IN THIS CHAPTER

- Forming 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
Getting to Know 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. Just like a legendary heart surgeon has to first take freshman biology, you must learn some basic concepts and terminology so that the more practical aspects you use later make sense.

Introducing VBA Basics

VBA, which stands for *Visual Basic for Applications,* is a programming language developed by Microsoft and built right into Excel (and a few other programs, like Word, Outlook, and PowerPoint) at no extra charge. In a nutshell, VBA is a tool that people use to write programs that automate Excel, such as a program to format a spreadsheet full of data into a monthly report. In the next section, I discuss in more detail the types of tasks you can automate with VBA.

Imagine a robot that knows all about Excel. This robot can read instructions, and it can also operate Excel quickly 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. With VBA, you don't need the extra chair at your desk for an actual robot. Just feed the special codes into VBA and it does the work.

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 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.

Knowing What VBA Can Do

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
- Building charts from data
- Keeping lists of items or info such as customers' names, students' grades, or holiday gift ideas (a nice fruitcake would be lovely)

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 may create a VBA program to import some numbers and then format and print your month-end sales report. After writing 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 simply click a button and then hop on over to TikTok 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. It happily performs the same actions and presses the same keystrokes over and over. And it's faster than a hundred caffeinated interns.

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 person sitting in the next cubicle will be duly 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 quite 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 offers 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 it is. (You can explore how to do it in Chapter 21.) Even better, the Insert Function dialog box displays your custom functions, making them appear built-in. Snazzy stuff.

Creating custom add-ins for Excel

You might be 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. Add-ins are just like regular Excel workbooks, except that they have no worksheets that users can see. Add-ins are all VBA, and they usually automate tasks on other workbooks.

Getting the Most from VBA

VBA provides a ton of benefits that are tempered by a few disadvantages you'll want to keep in mind. One advantage of VBA, for example, is that you can use it to automate almost anything you do in Excel. All you have to do is write instructions in VBA, and Excel carries out those instructions when you tell it to. The earlier section "Knowing What VBA Can Do" describes some specific tasks you can accomplish by using VBA. The following sections can help you decide whether VBA is the best tool to achieve the results you want.

Knowing what VBA does best

Here are some benefits of automating a task by using VBA:

- 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 knows nothing about Excel can perform the task by running the macro.
- You can perform tasks in Excel that are otherwise impossible — which can make you a 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.

Recognizing the disadvantages of using VBA

Using VBA can present some disadvantages (or *potential* disadvantages). Understanding these limitations upfront helps you avoid running down rabbit trails to nowhere. Most people use VBA to save time, not waste it!

Keep the following points in mind when deciding whether to use VBA:

- 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* (fixing errors) and, if others are using your macros, providing 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 might discover that the VBA code you've written doesn't work properly with older versions or with a future version of Excel. For more information on the importance of Excel compatibility, see the section "Ensuring Excel Compatibility," later in this chapter.

Understanding VBA Concepts

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 an 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 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.)

- **Collection objects hold other objects of the same type.** For example, the Worksheets collection consists of all 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 (entered as a period) as a separator.** For example, you can refer to the workbook Book1.xlsx as

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

  This line 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 *characteristic* of 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 read 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 numbers, text, and properties. 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 method 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 deletes the value in cell A1 using the ClearContents method:

```
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. Now you just have to discover the details by reading the rest of this book.

Ensuring Excel Compatibility

REMEMBER

This book is written for the desktop version of Excel 365. If you don't have that version, there's a slight risk you'll get confused in a few places, but mostly it will just work.

WARNING

If you plan to distribute your Excel/VBA files to other users, you *must* understand which versions of Excel they use. People using older versions can't 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 earlier than Excel 2007 will receive an error because those pre-Excel 2007 worksheets had only 65,536 rows and 255 columns (the last cell is IV65536).

Newer versions sometimes introduce new objects, methods, and properties. If you use these in your code, users with an older version of Excel will receive an error when they run your macro — and you'll get the blame.