Learning Excel VBA

Creating User Defined Functions

Prepared By Daniel Lamarche

ComboProjects

Creating User Defined Functions

By Daniel Lamarche (Last update January 2016).

User Defined Functions or UDFs are simply custom functions that you create and use in your workbook like any other Excel built-in functions. Awesome? Indeed it is!

What's a function again?

A function is a question you ask about some data in a worksheet and it returns *one* answer.

Think about it for a moment: What does VLookup(), SumIf() and PMT() have in common? They all return one and only one answer. This returned value is normally located in a cell.

Like built-in functions, UDFs can have one or more arguments. An argument can be a cell reference or a range. In most case your functions will simply require one or two cell reference.

What you want to remember though is that it is impossible for a function to *do* something in a worksheet. It can only return one value (or answer). So, to make sure this is clear you cannot use a function to make a cell bold or change the font size or change the width of a column.

If you need to *do* something in your worksheet you need to use a Sub procedure (some people like to use the word macro) but a function cannot do an action.

Structure of a Function

Below is the typical layout of a function.

```
Function FunctionName(Arg1 As DataType, Arg2 As DataType, ...) As DataType
' Description of what it does.
    <Statements>
    FunctionName = Some Result
End Function
```

Explanation of each element

- **FunctionName** is the name of the function. No punctuation (including spaces) and the first character cannot be a number.
- **Arg1** is the argument (or parameter). As shown, to use more than one argument separate each one with a comma. It is very important to declare the data type of each argument.
- After the closing parenthesis you also need to specify the data type of the returned value (the answer).
- Near the end of your UDF you specify what value to be returned by the function by assigning the final result to the name of the function.

Small Example

In a standard module type the function below. To indent hit the Tab key.

```
Function Cel2Fah(intCel As Integer) As Integer
' Converts a Celsius to Fahrenheit. Uses whole numbers only.

    Cel2Fah = 9 / 5 * intCel + 32
End Function
```

As you can see the **Cel2Fah** function has all the elements discussed above. You can quickly test a function in the Immediate Window.

Testing a Function in the Immediate Window

Press Ctrl+G and resize the **Immediate Window** as needed. In the Immediate Window type the following:

```
?Cel2Fah(0)
```

And hit Enter. The answer 32 appears immediately. So 0^{o}C = 32^{o}F.

The question mark is only necessary when you test your function in the Immediate Window. It stands for *evaluate the following expression*.

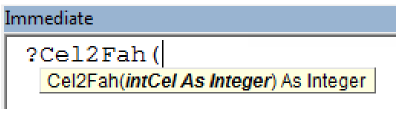
For the duration of the function the argument intCel will hold the value of the argument. When the function is complete that value cease to exist.

What do you mean by *returning* a value?

In the introduction of this document we learned that a function is like a question. In the programming world, getting back with an answer is called *returning* a value which is the answer to your *question.* All good?

Parameter List

Note that, as shown below, when you type the open bracket a screen tip appears giving you an indication about how many arguments are required and the data type.



```
Immediate
?Cel2Fah(
Cel2Fah(intCel As Integer) As Integer
```

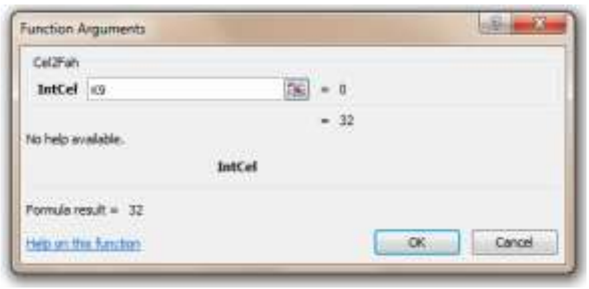
Note about formatting returned value

Do not be tempted to format a numeric value using the function. Instead have the function return a numeric value in Excel *then* format it in Excel. This way it is still a number in the worksheet. If you format the returned value at the function level (for instance 32F instead of simply 32) the answer would be a string and it would be difficult to use it in Excel for further processing.

Testing the function in Excel

Type 0 in a cell in your worksheet then follow these instructions:

- In another cell type =Cel. The CelFah function should be highlighted.
- Hit Tab then press Ctrl+A to open the Function Arguments box.
- Click the cell with the value in Celsius
- Note that the answer of the Formula Result is visible in the Function Arguments. Click OK.



Function Arguments

Cel2Fah

IntCel K9 = 0

= 32

No help available.

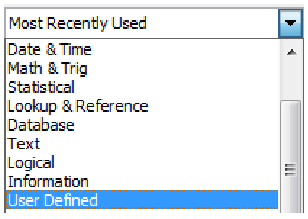
IntCel

Formula result = 32

Help on this function

OK Cancel

If you don't remember the name of the function you can click the **Insert Function** button and choose the **User Defined** category to see all your custom functions in scope (available in your workbook).

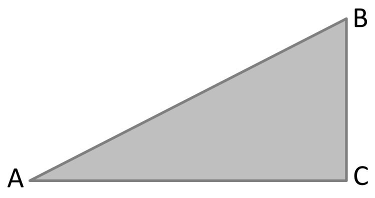


Functions are a fantastic because they hide the logic behind a simple name. This way you don't really care how it works, you just *know* that it works.

Couple more examples

In this section we will build a couple of functions and explore some built-in VBA tools you can use in custom UDFs.

Suppose you regularly need to calculate the diagonal of a triangle. For instance suppose you need to know the length of an extrusion (A-B) knowing the distance of the ground (A-C) and the height of the wall (C-B).



Don’t worry I’ll provide the formula. So the distance AB is given by summing the squares of the two sides then finding the square root of that sum. Right?

$$ab = \sqrt{ac^2 + bc^2}$$

Let’s explore how VBA can simplify doing such a formula. In VBE click in the Immediate Window and try the following expression and hit Enter:

```
?5^2
 25
```

This is 5 to the power of 2. The caret is the exponentiation character.

The following expression returns the square root (using the **SQR** function) of the sum of the two squares.

```
?sqr(3^2 + 4^2)
 5
```

Ok we understand the logic, let’s create the **Diagonal** function.

The Diagonal Function

Before we start typing the code we absolutely need to know in what units of measurement will be used to provide the distances. For instance will it be in millimetres or metres? This makes a big difference because if it’s in metres then we will need to use decimal numbers. If it’s in millimetres then we will use rather large integers (whole numbers).

For sake of simplicity we will use metres which will require the use of decimals. Type the following *exactly* as shown (remember to indent as shown using Tab):

```
Function Diagonal(sngBase As Single, sngHeight As Single) As Single
' Returns the Diagonal of triangle knowing the Base and Height.

    Dim sngGround As Single
    Dim sngWall As Single

    sngGround = sngBase ^ 2     ' Calculate the square of the base distance.
    sngWall = sngHeight ^ 2     ' Calculate the square of the height distance.

    Diagonal = Sqr(sngGround ^ 2 + sngWall ^ 2) ' Put it together.
End Function
```

The **Single** data type in a relatively small footprint data type that can hold very precise decimal values.

In the Immediate Window type your function with 2 arguments as shown here:

```
?Diagonal(5.674,3.652)
 34.84752
```

As you can see the name Diagonal hides all the complexity in the backstage. You do not need to know how it does the job. As long as it works, you’re happy.

In a worksheet, type 3 or 4 decimal numbers in column A and B. In column C launch you function using one of the techniques discussed above. Technically it doesn’t matter which one is the first or second argument since it’s an addition. Then copy your function down column C.

Using Conditional Statements

Using an If-Then-Else Structure

You can use conditional statements in your functions when the answer may *depend* on many factors. The next example uses the If-Then-Else-End If construct.

In a module type the following function:

```
Function Commission(dblSales As Double) As Double
' Calculates the amount of a commission.

    Dim dblComm As Double

    If dblSales > 1000 Then
        dblComm = dblSales * 0.08    ' Calculate 8% commission
    ElseIf dblSales > 500 Then
        dblComm = dblSales * 0.06    ' Calculate 6% commission
    Else
        dblComm = dblSales * 0.04    ' Calculate 4% commission
    End If

    Commission = dblComm
End Function
```

This example uses one argument using the **Double** data type. This data type is extremely precise holding about 15 decimal precision. A caveat could be that in a very large application this data type may slow down the processing. Use it only if you require a very precise value after being processed some number of times.

In the Immediate Window try your function with one argument in brackets:

```
?Commission(387.45)
 15.498
```

Stepping Into Your Code

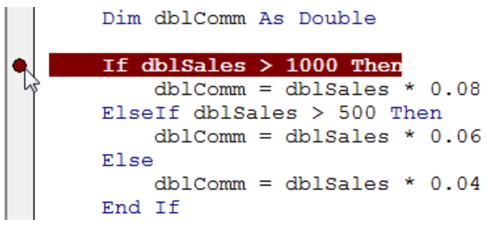
Whether your typed your own code or it was copied from a Web site and you're experimenting with it Stepping Into your code is an extremely powerful feature if you want to:

1. Learn how in works
2. Find a bug in your procedure
3. Examine the value of a variable at runtime

Please click in the grey margin on the left side of the code window where the **If** statement is located as shown here.

Doing this inserts a **Break Point** in the code. The code in that line is highlighted with a burgundy colour.

When you run your function and a break point is found, the execution halts at that line. Note that the line where the break point is found *is not* executed.



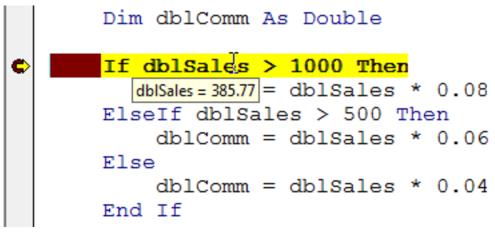
```
    Dim dblComm As Double

    If dblSales > 1000 Then
        dblComm = dblSales * 0.08
    ElseIf dblSales > 500 Then
        dblComm = dblSales * 0.06
    Else
        dblComm = dblSales * 0.04
    End If
```

In the Immediate Window (press Ctrl+G if it is not already open) type your function with any value as the argument and hit Enter. For example:

```
?Commission(385.77)
```

The line with a break point is highlighted in Yellow. You are now in **Break Mode** the execution is paused. Look in the Title Bar next to the name of your workbook. You will see the word [break] after the .xlsm

In this mode there are two important things you can do. View the value of a live variable and run the code *one line at a time* by stepping through each line while the procedure in live!



```
Dim dblComm As Double

If dblSales > 1000 Then
    dblComm = dblSales * 0.08
ElseIf dblSales > 500 Then
    dblComm = dblSales * 0.06
Else
    dblComm = dblSales * 0.04
End If
```

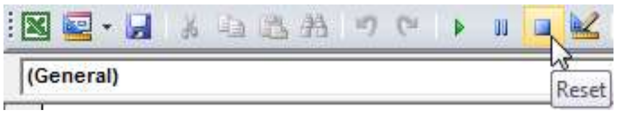
Move your mouse over one instance of the parameter **dblSales.** A screen tip appears under the mouse pointer showing the value of the argument you type when you launched it.

Now press the **F8** key a couple of times. The yellow bar moves down through the function. The F8 key is use to **Step In** the code one line at a time. Remember that the yellow line is only executed *after* you hit the F8 key.

If you enter the same parameter as the one above, pressing F8 will skip the statements for the first two conditions but execute the one in the Else clause. Hit F8 until you execute the rest of the lines.

Prematurely Stopping Execution of Some Code

You can end up in Break Mode for reasons other than having a **Break Point** in your code. Maybe the compiler found a nasty bug in your procedure and forced it to halt prematurely.



For whatever reason, if your code is in break mode and you want to stop the procedure click the **Reset** button on the tool bar.

Using a Select Case Structure

The next example uses various possibilities of State to determine the markup to charge the customer for some order. Look at the function below:

```
Function StateMarkup(strState As String) As Single
' Returns the markup to be charge depending on the state.

    Select Case strState
        Case Is = "VIC"
            StateMarkup = 0.5
        Case Is = "NSW", "SA"
            StateMarkup = 0.7
        Case Is = "QLD"
            StateMarkup = 0.4
        Case Is = "NT", "WA"
            StateMarkup = 0.6
        Case Else
            StateMarkup = 0.75
    End Select
End Function
```

Everything about this function is standard. However it uses a different approach to determine the markup, the Select Case structure.

The first line in this conditional structure:

```
Select Case strState
```

Simply mean "*Select* the *case* below that matches the value of the parameter strState". You can group together the various choices that share the same *outcome* by separating them with a comma.

Select Case is a tad different from the If-Then-Else structure. The **Using Conditional Statements** document highlights that difference.

How Many Days in a Month

If you need that information in your Excel workbooks then a function can do the job rather easily. There are ways to achieve the same results using Excel built-in functions but let's do it with a simple VBA function and you will learn something.

Type the following function in a module:

```
Function NumberOfDays(datDate As Date) As Byte
' Returns the number of days in a month.
' Supply a full date as the parameter.
' In Immediate Window type: NumberOfDays(#4/1/2016#) returns 30.

    NumberOfDays = Day(DateSerial(Year(datDate), Month(datDate) + 1, 1) - 1)
End Function
```

Note: Hard coded dates enter in VBA must use the format **month/day/year**.

The DateSerial() VBA function is extremely useful to create date using some calculation. It's just its name that is a bit strange. The syntax for **DateSerial** is:

```
DateSerial(Year, Month, Day)
```

That makes sense but the interesting thing it that you can use arithmetic expression for any of the parameters. It you can use expressions; you can use *variables* or *constants*.

```
DateSerial(Year(date)+2, Month(date)-5, 15)
```

This expression will return the date 2 years in the future, 5 months before the current month but on the 15.

One way to find the number of days in a month is to create a date on the 1st of the following month and move back *one* day. We know that this would be the *last* date of the month. Finally we would extract the *Day* part of that date. Look at the following expression:

```
DateSerial(Year(datDate), Month(datDate) + 1, 1) – 1
```

Here DateSerial builds a date using the **year** of the date passed as the argument, then one **month** after the argument then the 1st of that month. The minus 1 at the end moves back one day which brings us to the last date of the month passed as the argument.

To extract the day part of that date we use the **Day** function. This will return the last day (or how many days) in the month. Thus the expression:

```
Day(DateSerial(Year(datDate), Month(datDate) + 1, 1) - 1)
```

A variation of this function is 'How Many Days Left in the current Month'. Knowing the last date of the current month and the current date we can subtract the current date from the last date of the month to obtain the number of days left.

Look at the following example:

```
Function DaysLeftInMonth() As Byte
```

```
' How many days left in the current month.

    DaysLeftInMonth = (DateSerial(Year(Date), Month(Date) + 1, 1) - 1) - Date
End Function
```

This function should be fairly easy to understand.

Recalculating All Functions in a Workbook

If you've made amendments to a function that you used dozen times in a workbook you can easily update all the cells using this function by holding Ctrl+Alt+F9.

Just to make to point even stronger imagine you work hard making an Excel formula containing 5 nested IF() functions with some containing AND() as well as OR(). That formula is used in 83 cells in various worksheets.

You manager tells you that there was a misunderstanding and important amendments need to be done to the formula. Nightmare! You have to find all of them and fix manually. You wonder if the Replace command may help but you realize that the cell references are different in each formula!

No worries! You learned in you Excel VBA course that you simply need to make all the changes in the code then switch to the workbook and press Ctrl+Alt+F9. Presto!

How Old Are You?

This question may not be politically correct but it may be useful when you want to know how many years an employee worked for you or the age of an asset for depreciation purpose.

One way you may think will give you the correct result if by subtracting the year of birth (or being hired) from the current year. Unfortunately, in most case, this will not return the correct result unless you were born on a 31-December. Let's see if VBA can help.

In the Immediate Window type:

```
?DateDiff("yyyy",#08/21/1956#, Date)

60
```

When I typed this expression we were mid-January 2016. If the date in August is my birthdate I know that in mid-January I was *not* 60 years old but still 59! So the logic to keep in mind is as long as the current date is *before* the anniversary date I'm 59 (60-1).

Let see the correct solution using a bit of VBA. The DateSerial function will prove useful again:

```
Function CalcAge(datBirth As Date) As Byte
' Return the exact age.

    Dim intYearDiff As Integer    ' Difference between years.
    intYearDiff = DateDiff("yyyy", datBirth, Date)

    ' If today is prior to birthday then subtract 1.
    If Date < DateSerial(Year(Date), Month(datBirth), Day(datBirth)) Then
        intYearDiff = intYearDiff - 1
    End If
    CalcAge = intYearDiff

End Function
```

With this function I am 59 until the beginning of the day on 30 August 2016.

You can use this function in Excel with a couple of dates in a column and verify that the age for each date is accurate.

Conclusion

One of the best (and most enjoyable) ways to learn VBA in Excel is to start with a couple of cool (and useful) functions that you could use in your work.

Later you will hopefully develop the taste for Subs or Actions for use in your workbooks.

Daniel Lamarche

ComboProjects
Understanding does matter