

VBA Programming Examples and Techniques

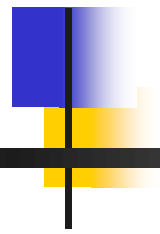


IOE 373 Lecture 11



Topics

- Using VBA to work with ranges
 - Copying/Moving
 - Counting/Looping
 - Reading/Writing to/from Arrays



Working with Ranges

- Very important for many more complex programs. Examples:
 - Selecting/copying/moving a range
 - Identifying types of information in a range
 - Prompting for a cell value
 - Finding the first empty cell in a column
 - Pausing a macro to allow the user to select a range
 - Counting cells in a range



Copying a range

- A very simple copy-and-paste operation can be done in five lines of VBA code:

```
Sub Macro1()  
Range("A1").Select  
Selection.Copy  
Range("B1").Select  
ActiveSheet.Paste  
End Sub
```



Copying a range

- Another way to approach this task is to use object variables to represent the ranges, as shown in the code that follows:

```
Sub CopyRange3()  
Dim Rng1 As Range, Rng2 As Range  
Workbooks.Open ThisWorkbook.Path & "\File1.xlsx"  
Workbooks.Open ThisWorkbook.Path & "\File2.xlsx"  
Set Rng1 =  
Workbooks("File1.xlsx").Sheets("Sheet1").Range("A1")  
Set Rng2 =  
Workbooks("File2.xlsx").Sheets("Sheet2").Range("A1")  
Rng1.Copy Rng2  
End Sub
```



Moving a range

- Very similar to copying a range
- The difference is in the use of the Cut method instead of the Copy method.
 - Note that you need to specify only the upper-left cell for the destination range.
- The following example moves 18 cells (in A1:C6) to a new location, beginning at cell H1:

```
Sub MoveRange1()  
Range("A1:C6").Cut Range("H1")  
End Sub
```



Copying a variably sized range

- In many cases, you need to copy a range of cells, but you don't know the exact row and column dimensions of the range. For example, you might have a workbook that tracks weekly sales, and the number of rows changes weekly when you add new data.
- The following macro demonstrates how to copy this range from Sheet1 to Sheet5 (beginning at cell A1). It uses the CurrentRegion property, which returns a Range object that corresponds to the block of cells around a particular cell (in this case, A1).

```
Sub CopyCurrentRegion2()  
Range("A1").CurrentRegion.Copy Sheets("Sheet5").Range("A1")  
End Sub
```

- **Generally, the CurrentRegion property setting consists of a rectangular block of cells surrounded by one or more blank rows or columns.**



Selecting various types of ranges

- In addition to the CurrentRegion property, we can use the End method of the Range object.
 - The End method takes one argument, which determines the direction in which the selection is extended. The following statement selects a range from the active cell to the last non-empty cell:
`Range(ActiveCell, ActiveCell.End(xlToRight)).Select`
 - A similar example that uses a specific cell as the starting point:
`Range(Range("A20"), Range("A20").End(xlUp)).Select`



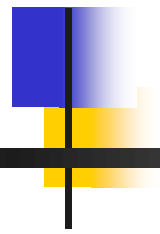
Pausing a macro to get a user-selected range

- In some situations, you may need an interactive macro.
 - For example, a macro that pauses while the user specifies a range of cells. This example describes how to do this with Excel's InputBox method.
 - Don't confuse Excel's InputBox method with VBA's InputBox function. Although these two items have the same name, they're not the same.



Example

```
Sub GetUserRange()  
    Dim UserRange As Range  
    Prompt = "Select a range for the random numbers."  
    Title = "Select a range"  
    ` Display the Input Box  
    Set UserRange = Application.InputBox( Prompt:=Prompt, Title:=Title, _  
    Default:=ActiveCell.Address, Type:=8) `Range selection  
    ` Was the Input Box canceled?  
    If UserRange Is Nothing Then  
        MsgBox "Canceled."  
    Else  
        UserRange.Formula = "=RAND()"  
    End If  
End Sub
```



Counting selected cells

- You can create a macro that works with the range of cells selected by the user. Use the Count property of the Range object to determine how many cells are contained in a range selection. For example:

`MsgBox Selection.Count`

- 
-
- If the active sheet contains a range named data, the following statement assigns the number of cells in the data range to a variable named CellCount:

`CellCount = Range("data").Count`

- We can also determine how many rows or columns are contained in a range:

`Selection.Rows.Count`


- We can also use the Rows property to determine the number of rows in a range. The following statement counts the number of rows in a range named data and assigns the number to a variable named RowCount:

`RowCount = Range("A1:B3").Rows.Count`

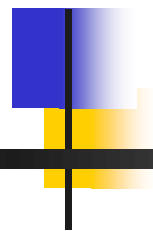


Looping through a selected range

- A common task is to create a macro that evaluates each cell in a range and performs an operation if the cell meets a certain criterion. The procedure that follows is an example of such a macro.
- The procedure sets the cell's background color to red for cells that contain a negative value. For non-negative value cells, it sets the background color to none.



```
Sub ColorNegative()  
` Makes negative cells red  
Dim cell As Range  
If TypeName(Selection) <> "Range" Then Exit Sub  
`Application.ScreenUpdating = False  
For Each cell In Selection  
    If cell.Value < 0 Then  
        cell.Interior.Color = RGB(255, 0, 0)  
    Else  
        cell.Interior.Color = xlNone  
    End If  
Next cell  
End Sub
```



Deleting all empty rows

- The following procedure deletes all empty rows in the active worksheet.
- This routine is fast and efficient because it doesn't check all rows. It checks only the rows in the used range, which is determined by using the `UsedRange` property of the `Worksheet` object.

```

Sub DeleteEmptyRows()
Dim LastRow As Long
Dim r As Long
Dim Counter As Long
Dim ans As Variant
Application.ScreenUpdating = False
MsgBox "Number of used rows: " & ActiveSheet.UsedRange.Rows.Count
LastRow = ActiveSheet.UsedRange.Rows.Count + _
ActiveSheet.UsedRange.Rows(1).Row - 1
MsgBox "First Row: " & ActiveSheet.UsedRange.Rows(1).Row
MsgBox "Last Row: " & LastRow
ans = MsgBox("Continue?", vbYesNo)
If ans = vbNo Then Exit Sub
For r = LastRow To 1 Step -1
    If Application.WorksheetFunction.CountA(Rows(r)) = 0 Then
        Rows(r).Delete
        Counter = Counter + 1
    End If
Next r
Application.ScreenUpdating = True
MsgBox Counter & " Empty rows were deleted."
End Sub

```




Determining a cell's data type

- Excel provides a number of built-in functions that can help determine the type of data contained in a cell. These include ISTEXT, ISLOGICAL, and ISERROR. In addition, VBA includes functions such as IsEmpty, IsDate, and IsNumeric.
- The following function, named CellType, accepts a range argument and returns a string (Blank, Text, Logical, Error, Date, Time, or Number) that describes the data type

```
Function CellType(Rng) As String
` Returns the cell type of the upper left
` cell in a range
Dim TheCell As Range
Set TheCell = Rng.Range("A1")
Select Case True
Case IsEmpty(TheCell)
CellType = "Blank"
Case Application.IsText(TheCell)
CellType = "Text"
Case Application.IsLogical(TheCell)
CellType = "Logical"
Case Application.IsErr(TheCell)
CellType = "Error"
Case IsDate(TheCell)
CellType = "Date"
Case InStr(1, TheCell.Text, ":") <> 0
CellType = "Time"
Case IsNumeric(TheCell)
CellType = "Number"
End Select
End Function
```



Reading and writing ranges

- Many VBA tasks involve transferring values either from an array to a range or from a range to an array. Excel reads from ranges much faster than it writes to ranges because the latter operation involves the calculation engine.
- The following example creates an array and uses For-Next loops to write the array to a range and then read the range back into the array. It calculates the time required for each operation by using the Excel Timer function.

```

Sub WriteReadRange()
Dim MyArray()
Dim Time1 As Double
Dim NumElements As Long, i As Long
Dim WriteTime As String, ReadTime As String
Dim Msg As String
NumElements = 60000
ReDim MyArray(1 To NumElements)
` Fill the array
For i = 1 To NumElements
MyArray(i) = i
Next i
` Write the array to a range
Time1 = Timer
For i = 1 To NumElements
Cells(i, 1) = MyArray(i)
Next i
WriteTime = Format(Timer - Time1, "00:00")
` Read the range into the array
Time1 = Timer
For i = 1 To NumElements
MyArray(i) = Cells(i, 1)
Next i
ReadTime = Format(Timer - Time1, "00:00")
` Show results
Msg = "Write: " & WriteTime
Msg = Msg & vbCrLf
Msg = Msg & "Read: " & ReadTime
MsgBox Msg, vbOKOnly, NumElements & " Elements"
End Sub

```



Faster Procedure

- The example that follows demonstrates a much faster way to produce the same result. This code inserts the values into an array and then uses a single statement to transfer the contents of an array to the range.

```

Sub ArrayFillRange()
` Fill a range by transferring an array
Dim CellsDown As Long, CellsAcross As Integer
Dim i As Long, j As Integer
Dim StartTime As Double
Dim TempArray() As Long
Dim TheRange As Range
Dim CurrVal As Long
` Get the dimensions
CellsDown = InputBox("How many cells down?")
If CellsDown = 0 Then Exit Sub
CellsAcross = InputBox("How many cells across?")
If CellsAcross = 0 Then Exit Sub
` Record starting time
StartTime = Timer
` Redimension temporary array
ReDim TempArray(1 To CellsDown, 1 To CellsAcross)
` Set worksheet range
Set TheRange = ActiveCell.Range(Cells(1, 1), _
Cells(CellsDown, CellsAcross))
` Fill the temporary array
CurrVal = 0
Application.ScreenUpdating = False
For i = 1 To CellsDown
For j = 1 To CellsAcross
TempArray(i, j) = CurrVal + 1
CurrVal = CurrVal + 1
Next j
Next i
` Transfer temporary array to worksheet
TheRange.Value = TempArray
` Display elapsed time
Application.ScreenUpdating = True
MsgBox Format(Timer - StartTime, "00.00") & " seconds"
End Sub

```



Transferring a range to a variant array

- Another way to work with data by transferring a range of cells to a two-dimensional variant array.
 - Message boxes display the upper bounds for each dimension of the variant array.

```
Sub RangeToVariant()
```

```
Dim x As Variant
```

```
Range("A1").CurrentRegion.Select
```

```
x = Selection.Value
```


```
'x = Range("A1:Y600").Value
```


```
MsgBox "Number of rows in variant x: " & UBound(x, 1)
```

```
MsgBox "Number of columns in variant x: " & UBound(x, 2)
```

```
Range("Data3") = x
```

```
End Sub
```

- 
-
- Transferring the range data to a variant array is virtually instantaneous.
 - The following example reads a range (named data2) into a variant array, performs a simple multiplication operation on each element in the array, and then transfers the variant array back to the range:



```
Sub RangeToVariant2()  
Dim x As Variant  
Dim r As Long, c As Integer  
` Read the data into the variant  
x = Range("data2").Value  
` Loop through the variant array  
For r = 1 To UBound(x, 1)  
For c = 1 To UBound(x, 2)  
` Multiply by 2  
x(r, c) = x(r, c) * 2  
Next c  
Next r  
` Transfer the variant back to the sheet  
Range("data2") = x  
End Sub
```

- This procedure runs amazingly fast. Working with 30,000 cells took less than one second.