The Ultimate VBA Tutorial Part One

MARCH 16, 2017 (HTTPS://EXCELMACROMASTERY.COM/VBA-TUTORIAL-1/) BY PAUL KELLY (HTTPS://EXCELMACROMASTERY.COM/AUTHOR/ADMIN/) · 59 COMMENTS (HTTPS://EXCELMACROMASTERY.COM/VBA-TUTORIAL-1/#COMMENTS)

“The noblest pleasure is the joy of understanding.” – Leonardo da Vinci

Welcome to part one of the Ultimate VBA Tutorial.

If you are brand new to VBA, then make sure that you have read the post How To Create a Macro From Scratch in Excel (https://excelmacromastery.com/excel-macro-create/) so that your environment is set up correctly to run macros.

In this tutorial you will learn how to create real-world macros. The focus is on learning by doing. This tutorial has coding examples and activities to help you on your way. You will find a quiz at the end of the tutorial. You can use this to test your knowledge and see how much you have learned.

In part one of the tutorial we will concentrate on the basics of creating Excel macros. See the next sections for the learning outcomes and for tips on getting started with VBA.

Contents [hide]

adbox/145db6b73f72a2%3A106f25298346dc/5676073085829120/)

Learning Outcomes

When you finish this tutorial you will be able to:

1. Create a module
2. Create a sub
3. Understand the difference between a module and sub
4. Run the code in a sub
5. Write a value to a cell
6. Copy the value from one cell to another
7. Copy values from one range of cells to another
8. Copy values between difference worksheets
9. Test your output using the Immediate Window
10. Write code faster using the With Statement

11. Create and use variables
12. Copy from a cell to a variable and vice versa

Before we get started, let's look at some simple tips that will help you on your journey.

6 Tips For Learning VBA

1. Practice, Practice, Practice – Don't try to learn by reading. Try the examples and activities.
2. Type the code examples instead of copying and pasting – this will help you understand the code better.
3. Have a clearly defined target for learning VBA. One you will know when you reach.
4. Don't be put off by errors. They help you write proper code.
5. Start by creating simple macros for your work. Then create more complex ones as you get better.
6. Don't be afraid to work through each tutorial more than once.The more times you do it the more deeply embedded the knowledge will become.

Basic Terms and What They Mean

Excel Macros: A macro is a group of programming instructions we use to create automated tasks.

VBA: VBA is the programming language we use to create macros. It is short for Visual Basic for Applications.

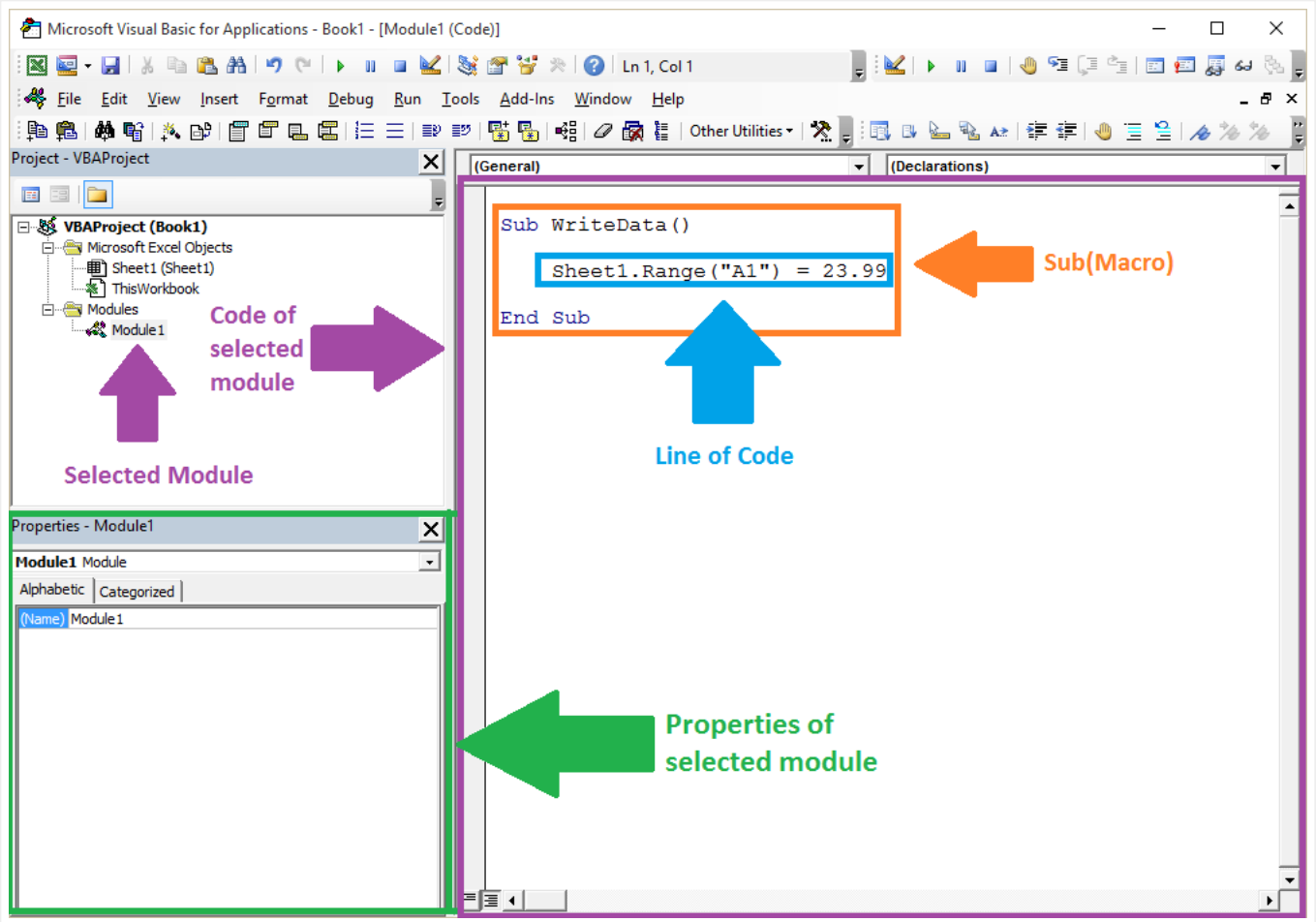
Line of code: This a VBA instruction. Generally speaking, they perform one task.

Sub: A sub is made up of one or more lines of code. When we "Run" the sub, VBA goes through all the lines of code and carries out the appropriate actions. A macro and a sub are essentially the same thing.

Module: A module is simply a container for our subs. A module contains subs which in turn contain lines of code. There is no limit(within reason) to the number of modules in a workbook or the number of subs in a module.

VBA Editor: This is where we write our code. Pressing Alt + F11 switches between Excel and the Visual Basic Editor. If the Visual Basic editor is not currently open then pressing Alt + F11 will automatically open it.

The screenshot below show the main parts of the Visual Basic Editor



The Visual Basic Editor

Tip for the Activities

When you are working on the activities it is a good idea to close all other Excel workbooks.

Creating a Module

In Excel, we use the VBA language to create macros. VBA stands for Visual Basic for Applications.

When we use the term *Excel Macros* we are referring to VBA. The term *macro* is essentially another name for a sub. Any time you see the terms Excel Macros or VBA just remember they are referring to the same thing.

In VBA we create lines of instructions for VBA to process. We place the lines of code in a sub. These subs are stored in modules.

We can place our subs in the module of the worksheet. However, we generally only place code for worksheet events here.

In VBA, we create new modules to hold most of our subs. So for our first activity let's go ahead and create a new module.

Activity 1

1. Open a new blank workbook in Excel.
2. Open the Visual Basic Editor(Alt + F11).
3. Go to the **Project – VBAProject** window on the left(Ctrl + R if it is not visible).
4. Right-click on the workbook and click *Insert* and then *Module*.
5. Click on Module1 in the **Project – VBAProject** window.
6. In the **Properties** window in the bottom left(F4 if not visible), change the module name from *module1* to MyFirstModule.

See solution to Activity 1 here...

End of Activity 1

The module is where you place your code. It is simply a container for code and you don't use it for anything else.

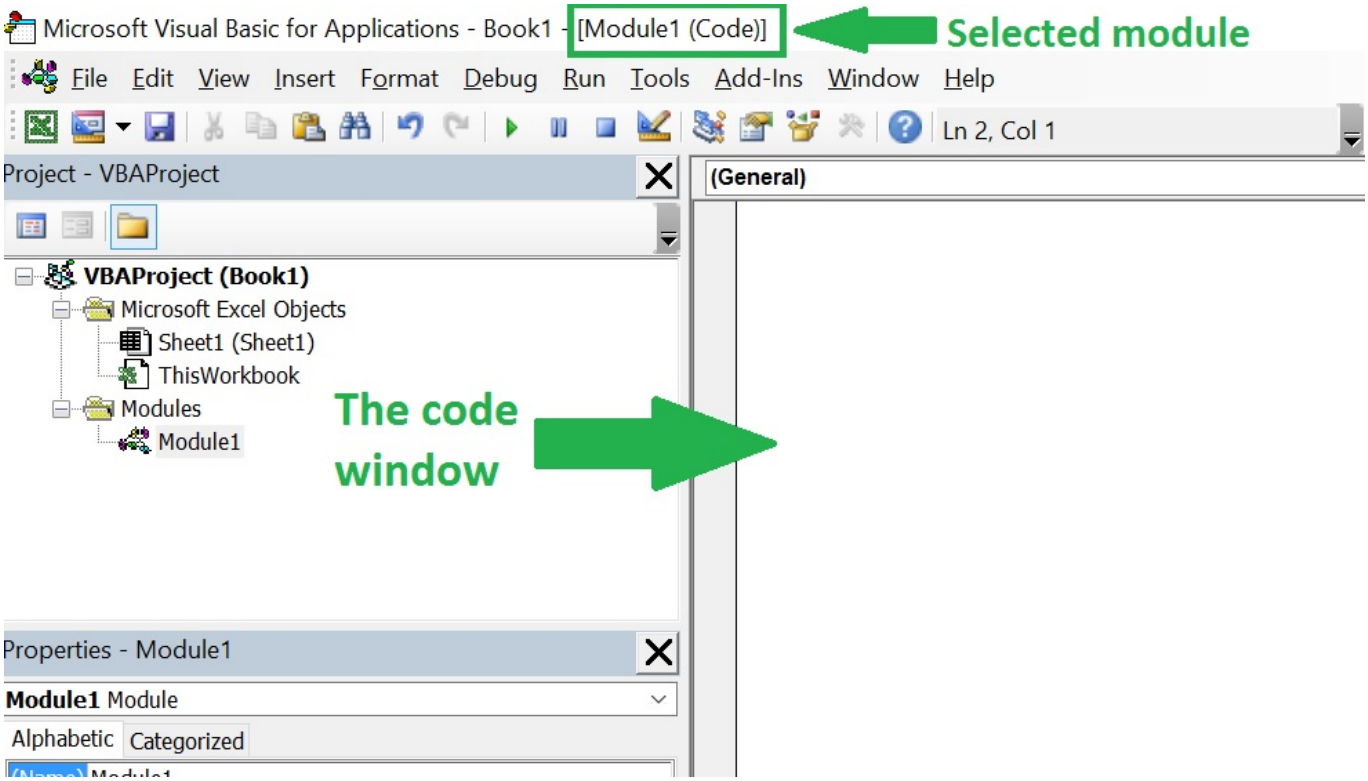
You can think of a module like a section in a bookshop. It's sole purpose is to store books and having similar books in a particular section makes the overall shop more organised.

The main window(or code window) is where the code is written. To view the code for any module including the worksheets you can double-click on the item in the *Project – VBAProject* window.

Let's do this now so you can become familiar with the code window.

Activity 2

1. Open a new workbook and create a new module like you did in the last activity.
2. Double-click on the new module in the *Project – VBAProject* window.
3. The code window for this module will open. You will see the name in the title bar of Visual Basic.



End of Activity 2

You can have as many modules as you like in a workbook and as many subs as you like within a module. It's up to you how you want to name the modules and how you organise your subs

within your modules.

How to Use Subs

A line of code is the instruction(s) we give to VBA. We group the lines of code into a *sub*. We place these subs in a module.

We create a sub so that VBA will process the instructions we give it. To do this we get VBA to *Run* the sub. When we select **Run Sub** from the menu, VBA will go through the lines of code in the sub and process them one at a time in the order they have been placed.

Let's go ahead and create a sub. Then afterwards, we will have a look at the lines of code and what they do.

Activity 3

1. Take the module you created in the last activity or create a new one.
2. Select the module by double-clicking on it in the **Project – VBAProject** window. Make sure the name is visible in the title bar.
3. Enter the following line in the code window and press enter.

```
Sub WriteValue
```

4. VBA will automatically add the second line **End Sub**. We place our code between these two lines.
5. Between these two lines enter the line

```
Sheet1.Range("A1") = 5
```

You have created a sub! Let's take it for a test drive.

6. Click in the sub to ensure the cursor is placed there. Select **Run->Run Sub/Userform** from the menu(or press F5).
   Note: If you don't place the cursor in the sub, VBA will display a list of available subs to run.
7. Open Excel(Alt + F11). You will see the value *5* in the cell *A1*.
8. Add each of the following lines to your sub, run the sub and check the results.

```
Sheet1.Range("B1") = "Some text"
Sheet1.Range("C3:E5") = 5.55
Sheet1.Range("F1") = Now
```

You should see *"Some text"* in cells *B1*, *5.55* in the cells *C3* to *E5* and the current time and date in the cell *F1*.

See solution to Activity 3 here...

End of Activity 3

Writing values to cells

Let's look at the line of code we used in the previous section

```
Sheet1.Range("A1") = 5
```

We can also write this line like this

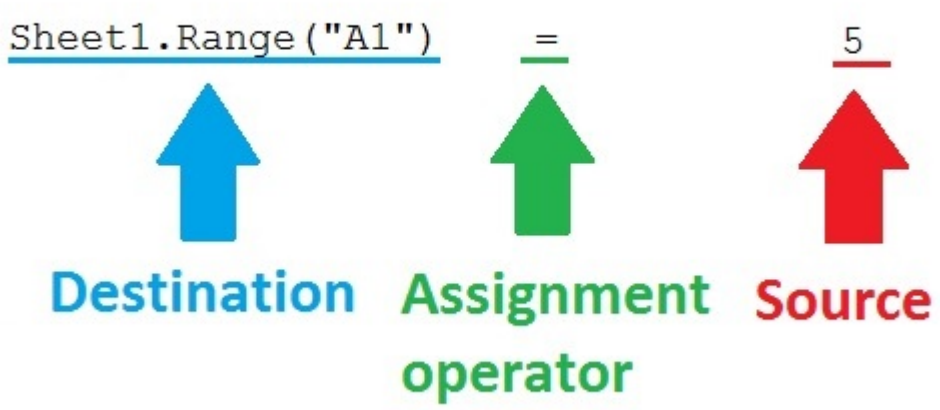
```
Sheet1.Range("A1").Value = 5
```

However in most cases we don't need to use *Value* as this is the default property.

We use lines of code like these to *assign(.i.e. copy)* values between cells and variables.

VBA evaluates the right of the *equals sign* and places the result in the variable/cell/range that is to the left of the equals.

The line is saying *"the left cell\variable\range will now be equal to the result of the item on the right"*.



Let's look the part of the code to the left of the equals sign

```
Sheet1.Range("A1") = 5
```

In this code , *Sheet1* refers to the code name of the worksheet. We can only use the code name to reference worksheets in the workbook containing the code. We will look at this in the section The code name of the worksheet.

When we have the reference to a worksheet we can use the *Range* property of the worksheet to write to a range of one or more cells.

Using a line like this we can copy a value from one cell to another.

Here are some more examples

```
Sub CopyValues()

    ' copies the value from C2 to A1
    Sheet1.Range("A1") = Sheet1.Range("C2")

    ' copies the value from D6 to A2
    Sheet1.Range("A2") = Sheet1.Range("D6")

    ' copies the value from B1 on sheet2 to A3 on sheet1
    Sheet1.Range("A3") = Sheet2.Range("B1")

    ' writes result of D1 + D2 to A4
    Sheet1.Range("A4") = Sheet2.Range("D1") + Sheet2.Range("D2")

End Sub
```

Now it's your turn to try some examples. Copying between cells is a fundamental part of Excel VBA, so understanding this will really help you on your path to VBA mastery.

Activity 4

1. Create a new Excel workbook.
2. Manually add values to the cells in sheet1 as follows: *20* to C1 and *80* to C2.
3. Create a new sub called **Act4**.
4. Write code to place the value from C1 in cell A1.
5. Write code to place the result of *C2 + 50* in cell A2.
6. Write code to multiply the values in cells C1 and C2. Place the results in cell A3.
7. Run the code. Cells should have the values A1 *20*, A2 *130* and A3 *1600*

See solution to Activity 4 here...

End of Activity 4

Cells in Different Sheets

We can easily copy between cells on difference worksheets. It is very similar to how we copy cells on the same worksheet. The only difference is the worksheet names which we use in our code.

In the next activity we are going to write between cells on different worksheets.

Activity 5

1. Add a new worksheet to the workbook from the last activity. You should now have two worksheets called which are called Sheet1 and Sheet2.
2. Create a new sub call **Act5**.
3. Add code to copy the value from C1 on *Sheet1* to cell A1 on *Sheet2*.
4. Add code to place the result from C1 + C2 on *Sheet1* to cell A2 on *Sheet2*.
5. Add code to place the result from C1 * C2 on *Sheet1* to cell A3 on *Sheet2*.
6. Run the code in the sub(F5). Cells on *Sheet2* should have the values as follows: A1 *20*, A2 *100* and A3 *1600*

See solution to Activity 5 here...

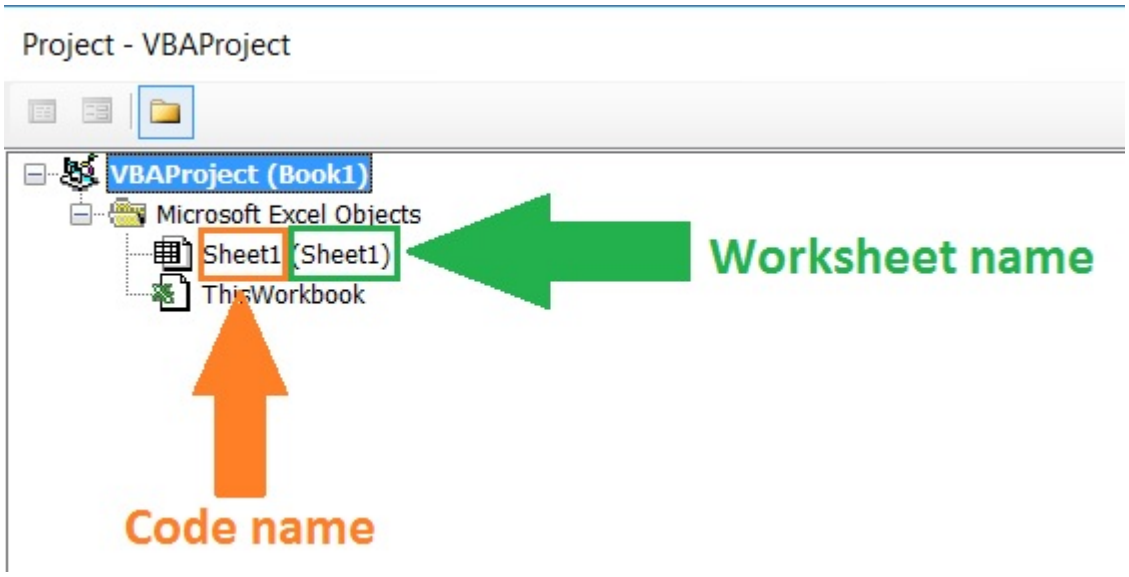
End of Activity 5

The Code Name of the Worksheet

In the activities so far, we have been using the default names of the worksheet such as *Sheet1* and *Sheet2*. It is better practice to give these sheets more meaningful names.

We do this by changing the *code name* of the worksheet. Let's look at the code name and what it is.

When you look in the *Project – VBAProject* window for a new workbook you will see *Sheet1* both inside and outside of parenthesis.



- *Sheet1* on the left is the code name of the worksheet.
- *Sheet1* on the right(in parenthesis) is the worksheet name. This is the name you see on the tab in Excel.

The **code name** has the following attributes

1. We can use it to directly reference the worksheet as we have been doing e.g.

```
Sheet1.Range("A1")
```

Note: We can only use the code name if the worksheet is in the same workbook as our code.

2. If the worksheet name is changed our code will still work if we are using the code name to refer to the sheet.

The **worksheet name** has the following attributes

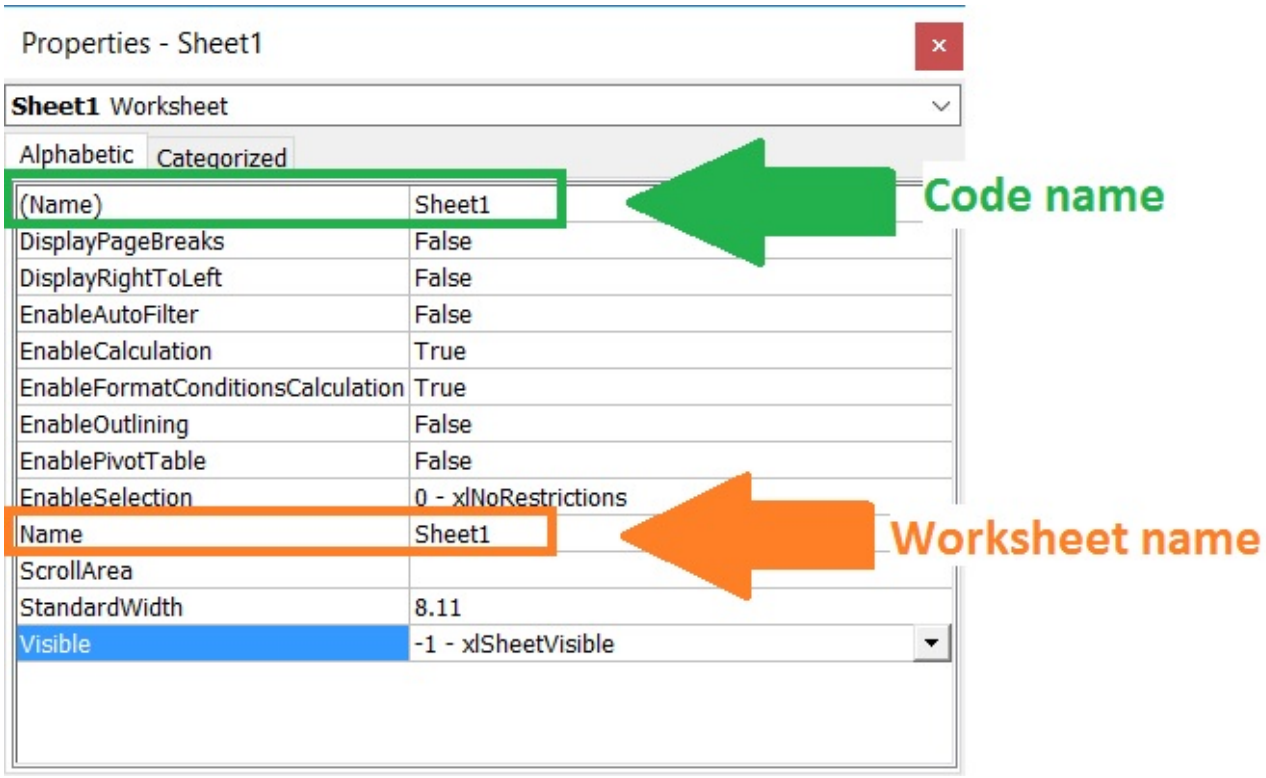
1. To reference the worksheet using the worksheet name we need to use the worksheets collection of the workbook. e.g.

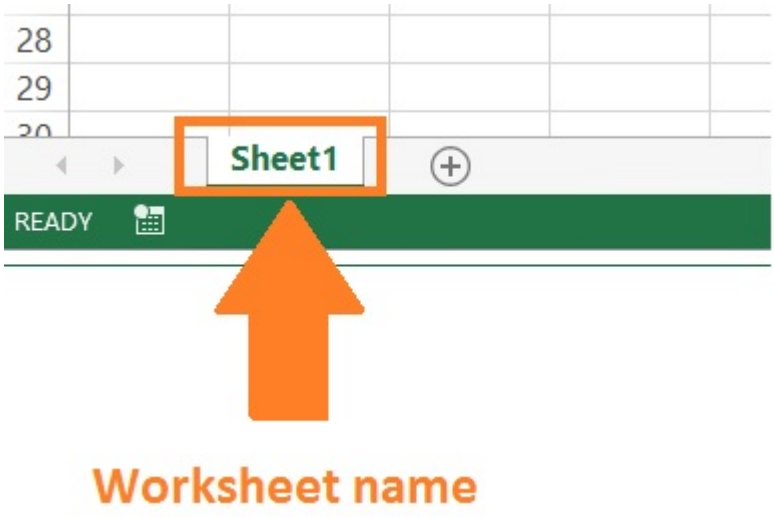
```
ThisWorkbook.Worksheets("Sheet1").Range("A1")
```

2. If the worksheet name changes then we need to change the name in our code. For example, if we changed the name of our sheet from Sheet1 to *Data* then we would need to change the above code as follows

```
ThisWorkbook.Worksheets("Data").Range("A1")
```

We can only change the code name in the **Properties** window.
We can change the worksheet name from both the worksheet tab in Excel and from the **Properties** window.





In the next activity we will change the code name of the worksheet.

Activity 6

1. Open a new blank workbook and go to the Visual Basic editor.
2. Click on *Sheet1* in the **Project – VBAProject** Window(Ctrl + R if not visible).
3. Go to the **Properties** window(F4 if not visible).
4. Change the code name of the worksheet to *shReport*.
5. Create a new module and call it **modAct6**.
6. Add following sub and run it(F5)

```
Sub UseCodename()
    shReport.Range("A1") = 66
End Sub
```

7. Then add following sub and run it(F5)

```
Sub UseWorksheetname()
    ThisWorkbook.Worksheets("Sheet1").Range("B2") = 55
End Sub
```

8. Cell A1 should now have the value *66* and cell B2 should have the value *55*.
9. Change the name of the worksheet in Excel to *Report* i.e. right-click on the worksheet tab and rename.
10. Delete the contents of the cells and run the *UseCodename* code again. The code should still run correctly.
11. Run the *UseWorksheetname* sub again. You will get the error *"Subscript out of Range"*. This crytically sounding error simply means that there is no worksheet called **Sheet1** in the worksheets collection.
12. Change the code as follows and run it again. The code will now run correctly.

```
Sub UseWorksheetname()
    ThisWorkbook.Worksheets("Report").Range("B2") = 55
End Sub
```

See solution to Activity 6 here...

End of Activity 6

The With keyword

You may have noticed that we need to use the worksheet name repeatedly – each time we refer to a range in our code.

Imagine there was a simpler way of writing the code. Where we could just mention the worksheet name once and VBA would apply to any range we used after that. The good news is we can do exactly that using the *With* statement.

In VBA we can take any item before a full stop and use the *With* statement on it. Let's rewrite some code using the *With* statement.

The following code is pretty similar to what we have been using so far

```
Sheet1.Range("A1") = Sheet1.Range("C1")
Sheet1.Range("A2") = Sheet1.Range("C2") + 50
Sheet1.Range("A3") = Sheet1.Range("C1") * Sheet1.Range("C2")
```

Let's update this code using the *With* statement

```
With Sheet1
    .Range("A1") = .Range("C1")
    .Range("A2") = .Range("C2") + 50
    .Range("A3") = .Range("C1") * .Range("C2")
End With
```

We use *With* and the worksheet to start the section. Anywhere VBA finds a full stop it knows to use the worksheet before it.

We can use the *With* statement with other types of objects in VBA including workbooks, ranges, charts and so on.

We signify the end of the *With* section by using the line *End With*.

Indenting(Tabbing) the Code

You will notice that the lines of code between the start and end *With* statments are tabbed once to right. We call this indenting the code.

We always indent the code between VBA sections that have a starting line and end line. Examples of these are as subs, the With statement, the If statement and the For loop.

You can tab the lines of code to the right by selecting the appropriate lines of code and pressing the *Tab* key. Pressing Shift and Tab will tab to the left.

Tabbing(or indenting) is useful because it makes our code more readable.

Activity 7

1. Rewrite the following code using the *With* statement. Don't forget to indent the code.

```
Sub UseWith()

Sheet1.Range("A1") = Sheet1.Range("B3") * 6
Sheet1.Cells(2, 1) = Sheet1.Range("C2") + 50
Sheet1.Range("A3") = Sheet2.Range("C3")


End Sub
```

See solution to Activity 7 here...

End of Activity 7

Copying values between multiple cells

You can copy the values from one range of cells to another range of cells as follows

```
Sheet2.Range("A1:D4") = Sheet2.Range("G2:I5").Value
```

It is very important to notice than we use the *Value* property of the source range. If we leave this out it will write blank values to our destination range.

```
' the source cells will end up blank because Value is missing
Sheet2.Range("A1:D4") = Sheet2.Range("G2:I5")
```

The code above is a very efficient way to copy values between cells. When people are new to VBA they often think they need to use some form of select, copy and paste to copy cell values. However these are slow, cumbersome and unnecessary.

It is important that both the destination and source ranges are the same size.

- If the destination range is smaller then only cell in the range will be filled. This is different to copy/pasting where we only need to specify the first destination cell and Excel will fill in the rest.
- If the destination range is larger the extra cells will be filled with **#N/A**.

Activity 8

1. Create a new blank workbook in Excel.
2. Add a new worksheet to this workbook so there are two sheets – Sheet1 and Sheet2.
3. Add the following data to the range C2:E4 on Sheet1

	C	D	E
1			
2	1	2	3
3	4	5	6
4	7	8	9

4. Write code to copy the data from Sheet1 to the range B3:D5 on Sheet2.
5. Run the code(F5).

6. Clear the results and then change the assignment range to be smaller than the source range. Run again and check the results.
7. Clear the results and then change the assignment range to be larger than the source range. Run again and check the results.

See solution to Activity 8 here...

End of Activity 8

Transposing a Range of Cells

If you need to transpose the date(convert from row to column and vice versa) you can use the **WorksheetFunction Transpose**.

Place the values 1 to 4 in the cells A1 to A4. The following code will write the values to E1 to H1

```
Sheet1.Range("E1:H1") = WorksheetFunction.Transpose(Sheet1.Range("A1:A4").Va
```

The following code will read from E1:H1 to L1:L4

```
Sheet1.Range("L1:L4") = WorksheetFunction.Transpose(Sheet1.Range("E1:H1").Va
```

You will notice that these lines are long. We can split one line over multiple lines by using the underscore(_) e.g.

```
Sheet1.Range("E1:H1") = _
    WorksheetFunction.Transpose(Sheet1.Range("A1:A4").Value)
```

```
Sheet1.Range("L1:L4") = _
    WorksheetFunction.Transpose(Sheet1.Range("E1:H1").Value)
```

We can also use a double *With* statement to make this line neater

```
With Sheet1

    With WorksheetFunction
        .Range("E1:H1") = .Transpose(.Range("A1:A4").Value)
        .Range("L1:L4") = .Transpose(.Range("E1:H1").Value)
    End With

End With
```

How to Use Variables

Variables are an essential part of every programming language.

So what are they and why do you need them?

Variables are like cells in memory. We use them to store temporary values while our code is running.

We do three things with variables

1. Declare(i.e. Create) the variable.
2. Store a value in the variable.
3. Read the value stored in the variable.

The variables types we use are the same as the data types we use in Excel.

The table below shows the common variables. There are other types but you will rarely use them. In fact you will probably use Long and String for 90% of your variables.

Type	**Details**

Boolean	Can be true or false only
Currency	same as decimal but with 4 decimal places only
Date	Use for date/time
Double	Use for decimals
Long	Use for integers
String	Use for text
Variant	VBA will decide the type at runtime

Declaring Variables

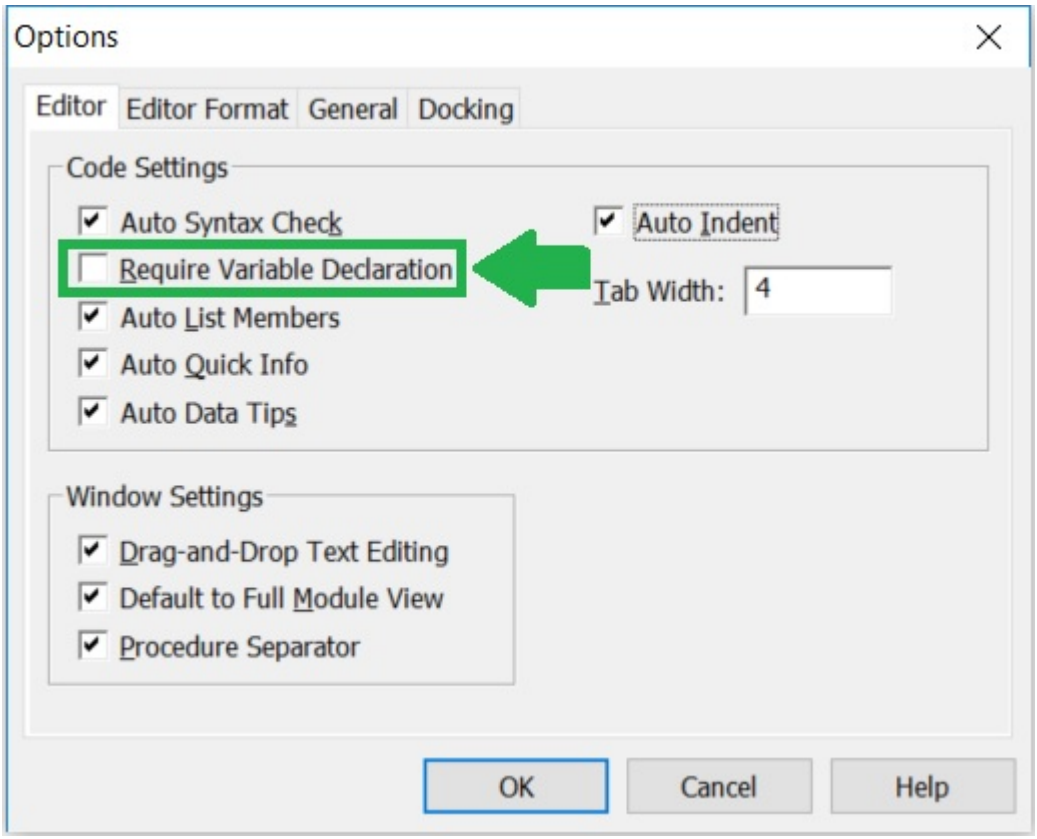
Before we use variables we should create them. If we don't then we can run into various problems.

By default, VBA doesn't make you declare variables. However, we should turn this behaviour on as it will save us a lot of pain in the long run.

To turn on *"Require Variable Declaration"* we add the following line to the top of our module

```
Option Explicit
```

To get VBA to automatically add this line, select *Tools->Options* from the menu and check *Require Variable Declaration*. Anytime you create a new module, VBA will add this line to the top.



Declaring a variable is simple. We use the format as follows

```
Dim variable_name As Type
```

We can use anything we like as the variable name. The type is one of the types from the table above. Here are some examples of declarations

```
Dim Total As Long
Dim Point As Double
Dim Price As Currency
Dim StartDate As Date
Dim CustomerName As String
Dim IsExpired As Boolean
Dim Item As Variant
```

To place a value in a variable we use the same type of statement we previously used to place a value in a cell. That is, the statement with the equals sign.

```
Dim Total As Long
Total = 1

Dim Price As Currency
Price = 29.99

Dim StartDate As Date
StartDate = #1/21/2018#

Dim CustomerName As String
CustomerName = "John Smith"
```

Activity 9

1. Create a new sub and call it **UsingVariables**.
2. Declare a variable for storing a count and set the value to *5*.
3. Declare a variable for storing the ticket price and set the value to *99.99*.
4. Declare a variable for storing a country and set the value to *"Spain"*.
5. Declare a variable for storing the end date and set the value to *21st March 2020*.
6. Declare a variable for storing if something is completed. Set the value to *False*.

See solution to Activity 9 here...

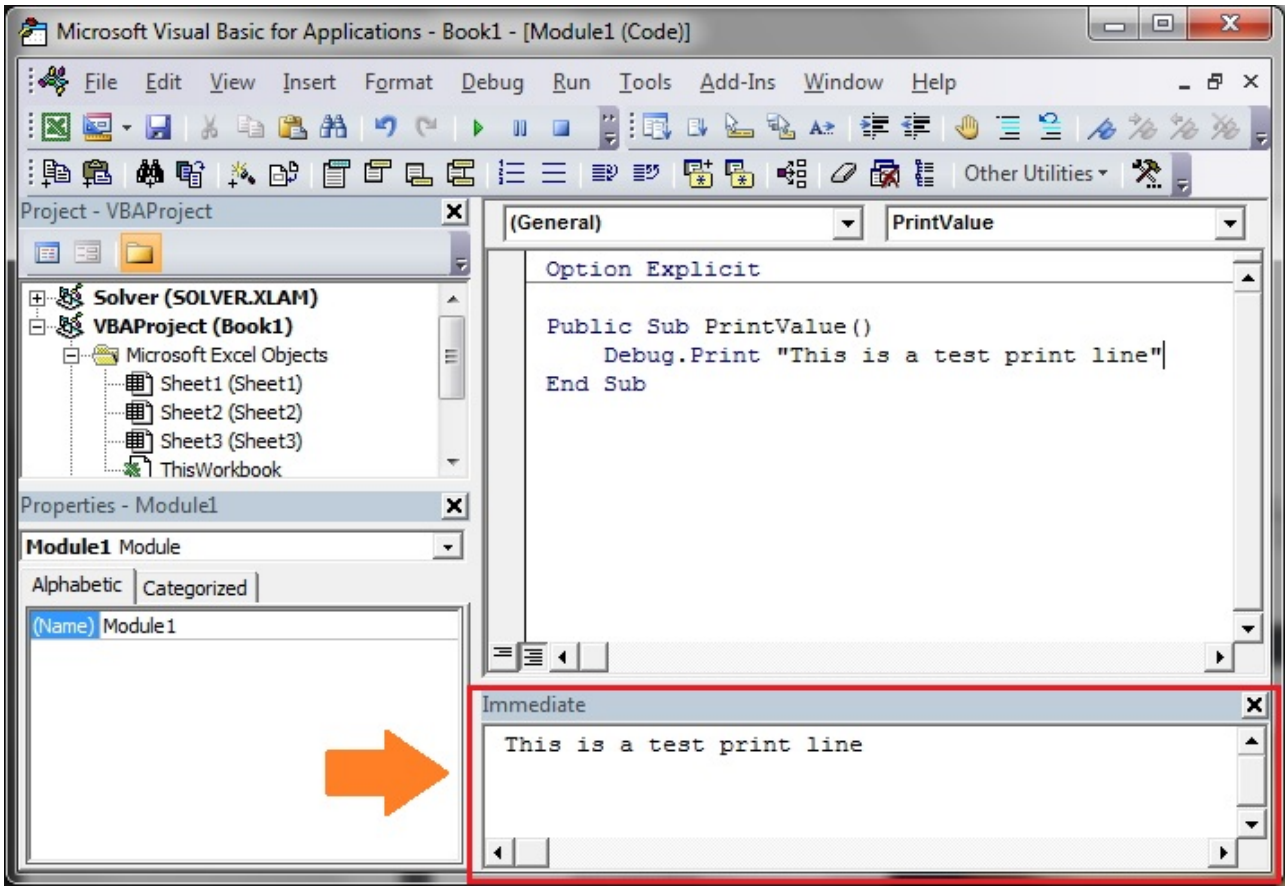
End of Activity 9

The Immediate Window

VBA has a real nifty tool that allows us to check our output. This tool is the Immediate Window. By using the **Debug.Print** we can write values, text and results of calculations to the Immediate Window.

To view this window you can select *View->Immediate Window* from the menu or press Ctrl + G.

The values will be written even if the Immediate Window is not visible.



We can use the Immediate Window to write out our variables so as to check the values they contain.

If we update the code from the last activity we can write out the values of each variable. Run the code below and check the result in the Immediate Window(Ctrl + G if not visible).

```
Sub WritingToImmediate()

    Dim count As Long
    count = 5
    Debug.Print count

    Dim ticketprice As Currency
    ticketprice = 99.99
    Debug.Print ticketprice

    Dim country As String
    country = "Spain"
    Debug.Print country

    Dim enddate As Date
    enddate = #3/21/2020#
    Debug.Print enddate

    Dim iscompleted As Boolean
    iscompleted = False
    Debug.Print iscompleted

End Sub
```

The Immediate is very useful for testing output before we write it to worksheets. We will be using it a lot in these tutorials.

Writing between variables and cells

We can write and read values between cells and cells, cells and variables,and variables and variables using the assignment line we have seen already.

Here are some examples

```
Sub VariablesCells()

    Dim price1 As Currency, price2 As Currency

    ' place value from A1 to price1
    price1 = Sheet1.Range("A1")

    ' place value from price1 to price2
    price2 = price1

    ' place value from price2 to cell b2
    Sheet1.Range("B2") = price2

    ' Print values to Immediate window
    Debug.Print "Prince 1 is " & price1
    Debug.Print "Prince 2 is " & price2

End Sub
```

Activity 10

1. Create a blank workbook and a worksheet so it has two worksheets: Sheet1 and Sheet2.
2. Place the text “New York” in cell A1 on Sheet1. Place the number 49 in cell C1 on Sheet2.
3. Create a sub that reads the values into variables from these cells.
4. Add code to write the values to the Immediate window.

See solution to Activity 10 here...

End of Activity 10

Type Mismatch Errors

You may be wondering what happens if you use an incorrect type. For example, what happens if you read the number 99.55 to a *Long*(integer) variable type.

What happens is that VBA does it best to convert the variable. So if we assign the number 99.55 to a *Long* type, VBA will convert it to an integer.

In the code below it will round the number to 100.

```
Dim i As Long
i = 99.55
```

VBA will pretty much convert between any number types e.g.

```
Sub Conversion()

    Dim result As Long

    result = 26.77
    result = "25"
    result = 24.55555
    result = "24.55"
    Dim c As Currency
    c = 23
    c = "23.334"
    result = 24.55
    c = result

End Sub
```

However, even VBA has it's limit. The following code will result in **Type Mismatch** errors as VBA cannot convert the text to a number

```
Sub Conversion()

    Dim result As Long

    result = "26.77A"

    Dim c As Currency
    c = "a34"

End Sub
```

Tip: The *Type Mismatch* error is often caused by a user accidently placing text a cell that should have numeric data.

Activity 11

1. Declare a *Double* variable type called *amount*.
2. Assign a value the causes a Type Mismatch error.
3. Run the code and ensure the error occurs.

See solution to Activity 11 here...

End of Activity 11

End of Tutorial Assignment

We've covered a lot of stuff in this tutorial. So let's put it all together in the following assignment

Tutorial One Assignment

I have created a simple workbook for this assignment. You can download it using the link below

Tutorial One Assignment Workbook (https://excelmacromastery.com/wp-content/uploads/2017/03/Tutorial-One-Assignment-Workbook.zip)

Open the assignment workbook. You will place your code here

1. Create a module and call it *Assignment1*.
2. Create a sub called **Top5Report** to write the data in all the columns from the top 5 countries to the *Top 5* section in the *Report* worksheet. This is the range starting at B3 on the **Report** worksheet. Use the *code name* to refers to the worksheets.
3. Create a sub call **AreaReport** to write all the areas size to the *All the Areas* section in the *Report* worksheet. This is the range H3:H30. Use the *worksheet name* to refer to the worksheets.
4. Create a sub called **ImmediateReport** as follows, read the area and population from Russia to two variables. Print the population per square kilometre(pop/area) to the Immediate Window.
5. Create a new worksheet and call it **areas**. Set the code name to be **shAreas**. Create a sub called **RowsToCols** that reads all the areas in D2:D11 from **Countries** worksheet and writes them to the range A1:J1 in the new worksheet **Areas**.

See solution to Tutorial One Assignment here...

End of Tutorial Assignment

The following quiz is based on what we covered this tutorial.

Tutorial One Quiz

1. What are the two main differences between the code name and the worksheet name?

   See answer to Question 1 here...

2. What is the last line of a Sub?

   See answer to Question 2 here...

3. What statement shortens our code by allowing us to write the object once but refer to it multiple times?

   See answer to Question 3 here...

4. What does the following code do?

```
Sheet1.Range("D1") = result
```

   See answer to Question 4 here...

5. What does the following code do?

```
Sheet1.Range("A1:C3") = Sheet2.Range("F1:H3")
```

   See answer to Question 5 here...

6. What is the output from the following code?

```
Dim amount As Long
amount = 7

Debug.Print (5 + 6)  * amount
```

See answer to Question 6 here...

7. What is the output from the following code?

```
Dim amt1 As Long, amt2 As Long

amt1 = "7.99"
Debug.Print amt1

amt2 = "14a"
Debug.Print amt2
```

See answer to Question 7 here...

8. If we have 1,2 and 3 in the cells A1,A2 and A3 respectively, what is the result of the following code?

```
Sheet1.Range("B1:B4") = Sheet1.Range("A1:A3").Value
```

See answer to Question 8 here...

9. What does the shortcut key Alt + F11 do?

See answer to Question 9 here...

10. In the following code we declare a variable but do not assign it a value. what is the output of the Debug.Print statement?

```
Dim amt As Long

Debug.Print amt
```

See answer to Question 10 here...

Conclusion of Tutorial One

Congratulations on finishing tutorial one. If you have completed the activities and the quiz then you will have learned some important concepts that will stand to you as you work with VBA.

In the Tutorial 2, we are going to deal with ranges where the column or row may differ each time the application runs. In this tutorial, we will cover

- How to get the last row or column with data.
- The amazingly efficient CurrentRegion property.
- How to use flexbile rows and columns.
- When to use *Range* and when to use *Cells*.
- and much more…

.

See you in Tutorial 2 (https://excelmacromastery.com/VBA-Tutorial-2)

Previous

VBA Error Handling – A Complete Guide (https://excelmacromastery.com/vba-error-handling/)

Next

CurrentRegion (Vault Code) (https://excelmacromastery.com/currentregion/)

59 COMMENTS

João Custódio

March 21, 2017 at 2:13 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-4508)

Hi Paul,
Congratulations for the clarity and objectivity of this tutorial.
Once again, a great job. 🙂

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4508#respond)

Paul Kelly

March 22, 2017 at 3:22 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4526)

Thanks João

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4526#respond)

Jakub

March 21, 2017 at 3:15 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-4512)

Great tutorial, Paul!

When I tried this line of code: Sheet1.Range(“A1”) = 5 , VBA throws “Variable not defined error”.

Any idea why? I just skip the Sheet1 or reference to the range by Worksheets(“Sheet1”), but I don’t understand why it doesn’t seem to be working.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4512#respond)

Paul Kelly

March 22, 2017 at 3:33 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4527)

Hi Jakub,

You need to use the quotes

```
"A1"
```

instead of

```
“A1”
```

Also ensure there is a worksheet with the code name **Sheet1**.

Regards
Paul

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4527#respond)

Rene

March 22, 2017 at 1:07 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4523)

This is absolutely awesome. Thanks for putting this together. I am so looking forward to the next tutorial.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4523#respond)

Paul Kelly

March 22, 2017 at 3:39 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4528)

Thanks Rene.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4528#respond)

meytithveasna (http://www.google.com)

March 22, 2017 at 3:04 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4525)

I have 2 worsheets.sheet1 named account sheet2 is invoice named,i want to copy n paste by every i put invoice num in sheet2 cell A8,it will copy from sheet invoice F32 n paste in sheet1 start col E7 automatically,could you please help?

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4525#respond)

John Ovens

March 22, 2017 at 7:21 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4532)

Paul, absolutely fantastic. A great building block to start with – I am looking forward to later tutorials.

John

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4532#respond)

Paul Kelly

March 22, 2017 at 11:44 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4541)

Thanks John.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4541#respond)

Jan (http://www.ExcelXL.nl)

March 22, 2017 at 8:30 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4533)

Thank you Paul for this great tutorial. Looking forward to the next one.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4533#respond)

Paul Kelly

March 22, 2017 at 11:42 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4540)

Thanks Jan.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4540#respond)

Karel Hoogeboom (http://www.officeforum.nl)

March 22, 2017 at 8:43 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4534)

Paul, this is great. The only remark I have is that it is not clear when you write a code into a module and when not

Regards, Karel Hoogeboom
Then Netherlands
Look at http://www.officeforum.nl (http://www.officeforum.nl) for my Excel code whitin the Exce part and the mini (my name is HighTree)

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4534#respond)

Paul Kelly

March 22, 2017 at 12:05 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-4543)

Hi Karel,

Almost all the code goes in the modules. The worksheet modules contain the events and they should have minimal code.

Paul

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4543#respond)

Pradip Chaudhuri

March 22, 2017 at 12:24 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-4544)

Great Job. It is written in such a lucid form that anyone can understand the meaning of the code.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4544#respond)

Paul Kelly

March 22, 2017 at 1:01 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-4545)

Thanks Pradip.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4545#respond)

Robert

March 22, 2017 at 4:46 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-4548)

This is great! Thank you Paul. Clear, easy to understand & a workbook to work on, FANTASTIC!

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4548#respond)

Paul Kelly

March 23, 2017 at 10:35 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4565)

Thanks Robert

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4565#respond)

Ray Walmer (Retiree)

March 23, 2017 at 1:18 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-4566)

Very pleased with how clearly you simplify a very complicated subject. thanks for “Dumbing it down” for us old folks

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4566#respond)

Paul Kelly

March 23, 2017 at 1:34 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-4567)

Thanks Ray. Glad you like it.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4567#respond)

Pradeep Maji

March 24, 2017 at 6:00 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4577)

Excellent.. this is what most people are looking forward to learn while practice. Hope to see more practical base in later tutorial by you. Hats off sir !!!

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4577#respond)

Paul Kelly

March 24, 2017 at 7:46 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4579)

Thanks Pradeep.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4579#respond)

Wellsley Over

March 25, 2017 at 7:28 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4591)

Hi Paul, simply put, “Awesome”. I can’t wait for tutorial 2.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4591#respond)

Paul Kelly

March 25, 2017 at 1:05 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-4595)

Thanks Wellsley

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4595#respond)

Pradeep Maji

March 30, 2017 at 5:32 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-4739)

Hi Paul, any idea how soon you can share your part two tutorial

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4739#respond)

Paul Kelly

March 31, 2017 at 3:24 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4755)

Hi Pradeep, I'm working on it when time permits but have no release date at the moment.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4755#respond)

Pradeep Maji

March 31, 2017 at 3:25 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-4775)

Thanks for your response. Can you brief about your later tutorials atleast? like the tutorial 2 will contains loops and what more tutorials & contents you are planning.. Your first tutorial is awesome, none in this web world has prepared such tutorial with lots of practical activities topic/subject wise and atlast you put all we have learned in a single assignment at the end. Great !

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4775#respond)

Paul Kelly

April 1, 2017 at 5:16 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4790)

Thanks for your enthusiasm Pradeep.
I only have a rough outline of actual content of future tutorials. I will know more of the future content as I create each one.

My aim is that someone can learn VBA from scratch following these tutorials in order. This is the plan I working with.
When I create a tutorial I have a starting outline but this always changes considerably as I go through it.

Regards
Paul

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4790#respond)

Fei Fei

April 1, 2017 at 2:08 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4788)

Thanks for your effort and it is very useful!

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4788#respond)

Pradeep Maji

April 3, 2017 at 9:15 am (https://excelmacromastery.com/vba-tutorial-1/#comment-4852)

Thanks for your thoughts on learning VBA from Scratch for non coders like me. Waiting for your Tutorial 2 soon.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=4852#respond)

Pradeep Maji

April 6, 2017 at 3:52 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-5124)

Hi Paul.. any idea when are you planning to add your tutorial 2 .. hope so you would link your tutorial activities within your 5 real time application that you explained earlier.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=5124#respond)

Santosh Subudhi

April 29, 2017 at 7:52 am (https://excelmacromastery.com/vba-tutorial-1/#comment-5849)

Hi,
I have 50 cells to be copied not in a single column or row and needs to be pasted in a single row. Say we have cells A1, A3, C2, D3, D24, E5,A10, ……. needs to be copied from sheet and pasted in different sheet in single row say in row 2 of sheet to.
Thanks in advance.
Regards
Santosh

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=5849#respond)

Paul Kelly

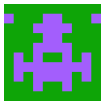
May 4, 2017 at 2:54 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-6209)

You can use an array to do this. You will have to add the cells one by one to the array.

Then you can write the array to the range e.g. row 2

Sheet1.Range("A2:Z2") = arr

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=6209#respond)



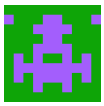
Shay

May 11, 2017 at 11:01 am (https://excelmacromastery.com/vba-tutorial-1/#comment-6539)

Great tutorial – keep up the good work. Really easy to follow.

Is it possible to be put on a mailing list as new tutorials are uploaded?

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=6539#respond)



Shay

May 11, 2017 at 2:31 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-6541)

As an aside, in the "double with" statement above, why is the "sheet1" needed in the second line for the last range but not elsewhere?

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=6541#respond)

Paul Kelly

May 11, 2017 at 2:35 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-6542)

It's a typo. I've updated the code. Thanks.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=6542#respond)

Paul Alexander

May 30, 2017 at 4:41 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-7322)

Great job Paul…. well just wanted to bring to your kind attention…. that Activity 5….. Its mentioned as "Run the code in the sub(F5). Cells on Sheet2 should have the values as follows: C1 20, C2 100 and C3 1600" ……..I think C should be changed to A since in the macro you mentioned refers to A1 20, A2 100, A 3 1600….when we run this Column A1 to A3 is what is displayed…….. I am also awaiting your second tutorial, please send a mail…….All the best…..

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=7322#respond)

Paul Kelly

May 31, 2017 at 9:35 am (https://excelmacromastery.com/vba-tutorial-1/#comment-7349)

Thank's Paul for pointing that out. That was a typo. I've updated the post.

I have created the second and third tutorial. They will be made available as part of membership contents of the site.

This is currently undergoing testing. I will send more details to my email list when I have them.

Regards
Paul

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=7349#respond)

Paul Alexander

May 31, 2017 at 10:10 am (https://excelmacromastery.com/vba-tutorial-1/#comment-7350)

Oh..great to hear that….. well hope you will include me too in that email list. Thanking you in advance…….Your explanation is so excellent……Keep going…..All the best…..

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=7350#respond)

N323100

June 15, 2017 at 2:11 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-7981)

How about copying data from different workbook?

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=7981#respond)

Paul Kelly

June 19, 2017 at 4:33 am (https://excelmacromastery.com/vba-tutorial-1/#comment-8100)

It is done in a similar way. You must first you must get the workbook and then the worksheet from the workbook.

```
Sub UseDifferentWorkbook()

    ' Open workbook
    Dim wk As Workbook
    Set wk = Workbooks.Open("data.xlsx")

    ' Get worksheet
    Dim sh As Worksheet
    Set sh = wk.Worksheets("sheet1")

    ' Copy value
    sh.Range("A1") = Sheet1.Range("C1")

    ' Close workbook when finished
    wk.Close

End Sub
```

You can read more about this here: Excel VBA Workbook (https://excelmacromastery.com/excel-vba-workbook) and Excel VBA Worksheet (https://excelmacromastery.com/excel-vba-worksheet).

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=8100#respond)

TauLieR

August 17, 2017 at 9:50 am (https://excelmacromastery.com/vba-tutorial-1/#comment-14767)

i'd suggest to use an array to increase speed when copying large amount of data.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=14767#respond)

Paul Kelly

August 18, 2017 at 3:04 am (https://excelmacromastery.com/vba-tutorial-1/#comment-14790)

Copying using the Range assignment is the same as using an array.

e.g.

```
Range("A1:A100").Value = Range("C1:C1

Dim arr As Variant
arr = Range("C1:C100").Value
Range("A1:A100").Value = arr
```

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=14790#respond)

Pradeep Maji

June 22, 2017 at 4:01 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-8243)

Desperately waiting for your Second and third tutorial sir.. !

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=8243#respond)

Paul Kelly

June 23, 2017 at 4:46 am (https://excelmacromastery.com/vba-tutorial-1/#comment-8258)

These are part of the membership contents of the website which I hope to open soon.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=8258#respond)

Carmina Nascimento

June 28, 2017 at 3:25 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-8502)

Pretty good tutorial, thanks a lot for sharing it!

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=8502#respond)

Ping

June 30, 2017 at 3:05 am (https://excelmacromastery.com/vba-tutorial-1/#comment-8591)

Hi, Paul,
I practice Double with. VBA shows the error message.
Code:

Sub DoubleWith()

With Sheet3
With WorksheetFunction
.Range(“E1:H1”) = .Transpose(.Range(“A1:A4”).Value)
.Range(“L1:L4”) = .Transpose(.Range(“E1:H1”).Value)
End With
End With

End Sub
Error message:
Object doesn’t support this property or method (Error 438)

The code statement is same as yours.
I don’t know how to fix it.
Please help me, thanks.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=8591#respond)

Paul Kelly

July 1, 2017 at 6:16 am (https://excelmacromastery.com/vba-tutorial-1/#comment-8641)

You cannot use a “Double with”statement like this. When VBA meets a full stops it refers to the last With statement.
So here it will put WorksheetFunction in from of .Range.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=8641#respond)

Rohit Sharma

July 5, 2017 at 5:40 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-8866)

Sir please help me on how to sort columns which has formulas in it…i have a column who picks value from other columns using vlookup function and this formula has to there…i have written vba code which only sorts values but not values which are derived from formulas…can you guide me…the only way i think is to copy whole sheet to new sheet and paste it as values then apply sort function..is there any other way…thank you in advance

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=8866#respond)

HemantH

July 14, 2017 at 10:38 am (https://excelmacromastery.com/vba-tutorial-1/#comment-9209)

dear Rohit sharma,
am not clear your problem please go through the below link.
https://excelmacromastery.com/vba-vlookup/
(https://excelmacromastery.com/vba-vlookup/)
i think you will get good result.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=9209#respond)

Database Error

August 4, 2017 at 10:34 am (https://excelmacromastery.com/vba-tutorial-1/#comment-13678)

I've been browsing through VBA tutorials and none come even close to this one. VBA coding is explained in a logical and straight forward way so that newcomers can easily understand why and how each element is used in VBA.

I'm eagerly looking forward to the following tutorials! Any idea when you plan to publish them?

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=13678#respond)

Paul Kelly

August 5, 2017 at 3:51 am (https://excelmacromastery.com/vba-tutorial-1/#comment-13707)

Thanks. Glad yob like them.

They are available in the membership section of the website.

There is a special offer on the membership section (https://excelmacromastery.com/vba-vault-special/) until Monday 7th August

Paul

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=13707#respond)

Jon Neagle

August 12, 2017 at 8:19 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-14570)

Great job on the Tutorial … Just wanted to bring to you attention on the assignment line 69 you reference the range A1:K1, I believe this should be A1:J1 as in the the answer otherwise the ranges are not equal and K1 will be #N/A error. Also in the range D2:D11 areas on shCountries and in the answer it references the population range of E2:E11. Thank you for making these tutorials available.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=14570#respond)

Paul Kelly

August 15, 2017 at 3:33 am (https://excelmacromastery.com/vba-tutorial-1/#comment-14678)

Thank's Jon for pointing out those typos. I have updated the text.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=14678#respond)

Joanna

August 28, 2017 at 11:32 am (https://excelmacromastery.com/vba-tutorial-1/#comment-15259)

great stuff! Thank you, Paul. Moving in to Tutorial 2!

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=15259#respond)

Paul Kelly

August 29, 2017 at 1:51 am (https://excelmacromastery.com/vba-tutorial-1/#comment-15289)

Glad you like it Joanna.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=15289#respond)

James Hill

September 3, 2017 at 12:54 pm (https://excelmacromastery.com/vba-tutorial-1/#comment-15941)

Paul, there was an issue opening the video for the Solution to Activity 3 in the The Ultimate VBA Tutorial Part One.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=15941#respond)

Paul Kelly

September 4, 2017 at 2:30 am (https://excelmacromastery.com/vba-tutorial-1/#comment-15965)

Hi James,

I have tried the video and it works fine for me. Try refreshing your browser and see if that helps.
It could also be caused by a slow internet connection.

Paul

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=15965#respond)

Ajay

September 7, 2017 at 10:59 am (https://excelmacromastery.com/vba-tutorial-1/#comment-16120)

Paul,
i gave code as Sheet1.Range(“A1”) = Now
but in excel sheet i am getting ####
how to solve this?

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=16120#respond)

Paul Kelly

September 8, 2017 at 3:24 am (https://excelmacromastery.com/vba-tutorial-1/#comment-16147)

You need to make the column wider on the spreadsheet.

Reply (https://excelmacromastery.com/vba-tutorial-1/?replytocom=16147#respond)

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Post Comment