IN THIS CHAPTER

- Getting a conceptual overview of VBA
- Finding out what you can do with VBA
- Discovering the advantages and disadvantages of using VBA
- Getting the lowdown on what VBA is
- Staying Excel compatible

Chapter 1
What Is VBA?

If you're eager to jump into VBA programming, hold your horses. This chapter is completely devoid of any hands-on training material. It does, however, contain some essential background information that assists you in becoming an Excel programmer. In other words, this chapter paves the way for everything else that follows and gives you a feel for how Excel programming fits into the overall scheme of the universe. It's not as boring as you might think, so please try to resist the urge to jump to Chapter 2.

Okay, So What Is VBA?

VBA, which stands for *Visual Basic for Applications,* is a programming language developed by Microsoft — you know, the company that tries to get you to buy a new version of Windows every few years. Excel, along with the other members of Microsoft Office, includes the VBA language (at no extra charge). In a nutshell, VBA is the tool that people use to develop programs that control Excel.

Imagine an intelligent robot that knows all about Excel. This robot can read instructions, and it can also operate Excel very fast and accurately. When you want the robot to do something in Excel, you write a set of robot instructions by using special codes. Then you tell the robot to follow your instructions while you sit back and drink a glass of lemonade. That's kind of what VBA is all about — a code language for robots. Note, however, that Excel does not come with a robot or lemonade.

A FEW WORDS ABOUT TERMINOLOGY

Excel programming terminology can be a bit confusing. For example, VBA is a programming language, but it also serves as a macro language. In this context, a macro is a set of instructions Excel performs to imitate keystrokes and mouse actions. What do you call something written in VBA and executed in Excel? Is it a *macro,* or is it a *program?* Excel's Help system often refers to VBA procedures as *macros,* so that terminology is used in this book. But you can also call this stuff a *program.*

You'll see the term *automate* throughout this book. This term means that a series of steps are completed automatically. For example, if you write a macro that adds color to some cells, prints the worksheet, and then removes the color, you have *automated* those three steps.

By the way, *macro* doesn't stand for **M**essy **A**nd **C**onfusing **R**epeated **O**peration. Rather, it comes from the Greek *makros,* which means large — which also describes your paycheck after you become an expert macro programmer.

What Can You Do with VBA?

You're probably aware that people use Excel for thousands of different tasks. Here are just a few examples:

- Analyzing scientific data
- Budgeting and forecasting
- Creating invoices and other forms
- Developing charts from data
- Keeping lists of things such as customers' names, students' grades, or holiday gift ideas (a nice fruitcake would be lovely)
- Yadda, yadda, yadda

The list could go on and on, but you get the idea. The point is simply that Excel is used for a wide variety of tasks, and everyone reading this book has different needs and expectations regarding Excel. One thing virtually every reader has in common is the *need to automate some aspect of Excel.* That, dear reader, is what VBA is all about.

For example, you might create a VBA program to import some numbers and then format and print your month-end sales report. After developing and testing the program, you can execute the macro with a single command, causing Excel to automatically perform many time-consuming procedures. Rather than struggle through a tedious sequence of commands, you can click a button and then hop on over to Facebook and kill some time while your macro does the work.

The following sections briefly describe some common uses for VBA macros. One or two of these may push your button.

Inserting a bunch of text

If you often need to enter your company name, address, and phone number in your worksheets, you can create a macro to do the typing for you. You can extend this concept as far as you like. For example, you might develop a macro that automatically enters a list of all salespeople who work for your company.

Automating a task you perform frequently

Assume you're a sales manager and you need to prepare a month-end sales report to keep your boss happy. If the task is straightforward, you can develop a VBA program to do it for you. Your boss will be impressed by the consistently high quality of your reports, and you'll be promoted to a new job for which you are highly unqualified.

Automating repetitive operations

If you need to perform the same action on, say, 12 different Excel workbooks, you can record a macro while you perform the task on the first workbook and then let the macro repeat your action on the other workbooks. The nice thing about this is that Excel never complains about being bored. Excel's macro recorder is similar to recording live action on a video recorder. But it doesn't require a camera, and the battery never needs to be recharged.

Creating a custom command

Do you often issue the same sequence of Excel commands? If so, save yourself a few seconds by developing a macro that combines these commands into a single custom command, which you can execute with a single keystroke or button click. You probably won't save *that* much time, but you'll probably be more accurate. And the guy in the next cubicle will be really impressed.

Creating a custom button

You can customize your Quick Access toolbar with your own buttons that execute the macros you write. Office workers tend to be very impressed by buttons that perform magic. And if you *really* want to impress your fellow employees, you can even add new buttons to the Ribbon.

Developing new worksheet functions

Although Excel includes hundreds of built-in functions (such as SUM and AVERAGE), you can create *custom* worksheet functions that can greatly simplify your formulas. You'll be surprised by how easy this is. (You can explore how to do this in Chapter 20.) Even better, the Insert Function dialog box displays your custom functions, making them appear built-in. Very snazzy stuff.

Creating custom add-ins for Excel

You're probably familiar with some of the add-ins that ship with Excel. For example, the Analysis ToolPak is a popular add-in. You can use VBA to develop your own special-purpose add-ins.

Advantages and Disadvantages of VBA

This section outlines the good things about VBA — and also its darker side.

VBA advantages

You can automate almost anything you do in Excel. To do so, you write instructions that Excel carries out. Automating a task by using VBA offers several advantages:

- Excel always executes the task in exactly the same way. (In most cases, consistency is a good thing.)
- Excel performs the task much faster than you can do it manually. (Unless, of course, you're Clark Kent.)
- If you're a good macro programmer, Excel always performs the task without errors (which probably can't be said about you, no matter how careful you are).

- If you set up things properly, someone who doesn't know anything about Excel can perform the task by running the macro.
- You can do things in Excel that are otherwise impossible — which can make you a very popular person around the office.
- For long, time-consuming tasks, you don't have to sit in front of your computer and get bored. Excel does the work while you hang out at the water cooler.

VBA disadvantages

It's only fair to give equal time to the disadvantages (or *potential* disadvantages) of VBA:

- You have to know how to write programs in VBA (but that's why you bought this book, right?). Fortunately, it's not as difficult as you might expect.
- Other people who need to use your VBA programs must have their own copies of Excel. It would be nice if you could press a button that transforms your Excel/VBA application into a stand-alone program, but that isn't possible. (And probably never will be.)
- Sometimes, things go wrong. In other words, you can't blindly assume that your VBA program will always work correctly under all circumstances. Welcome to the world of debugging and, if others are using your macros, technical support.
- VBA is a moving target. As you know, Microsoft is continually upgrading Excel. Even though Microsoft puts great effort into compatibility between versions, you may discover that the VBA code you've written doesn't work properly with older versions or with a future version of Excel.

VBA in a Nutshell

Just to let you know what you're in for, here's a quick-and-dirty summary of what VBA is all about.

- **You perform actions in VBA by writing (or recording) code in a VBA module.** You view and edit VBA modules by using the Visual Basic Editor (VBE).

- **A VBA module consists of Sub procedures.** A Sub procedure has nothing to do with underwater vessels or tasty sandwiches. Rather, it's a chunk of computer code that performs some action on or with objects (discussed in a moment). The following example shows a simple Sub procedure called AddEmUp. This amazing program, when executed, displays the result of 1 plus 1:

```
Sub AddEmUp()
    Sum = 1 + 1
    MsgBox "The answer is " & Sum
End Sub
```

  A Sub procedure that doesn't perform properly is said to be substandard.

- **A VBA module can also have Function procedures.** A Function procedure returns a single value. You can call it from another VBA procedure or even use it as a function in a worksheet formula. An example of a Function procedure (named AddTwo) follows. This Function accepts two numbers (called arguments) and returns the sum of those values:

```
Function AddTwo(arg1, arg2)
    AddTwo = arg1 + arg2
End Function
```

  A Function procedure that doesn't work correctly is said to be dysfunctional.

- **VBA manipulates objects.** Excel provides dozens and dozens of objects that you can manipulate. Examples of objects include a workbook, a worksheet, a cell range, a chart, and a shape. You have many more objects at your disposal, and you can manipulate them by using VBA code.

- **Objects are arranged in a hierarchy.** Objects can act as *containers* for other objects. At the top of the object hierarchy is Excel. Excel itself is an object called Application. The Application object contains other objects, such as Workbook objects and Add-In objects. The Workbook object can contain other objects, such as Worksheet objects and Chart objects. A Worksheet object can contain objects such as Range objects and PivotTable objects. The term *object model* refers to the arrangement of these objects. (Object-model mavens can find out more in Chapter 4.)

- **Objects of the same type form a collection.** For example, the Worksheets collection consists of all the worksheets in a particular workbook. The Charts collection consists of all Chart objects in a workbook. Collections are themselves objects.

- **You refer to an object by specifying its position in the object hierarchy, using a dot (aka a period) as a separator.** For example, you can refer to the workbook Book1.xlsx as

```
Application.Workbooks("Book1.xlsx")
```

  This refers to the workbook `Book1.xlsx` in the Workbooks collection. The Workbooks collection is contained in the Application object (that is, Excel). Extending this to another level, you can refer to Sheet1 in `Book1.xlsx` as

```
Application.Workbooks("Book1.xlsx").Worksheets("Sheet1")
```

  You can take this to still another level and refer to a specific cell (in this case, cell A1):

```
Application.Workbooks("Book1.xlsx").Worksheets("Sheet1").
   Range("A1")
```

- **If you omit specific references, Excel uses the *active* objects.** If `Book1.xlsx` is the active workbook, you can simplify the preceding reference as follows:

```
Worksheets("Sheet1").Range("A1")
```

  If you know that Sheet1 is the active sheet, you can simplify the reference even more:

```
Range("A1")
```

- **Objects have properties.** You can think of a property as a *setting* for an object. For example, a Range object has such properties as Value and Address. A Chart object has such properties as HasTitle and Type. You can use VBA to determine object properties and also to change properties.
- **You refer to a property of an object by combining the object name with the property name, separated by a dot.** For example, you can refer to the Value property in cell A1 on Sheet1 as follows:

```
Worksheets("Sheet1").Range("A1").Value
```

- **You can assign values to variables.** A *variable* is a named element that stores information. You can use variables in your VBA code to store such things as values, text, and property settings. To assign the value in cell A1 on Sheet1 to a variable called *Interest,* use the following VBA statement:

```
Interest = Worksheets("Sheet1").Range("A1").Value
```

- **Objects have methods.** A *method* is an action Excel performs with an object. For example, one of the methods for a Range object is ClearContents. This aptly named method clears the contents of the range.
- **You specify a method by combining the object with the method, separated by a dot.** For example, the following statement clears the contents of cell A1:

```
Worksheets("Sheet1").Range("A1").ClearContents
```

- **VBA includes all the constructs of modern programming languages, including variables, arrays, and looping.** In other words, if you're willing to spend a little time learning the ropes, you can write code that does some incredible things.

Believe it or not, the preceding list pretty much describes VBA in a nutshell. Now you just have to find out the details. That's why this book has more pages.

Excel Compatibility


REMEMBER

This book is written for the desktop versions of Excel 2016 and Excel 2019. If you don't have one of those versions, you run the risk of getting confused in a few places.

If you plan to distribute your Excel/VBA files to other users, it's vitally important that you understand which versions of Excel they use. People using older versions won't be able to take advantage of features introduced in later versions. For example, if you write VBA code that references cell XFD1048576 (the last cell in a workbook), those who use a version prior to Excel 2007 will get an error because those pre-Excel 2007 worksheets had only 65,536 rows and 255 columns (the last cell is IV65536).

Excel 2010 and later also have some new objects, methods, and properties. If you use these in your code, users with an older version of Excel will get an error when they run your macro — and you'll get the blame.