Greg Perry
Dean Miller

THIRD EDITION
Learn computer programming in just 24 one-hour lessons

Sams Teach Yourself

Beginning Programming in 24 Hours

SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Greg Perry
Dean Miller

Sams Teach Yourself Beginning Programming in 24 Hours

Third Edition

SAMS 800 East 96th Street, Indianapolis, Indiana, 46240 USA

Sams Teach Yourself Beginning Programming in 24 Hours, Third Edition
Copyright © 2014 by Pearson Education

ISBN-10: 0-672-33700-2

ISBN-13: 978-0-672-33700-0

Library of Congress Catalog Card Number: 2013951919

Printed in the United States of America

First Printing: December 2013

Acquisitions Editor
Mark Taber

Managing Editor
Kristy Hart

Project Editor
Elaine Wiley

Indexer
Brad Herriman

Proofreader
Anne Goebel

Technical Editor
Siddhartha Singh

Publishing Coordinator
Vanessa Evans

Cover Designer
Mark Shirar

Compositor
Nonie Ratcliff

Contents at a Glance

Table of Contents

About the Authors

Greg Perry is a speaker and writer in both the programming and applications sides of computing. He is known for bringing programming topics down to the beginner's level. Perry has been a programmer and trainer for two decades. He received his first degree in computer science and then earned a Master's degree in corporate finance. Besides writing, he consults and lectures across the country, including at the acclaimed Software Development programming conferences. Perry is the author of more than 75 other computer books. In his spare time, he gives lectures on traveling in Italy, his second-favorite place to be.

Dean Miller is a writer and editor with more than 20 years of experience in both the publishing and licensed consumer products businesses. Over the years, he has created or helped shape a number of bestselling books and series, including *Sams Teach Yourself in 21 Days*, *Sams Teach Yourself in 24 Hours*, and the *Unleashed* series, all from Sams Publishing. He has written books on C programming and professional wrestling, and is still looking for a way to combine the two into one strange amalgam.

Dedication

To my wife and best friend, Fran Hatton, who's always supported my dreams and was an incredible rock during the most challenging year of my professional career.
—Dean

Acknowledgments

Greg: My thanks go to all my friends at Pearson. Most writers would refer to them as editors; to me, they are friends. I want all my readers to understand this: The people at Pearson care about you most of all. The things they do result from their concern for your knowledge and enjoyment. On a more personal note, my beautiful bride, Jayne; my mother Bettye Perry; and my friends, who wonder how I find the time to write, all deserve credit for supporting my need to write.

Dean: Thanks to Mark Taber for considering me for this project. I started my professional life in computer book publishing, and it is so gratifying to return after a 10-year hiatus. I'd like to thank Greg Perry for creating outstanding first and second editions upon which this version of the book is based. It was an honor working with him as his editor for the first two editions and a greater honor to co-author this edition. I can only hope I did it justice. I appreciate the amazing work the editorial team of Elaine Wiley, Anne Goebel, and the production team at Pearson put into this book. On a personal level, I have to thank my three children, John, Alice, and Maggie and my wife Fran for their unending patience and support.

Introduction

Learning how to program computers is easier than you might think. If you approach computers with hesitation, if you cannot even spell *PC*, if you have tried your best to avoid the subject altogether but can do so no longer, the book you now hold contains support that you can depend on in troubled computing times.

This 24-hour tutorial does more than explain programming. This tutorial does more than describe the difference between JavaScript, C++, and Java. This tutorial does more than teach you what programming is all about. This tutorial is a *training tool* that you can use to develop proper programming skills. The aim of this text is to introduce you to programming using professionally recognized principles, while keeping things simple at the same time. It is not this text's singular goal to teach you a programming language (although you will be writing programs before you finish it). This text's goal is to give you the foundation to become the best programmer you can be.

These 24 one-hour lessons delve into proper program design principles. You'll not only learn how to program, but also how to *prepare* for programming. This tutorial also teaches you how companies program and explains what you have to do to become a needed resource in a programming position. You'll learn about various programming job titles and what to expect if you want to write programs for others. You'll explore many issues related to online computing and learn how to address the needs of the online programming community.

Who Should Use This Book?

The title of this book says it all. If you have never programmed a computer, if you don't even like them at all, or if updating the operating system of your phone throws you into fits, take three sighs of relief! This text was written for *you* so that, within 24 hours, you will understand the nature of computer programs and you will have written programs.

This book is aimed at three different groups of people:

- Individuals who know nothing about programming but who want to know what programming is all about.

- Companies that want to train nonprogramming computer users for programming careers.
- Schools—both for introductory language classes and for systems analysis and design classes—that want to promote good coding design and style and that want to offer an overview of the life of a programmer.

Readers who seem tired of the plethora of quick-fix computer titles cluttering today's shelves will find a welcome reprieve here. The book you now hold talks to newcomers about programming without talking down to them.

What This Book Will Do for You

In the next 24 hours, you will learn something about almost every aspect of programming. The following topics are discussed in depth throughout this 24-hour tutorial:

- The hardware and software related to programming
- The history of programming
- Programming languages
- The business of programming
- Programming jobs
- Program design
- Internet programming
- The future of programming

Can This Book Really Teach Programming in 24 Hours?

In a word, yes. You can master each chapter in one hour or less. (By the way, chapters are referred to as "hours" or "lessons" in the rest of this book.) The material is balanced with mountains of shortcuts and methods that will make your hours productive and hone your programming skills more and more with each hour. Although some chapters are longer than others, many of the shorter chapters cover more detailed or more difficult issues than the shorter ones. A true attempt was made to make each hour learnable in an hour. Exercises at the end of each hour will provide feedback about the skills you learned.

Conventions Used in This Book

This book uses several common conventions to help teach programming topics. Here is a summary of those typographical conventions:

- Commands and computer output appear in a special `monospaced` computer font. Sometimes a line of code will be too long to fit on one line in this book. The code continuation symbol (➥) indicates that the line continues.
- Words you type also appear in the `monospaced` computer font.
- If a task requires you to select from a menu, the book separates menu commands with a comma. Therefore, this book uses File, Save As to select the Save As option from the File menu.

In addition to typographical conventions, the following special elements are included to set off different types of information to make it easily recognizable.

TRY IT YOURSELF ▼

The best way to learn how to program is to jump right in and start programming. These Try it Yourself sections will teach you a simple concept or method to accomplish a goal programmatically. The listing will be easy to follow and then the programs' output will be displayed along with coverage of key points in the program. To really get some practice, try altering bits of the code in each of these sections in order to see what your tweaks accomplish.

NOTE

Special notes augment the material you read in each hour. These notes clarify concepts and procedures.

TIP

You'll find numerous tips that offer shortcuts and solutions to common problems.

CAUTION

The cautions warn you about pitfalls. Reading them will save you time and trouble.

HOUR 1
Hands-On Programming

You have this book because you want to go farther than the typical computer user. You don't just want to use computers; you want to write programs for them. You want the skills necessary to make computers do exactly what you want.

This 24-hour tutorial shows that you don't have to be a wizard to become a proficient (or even an expert) programmer. By the time you finish this one-hour lesson, you will have entered your first computer program, run it, and looked at the results. This chapter springboards you into programming so you can get a feel for programming right away. Once you've had this hands-on experience, subsequent hours will explain more of the background you need to understand how to design and write programs.

The highlights of this hour include:

- Learning what a program does
- Understanding the truth behind common programming myths
- Mastering source code concepts
- Entering and running your first program

Get Ready to Program

You'll find this hour divided into these two areas:

- An introduction to the nature of programming
- A hands-on practice with your first programming language

Most new programmers want to start programming right away. Yet, some background is necessary as well. This hour attempts to take care of both demands by offering an introductory background look at programming and then quickly jumping head-first into hands-on programming. If you went straight into a new programming language with absolutely no background first, you might miss the entire point of programming in the first place. So, the next few pages bring all readers up to a level playing field so that everyone who takes this 24-hour course can begin to write programs right away.

What a Computer Program Does

Most people today have some understanding of a computer's purpose. For example, a computer can help people balance their books or track their inventory. If you want to begin programming, chances are you have probably been using a computer for some time. Nevertheless, as a future programmer, you should review some fundamental computing concepts before mastering the ins and outs of a new computer language.

At its simplest level, a computer *processes data*. Figure 1.1 shows this as the primary goal of all computer programs. Many businesses call their computer programming departments *data processing departments* because computers process data into meaningful information. You may not have considered the difference between the words *data* and *information* before, but there is a tremendous difference to a computer professional. Raw data consists of facts and figures, such as hundreds of days of stock prices. A program might process that data into meaningful information, such as a line chart that shows the overall trend of the stock prices over time. It is the computer *program* that tells the computer what to do. Sometimes, a program might simply process data from another program without showing any output for the user. The processed data is still information because the program's output, stored on the disk, is changed in some way. For example, a program that closes monthly account balances may collect data from various accounting systems in a company and combine, balance, and close that data, resetting the data for the following month.

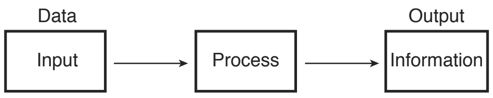


FIGURE 1.1
Programs convert raw data into meaningful information.

A *program* is a list of detailed instructions that the computer carries out. The program is the driving force behind any job that a computer does. The computer cannot do anything without a program. It is the job of the programmer to design and write programs that direct the computer to take raw data and transform that data into meaningful information for the end-user. The *end-user* (usually just called the *user*) of the computer is generally the nontechnical, nonprogramming person who needs the results (the information) that the program provides.

You, as the programmer, are responsible for guiding the computer with the programs you write. Learning to program computers takes a while, but it is certainly rewarding. Computer programming offers the advantage of instant feedback, unlike a lot of other jobs you can train for.

Common Programming Misconceptions

This text aims directly at the heart of the matter: Computers are easy to use and easy to program. A computer is nothing more than a dumb machine that “knows” absolutely nothing. You must supply the program that tells the computer what to do. A computer is like a robot that waits on your every command and acts out your instructions exactly as you give them. Sometimes your program instructions are incorrect. If they are, the computer goes right ahead and attempts them anyway.

CAUTION

Don’t fear computer programming. Computers are tools to help you get your job done. You can learn to program.

Many misconceptions about computers exist and stem from a lack of understanding about how computers work and what computers are physically capable of doing. This book wants to shoot down the myths and improve your understanding of these machines. You’ll be programming computers in no time. The computer is nothing more than a tool that helps you do certain types of work. The computer itself is not bad or good. A hammer is a tool you can use for good (to build houses) or for bad (to break things). A computer in the wrong hands can be used for bad purposes, but that isn’t the computer’s fault any more than it is the hammer’s fault if someone misuses it.

The next three sections attack the three most popular computer myths. Have you heard any of them? Did you think some were true?

Myth 1: Only Math Experts Can Program Computers

Thank goodness this is a myth and not reality—thousands of people would be out of work (including most computer book authors!). Computers would be elitist machines used by only the best engineers and scientists; the casual user could not master them. Computers would still be beneficial in some areas but they would not provide the benefits that so many people can enjoy.

Not only can you be poor at math—you don’t even have to like math or have the desire to learn math to be a good computer programmer. The computer does all the math for you; that’s one of its jobs. There are countless expert computer programmers in the world who cannot tell you the area of a circle or the square root of 64. Relax if you thought this myth was reality.

Programming can provide beneficial side effects. It turns out that, as you become a better programmer, you may find your math skills improving. Developing programming skills tends to improve your overall capability for logical thinking, which underlies many skills in math as well. Therefore, being better in math might be a result of programming but it’s not a prerequisite.

TIP

People who favor logic puzzles, crosswords, anagrams, and word-search games seem to adapt well to programming, but again, liking these gaming activities is not a programming prerequisite. You will find that you can learn to program computers, and actually become extremely good at it, without liking math, being good at math, or having any flair at all for puzzles or word games.

Myth 2: Computers Make Mistakes

You might have heard the adage, "To err is human, but to *really* foul things up takes a computer!" This might be accurate, but only in that a computer is so very fast that it duplicates a person's mistakes rapidly.

Computers do not make mistakes—people make mistakes. If you have heard a bank teller tell you that $24 was incorrectly deleted from your savings account because "the computer program made an error," the teller probably has no idea what really happened. People program computers, people run them, and people enter the data that the computer processes.

The odds of a computer randomly fouling up a customer's bank balance are minute. Computers simply do not make random mistakes unless they are programmed incorrectly. Computers are finite machines; when given the same input, they always produce the same output. That is, computers always do the same things under the same conditions. Your job, as you learn to program, will be to reduce the chance of computer mistakes.

When a computer malfunctions, it does not make a simple mistake; rather, it *really* messes things up. When a computer fails, it typically breaks down completely, or a storage device breaks down, or the power goes out. Whatever happens, computers go all out when they have a problem and it is usually very obvious when they have a problem. The good news is that computers rarely have problems.

Before people invented computers, banks kept all their records on ledger cards. When a teller found a mistake (possibly one that the teller had made), do you think the teller said, "The ledger card made a mistake"? Absolutely not. Computers can have mechanical problems, but the likelihood of small mistakes, such as an incorrect balance once in a while, is just too small to consider. Such mistakes are made by the people entering the data or by (gulp) the programmers.

Myth 3: Computers Are Difficult to Program

Computers are getting easier to use, and to program, every day. If you used a microwave, drove a car, or used an iPod recently, then chances are good that you used a computer when you did. Yet, did you know you were using a computer? Probably not. The makers of computers have found ways to integrate computers into your everyday life to monitor and correct problems that might otherwise occur without them.

Of course, if you are reading this book, you want to learn enough about computers to write your own programs. Writing computer programs does take more work than using a microwave oven's computerized timer functions. The work, however, primarily involves getting down to the computer's level and learning what it expects.

Not only are computers getting easier to program every day, but you have more opportunities to learn about them than ever before. Cable television channels are loaded with educational shows about using and programming computers. Books and videos on the subject are all around you. The Internet itself contains scores of classes on all aspects of computers and other topics. There is probably a computer programming class now in session somewhere within 15 minutes of your house as you read this.

Many Programs Already Exist

Although there are many programs already written for you to use, sometimes you need a program that fills a specific need and you cannot find one that does exactly what you want. When you are done with this book, you will know exactly what you need to design and write your own programs.

Programmers Are in Demand

Look at help-wanted websites. You'll find that there is a shortage of computer programmers. Amidst the requests for Java programmers, C++ programmers, Ruby on Rails programmers, mobile app developers, systems analysts, senior systems analysts, object-oriented programmers, systems programmers, HTML coders, and application programmers, you may find yourself lost in a sea of uncertainty and *TLAs* (three-letter acronyms) that might, at first, seem hopeless. Do not fret; this book will help direct you toward areas of programming that might be right for you.

Hour 22's lesson, "How Companies Program," explores the chain of computer jobs and describes what each type of programming job is all about. If you are just starting out, you probably won't be able to go to work as the most senior-level programmer, but you will be surprised at the salary your programming skills can bring you.

The Real Value of Programs

Although individual computer programs are going down in price, companies and individual computer owners invest more and more in programs every year. Not only do people purchase new programs as they come out, but they update the older versions of programs they already have.

Businesses and individuals must factor in the cost of programs when making computer decisions. Whereas an individual usually buys a computer—called *hardware* because the machine isn't changed often or easily—and is done with the hardware purchasing for a while, the purchasing of programs—the *software*—never seems to end, because software changes rapidly. As a future programmer, this is welcome news because you have a secure career. For the uninformed computer purchaser, the cost of software can be staggering.

A business must also factor in the on-staff and contract programmers and the time needed to write the programs it uses. More information on the programming and support staff appears in the next section.

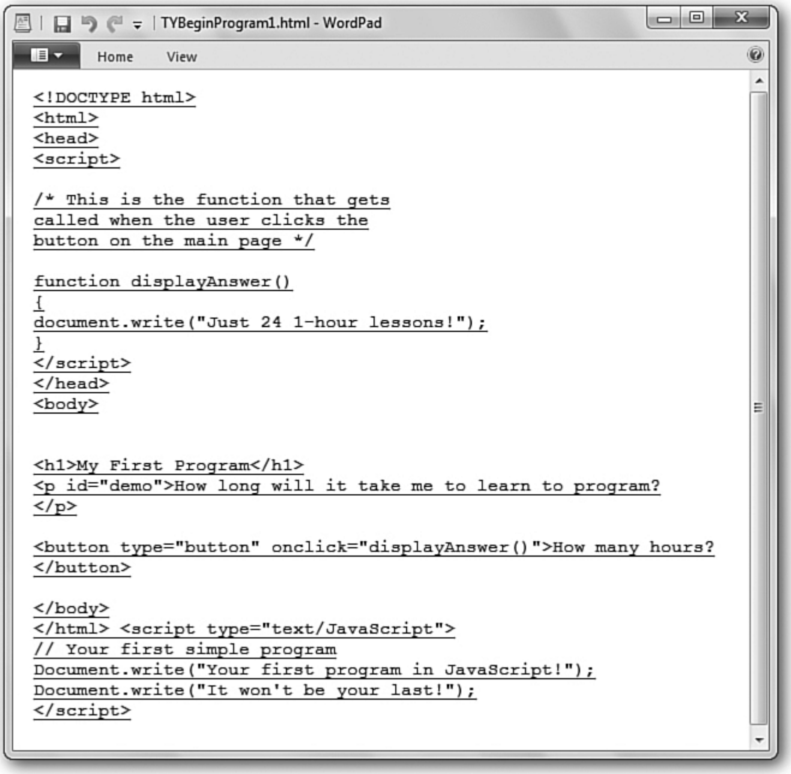
Users Generally Don't Own Programs

When a company purchases software, it most often purchases a *software license*. If a company wants to buy a word-processing program for 100 employees, legally it must purchase 100 copies of the program, or at least buy a *site license* that allows the company to use the software on more than one machine. When a company buys a program, it does not own the program. When you buy a record, you do not own the music; you have only purchased the rights to listen to the music. You cannot legally alter the music, record it, give away recordings of it, and most importantly, you cannot sell recordings that you make of it. The same is true for software that you buy. The license for individual software grants you permission to use the software on one computer at any one time.

Giving Computers Programs

Figure 1.2 shows the code for a simple program that creates a web page with an interactive button, all created by a computer programmer. As a matter of fact, that computer programmer is about to be you after you create the code using JavaScript. JavaScript is a programming language that enables you to improve websites by letting you customize the look and feel of your site's presentation, increasing interaction with your website users, and validating any data they enter. You may think JavaScript is specifically for web masters and site designers, but it's a simple language and learning its basics can help you learn basic programming concepts.

Another advantage to JavaScript is that most programming languages need you to install an interpreter or compiler on your computer in order to execute your code. With JavaScript (and HTML), you only need a text editor. You type in your code, save it with an .html extension, and then open the saved file with your web browser.



```
<!DOCTYPE html>
<html>
<head>
<script>

/* This is the function that gets
called when the user clicks the
button on the main page */

function displayAnswer()
{
document.write("Just 24 1-hour lessons!");
}
</script>
</head>
<body>


<h1>My First Program</h1>
<p id="demo">How long will it take me to learn to program?
</p>

<button type="button" onclick="displayAnswer()">How many hours?
</button>

</body>
</html> <script type="text/JavaScript">
// Your first simple program
Document.write("Your first program in JavaScript!");
Document.write("It won't be your last!");
</script>
```

FIGURE 1.2
A program's instructions are somewhat cryptic, but readable by people.

A computer is only a machine. To give a machine instructions, your instructions might be fairly readable, as Figure 1.2's are, but the *code* (another name for a program's instructions) must be fairly rigid and conform to a predefined set of rules and regulations according to the programming language you use. Therefore, to write a program in the JavaScript programming language, you must conform to JavaScript's rules of proper command spelling and placement. This programming language grammar is called *syntax*. (And you thought syntax was just a levy placed on cigarettes and liquor!)

Despite its name, JavaScript is not tied to the Java programming language (which is covered in Part III, "Object-Oriented Programming with Java," of this book). In fact, the name has caused some confusion as computer fans thought that JavaScript was either an extension or spin-off of Java, but it got the name because Java was a hot new language at the time, and some thought the connection of the scripting language to the web development language would benefit the adoption and growth of JavaScript. Misleading name aside, the more you learn JavaScript, the more easily you will be able to learn additional programming languages. Although computer languages have different goals, syntaxes, and reasons for being, they are often similar in structure.

NOTE

It is important to understand that the first section of this book is teaching you the basics of computer programming using the JavaScript language, and is not a complete JavaScript tutorial. This may seem like the same thing, but it is not. The aim of the book is to teach you basic computer programming techniques (with examples in JavaScript code) in the first half of the book and introduce you to a variety of programming languages and jobs in the second half. There are scores of excellent books, including *Teach Yourself JavaScript in 24 Hours*, also by Sams Publishing, that can take you through every in and out of the JavaScript language. That book will take you far more in-depth into the language than this book. Now that we've properly calibrated your expectations, let's get back to beginning programming!

Source Code

Even after you make the effort to learn a computer language such as JavaScript, and after you go to the trouble of typing a well-formed and syntactically accurate program such as the one in Figure 1.2, your computer still will not be able to understand the program! The program you write is called *source code*. It is the source code that you write, manipulate, and correct. Your computer, on the other hand, can understand only *machine language*, a compact series of computer-readable instructions that make no sense to people. They make sense to some advanced computer gurus, but my general assertion stands that they don't make sense to people.

Listing 1.1 shows machine language. Can you decipher any of it? Your computer can. Your computer loves machine language. Actually, it's the only language your computer understands. And different computers understand their own version of a machine language so what works on one type of computer will not necessarily work on another. It's best to stay away from machine language and let products such as JavaScript convert your higher-level language into machine language's cryptic 1's and 0's. To convert source code such as your JavaScript program to machine language, you need an interpreter (or for other languages, a *compiler)*.

LISTING 1.1 Machine language is extremely difficult for people to decipher.

```
01100100
10111101
10010011
10010100
00001111
01010101
11111110
```

All programs you run on your computer, phone, or tablet, including Microsoft Word, Internet Explorer, and programming languages, are already converted into machine language. That's why you can click a program's icon and the program begins immediately. No interpretation or compilation is required. By providing you with the machine language only, software vendors serve two purposes:

1. They give you programs that execute quickly without the intervening compiling step.
2. They ensure that you don't change the source code, thereby protecting their intellectual property.

Your First Program

The first program that you write will be simple. You may not understand much of it, and that's fine. The goal here is not to explain the program details, but to walk you through the entire process of the steps that are typical of computer programming:

1. Type a program's source code and save it.
2. Load the code document, which is now an HTML document saved to your hard drive, with your web browser and see what happens.
3. If the page is not doing what you want and has errors, called *bugs*, you'll need to fix those bugs in your source code and repeat these steps. Hour 7, "Debugging Tools," explains how to locate and fix common bugs. You may need to do some bug hunting earlier than Hour 7 if you experience problems with your programs in these first few hours. If a program does not work as described, you will need to compare your source code very carefully to the code in the book and fix any discrepancies before saving it again.

Starting with JavaScript

You can write your JavaScript code in either your text editor or a word processor. If you choose the latter, you must make sure to save your files as text only. Save your programs with an .html extension, and then when you double click on the file, it should automatically open with your default web browser.

NOTE

Unlike word processors, your editor will not wrap the end of a line down to the subsequent line. It's important that programming instructions (called *statements*) remain separate and stay on their own lines in many cases. Even if a programming language allows multiple program statements per line, you'll find that your programs are more readable and easier to debug if you keep your statements one per line. Sometimes, a single statement can get lengthy and you may span long statements over two or three lines, but rarely would you want multiple statements to appear on the same line.

Clarifying Comments

The program presented in Figure 1.2 not only has program statements, but also some lines that do not do anything when the program is run by the interpreter, but have great value to anyone reading your code. These lines start with either a slash and asterisk (/*) or two consecutive slashes (//) and are known as *comments*.

You may be asking if there's any difference between the slash and asterisk comment and the double-slash comments. Taking the second type first, the double-slash comment is known as a single-line comment. JavaScript will ignore all text to the right of the two slashes. As soon as you hit return, JavaScript will start paying attention to your code again. So if you want to keep commenting, you will need to start the next line with a fresh pair of slashes.

```
// The next five lines ask web visitors for their favorite
// color and Steinbeck novel
```

In the previous example, if you left off the two slashes, JavaScript would assume you were either typing color or calling a function named color and you might not get the results you were expecting.

When JavaScript encounters a comment that begins with the slash and asterisk, it will treat everything that follows as a comment until it sees a closing asterisk and slash (*/). This can be on the same line:

```
/* Next, the code calculates the area of the circle. */
```

Or it can go on for several lines:

```
/* This section is used to enter a contact
The user can enter a name, physical address,
up to 5 email addresses, home, work, and cell
phone numbers, and any additional comments
to help remember how they know the person. */
```

All languages support comments, which are also called *remarks*, although some languages use an apostrophe instead of the slashes. Although you'll find many comments in programs, and you should put many comments in the programs you write, comments are not for the interpreter or for the computer. Comments are for people. In JavaScript, for example, the interpreter ignores all text to the right of the slashes or slash/asterisk. The computer knows that comments are for you and not for it.

Reasons for Comments

Comments help produce more understandable program listings. The programming industry considers the inclusion of comments to be one of the most important good programming habits you can develop. Here are common reasons programmers put comments in their programs:

- Comments can identify programmers—If you develop programs with others or write programs for a company, you should put your name and contact information in comments at the top of your programs. Later, if someone has a question about the program or wants you to change the program, they'll know that you wrote the program and they can locate you.
- Before a section of tricky code, you can use comments to include your explanation of the code's purpose. Later, if you or someone else needs to change the program, the change will be easier; reading the comments is much simpler than trying to figure out the goals of the tricky code from uncommented code.

Placement of Comments

When you use the double slashes (//), a comment can appear on the same line as code. The following code shows comments to the right of lines of other program instructions. Notice that double slashes precede the text. These comments explain what each line of code does.

```
if hours > 12          // Test the hour for AM/PM indicator
   hours = hours - 12  // Convert 24-hour time to 12-hour
   amOrPm$ = " PM"     // Set indicator that will print
```

TIP

Most programming languages are ***free-form***, meaning that you can indent lines and add as many spaces and blank lines as you like. Doing so makes a program more readable. In Hour 7, you'll learn how to use this ***whitespace*** (extra spaces and lines in code) to improve your programs' readability by separating one program section from another and, thereby, making it easier for programmers to understand your code.

Sure, as a programmer, you will often understand code without the need to put a remark to the right of each line. But at the time you write a program, the program is clearest to you. When you later go back to edit a program, what may have been clear before is no longer clear because time has passed. A few comments will go a long way toward reminding you of your original intent.

However, do not overdo comments. Some comments are redundant, as the following shows:

```
document.write("Martha");   // Prints the word Martha
```

Such a comment is not helpful and serves only to cloud an already-understandable instruction. Don't overuse comments, but use them generously when needed to clarify your code.

Entering Your Own Program

You're now ready to type and run your own program from scratch. The program will be simple. The goal here is to get you used to using the editor.

▼ TRY IT YOURSELF

Follow these steps to create your own program:

1. In your text editor, select File and then New File to create a brand-new document.
2. Type the program in Listing 1.2 into the editor. Be careful that you type the program exactly as you see it in the listing. If you make a typing mistake, you will not be able to run the program. The program is a little bit complicated, but if you type it as is, you should have a working program. Notice that both styles of comments are used in the program.

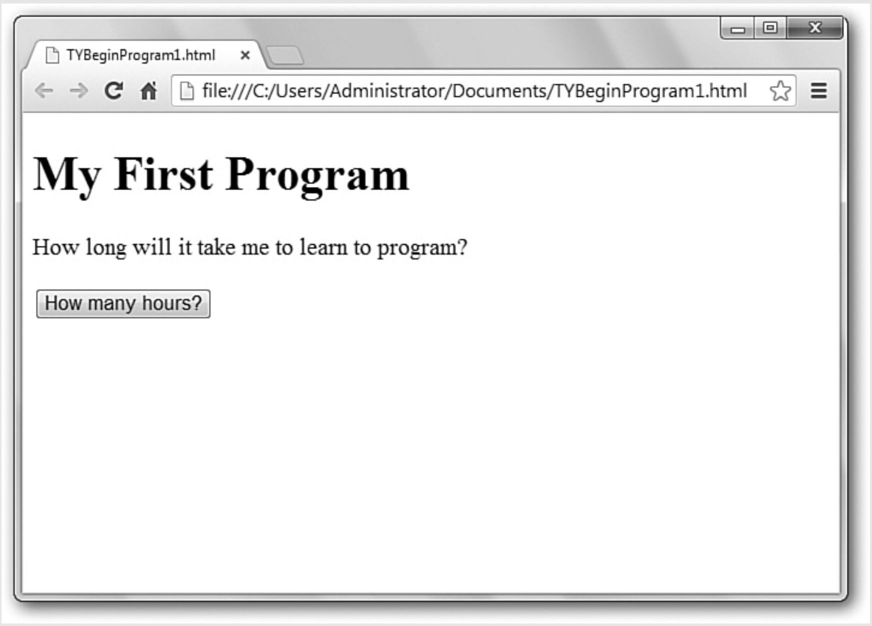
LISTING 1.2 Type your first program into the editor.

```
<!DOCTYPE html>
<html>
<head>
<script>

/* This is the function that gets
called when the user clicks the
button on the main page */

function displayAnswer()
{
document.write("Just 24 1-hour lessons!");
}
</script>
</head>
<body>


<h1>My First Program</h1>
<p id="demo">How long will it take for me to learn to program?.</p>

<button type="button" onclick="displayAnswer()">How many hours?</button>

</body>
</html>
```

3. Save the program as text only, and give it a name with an .html ending. I named mine TYBeginProgram1.html. After you save the program, either double click the file's icon in your explorer or open the file in your web browser. You should see the web page shown in Figure 1.3 to start and in Figure 1.4 when you click the button.

NOTE

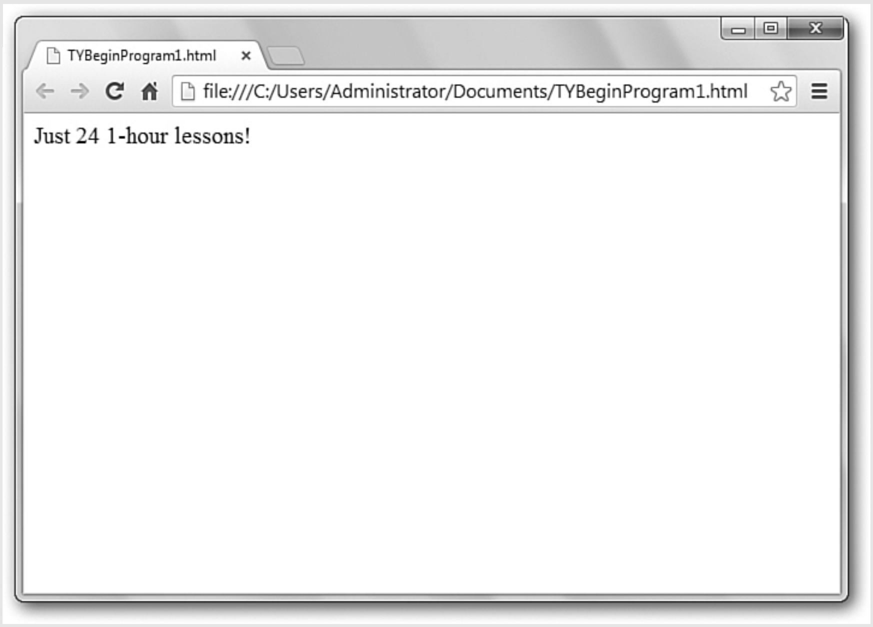
The page you load into your web browser should have an .html extension (.htm works as well). Eventually, you will want to separate your JavaScript code into its own files, using .js files. For now, you can embed all your code into .html files.


TYBeginProgram1.html
file:///C:/Users/Administrator/Documents/TYBeginProgram1.html
How many hours?

My First Program

How long will it take me to learn to program?

FIGURE 1.3
With less than two dozen lines of code, you've created a simple web page, complete with a button for the visitor to click.


TYBeginProgram1.html
file:///C:/Users/Administrator/Documents/TYBeginProgram1.html

Just 24 1-hour lessons!

FIGURE 1.4
Once you click the button on your web page, you get a new page with more information. Awesome interactivity, right?

If your program contained an error, the page would not show up like the ones in Figures 1.3 and 1.4. You would have to work through your version line-by-line to see where the error might be.

TIP

In addition to checking for errors, it is also a smart idea to check your created pages in different browsers when possible to see the subtle difference in presentation from browser to browser.

Summary

Now that you know a bit more about the computer industry as a whole, you have a better idea of where programmers fit into the picture. Computers can do nothing without programs and programmers are responsible for providing those programs.

You already got a taste of programming this hour by typing your first JavaScript program and opening the resulting document in your browser. As you saw by typing your own program into an editor, entering and running a code is not painful at all. As a matter of fact, the quick feedback that programming offers is a rewarding experience. Programmers often like their profession very much because they can see the results of their work quickly.

Q&A

Q. Once I write a program, is my job finished?

A. It is said that a program is written once and modified many times. Throughout this 24-hour tutorial, you will learn the importance of maintaining a program after it is written. As businesses change and computing environments improve, the programs that people use must improve as well. Companies provide upgrades to software as new features are needed. The programs that you write today will have to change in the future to keep up with the competition as well as with new features that your users require.

Q. Can I enter other programming language listings in my editor as well?

A. You can write programs in other programming language, such as C, C++, and Java, in your text editor. However, you would need a compiler or interpreter (depending on the language) to convert the programming language's code to machine language. A machine language program is your program's instructions compiled to a language your computer can understand.

Workshop

The quiz questions are provided for your further understanding.

Quiz

1. What is the difference between data and information?
2. What is a program?
3. What is a programming language?
4. True or false: Computers never make mistakes.
5. Why should people not fear the computer replacing their jobs?
6. What do programmers use editors for?
7. True or false: Java and JavaScript are related to each other.
8. What filename extension should you use when you create a JavaScript program?
9. True or False: There's no problem writing your programs in your typical word-processing program.
10. Tweak the program in Listing 1.2 so the dialog box asks, "Who wrote this program?" and the answer the user gets upon clicking the button is your name.

Answers

1. Data consists of raw facts and figures, and information is processed data that has more meaning.
2. A program is a set of detailed instructions that tells the computer what to do.

3. A programming language is a set of commands and grammar rules with which you write programs that give computers instructions.
4. False. A computer might make a mistake, but it's rare that it does so. In the vast majority of cases where a computer is blamed, a person entered bad data or typed a bad program.
5. Computers increase jobs, not replace them. The information industry is one of the reasons for the past two decades of economic growth.
6. Programmers use editors to type programs into the computer.
7. False. JavaScript and Java only share a name; JavaScript was a scripting language that was developed at the same time Java was hot, so it was named in a way to take advantage of Java's popularity.
8. If your code is going to stay in your HTML document, you can use .html or .htm, but when you create separate JavaScript files, use .js.
9. False. The default method of saving files in word processors will add formatting codes that will generate errors when you try to run your program. You can still use a word processor, but you must always remember to save your files as plain text.
10. Here is one possible solution:

```
<!DOCTYPE html>
<html>
<head>
<script>

/* This is the function that gets
called when the user clicks the
button on the main page */

function displayProgrammer()
{
document.write("Dean Miller did!");
}
</script>
</head>
<body>


<h1>My First Program</h1>
<p id="demo">Who wrote this program?.</p>

<button type="button" onclick="displayProgrammer()">And the answer is...</
button>

</body>
</html>
```

Index