method
    to check whether the input was an invalid Decimal and prompt the user
    accordingly.
    MessageBox.Show("Input is not in the correct format, only decimal numerals are
    allowed '0 - 9' including '.'")
Else ' Else prompt the user that the input was valid.
    MessageBox.Show("Input is valid")
End If
```

- Let's summarize the code for the entire form.

```
Private Sub btnValidateStringAnyLength_Click(...)
```

```

        If String.IsNullOrEmpty(txtStringAnyLength.Text) Then ' Check whether the input
            string is empty and prompt the user accordingly.
            MessageBox.Show("Input string is invalid")
        Else ' Else prompt the user that the input was valid.
            MessageBox.Show("Input is valid")
        End If
    End Sub

Private Sub btnValidateStringMaxLength_Click(...)
    If String.IsNullOrEmpty(txtStringMaxLength.Text) Then ' Check whether the input
        string is empty and prompt the user accordingly.
        MessageBox.Show("Input string is invalid")
    ElseIf txtStringMaxLength.Text.Length > 10 Then ' Check whether the input
        string is longer than 10 characters and prompt the user accordingly.
        MessageBox.Show("Input string is too long, maximum of 10 characters is
            allowed.")
    Else ' Else prompt the user that the input was valid.
        MessageBox.Show("Input is valid")
    End If
End Sub

Private Sub btnValidateInteger_Click(...)
    ' Declare an Integer type variable for use in the validation.
    Dim int As Integer

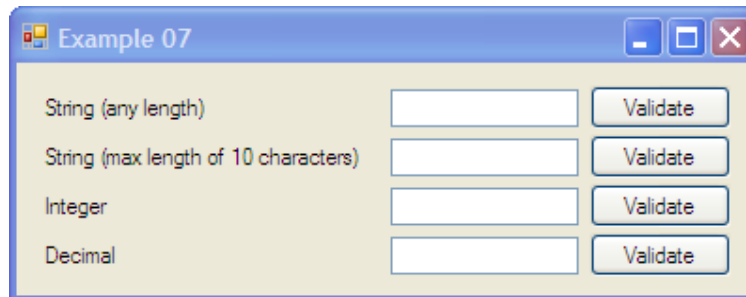
    If Not Integer.TryParse(txtInteger.Text, int) Then ' Use the
        Integer.TryParse() method to check whether the input was an invalid
        Integer and prompt the user accordingly.
        MessageBox.Show("Input is not in the correct format, only numerals are
            allowed '0 - 9'")
    Else ' Else prompt the user that the input was valid.
        MessageBox.Show("Input is valid")
    End If
End Sub

Private Sub btnValidateDecimal_Click(...)
    ' Declare an Decimal type variable for use in the validation.
    Dim dec As Decimal

    If Not Decimal.TryParse(txtDecimal.Text, dec) Then ' Use the Decimal.TryParse()
        method to check whether the input was an invalid Decimal and prompt the
        user accordingly.
        MessageBox.Show("Input is not in the correct format, only decimal
            numerals are allowed '0 - 9' including '.'")
    Else ' Else prompt the user that the input was valid.
        MessageBox.Show("Input is valid")
    End If
End Sub

```

- Run the application by pressing F5 or clicking on Debug > Start Debugging.



- Below is a list of screenshots of possible outcomes for this example application.
  - Validation 1: Invalid string input.

The screenshot shows a window titled "Example 07" with a light blue title bar and standard Windows window controls. The main area has a light tan background. It contains four rows of input fields with labels to their left and "Validate" buttons to their right:

- String (any length): [Empty text box] [Validate button]
- String (max length of 10 characters): [Empty text box] [Validate button]
- Integer: [Empty text box] [Validate button]
- Decimal: [Empty text box] [Validate button]

This screenshot shows the same "Example 07" window, but with an error dialog box overlaid in the center. The dialog has a blue title bar with a red "X" icon and contains the text "Input string is invalid" and an "OK" button. The input fields in the background are still empty.

- Validation 1: Valid string input.

The screenshot shows the "Example 07" window with the first input field, "String (any length)", now containing the text "This is text". The other three input fields remain empty. All "Validate" buttons are still present and enabled.

This screenshot shows the "Example 07" window with the same text "This is text" in the first input field. A success dialog box is overlaid in the center, with a blue title bar, a red "X" icon, and the text "Input is valid" and an "OK" button. The other input fields remain empty.

- Validation 2: Invalid string input.

The 'Example 07' window displays four validation rows. The first row, 'String (any length)', has the input 'This is text' and a 'Validate' button. The second row, 'String (max length of 10 characters)', has the input 'More than 10 chars' and a 'Validate' button. The third row, 'Integer', has an empty input field and a 'Validate' button. The fourth row, 'Decimal', has an empty input field and a 'Validate' button.

The 'Example 07' window shows an error message dialog box overlaid on the 'String (max length of 10 characters)' row. The dialog box contains the text 'Input string is too long, maximum of 10 characters is allowed.' and an 'OK' button. The input field for this row contains 'More than 10 chars'.

- Validation 2: Valid string input.

The 'Example 07' window displays four validation rows. The first row, 'String (any length)', has the input 'This is text' and a 'Validate' button. The second row, 'String (max length of 10 characters)', has the input '10 chars' and a 'Validate' button. The third row, 'Integer', has an empty input field and a 'Validate' button. The fourth row, 'Decimal', has an empty input field and a 'Validate' button.

The 'Example 07' window shows a success message dialog box overlaid on the 'String (max length of 10 characters)' row. The dialog box contains the text 'Input is valid' and an 'OK' button. The input field for this row contains '10 chars'.

- Validation 3: Invalid integer input.

The 'Example 07' window displays four input fields with their respective validation buttons:

Field Label	Input	Action
String (any length)	This is text	Validate
String (max length of 10 characters)	10 chars	Validate
Integer	aaa	Validate
Decimal		Validate

The 'Example 07' window is shown with an error dialog box overlaid. The dialog box contains the message: "Input is not in the correct format, only numerals are allowed '0 - 9'". The dialog box has an "OK" button and a close button (X).

- Validation 3: Valid integer input.

The 'Example 07' window displays the same four input fields as the previous screenshot, but with the following inputs:

Field Label	Input	Action
String (any length)	This is text	Validate
String (max length of 10 characters)	10 chars	Validate
Integer	5	Validate
Decimal		Validate

The 'Example 07' window is shown with a success dialog box overlaid. The dialog box contains the message: "Input is valid". The dialog box has an "OK" button and a close button (X).

- Validation 4: Invalid decimal input.

The 'Example 07' dialog box contains four rows of input fields and 'Validate' buttons:

String (any length)	<input type="text" value="This is text"/>	<input type="button" value="Validate"/>
String (max length of 10 characters)	<input type="text" value="10 chars"/>	<input type="button" value="Validate"/>
Integer	<input type="text" value="5"/>	<input type="button" value="Validate"/>
Decimal	<input type="text" value="aaa"/>	<input type="button" value="Validate"/>

An error message dialog box with a blue title bar and a close button. The message text reads: "Input is not in the correct format, only decimal numerals are allowed '0 - 9' including '.'". Below the message is an "OK" button.

- Validation 4: Valid decimal input.

The 'Example 07' dialog box contains four rows of input fields and 'Validate' buttons:

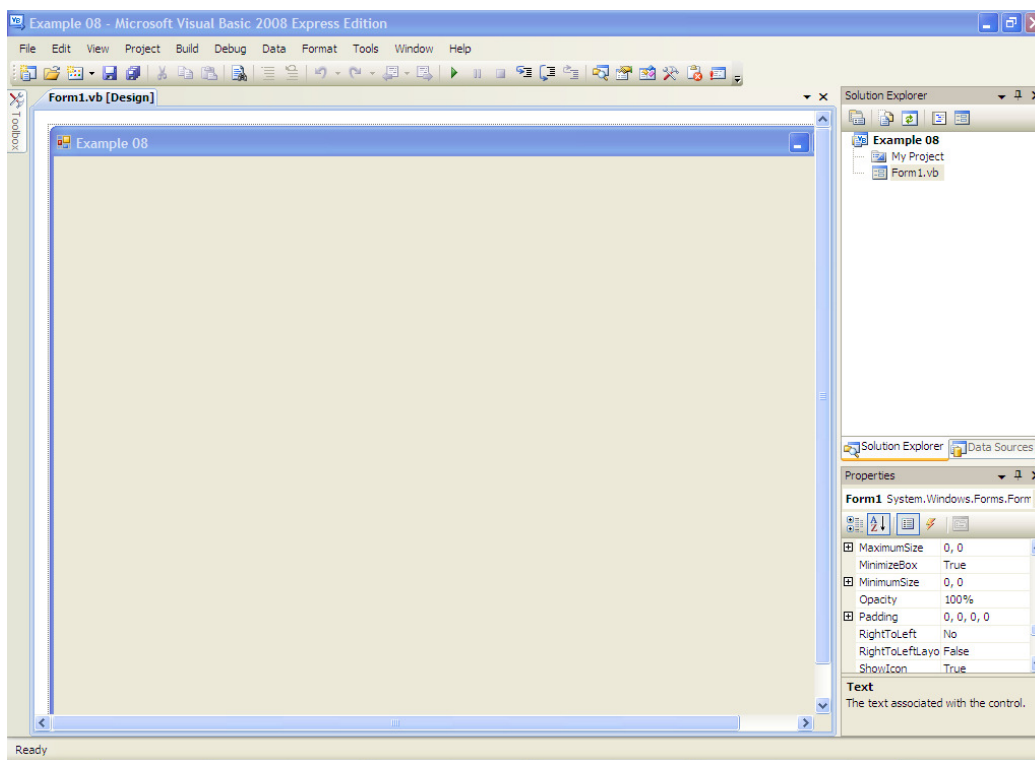
String (any length)	<input type="text" value="This is text"/>	<input type="button" value="Validate"/>
String (max length of 10 characters)	<input type="text" value="10 chars"/>	<input type="button" value="Validate"/>
Integer	<input type="text" value="5"/>	<input type="button" value="Validate"/>
Decimal	<input type="text" value="9.95"/>	<input type="button" value="Validate"/>

The 'Example 07' dialog box is shown with a small modal window overlaid on top. The modal window has a blue title bar, a close button, and the text "Input is valid". Below the text is an "OK" button. The background dialog box shows the same four rows of input fields and 'Validate' buttons as in the previous screenshot.

## Example 8: Creating a form to insert, update and delete records.

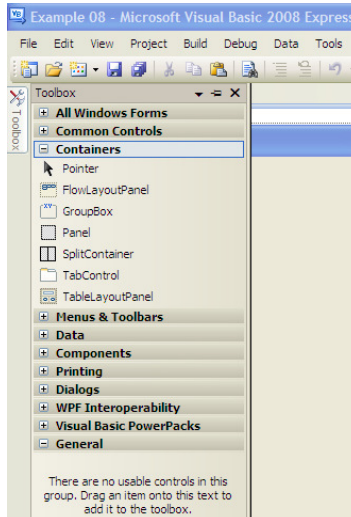
This example summarizes the insertion, updating and deletion data transaction into one form so that we may see how a real-life data form would look like and how to program such a form. The form also implements the data validations discussed in the previous example. Finally, a lot of the functionality in this example is a revision of examples 3, 4, 5 and 6 but it is important to understand how to implement the functionality of each example into one form. We will be using the Student table for this example.

- Copy the source code for Example 08 to your hard disk drive or flash drive and open the solution with Microsoft Visual Basic 2008 Express Edition.
- Double-click on the Form1.vb item in the Solution Explorer. You should now have the following screen in front of you.

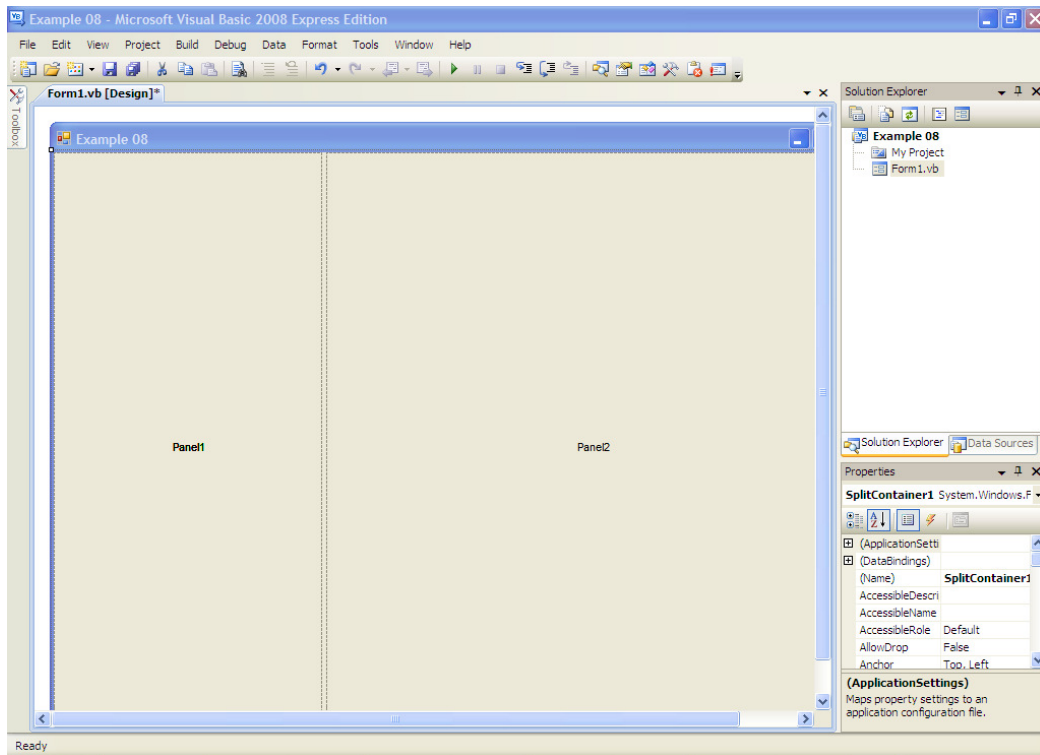


- This example will look at how to construct the user interface from the beginning; hence you'll have a blank form in front of you.
- Add a SplitContainer to the form. To do this, mouse-over on the ToolBox.

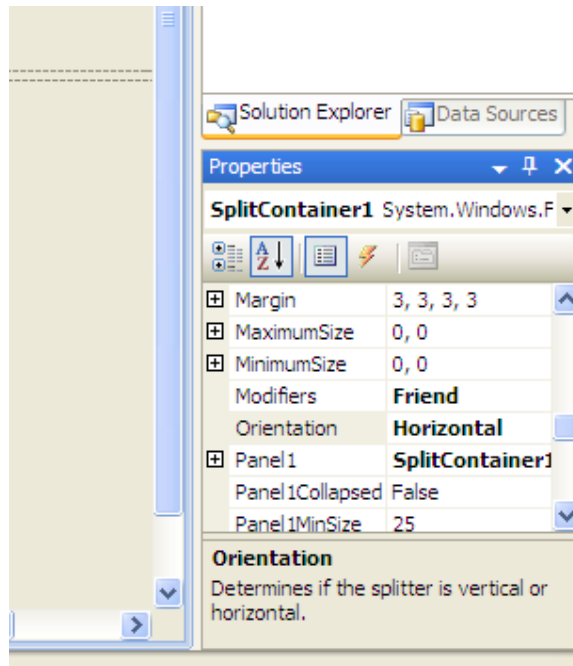




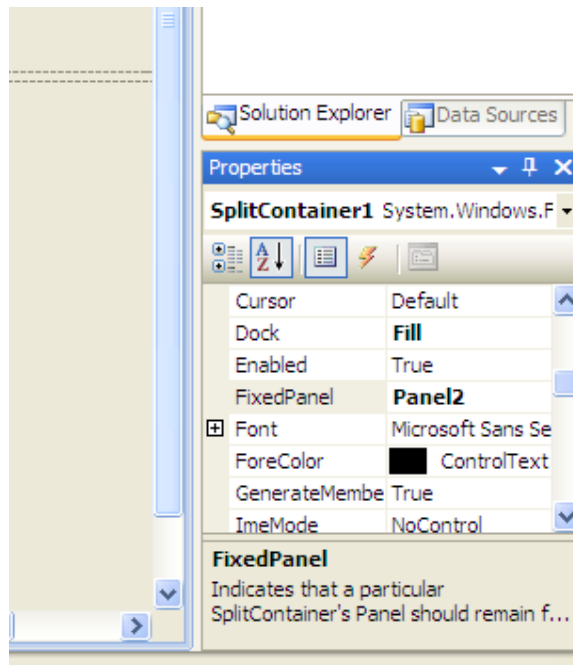
- Expand the Containers category and double-click on the SplitContainer item to add it to the form.



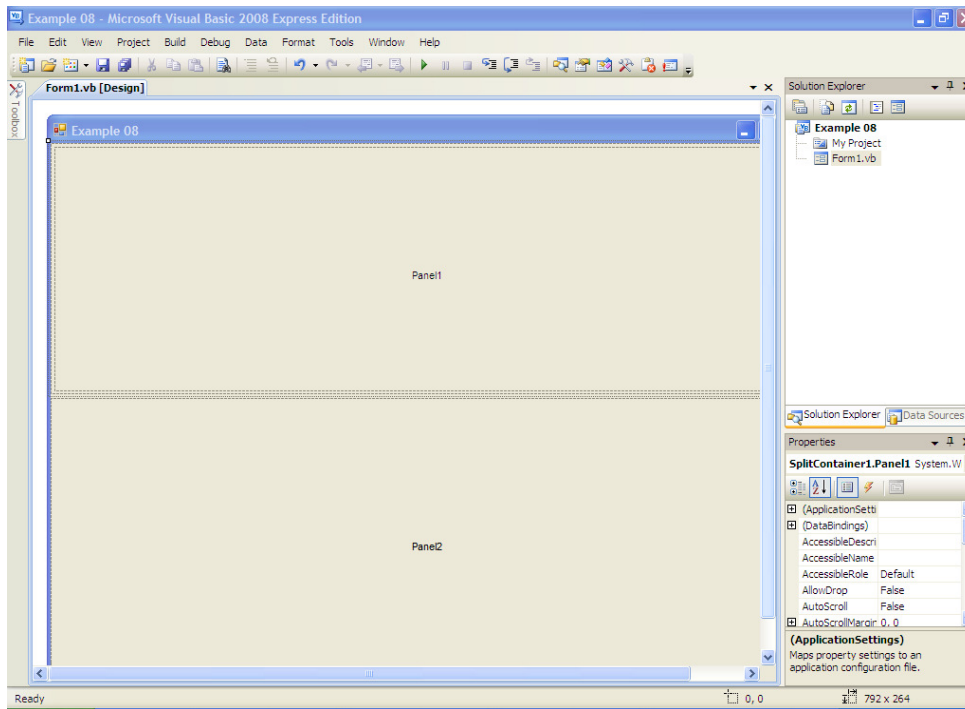
- Go to the Properties window of the SplitContainer, locate the Orientation property and set it to Horizontal.



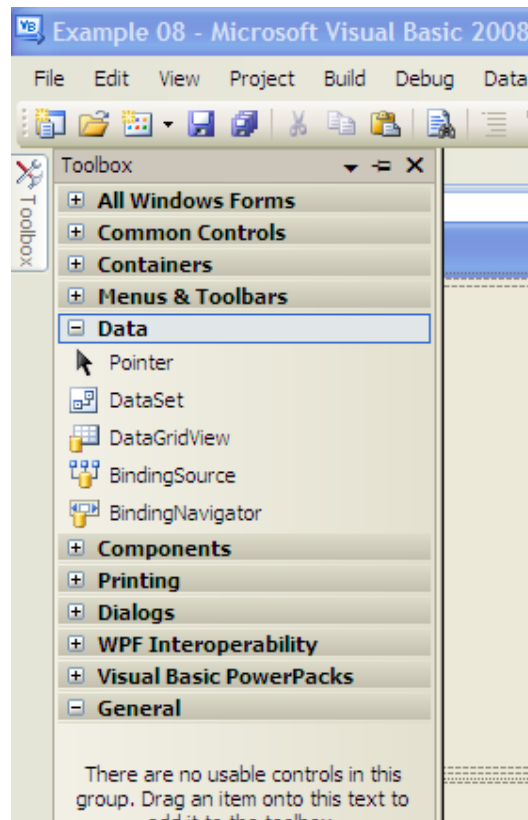
- The SplitContainer control will split the form into two sections. The top section will be used for data display and the bottom section for the data input.
- Additionally, we also need to set the FixedPanel property of the SplitContainer to Panel2. This will cause the top section of the form to resize dynamically whilst the bottom section will remain the same when the form is being resized.



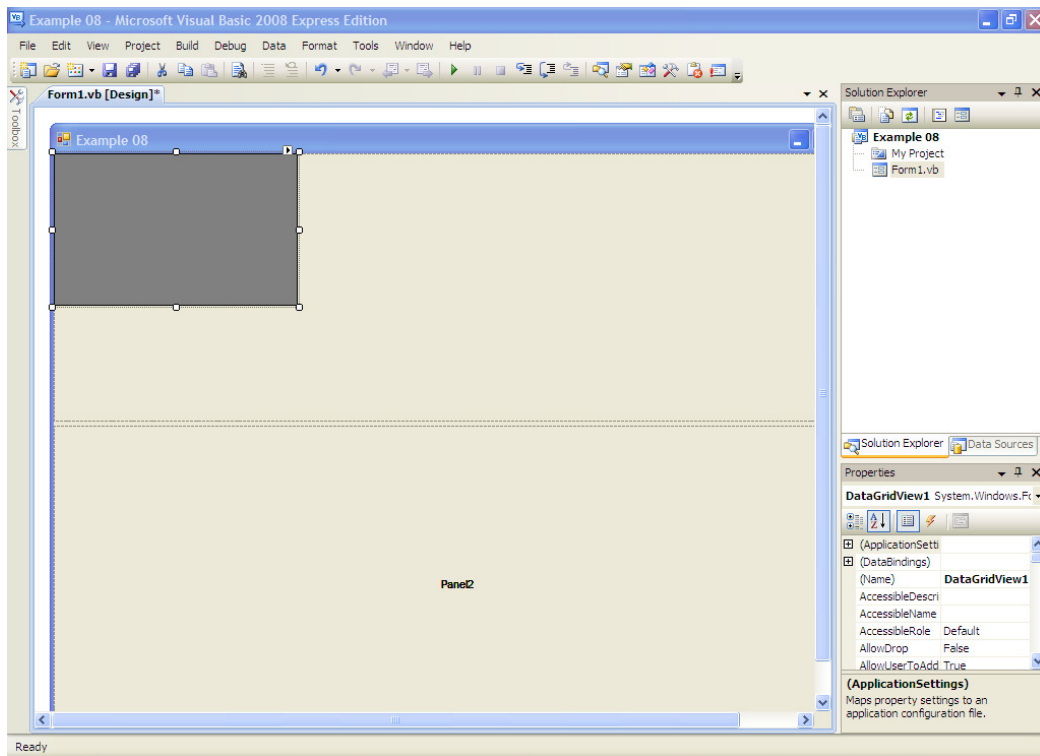
- Next, we need to add a DataGridView to the form for data display. First, select Panel1 of the SplitContainer by clicking on the top section of the SplitContainer.



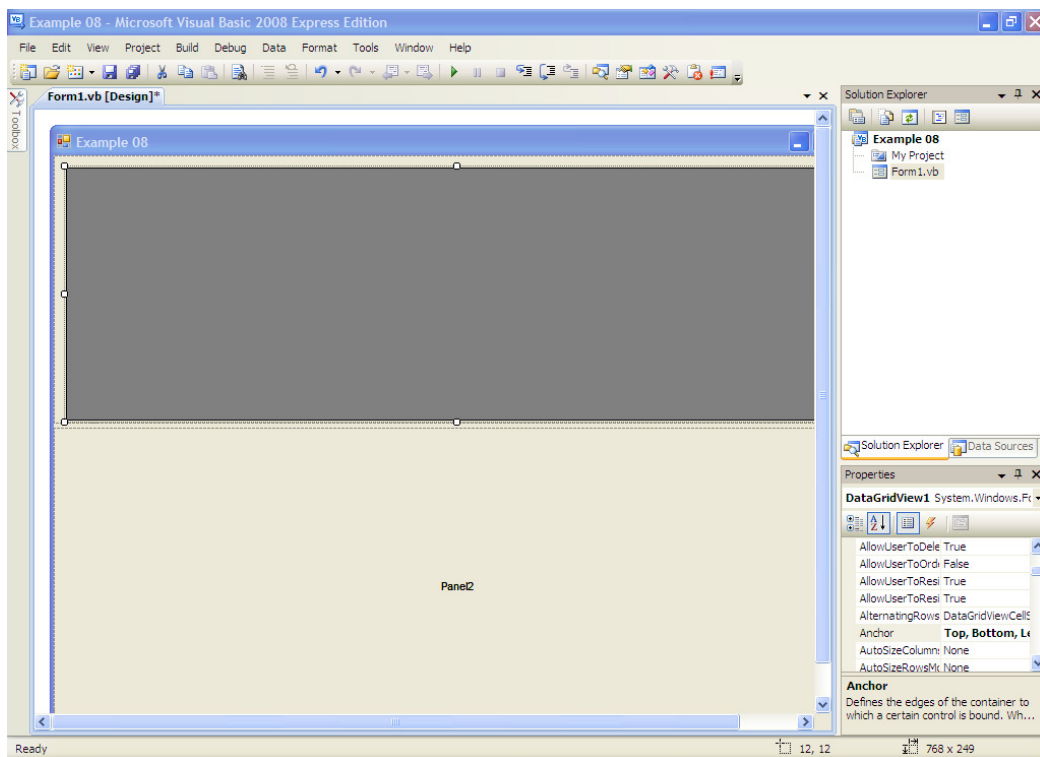
- Mouse-over on the ToolBox and expand the Data category.



- Double-click on the DataGridView control in the list to add it to the form.



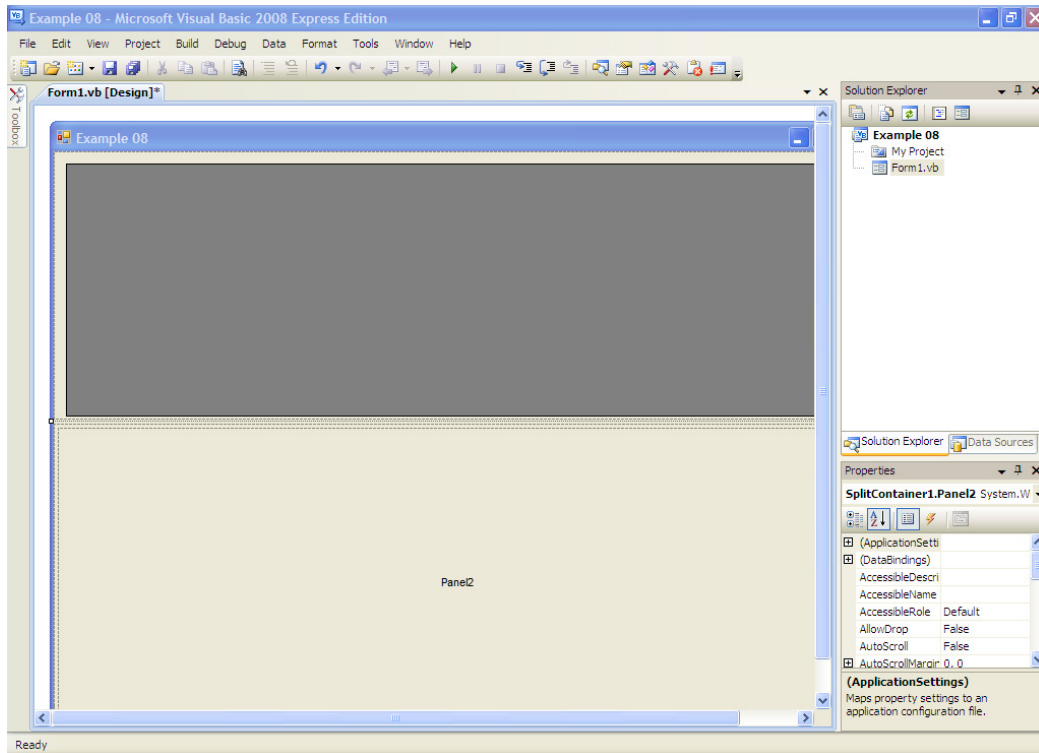
- Expand the DataGridView to the entire panel and set the Anchor property of the DataGridView to Top, Bottom, Left, Right.



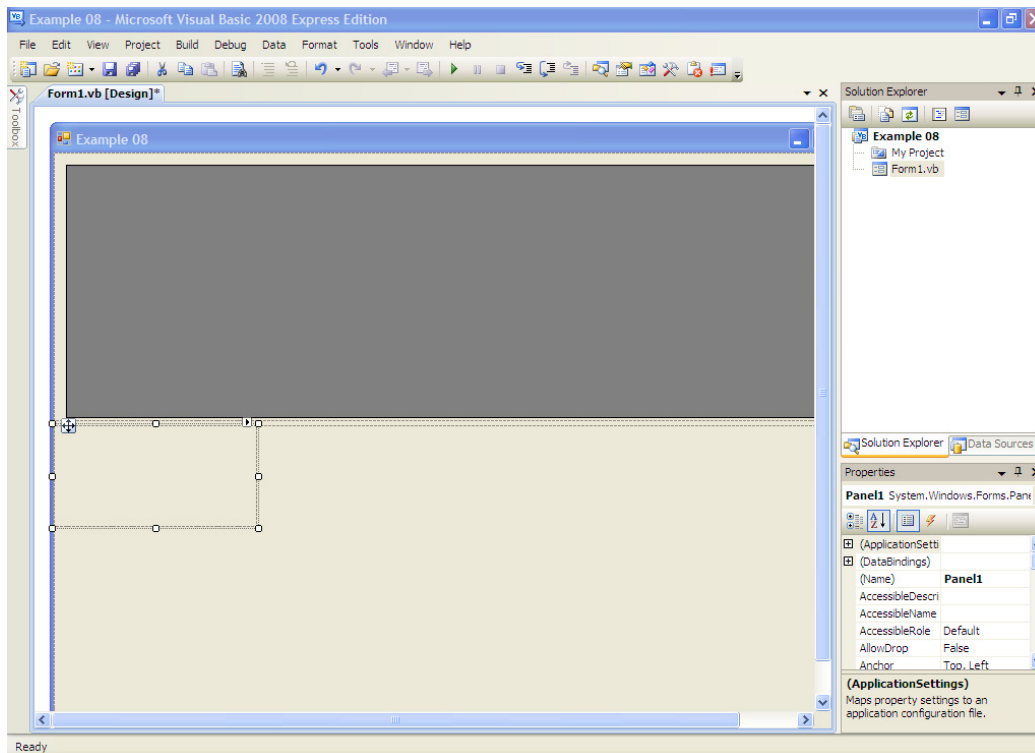
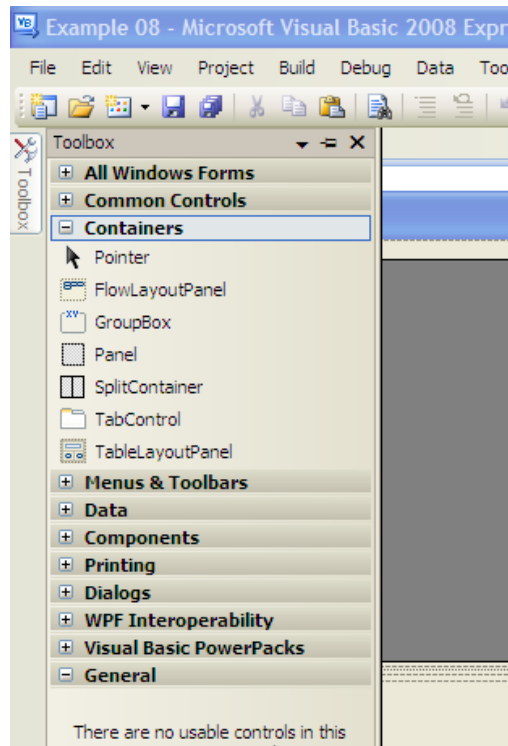
- Rename the DataGridView control to grvStudents and set the following properties of the DataGridView according to the table below.

Property	Value
<b>AllowUserToAddRows</b>	<b>False</b>
<b>AllowUserToDeleteRows</b>	<b>False</b>
<b>AutoSizeColumnsMode</b>	<b>Fill</b>
<b>MultiSelect</b>	<b>False</b>
<b>ReadOnly</b>	<b>True</b>

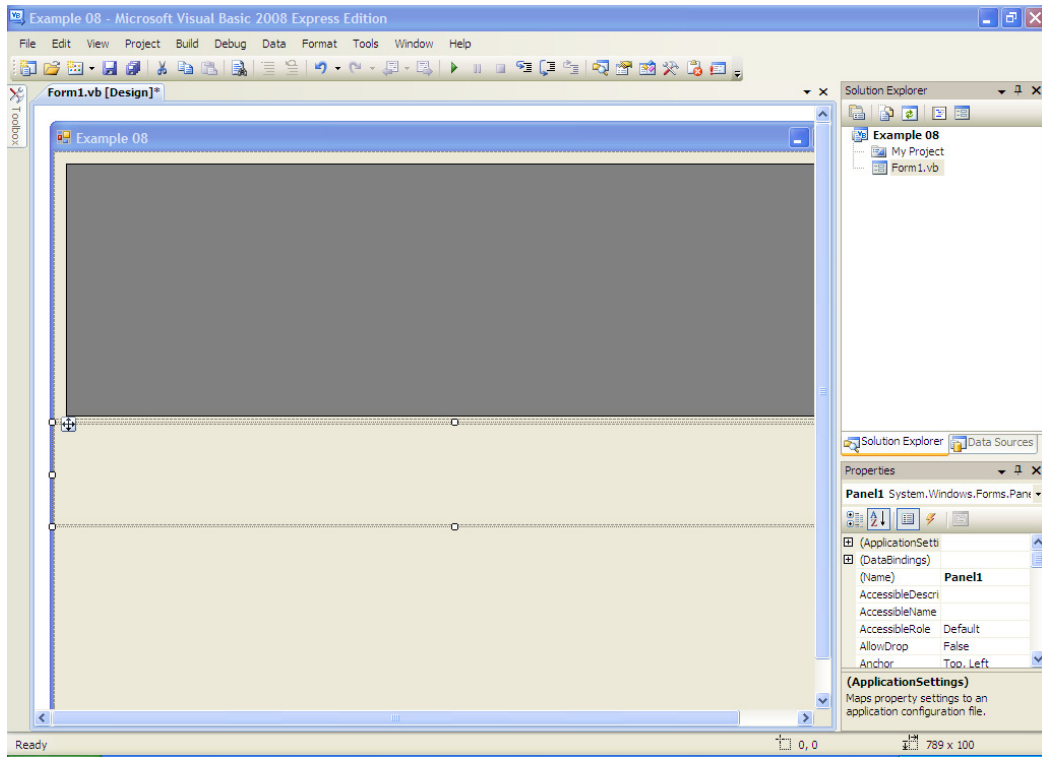
- Select Panel2 of the SplitContainer by clicking on the bottom section of the SplitContainer.



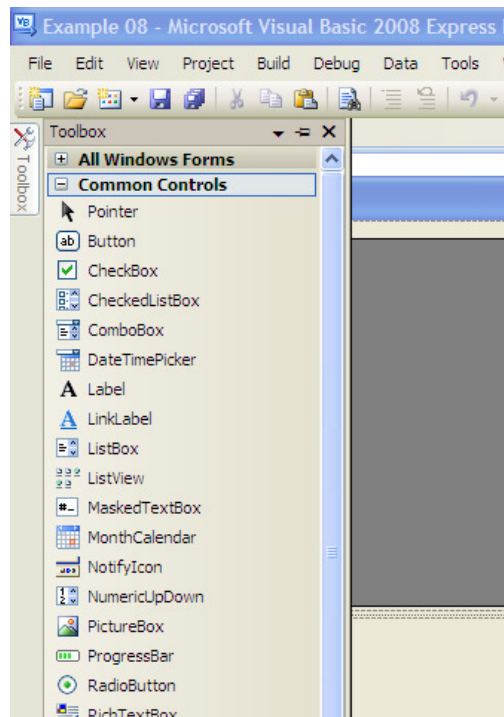
- Add a panel to the bottom section of the SplitContainer. To do this, mouse-over the ToolBox and expand the Containers category and double click on the Panel item in the list.

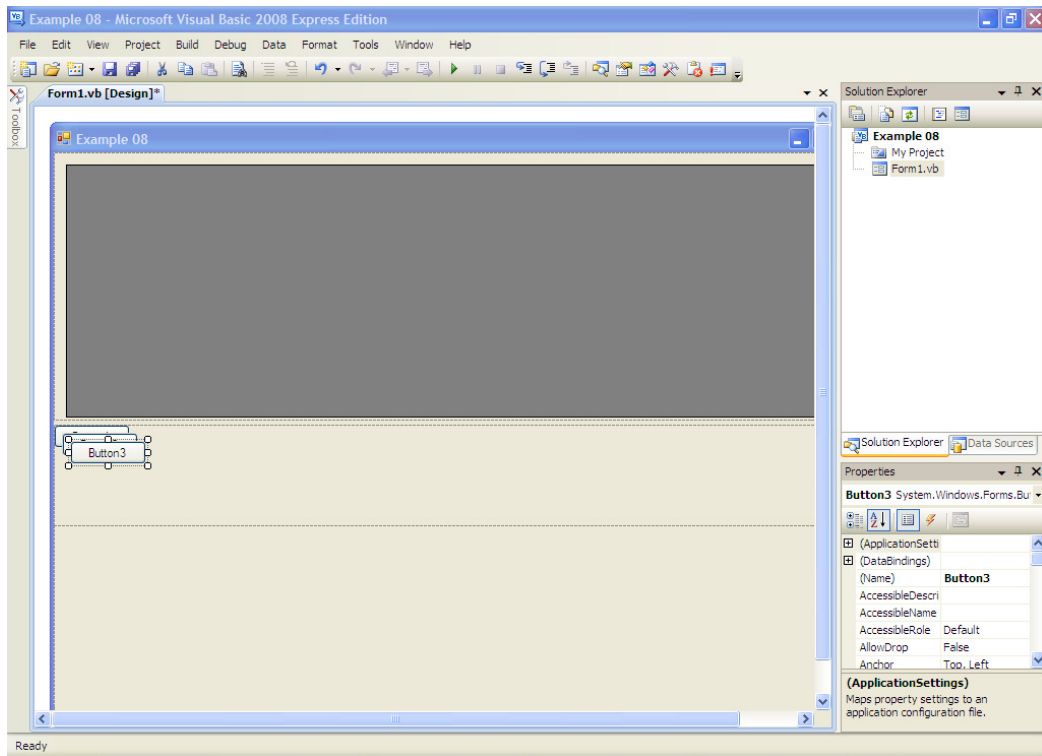


- Resize the panel to the width of the form.

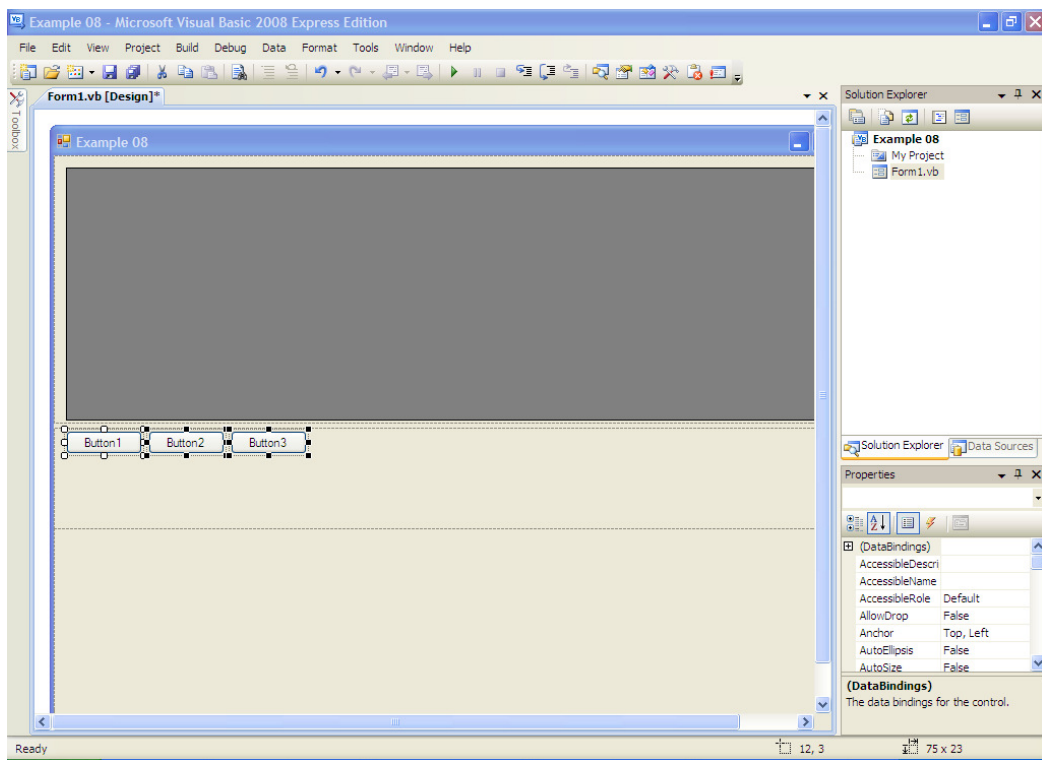


- With the panel still selected, add three buttons to the panel and align them next to each other on the left of the panel. To do this, mouse-over the ToolBox, expand the Common Controls category and double-click on the Button item. Repeat this process until you have three buttons on the form.



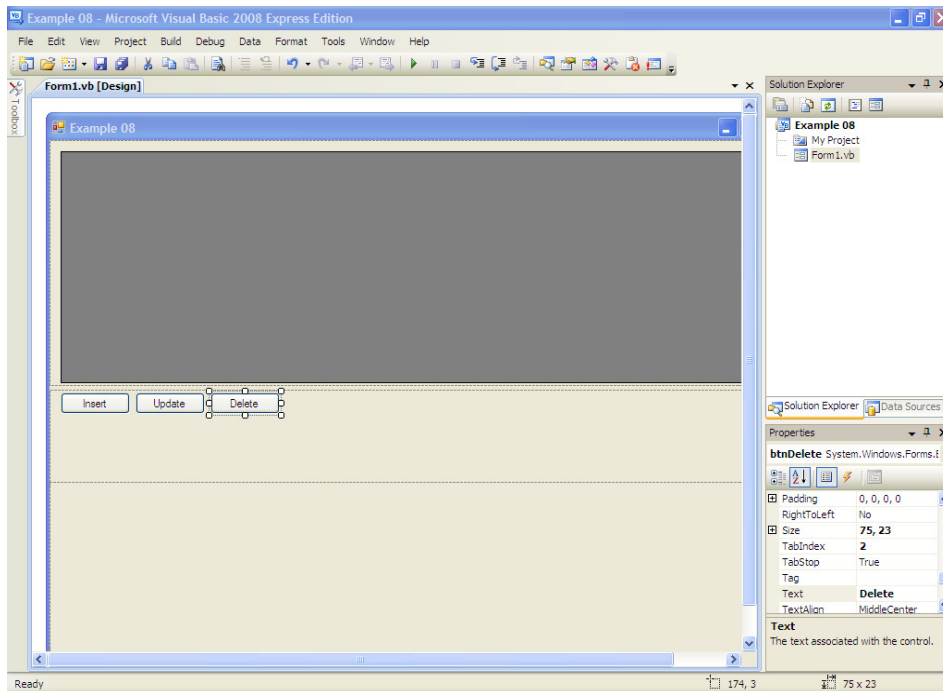


- Remember to align the buttons.

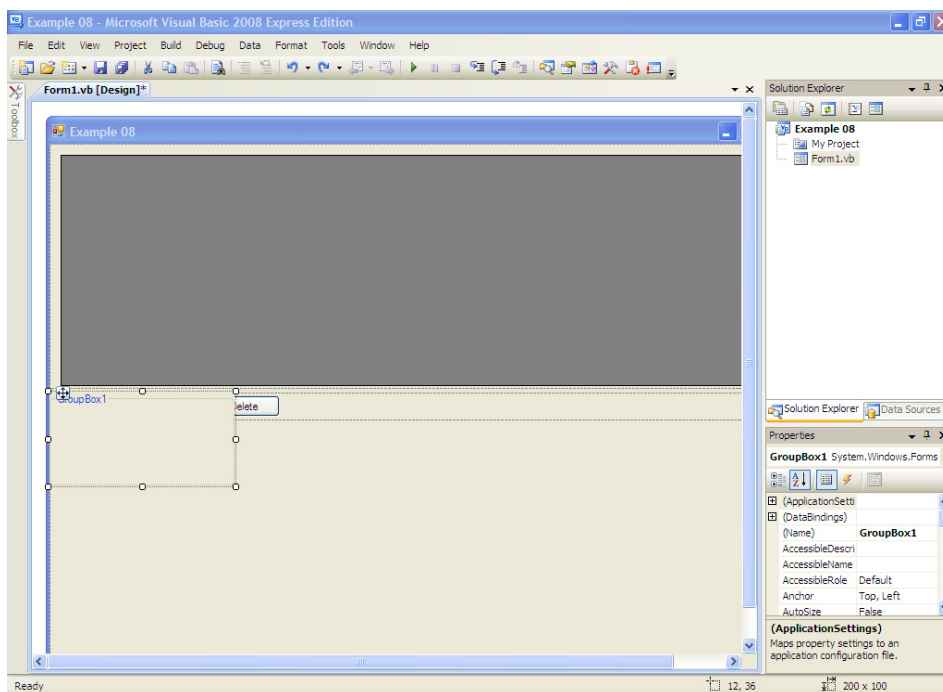




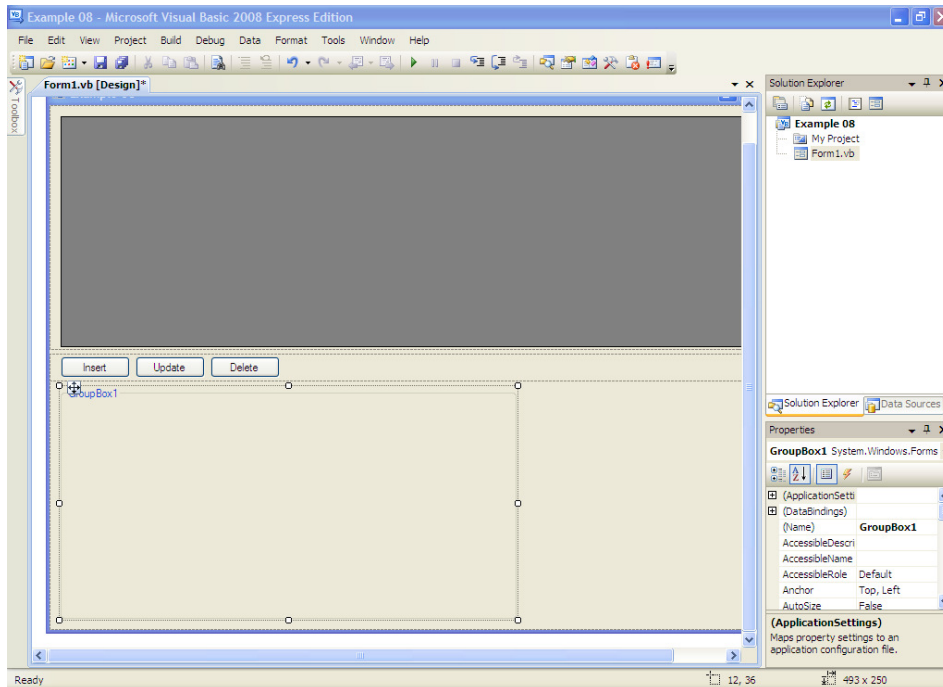
- Rename the buttons' names and Text properties to "Insert", "Update" and "Delete" in this order. Remember to prefix the button names with "btn", e.g. "btnInsert", "btnUpdate" and "btnDelete".



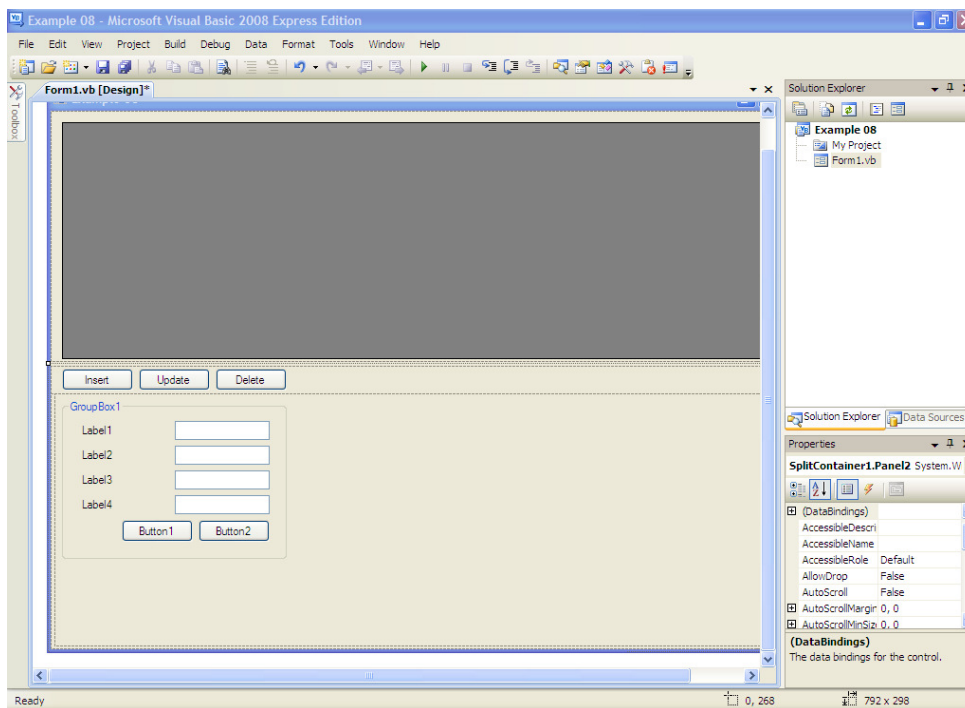
- Resize the panel on which the buttons reside to a little higher than the height of the buttons.
- Again, select Panel2 of the SplitContainer. Add a GroupBox control for the data input controls. To do this, mouse-over the ToolBox, expand the Containers category and double click on the GroupBox item.



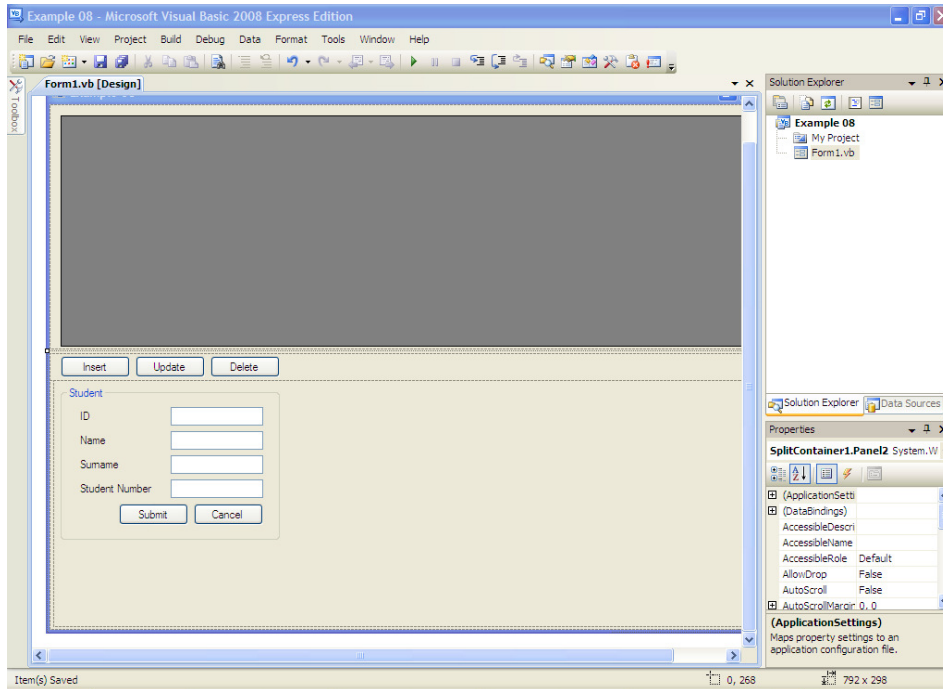
- Move the GroupBox to an appropriate location and resize it accordingly.



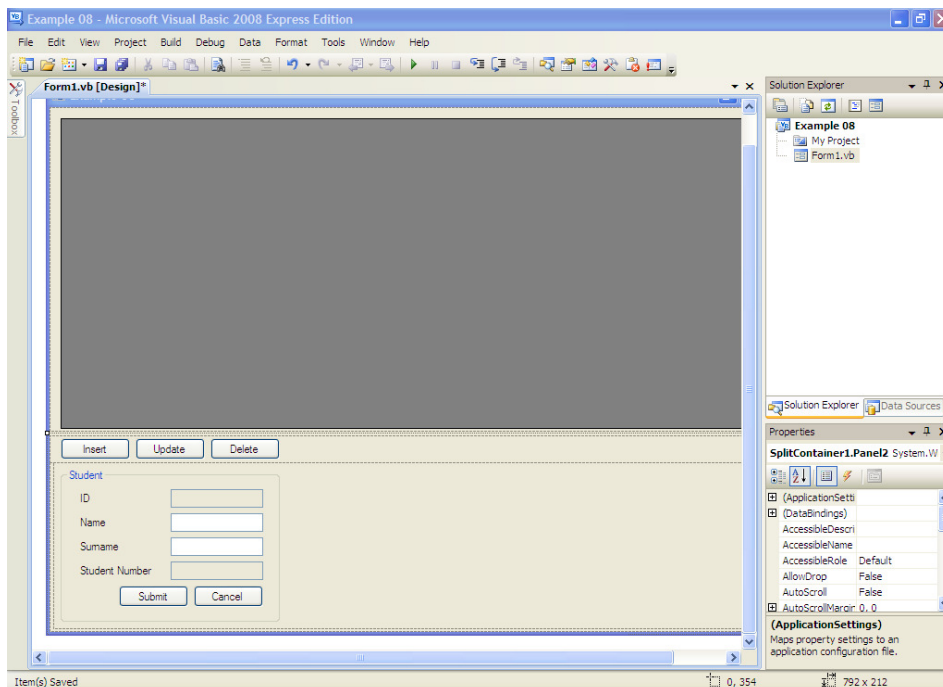
- Set the Text property of the GroupBox to Student.
- Add four Labels and four TextBoxes as well as two Buttons to the GroupBox. Arrange the controls in the GroupBox according to the screenshot below.



- Rename the controls in the GroupBox according to the screenshot below. Remember to prefix the control names with the appropriate prefixes, e.g. Buttons = "btn" and TextBoxes = "txt" etc.



- Resize the bottom section of the SplitContainer accordingly.
- Set the ReadOnly properties of the ID and Student Number TextBoxes to True. The reason for this, because we don't want the user to perform the data input for these two fields. The ID field is auto generated by the database and the Student Number field will be controlled from the code.



- This concludes the visual design of the form and we will also use this format for standardized forms in the examples to follow. Please acquaint yourself with building the form using this method.
- Next, we need to add the code for this application. To switch to the code file, right-click on the design area of the form and click on View Code.
- Declare a string variable called Mode. This variable will be used to keep track of the transaction mode of the form, e.g. INSERT or UPDATE.

```
' Declare the String variable to keep track of the state of the form. E.g. INSERT /
UPDATE
Dim Mode As String
```

- Add a method called GetStudents() to the form's code file. This method will contain the code to fetch the student records from the database and display it in the DataGridView control.

```
Private Sub GetStudents()

End Sub
```

- Add the following lines of code to the method. This code will make all the input control's values blank.

```
' Clear the input controls.
txtID.Text = ""
txtName.Text = ""
txtSurname.Text = ""
txtStudentNumber.Text = ""
```

- Add a Try-Catch block to the method.

```
Try

Catch ex As Exception

End Try
```

- Declare the connection, command and table objects.

```
' Declare the connection, command and table objects.
Dim connection As New OleDb.OleDbConnection
Dim command As New OleDb.OleDbCommand
Dim table As New Data.DataTable
```

- Set theConnectionString of the connection and open the database connection.

```
' Set the ConnectionString property and open the database connection.
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Incomplete\Example 08\CollegeDB.mdb;User Id=;Password="
connection.Open()
```

- Set the Connection and CommandText properties of the command.

```
' Set the Connection and CommandText properties of the command.
command.Connection = connection
command.CommandText = "SELECT * StudentNumber FROM Student ORDER BY StudentNumber"
```

- Execute the command and store the result set into the table.

```
' Execute the command and load the result set into the table.
table.Load(command.ExecuteReader())
```

- Close the database connection.

```
' Close the database connection.
connection.Close()
```

- Set the table as the DataSource for the DataGridView.

```
' Set the table as the DataSource for the DataGridView.
grvStudents.DataSource = table
```

- Add a prompt to the Catch clause to notify the user of any exceptions.

```
' Prompt the user with the exception.
MessageBox.Show(ex.ToString())
```

- Let's summarize this method's code.

```
Private Sub GetStudents()
    ' Clear the input controls.
    txtID.Text = ""
    txtName.Text = ""
    txtSurname.Text = ""
    txtStudentNumber.Text = ""

    Try
        ' Declare the connection, command and table objects.
        Dim connection As New OleDb.OleDbConnection
        Dim command As New OleDb.OleDbCommand
        Dim table As New Data.DataTable

        ' Set the ConnectionString property and open the database connection.
        connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
        Source=C:\Examples\Incomplete\Example 08\CollegeDB.mdb;User
        Id=;Password=;"
        connection.Open()

        ' Set the Connection and CommandText properties of the command.
        command.Connection = connection
        command.CommandText = "SELECT * FROM Student ORDER BY StudentNumber"

        ' Execute the command and load the result set into the table.
        table.Load(command.ExecuteReader())

        ' Close the database connection.
        connection.Close()

        ' Set the table as the DataSource for the DataGridView.
        grvStudents.DataSource = table
    Catch ex As Exception
        ' Prompt the user with the exception.
        MessageBox.Show(ex.ToString())
    End Try
End Sub
```

- This method will return all of the student records as contained in the database. We need to add another method that will return only a single student record based on the ID field of the student record. Add the following method.

```
Private Sub GetStudent(ByVal pID As Integer)

End Sub
```

- Note that this method takes a parameter called pID. This parameter will specify the single record that we require using a parameterized query.

- Add a Try-Catch block to the method.

```
Try
Catch ex As Exception
End Try
```

- Declare the connection, command and table objects.

```
' Declare the connection, command and table objects.
Dim connection As New OleDb.OleDbConnection
Dim command As New OleDb.OleDbCommand
Dim table As New Data.DataTable
```

- Set theConnectionString of the connection and open the database connection.

```
' Set the ConnectionString property and open the database connection.
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Incomplete\Example 08\CollegeDB.mdb;User Id=;Password=;"
connection.Open()
```

- Set the Connection and CommandText properties of the command.

```
' Set the Connection and CommandText properties of the command.
command.Connection = connection
command.CommandText = "SELECT * FROM Student WHERE ID = @ID"
```

- Note that we are working with a parameterized query.
- Declare a parameter object and initialize its name and value.

```
' Declare and initialize the parameter required by the command.
Dim parID = New OleDb.OleDbParameter("@ID", pID)
```

- Add the parameter to the command.

```
' Add the parameter to the command.
command.Parameters.Add(parID)
```

- Execute the command and store the result in the table.

```
' Execute the command and load the result set into the table.
table.Load(command.ExecuteReader())
```

- Close the database connection.

```
' Close the database connection.
connection.Close()
```

- Using an If statement, check whether the table has any rows and if the table has rows, load the values into the data input controls on the GroupBox.

```
If table.Rows.Count > 0 Then ' Check whether there was any results in the result
set and load the result set's data into the input controls.
    txtID.Text = table.Rows(0)("ID").ToString()
    txtName.Text = table.Rows(0)("Name").ToString()
    txtSurname.Text = table.Rows(0)("Surname").ToString()
    txtStudentNumber.Text = table.Rows(0)("StudentNumber").ToString()
End If
```

- Add a prompt to the Catch clause to notify the user of any exceptions.

```
' Prompt the user with the exception.
MessageBox.Show(ex.ToString())
```

- Let's summarize the code for this method.

```
Private Sub GetStudent(ByVal pID As Integer)
    Try
        ' Declare the connection, command and table objects.
        Dim connection As New OleDb.OleDbConnection
        Dim command As New OleDb.OleDbCommand
        Dim table As New Data.DataTable

        ' Set the ConnectionString property and open the database connection.
        connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
        Source=C:\Examples\Incomplete\Example 08\CollegeDB.mdb;User
        Id=;Password=;"
        connection.Open()

        ' Set the Connection and CommandText properties of the command.
        command.Connection = connection
        command.CommandText = "SELECT * FROM Student WHERE ID = @ID"

        ' Declare and initialize the parameter required by the command.
        Dim parID = New OleDb.OleDbParameter("@ID", pID)

        ' Add the parameter to the command.
        command.Parameters.Add(parID)

        ' Execute the command and load the result set into the table.
        table.Load(command.ExecuteReader())

        ' Close the database connection.
        connection.Close()

        If table.Rows.Count > 0 Then ' Check whether there was any results in
        the result set and load the result set's data into the input controls.
            txtID.Text = table.Rows(0)("ID").ToString()
            txtName.Text = table.Rows(0)("Name").ToString()
            txtSurname.Text =
            table.Rows(0)("Surname").ToString()
            txtStudentNumber.Text =
            table.Rows(0)("StudentNumber").ToString()
        End If
    Catch ex As Exception
        ' Prompt the user with the exception.
        MessageBox.Show(ex.ToString())
    End Try
End Sub
```

- Finally, before we are going to add the events to the form, we need to create a function to fetch the next student number to be used for new student insertions. This function will fetch the last student number used in the database and increment the student number to determine the next student number, in the case of no student numbers returned from the database, the student number will default to "200900001".
- Add the following function to the code file.

```
Private Function GetNextStudentNumber()
End Function
```

- Declare a string variable called StudentNumber to store the next student number to be used for the return statement of the function.

```
' Declare the StudentNumber variable for use in this method.
Dim StudentNumber As Integer
```

- Add a Try-Catch block to the function.

```
Try
Catch ex As Exception
End Try
```

- Declare the connection, command and table objects.

```
' Declare the connection, command and table objects.
Dim connection As New OleDb.OleDbConnection
Dim command As New OleDb.OleDbCommand
Dim table As New Data.DataTable
```

- Set theConnectionString of the connection and open the database connection.

```
' Set the ConnectionString property and open the database connection.
connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Incomplete\Example 08\CollegeDB.mdb;User Id=;Password=;"
connection.Open()
```

- Set the Connection and CommandText properties of the command.

```
' Set the Connection and CommandText properties of the command.
command.Connection = connection
command.CommandText = "SELECT TOP 1 StudentNumber FROM Student ORDER BY StudentNumber
DESC"
```

- Execute the command and store the result set into the table.

```
' Execute the command and load the result set into the table.
table.Load(command.ExecuteReader())
```

- Close the database connection.

```
' Close the database connection.
connection.Close()
```

- Using an If statement, check whether the table has a result set. If a result set was returned, take the last student number returned, increment it and store it into the local StudentNumber variable, else initialize the local StudentNumber variable to "200900001".

```
If table.Rows.Count > 0 Then ' Check whether there are any records in the result set.
    If Integer.TryParse(table(0)("StudentNumber").ToString(), StudentNumber) Then '
        Check the validity of the StudentNumber in the result set, and load it into the
        local variable.
        ' Increment the StudentNumber variable.
        StudentNumber = StudentNumber + 1
    End If
Else ' Else use the default starting StudentNumber.
    StudentNumber = 200900001
End If
```

- Add a prompt to the Catch clause to notify the user of any exceptions.

```
' Prompt the user with the exception.
MessageBox.Show(ex.ToString())
```

- Add the following return statement after the Try-Catch block.

```
' Return the new StudentNumber.
Return StudentNumber
```

- Let's summarize the code for this function.



```

Private Function GetNextStudentNumber()
    ' Declare the StudentNumber variable for use in this method.
    Dim StudentNumber As Integer

    Try
        ' Declare the connection, command and table objects.
        Dim connection As New OleDb.OleDbConnection
        Dim command As New OleDb.OleDbCommand
        Dim table As New Data.DataTable

        ' Set the ConnectionString property and open the database connection.
        connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Incomplete\Example 08\CollegeDB.mdb;User
Id=;Password=;"
        connection.Open()

        ' Set the Connection and CommandText properties of the command.
        command.Connection = connection
        command.CommandText = "SELECT TOP 1 StudentNumber FROM Student ORDER BY
StudentNumber DESC"

        ' Execute the command and load the result set into the table.
        table.Load(command.ExecuteReader())

        ' Close the database connection.
        connection.Close()

        If table.Rows.Count > 0 Then ' Check whether there are any records in
the result set.
            If Integer.TryParse(table(0)("StudentNumber").
ToString(), StudentNumber) Then ' Check the validity of
the StudentNumber in the result set, and load it into
the local variable.
                ' Increment the StudentNumber variable.
                StudentNumber = StudentNumber + 1
            End If
        Else ' Else use the default starting StudentNumber.
            StudentNumber = 200900001
        End If
    Catch ex As Exception
        ' Prompt the user with the exception.
        MessageBox.Show(ex.ToString())
    End Try

    ' Return the new StudentNumber.
    Return StudentNumber
End Function

```

- Next, we need to add an event handler for Load() event of the form. Switch back to the Design View and add an event handler for the Load() event of the form through the Event panel.
- Add the following lines of code to the event handler.

```

Private Sub Form1_Load(...)
    ' Update the DataGridView.
    GetStudents()

    ' Clear the Mode.
    Mode = ""
End Sub

```

- This code will call the GetStudents() method to load the student data into the DataGridView and secondly, it will initialize the form mode to nothing.
- Add an event handler for the SelectionChanged event on the DataGridView. Add the following lines of code to the event handler.

```

Private Sub grvStudents_SelectionChanged(...)
    ' Declare an Integer type variable used in this code block by the
Integer.TryParse() method.
    Dim ID As Integer

```

```

        If Integer.TryParse(grvStudents.CurrentRow.Cells("ID").
            Value.ToString(), ID) Then ' Use the Integer.TryParse() method to check
            the validity of the ID field.
            ' Fetch the Student's data and load it in the input controls.
            GetStudent(ID)
        End If
    End Sub

```

- This event handler's code will fetch the ID of the selected student record and call the GetStudent() method to fetch the individual record's data for use in the input controls.
- Switch back to the Design View of the form. Double-click on the Insert button to add an event handler for it.
- In the event handler, add the following code.

```

Private Sub btnInsert_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles btnInsert.Click
    ' Disable the DataGridView for the insert event.
    grvStudents.Enabled = False

    ' Disable the Insert / Update / Delete buttons whilst in insert mode.
    btnInsert.Enabled = False
    btnUpdate.Enabled = False
    btnDelete.Enabled = False

    ' Enable the input fields for the insert event.
    GroupBox1.Enabled = True

    ' Set the Mode of the form to INSERT for future reference.
    Mode = "INSERT"

    ' Reset the input controls for insert mode.
    txtID.Text = ""
    txtName.Text = ""
    txtSurname.Text = ""

    ' Get the next StudentNumber value and store it in the txtStudentNumber
    control.
    txtStudentNumber.Text = GetNextStudentNumber().ToString()
End Sub

```

- This code will firstly disable the DataGridView, Insert, Update and Delete buttons whilst in INSERT mode, secondly it will enable the data input controls for user interaction. Then it will set the mode of the form to INSERT and initialize the input controls. Note that the student number TextBox is initialized using the GetNextStudentNumber() function we created earlier on.
- Switch back to the Design View of the form and double-click on the Update button to add an event handler for it.
- Add the following code to the event handler.

```

Private Sub btnUpdate_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles btnUpdate.Click
    If grvStudents.Rows.Count > 0 Then ' Check whether there are any rows available
        for updating.
        ' Disable the DataGridView control.
        grvStudents.Enabled = False

        ' Disable the Insert / Update / Delete buttons whilst in update mode.
        btnInsert.Enabled = False
        btnUpdate.Enabled = False
        btnDelete.Enabled = False

        ' Enable the input controls for the update event.
        GroupBox1.Enabled = True

        ' Set the Mode of the form to UPDATE for future reference.
    End If
End Sub

```

```

        Mode = "UPDATE"
    Else ' Else prompt the user that there no records available for updating.
        MessageBox.Show("There are no rows to update...")
    End If
End Sub

```

- The code for this event handler will check whether there are any records available for updating using an If statement. If there are records available, the DataGridView, Insert, Update and Delete buttons will be disabled and the data input controls will be enabled for user interaction. The form mode will also be set the UPDATE. Else if there are no records available for updating, the user will receive a MessageBox.Show() prompt notifying him or her.
- Switch back to the Design View of the form and double-click on the Delete button to add an event handler for it. Add the following code to the event handler. Note that this code is similar to the code discussed in the examples prior to this one. Please reference the appropriate example for more information.

```

Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnDelete.Click
    If grvStudents.Rows.Count > 0 Then ' Check whether there are any rows available
        for deletion.
            If MessageBox.Show("Are you sure you want to delete this student?",
"Confirm delete", MessageBoxButtons.YesNo) =
Windows.Forms.DialogResult.Yes Then ' Prompt the user to confirm
deletion.
                ' On a Yes prompt, delete the record.
                Try
                    ' Declare the connection and command objects.
                    Dim connection As New OleDb.OleDbConnection
                    Dim command As New OleDb.OleDbCommand

                    ' Set the ConnectionString property and open the
                    database connection.
                    connection.ConnectionString =
"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Incomplete\Example
08\CollegeDB.mdb;User Id=;Password=;"
                    connection.Open()

                    ' Set the Connection and CommandText properties of the
                    command.
                    command.Connection = connection
                    command.CommandText = "DELETE FROM Student WHERE ID =
@ID"

                    ' Declare and initialize the parameter required for the
                    command.
                    Dim parID = New OleDb.OleDbParameter("@ID",
grvStudents.CurrentRow.Cells("ID").Value)

                    ' Add the parameter to the command.
                    command.Parameters.Add(parID)

                    ' Execute the command.
                    command.ExecuteNonQuery()

                    ' Close the connection.
                    connection.Close()

                    ' Update the DataGridView control.
                    GetStudents()
                Catch ex As Exception
                    ' Prompt the user with the exception.
                    MessageBox.Show(ex.ToString())
                End Try
            End If
        Else ' Else prompt the user that there are no records available for deletion.
            MessageBox.Show("There are no records to delete...")
        End If
    End Sub

```

- The code for the Delete button's click event handler will prompt the user to confirm deletion of the selected student record, and on a Yes confirmation, the student will be deleted using a DELETE transaction to the database. If there are no students available for deletion, the user will be notified by a MessageBox.Show() prompt.
- Switch back to the Design View of the form and double-click on the Submit button to add an event handler for it.
- In the Submit button's event handler, we will be doing the data input validation to determine whether the user has entered a name and surname for the student. Secondly, we will reset the controls so that the user may select records from the DataGridView but not change the record's data until he or she has clicked the Insert, Update or Delete buttons. This event handler will cater for both the INSERT and UPDATE transactions using the local Mode variable. The check is done using a Select Case statement and the database transaction follows accordingly. Finally, the local Mode variable is reset to blank. Add the following code to the Submit button's event handler.

```
Private Sub btnSubmit_Click(...)
    ' Validate the data input.
    If String.IsNullOrEmpty(txtName.Text) Then
        MessageBox.Show("Please enter a name")
        Return
    End If

    If String.IsNullOrEmpty(txtSurname.Text) Then
        MessageBox.Show("Please enter a surname")
        Return
    End If

    ' Disable the input controls.
    GroupBox1.Enabled = False

    ' Enable the DataGridView.
    grvStudents.Enabled = True

    ' Enable the Insert / Update / Delete buttons.
    btnInsert.Enabled = True
    btnUpdate.Enabled = True
    btnDelete.Enabled = True

    ' Test for the Mode of the form and perform the necessary actions accordingly.
    Select Case Mode
        Case "INSERT" ' If the mode is INSERT, perform a new record insertion.
            Try
                ' Declare the connection and command objects.
                Dim connection As New OleDb.OleDbConnection
                Dim command As New OleDb.OleDbCommand

                ' Set theConnectionString property and open the
                database connection.
                connection.ConnectionString =
                    "Provider=Microsoft.Jet.OLEDB.4.0;Data
                    Source=C:\Examples\Incomplete\Example
                    08\CollegeDB.mdb;User Id=;Password="
                connection.Open()

                ' Set the Connection and CommandText properties of the
                command.
                command.Connection = connection
                command.CommandText = "INSERT INTO Student
                (Name, Surname, StudentNumber) VALUES (@Name,
                @Surname, @StudentNumber)"

                ' Declare and initialize the parameters required by the
                command object.
                Dim parName = New OleDb.OleDbParameter("@Name",
                txtName.Text)
                Dim parSurname = New OleDb.OleDbParameter("@Surname",
                txtSurname.Text)
                Dim parStudentNumber = New
                OleDb.OleDbParameter("@StudentNumber",
```

```

txtStudentNumber.Text)

' Add the parameters to the command object.
command.Parameters.Add(parName)
command.Parameters.Add(parSurname)
command.Parameters.Add(parStudentNumber)

' Execute the command.
command.ExecuteNonQuery()

' Close the database connection.
connection.Close()

' Update the DataGridView.
GetStudents()
Catch ex As Exception
    ' Prompt the user with the exception.
    MessageBox.Show(ex.ToString())
End Try
Case "UPDATE" ' If the mode is UPDATE, perform an existing record
updating.
Try
    ' Declare the connection and command objects.
    Dim connection As New OleDb.OleDbConnection
    Dim command As New OleDb.OleDbCommand

    ' Set the ConnectionString property and open the
    database connection.
    connection.ConnectionString =
    "Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=C:\Examples\Incomplete\Example
    08\CollegeDB.mdb;User Id=;Password=;"
    connection.Open()

    ' Set the Connection and CommandText properties of the
    command.
    command.Connection = connection
    command.CommandText = "UPDATE Student SET Name = @Name,
    Surname = @Surname WHERE ID = @ID"

    ' Declare and initialize the parameters required by the
    command object.
    Dim parName = New OleDb.OleDbParameter("@Name",
    txtName.Text)
    Dim parSurname = New OleDb.OleDbParameter("@Surname",
    txtSurname.Text)
    Dim parID = New OleDb.OleDbParameter("@ID", txtID.Text)

    ' Add the parameters to the command object.
    command.Parameters.Add(parName)
    command.Parameters.Add(parSurname)
    command.Parameters.Add(parID)

    ' Execute the command.
    command.ExecuteNonQuery()

    ' Close the database connection.
    connection.Close()

    ' Update the DataGridView.
    GetStudents()
Catch ex As Exception
    ' Prompt the user with the exception.
    MessageBox.Show(ex.ToString())
End Try
Case Else ' Else prompt the user that there is nothing to
submit. This is only a failover mechanism and will rarely be used.
    MessageBox.Show("Nothing to Submit...")
End Select

' Clear the Mode.
Mode = ""
End Sub

```

Finally, we need to add an event handler for the Cancel button on the data input section of the form. Switch to Design View and double-click on the Cancel button to add an event handler for it. Add the following code to the event handler.

```
Private Sub btnCancel_Click(...)
    ' Enable the DataGridView control.
    grvStudents.Enabled = True

    ' Disable the input controls.
    GroupBox1.Enabled = False

    ' Enable the Insert / Update / Delete buttons.
    btnInsert.Enabled = True
    btnUpdate.Enabled = True
    btnDelete.Enabled = True

    ' Clear the Mode of the form.
    Mode = ""

    If grvStudents.Rows.Count > 0 Then ' Check whether there are records in the
        DataGridView.
            ' Declare an Integer type variable used in this code block by the
            Integer.TryParse() method.
            Dim ID As Integer

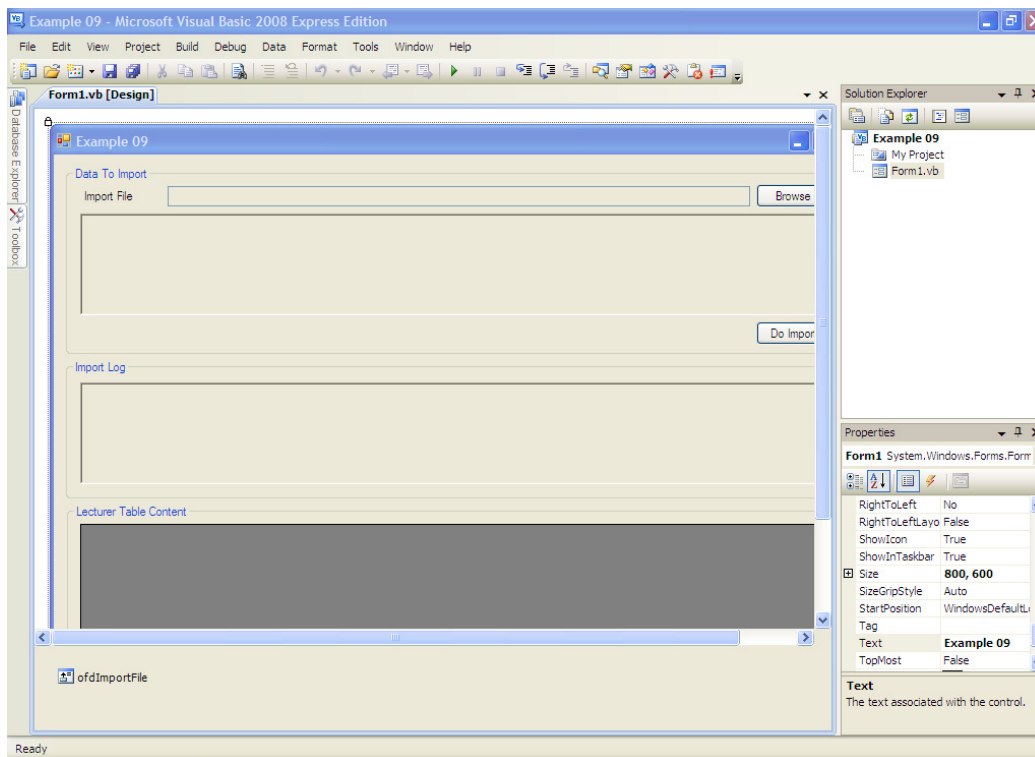
            If Integer.TryParse(grvStudents.CurrentRow.Cells("ID").
                Value.ToString(), ID) Then ' Use the Integer.TryParse() method
                to check the validity of the ID field.
                    ' Fetch the Student's data and load it in the input controls.
                    GetStudent(ID)
            End If
        Else ' Else clear the input controls.
            txtID.Text = ""
            txtName.Text = ""
            txtSurname.Text = ""
            txtStudentNumber.Text = ""
        End If
    End Sub
```

- The code for the Cancel button will reset the DataGridView to enabled and disable the input controls. Secondly, it will also fetch the selected record's data from the database and reset the input controls with those values in case the user changed it before clicking the Cancel button. If there were no record selected, the input controls will be initialized to blank.
- This concludes the source code for the form. We are now ready to run the application to test it.
- Press F5 or click Debug > Start Debugging to run the application.
- Test the form's functionality by selecting different student records from the DataGridView and checking whether the selected record's data is displayed in the input controls.
- Test the Insert, Update and Delete button's functionality by adding a new student, updating the student you've added and finally deleting the student again. Key functionality to check is whether a new student number is generated when inserting a new student, check whether the DataGridView is updated when a student is inserted, updated or deleted, and finally check whether the data display and input controls are enabled and disabled correctly for each function.

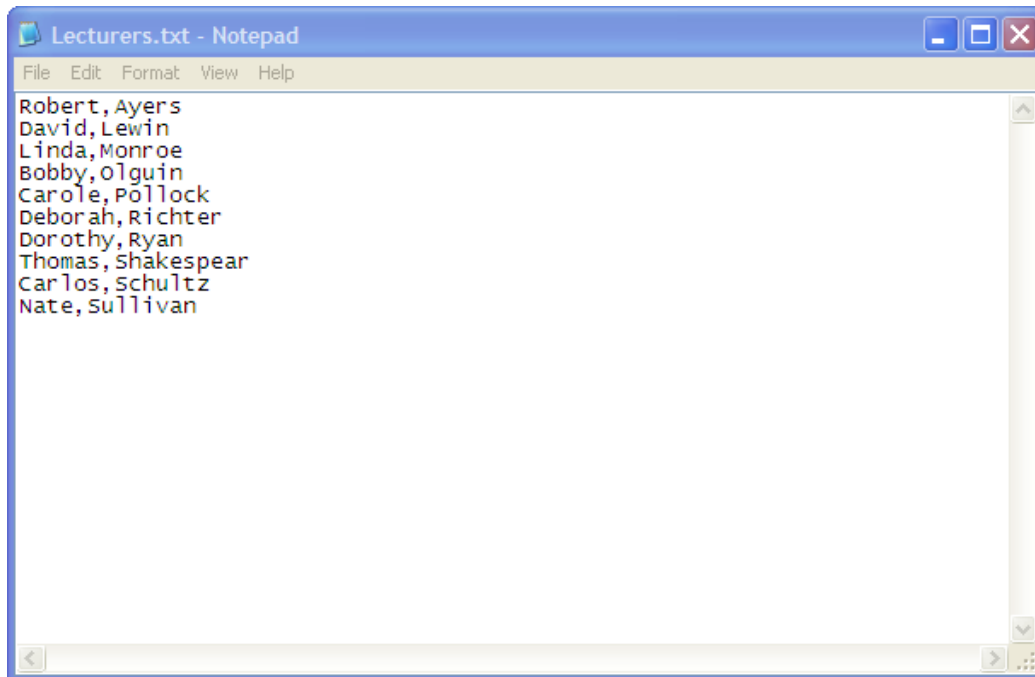
## Example 9: Importing data from a text file into a database table.

During this example, we will look at using alternative input for the content stored in a database table. In this case we are going to use a comma-delimited text file with the names and surnames of lecturers, extracting the names and surnames programmatically and storing them into the Lecturer table in the CollegeDB. Please note that this example does not focus on the reading of the text file but rather on the concept of an alternative input source.

- Copy the source code for Example 09 to your hard disk drive or flash drive and open the solution with Microsoft Visual Basic 2008 Express Edition.
- Open the Form1.vb item in the Solution Explorer. You should now have the following screen in front of you.



- The visual part of the screen is already built along with some of the functionality such as to fetch the existing Lecturer records, and the opening and reading of the text file. You may view this code by switching to the code file of the form.
- In the source file of the form, we will add a few methods to handle the data fetched from the text file but before we get to the code, let's look at the text file and how we are going to handle the data contained in the text file. Open the text file through Windows Explorer. You should now have the following screen in front of you.



- Each line in the text file contains one lecturer record, e.g. name and surname of the lecturer separated with comma. With this example, we will loop through the text file, line by line and split the fields for insertion into the database table.
- Switch to the source file of the form.
- For this example we need to add a method and a function for the handling of the data. The function will take a line and validate whether the lecturer record retrieved from the text file is valid. The method will take the validated lecturer record from the text file and insert the record into the database table.
- Add the following function for the validation of the lecturer record.

```
Private Function ValidateLecturer(ByVal Lecturer As String)
End Function
```

- Note that the function receives a string argument. This argument contains a single lecturer record from the text file.
- In order to validate the record, we need to split the data in the record. The record contains two fields, namely "Name" and "Surname", separated by a comma. The String class has a Split() method that takes a delimiter and split the string according to the delimiter. In this case, the delimiter is the comma and the result of the String.Split() method will be a string array containing two items. Add the following code to the function to declare the string array and split the record's content.

```
' Split the input value into string array.
Dim NameSurname() = Lecturer.Split(",")
```

- To keep track of the validity of the input argument, we need to declare a local Boolean variable.

```
' Declare a boolean variable to keep track of the validity.
Dim Valid = True
```



- The first check we'll be doing is to check whether there are two items in the string array. In other words, we need to check whether the record from the text file is complete. If the record is complete, the second set of check may be done in the following step, otherwise the error is recorded into the RichTextBox control on the form.
- Secondly, we will check each element of the string array to check whether the values are valid. If any of the values are not valid, the local Boolean variable is set to False indicating that the current record is not validated for insertion into the Lecturer table. Additionally, the error is also recorded into the RichTextBox control on the form with detail of the record.
- The complete set of checks will look like the following. Add this code to the function accordingly.

```

If NameSurname.Length = 2 Then ' Check whether the string array has two elements.
    If String.IsNullOrEmpty(NameSurname(0)) Then ' Check whether the first element
        is empty.
            ' Update the validity to invalid.
            Valid = False
            ' Log the invalid line.
            rtbImportLog.Text = rtbImportLog.Text + "Name is not valid - " +
                Lecturer + Environment.NewLine
        End If
    If String.IsNullOrEmpty(NameSurname(1)) Then ' Check whether the second element
        is empty.
            ' Update the validity to invalid.
            Valid = False
            ' Log the invalid line.
            rtbImportLog.Text = rtbImportLog.Text + "Surname is not valid - " +
                Lecturer + Environment.NewLine
        End If
    Else ' Else log the invalid line.
        ' Update the validity to invalid.
        Valid = False
        ' Log the invalid line.
        rtbImportLog.Text = rtbImportLog.Text + "Import source is not in the correct
            format - " + Lecturer + Environment.NewLine
    End If
End If

' Return the validity.
Return Valid

```

- Lastly, we need to return the Boolean state indicating the validity of the record.

- Let's summarize the code for this function.

```

Private Function ValidateLecturer(ByVal Lecturer As String)
    ' Split the input value into string array.
    Dim NameSurname() = Lecturer.Split(",")

    ' Declare a boolean variable to keep track of the validity.
    Dim Valid = True

    If NameSurname.Length = 2 Then ' Check whether the string array has two
        elements.
            If String.IsNullOrEmpty(NameSurname(0)) Then ' Check whether the first
                element is empty.
                ' Update the validity to invalid.
                Valid = False
                ' Log the invalid line.
                rtbImportLog.Text = rtbImportLog.Text + "Name is not valid - " +
                    Lecturer + Environment.NewLine
            End If
            If String.IsNullOrEmpty(NameSurname(1)) Then ' Check whether the second
                element is empty.
                ' Update the validity to invalid.
                Valid = False
                ' Log the invalid line.
                rtbImportLog.Text = rtbImportLog.Text + "Surname is not valid -
                    " + Lecturer + Environment.NewLine
            End If
        End If
    End If
End Function

```

```

Else ' Else log the invalid line.
    ' Update the validity to invalid.
    Valid = False
    ' Log the invalid line.
    rtbImportLog.Text = rtbImportLog.Text + "Import source is not in the
correct format - " + Lecturer + Environment.NewLine
End If

' Return the validity.
Return Valid
End Function

```

- Secondly, we need to add the insertion method. This method will take a validated lecturer record and perform an insert transaction into the Lecturer table. Add the following method for this purpose.

```

Private Sub ImportLecturer(ByVal Lecturer As String)
    Try
        ' Split the import data into an array.
        Dim NameSurname() = Lecturer.Split(",")

        ' Declare the connection and command objects.
        Dim connection As New OleDb.OleDbConnection
        Dim command As New OleDb.OleDbCommand

        ' Set the ConnectionString property and open the connection.
        connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Examples\Incomplete\Example 09\CollegeDB.mdb;User
Id=;Password=;"
        connection.Open()

        ' Set the Connection and CommandText properties.
        command.Connection = connection
        command.CommandText = "INSERT INTO Lecturer (Name, Surname) VALUES
(@Name, @Surname)"

        ' Declare and initialize the parameters required for the command.
        Dim parName = New OleDb.OleDbParameter("@Name", NameSurname(0))
        Dim parSurname = New OleDb.OleDbParameter("@Surname", NameSurname(1))

        ' Add the parameters to the command.
        command.Parameters.Add(parName)
        command.Parameters.Add(parSurname)

        ' Execute the command.
        command.ExecuteNonQuery()

        ' Close the database connection.
        connection.Close()
    Catch ex As Exception
        ' Prompt the user with the exception.
        MessageBox.Show(ex.ToString())
    End Try
End Sub

```

- Next, we are going to add an event handler for the "Do Import" button on the form. Switch back to the Design View of the form.
- Double-click the "Do Import" button to add an event handler for it.
- In the event handler of the "Do Import" button, we will add the code for the validation and insertion of the lecturer records once a text file has been opened.
- The code for the "Do Import" button will basically loop through the records shown in the "Data To Import" GroupBox using a For loop, validating each record individually and then inserting the record into the Lecturer table. After the execution of the For loop, the DataGridView will be updated with the GetLecturers() method. The event handler for the "Do Import" button will look like the following.

```

Private Sub btnDoImport_Click(...)

```

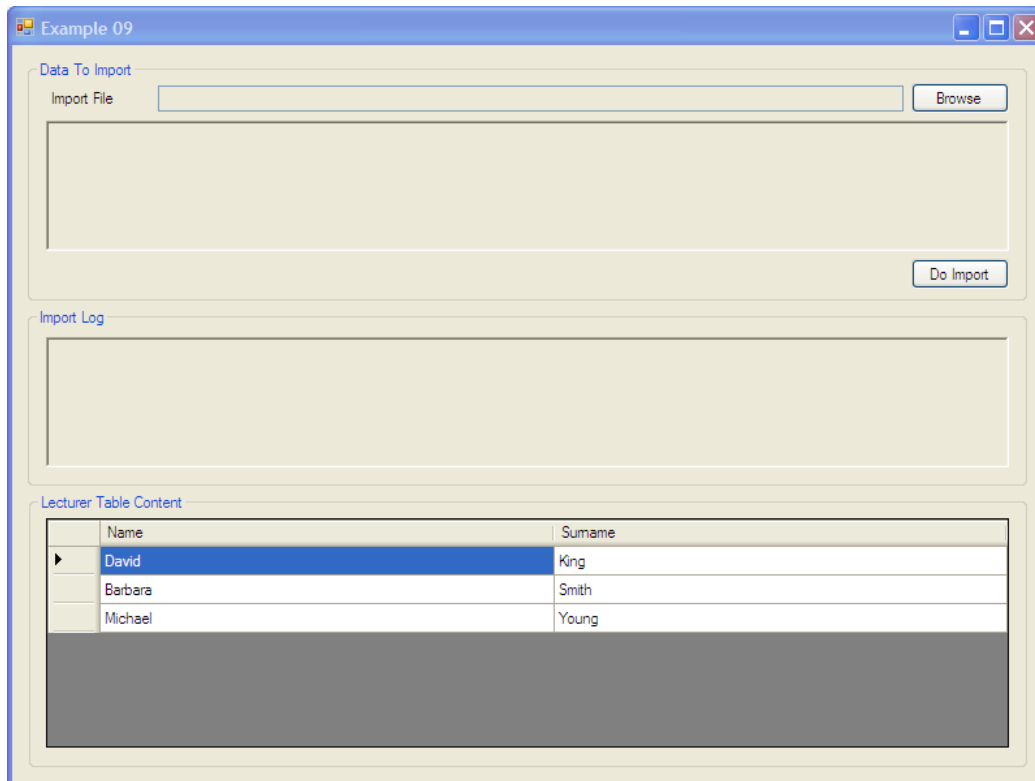
```

For Each line In rtbImportFile.Lines ' For each line, do the following.
    If ValidateLecturer(line) Then ' Validate whether line may be imported.
        ' Import the line into the database.
        ImportLecturer(line)
    End If
Next

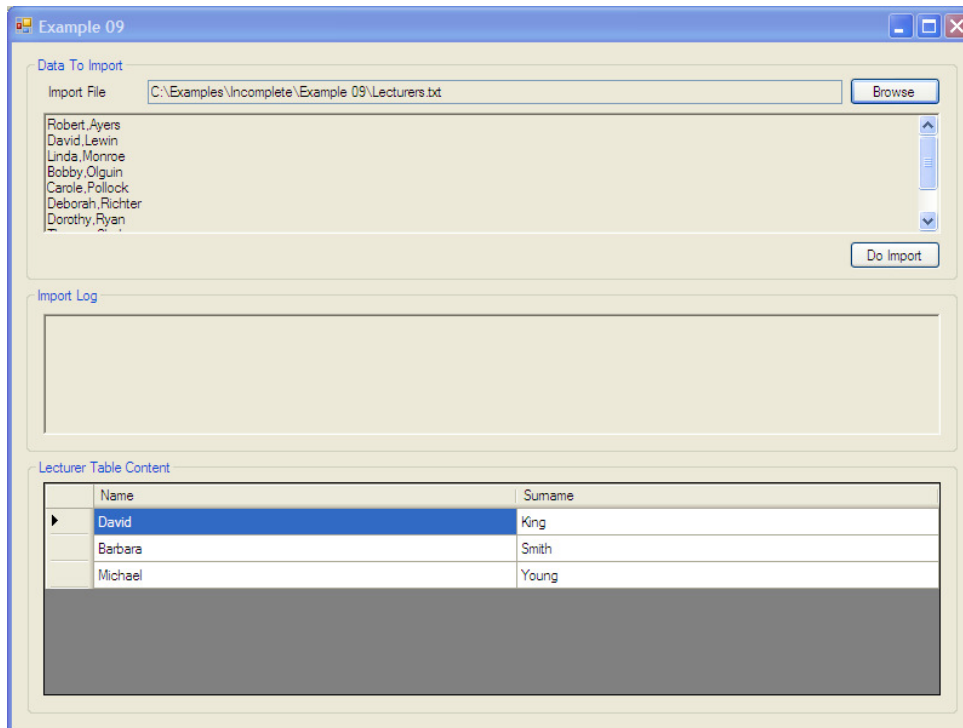
' Update the DataGridView content.
GetLecturers()
End Sub

```

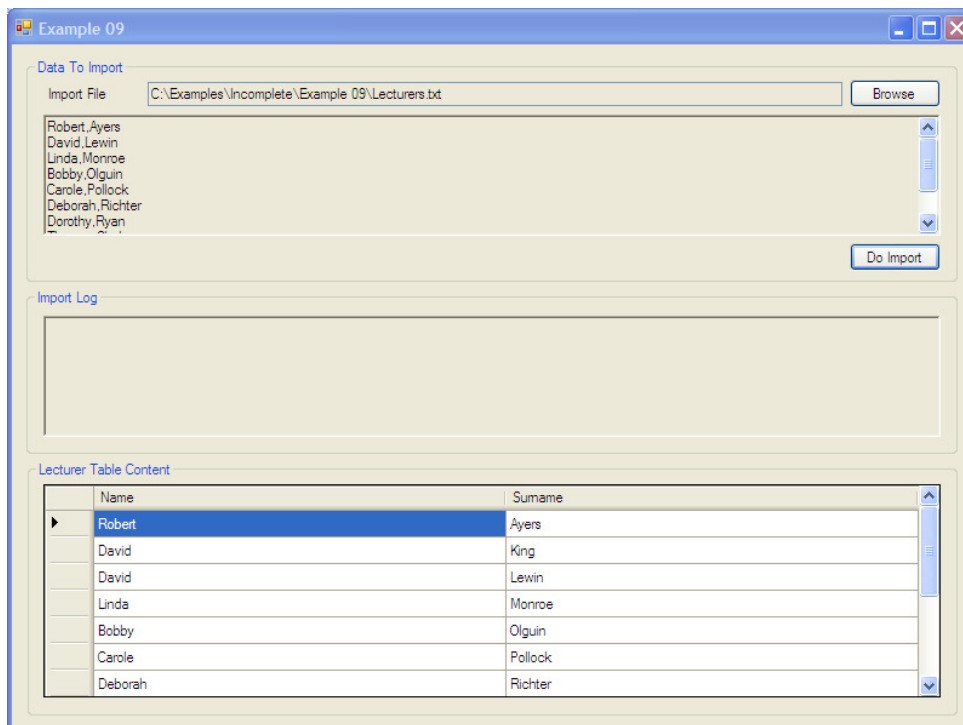
- This example is now complete. Run the application by pressing F5 or clicking on Debug > Start Debugging.



- Test the application by clicking on the "Browse" button. Locate the text file contained in the solution directory and open it.

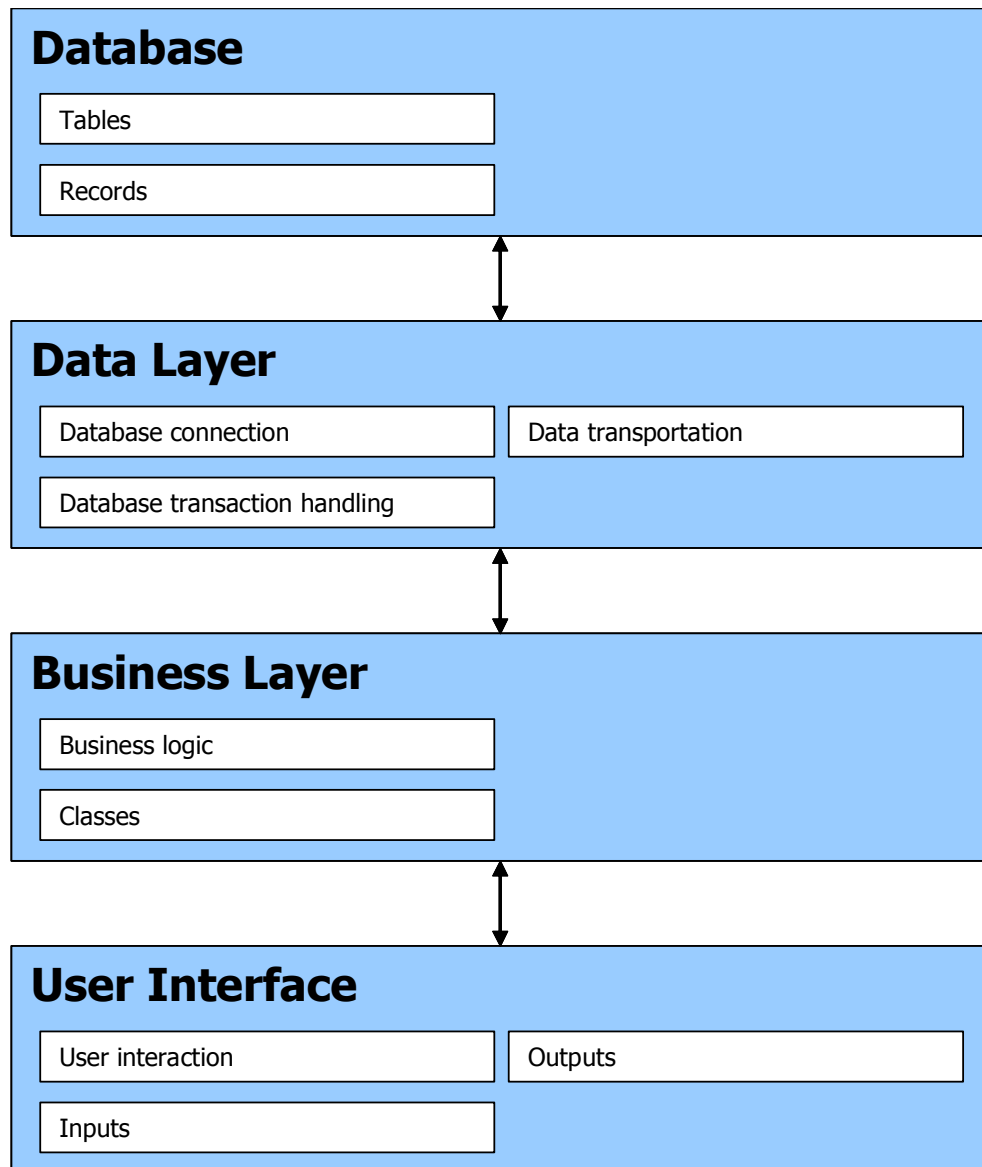


- You should now be able to view the records from the text file in the "Data To Import" GroupBox.
- To test the import, click on the "Do Import" button and check the output in the "Import Log" and "Lecturer Table Content" GroupBoxes. If any errors occurred, they should be listed in the "Import Log" and the imported records should now display in the DataGridView along with the original Lecturer table records.



## Combining data driven applications with other programming concepts.

Data driven applications may easily be combined with other programming techniques and it is very important to explore these possibilities. This section will look at combining topics such as classes, polymorphism, enumerations and multi-tier application design with data driven applications. Below is a graphical overview of what we can accomplish with this combination:



During the last example for this chapter, we will look at how to implement this model within the application. The example explores the concept of using a multi-form application to manage the data in the Crystal Creek College database along with the programming techniques listed in the abovementioned diagram.

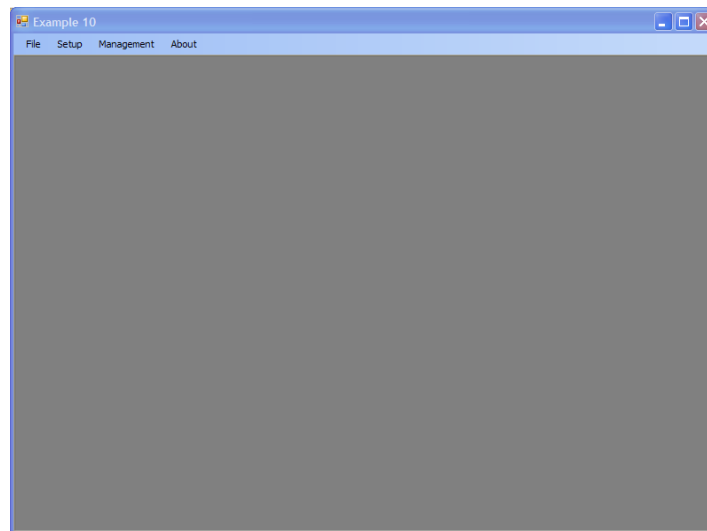
## Example 10: Data driven application using other programming techniques.

This example is already completely programmed; hence we will only look at some of the concepts as discussed in the previous section. As mentioned, this example will implement a multi-form application with database connectivity, database transactions, classing, and multi-tier application design.

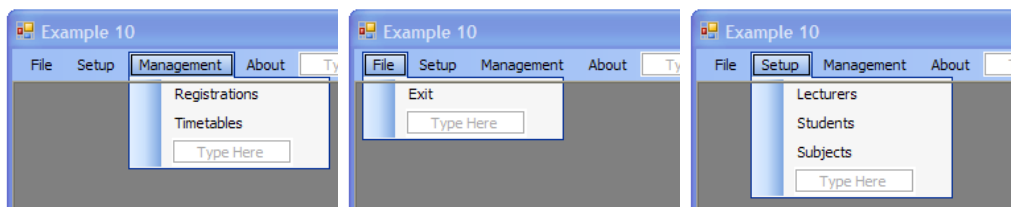
Copy the source code for Example 10 to your hard disk drive or flash drive and open the solution using Microsoft Visual Basic 2008 Express Edition.

### Multi-form application

This example is built on the multi-form application model with a parent container which is the main application with a menu to access other features of the application. Study the screenshots (or the application itself) below to get an idea of the application.



### MDI Application – Parent form.



### MDI Application – Menu Items.

The menu bar consists of four menus with sub-menu items namely "File", "Setup", "Management" and "About". The "File" menu has an "Exit" sub-menu item to exit the application. The "Setup" menu item contains sub-menu items to handle the administration of the students, lecturers and subjects. The "Management" menu item contains sub-menu items to manage the student registrations and timetable. The "About" menu item pops up a window to display information such as version, description and other application detail.

Each screen that opens on the parent application is a separately developed form with functionality unique to the form. The "Setup" menu item's screens are used for the administration of the content in

the database whereby the "Management" menu item's screens are used for the transactional functionality of the system such as student registrations and timetable.

## Classes

On the "Subjects" form, a multi-tier application design has been implemented. Open the code for this form. In the code form, you'll see that there are three classes namely, frmSubjects, Database and Subject. The frmSubjects class is the form's class and it is used for the event handlers on the form. The Database class provides functionality to connect to a database and perform data transactions. The Subject class uses the Database class to retrieve and send subject data and provides a business layer where business rules and programming logic may be implemented. Let's have a closer look at the classes.

### Database class

As mentioned in the previous section, the Database class provides the functionality to connect to a database file, and perform data transactions on the database file. This implementation of a data layer is very basic and the real-life implementation of the data layer may be much greater. Study the class layout and definition below.

```
Public Class Database
    Dim _Connection = New OleDb.OleDbConnection
    Dim _Command = New OleDb.OleDbCommand
    Dim _Table = New Data.DataTable

    Property Connection ...
    Property Command ...
    Property Table ...

    Creates a new instance of the Database class. Initializes the Connection, Command and T
    Public Sub New ...

    Returns a list of Subjects from the database.
    Public Sub GetSubjects ...

    Returns a single Subject from the database according to the ID parameter.
    Public Sub GetSubject ...

    Performs an INSERT transaction based on the Subject instance provided.
    Public Sub InsertSubject ...

    Performs an UPDATE transaction based on the Subject instance provided.
    Public Sub UpdateSubject ...

    Performs a DELETE transaction based on the Subject ID parameter provided.
    Public Sub DeleteSubject ...
End Class
```

The Database class contains three fields and properties to store the Connection, Command and DataTable instances whilst an instance of the class is alive. The class also contains a constructor which initializes the Connection, Command and DataTable members in the class on the instantiation of the Database class instance. See the code snippet below.

```
''' <summary>
''' Creates a new instance of the Database class.
''' Initializes the Connection, Command and Table objects.
''' </summary>
''' <remarks></remarks>
Public Sub New()
    ' Initialize the ConnectionString of the connection object.
    Connection.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=C:\Examples\Complete\Example 10\CollegeDB.mdb;User Id=;Password=;"

    ' Set the Connection property of the command object.
    Command.Connection = Connection

    ' Reset the table of all data.
    Table.Reset()
End Sub
```

The database class also contains methods to perform data transactions on the database. Below is a table with the methods and a description.

Method	Description
<b>GetSubjects</b>	<b>Returns all subjects from the database.</b>
<b>GetSubject</b>	<b>Returns a single subject based on the input parameter provided.</b>
<b>InsertSubject</b>	<b>Performs an INSERT transaction to add a new subject into the database.</b>
<b>UpdateSubject</b>	<b>Performs an UPDATE transaction on an existing subject record in the database.</b>
<b>DeleteSubject</b>	<b>Performs a DELETE transaction on an existing subject record in the database.</b>

The database functionality usually programmed into the event handlers on the form are now contained in a separate class. In this way we can group related functionality so that our source code is not too cluttered and to avoid duplicate source code. For instance, the ConnectionString property is now contained in the constructor of the Database class instead of initializing it in each event handler. You are welcome to study the source code of each method in the Database class. Remember that the functionality in this class is not different from the other examples, it is just the way we implemented the functionality that is different. This model provides a more robust and elegant solution for handling database transactions.



## Subject class

The Subject class is a middle-tier layer for handling business decisions and to act as a transport layer for data between the front-end (user interface) and the back-end (database) layers. The Subject class is very basic because the functionality on the screen does not require much business rules and decisions. However, if there was something that needed to be enforced such as an ISBN check validation (if it did exist in our database), you could implement this business rule in the Subject class. Let's look at the definition of the Subject class.

```
Public Class Subject
    Dim _ID As Integer
    Dim _Name As String
    Dim _Active As Boolean

    Property ID ...
    Property Name ...
    Property Active ...

    Creates a default instance of the Subject class.
    Public Sub New ...

    Creates a populated instance of Subject based on the parameters received.
    Public Sub New ...

    Returns a list of Subjects records.
    Public Function GetSubjects ...

    Returns a single instance of Subject according to the ID parameter.
    Public Function GetSubject ...

    Takes a populated instance of Subject and performs an INSERT transaction.
    Public Sub InsertSubject ...

    Takes a populated instance of Subject and performs an UPDATE transaction.
    Public Sub UpdateSubject ...

    Takes a Subject ID and performs a DELETE transaction on the particular record.
    Public Sub DeleteSubject ...
End Class
```

The Subject class has two constructors, namely a default constructors that will create an empty instance of the Subject class and parameterized constructors that takes "ID", "Name" and "Active" parameters and populates the Subject class instance accordingly. Additionally, the Subject class has methods for handling the data interaction between the other layers. The method names are the same as in the Database class, namely GetSubjects, GetSubject, InsertSubject, UpdateSubject and DeleteSubject. These method passes through their input values into a Database class instance to perform the actual data transactions. Have a look at the source code of the Subject class to see how these fit together. Let's have a look at some code snippets.

```

''' <summary>
''' Returns a single instance of Subject according to the ID parameter.
''' </summary>
''' <param name="pID">Subject ID</param>
''' <returns>A populated instance of Subject</returns>
''' <remarks></remarks>
Public Function GetSubject(ByVal pID As Integer)
    Dim Sbj = New Subject
    Dim DB = New Database
    DB.GetSubject(Sbj, pID)
    Return Sbj
End Function

```

Code snippet of the GetSubject method as found in the Subject class.

```

''' <summary>
''' Returns a single Subject from the database according to the ID parameter.
''' </summary>
''' <param name="pSubject">Populates a single instance of Subject with the record ''
''' returned from the database.</param>
''' <param name="pID">Subject ID</param>
''' <remarks></remarks>
Public Sub GetSubject(ByRef pSubject As Subject, ByVal pID As Integer)
    Try
        ' Open the database connection.
        Connection.Open()

        ' Set the CommandText of the Command.
        Command.CommandText = "SELECT * FROM Subject WHERE ID = @ID ORDER BY Name"

        ' Declare and initialize the parameter required by the command.
        Dim parID = New OleDb.OleDbParameter("@ID", pID)

        ' Add the parameter to the command.
        Command.Parameters.Add(parID)

        ' Execute the command and load the result set into the table.
        Table.Load(Command.ExecuteReader())

        ' Close the database connection.
        Connection.Close()

        If Table.Rows.Count > 0 Then
            pSubject.ID = Table.Rows(0)("ID")
            pSubject.Name = Table.Rows(0)("Name")
            pSubject.Active = Table.Rows(0)("Active")
        End If
    Catch ex As Exception
        ' Prompt the user with the exception.
        MessageBox.Show(ex.ToString())
    End Try
End Sub

```

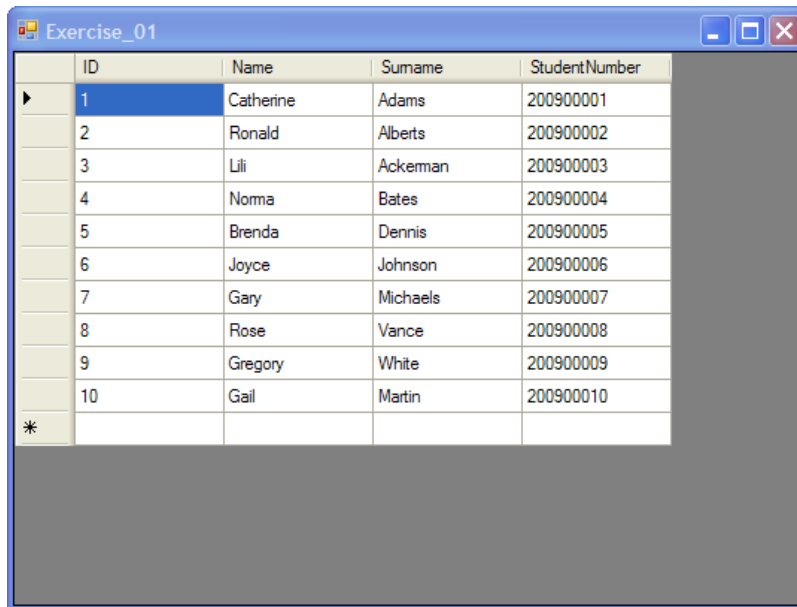
Code snippet of the GetSubject method as found in the Database class.

Note that both methods have the same names but contained in different classes. The GetSubjects method in the Subject class takes its input parameters, creates an instance of the Database class and calls the GetSubject method of the Database class to fetch the data from the database.

In multi-tier applications, this concept is the basic model on which every tier is built. In other words, grouping related functionality and source code is essential as this eliminates duplicate code, redundant functionality, creates a more stable application and is just a neater way of programming.

## Exercises

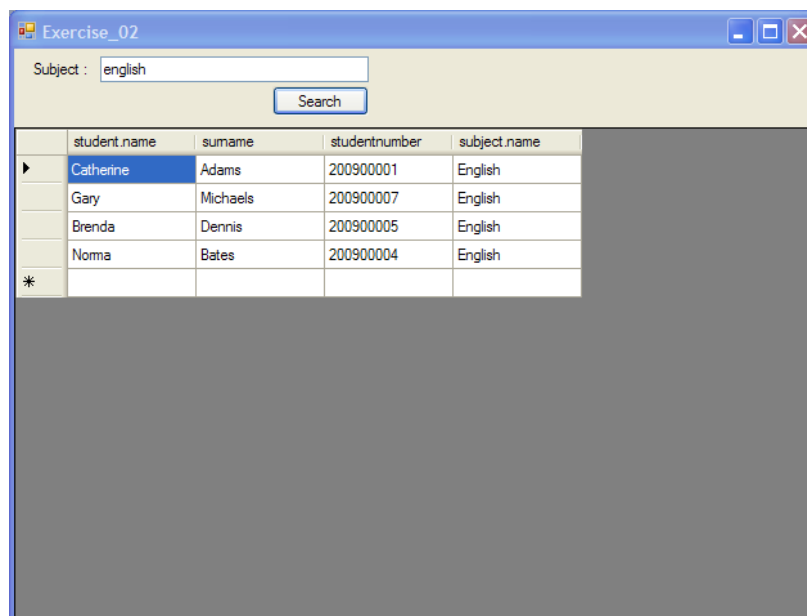
1. The student advisors of Crystal Creek College need a list with all students. They have requested that a screen with all the student details be created. The student details should be displayed in a tabular format. See the screenshot below as a guideline for the layout of the screen. Be sure to use the CollegeDB database.



The screenshot shows a window titled "Exercise\_01" with a table containing student details. The table has four columns: ID, Name, Surname, and StudentNumber. The rows are numbered 1 through 10, with a final row marked with an asterisk (\*). The first row is highlighted in blue.

ID	Name	Surname	StudentNumber
1	Catherine	Adams	200900001
2	Ronald	Alberts	200900002
3	Lili	Ackeman	200900003
4	Norma	Bates	200900004
5	Brenda	Dennis	200900005
6	Joyce	Johnson	200900006
7	Gary	Michaels	200900007
8	Rose	Vance	200900008
9	Gregory	White	200900009
10	Gail	Martin	200900010
*			

2. The registration process at Crystal Creek College is not running too smoothly because the student advisors don't have a searchable list of students according to a specific subject. They have requested that a screen be built that would allow them to see a list of students in a tabular format according to the subject name entered into a search box. See the screenshot below for an indication of what the screen could look like and use the CollegeDB database.



The screenshot shows a window titled "Exercise\_02" with a search interface. At the top, there is a text box labeled "Subject:" containing the word "english", and a "Search" button. Below this is a table with five columns: student.name, surname, studentnumber, and subject.name. The rows are filtered to show only students with the subject "English". The first row is highlighted in blue.

student.name	surname	studentnumber	subject.name
Catherine	Adams	200900001	English
Gary	Michaels	200900007	English
Brenda	Dennis	200900005	English
Norma	Bates	200900004	English
*			

3. KC's Trading needs a new data entry screen to add new products to their stock list along with the available quantity, purchase price and selling price. Create a data entry screen that would

be able to handle the data addition. Make sure to use the POSDB database and refer to the screenshot below for a guideline on what the screen should look like.

The screenshot shows a window titled "Exercise\_03" with a form for adding a new item. The form has four input fields: "Description" with the value "Coca Cola", "Purchase Price" with the value "3.50", "Available Quantity" with the value "20", and "Selling Price" with the value "5". Below the fields is an "Insert" button.

4. Fascination Productions & Events needs to update the coordinators details on a regular basis but at this stage they don't have a proper data management screen for this. Create a data management screen that would allow the administration staff to easily manage the existing coordinators. Make use of the EventsDB and refer to the screenshot below for a guideline on what the screen should look like.

The screenshot shows a window titled "Exercise\_04" with a form for updating coordinator details and a table of existing coordinators. The form has four input fields: "Name" with the value "Eugene", "Surname" with the value "Mares", "Contact" with the value "012-555-0128", and "Company" with the value "Mountain Works". Below the fields is an "Update" button. Below the form is a table with the following data:

	ID	Name	Surname	ContactNumber	Company
▶	1	Eugene	Mares	012-555-0128	Mountain Works
	2	Brian	Cook	012-555-0196	Light Speed
	3	Lionel	McKay	012-555-0191	Marsh
*					

5. Lecturers at Crystal Creek College are complaining that sometimes they add incorrect timetable entries but are not able to remove these entries. Provide them with a screen that would list all of the timetable entries and allow them to delete the selected entry accordingly. Use the CollegeDB database and refer to the screenshot below for guidance on the screen layout.

	id	Lecturer	Subject	dayofweek	timeofday
▶	1	Barbara	Programming	1	1899/12/30 08:0...
	2	Barbara	Programming	3	1899/12/30 09:0...
	3	David	Life Studies	2	1899/12/30 08:0...
	4	David	English	2	1899/12/30 09:0...
	5	Barbara	Mathematics	4	1899/12/30 08:0...
	6	Barbara	Mathematics	5	1899/12/30 08:0...
	7	Michael	Life Studies	4	1899/12/30 09:0...
*					

Delete

6. KC's Trading needs a stock management screen where the stock personnel may add and edit stock items. Create a stock management screen where they can add and edit stock items. Use the POSDB and refer to the screenshot below for a reference to the layout of the screen.

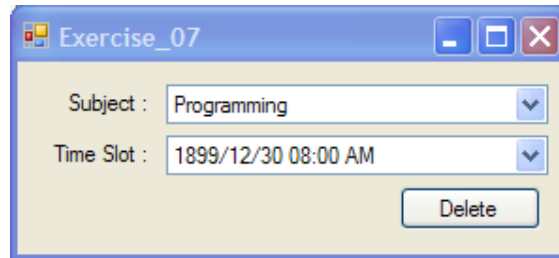
Description : VB.NET 2005 For Dummies      Purchase Price : 25033

Available Quantity : 20      SellingPrice : 300

Update      Insert

	ID	Description	PurchasePrice	AvailableQuantity	SellingPrice
▶	1	VB.NET 2005 Fo...	25033	20	300
	2	Exam Pad	5	500	10
	3	Diary	25	75	75
	4	Database Conce...	150	10	250
	5	Beginning C# Pro...	100	8	120
*					

7. Create an alternative to exercise 5 for deleting timetable entries. Use the screenshot below as a point of reference. The Subject dropdown list should populate the Time Slot dropdown list and when the Delete button is clicked, the timetable entry listed in the Time Slot dropdown list should be deleted. Use the CollegeDB database.



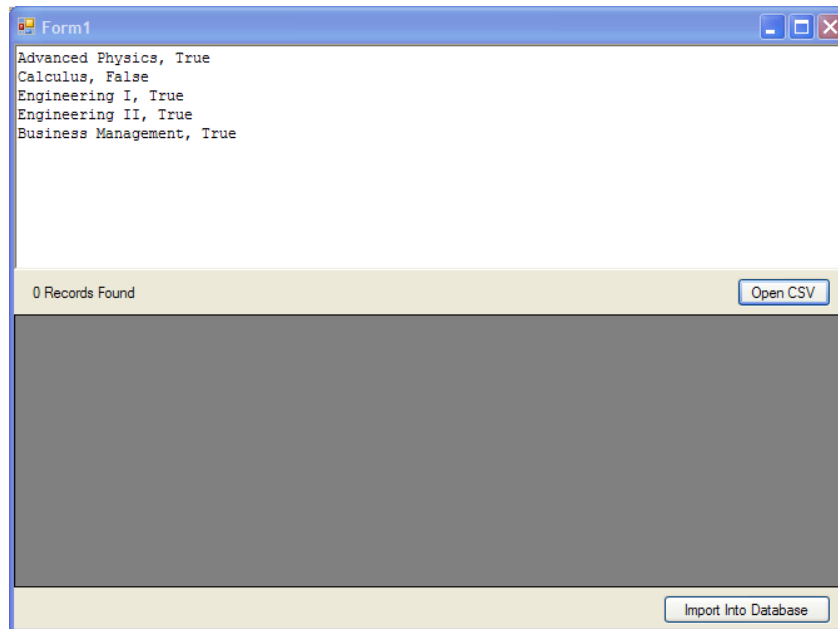
Exercise\_07

Subject : Programming

Time Slot : 1899/12/30 08:00 AM

Delete

8. Crystal Creek College has a list of subjects contained in a text file that needs to be imported into their database. Create a utility to import the data into the Subject table. Use the screenshot below for guidance on the layout of the screen. Use the top section of the screen to display the data that needs to be imported, and the bottom section of the screen for the data contained in the Subject table.



Form1

Advanced Physics, True  
Calculus, False  
Engineering I, True  
Engineering II, True  
Business Management, True

0 Records Found

Open CSV

Import Into Database

9. KC's Trading needs an improved stock management screen where the stock personnel may add, edit and also remove stock items. Create a new or improve the existing stock management screen so they can add, edit and remove stock items. Use the POSDB and refer to the screenshot below for guidance on the layout of the screen.

Exercise\_09

Description : VB.NET 2005 For Dummies

Purchase Price : 250

Available Quantity : 20

Selling Price : 300

Insert

Update

Delete

ID	Description	PurchasePrice	AvailableQuantity	SellingPrice
1	VB.NET 2005 Fo...	250	20	300
2	Exam Pad	5	500	10
3	Diary	25	75	75
4	Database Conce...	150	10	250
5	Beginning C# Pro...	100	8	120
*				

# Appendix A

## Introduction to Structured Query Language

Notes compiled by: [Leandré Roux](#)

### Introduction

**Structured Query Language** or **SQL**, is a Database script language designed for the purpose of retrieving data in such a way that it provides us with meaningful, usable information. Imagine you had a database that contained information on students that were registered at a certain institution and you were tasked with finding all the students that had registered the last two years for a certain subject, to do this you would write a **SQL query**. A SQL query is a plain text command you would issue to the **Database Management System** (DBS) that allows you to select certain data, update certain fields or tables as well as delete data from a database to name but a few possibilities.

We will look at the following SQL commands:

- ✓ `SELECT` and conditional operator `WHERE`
- ✓ `INSERT`
- ✓ `UPDATE`
- ✓ `DELETE`

We will also look at some functions you can use in conjunction with these queries to help you perform tasks such as sorting data, grouping results by certain columns and performing arithmetic functions like calculating the average, finding the maximum and summing data to provide us with more useful, structured information

Functions we will briefly discuss are:

- ✓ `AVG`
- ✓ `MAX`
- ✓ `SUM`
- ✓ `IN`
- ✓ `BETWEEN`
- ✓ `HAVING`

and ordering techniques such as:

- ✓ `GROUP BY`
- ✓ `ORDER BY`

Finally we will take a brief look at more advanced queries where we will discuss:

- ✓ `Sub Queries`
- ✓ `Parameterized Queries`

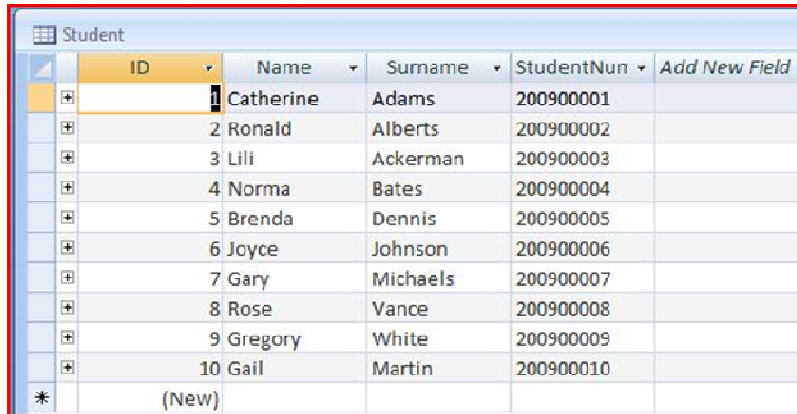


## The SELECT Statement

The `SELECT` statement allows us to retrieve data from a database by specifying the desired columns, the database to search as well as any optional group functions such as `SUM` for example. To illustrate a basic `SELECT` statement we will look at a table from **CollegeDB** which may be found under the databases directory on your computer.

*Feel free to work along on your computer, this will aid in your understanding of the topic covered.*

After loading **CollegeDB** in Microsoft Access, locate and open the **Student** table so you can look at the data in this table. You should see something similar to the screenshot below (you may have more rows):



ID	Name	Surname	StudentNun	Add New Field
1	Catherine	Adams	200900001	
2	Ronald	Alberts	200900002	
3	Lili	Ackerman	200900003	
4	Norma	Bates	200900004	
5	Brenda	Dennis	200900005	
6	Joyce	Johnson	200900006	
7	Gary	Michaels	200900007	
8	Rose	Vance	200900008	
9	Gregory	White	200900009	
10	Gail	Martin	200900010	
*	(New)			

In this example, imagine that we wanted to retrieve all the student's first names from this table, to do this we would write the following SQL query<sup>2</sup>.

```
SELECT name
FROM student;
```

This will return all the names from the *Name* column in CollegeDB. In this example we issue the `SELECT` command, telling the database that we wish to retrieve some data, then we tell it what data we want, in this case, all the student names under the *Name* column and finally we issue the `FROM` command, telling the database management system in what table to look for this data.

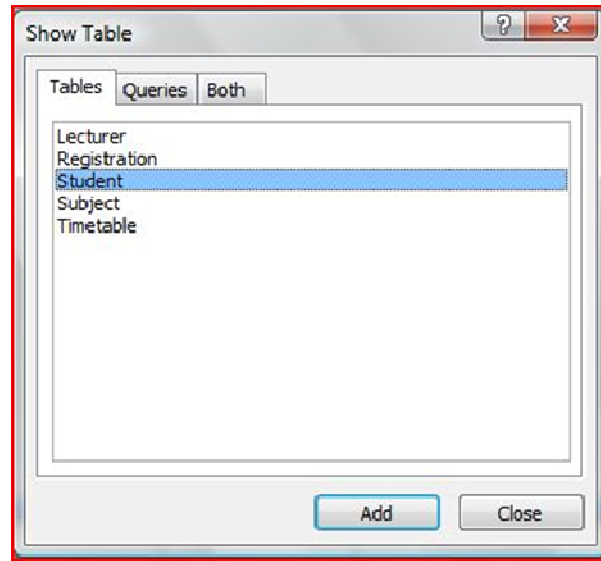
*NOTE: To select all data from a table use the wildcard \*, this will select all the data from the table:*

```
SELECT *
FROM student;
```

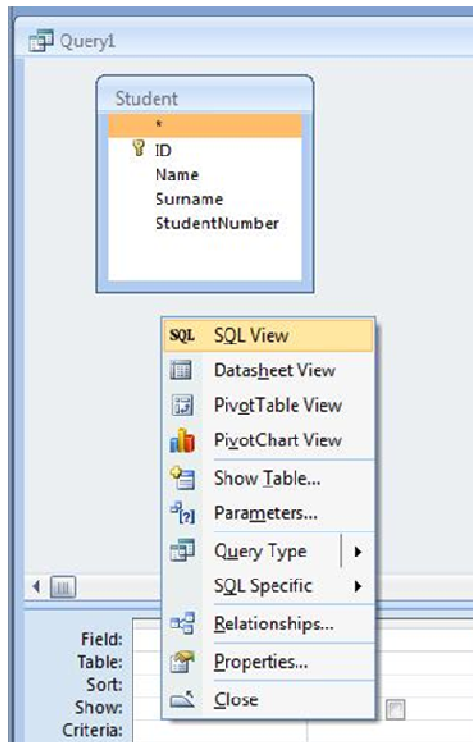
Let's execute this query from Microsoft Access and view the results. First open **CollegeDB** if it is not already open. Create a new query in design mode and select **Student** under the **Tables** tab, click on **Add** then **Close**.

---

<sup>2</sup> *NOTE: SQL is NOT case sensitive, but it is a general standard to write functions and operators in capital letters and table and column names in lower case.*



Next, **right-click** anywhere in the query view and select **SQL View** from the popup menu.



This will bring you to the SQL query editor. As you can see, Access has already written some of the query for selecting data from the Students table. Delete all the text in the window and type the following query:

```
SELECT name
FROM student;
```

Now click on the Run button (red exclamation mark) to execute the query, if you made any mistakes, the editor will notify you. You should get the following result (your result may contain more rows):

name
Catherine
Ronald
Lili
Norma
Brenda
Joyce
Gary
Rose
Gregory
Gail
*

Congratulations, you have just written your first SQL statement and have successfully executed it!

This was a simple example, but you may have to write more complex queries, for instance, let's imagine that you would like to retrieve both the *Name* and *Surname* off all students in the *Student* table, how would you go about writing this query?

It is a simple matter to retrieve multiple columns from a database, to do this you simply add more column names after the `SELECT` statement and separate each distinct column with a comma ( , ). For example, if we wish to select all the **names and surnames** from the student table we would write the following query:

```
SELECT name, surname
FROM student;
```

This tells the DBMS that we want it to `SELECT` the *Name* and *Surname* `FROM` the *Student* table in that order.

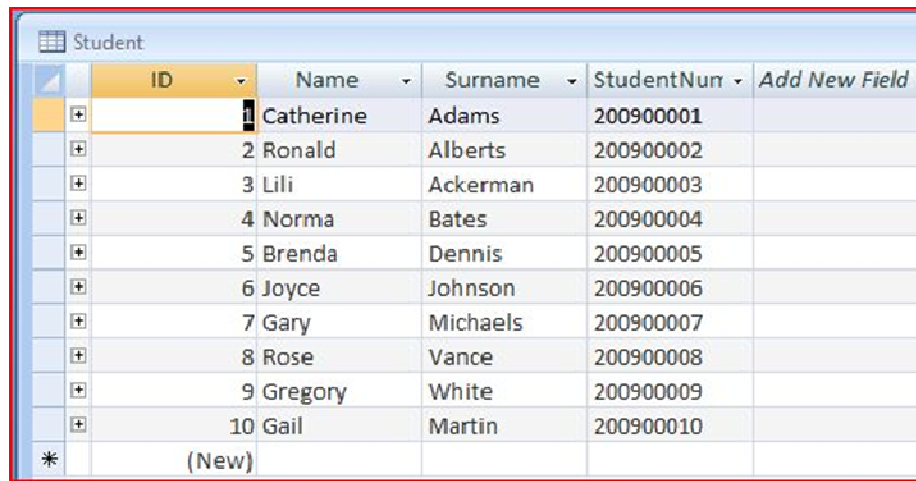
If you wish to have the surname retrieved first, you would write the following query:

```
SELECT surname, name
FROM student;
```

You should get the following result if you execute the above query in Access.

surname	name
Adams	Catherine
Alberts	Ronald
Ackerman	Lili
Bates	Norma
Dennis	Brenda
Johnson	Joyce
Michaels	Gary
Vance	Rose
White	Gregory
Martin	Gail
*	

We sometimes want to select data based on certain criteria; it is seldom useful to have a long list of information if you still need to sift through it to find what you were looking for. For example, let's say you wished to know what the name and surname of the student with student number 200900009 was.



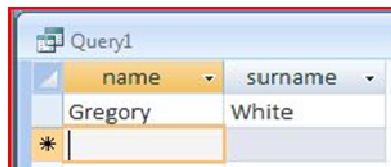
ID	Name	Surname	StudentNun	Add New Field
1	Catherine	Adams	200900001	
2	Ronald	Alberts	200900002	
3	Lili	Ackerman	200900003	
4	Norma	Bates	200900004	
5	Brenda	Dennis	200900005	
6	Joyce	Johnson	200900006	
7	Gary	Michaels	200900007	
8	Rose	Vance	200900008	
9	Gregory	White	200900009	
10	Gail	Martin	200900010	
(New)				

To do this, we perform another `SELECT` statement, provide the columns we wished to be displayed, the table from where to get it and the `WHERE` clause with a condition to help filter out all other irrelevant data.

The query would be written as:

```
SELECT name, surname
FROM student
WHERE studentnumber = '200900009'
```

You should get the following result:



name	surname
Gregory	White

In this query we told the DBMS to perform a `SELECT` and retrieve the data from the *Name* and *Surname* columns `FROM` the *Student* table and we told it to **limit** the results to rows `WHERE` the student number was **equal** to '200900009'.

### Comparing Partial Strings Using The LIKE Operator

The LIKE operator allows performing searches where a partial match of a column value is to be done. E.g. Finding all the Surnames start with an 'A'.

Consider the Student table from CollegeDB:

Student					
	ID	Name	Surname	StudentNun	Add New Field
+	1	Catherine	Adams	200900001	
+	2	Ronald	Alberts	200900002	
+	3	Lili	Ackerman	200900003	
+	4	Norma	Bates	200900004	
+	5	Brenda	Dennis	200900005	
+	6	Joyce	Johnson	200900006	
+	7	Gary	Michaels	200900007	
+	8	Rose	Vance	200900008	
+	9	Gregory	White	200900009	
+	10	Gail	Martin	200900010	
+	11	Koos	Adcock	200900011	
+	12	John	Brown	200900012	
+	13	Casey	Denell	200900013	
*	(New)				

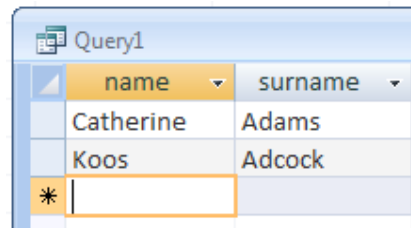
The following query will return all the student records from the student table whose surname starts with an 'A':

```
SELECT name, surname
FROM student
WHERE surname LIKE 'A*'
```

After you execute the query you should have similar results to the following:

Query1	
name	surname
Catherine	Adams
Ronald	Alberts
Lili	Ackerman
Koos	Adcock
*	

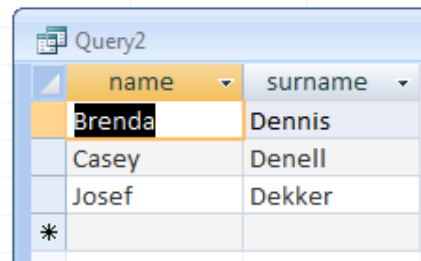
The asterisk '\*' is used as a wildcard character which represents any number of characters after the specified substring, which was an 'A' in the example changing the LIKE condition to 'Ad\*' would render the following records.



name	surname
Catherine	Adams
Koos	Adcock
*	

The asterisk is used as a wildcard character to represent a variable length condition string. The question character '?' could be used as a wildcard to represent a single character within a substring. Consider the following example and the subsequent result set returned by the query:

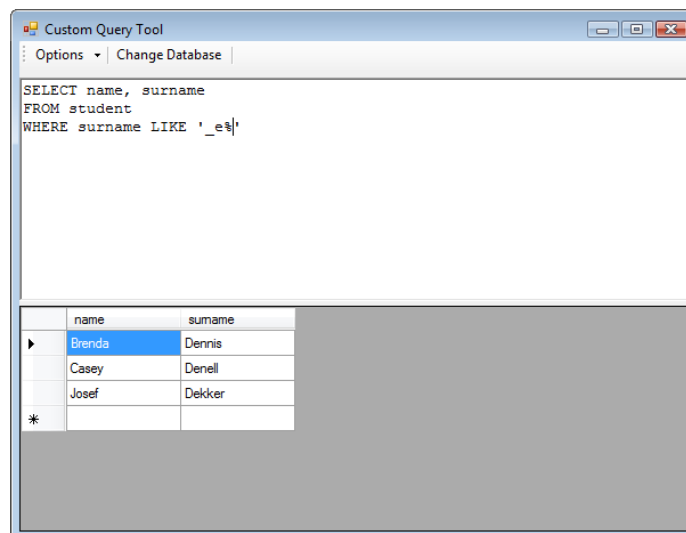
```
SELECT name, surname
FROM student
WHERE surname LIKE '?e*'
```



name	surname
Brenda	Dennis
Casey	Denell
Josef	Dekker
*	

In the above query we request the DBMS to **SELECT** the *Name* and *Surname* columns **FROM** the *Student* table **WHERE** the *Surname* column should contain an 'e' as the second letter in the surname and any amount of additional characters as indicated by the asterisk.

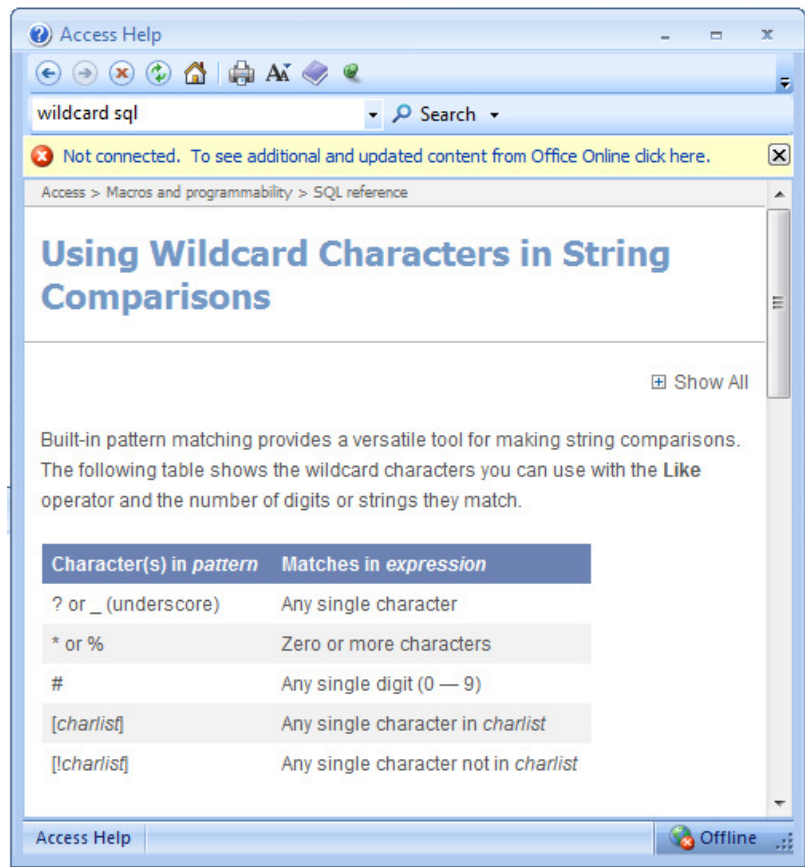
Depending on which driver is utilized by the application some of the functions and operands may differ slightly. For the LIKE statement the percentage character '%' is the equivalent of the asterisk '\*' character and the underscore character '\_' is the equivalent of the question character '?'.



```
SELECT name, surname
FROM student
WHERE surname LIKE '_e%'
```

name	surname
Brenda	Dennis
Casey	Denell
Josef	Dekker
*	

It can be seen in the figure above that the underscore and percanatge characters resturn similar results as the question and asterisk characters when used in our custom query tool. Below is a screenshot of the MS Access help file describing the operands for the LIKE function.



**Do It Yourself**

**Exercise 1:**

You have been tasked by the administration of the college to write them a SQL query that will retrieve the surname and student number of a student known as Joyce, she has forgotten to write this information on her exam paper and as she is always a top achiever, the lecturer has decided to ask for your help.

Write a SQL query that will retrieve the ***Surname*** and ***Student Number*** of the student who is known as **Joyce**.

We can perform SQL queries across multiple tables at the same time; for instance, take a look at the two tables illustrated below:

Subject				
	ID	Name	Active	Add New Field
	1	Programming	<input checked="" type="checkbox"/>	
	2	Life Studies	<input checked="" type="checkbox"/>	
	3	English	<input checked="" type="checkbox"/>	
	4	Mathematics	<input checked="" type="checkbox"/>	
*	(New)		<input checked="" type="checkbox"/>	

Timetable					
	ID	LecturerID	SubjectID	DayOfWeek	TimeOfDay
	1	1	1	1	8:00 AM
	2	1	1	3	9:00 AM
	3	2	2	2	8:00 AM
	4	2	3	2	9:00 AM
	5	1	4	4	8:00 AM
	6	1	4	5	8:00 AM
	7	3	2	4	9:00 AM
*	(New)	0	0	0	

Here we have two distinct tables named **Subject** and **Timetable**. Let's say we were tasked to write a SQL query that would be able to tell us at what time of the day a certain subject was running. You might wonder how we will achieve this, as the data is split across two different tables.

We may have multiple table names in our **FROM** clause, this tells the DBMS to search across multiple tables and combine the results. As an example, imagine we wanted to know at what time the subject **Programming** was running, how would we go about writing a query to display the desired result? Firstly, we need something that links the two tables to each other; in this case we would try to link the tables via **keys**.

As can be seen, Subject is linked to Timetable by means of **Foreign Keys**. The **Timetable** table has a column with the name 'SubjectID', this can be mapped back to the **Primary Key** field of the **Subject** table, i.e. the 'ID' column. This will become our *link* between the two tables, allowing us to write a query that uses both tables as a source.

As an example, let us write the SQL query that will retrieve the time of day that Programming is presented, in this example we want to display the course name and then the time. The query would be written as:

```
SELECT name, timeofday
FROM subject, timetable
WHERE subject.id = timetable.subjectid
AND subject.name = 'Programming';
```

You should retrieve the following result if you executed the query:

Query1	
name	timeofday
Programming	8:00 AM
Programming	9:00 AM



In the above query we told the DBMS to `SELECT` the *Name* and *TimeOfDay* columns `FROM` the tables *Subject* and *Timetable*, then we limited the results to `WHERE` the *Subject* table's *ID* column matched the *Timetable* table's *SubjectID* column, finally we added another condition using the `AND` operator, telling it to select only results where the *Subject* table's *Name* column contained the value 'Programming'.

It should be noted that it was not a requirement to prefix the columns in the `WHERE` clause with the table names, but it is preferred this way. It is necessary to do this if two columns in different tables share the same name and they appear in the same query.

You may also create an **alias** for a table if you prefer not to type out lengthy table names, for example:

```
SELECT name, timeofday
FROM subject s, timetable t
WHERE s.id = t.subjectid
AND s.name = 'Programming';
```

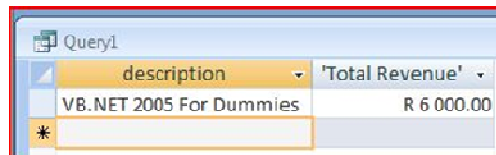
SQL queries are not limited to simply retrieving data from a database but also allow us to perform arithmetic on columns, as will be demonstrated in the following example. To follow with the next example, look at **POSDB** and located and open the *Stock* table.

Let's imagine we want to determine the expected total revenue for a certain item in stock, if we are lucky enough to sell all of this particular item. In this example we would like to know the expected revenue for the book title 'VB.NET 2005 for Dummies'.

To do this we will write the following query:

```
SELECT description, availablequantity * sellingprice AS 'Total Revenue'
FROM stock
WHERE description = 'VB.NET 2005 for Dummies';
```

You should receive the following result after executing the query:



description	Total Revenue
VB.NET 2005 For Dummies	R 6 000.00
*	

In the above query we told the DBMS to `SELECT` the column *Description* and to take the value stored in the column *AvailableQuantity* and **multiply** it by the amount stored in *SellingPrice* and to display the result `AS` *Total Revenue*, it needs to retrieve data `FROM` the *Stock* table `WHERE` the *Description* matches 'VB.NET 2005 for Dummies'.

The `AS` operator we used above is used to rename a column to a suitable or desired title during the display of the data and comes in handy we performing arithmetic operations. To see why we say this, run the above query again but omit the `AS 'Total Revenue'` and view the column name given by the DBMS.

## The INSERT Statement

The `INSERT` statement is used to add data to a table by means of writing a query, it is mostly used in applications where you wish to capture data from the user and store this data in a database so you can perform various queries to help analyze the data stored in the database.

As an example, let's look at the **POSDB** database, open the database in Microsoft Access and locate and open the table *Stock*. The structure of the table is as follows (You may have more rows of data):



ID	Description	PurchasePrice	AvailableQuantity	SellingPrice	Add New Field
1	VB.NET 2005 For Dummies	R 250.00	20	R 300.00	
2	Exam Pad	R 5.00	500	R 10.00	
3	Diary	R 25.00	75	R 75.00	
4	Database Concepts	R 150.00	10	R 250.00	
5	Beginning C# Programming	R 100.00	8	R 120.00	
(New)		R 0.00	0	R 0.00	

We have just received new stock and wish to add this to our Stock table so we can keep track of the sale of these items without having to keep a manual written record.

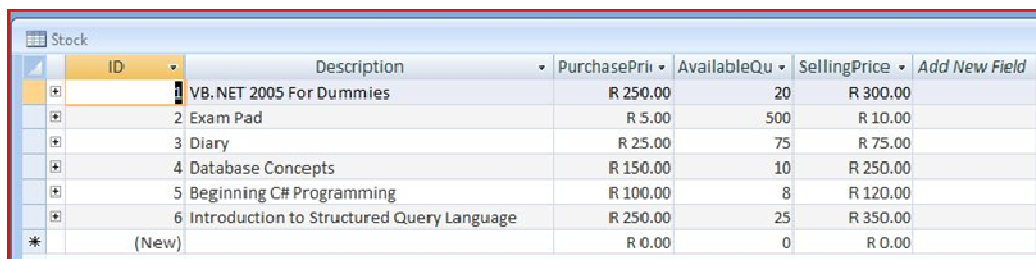
The new item has the following characteristics:

Expected Column	Data
Description	Introduction to Structured Query Language
PurchasePrice	250
AvailableQuantity	25
SellingPrice	350

If we wished to add the data to the table we would write a SQL query similar to this:

```
INSERT INTO stock (ID, description, purchaseprice, availablequantity, sellingprice)
VALUES (6, 'Introduction to Structured Query Language', 250, 25, 350);
```

The expected result should be as follows (you may have more rows):



ID	Description	PurchasePrice	AvailableQuantity	SellingPrice	Add New Field
1	VB.NET 2005 For Dummies	R 250.00	20	R 300.00	
2	Exam Pad	R 5.00	500	R 10.00	
3	Diary	R 25.00	75	R 75.00	
4	Database Concepts	R 150.00	10	R 250.00	
5	Beginning C# Programming	R 100.00	8	R 120.00	
6	Introduction to Structured Query Language	R 250.00	25	R 350.00	
(New)		R 0.00	0	R 0.00	

**NOTE:** When executing the query in Microsoft Access, you will be blocked from executing the query, allow the insert by following the prompts on screen.

The above query told the DBMS to perform an `INSERT INTO` the table *Stock*, we **optionally** listed the columns found in the table and told it to add the `VALUES` listed between the brackets, the order of which matched the order of the columns in the table *Stock*.

We could write the same query as:

```
INSERT INTO stock
VALUES(6, 'Introduction to Structured Query Language', 250, 25, 350);
```

In this example we added the ID column manually; the ID field is of type *auto increment* which means that it automatically increases by 1 every time a row is inserted, so how can we insert a row without knowing the current value of the primary key ID? We simply list the values we wish to insert and omit the ones that we want to use the default database value for.

**NOTE:** Some database columns do not have default values and may require you to provide one when performing an `INSERT`.

As an example, let's add another row to the table:

```
INSERT INTO stock(description,purchaseprice,availablequantity,sellingprice)
VALUES('Easy SQL Vol.1', 150, 5, 250);
```

The result should be something similar (your insert may be at the end of the table):

Stock						
	ID	Description	PurchasePri	AvailableQu	SellingPrice	Add New Field
	1	VB.NET 2005 For Dur	R 250.00	20	R 300.00	
	2	Exam Pad	R 5.00	500	R 10.00	
	3	Diary	R 25.00	75	R 75.00	
	4	Database Concepts	R 150.00	10	R 250.00	
	5	Beginning C# Progra	R 100.00	8	R 120.00	
	6	Introduction to Stru	R 250.00	25	R 350.00	
	7	Easy SQL Vol.1	R 150.00	5	R 250.00	

## The UPDATE Statement

The `UPDATE` statement is used to update data in a database; this is used to keep the data in the database relevant or to apply changes to certain fields without the need of performing new data inserts or creating a new database from scratch.

Still working with our **POSDB** let us update some of the data found in the *Stock* table. Recently we have purchased stock of items already in the database; we only wish to update the relevant fields in the table to reflect the new stock levels and prices.

Stock						
	ID	Description	PurchasePri	AvailableQu	SellingPrice	Add New Field
	1	VB.NET 2005 For Dur	R 250.00	20	R 300.00	
	2	Exam Pad	R 5.00	500	R 10.00	
	3	Diary	R 25.00	75	R 75.00	
	4	Database Concepts	R 150.00	10	R 250.00	
	5	Beginning C# Progra	R 100.00	8	R 120.00	
	6	Introduction to Stru	R 250.00	25	R 350.00	
	7	Easy SQL Vol.1	R 150.00	5	R 250.00	

We have bought, among others, the following stock:

Stock Name	Amount
Exam Pad	40
Easy SQL Vol.1	30

Also the *Selling Price* of 'Beginning C# Programming' has changed to R 160.00.

Let's start off by updating the selling price of "Beginning C# Programming'. To update the *SellingPrice* column with the new price is accomplished by writing the following query:

```
UPDATE stock
SET sellingprice = 160
WHERE description = 'Beginning C# Programming';
```

You should see the updated value is reflected in the database after executing the query:

Stock					
	ID	Description	PurchasePrice	AvailableQty	SellingPrice
+	1	VB.NET 2005 For Dummies	R 250.00	20	R 300.00
+	2	Exam Pad	R 5.00	500	R 10.00
+	3	Diary	R 25.00	75	R 75.00
+	4	Database Concepts	R 150.00	10	R 250.00
+	5	Beginning C# Programming	R 100.00	8	R 160.00
+	6	Introduction to Structured Query Language	R 250.00	25	R 350.00
+	7	Easy SQL Vol.1	R 150.00	5	R 250.00

In the above query we told the DBMS to `UPDATE` the *Stock* table and to `SET` the *SellingPrice* column's value to 160 `WHERE` the *Description* column was equal to 'Beginning C# Programming'.

Let's update the stock count for the Exam Pads in the Stock table, to accomplish this we would write an update query with an arithmetic operation to add the new stock to the existing stock in our table.

The query would be written as follows:

```
UPDATE stock
SET availablequantity = availablequantity + 40
WHERE description = 'Exam Pad';
```

After executing the query you should see the updates AvailableQuantity field has increased by 40:

Stock						
	ID	Description	PurchasePrice	AvailableQty	SellingPrice	Add New Field
+	1	VB.NET 2005 For Dummies	R 250.00	20	R 300.00	
+	2	Exam Pad	R 5.00	540	R 10.00	
+	3	Diary	R 25.00	75	R 75.00	
+	4	Database Concepts	R 150.00	10	R 250.00	
+	5	Beginning C# Programming	R 100.00	8	R 160.00	
+	6	Introduction to Structured Query Language	R 250.00	25	R 350.00	
+	7	Easy SQL Vol.1	R 150.00	5	R 250.00	

## The DELETE Statement

The `DELETE` statement is used to remove rows from a database, this statement is used to remove irrelevant information or to perform audit functions. As an example, let us imagine that we have sold all our copies of 'Introduction to Structured Query Language' and due to the book becoming disused by institutions we wish to remove it from the *Stock* table as we will not purchase new stock in the future.

We would write the following SQL query to accomplish this task:

```
DELETE FROM stock
WHERE description = 'Introduction to Structured Query Language';
```

You should find that the entry for this item has been deleted from the database:



ID	Description	PurchasePrice	AvailableQu	SellingPrice	Add New Field
1	VB.NET 2005 Fc	R 250.00	20	R 300.00	
2	Exam Pad	R 5.00	540	R 10.00	
3	Diary	R 25.00	75	R 75.00	
4	Database Conc	R 150.00	10	R 250.00	
5	Beginning C# P	R 100.00	8	R 160.00	
7	Easy SQL Vol.1	R 150.00	5	R 250.00	

**WARNING:** Always include a `WHERE` clause, omitting the `WHERE` clause will **delete all information** in the table specified.

In the above query we requested the DBMS to `DELETE` data `FROM` the `Stock` table `WHERE` the `Description` column was equal to `'Introduction to Structured Query Language'`.

It should be noted that you cannot delete a row from a table if it is being referenced by another table's foreign key, i.e. if the row you wish to delete's primary key is referenced by another table as a foreign key. This is known as integrity constraints. To delete the row, you must first delete the row that references the primary key.

## The AVG, MAX And SUM Functions

We will briefly discuss a few group functions that perform useful operations on selected data from a database. These functions allow you to perform arithmetic on columns automatically, for example the `SUM` function adds all data from a certain column and returns the result. The `MAX` function can be used to find the maximum value in a column and finally the `AVG` function may be used to find the average in a selected column.

Group functions are used in the `SELECT` statement clause during the retrieval of information. As an example, we will perform a `SUM` operation on the `Salary` column of **DaringDevelopmentInc**. Locate and open the database **DaringDevelopmentInc** and open the `Employee` table.

You should see something similar to the following illustration:

ID	Firstname	Surname	Job_Id	Salary	Add New Field
1	John	Peterson	1	R 60,000.00	
2	Sally	Gates	2	R 12,000.00	
3	Dave	Young	2	R 17,000.00	
4	Pete	Shire	3	R 20,000.00	
5	Bethany	Andrews	4	R 20,000.00	
6	Alice	Young	6	R 19,000.00	
7	Peter	McFarlain	2	R 10,000.00	
8	Michelle	Van Zyl	3	R 20,000.00	
9	Henry	Davis	3	R 23,000.00	
10	Carl	Kent	6	R 17,000.00	
11	Rupert	West	3	R 21,000.00	
12	Samuel	Michaels	7	R 10,000.00	
13	Claire	Titmus	7	R 9,000.00	
14	Denise	Casta	7	R 10,000.00	
15	Samuel	West	7	R 11,000.00	
16	Dorothy	De Haan	2	R 13,000.00	
17	Jane	Hartstein	4	R 20,000.00	
18	Graig	Higgins	2	R 10,000.00	
19	James	Rajs	2	R 14,000.00	
20	Lynette	Matos	3	R 21,000.00	

The query should look similar to this one:

```
SELECT SUM(salary) AS 'Total Monthly Expenses'
FROM employee;
```

The expected result should be as follows:

'Total Monthly Expenses'		
R 414,000.00		

In the above query, we requested the DBMS to `SELECT` the *Salary* column and to perform the `SUM` function on the retrieved data and to display it `AS` 'Total Monthly Expenses'. The data is to be retrieved `FROM` the *Employee* table.

If we wish to calculate the average payout to employees we can use the `AVG` function to calculate this on the *Salary* column. For example:

```
SELECT AVG(salary) AS 'Average Salary Payout'
FROM employee;
```

The expected result should be similar to the following illustration:

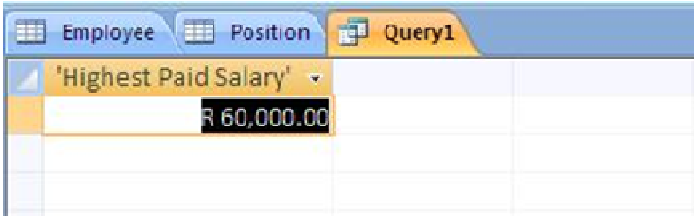
'Average Salary Payout'		
R 18,000.00		

In the above query, we requested the DBMS to `SELECT` the *Salary* column and to perform the `AVG` function on the retrieved data and to display it `AS` 'Average Salary Payout'. The data is to be retrieved `FROM` the *Employee* table.

Finally, if we wish to know what the largest value in a column is we can use the `MAX` function to display this. If we wished to know what the largest salary paid out to an individual was in the *Employee* table we would write the following query:

```
SELECT MAX(salary) AS 'Highest Paid Salary'
FROM employee;
```

The expected result should be similar to the following illustration:



In the above query, we requested the DBMS to `SELECT` the *Salary* column and to perform the `MAX` function on the retrieved data and to display it `AS` 'Highest Paid Salary'. The data is to be retrieved `FROM` the *Employee* table.

There are numerous other group functions at your disposal and they are used in exactly the same fashion as the previous illustrations, here is a brief list of available group functions you might find useful:

Function	Intended Use	Example
<b>MIN</b>	Find the lowest value	<code>SELECT MIN(salary)</code>
<b>MAX</b>	Find the highest value	<code>SELECT MAX(Salary)</code>
<b>AVG</b>	Find the average value	<code>SELECT AVG(salary)</code>
<b>SUM</b>	Find the totaled value	<code>SELECT SUM(salary)</code>
<b>COUNT</b>	Counts the rows of the column	<code>SELECT COUNT(surname)</code>

### The IN Condition

The `IN` condition is used to display results from a `SELECT` statement that match the specified values exactly. When using the `IN` condition, you specify a comma delimited list of legal values to check for. For this example we will still be using the *Stock* table from **POSDB**.

A screenshot of a database table named 'Stock'. The table has six columns: ID, Description, PurchasePrice, AvailableQuantity, SellingPrice, and Add New Field. The data rows are as follows:

ID	Description	PurchasePrice	AvailableQuantity	SellingPrice	Add New Field
1	VB.NET 2005 Fc	R 250.00	20	R 300.00	
2	Exam Pad	R 5.00	540	R 10.00	
3	Diary	R 25.00	75	R 75.00	
4	Database Conc	R 150.00	10	R 250.00	
5	Beginning C# P	R 100.00	8	R 150.00	
7	Easy SQL Vol.1	R 150.00	5	R 250.00	

Let us imagine you were asked to retrieve the amount of stock available for Exam Pads and Diaries. To achieve our goal, we would write a SQL query similar to this one, we also chose to display the *Description* column data so we could identify what stock belongs to what product:



```
SELECT description, availablequantity
FROM stock
WHERE description IN('Exam Pad', 'Diary');
```

You should receive all the product listed using the IN clause:

description	availablequantity
Exam Pad	540
Diary	75
*	0

In the above query we requested the DBMS to **SELECT** the *Description* and the *AvailableQuantity* column's data **FROM** the *Stock* table **WHERE** the *Description* column had values **IN** the ones listed between the brackets.

## The BETWEEN Condition

The **BETWEEN** condition is used to display results from a **SELECT** statement that are in a certain range, this is very useful if you wish to retrieve all data that satisfy the lower and upper limit you specify.

As an example, let us look at the *Stock* table in **POSDB**:

ID	Description	PurchasePrice	AvailableQuantity	SellingPrice	Add New Field
1	VB.NET 2005 Fc	R 250.00	20	R 300.00	
2	Exam Pad	R 5.00	540	R 10.00	
3	Diary	R 25.00	75	R 75.00	
4	Database Concepts	R 150.00	10	R 250.00	
5	Beginning C# Programming	R 100.00	8	R 160.00	
7	Easy SQL Vol.1	R 150.00	5	R 250.00	

For this example, imagine you were tasked to find all stock that had a *Selling Price* above R100.00 but not greater than R250.00. To do this is quite simple when you use the **BETWEEN** conditional operator, you would write a query similar to the one below:

```
SELECT description, sellingprice
FROM stock
WHERE sellingprice BETWEEN 100 AND 250;
```

You should receive a list of items that fall in this price range as can be seen in the image below (your results may vary):

description	sellingprice
Database Concepts	R 250.00
Beginning C# Programming	R 160.00
Easy SQL Vol.1	R 250.00
*	R 0.00

In the above query we requested the DBMS to **SELECT** the *Description* as well as the *SellingPrice* columns **FROM** the *Stock* table **WHERE** the *SellingPrice* column had a value **BETWEEN** 100 AND 250.



## The ORDER BY Clause

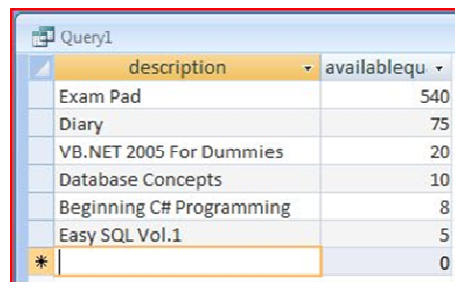
The `ORDER BY` clause is used to sort data after a `SELECT` statement has executed, this helps you order results either ascending or descending based on a specified column or columns. For this example we will use the *Stock* table from **POSDB**.

Imagine you have been tasked to perform a query on the *Stock* table, displaying all the stock from most available quantity to least available quantity, the job requires you to display only the *Description* and *AvailableQuantity* columns sorted **descending**.

To achieve this we would write the following query:

```
SELECT description, availablequantity
FROM stock
ORDER BY availablequantity DESC;
```

You should receive all the stock in the table, sorted descending by the available quantity (your result may vary):



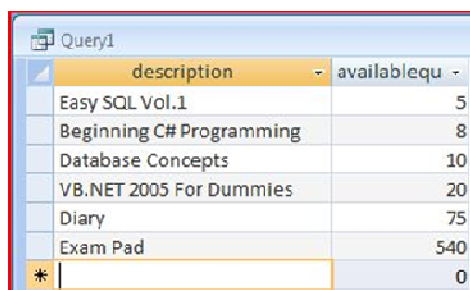
description	availablequ
Exam Pad	540
Diary	75
VB.NET 2005 For Dummies	20
Database Concepts	10
Beginning C# Programming	8
Easy SQL Vol.1	5
*	0

In the above example we requested the DBMS to `SELECT` the *Description* and *AvailableQuantity* columns `FROM` the *Stock* table and to take the results and `ORDER BY` the *AvailableQuantity* column `DESC` (descending).

The same query may be modified to display the data in a ascending format, as shown below:

```
SELECT description, availablequantity
FROM stock
ORDER BY availablequantity ASC;
```

You should receive all the stock in the table, sorted ascending by the available quantity (your result may vary):



description	availablequ
Easy SQL Vol.1	5
Beginning C# Programming	8
Database Concepts	10
VB.NET 2005 For Dummies	20
Diary	75
Exam Pad	540
*	0

There may be times that you need to order by a column name that has been changed or is using an alias, as an example recall this exercise from the `SELECT` tutorial:

```
SELECT description, availablequantity * sellingprice AS 'Total Revenue'
FROM stock
WHERE description = 'VB.NET 2005 for Dummies';
```

In this example Total Revenue is a column **alias**, used to create a more readable column name than the one the DBMS would choose if we did not supply any. To illustrate this, let us execute the above query without using the AS clause, we should see a similar result:

description	Expr1001
VB.NET 2005 For Dummies	R 6 000.00

As you may notice, the column has been named 'Expr1001' by the DBMS, it does in no way tell us what the value displayed beneath it represents. This is why we choose to use aliases for columns that are the result of arithmetic or column combinations, you may optionally provide aliases for all columns during a `SELECT` if you wish to do so.

What would happen if we wanted to execute the above query and wanted to order the results by the second column, it will not always be the column name "Expr1001".

We can `ORDER BY` the alias given to the column name, as shown below, we will omit the `WHERE` clause so we have more data to illustrate the result:

```
SELECT description, availablequantity * sellingprice AS 'Total Revenue'
FROM stock
WHERE description = 'VB.NET 2005 for Dummies'
ORDER BY 'Total Revenue' ASC;
```

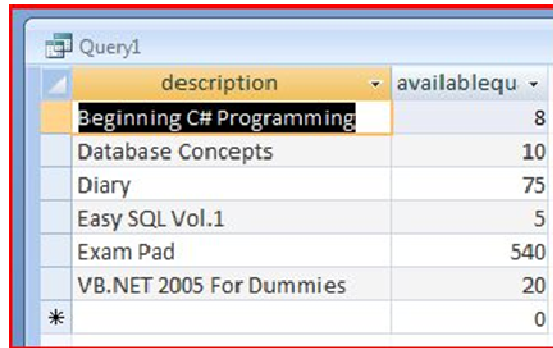
You should receive all the stock in the table, sorted ascending by the available quantity (your result may vary):

description	Total Rever
Easy SQL Vol.1	R 1 250.00
Beginning C# Programming	R 1 280.00
Database Concepts	R 2 500.00
Diary	R 5 625.00
Exam Pad	R 5 400.00
VB.NET 2005 For Dummies	R 6 000.00

You may also `ORDER BY` multiple columns by listing them:

```
SELECT description, availablequantity
FROM stock
ORDER BY description, availablequantity DESC;
```

You should receive all the stock in the table, sorted descending by the description, then available quantity (your result may vary):



description	availablequantity
Beginning C# Programming	8
Database Concepts	10
Diary	75
Easy SQL Vol.1	5
Exam Pad	540
VB.NET 2005 For Dummies	20
*	0

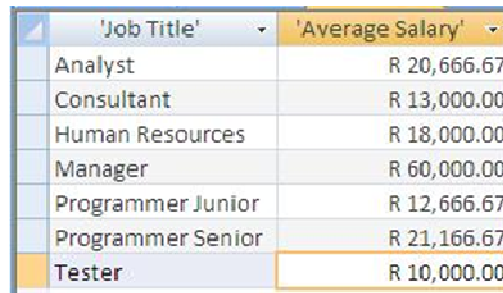
## The GROUP BY Clause

The GROUP BY clause is used to combine rows into a single result, for instance, imagine you had to calculate the average salary for a certain position in a company, using the group by function you can then group the results by the positions in the company.

As an example we will look at the **DaringDevelopmentInc** database. Open the database in Microsoft Access and perform the following SQL query:

```
SELECT jobtitle AS 'Job Title', AVG(salary) AS 'Average Salary'
FROM position, employee
WHERE position.id = employee.job_id
GROUP BY jobtitle;
```

You should receive the following results:



'Job Title'	'Average Salary'
Analyst	R 20,666.67
Consultant	R 13,000.00
Human Resources	R 18,000.00
Manager	R 60,000.00
Programmer Junior	R 12,666.67
Programmer Senior	R 21,166.67
Tester	R 10,000.00

In the above query we requested the DBMS to `SELECT` the *JobTitle* column and display it as 'Job Title' then calculate the `AVG` of the column *Salary* and display it as 'Average Salary' `FROM` the table *Position* and *Salary* `WHERE` the *ID* column in the table *Position* matched the *Job\_Id* column in the *Employee* table. Finally we told the DBMS to `GROUP BY` the *JobTitle* column, i.e. we told the DBMS to calculate the average of all salary entries in the database and to link it to a relevant job title.

## The HAVING Clause

The `HAVING` clause is used to set a condition for restricting groups in a `GROUP BY` query's result. As an example, imagine that you wish to know the average salary for all employees that receive a salary below R20, 000.00 a month.

Recall the previous query:

```
SELECT jobtitle AS 'Job Title', AVG(salary) AS 'Average Salary'
FROM position, employee
WHERE position.id = employee.job_id
GROUP BY jobtitle;
```

This query provided us with the average of **all salary** entries in the *Employee* table, but if we wish to know only the average of salaries below R20, 000.00 we could modify the query as follows:

```
SELECT jobtitle AS 'Job Title', AVG(salary) AS 'Average Salary'
FROM position, employee
WHERE position.id = employee.job_id
GROUP BY jobtitle
HAVING AVG(salary) < 20000;
```

The result is as follow:



The screenshot shows a database window with a query named 'Query1'. The query results are displayed in a table with two columns: 'Job Title' and 'Average Salary'. The data is as follows:

'Job Title'	'Average Salary'
Consultant	R 13,000.00
Human Resources	R 18,000.00
Programmer Junior	R 12,666.67
Tester	R 10,000.00

## Advanced Topics

In the following section we will look at more advanced topics that you may use in SQL to retrieve or modify data in a database that would be either impossible or a lot more work to achieve without knowledge of the topics discussed here.

First, let us look at **Sub Queries** and how they can be used to retrieve data from the database. A sub query is nothing more than an additional `SELECT` statement that is performed at the `WHERE` clause of a query.

To demonstrate a basic sub query, let's look at the POSDB database's Stock table.



The screenshot shows a database window with a table named 'Stock'. The table has the following columns: ID, Description, PurchasePrice, AvailableQuantity, SellingPrice, and Add New Field. The data is as follows:

ID	Description	PurchasePrice	AvailableQuantity	SellingPrice	Add New Field
1	VB.NET 2005 Fc	R 250.00	20	R 300.00	
2	Exam Pad	R 5.00	540	R 10.00	
3	Diary	R 25.00	75	R 75.00	
4	Database Conc	R 150.00	10	R 250.00	
5	Beginning C# P	R 100.00	8	R 160.00	
7	Easy SQL Vol.1	R 150.00	5	R 250.00	

Let's imagine you were tasked by the manager to determine what products in the *Stock* table were more expensive than the best seller **Beginning C# Programming** so he can decide if a discount can be applied on those titles to try and improve sales on those titles.

To easily retrieve the information we can use a sub query to retrieve the price of Beginning C# Programming, this is helpful if you did not know what the SellingPrice was for this title, which will be the case most of the time in real life situations.

We could write a query similar to the following

```
SELECT description, sellingprice
FROM stock
WHERE sellingprice > (
    SELECT sellingprice
    FROM stock
    WHERE description = 'Beginning C# Programming');
```

The result is as follows:

Query1	
description	sellingprice
VB.NET 2005 For Dummies	R 300.00
Database Concepts	R 250.00
*	R 0.00

In the above query we requested that the DBMS retrieves the *Description* and *SellingPrice* columns FROM the *Stock* table WHERE the *SellingPrice* is greater than the *SELECT(ed) SellingPrice* FROM the *Stock* table WHERE *Description* is equal to 'Beginning C# Programming'.

The sub query is nothing else than another query and may contain Group Functions such as MAX, AVG, SUM etc. You may have sub queries for each distinct WHERE clause as well as nested sub queries if needed.

We will demonstrate how you can use multiple sub queries as well as group functions when performing data retrieval. Imagine you were asked to find all the products in the *Stock* table more expensive than the Exam Pad but less expensive than the most expensive product in the database.

We could write a similar query to the following:

```
SELECT description, sellingprice
FROM Stock
WHERE sellingprice > (
    SELECT sellingprice
    FROM stock
    WHERE description = 'Exam Pad')

AND sellingprice < (
    SELECT MAX(sellingprice)
    FROM stock);
```

The results are as follows:

Query1	
description	sellingprice
Diary	R 75.00
Database Concepts	R 250.00
Beginning C# Programming	R 120.00
*	R 0.00

In the above query we requested that the DBMS retrieves the *Description* and *SellingPrice* columns FROM the *Stock* table WHERE the *SellingPrice* is greater than the SELECT(ed) *SellingPrice* FROM the *Stock* table WHERE *Description* is equal to 'Exam Pad' AND the *SellingPrice* was less than the SELECT(ed) MAX *SellingPrice* FROM the *Stock* table.

To order the results you may add an ORDER BY clause as the last statement of the query:

```
SELECT description, sellingprice
FROM Stock
WHERE sellingprice > (      SELECT sellingprice
                           FROM stock
                           WHERE description = 'Exam Pad')

AND sellingprice < (      SELECT MAX(sellingprice)
                           FROM stock)
ORDER BY sellingprice DESC;
```

The results are as follows:

	description	sellingprice
	Database Concepts	R 250.00
	Beginning C# Programming	R 120.00
	Diary	R 75.00
*		R 0.00

In the following section we will look at **Parameterized Queries**, these queries are SQL commands that contain **variables** for values that we wish to change during runtime. The simplest example for the need for such queries is in the event that we want to provide the parameters which must be used to perform the SQL statement.

An example might be if we wish to search the *SellingPrice* of an item in the *Stock* table by providing the product *Description* and then searching for the result.

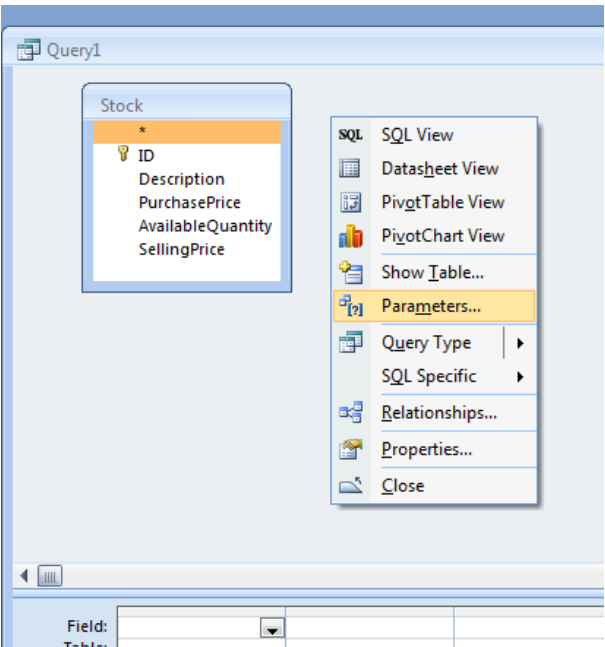
The use of parameters differs from DBMS to DBMS, we will therefore only discuss the concept behind this approach using Microsoft Access as the tool for executing parameterized queries.

Let's start with a basic example of the use of parameterized queries. In this example, imagine that you wish to see the *PurchasePrice* as well as the *SellingPrice* for a product in the *Stock* table of POSDB, but you wish to supply the *Description* during execution so you may change it without having to modify the query every time.

Consider the data in the *Stock* table:

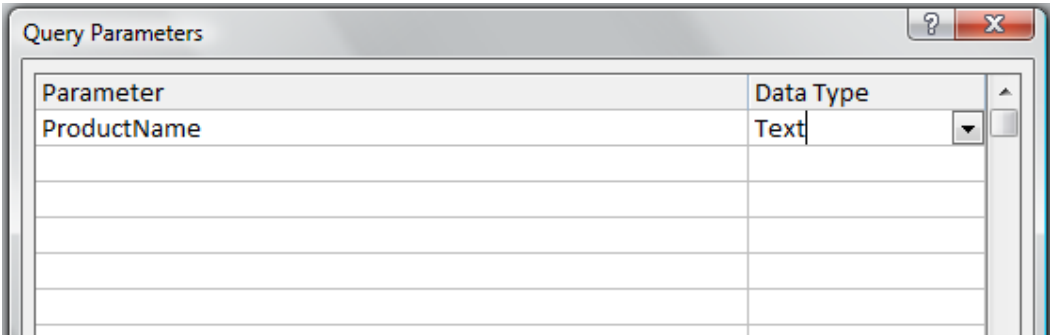
	ID	Description	PurchasePrice	AvailableQuantity	SellingPrice	Add New Field
	1	VB.NET 2005 Fc	R 250.00	20	R 300.00	
	2	Exam Pad	R 5.00	540	R 10.00	
	3	Diary	R 25.00	75	R 75.00	
	4	Database Conc	R 150.00	10	R 250.00	
	5	Beginning C# P	R 100.00	8	R 160.00	
	7	Easy SQL Vol.1	R 150.00	5	R 250.00	

In Access locate and open the *Stock* table of **POSDB** then proceed to the **Query Designer** as explained in the beginning of this chapter. Add the *Stock* table from the table selection popup. Right-Click anywhere in the designer window and select **Parameters** from the list a new popup should become visible.

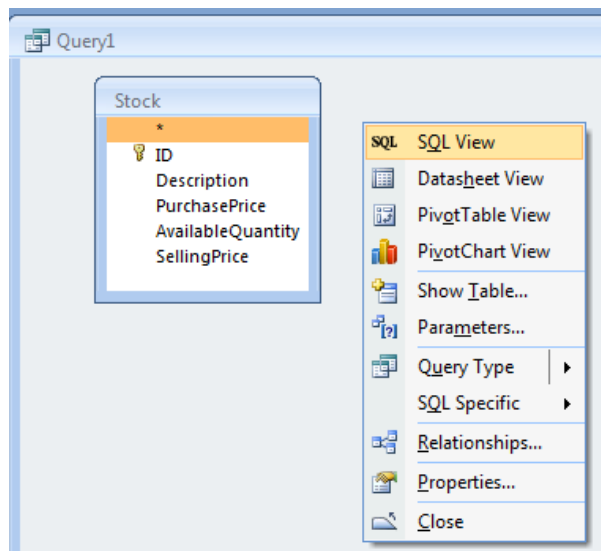


This window allows you to specify parameters to be used within the queries for the *Stock* table. Here you specify a parameter as a text value and then select a data type from the available options to the right.

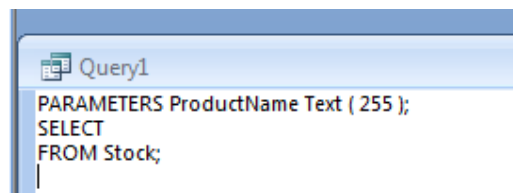
In this example we name the parameter **ProductName** and select **Text** as the data type.



When you are done, select OK to close the dialog. Again, right-click in the designer view and select SQL View to open the query editor.



You should see the following text in the query editor:

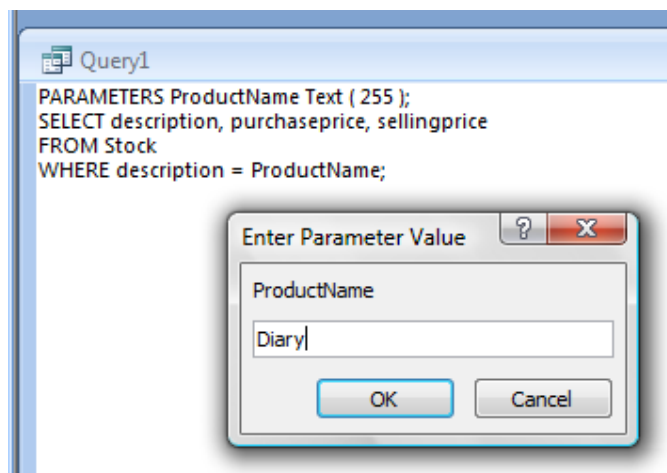


Access has completed most of the required functions needed to execute the query, we just need to specify the columns and add a WHERE clause to the existing query.

Modify the query to reflect the following changes:

```
PARAMETERS ProductName Text ( 255 );
SELECT description, purchaseprice, sellingprice
FROM Stock
WHERE description = ProductName;
```

Next, execute the query; you should be prompted to enter the value for ProductName. In this example we give it 'Diary' as the input parameter.





When you click OK the DBMS executes the query in the following manner. It places the value 'Diary' into the variable **ProductName** and then performs a `SELECT` of the *Description*, *PurchasePrice* and *SellingPrice* columns `FROM` the *Stock* table `WHERE` *Description* is equal to the variable content of **ProductName**, in this case 'Diary'.

You should see the following result:

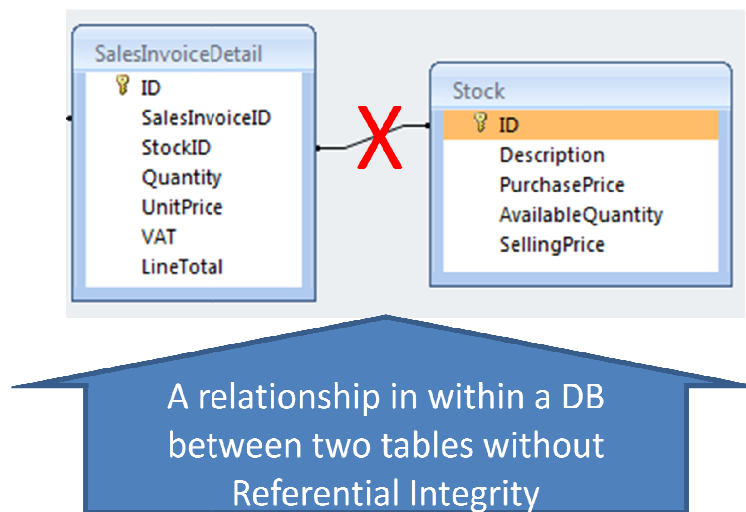
Query1		
description	purchasepri	sellingprice
Diary	R 25.00	R 75.00
*	R 0.00	R 0.00

## Understanding table joins

Data stored in a database is usually grouped and stored as separate entities that share common characteristics, for instance, if you have a database that stores information for both students and lecturers at an institution, you do not want to store this information in the same table even though they both may contain entries such as first name and last name. The idea is to separate data into smaller meaningful and related entities.

The separated data needs a way of relating to other data stored in other tables in the database, for this we have a common field in tables that are related. This field contains a unique key that can be used to link data that is related between two or more tables.

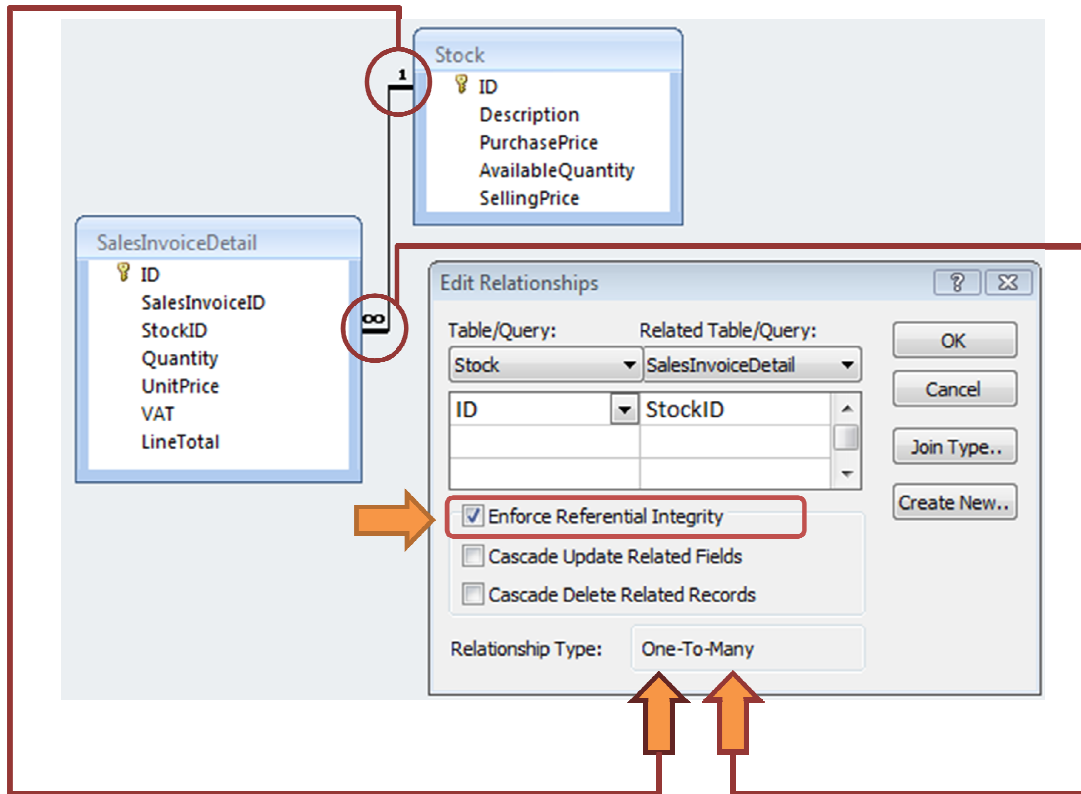
As an example, consider this fragment from POSDB's **Entity Relationship Diagram (ERD)**:



Consider the following relationship as supplied by the database relationship viewer in MS Access. If one closely study the relationship between the two tables as indicated in the figure above, one will note that referential integrity has not been set up between the two tables. The rule of referential integrity within the context of relational database design states that for every foreign key field a corresponding primary key value should exist. Applying strict referential integrity ensures that data anomalies are kept to the minimum. In other words, it will not be possible for the user of a database to delete a primary key record in the parent table if corresponding foreign key references still exist in the child table.

## Setting up referential Integrity in Access

This diagram is a visual representation of the columns in distinct tables as well as what links or relationships exists between them. In this example we are looking at the SalesInvoiceDetail table as well as the Stock table. In both fields the unique identifier is denoted by the field with the image of the 'key', this is known as a **Primary Key (PK)** and may never repeat itself in the same table it is declared, i.e. Imagine the Stock table's Primary Key, in this instance ID, had a value of 1 for the first row of data contained therein (See image below), any other row of data inserted into this table may never have the value 1 for its ID field.



ID	Description	PurchasePrice	AvailableQu	SellingPrice	Add New Field
1	VB.NET 2005 Fc	R 250.00	20	R 300.00	
2	Exam Pad	R 5.00	540	R 10.00	
3	Diary	R 25.00	75	R 75.00	
4	Database Conc	R 150.00	10	R 250.00	
5	Beginning C# P	R 100.00	8	R 160.00	
7	Easy SQL Vol.1	R 150.00	5	R 250.00	

SalesInvoiceDetail uses its relationship with Stock to "Know" what items have been sold, the reason for this is that you should try to keep repeating data to a minimum or completely prevent data from repeating in a database. It simply stores a reference to the data that symbolizes the sold products, by adding the Stock table's Primary Key as a **Foreign Key (FK)** in itself.

In this example the table field StockID is the Foreign Key that links to Stock table's Primary Key, linking all data in that row to the data in SalesInvoiceDetail's row that contains it as a Foreign Key. Consider the image below to help clarify this point:

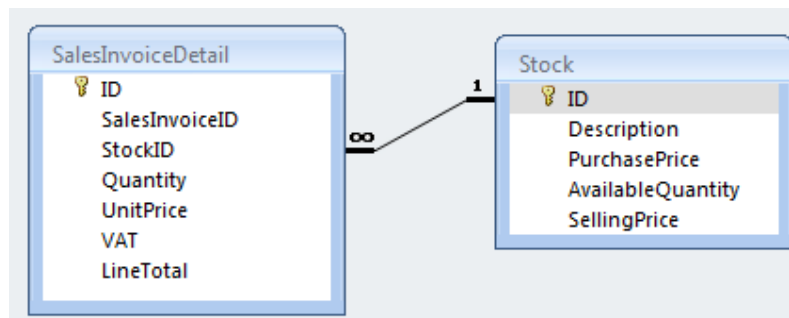
Stock					
	ID	Description	PurchasePri	AvailableQu	SellingPrice
+	1	VB.NET 2005 For Dummies	R 250.00	20	R 300.00
+	2	Exam Pad	R 5.00	500	R 10.00
+	3	Diary	R 25.00	75	R 75.00
+	4	Database Concepts	R 150.00	10	R 250.00
+	5	Beginning C# Programming	R 100.00	8	R 120.00
*	(New)		R 0.00	0	R 0.00

SalesInvoiceDetail					
	ID	SalesInvoiceID	StockID	Quantity	UnitPrice
	1	1	2	2	R 10.00
	2	1	1	1	R 300.00
	3	2	5	1	R 120.00
*	(New)	0	0	0	R 0.00

If you look at the above image, it becomes clear that the second item of **SalesInvoiceDetail** is a copy of **VB.NET 2005 For Dummies** because **StockID's** key (Foreign Key) is the same as the **Stock** table's **ID** key (Primary Key). We can also see that the third item purchased was **Beginning C# Programming** by using the same logic.

Entity Relationship Diagrams provide a visual representation of these links as can be seen by the fragment displayed earlier:



When we wish to obtain information, such as what items were purchased by looking at the **SalesInvoiceDetail** table, we must first **join** the tables together when we write our SQL query; we do this by telling the DBMS what fields are related between the two tables.

Consider the following SQL Query:

```

SELECT description
FROM stock, salesinvoicedetail
WHERE stock.id = salesinvoicedetail.stockid
  
```

In the above query we **SELECT** the *Description* column **FROM** the *Stock* table **WHERE** the **Primary Key** (*stock.id*) is equal to the **Foreign Key** in the *SalesInvoiceDetail* table (*salesinvoicedetail.stockid*). In short, we basically tell the DBMS to show us what product(s) were purchased and placed on the Sales Invoice.

The Query should produce the following result:

Query1
description
VB.NET 2005 For Dummies
Exam Pad
Beginning C# Programming

It turned out exactly as we predicted earlier when we looked at the two tables and their links.

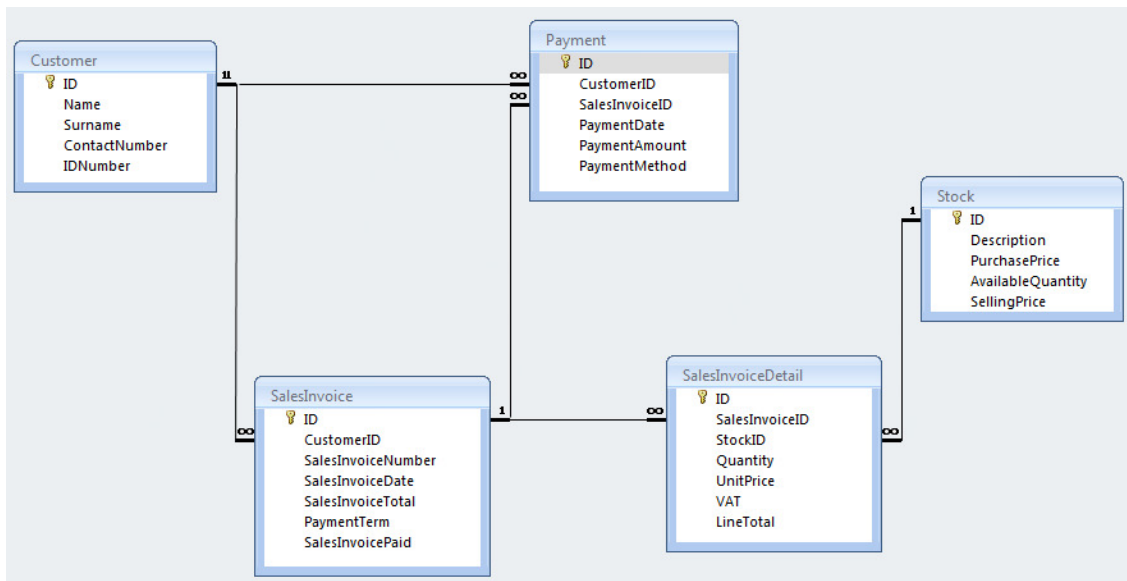
You can use this method to link multiple tables together, providing you the ability to display information in any way you wish. There is a simple rule to remember when joining multiple tables, if you do not follow this rule you will end up with multiple or erroneous results known as **Cartesian Products** where a distinct row of data is displayed more than once.

The rule can be remembered as follow:

**The Amount of Joins Are Equal to The Amount of Tables Being Used, Minus One.**

So, if you are joining 6 tables, you will have 5 clauses (joins) in your WHERE section and any optional or additional statements.

As a final example, we will use the POSDB database to determine what customers bought what items, first we must find out what links we will require to do this:



From the ERD we can see that we will need the following tables, Customer, SalesInvoice, SalesInvoiceDetail and Stock. Thus, we have four distinct tables and we will need at least three joins to prevent Cartesian Products.

We can link Customer to SalesInvoice via the CustomerID foreign key, we can link SalesInvoice to SalesInvoiceDetail via SalesInvoiceID and finally we can link SalesInvoiceDetail to Stock via StockID.

Our Query should look something like this, if we wish to display the Name, Surname and Description fields from these tables:

```

SELECT name, surname, description
FROM customer, salesinvoice, salesinvoicedetail, stock
WHERE customer.ID = salesinvoice.customerid
AND salesinvoice.ID = salesinvoicedetail.salesinvoiceid
AND stock.id = salesinvoicedetail.stockid

```

In the above query we requested the DBMS to SELECT the Name, Surname and Description columns FROM the tables Customer and Stock. We also included the tables SalesInvoice and SalesInvoiceDetail needed to join the related data.

It is evident from the WHERE clauses that we have used at least three joins by telling the DBMS what tables shared what key's  
The results should be as follows:

Query1		
name	surname	description
Ronald	Alexander	Exam Pad
Ronald	Alexander	VB.NET 2005 For Dummies
Henry	Doyle	Beginning C# Programming

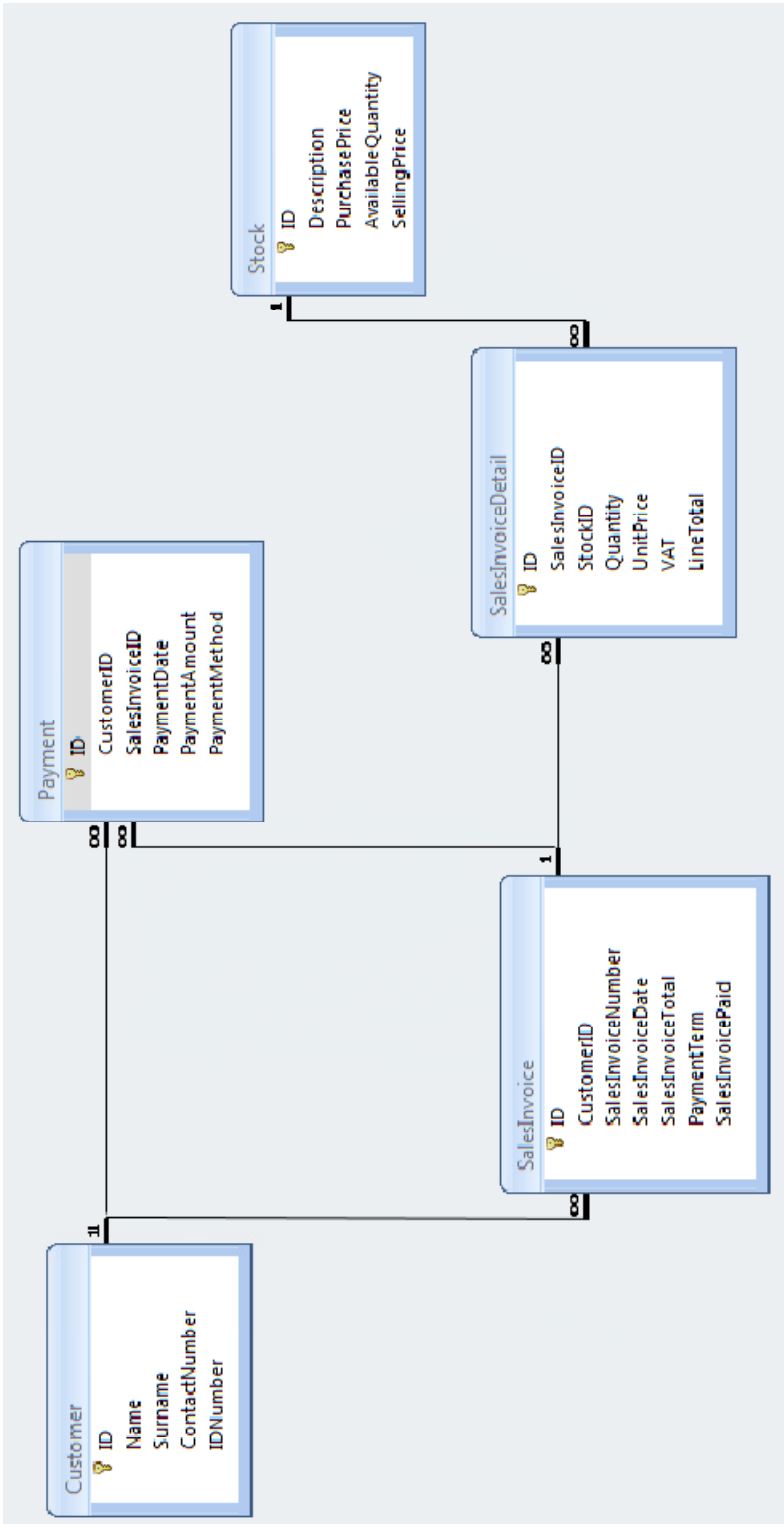
As an aside, the query may also be written in a shorter fashion by using column **aliases**:

```

SELECT name, surname, description
FROM customer c, salesinvoice si, salesinvoicedetail sid, stock s
WHERE c.ID = si.customerid
AND si.ID = sid.salesinvoiceid
AND s.id = sid.stockid

```

POSDB Table Relationships



## Appendix B

### Application: Custom Query Tool in VB .NET

Notes compiled by: [Leandré Roux](#)

#### Introduction

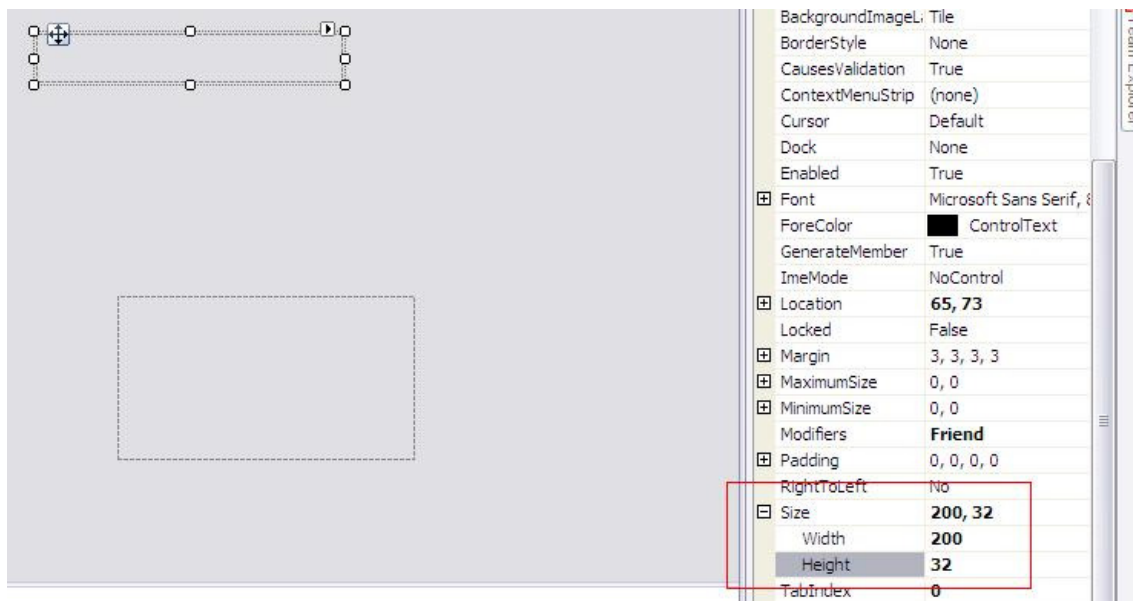
Executing queries in Microsoft Access gives us an environment in which we may grow familiar with the Structured Query Language. In this section we will delve into techniques for developing a custom tool for executing queries and displaying the results.

In the programming industry, it is necessary for developers to be able to interact with databases in the software packages they develop; most software requires a means of saving data to persistent storage and this is usually accompanied by the need to read and write data to a database.

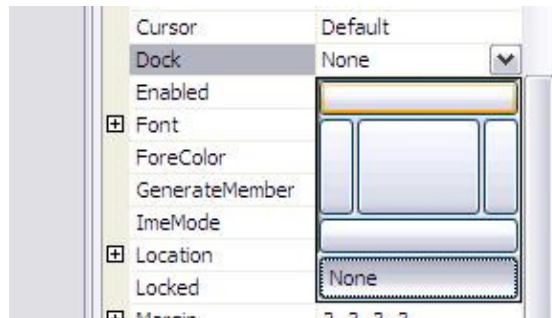
In this tutorial, we will design very basic software that will allow you to select a database, perform a query and view the results of the query. This will be a step-by-step tutorial, and you are encouraged to recreate the application as it is explained to help you better understand the mechanics of database access.

#### Creating The Graphical User Interface

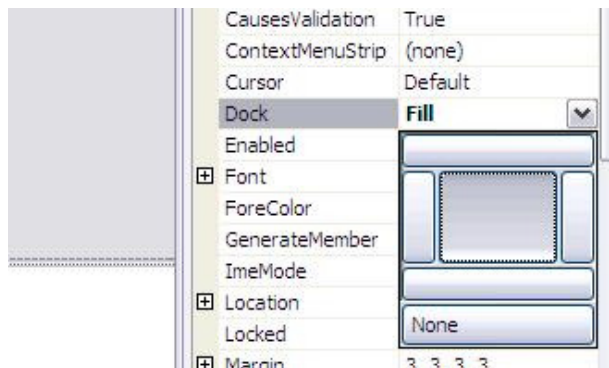
1. Start in Visual Studio 2008 by creating a new Project, Select Windows Forms Application and name it **CustomQueryTool**.
2. Resize the Form to a suitable size and add 2 **Panel** controls from the **Containers** group in the Toolbar.
3. Set the **Height** property of one of the Panel controls to **32**.



4. **Dock** this Panel to the **top** of the Form.

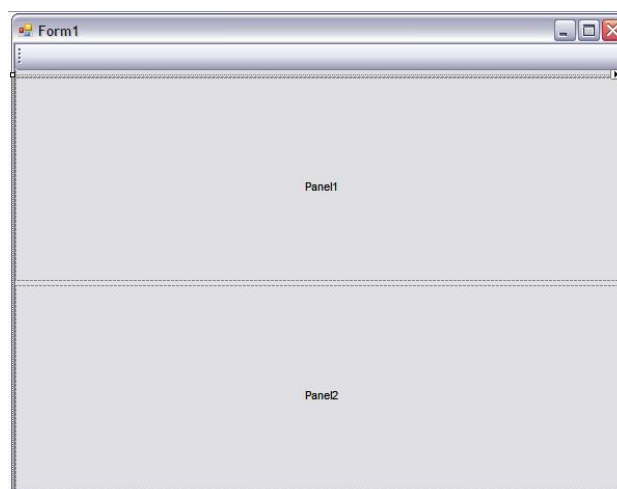


5. Rename this Panel to **pnlToolBarSpace**.
6. Dock the remaining Panel by selecting the **Fill** option in the Dock property and rename it to **pnlSplitSpace**.



7. Add a **ToolStrip** from the Menus & Toolbars group to **pnlToolBarSpace**, this will act as our menu from which we will execute our database functions. Rename this ToolStrip control as **tspMenu**.
8. Add a **SplitContainer** to **pnlSplitSpace** and set its **Orientation** Property to **Horizontal**. Rename it to **splQueryEditor**.

Your Form should look similar to the following illustration:



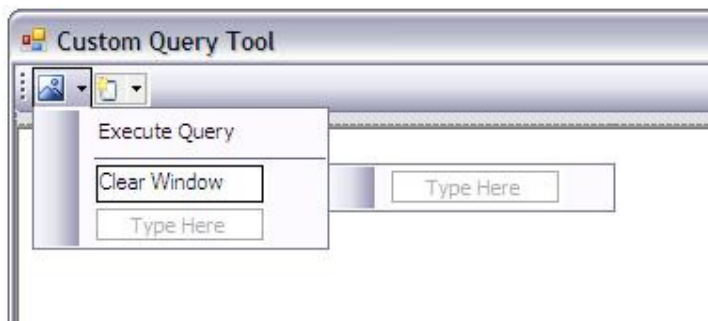


9. Select Form1 and rename it to **CustomQueryTool**. Set the Text property to Custom Query Tool.
10. Add a **RichTextBox** control to Panel1 of **sqlQueryEditor** and rename it to **txtEditor**. Set the Dock property of txtEditor to **Fill**.
11. Add a **DataGridView** control from the **Data** group to Panel2 of **sqlQueryEditor**. Rename it to **dgvResult** and set its Dock property to **Fill**.



The RichTextBox will act as our query input field, here we will type our SQL queries and execute them from our menu. The DataGridView will display the results of the query after it is executed. We will add the needed buttons and other controls to our menu in the next few steps.

1. Select **tspMenu** and click on the drop down control that appears, select **SplitButton** from the list.
2. Click on the control that says 'Type Here' and type **Execute Query**, then click on the next control that displays 'Type Here' and type a single – (minus), this will create a separator, and finally on the next menu item type **Clear Window**. You should now have a menu that is similar to the illustration below.



3. While the menu item is still selected locate the **DisplayStyle** property and set it to **Text**. Then change the Text property to 'Options'.
4. Add a **Seperator** and then a **Button** control to **tspMenu**. Set the Button's **DisplayStyle** to Text, then change the Text property to 'Change Database'

## Coding The Functionality

We will go through the functional code for the application in the following steps.

1. Select the Options menu item and Double Click on Execute Query. This will automatically create the Action Event for this menu item.

ExecuteQueryToolStripMenuItem\_Click functional code:

```
If txtEditor.Text <> String.Empty Then
    If txtEditor.SelectedText <> String.Empty Then
        ExecuteQuery(txtEditor.SelectedText)
    Else
        ExecuteQuery(txtEditor.Text)
    End If
End If
```

In the code fragment depicted above we perform the following operations, from top to bottom.

- In the first line we check to see if the txtEditor control is not empty, we do this because we want to ensure that we do not accidentally execute an empty query.
- Next we check if any text is highlighted or selected by the user, this will allow us to perform a feature that only executes the selected text of a query. This is useful if you want to type out multiple queries and only execute the selected one or only part of a query.
- Otherwise we just execute whatever is in the query window.

We will need the following Import and variable declarations before proceeding to the next step:

```
Imports System.Data.OleDb
Imports System.Data

Friend WithEvents Connection As OleDbConnection
Friend WithEvents DataAdapter As OleDbDataAdapter
Friend WithEvents ResultSet As DataSet
Friend WithEvents ConnectionString As String
Friend WithEvents SelectedDatabasePath As String
```

The above imports are needed to interact with the Access OLEDB and provide the functionality we need to access and manipulate data within these databases.

The first import, `System.Data.OleDb`, provides us with OLEDB functionality and associated classes that allow us to communicate with the database driver, the import `System.Data` allows us to use generic database functions and data containers like the `DataSet`.

We declare the following variables:

- `Connection As OleDbConnection`, we will use this object for creating and maintaining a connection to our database.
- `DataAdapter As OleDbDataAdapter`, we will use this object for performing queries and manipulating the returned data before we display it.
- `ResultSet As DataSet`, we will use this object as a container of the returned data, the data is stored within table-like structures, mimicking the database.
- `ConnectionString As String`, this object will act as the command for connecting to the database, it holds values such as the driver we want to use, as well as the database location.
- `SelectedDatabasePath As String`, this object will hold the string representation of the fully-qualified database path.

Next we create a custom function block for executing our queries, this block will execute queries based on the data passed to it via the query parameter.

```

Private Sub ExecuteQuery(ByVal query As String)
    Try
        If ConnectionString <> String.Empty Then
            DataAdapter = New OleDbDataAdapter(query, Connection)
            ResultSet = New DataSet()

            If ResultSet.Tables.Count > 0 Then
                ResultSet.Clear()
            End If

            DataAdapter.Fill(ResultSet, "Results")
            dgvResult.DataMember = "Results"
            dgvResult.DataSource = ResultSet
        Else
            MsgBox("No Database has been selected, first select a
                database", MsgBoxStyle.OkOnly, "No Database")
        End If
    Catch ex As Exception
        MsgBox(String.Format("An Exception has occurred : {0}", ex.Message()))
    End Try
End Sub

```

We create a `Private` function called `ExecuteQuery` that receives one parameter by value named `query` of the `String` data type. The `query` parameter is received from the functional call during the `ExecuteQueryToolStripMenuItem_Click` event and holds the string representation of the query we wish to issue to the DBMS.

The next code block is placed between a `Try-Catch` block, this is the best approach when designing an application that uses functionality that may fail during execution for reasons outside your control, using this approach allows you to recover from a serious error and to continue program flow. In our case we choose to just inform the end-user of the problem by printing a message to a simple implementation of `MessageBox`.

The first `If` statement is used to first check if a connection has been made before attempting to execute any of the SQL queries. Next we initialize our `OleDbDataAdapter` and pass it the query we wish it to execute as well as the connection we wish it to use. This method automatically opens the connection, performs the query and closes the connection when done. Consider the following fragment:

```

If ResultSet.Tables.Count > 0 Then
    ResultSet.Clear()
End If

DataAdapter.Fill(ResultSet, "Results")
dgvResult.DataMember = "Results"
dgvResult.DataSource = ResultSet

```

We initialize the `DataSet` and check if it has any tables already associated to it, usually this will be true if it is the second time the application calls this function. A `Fill` command is issued to the `DataAdapter` and all data received from the SQL execution is placed within the `DataSet` object and the virtual table is given a name, in this case "Results".

We associate this virtual table as a `DataMember` of our `DataGridView` and set the `DataSource` of the `DataGridView` to the `DataSet` that has just been filled. This tells our `DataGridView` to load data from our `DataSet` and to load it from its associated `DataMember`.

```

Else
    MsgBox("No Database has been selected, first select a
        database", MsgBoxStyle.OkOnly, "No Database")
End If
Catch ex As Exception
    MsgBox(String.Format("An Exception has occurred : {0}", ex.Message()))
End Try

```

Finally if the connection has not been set up, we notify the user by using a `MessageBox` to display the notification. As noted earlier we catch all exceptions and simply display them to the end-user. Note the use of `String.Format`, it is better programming practice to use `String.Format` for concatenating strings than using the `+` (plus).

To display the string “The employee’s name is James, he is a consultant”, you can use the function like this:

```
String.Format("The employee's name is {0}, he is a {1}", name, position)
```

Where `name` and `position` are variables, we may add additional parameters as needed, given you provide an index for each in the string starting with 0 for the first listed variable.

For the `ToolStripButton1_Click` event, add an **OpenFileDialog** from the **Dialogs** group to your form, this will allow us to browse for the database we wish to use:



Rename the added control by selecting it from the menu at the bottom of the designer to **dlgLoadDatabase** and set the **Filter** property to **Access 2003 Database|\*.mdb**. Set the **Title** property to something suitable like **Open Access Database**.

The following code fragment forms the body of `ToolStripButton1_Click`

```
dlgLoadDatabase.CheckFileExists = True
dlgLoadDatabase.ShowDialog()

If dlgLoadDatabase.FileName <> String.Empty Then
    SelectedDatabasePath = dlgLoadDatabase.FileName
    ConnectionString = String.Format("Provider=Microsoft.Jet.OLEDB.4.0;Data
                                   Source={0};", SelectedDatabasePath)
    Connection = New OleDbConnection(ConnectionString)
End If
```

Start off by setting the `CheckFileExists` property of **dlgLoadDatabase** to **true**, this will ensure that only legal files can be selected in the dialog, next tell the `OpenFileDialog` to display to the end-user and request him or her to select a file to load.

In the next step we check to see if any file has been selected, if so, set this to the database path. The connection string is used to provide the needed information about the database we wish to use, in this instance we set the following properties:

- The `Provider`, which is the driver we wish to use, in this case **Microsoft.Jet.OLEDB.4.0**. This is the Microsoft Jet driver that allows us to connect to OLEDB compliant data sources.
- The `Data Source`, which is a URL path to a database or database server.

Finally we set the connection object by initializing it and providing the connection string as a parameter. This will now create a connection to the database we selected with the dialog and use the setting we provided in the connection string.

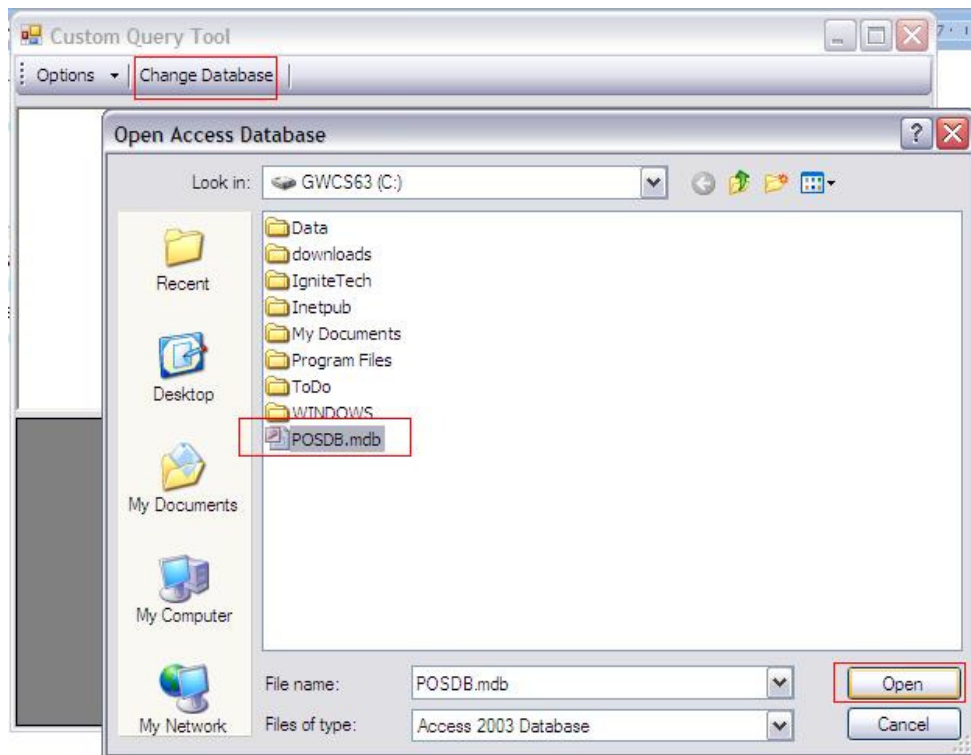
The `ClearWindowToolStripMenuItem_Click` event will be used to clear the input window, and has the simplest code:

```
txtEditor.Clear()
```

As it may be guessed, this function clears all data in the RichTextBox control.

## Using The Application

In the following section, we will briefly discuss how to use your basic SQL query execution tool, the functionality is limited and you are encouraged to add additional functionality as you see fit. With the application running, click on the '**Change Database**' button on the **Toolbar**, the `OpenFileDialog` should appear, locate the **POSDB.mdb** file on your hard drive and click on the **Open** button to load the database.



After the database has been selected, start by typing a query in the text editor, then click on the arrow next to the **Options** button and select '**Execute Query**', you should see the result of your queries in the data grid below the text editor. Note that only `SELECT` statements will provide visual clues that the query succeeded, if you execute an `UPDATE`, `INSERT` or `DELETE`, you will have to perform a `SELECT` to see the changes.

Here is an example of a basic `SELECT` statement performed on the Stock table:

Custom Query Tool

Options | Change Database

```
SELECT *
FROM Stock
```

ID	Description	PurchasePrice	AvailableQuantity	SellingPrice
1	VB.NET 2005 Fo...	250	20	300
2	Exam Pad	5	540	10
3	Diary	25	75	75
4	Database Conce...	150	10	250
5	Beginning C# Pro...	100	8	160
7	Easy SQL Vol.1	150	5	250
*				

Write the following query just below the SELECT statement:

```
INSERT INTO Stock(Description, PurchasePrice, AvailableQuantity, SellingPrice)
VALUES('SQL Coding', 120, 20, 200)
```

Custom Query Tool

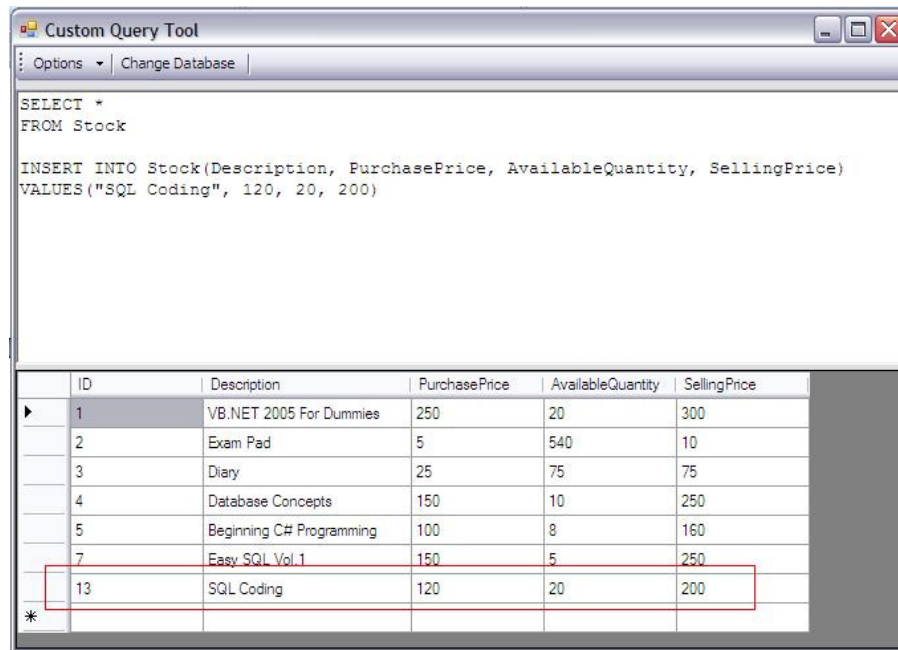
Options | Change Database

```
SELECT *
FROM Stock

INSERT INTO Stock(Description, PurchasePrice, AvailableQuantity, SellingPrice)
VALUES('SQL Coding', 120, 20, 200)
```

ID	Description	PurchasePrice	AvailableQuantity	SellingPrice
1	VB.NET 2005 Fo...	250	20	300
2	Exam Pad	5	540	10
3	Diary	25	75	75
4	Database Conce...	150	10	250
5	Beginning C# Pro...	100	8	160
7	Easy SQL Vol.1	150	5	250
*				

Highlight the new query, and execute it. Next, highlight the SELECT statement and execute only it. You should now be able to see the inserted data.



If you wish to change the current database you are performing the queries upon, simply click on Change Database and load the new database.

## Appendix C

### Multi Document Interface – Encryption/Decryption

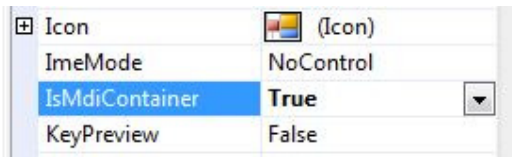
Notes compiled by: [Leandré Roux](#)

#### Introduction

In this step-by-step guide, we will be looking at a Multi Document Interface application that will have the functionality to encrypt or decrypt text and provide functionality such as save and load. The encryption algorithm used in this example is unsuitable for real-life use and is simply used to illustrate the concept of encryption.

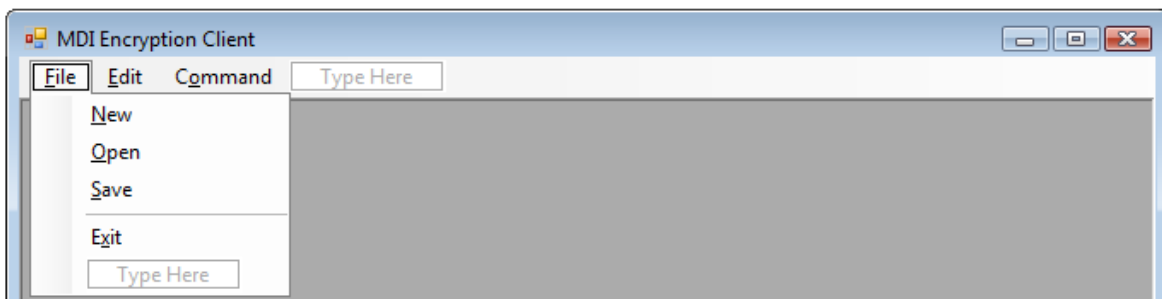
#### The Application

In Visual Studio, create a new Windows Forms application and name it **MDIEncryptionClient**. Select Form1 and set its **IsMdiContainer** to **True**. This will change Form1 to a MDI parent container which allows us to open other forms inside it as **child** forms. Child forms cannot leave the parent form which acts as a Desktop Environment for them. In the past, MDI applications were the most widely used format for applications with multiple windows open at the same time, you can still see this in Microsoft Access.



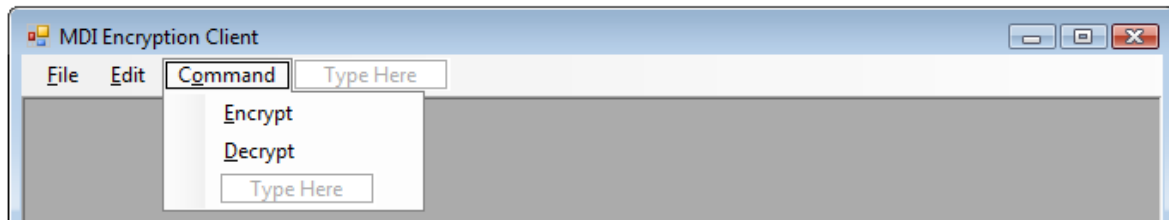
After setting the form as a MDI parent, add a **MenuStrip** to the Form from the **Menus & Toolbars** group, this is where we will place all our functionality for the application.

Create a File menu that contains the following menu items:



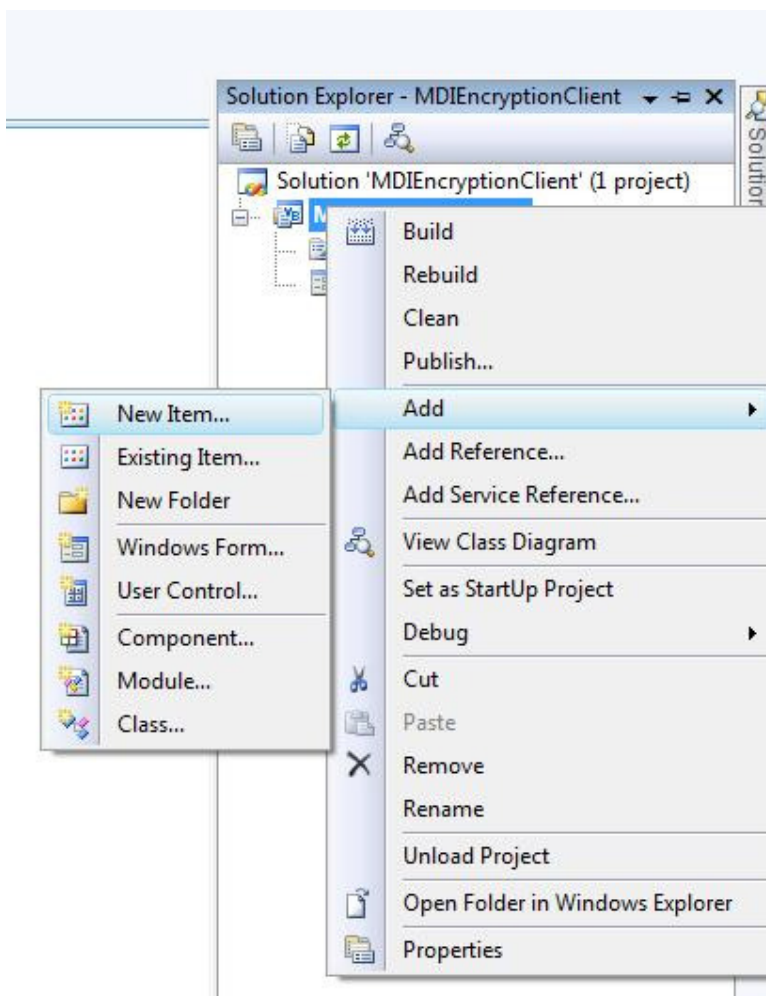
Next, create an **Edit** command, which will be left as an exercise. And finally add a **Command** menu item that has two menu items, **Encrypt** and **Decrypt**.



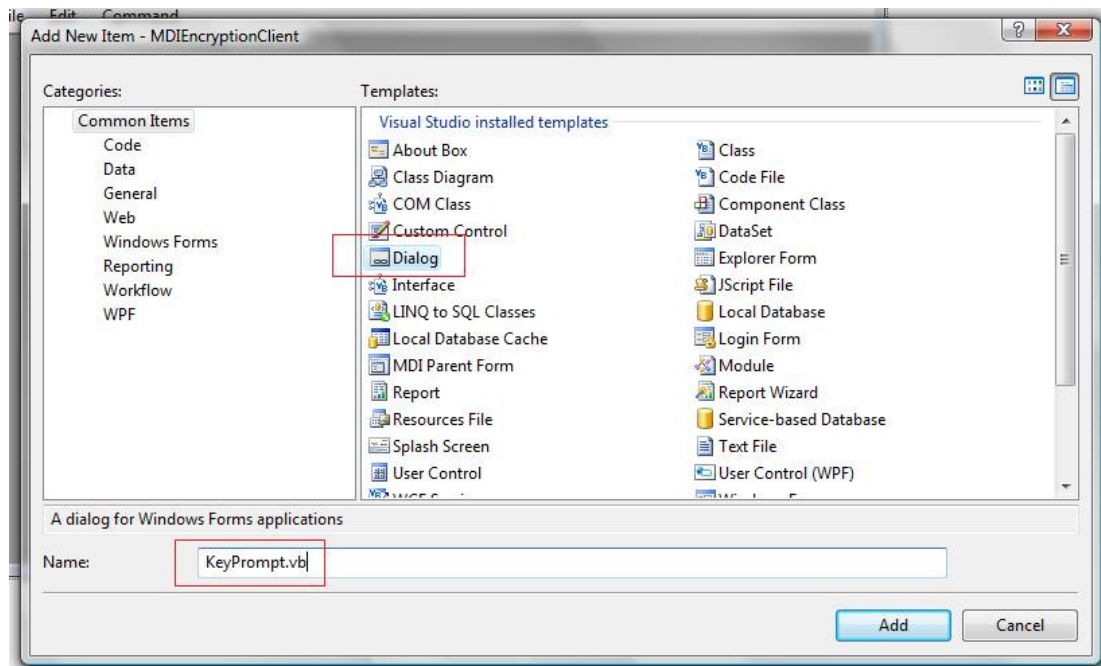


Next, we will create the prompt for the encryption password, for this we will use a Dialog. A Dialog is the better choice here, because it forces the user to first attend to it before allowing any further action in the application, in this case we want the user to first enter the password before we begin the encryption process.

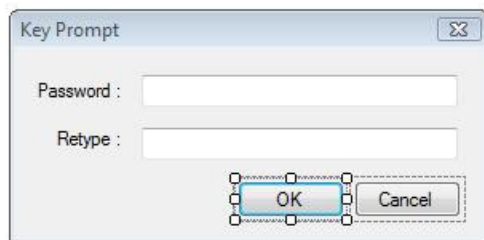
To add a Dialog, right click on your Project file in the **Solution Explorer**, navigate to Add, then New Item as demonstrated in the image below.



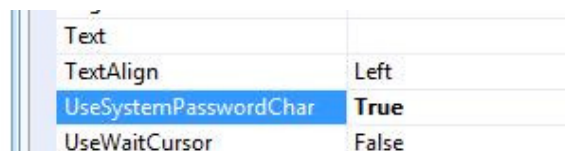
A new window should now open allowing you to choose what items you wish to add to the project, click on the **Dialog** item and rename it to **KeyPrompt.vb**



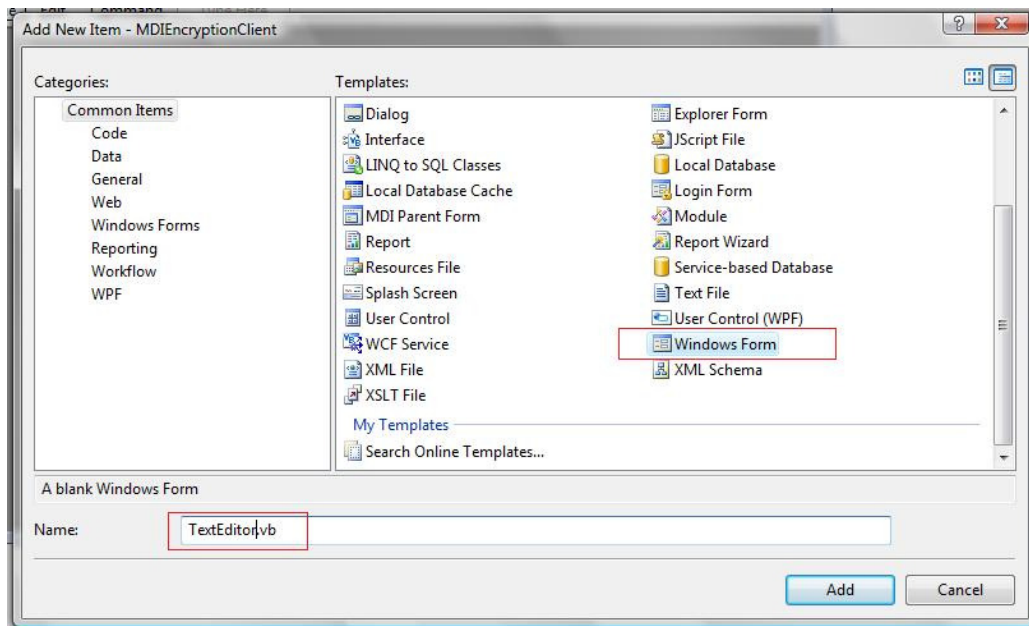
Resize the dialog and add two **Labels** and two **TextBox** control to it, the dialog already has two buttons placed on it by the Visual Studio Designer. Change the **Text** properties of both labels so that the first displays **Password** and the second **Retype**. We will mask the user input, so it is necessary to have the user retype his or her password to ensure that it is correct.



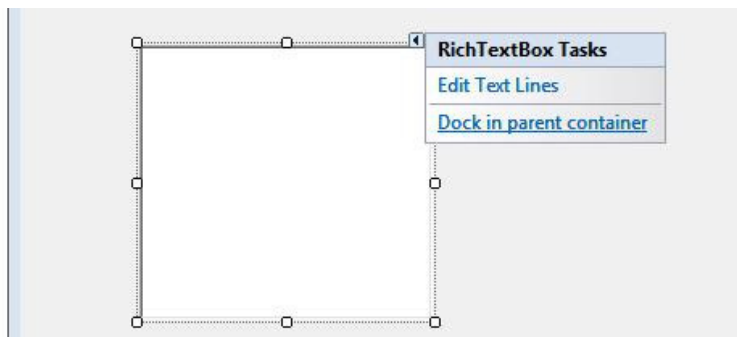
Select the first Textbox control and change its Name property to **txtPassword** and then the second Textbox control to **txtRetype**. Select both Textbox controls and in the Properties window set the **UseSystemPasswordCharacter** to **True**. This will mask the password using the system's default mask.



Finally, we will add our **TextEditor** control, for this you need to add a **Form** item to your project. Again, right-click on your project in **Solution Explorer**, navigate to **Add** then **New Item** and select **Form** from the items, name it **TextEditor.vb** and click on **OK**.



From the Common Controls add a RichTextbox control to the Form and set its Dock property to Fill. In the image below, you may optionally click on the link 'Dock in parent container', or you may set it in the properties of the RichTextbox control.



This will create a form that consists of a title bar and a text window. Change the Name property of the RichTextbox control to **txtInput**.

Finally select the **KeyPrompt** designer and change the **Enabled** property of OK\_button to **False**. We do this to ensure that the password is only used if it was typed in correctly, otherwise the file will not be able to be decrypted.

## The Source Code

We will start by adding the functionality to the KeyPrompt form first, this is where the password processing will take place, so we want to focus on completing this part first. Select KeyPrompt and select View Code.

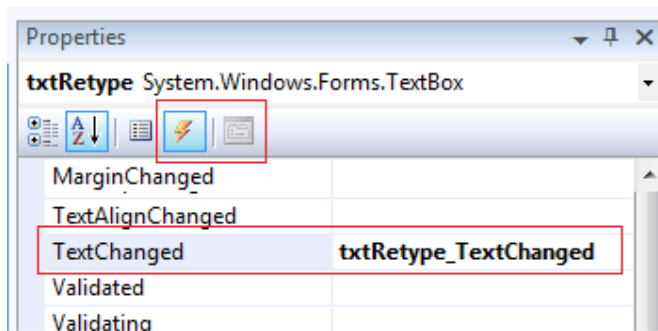
Add the following global variables:

```
Dim passwordShift As Integer
Dim charArray As Char()
```

For our encryption algorithm we will use a simple character shift operation by which we will take all the numerical values of each character in the password provided by the user and add them together to form the shift we will use to encrypt or decrypt the text. The variable passwordShift will be used to store the value by which to shift each character we wish to encrypt or decrypt.

charArray will be our character array that will act as a temporary holder for manipulating the input strings.

Add a **TextChanged** event listener to the txtRetype control on this form as illustrated below. You can simply double-click in the area next to **TextChanged** to generate the event listener.



Add the following code into the listener:

```
'This function checks if the password matches
If txtRetype.Text.Equals(txtPassword.Text) Then
    OK_Button.Enabled = True
End If
```

The code above will check the text being typed into the textbox and if it matches the text of the txtPassword Textbox then it will enable the OK button on the dialog. This function ensures that the user has typed his password correctly.

Next, we will write the code that will return the character shift to the calling agent, for this we wrote a custom Function called GetPasswordShifter(). The code follows:

```
Function GetPasswordShifter() As Integer
    'This function creates a password 'shift' value by taking each
    character of the
    'user provided password, finding the numerical value and adding
them.
    charArray = New Char(txtPassword.TextLength) {}
    txtPassword.Text.CopyTo(0, charArray, 0, txtPassword.TextLength)

    For index As Integer = 0 To charArray.Length - 1
        passwordShift += Convert.ToInt32(charArray(index))
    Next

    Return passwordShift
End Function
```

We start by creating a character array with the same length as our password and we initialize it with zero items. Next we use the String function CopyTo on the Text of the password input field, this copies each character into our character array, starting at index zero, we also specify that it must start copying from the first character in the Textbox ( 0 ) and stop when it has copied all the characters by specifying the length to be the same as the length of the input text.

Next we you a For loop to add all the character's numerical values together and store it in our variable passwordShift, which is our object returned to the calling environment.

This is all the code needed for the KeyPrompt Dialog, next we will look at our main application, open Form1 and select View Code.

Start by declaring our global variables:

```
Dim popup As KeyPrompt
Dim textEditor As TextEditor
Dim passwordShift As Integer
Dim textChars As Char()
Dim encryptedDecryptedText As String
```

We create variables for our Dialog and TextEditor so that we can manipulate them from within our code, we also create temporary holders for the passwordShift as well as a character array for manipulating the text from our TextEditor.

We create a String variable encryptedDecryptedText that will hold a copy of the text after it has been processed.

Let's start off by writing the functions we will need to manipulate the text in TextEditor:

```
Function EncryptText(ByVal text As String, ByVal shift As Integer) As String
    If Not textEditor Is Nothing Then

        'Copy the String values into a new character array
        textChars = New
            Char(Me.ActiveMdiChild.Controls("txtInput").Text.Length- 1){}

        Me.ActiveMdiChild.Controls("txtInput").Text.CopyTo(0,textChars, 0,
        Me.ActiveMdiChild.Controls("txtInput").Text.Length)

        'Get each character in the texfield and add the shift numerical
        value to it
        For index As Integer = 0 To textChars.Length - 1
            textChars(index) =
                Convert.ToChar(Convert.ToInt32(textChars(index)) + shift)
        Next

        'Create a new string from the shifted characters
        encryptedDecryptedText = New String(textChars)

    End If
    Return encryptedDecryptedText
End Function
```

In the above code fragment, we created a Function named EncryptText that takes the text to be encrypted as well as the shift to apply to each character as parameters, it returns the encrypted text as a String.

We first check to see if there is at least one textEditor object before attempting to execute the function, we do this by checking if an object exists. Next, we create a new array with the same length as the text present in the textEditor controls txtInput.

The following code used above simply performs whatever we want on the active MDI child's txtInput control; this is needed if we have multiple windows open at a time so we perform actions only on the selected child window:

```
Me.ActiveMdiChild.Controls("txtInput")
```

Then we copy the text into the array before we use a For loop to increment each character at a time with the numerical shift value. We do this by determining the current characters numerical value and simply adding the shift to it. Finally, we create a new String object and pass the character array as a parameter, this is then returned to the calling function.

Decrypting is performed in exactly the same manner, and therefore only the code listing will be displayed below. The only difference is the shift is subtracted from each character providing us with the original value.

```
Function DecryptText(ByVal text As String, ByVal shift As Integer) As
String
    If Not textEditor Is Nothing Then
        textChars = New
            Char(Me.ActiveMdiChild.Controls("txtInput").Text.Length - 1){}
        Me.ActiveMdiChild.Controls("txtInput").Text.CopyTo(0, textChars,0,
        Me.ActiveMdiChild.Controls("txtInput").Text.Length)

        'Get each character in the texfield and subtract the shift
numerical
        value from it,
        'this will give the original text
        For index As Integer = 0 To textChars.Length - 1
            textChars(index) =
                Convert.ToChar(Convert.ToInt32(textChars(index)) - shift)
        Next

        encryptedDecryptedText = New String(textChars)

    End If
    Return encryptedDecryptedText
End Function
```

We create a small function for initializing and displaying new windows:

```
Private Sub CreateNewTextWindow()
    'Creates a new MDI child form
    textEditor = New TextEditor
    'Set the parent of the MDI child form
    textEditor.MdiParent = Me
    'Display the form
    textEditor.Show()
End Sub
```

As seen above, it is necessary to set the parent control for the TextEditor instances, you do this by setting the MdiParent property.

Create event listeners for all the menu items by simply double clicking each entry. Add the following code to the Encrypt menu item's listener:

```
'Create the popup and display it as a dialog
popup = New KeyPrompt
popup.ShowDialog()

'Check the button the user has clicked on the dialog
If (popup.DialogResult = Windows.Forms.DialogResult.OK) Then
    'Call the Function from the popup to retrieve the character increment
    passwordShift = popup.GetPasswordShifter()
End If

'Ensure the TextEditor control exists before executing the code
If Not textEditor Is Nothing Then
    'Set the text of the currently focused MDI child to the returned string
value received
    'from the EncryptText Function
    Me.ActiveMdiChild.Controls("txtInput").Text =
```

```

        EncryptText(Me.ActiveMdiChild.Controls("txtInput").Text,
passwordShift)
    End If

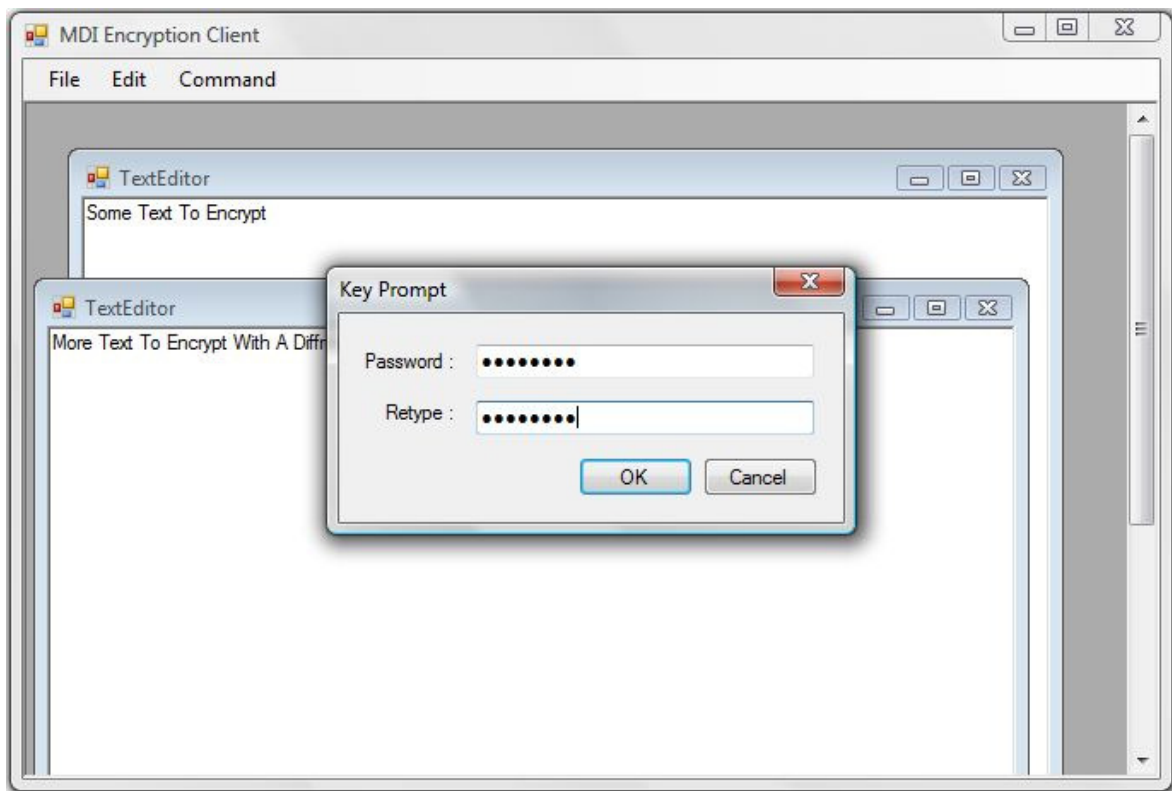
```

When the user clicks on the Encrypt menu, a new instance of KeyPrompt is created and is displayed as a Dialog. The user then at this point enters his or her password and clicks on the OK button. Next, we check what button the user had clicked by checking the dialog result, if the user clicked on OK then we call the GetPasswordShifter() function and store the returned value in our global variable passwordShift.

Next we check to see that there is at least on textEditor object available, after which we set the active Mdi child's txtInput control's text to the returned value of the EncryptText function. We pass the active child's text as the text to encrypt and the shifter we have just received.

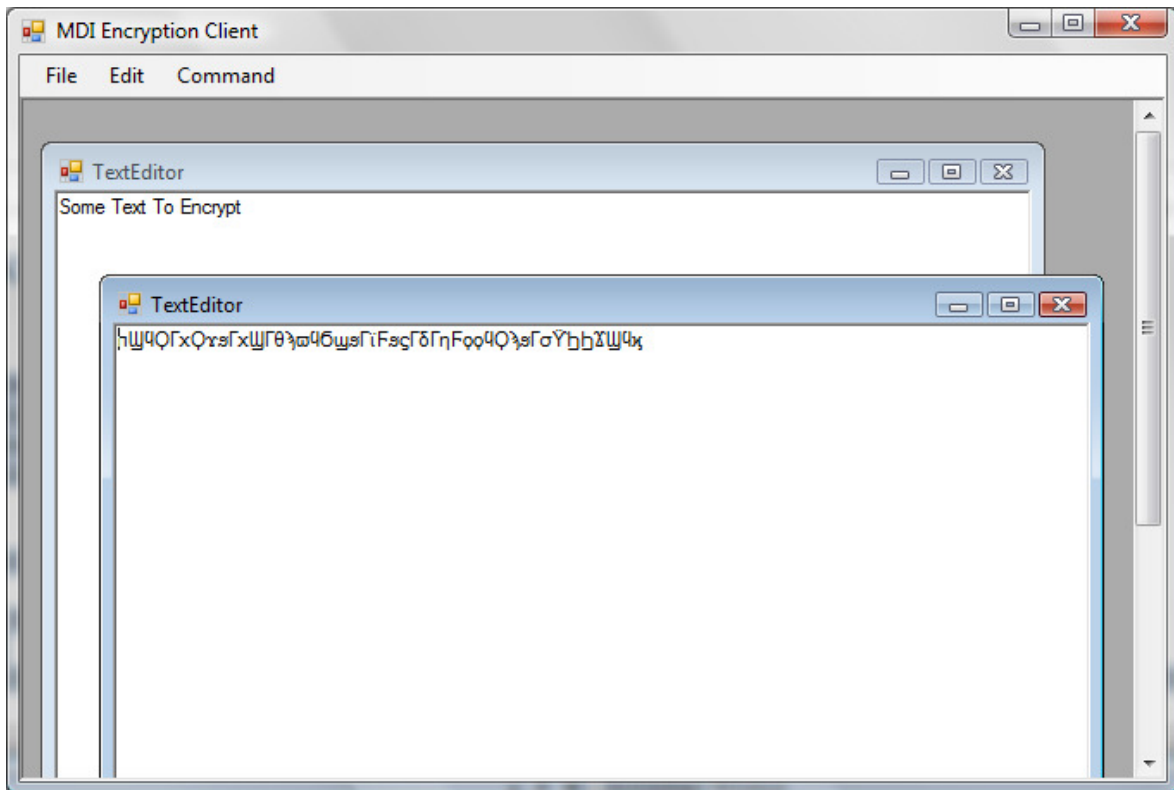
The following images show these steps graphically:

1. Provide the password to use for encrypting, click on OK.





2. The encrypted text in the active TextEditor window.



The Decrypt menu performs the same task, as such only the code listing is provided. The Decrypt function is called in this instance:

```
popup = New KeyPrompt
popup.ShowDialog()

If (popup.DialogResult = Windows.Forms.DialogResult.OK) Then
    passwordShift = popup.GetPasswordShifter()
End If

If Not textEditor Is Nothing Then
    Me.ActiveMdiChild.Controls("txtInput").Text =
        DecryptText(Me.ActiveMdiChild.Controls("txtInput").Text,
passwordShift)
End If
```

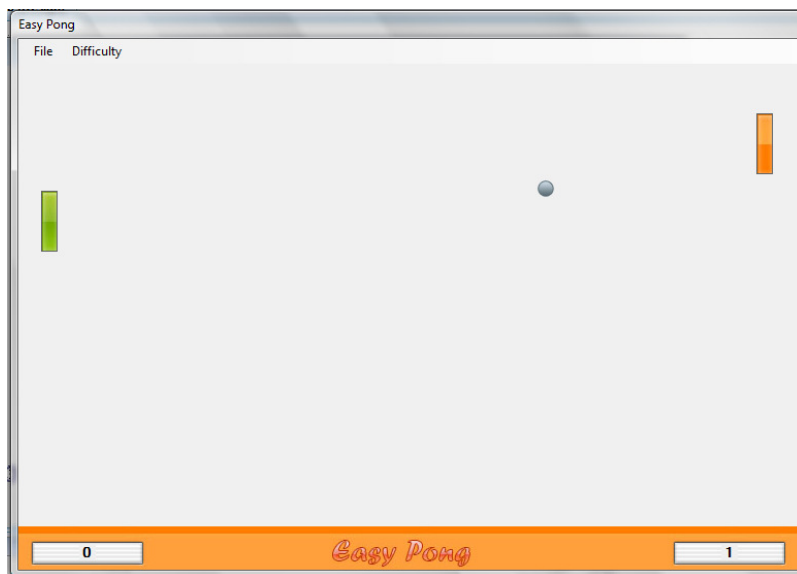
## Appendix D

### A Game - EasyPong

Compiled by: [Leandré Roux](#)

#### Introduction

EasyPong is a game similar to the original Pong, the game is similar to tennis. To play the game, click on the File menu and select New Game. The game is played by moving the mouse up and down to control the paddle and preventing the ball from crossing the line behind the player (right side). To stop the game, press the Escape key.



#### Game Design

EasyPong uses the standard controls shipped with Visual Studio 2008 for all the units used in the game. The game uses the Bounds class to check intersecting objects, in this case the paddle and ball objects and react accordingly. There are also individual checks for collisions with the top, bottom as well as the left and right bounds of the window.

The Computer as well as the ball objects is animated using individual Timer controls to simulate animation set at different tick intervals to give the human player a chance to win. The computer player is coded to always follow the ball, and as such needs to be on a separate, slower Timer instance. The MouseMove event is used to trigger the player paddle animation according to current mouse position.

Improvement of the game physics and AI is left to the student as a creative exercise that can build upon the already present functionality.

## Suggested solution

```
Public Class PongPlayField

    Private ballSpeed As Single = 10
    Private velocityXAxis As Single
    Private velocityYAxis As Single

    Private ComputerScore As Integer = 0
    Private PlayerScore As Integer = 0

    Private result As DialogResult

    Private Sub SetDifficulty(ByVal difficulty As Difficulty)
        Select Case difficulty
            Case PongPlayField.Difficulty.Easy
                Animator.Interval = 400
                velocityXAxis = Math.Cos(7) * ballSpeed
                velocityYAxis = Math.Sin(8) * ballSpeed
            Case PongPlayField.Difficulty.Hard
                Animator.Interval = 150
                velocityXAxis = Math.Cos(7) * ballSpeed
                velocityYAxis = Math.Sin(8) * ballSpeed
        End Select
    End Sub

    Private Sub NewGame()
        mnuGameMenu.Visible = False
        Windows.Forms.Cursor.Hide()
        ResetPlayingField()
        Animator.Start()
        BallAnimator.Start()
    End Sub

    Private Sub ResetPlayingField()
        PlayerScore = 0
        ComputerScore = 0

        Ball.Location = New Point(Me.Size.Width / 2, Me.Size.Height / 2)
        Computer.Location = New Point(Computer.Location.X, Me.Size.Height / 2)
        Player.Location = New Point(Player.Location.X, Me.Size.Height / 2)

    End Sub

    Private Sub StopGame()
        mnuGameMenu.Visible = True
        Windows.Forms.Cursor.Show()
        BallAnimator.Stop()
        Animator.Stop()
    End Sub

    Private Function GetPaddleY() As Integer
        Return Computer.Location.Y
    End Function

    Private Sub Animator_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Animator.Tick
        If Ball.Location.Y > 5 And Ball.Location.Y < (Me.Height - ScoreZone.Height) - 40 -
Player.Height Then
            Computer.Location = New Point(Computer.Location.X, Ball.Location.Y)
        End If
    End Sub

    Private Sub BallAnimator_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles BallAnimator.Tick
        Ball.Location = New Point(Ball.Location.X + velocityXAxis, Ball.Location.Y +
velocityYAxis)
        CheckWallCollition()
        CheckPaddleCollition()
        CheckOutOfPlayArea()
    End Sub

    Private Sub CheckWallCollition()
        If Ball.Location.Y < 0 Then
            Ball.Location = New Point(Ball.Location.X, 0)
        End If
    End Sub
```

```

        velocityYAxis = -velocityYAxis
    End If

    If Ball.Location.Y > (Me.Height - ScoreZone.Height) - Ball.Size.Height - 45 Then
        Ball.Location = New Point(Ball.Location.X, (Me.Height - ScoreZone.Height) -
Ball.Size.Height - 45)
        velocityYAxis = -velocityYAxis
    End If
End Sub

Private Sub CheckPaddleCollision()
    If Ball.Bounds.IntersectsWith(Player.Bounds) Then
        Ball.Location = New Point(Player.Location.X - Ball.Size.Width, _
Ball.Location.Y)
        velocityXAxis = -velocityXAxis
    End If

    If Ball.Bounds.IntersectsWith(Computer.Bounds) Then
        Ball.Location = New Point(Computer.Location.X + Computer.Size.Width + 1, _
Ball.Location.Y)
        velocityXAxis = -velocityXAxis
    End If
End Sub

Private Sub CheckOutOfPlayArea()
    If Ball.Location.X < 0 Then
        PlayerScore += 1
        lblPlayer.Text = PlayerScore.ToString()
        Ball.Location = New Point(Me.Size.Width / 2, Me.Size.Height / 2)
    End If

    If Ball.Location.X > Me.Width - Ball.Size.Width - Player.Width Then
        ComputerScore += 1
        lblComputer.Text = ComputerScore.ToString()
        Ball.Location = New Point(Me.Size.Width / 2, Me.Size.Height / 2)
    End If
End Sub

Private Sub PongPlayField_MouseMove(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseMove
    If e.Y > 5 And e.Y < (Me.Height - ScoreZone.Height) - 40 - Player.Height Then
        Player.Location = New Point(Player.Location.X, e.Y)
    End If
End Sub

Private Sub PongPlayField_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

End Sub

Private Sub PongPlayField_KeyDown(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles MyBase.KeyDown
    If e.KeyValue = Keys.Escape Then
        StopGame()
    End If
End Sub

Private Sub NewGameToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles NewGameToolStripMenuItem.Click
    NewGame()

    If (itmEasy.Checked) Then
        SetDifficulty(Difficulty.Easy)
    ElseIf (itmHard.Checked) Then
        SetDifficulty(Difficulty.Hard)
    End If
End Sub

Private Sub ExitToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ExitToolStripMenuItem.Click
    Me.Close()
End Sub

Private Sub EasyToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles itmEasy.Click
    itmHard.Checked = False

```

```

End Sub

Private Sub HardToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles itmHard.Click
    itmEasy.Checked = False
End Sub

Public Enum Difficulty
    Easy
    Hard
End Enum

Private Sub lblComputer_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles lblComputer.TextChanged
    If lblComputer.Text.Equals("5") Then
        StopGame()
        result = MessageBox.Show("You Lose... Play Again?", "Game Over",
MessageBoxButtons.YesNo, MessageBoxIcon.Information)

        If result = Windows.Forms.DialogResult.Yes Then
            ResetPlayingField()
        End If
    End If
End Sub

Private Sub lblPlayer_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles lblPlayer.TextChanged
    If lblPlayer.Text.Equals("5") Then
        StopGame()
        result = MessageBox.Show("You Win! Play Again?", "Game Over",
MessageBoxButtons.YesNo, MessageBoxIcon.Information)

        If result = Windows.Forms.DialogResult.Yes Then
            ResetPlayingField()
        End If
    End If
End Sub

Private Sub mnuGameMenu_ItemClicked(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.ToolStripItemClickedEventArgs) Handles mnuGameMenu.ItemClicked

End Sub
End Class

```