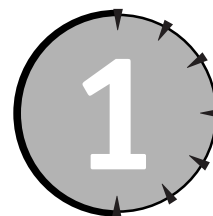


SESSION



Microsoft Excel Programming — Why and How

Session Checklist

- ✓ The advantages of Excel programming
- ✓ Fundamentals of programming
- ✓ The Excel object model
- ✓ Programming and macros
- ✓ Designing your custom application
- ✓ Your first Excel program



30 Min.
To Go

Most people think of Excel as merely a spreadsheet program, and with good reason — Excel *is* a spreadsheet program. As a spreadsheet program, Excel is a powerful application that provides a wide range of tools for the manipulation, analysis, and display of data. The majority of users never go beyond using Excel in this way — truth be told, many users have no need.

Under the surface, however, Excel is much more than an application program. It provides a sophisticated programming language that enables you to control any and all aspects of the program. Anything you can do with the keyboard and mouse you can also do with programming. For the power user, programming turns Excel into a flexible development tool for the creation of custom solutions to your data manipulation and analysis needs. This session takes a look at the advantages of Excel programming, and provides necessary background information on the technologies that are involved.

Advantages of Programming

Programming offers several important advantages to the Excel user. As mentioned earlier in this session, these advantages will not be relevant to all users, but they can apply to a surprisingly large percentage of situations.

Saving Time

Just about anything that a program can do in Excel can also be accomplished by a user with a keyboard and mouse. In a race, however, the program is always faster. Even if you are an expert user and do not need to spend any time figuring out how to perform the required tasks, a program will still beat you by a huge margin. What might take you half an hour to perform manually will be done in a few seconds by a program.

Reducing Errors

Even the most skilled typist hits the wrong key now and then, and every “mouse master” has been known to click the wrong button or command once in a while. In contrast, programs do not make mistakes — they are reliable servants, carrying out your commands over and over again with complete accuracy.

This is not to say that programs cannot contain errors. A program will do exactly what the programmer tells it to do; if your instructions are wrong or incomplete, the resulting program can cause errors. Dealing with program errors is an important topic, enough so that an entire session is devoted to it.

Enforcing Standards

In many organizations, adherence to data-processing standards is an important aspect of maximizing productivity. For example, each sales representative may be required to submit a weekly summary spreadsheet. If those spreadsheets all follow the same structure and format, it is fairly straightforward to automate the process of extracting data into a summary report. The slightest deviation from the proper format, however, is likely to throw a wrench into the gears. By using programming for the individual spreadsheets rather than manual data entry, you can ensure that there are no deviations from the correct spreadsheet format.

Integrating with Other Applications

Excel does not always work alone — it has the capability to share data and interact with other applications. These capabilities are most developed with, but not restricted to, other Microsoft Office applications. For example, an Excel program could use Outlook to create an e-mail message containing data from a spreadsheet and then send the message to a list of recipients. Programming is not required for Excel to interact with other programs, but it makes tasks possible that would be difficult or impractical to perform otherwise.

Programming Fundamentals

What exactly is programming? It's really not as mysterious as it may sound. Perhaps you already have some experience with computer programming of one sort or another. If not, this section gives you some background information about what programming is and how it works.

Creating Instructions

Programming is really nothing more than creating instructions that tell the computer what to do. In an office, for example, you might ask your assistant to make copies — that's an instruction. A computer program is the same — you tell the computer what to do. The primary difference is that computers are really dumb and can't figure out the fine points on their own, so you must tell them exactly what to do in excruciating detail. Explicit instructions are at the heart of any program.

Some instructions manipulate data. This can be as simple as adding two numbers, or as complex as creating a chart. Other instructions control the execution of the program itself. For example, a program could be designed to perform one task on weekdays and another task on weekends. Still other instructions control how the program interacts with the user, such as how it responds to the user's selections from a menu or dialog box.

Handling Data

Every computer program works with data. This data can come in many forms — text, numbers, and graphics — but for now you don't need to be concerned with these details. A fundamental part of programming consists of handling the data that the program uses. You need a specific place to keep the data, and you also need to be able to get at the data when necessary. From this perspective, data can be divided into two categories:

- **Data stored outside the program.** For the most part, this category comprises data that is stored in the cells of an Excel worksheet. Your program does not need to create storage for such data, although the program can read and write it.
- **Data stored within the program.** For data that is not stored elsewhere, a program needs to create a storage location. A program uses *variables* to store internal data. As you'll see in later sessions, Excel programming offers a wide range of internal data storage capabilities.

When you are programming with Excel, the data-handling capabilities available to you are quite impressive. They provide a variety of data types that are specialized for storing different kinds of data. You learn more about these data types in later sessions.

The VBA Language

You will use the VBA language to write Excel programs. VBA stands for Visual Basic for Applications, and it is one of the two essential parts of Excel programming. The name Visual Basic for Applications reflects the fact that VBA is based on Microsoft's Visual Basic programming language, and that it is designed for programming within applications — specifically, the applications that comprise the Microsoft Office suite (Excel, Word, Access, PowerPoint, and Outlook). VBA is relatively easy to learn, as programming languages go, but does not sacrifice power and flexibility.

The task of programming in VBA is simplified by the VBA Editor, which is part of your Excel installation. You can open the VBA Editor by pressing Alt+F11 when in Excel, or by

selecting Tools ⇨ Macro ⇨ Visual Basic Editor from the menu. The VBA Editor is shown in Figure 1-1. The blank window is where you enter your program's VBA code. Other elements in the Editor provide tools for organizing, running, and debugging your programs. Later sessions cover these features.

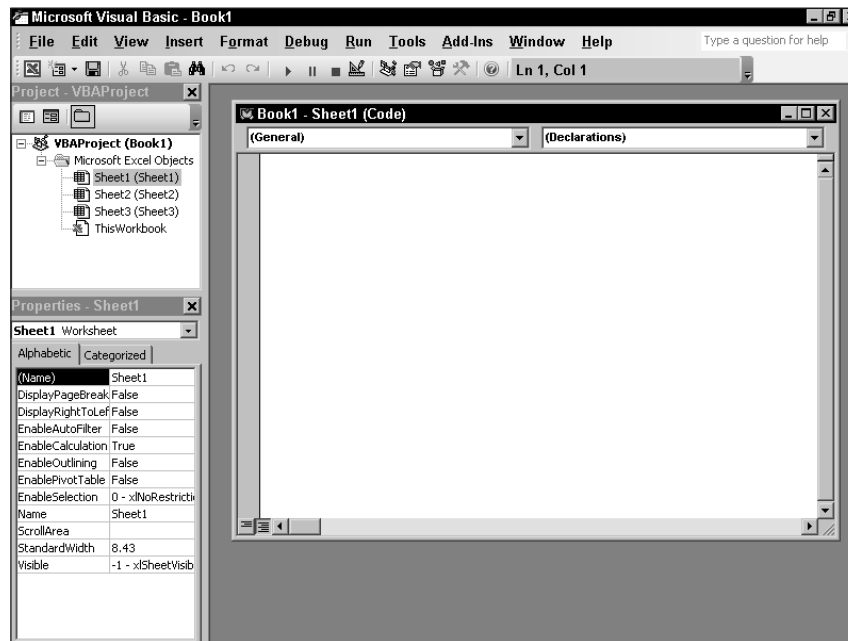


Figure 1-1 The VBA Editor



20 Min.
To Go

The Excel Object Model

The other essential component of Excel programming is the Excel object model. To understand the object model, it helps to have some background information about the inner workings of programs such as Excel.

Objects

As computer programming evolved over the years, programs have gotten more powerful and, unavoidably, more complicated. With increasing complexity came an increased possibility of errors, and programmers found themselves spending more and more time tracking down and fixing the causes of these errors. It soon became apparent that many, if not most, program errors were caused by unexpected and unintended interactions between various parts of a program. If a programmer could reduce or eliminate these interactions, errors would be drastically curtailed.

At the same time, programmers found themselves writing the same program functionality over and over again. Most Windows programs have a menu, for example, and programmers

Objects and Classes

You'll see the words *object* and *class* used in discussions of object-oriented programming, often interchangeably. Technically, they have different meanings. A class is a plan or definition, where an object is an implementation of that plan (sometimes called an *instance*). To use an analogy, the blueprints for a car would be the class, while an actual car built from those plans would be an object. You can create multiple objects from the same class.

would have to write the code for a menu from scratch for each new program. It would be much better if the code for a menu could be written once and then reused as needed in new programs.

These (and other) factors were the impetus behind the development of a programming technique called *object-oriented programming*, or OOP. With OOP, a program is viewed as a collection of related sections, or modules, that have different functions. Some of these modules are part of the program's interface, such as menus, toolbars, and dialog boxes. Other modules relate to the data with which the program works, such as (for Excel) workbooks, worksheets, and cells. Each of these modules is an object; see the "Objects and Classes" sidebar for more information on nomenclature.

OOP offers numerous advantages compared with older traditional methods of programming. These include:

- **Reduced errors.** By design, objects are self-contained units that are isolated from each other as much as possible. An object's interactions with the rest of the program are tightly controlled, and unintended interactions (and the resulting errors) are eliminated.
- **Code reuse.** An object — or more accurately, a class — is by its very nature reusable, not only in the same program, but in other programs as well.

Excel, as well as the other Office programs, was created using OOP techniques. Under the skin, therefore, Excel consists of a large collection of objects that work together to provide the program's functionality. You'll see how this relates to programming Excel in the next section.

Components and Automation

The objects that are part of the Excel application are written so that they are available to other programs. In computer talk, the objects are said to be *exposed*. This is part of the Component Object Model (COM) technology that is central to the Windows operating system itself as well as to most applications that run on Windows. The term *component* or *COM component* is used to refer to objects that are exposed in this manner; therefore, the objects that are exposed by Excel are sometimes referred to as components. Note that a single component may expose more than one class.

How does a programmer make use of exposed components? The answer is another COM technology called *automation* (called OLE — object linking and embedding — automation in the past). Automation permits an external program to access and control exposed

components. Automation also permits components to interact with each other — for example, you could embed an Excel spreadsheet in a Word document. For the present purposes, automation permits a VBA program to use the Excel components. Other programming languages, such as C++ or Java, can use automation too, but that is not relevant here.

The COM components that your VBA programs can use exist as files on your hard disk and were installed as part of the Office or Excel installation. A component can operate in two ways:

- An automation client controls and makes use of classes exposed by other components.
- An automation server exposes classes for use by other components.

An automation component can act in one or both of these roles. For programming Excel, VBA is acting as a client, and the Excel components are acting as servers. Some Excel components act as clients to manipulate other components.

The result of this arrangement is that an Excel programmer has VBA, a powerful programming language, as well as access to all of the components that comprise the Excel application. This is an extremely powerful combination — Excel is your well-trained and capable servant, and VBA is the means you use to tell it what to do. The sum of all the components exposed by Excel is referred to as the Excel object model.



The Excel object model is covered in detail later in the book, primarily in Session 3.

Macros and Programming

If you have ever used Excel's macro feature, you have already done some Excel programming. A macro is a sequence of user actions that is recorded and can be played back later to duplicate the original actions. This saves time because you don't have to manually redo the steps each time. To record a macro, select Macro from Excel's Tools menu and then select Record New Macro. As you perform actions in Excel, they are translated into the corresponding VBA commands. When you stop the recording, the resulting VBA is saved and can be played back as needed. Recording macros can be a useful tool for the Excel programmer.

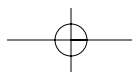


Recording macros is covered in more detail in Session 2.

Designing Your Custom Application

When creating a custom Excel program, as with all programming, it is important to do some planning before you start writing code. The importance of planning cannot be overemphasized. For a simple project it may take only a few minutes; for a large, complex project, it may take days. In either case it is going to save you time and hassles down the road.

Before you begin, make certain that you know what is needed. A lot of problems can arise when there is a misunderstanding between the client and the programmer. You may



write a great program, but if it is not what the customer needs you have wasted your time! Don't assume that you know what is wanted — make sure, in detail.

After you know your goal, start planning the details of the program. For simple programs this step may be trivial because there is only one way to accomplish the desired ends. With more complex programs, you have choices to make. How many worksheets and workbooks will be needed? Will any user forms be required? How will the functionality be divided up? Should you define any classes for the project? Of course, plans that you make at this stage are not set in stone — you can always modify them later as circumstances dictate. Even so, having some plan at the outset, even if it is an incomplete plan, is a real help.



**10 Min.
To Go**

Your First Excel Program

To give you a taste for Excel programming, this section presents a short program that illustrates some of the basic principles. You are not expected to understand all the details of the program at this point, but working through the required procedures will help in understanding the material in the following sessions.

This program does not do anything complicated. The desired goal is to create a program that does the following:

1. Starts with a blank worksheet.
2. Enables the user to enter a series of numbers in specified cells.
3. Calculates the sum of the numbers.
4. Saves the workbook to disk.

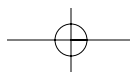
The following sections guide you through the process of creating and running this program.

Creating and Naming the Program

These first steps create the basic program structure and assign a name to it.

1. Start Excel. As usual, it starts with three blank workbooks displayed.
2. Press Alt+F11 to open the VBA Editor.
3. At the top left of the VBA Editor, there is a window labeled "Project - VBAProject" that lists the currently open worksheets: Sheet1, Sheet2, and Sheet3, as shown in Figure 1-2. Double-click the Sheet1 entry, and a code-editing window opens.
4. Select Procedure from the Insert menu. The Add Procedure dialog box displays. Enter a name for your program, such as MyFirstProgram. You can use letters and numbers, but no spaces (some programmers use an underscore in place of spaces to separate words).
5. Click OK. The VBA Editor inserts the first and last lines of the procedure (program) in the code-editing window (see Figure 1-3).

At this point, you have created the empty "skeleton" of your program. Next, you add the code that provides its functionality.



Programs and Procedures

Is a program the same thing as a procedure? Not really. Some simple programs such as this one are contained in a single procedure. More complex programs contain multiple procedures.

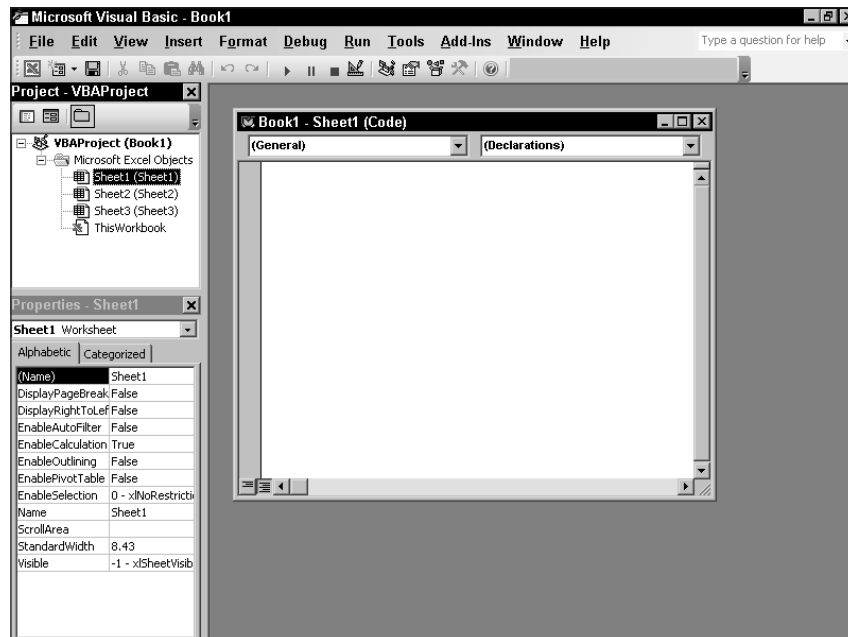


Figure 1-2 Double-click the Sheet1 entry to open a code-editing window.

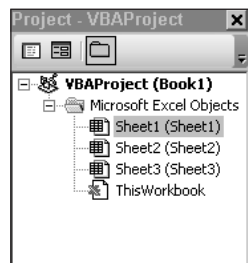


Figure 1-3 The VBA Editor inserts the first and last lines of the program.

Writing the Code

The tasks that the program's code needs to perform are as follows:

1. Move the cursor to cell B2.
2. Accept a number from the user, and enter it in the cell.
3. Move to cell B3.
4. Repeat until numbers have been entered in five cells: B2 through B6.
5. Enter a formula in cell B7 to add the column of numbers.
6. Save the workbook to disk.

The code uses several elements of VBA and the Excel object model. Before entering the code, you should have some idea of how it works:

```
Range("B2").Select
```

This code uses the Range object to move the Excel cursor to the indicated cell — in this case, cell B2.

```
ActiveCell.Value = InputBox("Enter Value")
```

This code has two parts. The InputBox function displays a dialog box on the screen and prompts the user to enter data, using the supplied text "Enter Value" as the prompt text. The ActiveCell object is then used to take the value entered by the user and insert it into the currently active worksheet cell (the cell that the cursor is on).

The above two code elements repeat five times to enter values in cells B2 through B6. After moving the cursor to the cell B7, the following code executes:

```
ActiveCell.Formula = "=Sum(B2..B6)"
```

Here, the ActiveCell object is used again, this time to enter a formula into the active worksheet cell. The formula uses Excel's built-in Sum function to calculate the sum of the values in cells B2 through B6.

```
ActiveWorkbook.SaveAs Filename:="MyFirstProgram.xls"
```

This final line of code uses the ActiveWorkbook object to save the workbook to disk under the specified filename.

With an understanding of how the code works, you can now enter it into the VBA Code Editor. The full program is shown in the following listing. Remember that the Code Editor already entered the first and last lines — you need to enter only the remainder of the code. Try to be accurate because even a minor spelling error will prevent the program from running.

```

Public Sub MyFirstProgram()

Range("B2").Select
ActiveCell.Value = InputBox("Enter value")
Range("B3").Select
ActiveCell.Value = InputBox("Enter value")
Range("B4").Select
ActiveCell.Value = InputBox("Enter value")
Range("B5").Select
ActiveCell.Value = InputBox("Enter value")
Range("B6").Select
ActiveCell.Value = InputBox("Enter value")
Range("B7").Select
ActiveCell.Formula = "=Sum(B2..B6)"
ActiveWorkbook.SaveAs Filename:="MyFirstProgram.xls"

End Sub

```

Running the Program

After the code has been entered, you can run the program and see how it works. Here are the steps required:

1. Use the Windows taskbar to activate Excel.
2. Press Alt+F8 to open the Macros dialog box, shown in Figure 1-4. (You can also display this dialog box by selecting Tools ⇨ Macro ⇨ Macros from the Excel menu.)

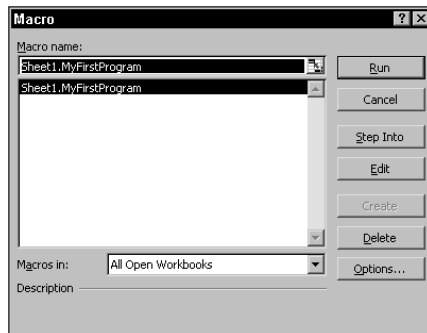


Figure 1-4 Selecting the program to run

3. In the Macros dialog box, select Sheet1.MyFirstProgram (this name indicates that the program is stored in Sheet1); then click the Run button.
4. The program prompts you to enter a value, repeating this action five times.

When the program finishes, your worksheet should look like Figure 1-5 (although you entered different numbers, of course).

Even though this is a simple program, it illustrates some of the advantages of Excel programming as related to accuracy and consistency. Specifically, it ensures that:

- Data is entered into the proper worksheet cells.
- The correct formula is used.
- The workbook is saved under the correct name.

With the fundamentals under your belt, you are ready to move on to the next session and start learning the details of Excel programming.

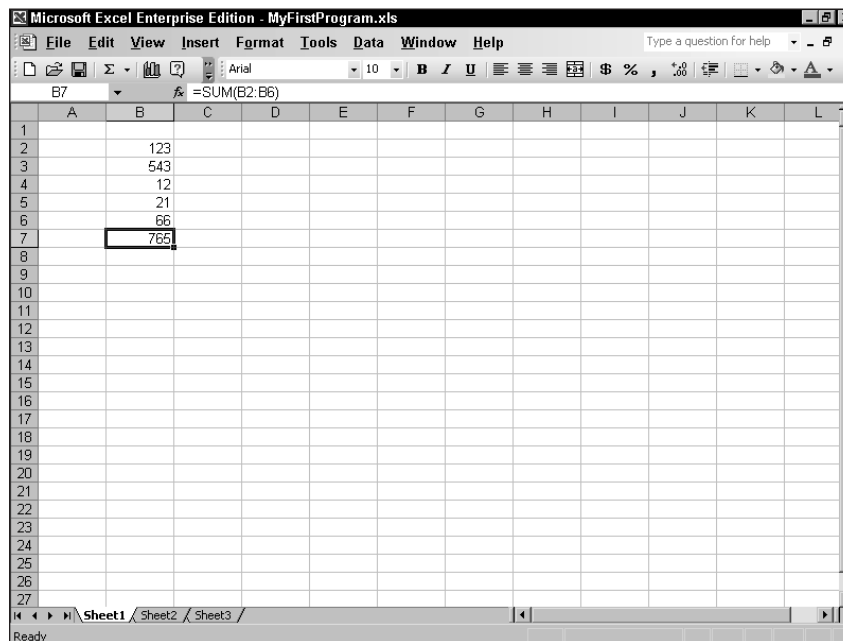


Figure 1-5 The worksheet after the program runs

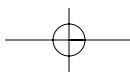


Done!

REVIEW

In this session you learned some of the fundamentals and background of Excel programming, including:

- There are several advantages of programming Excel.
- You use the VBA language to write Excel programs.
- Excel makes its functionality available to VBA as components.
- The components that Excel exposes are collectively called the Excel object model.
- Excel macros are a simple form of programming.
- Planning ahead is important when designing a program.



QUIZ YOURSELF

1. Name three advantages of programming with Excel. (See the “Advantages of Programming” section.)
2. What are the two main parts of a program? (See the “Programming Fundamentals” section.)
3. What’s the distinction between a class and an object? (See the “The Excel Object Model” section.)
4. How does a program access Excel’s functionality? (See the “The Excel Object Model” section.)
5. What is a macro? (See the “Macros and Programming” section.)
6. What should a programmer do before starting to write code? (See the “Designing Your Custom Application” section.)

