Loops & Flow Control

Loops are especially useful in Excel because so much of what is done involves ranges of values, each which needs to be inspected, evaluated and perhaps modified. VBA provides the following loop constructs for iterating range values.

A Note on Programming Style

With loops and nested conditional statements, it is good to develop a strategy to indent your code to delineate where the nested clauses in your code are.

This makes it easy to see where sub structures end. This is handy when troubleshooting to find where there might be a missing or extra End or Next tag. Each new group is nested deeper in the code.

Example  // 2, 3 or 4 spaces of indentation

Sub XIT()
    For j = 1 To 7                   // indented for loop
        Cells(j, j).Value = j & "," & j // indent contained statement
        If j = 4 Then                 // further indent the if clause
            Exit For
        End If
    Next j                          // match closing phrases
End Sub

VBA Loops

- For ...Next
- Do ...While
- Do ...Until
- While...Loop
- For Each... Next
The For...Next Loop

The For...Next Loop iterates over a specified range of values. Usually the value specified is used to change which item is being considered in the loop.

Example

Sub ForNx()
    For i = 1 To 10
        Cells(i, 1).Value = "A" & i
    Next i
End Sub

// it seems to be optional to state the loop variable after Next. Perhaps it is clearer
// to explicitly state the loop variable, in the sense of, 'the next value of i'.

What is going on?

In the first case the index "i" is set to 1. VBA compares 1 to the limit value, in this loop is set to 10. Because 1 is less than 10, the loop proceeds and 1 is entered as the value for variable i in Cells(i,1) and where it appears being concatenated to A. The evaluation proceeds with the result stored in Cell(1,1).

The Next directive re-enters the loop but this time index is incremented by 1 so the value is set to 2. 2 is less than 10 so the loop proceeds with the value 2 entered into the loop statement. This continues until the index value is 10 at which time the value is no longer less than 10 which creates the condition that breaks out of the loop and control proceeds to the statement immediately after the Next statement of the loop.

The Exit Directive  // planning an early exit

If a condition is met that makes continuing unnecessary, the Exit directive may be used. Exit is used in loops to exit when some condition has been met.

Example

Sub XIT()
    'loop never gets past 4 as loop exits at j = 4
    For j = 1 To 7
        Cells(j, j).Value = j & "," & j
        If j = 4 Then
            Exit For
        End If
    Next j
End Sub
The Step Keyword

Loops can also be created that increment in Steps using the Step keyword. In the following example the effect is that every second step is skipped. This could be used for putting a colored row at regular increments in a spreadsheet.

Sub Steps()
    For j = 1 To 6 Step 2
        For k = 1 To 6 Step 2
            Cells(j, k).Interior.ColorIndex = 13
        Next k
    Next j
End Sub

Nested Loops

In order to process a range that is two-dimensional that is more than just values of a row or column nested loops are needed.

Example

Sub Nest()
    ' for every j value 5 k values are processed
    For j = 1 To 5
        For k = 1 To 5
            Cells(j, k).Value = j & "," & k
            If k = j Then
                Cells(j, k).Interior.Color = RGB(230, 250, 220)
            End If
        Next
    Next j
End Sub

What is going on in nested loops?

Essentially the same process is taking place, however for each of the outer loop index values, all the inner values are being evaluated.

Example

For j = 1, loop through all k values,
    . . . when inner loop completes, increment outer loop
For j = 2, loop through all k values etc.
Do While Loops  // a plain Do … Loop is a ‘forever loop’

Sometimes you might have a dynamic list of data of variable length. For Next loops are restricted in that you must know the length of the collection you are iterating over. (Not really a big issue as there are means to get the length of the list you are iterating.)

You can have a Do loop that runs forever. There are some applications for 'forever loops' but they are unlikely to be of much use processing finite data sets.

Mr. Excel suggests, in the event you have say 5000 entries that are being processed you can watch the Do loop and then, on observing the task is complete, stop the process with a Ctrl Break key combo.

Even in this circumstance, it probably would be better just to include a condition that stops the loop when the last piece of data has been processed. While used routinely in server applications that are always online 'listening', Excel probably is never very happy hosting a forever loop.

Generally you run a Do loop until some condition is satisfied. This condition is supplied to a While or Until clause.

### A Forever Loop May Cause Unpredictable and Unstable Conditions in Excel

// Be prepared for Excel to crash on this one. It made the application very unhappy. // Worked nicely while it lasted. Ctrl -Break killed it but the application remained // in an unstable state and had to be restarted.

### A Colorful But Unstable Relationship Between a Macro and Excel

<table>
<thead>
<tr>
<th>Sub doo( )</th>
<th>Do</th>
<th>Cells(1, 1).Interior.ColorRange = 35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cells(1, 1).Interior.ColorRange = 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cells(1, 1).Interior.ColorRange = 50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cells(1, 1).Interior.ColorRange = 19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cells(1, 1).Interior.ColorRange = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'escape with a Ctrl Break</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>End Sub</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Following is the Do Loop Form.
The Do Loop Form // a forever loop

Do
‘goes forever!
Loop

Next is shown the form of the Do While statement with condition preceding the Do loop.

The Do While Form

Do While Boolean_condition Loop

' what to do until condition is met

The Do Loop with a While statement repeats indefinitely until the while condition proves to be true. In the following example, the equality statement is 'inverted' with the Not keyword used to mean 'Do while t is not 8'. The loop iterates seven times but not the eighth.

Example

Sub DW( )
    Dim t As Integer
    t = 1
    Do While Not t = 8
        Cells(1, t).Value = t
        t = t + 1
    Loop
End Sub

'Until' Replaces 'While Not'

To avoid the use of a negated Boolean statement, the 'Until' keyword may be used. The above example becomes the following equivalent code using the 'Until' keyword. This form adds 'error resistance' and better readability.

Example // 'Do until z is 8 is easier to understand

Sub DU( )
    Dim z As Integer
    z = 1
    Do Until z = 8
        Cells(1, z).Value = z
        z = z + 1
    Loop
End Sub
Repositioning the Conditional Clause

In the event that you want the loop to execute at least once, the condition can be placed at the end of the loop, following the Loop keyword.

Example

Sub AtLeastOnce()
    Do Until True
        Cells(1, 1).Value = " Condition Before Loop --> Doesn't get to go even once"
        Loop
    Do
        Cells(2, 1).Value = "Condition After Loop Body --> Goes At Least Once"
        Loop Until True
    End Sub

A Snippet of IO Code in Excel

The text on page 115 shows some partial code using the EOF character, which abbreviates "End Of File" and is a standard symbol that can be used as a condition to terminate a loop.

Example

Do until EOF(1)

Importing and exporting files and data to and from different computer devices is generally referred to as IO. To illustrate the use of the EOF symbol we can add a bit of working IO code.

Put Robert Frost's poem in a file called Frost.txt directly under the C:\.

The following macro is 'hard wired', specifically written to import the Frost.txt file and display it on a spreadsheet.

Macro That Imports a File // another approach to that created by the macro recorder.

Sub ReadFile()
    Dim ExApp As Object
    Dim book_frost As Object
    Dim sheet_frost As Object
    Dim filename As String
    Set ExApp = CreateObject("Excel.Application")
    filename = "Frost.txt"
    Set book_frost = ExApp.Workbooks.Open(filename)
    Set book_frost = ExApp.ActiveWorkbook
    book_frost.Sheets(1).Activate
    Set sheet_frost = book_frost.Sheets(1)
    ExApp.Visible = True
    ExApp.UserControl = True
End Sub
The Road Not Taken   Robert Frost

Two roads diverged in a yellow wood,  
And sorry I could not travel both  
And be one traveler, long I stood  
And looked down one as far as I could  
To where it bent in the undergrowth;

Then took the other, as just as fair,  
And having perhaps the better claim  
Because it was grassy and wanted wear,  
Though as for that the passing there  
Had worn them really about the same,

And both that morning equally lay  
In leaves no step had trodden black.  
Oh, I marked the first for another day!  
Yet knowing how way leads on to way  
I doubted if I should ever come back.

I shall be telling this with a sigh  
Somewhere ages and ages hence:  
Two roads diverged in a wood, and I,  
I took the one less traveled by,  
And that has made all the difference.

Grabbing a File's Content By Line

The following code shows the EOF token in use. Inside the Do loop the Cells that the lines are assigned to are incremented by the variable j that increases by 1 each time the Do loop executes.

Sub FC()  
Dim line As String  
Dim Fields As String  
Dim j As Integer  
j = 1  
Open "c:\Frost.txt" For Input As #1 ' Open file.  
Do While Not EOF(1) ' Loop until end of file.  
    Line Input #1, Fields  
    Cells(j, 1).Value = Fields  
    j = j + 1  
    Loop  
Close #1 ' Close file.  
End Sub
The While . . . Wend Loop  // speculating → Wend → 'While End'

Microsoft replaced the While . . . Wend loop with the more flexible Do Loop. The While . . . Wend loop is supported for backwards compatibility. Run the following directly from the VB Editor and see the results in the Immediate Window. Note

Example

Sub Wendy()
    Dim j As Integer
    j = 1
    While j < 4
        Debug.Print "While Jay is " & j
        j = j + 1
    Wend
    Debug.Print ("Wend to Go")
End Sub

Object Variables  // brings attention to the macro editor not using loops

The macro recorder doesn't use loops. It is so often is dealing with collections of objects that it uses a looping mechanism more suited to object iteration.

Variables typically hold a single value, for instance, an Integer will hold the number 41.

A variable that points to an object, is still holding a single value, but in this case it is a reference to an object. This is a powerful relationship.

The object being referred to may be a terrifically complex data type that holds compound values and other properties.

// see object variable discussion on lower page 117 in 2010 text, or page 89 in 2013 book

The text describes two developer camps, those that do not declare variables and those that recommend using DIM statements to declare all variables

Even though Mr. Excel himself seems to be in the camp of those that don't bother with variable declarations, he still promotes the use of declarations for Object types. The following shows an object declaration accompanied by the Set command used with an assignment.

Example

Dim cello as Range
Set cello = Range("F9")
One advantage to using named references, is they provide easy-to-use short forms for the object in question.

As Mr. Excel says, 'WSD' is shorter than "ThisWorkbook.Worksheets("Data")

The For . . . Each Loop

The For . . . Each Loop uses a changing object variable to iterate over an object collection. Note in the following line, Var is used but can be any name.

Example

For Each Var in Object_Collection

Typically, when dealing with Ranges this variable will be called cell or Cell.

Example

Sub ForEach()
For Each Cell In Range("A1:A5")
Cell.Value = Cell.Address(False, False)
Next Cell
End Sub

// the use of the term 'Cell' in loops may initially mislead us to think there is a Cell object
// to correspond to the Cells.

'For . . . Each' Advantage for Dis-Contiguous Ranges

While dis-contiguous ranges may be iterated with conventional loops (if logic is included to exclude cells within the range that are not part of the range), the 'For Each' loop is simpler to use as it just sees a collection of object that it is going to reference in sequence, contiguous or not.

Example

Sub M()
For Each Cell In Range("A1:A5,B2,C3,D2,E4,E1:E5")
   Cell.Value = Cell.Address(False, False)
Next Cell
End Sub
We might add an example from the book on page 90 (2013 text), bottom of page 118 (2010 text) where a search is done for a open workbook with a first worksheet called Menu.

Example  // from 'VBA & Macros' Microsoft Excel 2010 / 2013, Bill Jelen & Tracy Syrstad

For Each wb in Workbooks
    If wb.Worksheets(1).Name = "Menu" Then
        WBFound = True
        WBName = wb.Name
    Exit For
    End If
Next wb

// Case study is a look another recursive file check this time JPG files are found and put into a single directory

Flow Control Using If Then Else and Case Statements

Flow control refers to using logic to decide which path processing will take. This is accomplished in VBA, as in other programming languages using 'If ...Then ...Else' constructs. (An interesting aside, the VBA macro recorder makes no use of If constructs.)

The If statement in VBA take the following basic form.

If Then Else Form

If  Boolean_value Then
    'do something
    'do some more
Else
    'do something else
End If

A Word on Conditions

The If statements take Boolean values that is True or False, or the equivalent evaluations, 1 or 0.
Boolean Operators

Boolean Values evaluate as follows when combined with each of the specified Boolean operators. With respect to the AND operator, all ANDed Booleans have to be true in order for the result to be True.

**AND**
- True And True → True
- True And False → False
- False And True → False
- False And False → False

With OR as long as one or more values are true the expression evaluates to true.

**OR**
- True OR True → True
- True OR False → True
- False OR True → True
- False OR False → False

**XOR**
XOR stands for exclusive OR and evaluates like an OR except when both values are true then the output is false.

- True XOR True → False
- True XOR False → True
- False XOR True → True
- False XOR False → False

The following macro demonstrates the And and Or Boolean combinations. It is left to as part of the assignment to extend the following macro for XOR, the exclusive or operator.

**Boolean Example**

Sub Bool()
Dim ray As Variant
' Just serving as labels
ray = Array("T", "T", "T", "F", "F", "T", "F", "F")
Dim j As Integer
' counter starts at zero as the above array has an offset count starting at zero
j = 0
' another label
Range("A1").Value = "AND Operator"
For Each Cell In Range("A3:B6")
Cell.Value = ray(j)
j = j + 1
Next Cell

' these are the real And evaluations
Range("C3").Value = True And True
Range("C4").Value = True And False
Range("C5").Value = False And True
```vba
Range("C6").Value = False And False
Range("A8").Value = "OR Operator"
j = 0 'reset j

For Each Cell In Range("A10:B13")
    Cell.Value = ray(j)
j = j + 1
Next Cell

' these are the real Or evaluations
Range("C10").Value = True Or True
Range("C11").Value = True Or False
Range("C12").Value = False Or True
Range("C13").Value = False Or False

' exclusive or, Xor like Or except True Xor True --> False

End Sub

The following code's output is all sent to the Immediate Window. We might have also called it the Debug window.

**Example With Various Conditions**

```vba
Sub Conditions()
    If True Then
        Debug.Print "Literal True evaluates to " & True
    Else
        Debug.Print "False"
    End If

    ' literal True and False is the same as 1 and 0
    ' the next example evaluates a simple expression

    If 1 - 1 Then
        Debug.Print True
    Else
        Debug.Print "1-1 is zero evaluates to False: " & False
        Debug.Print 'to add a blank line
    End If

    ' compound boolean expressions

    If Workbooks(2).Name = "Book2.xlsm" And Worksheets(1).Name = "Sheet1" Then
        Debug.Print "The 2nd workbook's name is Book2.xlsm & the 1st"
        Debug.Print "Worksheet is called Sheet1 so evaluation is " & True
    Else
        Debug.Print False
    End If
End Sub
```
In Excel, when looping over a range, you will often be looking for a particular value.

The following using the 'If Then' construct with an Else. In a row at the very end, there are two If constructs without Else clauses. These two could have been simplified with an Else as in 'if it isn't a head it couldn't be anything else but a tail, but we wanted to show all form combinations.

Example

Sub CoinToss()
    Dim j, k, H_count, T_count As Integer
    Dim r As Double
    For k = 1 To 8
        For j = 1 To 8
            r = Math.Rnd
            If r > 0.5 And r <> 0.5 Then 'rid of .5 as the round up or down issue
                Cells(k, j).Value = "H"
            Else
                Cells(k, j).Value = "T"
            End If
        Next j
    Next k
    H_count = 1
    T_count = 1

    For Each Cell In Range("A1:H8")
        ' could have used Else
        ' a series of Ifs is structural similar to a Select Case control form
        If Cell.Value = "H" Then
            H_count = H_count + 1
        End If

        If Cell.Value = "T" Then
            T_count = T_count + 1
        End If
    Next Cell

    Range("A10").Value = "Heads "
    Range("B10").Value = H_count
    Range("A11").Value = "Tails ".
    Range("B11").Value = T_count
End Sub

Mr. Excel provides examples of practical conditions that you might use to determine subsequent actions in a macro.

// page 92 in 2013 text, ( page 121 in 2010 text )
Following is a selection from those examples.

**Example**  // from 'VBA & Macros' Microsoft Excel 2010/2013, Bill Jelen & Tracy Syrstad

If Range("A1").Value = “Inventory” And Range("C2").Value = “Fruit” Then

**Nested If Else Statements**  // the ElseIf Keyword

Excel allows nesting If Else statements using the **ElseIf** keyword as is shown in the following example.

**Example**

Sub Candidate()
    water = True
    atmosphere = False
    plantlife = False

    If water And atmosphere And plantlife Then
        Cells(1, 1).Value = "Inhabitable"
        ElseIf water And atmosphere Then
            Cells(1, 1).Value = "Inhabitable but bring biological supplies"
            ElseIf atmosphere Then
                Cells(1, 1).Value = "Bring water to atmosphere converters & biological equipment"
                ElseIf plantlife Then
                    Cells(1, 1).Value = "Bring atmosphere to water converter & biological equipment"
        End If
    End Sub

**Select Case Statement**

We saw the Select Case statement earlier, but include it here for completeness. Often the Case is used in a Loop to avoid a lot of nested If Else statements.

**Select Case Form**

Select Case  *comparison_value*
    Case  *case_1_value*
        Case 1 statement executed
    Case  *case_2_value*
        Case 2 statement executed
    Case  *case_n_value*
        Case n statement executed
    Case Else
        **default Case executed**
End Select
Example

Sub Shapes()
Dim my_val As String

my_val = "Triangle"
Select Case my_val
    Case "Triangle": MsgBox "The shape is a three-sided triangle."
    Case "Square" : MsgBox "The shape is a four-sided square"
    Case "Pentagon": MsgBox "The shape is a five-sided pentagon"
    Case Else MsgBox "The shape is unknown"
End Select
End Sub

// Mr. Excel's ends his chapter with a complicated flow control macro using
// If then and Select Case Statements. Look at it if you wish.

Assignment

1 ). Extend the Boolean Macro in this note, which already has results
for And and Or operators, to include the Xor Boolean Operator.

2 ) Create a single macro that demonstrates each of the following loops.

   Use a comment to label and briefly describe what is happening in
   the loop being exampled.

   • For ...Next

Put the index number associated with the For loop into 7 columns in the
first row of a spreadsheet.

   • Do ...While

Demonstrate that this form of loop executes only once if the while statement
follows the body of the Do loop evaluates to false. Use the one loop to put
data into a range of a spreadsheet.

   • Do ...Until

Copy the simple Do While loop that executes only once from above and re-write
it using an Until control.
• For Each... Next

Create a non-contiguous range of something around a dozen cells and use a 'For each Next' loop to count the number of cells in it. Collect the total number of cells using the statement, 'count = count + 1 '. Label a cell under or beside the range “Cell Total” and, in the cell beside this label, store the total number of cells counted.

3 ) Create a new macro with a Range of values containing the string values for each day of the week. Inside a For Each loop place a Select Case statement that takes the string values from the range you have created. (Use the Range “A4:G4”)

You may use the offset method, to place the output of each Case into a range below the range you have created. (For instance use offset to place elements in Range “A6:G6”).

For the string Case Values Monday to Friday have the Case put into the A1 cell the word “WORK”. For Saturday, the case should output “RECREATION” and for Sunday output “REST”.