The Complete Guide to the VBA Sub

BY PAUL KELLY

Sub	Example
Sub • Cannot return a value.	
Function • Can return a value or object. • Can run as Worksheet function.	
Create a sub	**Sub** CreateReport() **End Sub**
Create a function	**Function** GetTotal() **As Long** **End Function**
Create a sub with parameters	**Sub** CreateReport(**ByVal** Price **As** Double) **Sub** CreateReport(**ByVal** Name **As** String)

Sub	Example
Create a function with parameters	**Function** GetTotal(Price **As** Double) **Function** GetTotal(Name **As** String)
Call a sub	**Call** CreateReport ' Or CreateReport
Call a function	**Call** CalcPrice ' Or CalcPrice
Call a sub with parameters	**Call** CreateReport(12.99) CreateReport 12.99
Call a function with parameters	**Call** CalcPrice(12.99) CalcPrice 12.99
Call a function and retrieve value (cannot use Call keyword for this)	Price = CalcPrice
Call a function and retrieve object	Set coll = GetCollection
Call a function with params and retrieve value/object	Price = CalcPrice(12) **Set** coll = GetCollection("Apples")
Return a value from Function	**Function** GetTotal() **As Long** GetTotal = 67 **End Function**
Return an object from a function	**Function** GetCollection() **As** Collection **Dim** coll **As New** Collection **Set** GetCollection = coll **End Function**

Exit a sub	**If** IsError(Range("A1")) **Then**
	Exit Sub
	End If

Exit a function	**If** IsError(Range("A1")) **Then**
	Exit Function
	End If

Sub	Example
Private Sub/Private Function (available to current module)	**Private Sub** CreateReport()
Public Sub/Public Function (available to entire project)	**Public Sub** CreateReport()

Introduction

The VBA Sub is an essential component of the VBA language. You can also create functions which are very similar to subs. They are both procedures where you write your code. However, there are differences and these are important to understand. In this post I am going to look at subs and functions in detail and answer the vital questions including:

- What is the difference between them?
- When do I use a sub and when do I use a function?
- How do you run them?
- Can I return values?
- How do I pass parameters to them?
- What are optional parameters?
- and much more

Let’s start by looking at what is the VBA sub?

What is a Sub?

In Excel VBA a sub and a macro are essentially the same thing. This often leads to confusion so it is a good idea to remember it. For the rest of this post I will refer to them as subs.

A sub/macro is where you place your lines of VBA code. When you run a sub, all the lines of code it contains are executed. That means that VBA carries out their instructions.

The following is an example of an empty sub

```
Sub WriteValues()


End Sub
```

You declare the sub by using the *Sub* keyword and the name you wish to give the sub. When you give it a name keep the following in mind:

- The name must begin with a letter and cannot contain spaces.
- The name must be unique in the current workbook.
- The name cannot be a reserved word in VBA.

The end of the Sub is marked by the line **End Sub**.

When you create your Sub you can add some code like the following example shows:

```
Sub WriteValues()
    Range("A1") = 6
End Sub
```

What is a Function?

A Function is very similar to a sub. The major difference is that a function can return a value – a sub cannot. There are other differences which we will look at but this is the main one.

You normally create a function when you want to return a value.

Creating a function is similar to creating a sub

```
Function PerformCalc()
```

```
End Function
```

It is optional to add a return type to a function. If you don't add the type then it is a Variant type by default. This means that VBA will decide the type at runtime.

The next example shows you how to specify the return type

```
Function PerformCalc() As Long


End Function
```

You can see it is simple how you declare a variable. You can return any type you can declare as a variable including objects and collections.

A Quick Comparison

Sub:

1. Cannot return a value.
2. Can be called from VBA\Button\Event etc.

Function

1. Can return a value but doesn't have to.
2. Can be called it from VBA\Button\Event etc. but it won't appear in the list of Macros. You must type it in.
3. If the function is public, will appear in the worksheet function list for the current workbook.

Note 1: You can use "Option Private Module" to hide subs in the current module. This means that subs won't be visible to other projects and applications. They also won't appear in a list of subs when you bring up the Macro window on the developer tab.

Note 2: We can use the word *procedure* to refer to a function or sub

Calling a Sub or Function

When people are new to VBA they tend to put all the code in one sub. This is not a good way to write your code.

It is better to break your code into multiple procedures. We can run one procedure from another.

Here is an example:

```
' https://excelmacromastery.com/
Sub Main()

    ' call each sub to perform a task
    CopyData

    AddFormulas

    FormatData

End Sub

Sub CopyData()
    ' Add code here
End Sub

Sub AddFormulas()
    ' Add code here
End Sub
```

```
Sub FormatData()

    ' Add code here

End Sub
```

You can see that in the *Main* sub, we have added the name of three subs. When VBA reaches a line containing a procedure name, it will run the code in this procedure.

We refer to this as calling a procedure e.g. We are calling the *CopyData* sub from the *Main* sub.

There is actually a *Call* keyword in VBA. We can use it like this:

```
' https://excelmacromastery.com/
Sub Main()


    ' call each sub to perform a task
    Call CopyData


    Call AddFormulas


    Call FormatData


End Sub
```

Using the *Call* keyword is optional. There is no real need to use it unless you are new to VBA and you feel it makes your code more readable.

If you are passing arguments using *Call* then you must use parentheses around them.

For example:

```
' https://excelmacromastery.com/
Sub Main()
```

```
    ' If call is not used then no parentheses
    AddValues 2, 4

    ' call requires parentheses for arguments
    Call AddValues(2, 4)
End Sub

Sub AddValues(x As Long, y As Long)

End Sub
```

How to Return Values From a Function

To return a value from a function you assign the value to the name of the Function. The following example demonstrates this:

```
' https://excelmacromastery.com/
Function GetAmount() As Long
    ' Returns 55
    GetAmount = 55
End Function

Function GetName() As String
    ' Returns John
    GetName = "John"
End Function
```

When you return a value from a function you will obviously need to receive it in the function/sub that called it. You do this by assigning the function call to a variable. The following example shows this:

```
' https://excelmacromastery.com/
Sub WriteValues()
    Dim Amount As Long
    ' Get value from GetAmount function
    Amount = GetAmount
End Sub

Function GetAmount() As Long
    GetAmount = 55
End Function
```

You can easily test your return value using by using the *Debug.Print* function. This will write values to the Immediate Window (View->Immediate window from the menu or press Ctrl + G).

```
' https://excelmacromastery.com/
Sub WriteValues()
    ' Print return value to Immediate Window
    Debug.Print GetAmount
End Sub

Function GetAmount() As Long
    GetAmount = 55
End Function
```

Using Parameters and Arguments

We use parameters so that we can pass values from one sub/function to another.

The terms parameters and arguments are often confused:

- A parameter is the variable in the sub/function declaration.
- An argument is the value that you pass to the parameter.

```
' https://excelmacromastery.com/
Sub UsingArgs()

    ' The argument is 56
    CalcValue 56

    ' The argument is 34
    CalcValue 34

End Sub

' The parameter is amount
Sub CalcValue(ByVal amount As Long)

End Sub
```

Here are some important points about parameters:

- We can have multiple parameters.

- A parameter is passed using either ByRef or ByVal. The default is ByRef.
- We can make a parameter optional for the user.
- We cannot use the *New* keyword in a parameter declaration.
- If no variable type is used then the parameter will be a variant by default.

The Format of Parameters

Subs and function use parameters in the same way.

The format of the parameter statement is as follows:

```
' All variables except arrays
[ByRef/ByVal]  As [Variable Type]


' Optional parameter - can only be a basic type
[Optional] [ByRef/ByVal] [Variable name] As <[Variable Type] =


' Arrays
[ByRef][array name]() As [Variable Type]
```

Here are some examples of the declaring different types of parameters:

```
' https://excelmacromastery.com/
' Basic types


Sub UseParams1(count As Long)
End Sub


Sub UseParams2(name As String)
End Sub
```

```
Sub UseParams3(amount As Currency)
End Sub

' Collection
Sub UseParamsColl(coll As Collection)
End Sub

' Class module object
Sub UseParamsClass(o As Class1)
End Sub

' Variant
' If no type is give then it is automatically a variant
Sub UseParamsVariant(value)
End Sub

Sub UseParamsVariant2(value As Variant)
End Sub

Sub UseParamsArray(arr() As String)
End Sub
```

You can see that declaring parameters looks similar to using the Dim statement to declare variables.

Multiple Parameters

We can use multiple parameters in our sub/functions. This can make the line very long

```
Sub LongLine(ByVal count As Long, Optional amount As Currency = 56.77, Opt
ional name As String = "John")
```

We can split up a line of code using the underscore (_) character. This makes our code more readable

```
Sub LongLine(ByVal count As Long _
            , Optional amount As Currency = 56.77 _
            , Optional name As String = "John")
```

Parameters With a Return Value

This error causes a lot of frustration with new users of VBA.

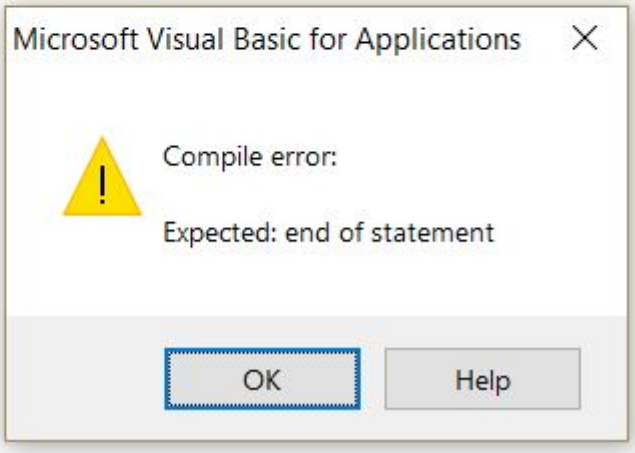
If you are returning a value from a function then it must have parentheses around the arguments.

The code below will give the “Expected: end of statement” syntax error.

```
' https://excelmacromastery.com/
Sub UseFunction()


    Dim result As Long


    result = GetValue 24.99


End Sub
```

```
Function GetValue(amount As Currency) As Long
    GetValue = amount * 100
End Function
```

```
Sub UseFunction()
    Dim result As Long
    result = GetValue 24.99
End Sub

Function GetValue(amount As Currency) As Long
End Function
```



We have to write it like this

```
result = GetValue (24.99)
```

ByRef and ByVal

Every parameter is either **ByRef** or **ByVal**. If no type is specified then it is ByRef by default

```
' https://excelmacromastery.com/

' Pass by value

Sub WriteValue1(ByVal x As Long)


End Sub


' Pass by reference

Sub WriteValue2(ByRef x As Long)


End Sub


' No type used so it is ByRef

Sub WriteValue3(x As Long)


End Sub
```

If you don't specify the type then ByRef is the type as you can see in the third sub of the example.

The different between these types is:

ByVal – Creates a copy of the variable you pass.
This means if you change the value of the parameter it **will not be changed** when you return to the calling sub/function

ByRef – Creates a reference of the variable you pass.
This means if you change the value of the parameter variable it **will be changed** when you return to the calling sub/function.

The following code example shows this:

```
' https://excelmacromastery.com/
Sub Test()

    Dim x As Long

    ' Pass by value - x will not change
    x = 1
    Debug.Print "x before ByVal is"; x
    SubByVal x
    Debug.Print "x after ByVal is"; x

    ' Pass by reference - x will change
    x = 1
    Debug.Print "x before ByRef is"; x
    SubByRef x
    Debug.Print "x after ByRef is"; x

End Sub

Sub SubByVal(ByVal x As Long)
    ' x WILL NOT change outside as passed ByVal
    x = 99
End Sub

Sub SubByRef(ByRef x As Long)
```

```
    ' x WILL change outside as passed ByRef
    x = 99
End Sub
```

The result of this is:
x before ByVal is 1
x after ByVal is 1
x before ByRef is 1
x after ByRef is 99

You should avoid passing basic variable types using ByRef. There are two main reasons for this:

1. The person passing a value may not expect it to change and this can lead to bugs that are difficult to detect.
2. Using parentheses when calling prevents ByRef working – see next sub section

A Little-Known Pitfall of ByRef

There is one thing you must be careful of when using *ByRef* with parameters. If you use parentheses then the sub/function cannot change the variable you pass even if it is passed as ByRef.

In the following example, we call the sub first without parentheses and then with parentheses. This causes the code to behave differently.

For example

```
' https://excelmacromastery.com/
Sub Test()

    Dim x As Long
```

```
    ' Call using without Parentheses - x will change
    x = 1
    Debug.Print "x before (no parentheses): "; x
    SubByRef x
    Debug.Print "x after (no parentheses): "; x

    ' Call using with Parentheses - x will not change
    x = 1
    Debug.Print "x before (with parentheses): "; x
    SubByRef (x)
    Debug.Print "x after (with parentheses): "; x

End Sub

Sub SubByRef(ByRef x As Long)
    x = 99
End Sub
```

If you change the sub in the above example to a function, you will see the same behaviour occurs.

However, if you return a value from the function then ByRef will work as normal as the code below shows:

```
' https://excelmacromastery.com/
Sub TestFunc()

    Dim x As Long, ret As Long

    ' Call using with Parentheses - x will not change
```

```
    x = 1
    Debug.Print "x before (with parentheses): "; x
    FuncByRef (x)
    Debug.Print "x after (with parentheses): "; x

    ' Call using with Parentheses and return - x will change
    x = 1
    Debug.Print "x before (with parentheses): "; x
    ret = FuncByRef(x)
    Debug.Print "x after (with parentheses): "; x



End Sub


Function FuncByRef(ByRef x As Long)
    x = 99
End Function
```

As I said in the last section you should avoid passing a variable using ByRef and instead use ByVal.

This means

1. The variable you pass will not be accidentally changed.
2. Using parentheses will not affect the behaviour.

ByRef and ByVal with Objects

When we use ByRef or ByVal with an object, it only affects the variable. It does not affect the actual object.

If we look at the example below:

```
' https://excelmacromastery.com/
Sub UseObject()

    Dim coll As New Collection
    coll.Add "Apple"

    ' After this coll with still reference the original
    CollByVal coll



    ' After this coll with reference the new collection
    CollByRef coll

End Sub

Sub CollByVal(ByVal coll As Collection)

    ' Original coll will still reference the original
    Set coll = New Collection
    coll.Add "ByVal"

End Sub

Sub CollByRef(ByRef coll As Collection)
```

```
    ' Original coll will reference the new collection
    Set coll = New Collection
    coll.Add "ByRef"

End Sub
```

When we pass the *coll* variable using ByVal, a copy of the variable is created. We can assign anything to this variable and it will not affect the original one.

When we pass the *coll* variable using ByRef, we are using the original variable. If we assign something else to this variable then the original variable will also be assigned to something else.

You can see find out more about this here.

Optional Parameters

Sometimes we have a parameter that will often be the same value each time the code runs. We can make this parameter *Optional* which means that we give it a default value.

It is then optional for the caller to provide an argument. If they don't provide a value then the default value is used.

In the example below, we have the report name as the optional parameter:

```
Sub CreateReport(Optional reportName As String = "Daily Report")


End Sub
```

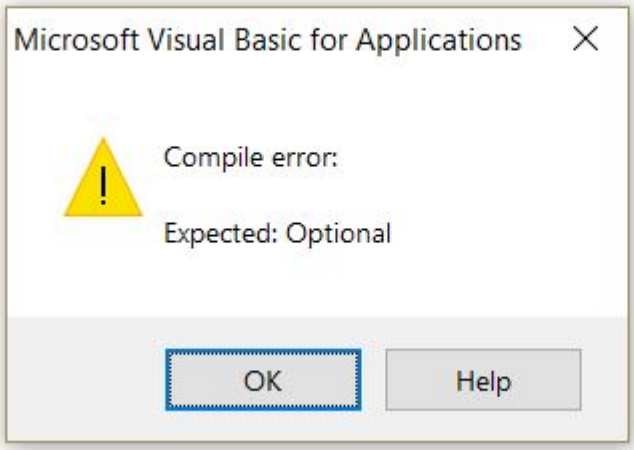
If an argument is not provided then *name* will contain the "Daily Report" text

```
' https://excelmacromastery.com/
Sub Main()
```

```
    ' Name will be "Daily Report"
    CreateReport

    ' Name will be "Weekly Report"
    CreateReport "Weekly Report"

End Sub
```

The Optional parameter cannot come before a normal parameter. If you do this you will get an *Expected: Optional* error.

```
Sub CreateReport(Optional name As String = "Daily Report",amount as Long

End Sub
```



When a sub/function has optional parameters they will be displayed in square parentheses by the Intellisense.

In the screenshot below you can see that the *name* parameter is in square parentheses.

```
    ' Name will be "Weekly Report"
    CreateReport
    CreateReport(x As Long, [name As String = "Daily Report"])
End Sub

Sub CreateReport(x As Long, Optional name As String = "Daily Report")

End Sub
```

A sub/function can have multiple optional parameters. You may want to provide arguments to only some of the parameters.

There are two ways to do this:
If you don't want to provide an argument then leave it blank.

A better way is to use the parameter name and the ":=" operator to specify the parameter and its' value.

The examples below show both methods:

```
' https://excelmacromastery.com/
Sub Multi(marks As Long _
            , Optional count As Long = 1 _
            , Optional amount As Currency = 99.99 _
            , Optional version As String = "A")

    Debug.Print marks, count, amount, version

End Sub



Sub UseBlanks()

    ' marks and count
```

```
    Multi 6, 5

    ' marks and amount
    Multi 6, , 89.99

    ' marks and version
    Multi 6, , , "B"

    ' marks,count and version
    Multi 6, , , "F"

End Sub

Sub UseName()

    ' marks and count
    Multi 12, count:=5

    ' marks and amount
    Multi 12, amount:=89.99

    ' marks and version
    Multi 12, version:="B"

    ' marks,count and version
    Multi 12, count:=6, version:="F"
```

```
End Sub
```

Using the name of the parameter is the best way. It makes the code more readable and it means you won't have error by mistakenly adding extra commas.

```
wk.SaveAs "C:\Docs\data.xlsx", , , , , , xlShared


wk.SaveAs "C:\Docs\data.xlsx", AccessMode:=xlShared
```

IsMissing Function

We can use the *IsMissing* function to check if an Optional Parameter was supplied.

Normally we check against the default value but in certain cases we may not have a default.

We use IsMissing with Variant parameters because it will not work with basic types like Long and Double.

```
' https://excelmacromastery.com/
Sub test()
    ' Prints "Parameter not missing"
    CalcValues 6

    ' Prints "Parameter missing"
    CalcValues

End Sub

Sub CalcValues(Optional x)

    ' Check for the parameter
```

```
    If IsMissing(x) Then
        Debug.Print "Parameter missing"
    Else
        Debug.Print "Parameter Not missing"
    End If

End Sub
```

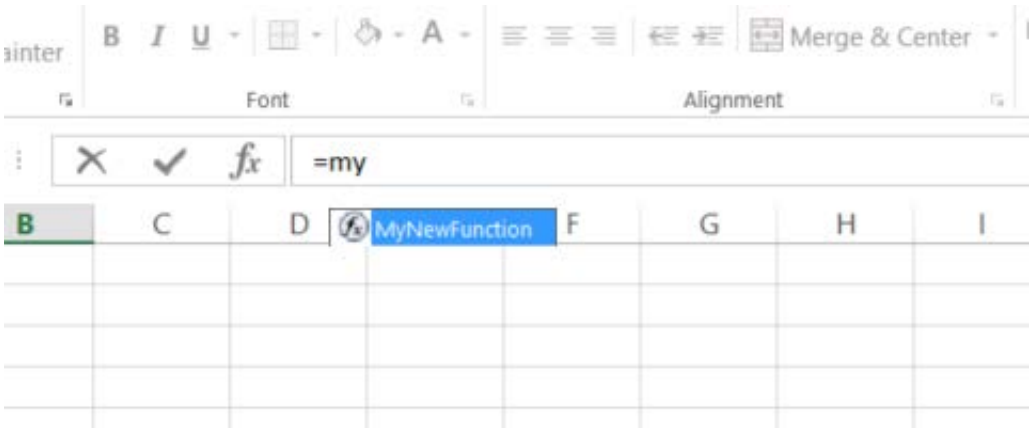
Custom Function vs Worksheet Function

When you create a function it appears in the function list for that workbook.

Have a look at the function in the next example.

```
Public Function MyNewFunction()
    MyNewFunction = 99
End Function
```

If you add this to a workbook then the function will appear in the function list. Type “=My” into the function box and the function will appear as shown in the following screenshot.



If you use this function in a cell you will get the result 99 in the cell as that is what the function returns.

Conclusion

The main points of this post are:

- Subs and macros are essentially the same thing in VBA.
- Functions return values but subs do not.
- Functions appear in the workbook function list for the current workbook.
- ByRef allows the function or sub to change the original argument.
- If you call a function sub with parentheses then ByRef will not work.
- Don’t use parentheses on sub arguments or function arguments when not returning a value.
- Use parentheses on function arguments when returning a value.