Visual Basic Functions

Adding spinners to your ride
(I have no idea what that means)

Subs and Functions

- Up to this point in our work, we have just developed stand alone macros
- Starting from a new worksheet we inserted a module to hold out macros

Subs and Functions

- Once we have this module within our spreadsheet, we have a blank workspace to work from

Subs and Functions

- We can add our programs (macros) to this by creating them from the taskbar
Subs and Functions

Where we are given a choice of the characteristics of the Procedure:

Subs and Functions

Up to this point, we have selected the Sub option.

Subs and Functions

And when we have developed code inside of a Sub, we have always run this as a stand alone macro program.

Subs and Functions

You can think of this as a construction analogy.

If the job is a small one, one small contractor can do everything.
Subs and Functions

- As the jobs grow in complexity, we might move to a general contractor and a number of sub-contractors.

<table>
<thead>
<tr>
<th>Subs and Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>Input</td>
</tr>
</tbody>
</table>

- Finally, as the jobs get really complex, we might call in specialist contractors to do specific tasks.

<table>
<thead>
<tr>
<th>Subs and Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>Input</td>
</tr>
</tbody>
</table>

General contractors are always Sub programs.

- Sub-contractors are also Sub programs.
- Specialist contractors are Function programs.

<table>
<thead>
<tr>
<th>Subs and Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>Input</td>
</tr>
</tbody>
</table>

Function programs have the advantage that they can also work from any sheet within the workbook that contains the module without running a macro.
Subs and Functions

• Function programs have the limitation that they can only return one value and that value is returned when the function is evaluated.

You have already utilized a number of functions that are provided by EXCEL.

We are just going to develop some on our own for specific purposes.

For example, when we use the square root function in our spreadsheet.

We are asking EXCEL to evaluate the function for the given arguments.

In this case, as we typed the function, EXCEL tells us that the argument to the function is a single number.
Subs and Functions

- In computer speak, we are passing a number to the function

Here we are locating what is in cell B2 and asking the function SQRT to be evaluated using the value from B2

Once the function is evaluated, the evaluation is returned

Whenever the value in B2 is changed, the value in C2 is automatically changed

The function is re-evaluated whenever the spreadsheet is recalculated
By default, a sheet is recalculated any time anything is changed on the sheet.

We don't want to be limited by the functions EXCEL provides, even though it provides lots of functions.

For example, we want to set up a function to calculate \( y \) given \( x \) from the algebraic function:

\[
\begin{align*}
y &= 3.45x^{1.45} + 6.334 \left( \frac{x}{21.45 + x} \right)
\end{align*}
\]

We can set this up to be calculated in a cell and then copy that to any other place on the sheet but we want to develop a special contractor (function) to calculate this:

\[
\begin{align*}
y &= 3.45x^{1.45} + 6.334 \left( \frac{x}{21.45 + x} \right)
\end{align*}
\]
Given a value for $x$ 
Calculate a value for $y$ and return it to the calling cell or program

$$y = 3.45x^{1.45} + 6.334 \left( \frac{x}{21.45 + x} \right)$$

We will start by adding a Function procedure to our module

Notice that we have changed the radio button from Sub to Function

Again, we give a name to our Function 
In this case, the name will be CalcY
Subs and Functions

- The shell of the function is added to our module

Subs and Functions

- In our other cases, when we added a Sub, the shell was slightly different

Subs and Functions

- The code for this function will still be contained between these two statements

Subs and Functions

- The largest difference is that we have to provide a parameter list for the function
Subs and Functions

- The parameter list is a list of the variables which will be used in evaluating the function.
- The variables on the right side of the equals sign are the parameters.

You may also see these called arguments to the function.
It is just really a matter of semantics to us.

In our case, we have one variable that is necessary to evaluate the function.
The variable will be replace x everywhere in the function when it is evaluated.

We will tell the function to expect a value by putting it in to the argument list.

\[ y = 3.45 \times 10^6 + 6.33 \left( \frac{x}{21.45 + x} \right) \]
Subs and Functions

- What we call the variable/parameter/argument here really doesn’t matter
- It will only be used within this function

```
Public Function CalcY(x As Single)
    End Function
```

Subs and Functions

- We do need to tell the function what type of value to expect and we do this in the same manner that we did in the Dim statement earlier

```
Public Function CalcY(x As Single)
    End Function
```

Subs and Functions

- In this case, we are going to expect a numerical argument

```
Public Function CalcY(x As Single)
    End Function
```

Subs and Functions

- We also need to tell the function what type of value to return to the calling program
- We need to type the function

```
Public Function CalcY(x As Single) As Single
    End Function
```
Subs and Functions

- We do this by adding the As keyword and then the type after the parameter list outside of the parenthesis

```vbnet
Public Function CalcY(x As Single) As Single
    ' Calculation code here
End Function
```

Subs and Functions

- Now we have set up the type for the parameters and for the function
- Any other variables that we will use in the function are typed in the usual manner

```vbnet
Public Function CalcY(x As Single) As Single
End Function
```

Subs and Functions

- In this case, all we want to function to do is to calculate using the equation we started with

```vbnet
Public Function CalcY(x As Single) As Single
    CaloY = 3.45 + x * 1.95 + 6.33x / (21.45 + x)
End Function
```

Subs and Functions

- Notice that the expression is developed with the function name on the left side of the equal sign

```vbnet
Public Function CalcY(x As Single) As Single
    CaloY = 3.45 + x * 1.95 + 6.33x / (21.45 + x)
End Function
```
Subs and Functions

- It is critical that you do this
- This is where the evaluation of the function is placed and when End Function is encountered, it is this evaluation that is sent back to the calling program

Public Function CalcY(x As Single) As Single
    CalcY = 9.45 * x * 1.95 + 6.334 * |x / (21.45 + x)|
End Function

Subs and Functions

- If the program is free of syntactical errors, we can just use it as we do any other function in the spreadsheet

Public Function CalcY(x As Single) As Single
    CalcY = 9.45 * x * 1.95 + 6.334 * |x / (21.45 + x)|
End Function

Subs and Functions

- In addition to being able to call this new function from a spreadsheet, we can call it from a macro

Public Function CalcY(x As Single) As Single
    CalcY = 9.45 * x * 1.95 + 6.334 * |x / (21.45 + x)|
End Function

Subs and Functions

- This allows us a wide range of versatility in how we can use these functions

Public Sub DoThis()
    Dim SelectedCell As Range
    For Each SelectedCell In Selection
        If Not IsDefined(SelectedCell) Then
            SelectedCell.Offset(0, 0).Value = CalcY(SelectedCell.Value)
        End If
    Next SelectedCell
End Sub
Homework

• Generate a function to take three integer values as arguments
• Return as a function evaluation the value of the argument which is closest to the average of the three values
• DUE 9 FEB 2004

Public Function QuadRoot(a As Single, b As Single, c As Single) As Single
  ' New Calculate the largest root of a quadratic
  Dim b2 As Single, c2 As Single
  Dim quadRoot As Single
  Dim center As Single
  Dim sqrt As Single
  center = (-b * 2 - c * a) / 8 * a
  sqrt = Sqr(center) / 2 * a
  If sqrt > sqrt2 Then
    quadRoot = sqrt
  Else
    quadRoot = sqrt2
  End If
  QuadRoot = quadRoot
End Function

Here is a working version of the quadratic root program. The square root function is not available. More on that WED.