VBA Summary

Overview:

VBA is a very powerful feature of Microsoft applications such as Excel. Below are properties, terminology, and other useful features of VBA.

Document Quick-Links:

VBA Summary:

1. **Code:** You perform actions in VBA by executing VBA code. You write (or record) VBA code, which is stored in a VBA module.
2. **Module:** VBA modules are stored in an Excel workbook file, but you edit a module by using Visual Basic Editor (VBE). A VBA module consists of procedures.
3. **Procedures:** A procedure is basically a unit of computer code that performs some action. VBA supports two types of procedures: *Sub procedures* and *Function procedures*.
    - **Sub:** A Sub procedure consists of a series of statements and can be executed in a number of ways.
    - **Function:** A Function procedure returns a singe value (or possibly from an Array). A Function can be called from another VBA procedure or used in a worksheet formula.
4. **Objects:** Workbook, worksheet, chart, range, shape, etc.
5. **Collections:** Like objects form a collection. For example, the Worksheets collection consists of all the worksheets in a particular workbook. Collections are objects in themselves.
6. **Object Hierarchy:** When you refer to an object, you specify its position in the object hierarchy by using a period (dot) as a separator between the container and its member.
    - For example, you can refer to a workbook named Book1.xlsx as Application.Workbooks("Book1.xlsx"). This code refers to the Book1.xlsx workbook in the Workbooks collection.
    - Another example is referring to Sheet1 in Book1 as Application.Workbooks("Book1.xlsx").Worksheets("Sheet1").
    - A step further is referring to a specific cell: Application.Workbooks("Book1.xlsx").Worksheets("Sheet1").Range("A1").
7. **Active Objects:** If you omit a specific reference to an object, Excel uses the active objects.

    - If Book1 is the active workbook, the preceding reference can be simplified as Worksheets("Sheet1").Range("A1").
    - If you know that Sheet1 is the active worksheet, you can simplify the reference even more: Range("A1").
8. **Objects properties:** Objects have *properties*. Just like VB, a property can be thought of as a *setting* for an object. For example, a range object has properties such as Value and Address. A chart object has properties such as HasTitle and Type. You can use VBA to determine object properties and also to change them. You refer to properties by combining the object with the property, separated by a period. For example, you can refer to the value in cell A1 on Sheet1 as Worksheets("Sheet1").Range("A1").Value.
9. **VBA Variables:** You can assign values to VBA variables. To assign the value in cell A1 on Sheet1 to a variable called interest, you would use Interest = Worksheets("Sheet1").Range("A1").Value.
10. **Object methods:** Objects have *methods*. A method is an action that is performed with the object. For example, one of the methods for a Range object is *ClearContents*. This method clears the contents of the range. For example, to clear the contents of cell A1 on the active worksheet, use Range("A1").ClearContents.
11. **Standard programming constructs:** VBA also includes many familiar constructs such as arrays, conditional statements, and loops.
12. **Events:** Some objects recognize specific events, and you can write VBA code that is executed when the event occurs. For example, opening a workbook triggers a *Workbook_Open* event. Changing a cell in a worksheet triggers a *Worksheet_Change* event.

Top of Page

Application Object Properties:

The table below lists application properties that are useful when working with cells and ranges followed by several examples.

Property	Object Returned
ActiveCell	The active cell.
ActiveChart	The active chart sheet or chart contained in a ChartObject on a worksheet. This property is Nothing if a chart isn't active.
ActiveSheet	The active sheet (worksheet or chart sheet).
ActiveWindow	The active window
ActiveWorkbook	The active workbook
Selection	The object selected. It could be a Range object, Shape, ChartObject, and so on.
ThisWorkbook	The workbook that contains the VBA procedure being executed. This object may or may not be the same as the ActiveWorkbook object.

The advantage of using the properties to return an object is that you don't need to know which cell, worksheet, or workbook is active, and you don't need to provide a specific reference to it. This allows you to write VBA code that isn't specific to a particular workbook, sheet, or range. For example, the following instruction clears the contents of the active cell, even though the address of the active cell isn't known: ActiveCell.ClearContents.

This example displays a message that tells you the name of the active sheet: MsgBox ActiveSheet.Name.

If you want to know the name and directory path of the active workbook, use a statement like this: MsgBox ActiveWorkbook.FullName.

If you have a range selected, the following statement will fill all cells in the range with the value of 12: Selection.Value = 12.

To find out how many cells are selected in the active window, access the *Count* property: MsgBox ActiveWindow.RangeSelection.Count.

Top of Page

The Range Object:

Much of the work you do in VBA involves cells and ranges in worksheets. The Range object is contained in a Worksheet object and consists of a single cell or range of cells on a single worksheet. You typically refer to Range objects in your VBA code with the Range, Cells, and Offset properties.

The Range Property:

The Range property returns a range object. If you consult the Help system for the Range property, you learn that this property has two syntaxes:

- object.Range(cell1)
- object.Range(cell1, cell2)

The Range property applies to two different types of objects: a Worksheet or a Range object. The following statement simply enters a value into the specified cell. In this case, it puts the value 12.3 into cell A1 on the Sheet1 of the active workbook: Worksheets("Sheet1").Range("A1").Value = 12.3.

The Range property also recognizes defined names in workbooks. If a cell is named *Input*, you can use the following statement to enter a value into that named cell: Worksheets("Sheet1").Range("Input").Value = 100.

To enter the same value in a range of 20 cells on the active sheet, you would use ActiveSheet.Range("A1:B10").Value = 2. Assuming the worksheet is activated, you could shorten it to Range("A1", "B10") = 2.

To enter the value of 4 to several cells in a non-contiguous range, you use the comma as the union operator: Range("A1,A3,A5,A7,A9") = 4.

Top of Page

The Cells Property:

Another way to reference a range is to use the Cells property. Like the Range property, you can use the Cell property on Worksheet and Range objects. There are three syntaxes:

- object.Cells(rowIndex, columnIndex)
- object.Cells(rowIndex)
- object.Cells

The following statement enters the value 9 in cell A1 on Sheet1. This example is from the first syntax: Worksheets("Sheet1").Cells(1, 1) = 9.

This example enters the value 7 in cell D3 (that is, row 3, column 4) in the active worksheet: ActiveSheet.Cells(3, 4) = 7.

To enter a value of 5 in the cell directly below the active cell, you can use ActiveCell.Cells(2, 1) = 5. You could think of this as "Start with the active cell and consider this cell as cell A1. Place 5 in the cell in the second row and the first column."

Top of Page

The Offset Property:

The Offset property, like the Range and Cells properties, also returns a Range object. But unlike the other two methods, the Offset properties only applies to a Range object and no other class. Here's the syntax:

- object.Offset(rowOffset, columnOffset)

The Offset property takes two arguments that correspond to the relative position from the upper-left cell of the specified Range object. The arguments can be positive (down or to the right), negative (up or to the left), or 0. To enter a value of 12 into the cell directly below the active cell would be ActiveCell.Offset(1,0).Value = 12.

To enter a value of 15 in the cell directly above the active cell: ActiveCell.Offset(-1,0).Value = 15.

Top of Page

With…End Statement:

Notice in the preceding examples that an object may be repeated several times when changing its properties. For example, if you selected a range and wanted to change the font properties, the code might be:

ActiveSheet.Range("A1:B10").Select

Selection.Font.Bold = True
Selection.Font.Color = vbBlue
Selection.Font.Name = "Arial"
Selection.Font.Size = 22
Selection.Font.Italic = True

Using the **With...End With** statement allows you to skip repeat objects up to the property value. For example the code would be:

With Selection.Font

.Bold = True
.Color = vbBlue
.Name = "Arial"
.Font.Size = 22
.Italic = True

End With

The syntax for the With...End With statement is:

With *object*

.property

End With

Top of Page

Form vs. ActiveX Controls:

A few words about **Form Controls** versus **ActiveX Controls**:

You use ***Form controls*** when you want to easily reference and interact with cell data without using VBA code. For example, adding a form control button allows you record a macro and assign it to the button without ever opening the VBE. However, these controls cannot be added to UserForms, used to control events, or modified to run Web scripts on Web pages.

ActiveX controls can be used on worksheet forms, with or without the use of VBA code, and on VBA UserForms. In general, use ActiveX controls when you need more flexible design requirements than those provided by Form controls. ActiveX controls have extensive properties that you can use to customize their appearance, behavior, fonts, and other characteristics.

Important: Not all ActiveX controls can be used directly on worksheets; some can be used only on VBA UserForms. If you try to add any one of these particular ActiveX controls to a worksheet, Excel displays the message "Cannot insert object."

Top of Page