PAUL KELLY

HOW TO ACE THE 21 MOST COMMON QUESTIONS IN VBA

TABLE OF CONTENTS

INTRODUCTION

This eBook contains the answers to the most frequently asked questions about VBA. These are the questions that get repeatedly asked in places like StackOverFlow, VBA on Reddit and various VBA Forums.

The reasons these questions are asked so much is that they relate to the most common and important tasks in VBA. If your plan is to get good at VBA then being able to ace these questions will put you ahead of most VBA users.

The answers here come with examples that you can try for yourself. The best way to learn VBA or any programming language is to practise writing code. Take these examples and run them. Then make changes and run them again. Doing this will allow you to see how the code works.

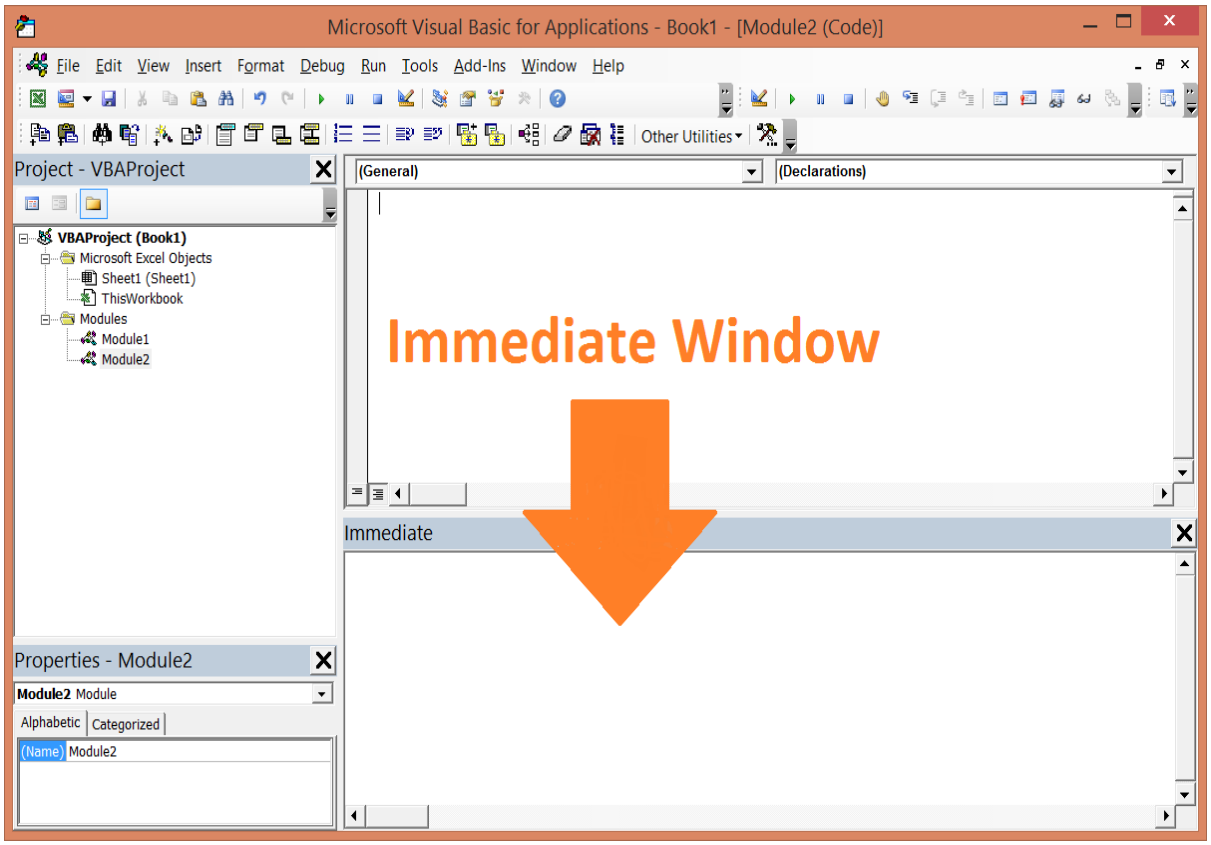
If you have any questions about the contents of this eBook or VBA in general then please feel free to email me at paulkellykk@gmail.com or leave a comment on my blog at Excel Macro Mastery.

1 WHERE DOES DEBUG.PRINT WRITE TO?

Debug.Print is very useful for showing how code works. Many people get confused to where the output ends up.

It goes to the Immediate Window. To view this Window select **View->Immediate Window** form the menu. The shortcut keys is **Ctrl + G**.

Note: Debug.Print will still write values to the window even if it is not visible



2 HOW TO OPEN A CLOSED WORKBOOK

This is a common task to perform in VBA. You can use the *Workbooks.Open* function which takes the name, of the workbook to open, as an argument

```
1  Workbooks.Open "C:\Docs\Example.xlsx"
```

Of course you will normally want to do something with the workbook when you open it. You can access it using the workbooks collection. The following code prints out the name of the open workbook, “Example.xlsx”.

```
1  Debug.Print Workbooks("Example.xlsx").Name
```

A much neater way to use a workbook is by declaring a workbook variable

```
1  ' Declare a workbook variable called wk
2  Dim wk As Workbook
3
4  ' Set the variable to the workbook that is opened
5  Set wk = Workbooks.Open("C:\Docs\Example.xlsx")
6
7  ' Print the name
8  Debug.Print wk.Name
```

If you would like more information about using Workbooks I have written an in-depth post about them here

3 HOW TO FIND THE LAST ROW

To find the last row containing text, we search from the bottom of the worksheet using **xlUp.** It stops at the first cell that contains text. We can search from the top with **xlDown** but this will stop before the first blank cell it reaches.

```
Sub GetLastRow()

  ' Get the worksheet call Sheet1 in the current workbook
  Dim sh As Worksheet
  Set sh = ThisWorkbook.Worksheets("Sheet1")

  ' Get last row with text by searching from the bottom of the worksheet
  Dim lLastRow As Long
  lLastRow = sh.Cells(sh.Rows.Count, 1).End(xlUp).Row

  ' Print the row number
  Debug.Print lLastRow

End Sub
```

We get the last column with text using a similar method. In this case we search from right to left so as to find the last column with text.

```
' Get last with text by searching from the right of the worksheet
lLastCol = sh.Cells(1, sh.Columns.Count).End(xlToLeft).Column
```

4 HOW TO USE VLOOKUP

Imagine the cells A1 to B5 have the following values

Person	Score
John	53
Maria	78
Michael	12
Anne	67

To find the score for Michael we can use the following code

```
1  ' Look up John
2  Debug.Print WorksheetFunction.VLookup("John",Range("A2:B5"),2, False)
```

To look up Anne we use similar code

```
1  ' Look up Anne
2  Debug.Print WorksheetFunction.VLookup("Anne",Range("A2:B5"),2, False)
```

The arguments you use are the same as if you used the VLookup function in a formula in a worksheet.

5 HOW TO RETURN A VALUE FROM A FUNCTION

There are two types of procedures in VBA: Functions and Subs. The major difference between them is that functions return a value and subs don't. To return a value from a function you assign the value to the name of the function.

```
Function Calc(a As Long, b As Long) As Long
  ' This will return the result from this function
  Calc = (a + b) * 10
End Function
```

To get the value, returned by the function, we need to assign it to a variable.

Note: when assigning a value from a function you must have parentheses around the function arguments or you will get an error

```
Sub CallFunc()

  Dim result As Long
  ' Get value from function
  result = Calc(5, 6)

End Sub
```

If you are assigning an object(e.g. Range, Workbook etc.) to a variable you need to use the **Set** keyword. This is the same when returning an object from a function

```
Sub CallFunc2()
  Dim wk As Workbook
  ' Use Set to assign to an object
  Set wk = GetWorkbook
End Sub

Function GetWorkbook() As Workbook
  ' Use Set to return the workbook object
  Set GetWorkbook = ThisWorkbook
End Function
```

For a complete guide to Subs and Functions please **check out this** article

6 HOW TO ADD A FORMULA TO A CELL

To add a formula to a cell you use the **Formula** property of Range. You assign a formula to this property using the equals sign followed by the formula in quotes.

```
1 Worksheets(1).Range("A1").Formula ="=Sum(A2:A5)"
```

You can check if a formula is valid before using it. This is a good idea particularly if the formula is built at run time and there is a possibility it can end up being invalid. To check the formula you use the **Evaluate** and **IsError** functions together.

The following code examples show you how to use them. The function IsError returns true if the Evaluate function detects an error in a formula.

```
1 sFormula = "=Sum(A1:A5)"
2
3 If IsError(Evaluate(sFormula)) Then
4   Debug.Print "Error in formula"
5 End If
```

7 WHAT DOES "OPTION EXPLICIT" DO?

When this is placed at the top of a module it means a variable must be declared before you can use it.

In the following two examples we use a variable called *result*. The second example uses **Option Explicit** but the first one doesn't.

In the first example we can use the variable without declaring it and the code will run fine.

```
1  Sub NoOption()
2    ' no error
3    result = 5
4  End Sub
```

In the next example **Option Explicit** is used and so you will get an error message if you forget to declare you variable or misspell a variable name.

```
1  Sub UsingOption()
2    ' error – 'Option Explicit' means the variable must be declared first
3    result = 5
4  End Sub
```

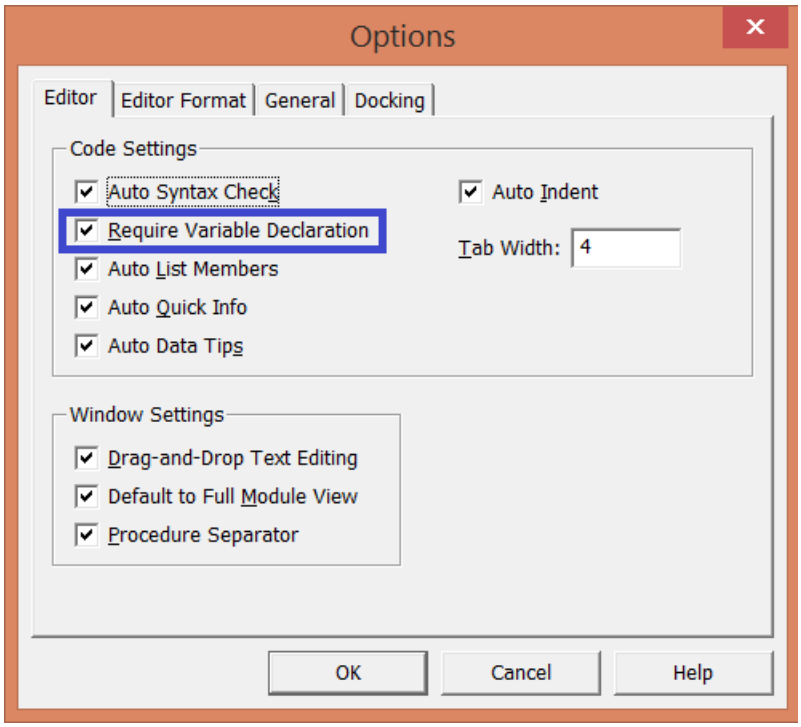
The second example is much better as we get notified straight away if we forget to declare the variable. The following example shows why this is a good idea.

```
1  Sub NoOption2()
2    Dim result As Long
3    ' Accidentally spelled result wrong
4    reesult = 5
5  End Sub
```

Here we declared a variable *result*. In the next line we spell the variable name incorrectly. Because we are not using *Option Explicit* the code will compile and we will not be notified of there is an error.

To add “Option Explicit” to all new modules turn on the “Require Variable Declaration” option.

Select **Tools->Options** from the Menu and check on the **Require Variable Declaration** option.



8 WHAT ARE THE MOST COMMON VARIABLE TYPES?

The four most common variable types are listed below.

For Item of Type	Use the VBA variable type
Text	String
Integer	Long
Decimal	Double
Currency	Currency(4 decimal places only)
Date	Date

You should avoid using the type **Integer** in VBA. It can only contain a small range of integers (-32,768 to 32,767). Instead use **Long** which can hold a much bigger range (-2,147,483,648 to 2,147,483,647).

There is another decimal type in VBA called **Currency.** It is similar to **Double** except it can only have four decimal places.

9 HOW TO ACCESS EVERY WORKSHEET

Sometimes you may need to go through all the worksheets in a workbook. You can do this using a **For Each** loop. This will always give you the order from left worksheet tab to right worksheet tab.

```
Dim sh As Worksheet
For Each sh in ThisWorkbook.Worksheets()
  Debug.Print sh.Name
Next
```

If you want to get the worksheets from right to left you can use a normal For loop

```
Dim i As Long
For i = ThisWorkbook.Worksheets.Count To 1
  Debug.Print ThisWorkbook.Worksheets(i).Name
Next
```

If you want to read a certain number of worksheets you can do this using the For loop. The following example reads the first five worksheets from the current workbook

```
Dim i As Long
For  i = 1 To 5
  Debug.Print ThisWorkbook.Worksheets(i).Name
Next
```

10 HOW TO COPY A RANGE OF CELLS

You can easily copy a range of cells using the Copy function of Range. Copying this way copies everything including formulas, formatting etc.

```
Range("A1:B4").Copy Destination:=Range("C5")
```

If you want to paste values only or formatting only you can use **PasteSpecial**

```
Range("A1:B4").Copy
Range("F3").PasteSpecial Paste:=xlPasteValues
Range("F3").PasteSpecial Paste:=xlPasteFormats
Range("F3").PasteSpecial Paste:=xlPasteFormulas
```

11 FIND TEXT IN A RANGE OF CELLS

If you want to find text in a range of cells then you can use the **Find** function of Range.

```
Dim rg As Range
Set rg = Worksheets(1).Range("A1:B10").Find("aa")

' Print address, row and column
Debug.Print rg.Address, rg.Row, rg.Column
```

To find all instances of text in a range you can use **FindNext** with **Find** as the following example shows.

```
Sub FindAllText()

  Dim rg As Range
  Set rg = Worksheets(1).Range("A1:B10").Find("aa", LookIn:=xlValues)
  If rg Is Nothing Then
    Debug.Print "Value not Found"
    Exit Sub
  End If

  Dim firstAdd As String
  firstAdd = rg.Address
  Do
    ' Print address, row and column
    Debug.Print rg.Address, rg.Row, rg.Column
    ' Find next item
    Set rg = Worksheets(1).Range("A1:B10").FindNext(rg)
  Loop Until rg Is Nothing Or firstAdd = rg.Address

End Sub
```

12 DOES VBA HAVE A DICTIONARY STRUCTURE?

VBA has a Dictionary structure which is very useful. A Dictionary is a collection where you can access an item directly using a key. It is similar to a real world dictionary where you use the word(the key) to access the definition(the value).

This has a major advantage. If you wish to access an item you do not have to read through each item in a collection until you find it.

To give an example: Imagine you store a list of ten thousand student marks. This list is regularly queried to find the marks of individual students.

If you stored these values in a normal collection then for each query you need to go through the collection until you find a matching student ID. With a Dictionary the student ID is the key and you use this to get the marks in the list.

You can also use keys with VBA Collections but they provide much less functionality for using these keys.

```
Sub Dictionary()

    ' Declare and Create
    Dim dict As Object
    Set dict = CreateObject("Scripting.Dictionary")

    ' Add item - error if already exists
    dict.Add "Apples", 50

    ' Silent Add item - updates if already exists
    dict("Apples") = 100
    dict("Pears") = 67

    ' Access Item stored at Key – i.e. print 50
    Debug.Print "Apple value is:"; dict("Apples")

    ' Check item exists
    If dict.Exists("Apples") Then
        Debug.Print "Apples exist. Value is : "; dict("Apples")
    End If

    ' Loop through all items
    Dim Key As Variant
    Debug.Print "Print All items in Dictionary"
    For Each Key In dict.keys
        Debug.Print Key, dict(Key)
    Next

    ' Remove all items
    dict.RemoveAll

End Sub
```

13 HOW TO SORT A RANGE

You can use the range sort function to easily sort a range. In the following example the **rgSort** is the range to sort and the **rgCol** is the column that contains to value to sort by

```
Dim rgSort As Range, rgCol As Range
' Range to sort
Set rgSort = Worksheets(1).Range("A1:F20")
' Column to sort by
Set rgCol = Worksheets(1).Range("A1:A10")

' Sort the items
rgSort.Sort Key1:=rgCol, order1:=xlAscending
```

You can also sort by more than one column as the next example shows

```
Dim rgSort As Range, rgCol1 As Range, rgCol1 As Range
' Range to sort
Set rgSort = Worksheets(1).Range("A1:F20")

' Columns to sort by
Set rgCol1 = Worksheets(1).Range("A1:A10")
Set rgCol2 = Worksheets(1).Range("B1:B10")

' Sort the items using 2 columns
rgSort.Sort Key1:=rgCol1, order1:=xlAscending _
        , Key2:=rgCol2, orders:=xlDescending
```

14 HOW TO SUM A RANGE

The Excel worksheet functions can be very useful in VBA. You can access them using the **WorksheetFunction** object.

The following shows an example of using the Sum function to sum the values of a range.

```
Sub sumRange()

  Dim rg As Range
  Set rg = Worksheets(1).Range("A1:A10")

  Debug.Print "Sum of range is:"; WorksheetFunction.Sum(rg)

End Sub
```

By using the S*um* function in the example above, it saves you having to use a loop to add each cell value in the range. If you have to perform a task on a range always check if an existing worksheet function could perform the task. This will make your life much easier when it comes to VBA.

15 HOW TO FORMAT A RANGE

You can format a range easily using properties of the Range. The following code shows examples of the different types of formatting you can use

```
Sub FormatCells()

    With Worksheets(1).Range("A1:A10")
    ' Set Font attributes
    .Font.Bold = True
    .Font.Size = 10
    .Font.Color = rgbRed

    ' Set Fill color
    .Interior.Color = rgbLightBlue

    ' Set Borders
    .Borders.LineStyle = xlDouble
    .Borders.Color = rgbGreen
  End With

End Sub
```

16 HOW TO HIDE A ROW OR COLUMN

Hiding a row is very straightforward using VBA. The worksheet has a **Rows** collection that you can use to access a row. It takes the number of the row as an argument. By setting the **Hidden** property to true or false you can show(set hidden to false) or hide(set hidden to true) the row.

There is an equivalent **Columns** collection than works the same way for columns. The following example hides row 1 and column A in a worksheet.

```
With Worksheets(1)
  ' Hide the first row and column
  .Rows(1).Hidden = True
  .Columns(1).Hidden = True
End With
```

To a number of sequential rows or columns you use a slightly different format for the *Rows* and *Columns* arguments

```
With Worksheets(1)
  ' Hide the first 3 rows and columns
  .Rows("1:3").Hidden = True
  .Columns("1:3").Hidden = True
End With
```

To hide more than one row or column that are not sequential you need to use a loop. The following example hides every even numbered row between 1 and 20

```
With Worksheets(1)
  Dim i As Long
  ' Hide even rows from 1 to 20
  For i = 1 To 20 Step 2
    Worksheets(1).Rows(i).Hidden = True
  Next
End With
```

17 HOW TO COPY A WORKSHEET

To copy a worksheet you use the **Copy** function of the worksheet. You can use the *Before* or *After* arguments to specify where you wish to copy the worksheet.

The following code shows how to do this

```
1  ' Copy sheet to last position and rename
2  Worksheets(1).Copy After:=Worksheets(Worksheets.Count)
3  Worksheets(Worksheets.Count).name = "Report"
4
5  ' Copy sheet to first position and rename
6  Worksheets(1).Copy Before:=Worksheets(1)
7  Worksheets(1).name = "Data"
```

18 HOW TO ADD A NEW WORKSHEET

To add a new worksheet you use the **Add** function of Worksheets.

The *count* argument allows you to create more than one at a time.

The *before* and *after* arguments are the same of for the worksheet **Copy** function. They are used to specify the worksheet before or after you want to add the new worksheet(s). Remember you can only use either *before* or *after* as an argument. You cannot use them together.

```
' Add a new sheet
Dim sh As Worksheet
Set sh = Worksheets.Add (Before:=Worksheets(1))

' Rename the sheet
sh.Name = "Accounts"
```

The following code shows examples of adding two new worksheets and renaming them

```
' Add 2 new sheets
Worksheets.Add Before:=Worksheets(1), Count:=2

' Rename the sheets
Worksheets(1).name = "NewSheet1"
Worksheets(2).name = "NewSheet2"
```

19 HOW TO CREATE A NEW WORKBOOK

Creating a new workbook is similar to creating a new worksheet. In this case you use the **Add** function of Workbooks. You can base the new workbook on a template but most of the time you will use add without an argument.

In the following code we create a new workbook and then save it with a given name.

```
' Add new workbook
Dim bk As Workbook
Set bk = Workbooks.Add

' Save file
bk.SaveAs "C:\temp\Reports.xlsx"
```

20 HOW TO INSERT A ROW OR COLUMN

We previously used the **Rows** and **Columns** collections of the worksheet to hide rows or columns. These collections also have an **Insert** function that allows you to insert a row or column.

```
' Insert new row before first row
Worksheets(1).Rows(1).Insert

' Insert new column before first column
Worksheets(1).Columns(1).Insert
```

To insert more than one we change the format slightly. The number of rows you select is the number of rows that get inserted. In the following example we insert rows 1 to 3 which inserts 3 rows before the first 3 rows.

```
' Insert 3 rows before rows 1:3
Worksheets(1).Rows("1:3").Insert

' Insert 3 columns before columns 1:3
Worksheets(1).Columns(1:3).Insert
```

To insert more than one row or column that is not sequential we use a for loop as the following example shows

```
Dim i As Long

' Insert a row at every third row
For i = 1 To 21 Step 3
  Worksheets(1).Rows(1).Insert
Next i
```

21 WHAT IS THE DIFFERENCE BETWEEN RANGE, CELLS AND OFFSET

When you are accessing cells you can use the Range or Cells property of the worksheet. Range has a property *Offset* which can also be useful for accessing cells.

Using Range

Range takes an argument similar to Sum, Count functions in Excel. Examples of these are “A1” or “B2:B7” or “C1:C7,D6”.

You can also set the row of the Range using a number or variable e.g. Range(“A” & 1) or Range(“A” & row).

Ranges are best used when the macro will use the same column each time.

```
With Worksheets("sheet1")

  ' Write values to ranges
  .Range("A1") = 6
  .Range("C1:C7,D6") = 99.99

  ' Read from range - range can only be one cell
  Dim lVal As Long
  lVal = .Range("A2")
  Debug.Print lVal

End With
```

Using Cells

The Cells property takes two arguments. The row number and the column number. This means you can use a number to set the row and column at run time.

Some examples are Cells(1,1) is A1, Cells(10,1) is A10, Cells(2,5) is E2.

Cells only returns one cell. If you want to return a range of cells you can use it with range e.g. Range(Cells(1,1),Cells(3,3)) gives the Range of cells A1 to C3.

```
With Worksheets("sheet1")

  ' Write value to cell A1
  .Cells(1, 1) = 6
  ' A3
  .Cells("3,1") = 67
  ' C1
  .Cells("1,3") = 56

  ' Read from A2
  Dim lVal As Long
  lVal = .Cells(2, 1)
  Debug.Print lVal

End With
```

Using Offset

Offset is a property of Range. As the number suggests it counts a number of cells from the original Range. It takes two parameters: Row offset and Column Offset.

The following example shows how to use offset

Range(“A1”).Offset(1,0) returns the range A2

Range(“A1”).Offset(0,1) returns the range B1

Range(“B2:C3”).Offset(2,2) returns the range D4:E5

```
With Worksheets("Sheet1")

  ' Write value to cell B1
  .Range("A1").Offset(1, 0) = 89
  ' Set colour of D4:E5 to red
  .Range("B2:C3").Offset(2, 2).Interior.Color = rgbRed

End With
```

The following table gives a quick overview of these three properties

Property	You provide	It returns a range of	Example
Range	Cell Address	Multiple Cells	.Range("A1:A4")
Cells	Row , Column	One Cell	.Cells(1,5)
Offset	Row , Column	Multiple Cells	.Offset(1,2)

For an in-depth guide to Cells, Ranges and Offsets please check out my article on The Complete Guide to Ranges and Cells in Excel VBA

CONCLUSION

I hope you found these answers and code examples useful. There is tons more VBA resources on my blog. If you have any queries about anything in this eBook or other VBA questions then please feel free to email me at mailto:paullkellykk@gmail.com.

INDEX

