Year 12 : Visual Basic Tutorial.

STUDY THIS

Subroutines.

A **subroutine** is a small program that performs a specific task. It can be '**called**' from anywhere in a larger program. When the subroutine has been run, program execution returns to the larger program.

There are two main types of subroutine:

Procedures – that perform a specific task.

Functions – perform a task and return a **value**. Functions are frequently used for calculating something.

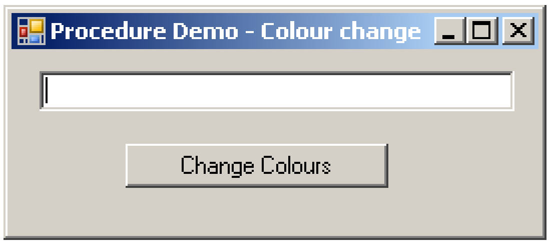
You have already met procedures because the event handlers are examples of procedures… but you can make your own. You are encouraged to do this because it creates a better structure to your program.

Bad programs have lots of repeated code. Good programs have lots of subroutines.

HANDS ON

[1] Create a new Windows Application.

On your form place a TextBox (TextBox1) and a Buttons (btnChange). Arrange them like this…



You are going to write a program using procedures, that toggles the colour schemes between two different schemes.

[2] You will need a variable to keep track of the current colour scheme, so make this declaration immediately after the Public Class Form1 line…

```
    Dim CurrentScheme As Integer = 1
```

Remember this means that we can use this variable in any subroutine on this form (Class).

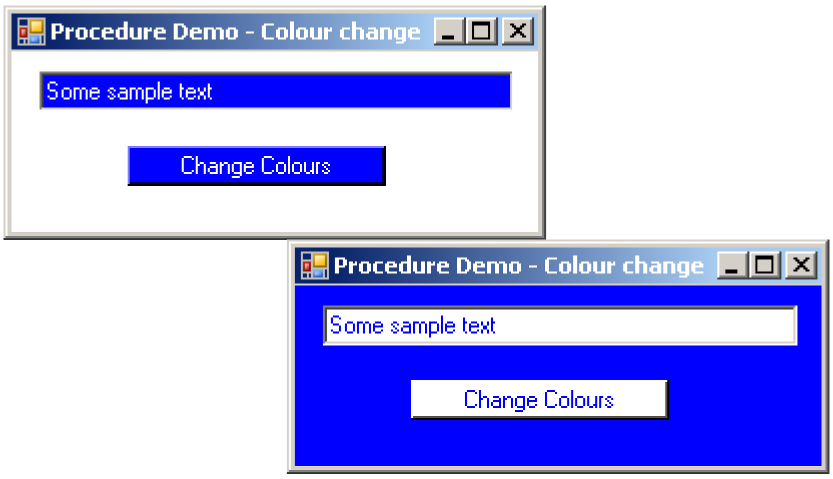
It is initialised to the value 1 when the program is run.

[3] The program consists of two subroutines (procedures). They are called SetColourScheme1 and SetColourScheme2. They are both called from the event handler btnChange_Click.

Type in the rest of the program as you see it here:

```
Public Class Form1

    Dim CurrentScheme As Integer = 1

    Private Sub SetColourScheme1()
        Me.BackColor = Color.Blue
        TextBox1.BackColor = Color.White
        TextBox1.ForeColor = Color.Blue
        btnChange.BackColor = Color.White
        btnChange.ForeColor = Color.Blue
        CurrentScheme = 1
    End Sub

    Private Sub SetColourScheme2()
        Me.BackColor = Color.White
        TextBox1.BackColor = Color.Blue
        TextBox1.ForeColor = Color.White
        btnChange.BackColor = Color.Blue
        btnChange.ForeColor = Color.White
        CurrentScheme = 2
    End Sub

    Private Sub btnChange_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
btnChange.Click
        If CurrentScheme = 1 Then
            SetColourScheme2()
        Else
            SetColourScheme1()
        End If
    End Sub
End Class
```

[4] **Run** the program, and enter text in the text box before pressing the button to change the colours.


Procedure Demo - Colour change
Some sample text
Change Colours
Procedure Demo - Colour change
Some sample text
Change Colours

STUDY THIS

Functions

Functions are procedures that return a value – in other words they work something out and assign the answer to the function name.

A **function** must have two things…

[1] a declared **type** for the returned value

[2] an **assignment** to the function **name** (saying what the value of the function is).

This example is a function that works out the largest of two numbers entered into two text boxes (txtFirst and txtSecond).

```
    Private Function Largest() As Integer
        If txtFirst.Text > txtSecond.Text Then
            Largest = txtFirst.Text
        Else
            Largest = txtSecond.Text
        End If
    End Function
```

The function is called inside an event handler by name and then, for example assigning it to a variable of the correct type…

```
Dim BestMark As Integer

BestMark = Largest()
```

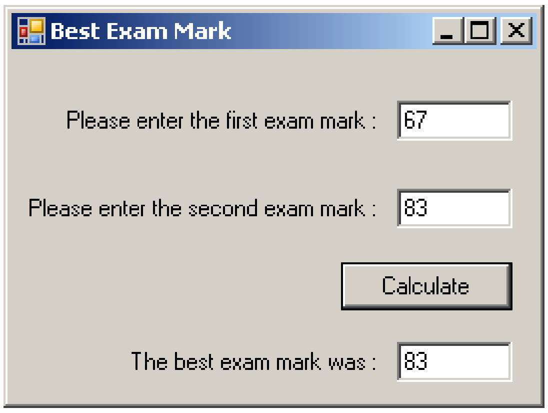
…or assigned to the property of an object…

```
txtBestMark.Text = Largest()
```

HANDS ON

Visual Basic Challenges 8.

[1] Use the function above to create a program that allows the user to enter two exam marks and displays which of the two is the highest mark.


Best Exam Mark

Please enter the first exam mark : 67

Please enter the second exam mark : 83

The best exam mark was : 83

Parameters

Subroutines only really become useful when we pass **parameters** to them.

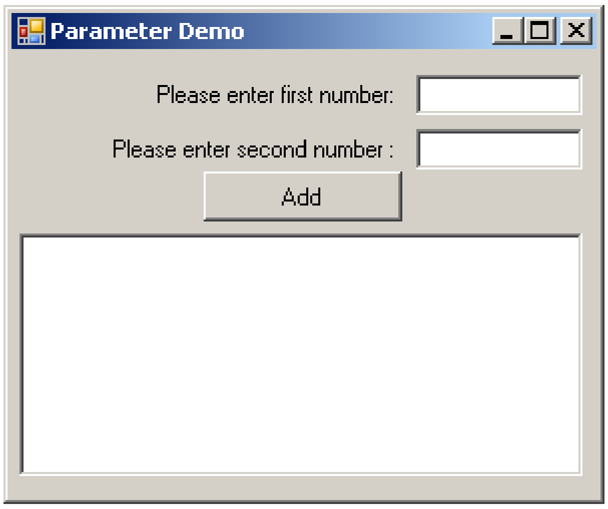
A **parameter** is a value that is passed to the subroutine. When the subroutine is executed, it will use this value.

Example : A procedure that draws a line of Xs in a TextBox…

HANDS ON

[1] Create a new Windows application.

Add two Labels, two TextBoxes(txtFirst and txtSecond) and a Button (btnAdd). Also add RichTextBox(rtbAdd)…

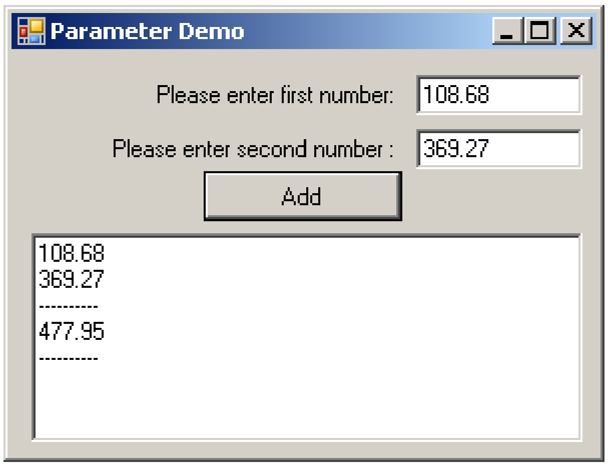


[2] On the Click event of button btnAdd add the event handler…

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

Dim Answer As Double
   Answer = Val(txtFirst.Text) + Val(txtSecond.Text)

   'Display calculation in TextBox
   rtbAdd.Clear()
   rtbAdd.AppendText(txtFirst.Text)
   rtbAdd.AppendText(vbCrLf) 'takes a new line
   rtbAdd.AppendText(txtSecond.Text)
   rtbAdd.AppendText(vbCrLf)
   rtbAdd.AppendText("----------") '10 dashes
   rtbAdd.AppendText(vbCrLf)
   rtbAdd.AppendText(Answer)
   rtbAdd.AppendText(vbCrLf)
   rtbAdd.AppendText("----------")
   rtbAdd.AppendText(vbCrLf)
End Sub
```

There are two lines that are repeated here for drawing the line of dashes – this is never a good thing and you should avoid repeated code in programming.

[3] Run the program and enter two numbers. The addition calculation should be displayed.



108.68
369.27

```
108.68
369.27
----------
477.95
----------
```

[4] To avoid repeated code … create your own subroutine…Change your code to the following:

```
    Private Sub DrawLine()
        rtbAdd.AppendText("----------")
        rtbAdd.AppendText(vbCrLf)
    End Sub

    Private Sub btnCalculate_Click(ByVal sender As
    System.Object, ByVal e As System.EventArgs) Handles
    btnCalculate.Click

        Dim Answer As Double
        Answer = Val(txtFirst.Text) + Val(txtSecond.Text)

        'Display calculation in TextBox
        rtbAdd.Clear()
        rtbAdd.AppendText(txtFirst.Text)
        rtbAdd.AppendText(vbCrLf) 'takes a new line
        rtbAdd.AppendText(txtSecond.Text)
        rtbAdd.AppendText(vbCrLf)
        DrawLine()
        rtbAdd.AppendText(Answer)
        rtbAdd.AppendText(vbCrLf)
        DrawLine()
    End Sub
```

Note that your procedure is called DrawLine and is called twice by the event handler.

[5] Now for some improvements….

First, it is better to use a loop in the DrawLine procedure, so change it to…

```
    Private Sub DrawLine()
        Dim i As Integer
        For i = 1 To 10
            rtbAdd.AppendText("-")
        Next i
        rtbAdd.AppendText(vbCrLf)
    End Sub
```

[6] The procedure is fine for drawing lines of 10 dashes…but maybe sometimes we would like lines of 20 dashes, …or 25 dashes etc…

To make the procedure more useful we pass a **parameter** to it….

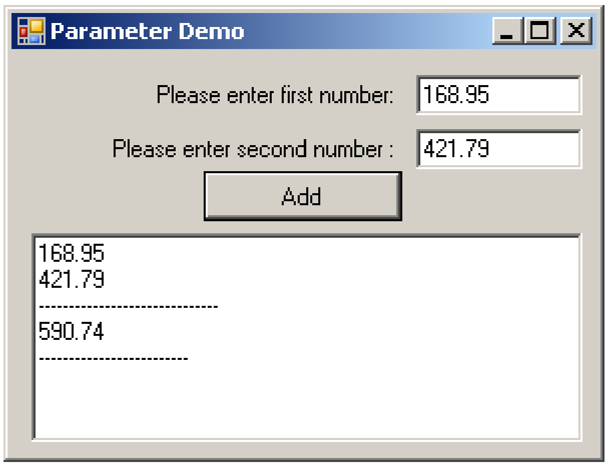
```
     Private Sub DrawLine(ByVal NumDashes As Integer)
         Dim i As Integer
         For i = 1 To NumDashes
             rtbAdd.AppendText("-")
         Next i
         rtbAdd.AppendText(vbCrLf)
     End Sub
```

NumDashes is the parameter. It is declared in the heading of the procedure- the data type of the parameter must also be declared.

In the event handler you will need to pass a value for the parameter – this must match the data type (integer in this case)…so change the lines that call the procedure to…

```
………
         DrawLine(30)
………
         DrawLine(25)
```

Running the program now should result in a display similar to this…



```
168.95
421.79
-----------------------------
590.74
------------------------
```

Summary

A subroutine is a small section of program code that can be called from other parts of a program. There are two types:

Procedure – that performs a specific task

Function – that performs a task and **returns a value**.

Parameters are passed to subroutines to make them useful.

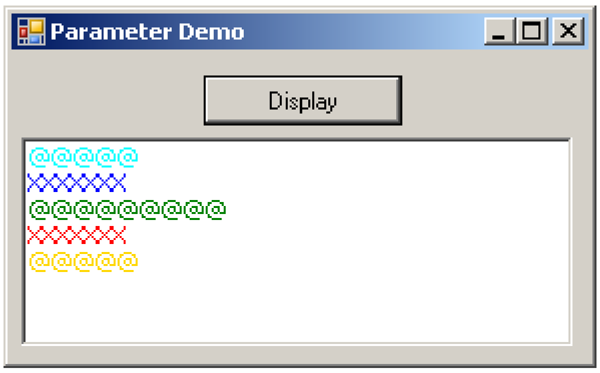
Visual Basic Challenges 8

[2] Enhance the above program so that you can pass two parameters to the DrawLine procedure –

- the number of characters to be drawn
- the character to use

So the instruction DrawLine(12,"@") would produce "@@@@@@@@@@@@"

Test your program by seeing if you can reproduce this screen display…

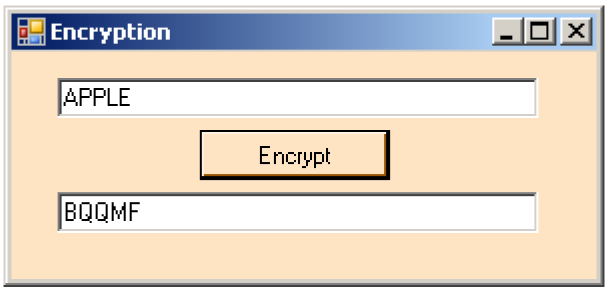


```
@@@@@
XXXXXX
@@@@@@@@@
XXXXXX
@@@@@
```

[3] **(a)** Write a new application that allows the user to input a string and encodes it by taking the 'next' character in the alphabet for each letter.

Include a function in your application that encrypts a string.

Test data : Input – APPLE   Output – BQQMF



APPLE

BQQMF

(b) Add a new section that decrypts a coded string.

Test Data : Input – BQQMF   Output – APPLE

STUDY THIS

Modules.

A module is a library of subroutines that can be used in other programs. It is a really good idea to use modules because...

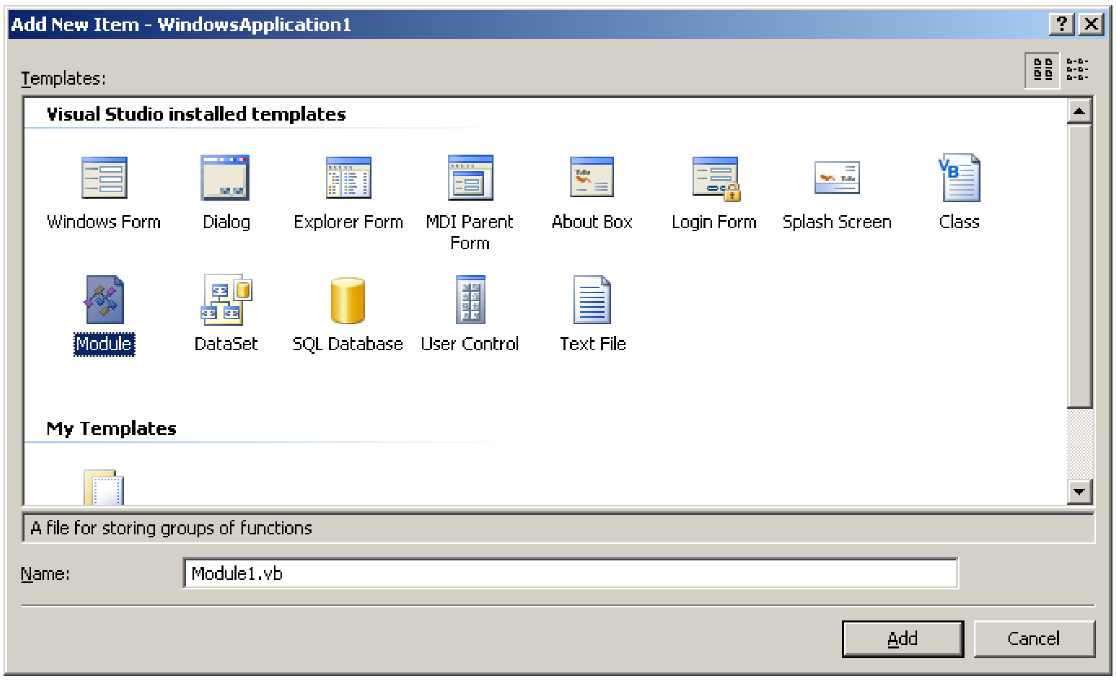
- it saves time programming if you can use subroutines you have created in other programs
- you know they will work because they have already been tested.

Any subroutine in a module can be called from anywhere in your program.

Global variables and constants can be declared in a module and used anywhere in the program.

Using constants is also a good idea because if their value changes, then you only have to change the value once in the module, and not in all the places the value is used in the program.

To add a module to a program click the [Project] menu and the [Add New Item...] option. Make sure you select the [Module] option...


Add New Item - WindowsApplication1

Templates:

Visual Studio installed templates

My Templates

A file for storing groups of functions

Name: Module1.vb

The Module should appear in your Solution Explorer window.