VBA Programming Logic and Loops
(See Chapter 7 of Albright)

Kipp Martin

January 4, 2012
Files used in this lecture:

- programminglogic.xlsm

See the module \texttt{IfThenLogic} for illustrations of If Then logic.

See the module \texttt{Looping} for examples of For and Do loops.
Outline

If, Then Logic

If, Then, Else

If, Then, Elseif, Else

Select, Case

Looping
  For Loops
  For Each Loops
  Do While Loops
Motivation

Recording a Macro is often an effective way to generate VBA code.

Recording a Macro **NOT GOOD** when:

- there is conditional logic (if-then)
- looping – repeat essentially the same task with minor modification many times
If, Then Logic

It is very common to alter the flow of a program depending upon variable values or other logical conditions. This is achieved with **If Then** logic. The format is

```
If logical_condition Then
    code to execute
End If
```

```
Dim x As Double
x = InputBox("Enter Number")
If x > 0 Then
    MsgBox "Natural Logarithm = " & Log( x)
End If
```
### If, Then Logic

Relational Operators for If Then logic.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal To</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Strictly greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Strictly less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>
The If statement can be based on more than one logical condition joined together by logical operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>All conditions must be true</td>
</tr>
<tr>
<td>Or</td>
<td>At least one condition must be true</td>
</tr>
<tr>
<td>Not</td>
<td>negate a condition</td>
</tr>
</tbody>
</table>

Dim order_quantity As Double
Dim discount_rate As Double
If x >= 50 And x <= 100 Then
    discount_rate = .05
End If
If, Then Logic, Logical Operators

If only one statement is executed as a result of the If you do not need an End If you can use

If x > 0 Then y = 5

instead of

If x > 0 Then
  y = 5
End If
If, Then, Else Logic

Another logical structure is

If conditions Then
    statements if true
Else
    statements if false
End If

Dim x As Double
x = InputBox("Enter Number")
If x > 0 Then
    MsgBox "Natural Logarithm = " & Log( x)
Else
    MsgBox "The log function requires a positive number"
End If
If, Then, Else Logic

Yet, another logical structure is

If condition set 1 Then
    statements if true
ElseIf condition set 2
    statements if true
ElseIf condition set 3
    statements if true
.
.
.
ElseIf condition set N
    statements if true
Else
    statements to execute if all of the above false
End If
Sub IfThenElseIf()
    Dim x As Double, discount_rate As Double
    x = InputBox("Enter Number")
    If x > 0 And x <= 50 Then
        discount_rate = 0.05
        MsgBox "Discount Rate = " & discount_rate
    ElseIf x > 50 And x <= 100 Then
        discount_rate = 0.06
        MsgBox "Discount Rate = " & discount_rate
    ElseIf x > 100 And x <= 200 Then
        discount_rate = 0.07
        MsgBox "Discount Rate = " & discount_rate
    Else
        discount_rate = 0.08
        MsgBox "Discount Rate = " & discount_rate
    End If
End Sub
Select, Case

It is often desirable to avoid a lot of complicated **If, Then, Elself** statements. This is done with **Select, Case**.

Select Case testVal
  Case Value1
    statements if testVal = Value1
  Case Value2
    statements if testVal = Value2
  .
  .
  .
  Case ValueN
    statements if testVal = ValueN
  Case Else
    statements if all tests fail
End Select
Sub CaseExample1()
    Dim testNum As Integer
    testNum = InputBox("Enter Case from 1 to 3")
    Select Case testNum
        Case 1
            MsgBox "Case 1 Logic"
        Case 2
            MsgBox "Case 2 Logic"
        Case 3
            MsgBox "Case 3 Logic"
        Case Else
            MsgBox "Number not valid"
    End Select
End Sub
x = InputBox("Enter Number")
Select Case x
    Case 1 To 50
        discount_rate = 0.05
        MsgBox "Discount Rate = " & discount_rate
    Case 51 To 100
        discount_rate = 0.06
        MsgBox "Discount Rate = " & discount_rate
    Case 101 To 200
        discount_rate = 0.07
        MsgBox "Discount Rate = " & discount_rate
    Case Is > 200
        discount_rate = 0.08
        MsgBox "Discount Rate = " & discount_rate
    Case Else
        MsgBox "Number not valid"
End Select
Select, Case (Example 3)

The first Case that is passed will be selected.

```vbnet
Select Case x
    Case Is <= 50
        discount_rate = 0.05
        MsgBox "Discount Rate = " & discount_rate
    Case Is <= 100
        discount_rate = 0.06
        MsgBox "Discount Rate = " & discount_rate
    Case Is <= 200
        discount_rate = 0.07
        MsgBox "Discount Rate = " & discount_rate
    Case Is > 200
        discount_rate = 0.08
        MsgBox "Discount Rate = " & discount_rate
    Case Else
        MsgBox "Number not valid"
End Select
```
Select, Case (Example 3)

In VBA we exit the **Case** block when a true condition is found. This is not true in C++ for the **switch** statement.

In VBA we exist when the **first** true conditional is found – be careful how you order.
Looping

It is often desirable to “loop” or “iterate” over a range, an array, or variable values. We will study:

- For Loops
- Do Loops

The general construct of the For loop is:

```
For counter = start To end [Step step]
    [statements]
    [Exit For]
    [statements]
Next [counter]
```

Use VBA Editor Help Function
Looping

Use VBA Editor Help Function – Illustration
For Loops

The components of the For loop

- **For**  VBA keyword to start loop – required
- counter – gets incremented at each step – required
- start – initial value of counter – required
- **To**  VBA keyword – required
- end – final value of counter – required
- **Step**  VBA keyword – optional
- step – the amount by which the counter is incremented (1 by default) – optional
- **Exit For**  VBA keywords – optional
- **Next**  VBA keyword – required
- counter – optional but use it to close loop
For Loops

See *programminglogic.xls*

Sub ForEx1()
    'Illustrate the For loop
    Dim i As Integer
    Dim total As Double
    total = 0
    For i = 1 To 20
        total = total + i ^ 2
    Next i
    MsgBox "Total = " & total
    total = 0
    For i = 1 To 20 Step 2
        total = total + i ^ 2
        Debug.Print i
    Next i
End Sub
For Loops

Important Concepts:

▶ In the For loop, the statement

\[ \text{total} = \text{total} + i^2 \]

is **not** an equality. It is an **assignment** statement (contrast with `==` in C++). It replaces the memory location that stores the variable `total` with the value that is currently in this memory location plus the value of \(i^2\)

▶ the **Next** \(i\) statement does two things:
  
  ▶ it increments the counter \(i\) by the amount of the step size (which is 1 by default)
  
  ▶ it returns control to the beginning of the loop (the For statement)
For Loops

- You can nest loops
- Get values from a named range

'Illustrate nested For loop
'Also illustrate getting a value from a range
'Range student_scores is 16 by 5
Dim i As Integer, j As Integer
Dim avg As Double, rng As Range
Set rng = Range("quiz_scores")
'Loop over each quiz
For i = 1 To 5
  avg = 0
  'Loop over each student
  For j = 1 To 16
    avg = avg + rng.Cells(1 + j, i).Value
  Next j
  avg = avg / 16
Next i
For Loops

Loop over values in a range of unknown size

' Illustrate looping over range of unknown size
Dim ws As Worksheet, isThere As Boolean
isThere = False
Dim i As Integer, N As Integer
Dim rng As Range, lastName As Range
Set rng = Range("names")
Set lastName = Range(rng.Cells(2, 2), _
    rng.End(xlDown).Cells(1, 2))
N = lastName.Rows.Count
For i = 1 To N
    If lastName.Cells(i, 1).Value = "Uehling" Then isThere = True
Next i
For Each Loops

**Key Idea:** loop over objects in a set of objects. For example, worksheets in a workbook or cells in a range.

Here is the syntax

```
For Each element In group
   [statements]
   [Exit For]
   [statements]
Next [element]
```

Sub ForEach()
    'Illustrate the For Each construct
    Dim ws As Worksheet, isThere As Boolean
    isThere = False
    Dim rng As Range
    Dim cell As Range
    Dim lastName As Range
    Set rng = Range("names")
    Set lastName = Range(rng.Cells(2, 2), _
            rng.End(xlDown).Cells(1, 2))
    For Each cell In lastName
        If cell.Value = "Uehling" Then
            isThere = True
            Exit For
        End If
    Next cell
End Sub
Another loop construct is the Do While (there are other variations on this theme).

The construct is

Do [{While | Until} condition]
   [statements]
   [Exit Do]
   [statements]
Loop
Sub DoWhile1()
    'Illustrate the DoWhile construct
    'Also illustrate another way to loop
    'over a range without knowing its size
    Dim ws As Worksheet
    Set ws = Worksheets("Looping")
    ws.Activate
    Dim total As Double, i As Integer
    total = 0
    i = 1
    Dim rng As Range
    Set rng = Range("A23")
    Do While IsEmpty(rng.Cells(i)) = False
        total = total + rng.Cells(i).Value
        i = i + 1
    Loop
End Sub
Do While (Until) Loops

You can take:

```vba
Do While IsEmpty(rng.Cells(i)) = False
    total = total + rng.Cells(i).Value
    i = i + 1
Loop
```

and replace with:

```vba
Do While IsNumeric(rng.Cells(i)) = True
    total = total + rng.Cells(i).Value
    i = i + 1
Loop
```

Be careful!
You can implement a For loop using a Do While

Dim counter As Integer
Dim start As Integer
Dim end As Integer
Dim step As Integer
counter = start
Do While counter <= end
    counter = counter + step
Loop
Do While (Until) Loops

You can implement a For loop using a Do While

Sub DoWhile2()
    ’Write a For loop as Do While
    Dim i As Integer
    Dim total As Double
    ’Sum the square of the integers from 1 to 20
    total = 0
    For i = 1 To 20 Step 2
        total = total + i ^ 2
    Next i
    total = 0
    i = 1
    Do While i <= 20
        total = total + i ^ 2
        i = i + 2
    Loop
End Sub