Excel VBA Course Notes: 2. Using Loops (File: VBA02-Loops.xls)

Why Loops?

The purpose of a loop is to get Excel to repeat a piece of code a certain number of times. How many times the code gets repeated can be specified as a fixed number (e.g. do this 10 t imes), or as a variable (e.g. do this for as many times as there are rows of data).

Loops can be constructed many different ways to suit different circumstances. Often the same result can be obtained in different ways to suit your personal preferences. The se exercises demonstrate a selection of different ways to use loops.

There are two basic kinds of loops, both of which are demonstrated here: **Do...Loop** and **For...Next** loops. The code to be repeated is placed between the key words.

Open the workbook **VBA02-Loops.xls** and take a look at the four worksheets. Each contains two columns of numbers (columns A and B). The requirement is to calculate an average for the numbers in each row using a VBA macro.

Now open the Visual Basic Editor (**Alt+F11**) and take a look at th e code in *Module1*. You will see a number of different macros. In the following exercises, first run the macro then come and read the code and figure out how it did what it did.

You can run the macros either from the Visual Basic Editor by placing your cur sor in the macro and pressing the **F5** key, or from Excel by opening the Macros dialog box (**ALT+F8**) choosing the macro to run and clicking **Run**. It is best to run these macros from Excel so you can watch them as they work.

Exercise 1: Do… Loop Until…

The object of this macro is to run down column C as far as is necessary putting a calculation in each cell as far as is necessary.

On **Sheet1** select cell **C2** and run the macro **Loop1**.

Here's the code:

```
Sub Loop1()
' This loop runs until there is nothing in the next column
    Do
    ActiveCell.FormulaR1C1 = "=Average(RC[-1],RC[-2])"
    ActiveCell.Offset(1, 0).Select
    Loop Until IsEmpty(ActiveCell.Offset(0, 1))
End Sub
```

This macro places a formula into the active cell, and moves into the next cell down. It uses **Loop Until** to tell Excel to keep repeating the code *until* the cell in the adjacent column (column D) is empty. In other words, it will keep on repeating as long as there is something in column D.

Delete the data from cells **C2:C20** and ready for the next exercise

Exercise 2: Do While… Loop

The object of this macro is to run down column C as far as is necessary putting a calculation in each cell as far as is necessary.

On **Sheet1** select cell **C2** and run the macro **Loop2**

Here's the code

```
Sub Loop2()
' This loop runs as long as there is something in the next column
    Do While IsEmpty(ActiveCell.Offset(0, 1)) = False
    ActiveCell.FormulaR1C1 = "=Average(RC[-1],RC[-2])"
    ActiveCell.Offset(1, 0).Select
    Loop
End Sub
```

This macro does the same job as the last one usin g the same parameters but simply expressing them in a different way. Instead of repeating the code *Until* something occurs, it does something *While* something is the case. It uses **Do While** to tell Excel to keep repeating the code *while* there is something in the adjacent column as opposed to *until* there is nothing there. The function **IsEmpty = False** means "Is Not Empty".

Delete the data from cells **C2:C20** and ready for the next exercise

Exercise 3: Do While Not… Loop

The object of this macro is to run down colu mn C as far as is necessary putting a calculation in each cell as far as is necessary.

On **Sheet1** select cell **C2** and run the macro **Loop3**.

Here's the code:

```
Sub Loop3()
' This loop runs as long as there is something in the next column
    Do While Not IsEmpty(ActiveCell.Offset(0, 1))
    ActiveCell.FormulaR1C1 = "=Average(RC[-1],RC[-2])"
    ActiveCell.Offset(1, 0).Select
    Loop
End Sub
```

This macro makes exactly the same decision as the last one but just expresses it in a different way. **IsEmpty = False** means the same as **Not IsEmpty**. Sometimes you can't say what you want to say one way so VBA often offers an alternative syntax.

Delete the data from cells **C2:C20** and ready for the next exercise

Exercise 4: Including an IF statement

The object of this macro is as before, but without replacing any data that may already be there.

Move to **Sheet2**, select cell **C2** and run the macro **Loop4**.

Here's the code:

```
Sub Loop4()
' This loop runs as long as there is something in the next column
' It does not calculate an average if there is already something in the cell
    Do
    If IsEmpty(ActiveCell) Then
        ActiveCell.FormulaR1C1 = "=Average(RC[-1],RC[-2])"
    End If
    ActiveCell.Offset(1, 0).Select
    Loop Until IsEmpty(ActiveCell.Offset(0, 1))
End Sub
```

The previous macros take no account of any possible contents that might already be in the cells into which it is placing the calculations. This macro uses an IF statement that tells Excel to write the calculation *only if* the cell is empty. This prevents any existing data from being overwritten. The line telling Excel to move to the next cell is *outside* the IF statement because it has to do that anyway.

Exercise 5: Avoiding Errors

This macro takes the IF statement a stage further, and doesn't try to calculate an average of cells that are empty.

First, look at the problem. Move to **Sheet3**, select cell **C2** and run the macro **Loop4**.

Note that because some of the pairs of cells in columns A and B are empty, the =AVERAGE function throws up a #DIV/0 error (the Average function adds the numbers in the cells then divides by the number of numbers - if there aren't any numbers it tries to divide by zero and you can't do that!).

Delete the contents of cells **C2:C6** and **C12:C20**. Select cell **C2** and run the macro **Loop5**.

Here's the code:

```
Sub Loop5()
' This loop runs as long as there is something in the next column
' It does not calculate an average if there is already something in the cell
' nor if there is no data to average (to avoid #DIV/0 errors).
    Do
    If IsEmpty(ActiveCell) Then
        If IsEmpty(ActiveCell.Offset(0, -1)) And IsEmpty(ActiveCell.Offset(0, -2)) Then
            ActiveCell.Value = ""
        Else
            ActiveCell.FormulaR1C1 = "=Average(RC[-1],RC[-2])"
        End If
    End If
    ActiveCell.Offset(1, 0).Select
    Loop Until IsEmpty(ActiveCell.Offset(0, 1))
End Sub
```

Note that this time there are no error messages because Excel hasn't tried to calculate averages of numbers that aren't there.

In this macro there is a second IF statement *inside* the one that tells Excel to do something only if the cell is empty. This second IF statement gives excel a choice. Instead of a simple **If** there is an **If** and an **Else**. Here's how Excel reads its instructions…

"If the cell has already got something in, go to the next cell. But if th e cell is empty, look at the corresponding cells in columns A an B and if they are both empty, write nothing (""). Otherwise, write the formula in the cell. Then move on to the next cell."

Exercise 6: For… Next Loop

If you know, or can get VBE to find out, how many times to repeat a block of code you can use a **For... Next** loop.

Move to **Sheet4**, select cell **C2** and run the macro **Loop6**.

Here's the code:

```
Sub Loop6()
' This loop repeats for a fixed number of times determined by the number of rows
' in the range
    Dim i As Integer
    For i = 1 To Selection.CurrentRegion.Rows.Count - 1
    ActiveCell.FormulaR1C1 = "=Average(RC[-1],RC[-2])"
    ActiveCell.Offset(1, 0).Select
    Next i
End Sub
```

This macro doesn't make use of an adjacent column of cells like the previ ous ones have done to know when to stop looping. Instead it counts the number of rows in the current range of data and uses the **For... Next** method to tell Excel to loop that number of times (minus one, because when VBA counts it starts at zero).

Exercise 7: Getting the Reference From Somewhere Else

Select cell **G2** and run the macro **Loop7**.

Here's the code:

```
Sub Loop7()
' This loop repeats a fixed number of times getting its reference from elsewhere
    Dim i As Integer
    Dim intRowCount As Integer
    intRowCount = Range("A1").CurrentRegion.Rows.Count - 1
    For i = 1 To intRowCount
    ActiveCell.FormulaR1C1 = "=Average(RC[-5],RC[-6])"
    ActiveCell.Offset(1, 0).Select
    Next i
End Sub
```

You can get the reference for the number of loops from anywhere. This m acro places a set of calculations in column G for a number of times dictated by the number of rows in the block of data starting with cell A1. The **For... Next** statement has been simplified a bit by first declaring a variable **intRowCount** and filling it with the appropriate information (how many rows in the block by A1). This variable gets used in the next line instead of a long line of code. This is just another example of doing the same job a different way.

If you wanted to construct a loop that always ran a block of code a fixed number of times, you could simply use an expression like:

```
For i = 1 To 23
```

Exercise 8: About Doing Calculations…

All the previous exercises have placed a calculation into a worksheet cell by actually writing a regular Excel function into the cell (and leaving it there) just as if you had typed it yourself. The syntax for this is:

```
ActiveCell.FormulaR1C1 = "TYPE YOUR FUNCTION HERE"
```

These macros have been using:

```
ActiveCell.FormulaR1C1 = "=Average(RC[-5],RC[-6])"
```

Because this method actual ly places a function into the cell rather than a value, their results will change as the cells that they refer to change, just like regular functions – because they are regular functions. The calculating gets done in Excel because all that the macro did wa s to write the function.

If you prefer, you can get the macro to do the calculating and just write the result into the cell. VBA has its own set of functions, but unfortunately AVERAGE isn't one of them. However, VBA does support many of the commoner Excel functions with its **WorksheetFunction** method.

On **Sheet1** select cell **C2** and run the macro **Loop1**.

Take a look at the cells you just filled in. Each one contains a function, written by the macro.

Now delete the contents from the cells **C2:C20**, select cell **C2** and run the macro **Loop8**.

Here's the code:

```
Sub Loop8()
    Do
    ActiveCell.Value = WorksheetFunction.Average(ActiveCell.Offset(0, -1).Value, _
         ActiveCell.Offset(0, -2).Value)
    ActiveCell.Offset(1, 0).Select
    Loop Until IsEmpty(ActiveCell.Offset(0, 1))
End Sub
```

Take a look at the cells you just filled in. This time there's no function, just the value. All the calculating was done by the macro which then wrote the value into the cell.