

Writing VBA Functions in Excel

About User Defined Functions

Excel provides the user with a large collection of ready-made functions, more than enough to satisfy the average user. Many more can be added by installing the various add-ins that are available.

Most calculations can be achieved with what is provided, but it isn't long before you find yourself wishing that there was a function that did a particular job, and you can't find anything suitable in the list. You need a UDF.

A UDF (User Defined Function) is simply a function that you create yourself with VBA. UDFs are often called "Custom Functions". A UDF can remain in a code module attached to a workbook, in which case it will always be available when that workbook is open. Alternatively you can create your own add-in containing one or more functions that you can install into Excel just like a commercial add-in.

UDFs can be accessed by code modules too. Often UDFs are created by developers to work solely within the code of a VBA procedure and the user is never aware of their existence.

Like any function, the UDF can be as simple or as complex as you want. Let's start with an easy one...

A Function to Calculate the Area of a Rectangle

Yes, I know you could do this in your head! The concept is very simple so you can concentrate on the technique.

Suppose you need a function to calculate the area of a rectangle. You look through Excel's collection of functions, but there isn't one suitable. This is the calculation to be done:

$$\text{AREA} = \text{LENGTH} \times \text{WIDTH}$$

Open a new workbook and then open the Visual Basic Editor (Tools > Macro > Visual Basic Editor or ALT+F11).

You will need a module in which to write your function so choose **Insert > Module** (Fig. 1). Into the empty module type: *Function Area* and press **ENTER**.

The Visual Basic Editor completes the line for you and adds an End Function line as if you were creating a subroutine.

So far it looks like this...

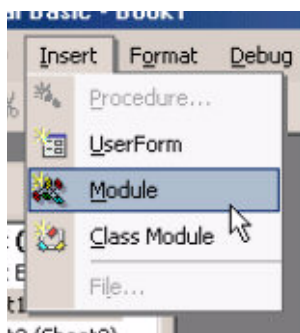


Fig. 1 Inserting a new code module

```
Function Area()  
  
End Function
```

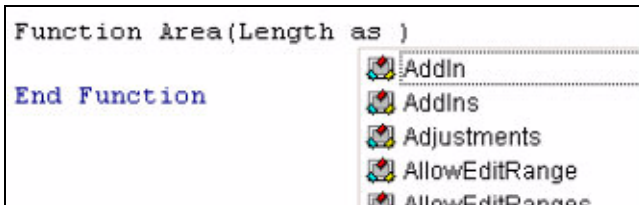


Fig. 2 Auto List Members

Place your cursor between the brackets after "Area". If you ever wondered what the brackets are for, you are about to find out! We are going to specify the "arguments" that our function will take (an *argument* is a piece of information needed to do the calculation). Type *Length as double*, *Width as double* and click in the empty line underneath. Note that as you type, a scroll box pops-up listing all the things appropriate to what you are typing (Fig. 2).

This feature is called **Auto List Members**. If it doesn't appear either it is switched off (turn it on at **Tools > Options > Editor**) or you might have made a typing error earlier. It is a very useful check on your syntax. Find the item you need and double-click it to insert it into your code. You can ignore it and just type if you want. Your code now looks like this...

```
Function Area (Length As Double, Width As Double)  
  
End Function
```

Declaring the data type of the arguments is not obligatory but makes sense. You could have typed *Length*, *Width* and left it as that, but warning Excel what data type to expect helps your code run more quickly and picks up errors in input. The *double* data type refers to number (which can be very large) and allows fractions.

Now for the calculation itself. In the empty line first press the **TAB** key to indent your code (making it easier to read) and type :

*Area = Length * Width.*

Here's the completed code...

```
Function Area (Length As Double, Width As Double)  
    Area = Length * Width  
End Function
```

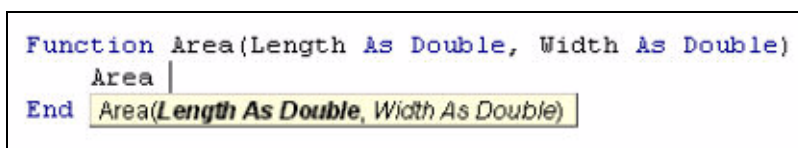


Fig. 3 Auto Quick Info

You will notice another of the Visual Basic Editor's help features pop up as you were typing, **Auto Quick Info** (Fig. 3).

It isn't relevant here. Its purpose is to help you write functions in VBA, by telling you what arguments are required.

	A	B	C	D
1	17	6.5	110.5	
2				

Fig. 4 Using the AREA function

You can test your function right away. Switch to the Excel window and enter figures for Length and Width in separate cells. In a third cell enter your function as if it were one of the built-in ones. In this example cell A1 contains the length (17) and cell B1 the width (6.5). In C1 I typed `=area(A1,B1)` and the new function calculated the area (110.5). (Fig. 4)

Sometimes, a function's arguments can be optional. In this example we could make the **Width** argument optional. Supposing the rectangle happens to be a square with Length and Width equal. To save the user having to enter two arguments we could let them enter just the Length and have the function use that value twice (i.e. multiply Length x Length). So the function knows when it can do this we must include an **IF Statement** to help it decide.

Change the code so that it looks like this...

```
Function Area(Length As Double, Optional Width As Variant)
    If IsMissing(Width) Then
        Area = Length * Length
    Else
        Area = Length * Width
    End If
End Function
```

	A	B	C	D
1	12.3		151.29	
2				

Fig. 5 Using the AREA function with a single argument

Note that the data type for Width has been changed to *Variant* to allow for null values. The function now allows the user to enter just one argument e.g. `=area(A1)`. The IF Statement in the function checks to see if the Width argument has been supplied and calculates accordingly. (Fig. 5)

Now for a more practical example...

A Function to Calculate Fuel Consumption

I like to keep a check on my car's fuel consumption so when I buy fuel I make a note of the mileage and how much fuel it takes to fill the tank. Here in the UK fuel is sold in litres. The car's milometer (OK, so it's an odometer) records distance in miles. And because I'm too old and stupid to change, I only understand MPG (miles per gallon).

Now if you think that's all a bit sad, how about this. When I get home I open up Excel and enter the data into a worksheet that calculates the MPG for me and charts the car's performance.

The calculation is the number of miles the car has travelled since the last fill-up divided by the number of gallons of fuel used...

MPG = (MILES THIS FILL - MILES LAST FILL) /
GALLONS OF FUEL

but because the fuel comes in litres and there are
4.546 litres in a gallon..

MPG = (MILES THIS FILL - MILES LAST FILL) /
LITRES OF FUEL x 4.546

Here's how I wrote the function...

```
Function MPG(StartMiles As Integer, FinishMiles As Integer, Litres As Single)
    MPG = (FinishMiles - StartMiles) / Litres * 4.546
End Function
```

	A	B	C	D
1	Miles	Litres	MPG	
17	3252	23.87	36.19	
18	3507	30.59	37.90	
19	3829	37.05	39.51	

...and here's how it looks on the worksheet.
(Fig. 6)

Not all functions perform mathematical
calculations. Here's one that provides
information...

Fig. 6 Using the MPG function

A Function That Gives the Name of the Day

I am often asked if there is a date function that
gives the day of the week as text (e.g. Monday).
The answer is no, but it's quite easy to create
one.

Excel has the WEEKDAY function, which returns
the day of the week as a number from 1 to 7. You
get to choose which day is 1 if you don't like the
default (Sunday). In the example below the
function returns "5" which I happen to know
means "Thursday". (Fig. 7)

	A	B	C	D
1	27/09/2001	5		

Fig. 7 Using Excel's WEEKDAY function

But I don't want to see a number, I want to see
"Thursday". I could modify the calculation by
adding a VLOOKUP function that referred to a
table somewhere containing a list of numbers and
a corresponding list of day names. Or I could
have the whole thing self-contained with multiple
nested IF statements. Too complicated! The
answer is a custom function...

```
Function DayName(InputDate As Date)
    Dim DayNumber As Integer
    DayNumber = Weekday(InputDate, vbSunday)
    Select Case DayNumber
        Case 1
            DayName = "Sunday"
        Case 2
            DayName = "Monday"
        Case 3
```

```

    DayName = "Tuesday"
Case 4
    DayName = "Wednesday"
Case 5
    DayName = "Thursday"
Case 6
    DayName = "Friday"
Case 7
    DayName = "Saturday"
End Select
End Function

```

I've called my function "DayName" and it takes a single argument, which I call "InputDate" which (of course) has to be a date. Here's how it works...

- The first line of the function declares a variable that I have called "DayNumber" which will be an Integer (i.e. a whole number).
- The next line of the function assigns a value to that variable using Excel's WEEKDAY function. The value will be a number between 1 and 7. Although the default is 1=Sunday, I've included it anyway for clarity.
- Finally a **Case Statement** examines the value of the variable and returns the appropriate piece of text.

	A	B	C	D
1	27/09/2001	Thursday		

Fig. 8 Using the DAYNAME function

Here's how it looks on the worksheet. (Fig. 8)

Accessing Your Custom Functions

If a workbook has a VBA code module attached to it that contains custom functions, those functions can be easily addressed within the same workbook as demonstrated in the examples above. You use the function name as if it were one of Excel's built-in functions.

You can also find the functions listed in the Function Wizard (sometimes called the Paste Function tool). Use the wizard to insert a function in the normal way (**Insert > Function**).

Scroll down the list of function categories to find **User Defined** (Fig. 9) and select it to see a list of available UDFs. (Fig. 10)

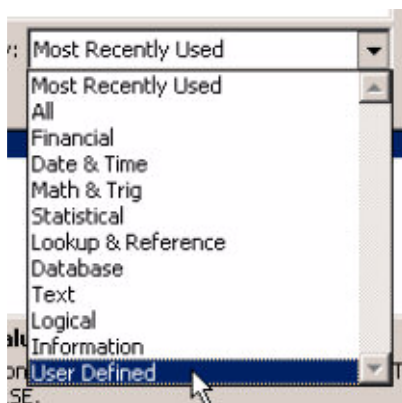


Fig. 9 Select the User Defined function category

You can see that the user defined functions lack any description other than the unhelpful "No help available" message, but you can add a short description.

Make sure you are in the workbook that contains the functions. Go to **Tools > Macro > Macros**.

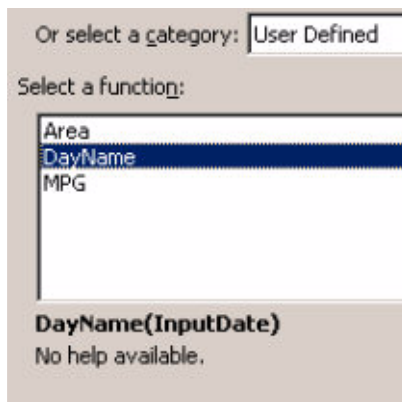


Fig. 10 User defined functions listed without descriptions

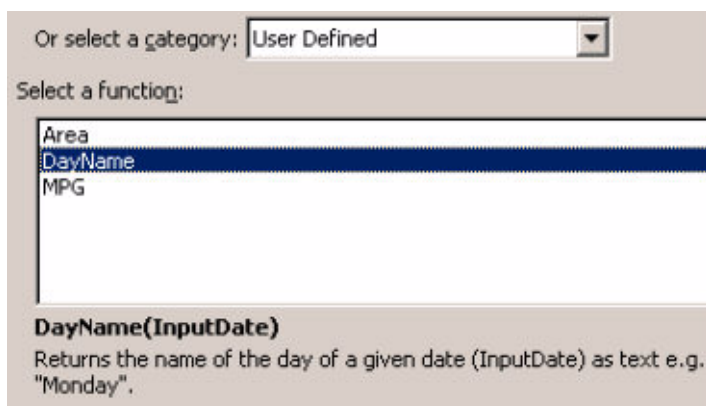


Fig. 11 A custom description for the user defined function

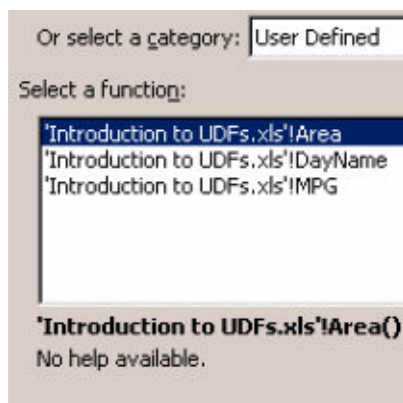


Fig. 12 User defined functions labelled with the name of their host workbook

You won't see your functions listed here but Excel knows about them!

In the **Macro Name** box at the top of the dialog, type the name of the function, then click the dialog's **Options** button. If the button is greyed out either you've spelled the function name wrong, or you are in the wrong workbook, or it doesn't exist! This opens another dialog into which you can enter a short description of the function. Click **OK** to save the description and (here's the confusing bit) click **Cancel** to close the Macro dialog box. Remember to Save the workbook containing the function. Next time you go to the Function Wizard your UDF will have a description. (Fig. 11)

Like macros, user defined functions can be used in any other workbook as long as the workbook containing them is open. However it is not good practice to do this. Entering the function in a different workbook is not simple. You have to add its host workbook's name to the function name. This isn't difficult if you rely on the Function Wizard, but clumsy to write out manually. The Function Wizard shows the full names of any UDFs in other workbooks. (Fig. 12)

If you open the workbook in which you used the function at a time when the workbook containing the function is closed, you will see an error message in the cell in which you used the function. Excel has forgotten about it! Open the function's host workbook, recalculate, and all is fine again. Fortunately there is a better way.

If you want to write User Defined Functions for use in more than one workbook the best method is to create an Excel **Add-In**.

Find out how to do this in the Excel Add-Ins tutorial.