Once you become acquainted with VBA, you'll wonder how you ever got along without it.

Reed Jacobson

Introduction to VBA

You have learned many new Excel functions, created a variety of different charts in Excel, and used Solver, Comparative Statics Wizard, Monte Carlo simulation, and FRED Excel add-ins. But those are all frontend Excel skills.

For computer scientists, the frontend is the presentation layer or the user interface of software. The frontend is that part of the restaurant where the diner sits down and orders food.

The backend of a restaurant is the kitchen where the meal is actually made. Excel's backend is built on a computer language called Visual Basic. Excel and other Office products such as Word and PowerPoint run a version called Visual Basic Applications edition or VBA, for short.

It is in VBA that we write code, also known as macros, to perform especially complicated tasks. The code is written and stored in *modules*. The resulting Excel workbook is a combination of spreadsheets and modules that must be saved as a macro-enabled workbook with extension *.xlsm*. Without macros, an Excel spreadsheet has file extension *.xlsx*.

EXCEL TIP Excel warns you if you save a workbook with macros as a simple Excel workbook (.xlsx). If you ignore the warning, it will simply save the workbook without the macros and your code will be lost.

There is no question that a macro-enabled workbook is more powerful than a simple spreadsheet, but this power comes at a cost. Not only do you have to know how to write code in VBA, but your end-user will probably have to enable macros when opening the file. Sometimes, security settings in a particular installation of Excel are set so high that the macros will not be allowed to run. The user has to change Excel's settings, adding another layer of difficulty, just to open the file.

EXCEL TIP If you can accomplish a task without macros, *always* do so. Sometimes, however, VBA is the only solution.

Hello, World!

It is a tradition in computer science to introduce a new language by outputting "Hello, World!" Let's do it.

STEP Open Excel and save the file as *IntroVBA.xlsm*, making sure to save it as a macro-enabled workbook with the .xlsm extension by clicking the *Save as type* in the save window and choosing *.xlsm*. Click the Visual Basic button in the *Developer* tab in the *Ribbon*. If the Developer tab is not visible, press *alt, f, t* and click *Customize Ribbon*, then check *Developer* in the list. You can also access Vosual Basic by pressing *alt-F11* (you may need to use the *function* (*fn*) key) or right-clicking the sheet tab and selecting *View Code*.

You have a new window on your screen with a lot of things you have never seen before. Welcome to Excel's kitchen! You are in the Visual Basic Editor (VBE).

Across the top there is a familiar menu of items. The top-left panel should be the *Project Explorer* (press ctrl-r if you do not see it). You may also see other panels.

STEP In the *Project Explorer* panel, scroll, if needed, to find your *IntroVBA.xlsm* workbook (it will be in parentheses after VBAProject), and select it (highlighted in blue). Click *Insert* in the top menu and select *Module*.

You now see a blank window. This is where you will write your code. Notice also that your workbook in the *Project Explorer* panel now has a new component, the module you just inserted.

STEP The cursor should be blinking in the blank window, but if not, click in the window. Enter the text *sub myfirstmacro* and press enter.

The text is transformed. The *S* is capitalized and *Sub* is in blue, parentheses have been appended, and a new line *End Sub* has been added. Apparently, VBE is a high-level editor with a great deal of support.

Sub stands for subroutine, a set of instructions or lines of code. The statements between the *Sub* and *End Sub* lines are the *body* of the macro. You could pass *arguments* to your *Sub* by entering them in the () on the first line.

STEP In the middle line (where the cursor is blinking), enter the text *msgbox "Hello, World!"* and press enter.

As you entered the text, you undoubtedly noticed the yellow pop-up showing the various options for the *MsgBox* object. This shows again the strong support offered in the editor in the VBA environment. Also, this example reveals that VBA is an object-oriented programming language. We write code to apply different methods and options to the objects.

We have finished our first macro and are ready to run it.

STEP Click *Run* in the top menu and select *Run Sub/User Firm* (or press F5).

You did it! You are returned to the Excel spreadsheet and it displays a message box with *"Hello, World!"* on it.

This is exciting, but how can we run the macro from the spreadsheet?

STEP Press OK to close the message box and click the *Developer* tab. Click the Insert button and select the top-left *Button* icon. Click and drag in the spreadsheet to create a button. In the *AssignMacro* dialog box, select *myfirstmacro* and click OK. Click on an empty cell in the spreadsheet.

You have added a button to the spreadsheet and attached a macro to it. When you click the button, the macro will run. Try it.

STEP Click the button to see the message box pop up.

Now the user does not have to know how to run macros in VBA. By attaching the macro to the button, you have made it easy for the user to run your code.

EXCEL TIP Regular (left) clicks enable you to use objects in Excel. Right-clicks select objects so that you can modify them.

STEP Right-click your button and replace the *Button 1* text with *Click Me*. Click the button to see that it works.

This shows that the caption of the button can be different from the name of the macro.

We used scroll bar and combo box controls earlier, but they were not macro-enabled. By assigning macros to controls, we greatly expand the power of Excel.

Recording Macros

VBA is hard at first because beginning users do not know any of the objects or commands. It is like learning a new spoken language, you know the words exist for what you want to say, but you do not know what they are.

One way to start growing your VBA vocabulary is by recording macros. You turn on the recorder and do things in Excel, then examine the recorded code in VBA.

STEP Click the *Developer* tab and click the Record Macro button. Click OK in the pop up dialog box. Select cell A1 and enter *100*. Make the formatting $. Click the Stop Recording button. Press alt-F11 to go to VBA.

There is a new module sheet in your VBA project.

STEP Double-click *Module2* in the *Project Explorer* panel.

You are looking at the code needed to do the steps you did in Excel. You now know that *Selection.Style = "Currency"* applies currency formatting to the selected cell.

STEP Change the *100* in the recorded macro to *0.1* and change *Currency* to *Percent*. Return to Excel by pressing alt-F11 (this toggles you between Excel and VBA). Right-click your button and select *AssignMacro*. Choose *Macro1* (that you just recorded and edited) and click *OK*. Click an empty cell in the spreadsheet and click your button.

Your macro changed A1 to 0.1 and formatted it as percent! This demonstrates that you can definitely control Excel from VBA.

This example shows how you can record a macro to reveal the code needed to perform a task in Excel. The usual procedure is to record a series of steps and then edit the code, removing superfluous lines and changing values or other attributes.

STEP Use the macro recorder to write a macro that takes a random number from a cell and pastes its value in the cell below it. Assign your macro to your button and click the button to run it. If you need help, see the appendix.

Takeaways

In the late 20th century, Excel was in a race with other spreadsheets that you have never heard of because Excel completely overwhelmed them. In 1993, Excel 5.0 debuted with VBA and Microsoft crushed the competition.

VBA is an implementation of Visual Basic that runs within Excel. Macros are written in VBA in modules and then assigned to controls (such as buttons) on the spreadsheet.

If you want to continue learning about VBA, Jacobson's *Step-by-Step* book is a great place to start. The files needed were originally on a CD (an older technology that could store "huge" amounts of data), but they are available for download at tiny.cc/hbvba.

You can also look at open-source VBA code. For example, the Comparative Statics Wizard and Monte Carlo simulation add-ins are freely accessible. So are the macro-enabled workbooks we have used. You can open their modules and inspect the code to learn VBA.

References

The epigraph is from the first chapter of Reed Jacobson's *Microsoft Office Excel 2007 Visual Basic for Applications Step by Step* (Pearson Education) Kindle Edition. This book is out of print, but there are many copies available and it remains a great way to learn how to write macros in Excel.

Appendix

STEP Click the Record Macro button and enter the formula *=RAND()* in a cell. Copy the cell, select another cell, and *Paste Special* as *Values*. Stop recording and go to VBA. The body of your recorded macro (the code between the *Sub* and *End Sub* lines) should look something like this:

```
'
' Macro3 Macro
'

'
Range("A3").Select
ActiveCell.FormulaR1C1 = "=RAND()"
Range("A3").Select
Selection.Copy
Range("A4").Select
Selection.PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, _
SkipBlanks :=False, Transpose:=False
Range("A5").Select\\
```

These lines are the actual commands you gave Excel. Notice that an apostrophe at the start of a line comments out that line (it is not executed) and is displayed in green-colored text (e.g., the line Macro3 Macro). The underscore character, _, continues the current statement on the next line.

You could clean up this code so it looks like this:

```
Range("A3").FormulaR1C1 = "=RAND()"
Range("A3").Copy
Range("A4").PasteSpecial Paste:=xlPasteValues, Operation:=xlNone, _
SkipBlanks:=False, Transpose:=False
Range("A5").Select
```

You could comment out the first line and it would still work since the *RAND()* function is already in cell A3. Running this macro replaces the contents of cell A3 without any warning. You do not notice this because the code writes the same formula that was there.