Functions in VBA

Attaching a macro to a button or other object allows you and your users to run VBA code, but there is another way to access code from an Excel workbook: a *user-defined function* (UDF).

Unlike a native function, like =RAND() or =SUM(cell range), which is part of the Excel application itself, UDFs are functions that you write in VBA and are stored in a module in a macro-enabled workbook (filename extension .xlsm).

To the user, UDFs work the same as native functions, so they require no special knowledge. The only extra step is that the user has to enable the content in the workbook (or disable all security, which is not recommended).

To demonstrate how UDFs work, we will create two user-defined functions, the first without any arguments and the second will allow the user to pass information to the function, making it more flexible and useful.

UDF with No Arguments

Recall from our work on Monte Carlo simulation that we can create a 90% free throw shooter in Excel with the formula, *=IF(RAND()<0.9,1,0)*. Alternatively, we could create a function in VBA, say *FT()*, that the user could enter in any cell. Ninety percent of the time it would produce a 1 and the remaining ten percent would be 0.

STEP Open your *IntroVBA.xlsm* macro-enabled workbook and press alt-F11 to go to Visual Basic. Insert a new module sheet in this workbook. Type the text *function FT()* (uppercase FT) and hit enter.

Excel's Visual Basic Editor capitalizes the F in *function*, adds an *End Function* for you, and colors the text blue for the beginning and ending lines of the code.

Instead of the Excel function *=RAND()*, VBA has its own random number generator *Rnd*. It produces random numbers in the interval from 0 to 1. It is super easy to make our UDF output a random number.

STEP Enter the text $FT = Rnd$ and press enter. Return to Excel (alt-F11 is a toggle) and click on cell H1. Enter the function *=FT()* and press enter.

Congratulations! You just wrote your first function in Excel and accessed it from Excel.

EXCEL TIP For UDFs, Excel remembers how you first entered the function. If you enter *=ft()* (lowercase), it will keep using lowercase (even if you enter FT in a different cell). This is not true for native functions: enter *=rand()* and Excel converts it to *=RAND()*.

There is, however, a problem with our UDF.

STEP Press F9 a few times. Nothing happens to cell H1. It is not bouncing like *RAND()*, generating a new random number each time we recalculate the sheet.

The problem is that UDFs, by default, are non-volatile functions. This means they do not get recalculated unless they depend on other cells that have changed. We must add code to make our function recalculate when F9 is pressed.

STEP Return to your VBA code for the FT function. Click on the top line, after the close parenthesis and press enter so that you are on a new, blank line and enter the text *Application.Volatile True*. Press enter. Return to Excel and press F9 a few times. Success!

But we do not want to simply output a random number, we want to see if we made (1) or missed (0) a free throw attempt. We need to add some code to do this. Like the Excel formula we used to model a free throw result, we need an *If* statement to separate made from missed free throws.

STEP Go to the *FT* code. Click after *True* and press enter to create a new line after the volatile statement. Enter the text *if rnd < 0.9 then* and press enter.

As usual, the editor capitalizes and colors the text for you. The next line of code defines what happens if the random number is less than 0.9. It is followed by lines of code for missed free throws.

STEP Press the *tab* key to indent, type the text *FT = 1*, and press enter. Type the word *Else* and press enter. Press *tab*, type *FT = 0*, and press enter. Type *end if* and press enter. Put an apostrophe (') in front of the *FT=Rnd* line to comment it out (so it does not get executed).

Your masterpiece of code should look like this (with color added to keywords and lines):

```
Function FT()
Application.Volatile True
If Rnd < 0.9 Then
     FT = 1
Else
     FT = 0
End If
'FT = Rnd
End Function
```

The macro draws a uniformly distributed random number on the interval from 0 to 1 (*Rnd*) and if it is less than 0.9 it goes to the *FT = 1* line. Since the function is called *FT*, it outputs the number 1 if *Rnd* < 0.9. If the random number drawn is not less than 0.9, it goes to the *FT = 0* line and outputs a 0.

The code is easy to read, but does it actually work? Let's find out.

STEP Return to Excel, fill cell H1 down to cell H10 and press F9 a few times.

That is pretty cool, but what if we wanted a more generalized version, where the user tells us the chances of success?

UDF with an Argument

In native Excel functions, like *SUM*, for example, the arguments are passed in the parentheses: *SUM(A1:A3)*. UDFs work the same way. We will add an argument to our code in the parentheses and modify the code to enable it to incorporate the information provided by the user.

STEP Copy the *FT* code and paste it below the *End Function* line. Since you cannot have two functions with the same name, change the name of the newly pasted function to *FTARG* (for argument). In the parentheses, type the text *Shoot as Double*. Replace the *0.9* in the *If* statement line with *Shoot*. Return to Excel and enter the formula *=FTARG(0.5)* in cell I1.

Excel displays an error message in cell I1. Can you figure out what is wrong with the UDF and fix it? If you cannot, take a look at the appendix.

We can make the function even more flexible by having it accept a value in another cell.

STEP Copy the *FTARG* code and paste it below the *End Function* line. Change the name of the newly pasted function to *FTARGCELL* (for argument from another cell). In the parentheses, change the text to *myShootCell as Range*. Replace *Shoot* in the *If* statement line with *myShootCell.Value*. Change the *FTARG = 1* and *FTARG = 0* lines to *FTARGCELL = 1* and *FTARGCELL = 0*. Return to Excel and enter the formula *=FTARGCELL(K1)* in cell J1. Fill it down to cell J10. In cell K1, enter a number from 0 to 1, such as 0.25. Press F9 a few times.

With a 25% chance of success, your ten numbers in column J are bouncing around every time you press F9 and you usually get 2 or 3 ones.

Is it better to allow the user to input a number as the argument or a cell address? That depends on the context of the problem, including the user's familiarity with Excel. In fact, if you do not need to change the success rate, our original UDF, *FT()*, might be the best choice.

Takeaways

VBA code can be accessed by users from Excel's front-end by attaching macros to buttons and other controls.

Another option is a user-defined function or UDF. The user enters a formula that runs the UDF code to do computations or other manipulations that are not available in native Excel functions.

Like native Excel functions, UDFs usually require arguments. These variables are declared in the *Function* statement, inside the parentheses.

Writing code is science and art. Deciding whether arguments are needed and, if so, how to pass them (numbers or cell addresses, for example) requires knowledge of what needs to be done and who is going to do it. You need to know your audience.

Most people think of Excel as some kind of sophisticated adding machine or calculator. Excel can certainly do arithmetic and other mathematical operations, but its back-end or kitchen opens up a whole new world of opportunities and possibilities.

Appendix

The *FTARG* function fails when you change the name of the function to *FTARG*, but do not likewise update the name of the function in the code. You must change the *FT = 1* line to *FTARG = 1* and the *FT = 0* line to *FTARG = 0*.

When you write a UDF, you always use the function name to output a result. If you need more than one cell to output results, you can use an array function.