JavaScript for Kids

- Build an Animated Robot App
- Create Cool Games
- Make a Web Page
- Learn Real Coding Skills

Chris Minnick
Eva Holland
Cofounders of WatzThis?

For Dummies
# Table of Contents

## Introduction

- About This Book .............................................................. 1
- Foolish Assumptions .......................................................... 3
- Icons Used In This Book .................................................... 3
- Beyond the Book ............................................................... 4
- Where to Go from Here ...................................................... 4


### Chapter 1: Programming the Web ............................................. 7
- What Is Programming? ...................................................... 8
- Talking to Computers ...................................................... 9
- Choosing a Language ..................................................... 11
- What Is JavaScript? .......................................................... 12
- Get Your Browser Ready .................................................. 15
- Opening the Web Developer Tools .................................... 16
- Introducing the JavaScript Console ................................. 18
- Running Your First JavaScript Commands ...................... 19
- Having Fun with Math .................................................... 21

### Chapter 2: Understanding Syntax ........................................ 22
- Saying Precisely What You Mean ....................................... 23
- Making a Statement ...................................................... 24
- Following the Rules .......................................................... 25

### Chapter 3: Giving and Receiving Data ................................. 33
- Mastering Variables ...................................................... 34
- Understanding Data Types .............................................. 38
- Prompting the User for Input .......................................... 42
- Responding to Input ..................................................... 44
- Combining Input and Output ........................................... 48

### Chapter 4: Fiddling with Web Applications ......................... 51
- Introducing JSFiddle ...................................................... 52
- Creating a JSFiddle Account ............................................ 63
- Sharing Your Fiddle ..................................................... 65
- Saving Your App ........................................................ 67
Chapter 12: Making It Functional ................... 190
  Understanding Functions ................................................................. 191
  Knowing What Functions Are Made Of ........................................... 193
  Building Function Junction .............................................................. 196

Chapter 13: Creating a Wish List Program ........... 206
  Introducing the Wish List Program ................................................... 207
  Forking the Code ................................................................................. 208
  Writing the HTML ................................................................................ 210
  Writing the JavaScript Code ............................................................... 212

Part V: Freedom of Choice ..................................231

Chapter 14: Making Decisions with the
If...Else Statement ................................................................. 233
  Boolean Logic ....................................................................................... 234
  Introducing if...else Statements ......................................................... 236
  Combining Comparisons with Logical Operators ......................... 238
  Freshening Up the JavaScript Pizzeria .............................................. 240

Chapter 15: Doing Different Things with Switch ...... 252
  Writing a Switch .................................................................................... 253
  Building the Activity-of-the-Day Calendar ....................................... 255

Chapter 16: Choose Your Own Adventure ........... 267
  Planning the Story ................................................................................ 268
  Playing the Game ................................................................................ 269
  Forking the Code ............................................................................... 271
  Tiptoeing through the HTML and CSS ............................................ 272
  Writing the Martian Rescue! JavaScript ............................................ 276

Part VI: Loops ............................................................... 293

Chapter 17: What’s This Loop For? ..................... 295
  Introducing the for Loop ................................................................... 296
  Random Weather Forecasting ........................................................... 299

Chapter 18: Using While Loops ............................ 309
  Writing a while Loop ........................................................................... 310
  Coding the Lunch Game ................................................................. 312
  Moving to Your Own Website .......................................................... 318
Introduction

JavaScript For Kids For Dummies is an introduction to the basics of JavaScript coding. In each chapter, we walk you step-by-step through creating JavaScript programs for the web. Designed for kids of all ages, with no coding experience, we strive to introduce this technical topic in a fun, engaging, and interactive way.

JavaScript is the most widely used programming language in the world today. That’s why we think you’ve made a great decision by beginning your journey into the world of coding by picking up this book.

JavaScript is fun and easy to learn! With some determination and imagination, you’ll be on your way to creating your very own JavaScript programs in no time!

Just as the only way to Carnegie Hall is to practice, practice, practice, the only way to become a better programmer is to code, code, code!

About This Book

We seek to “de-code” the language of JavaScript for you and give you an understanding of the concepts. With the ability to move at your own pace, JavaScript For Kids For Dummies will get you up to speed. In this book, you learn how to create fun games and programs. We even show you how to customize and build your own versions of the games that you can post to the web and share with your friends!

Whether you know a little JavaScript or you’ve never seen it before, this book shows you how to write JavaScript the right way.
Topics covered in this book include the following:

- The basic structures of JavaScript programs
- JavaScript expressions and operators
- Structuring your programs with functions
- Writing loops
- Working with JavaScript, HTML5, and CSS3
- Making choices with `if...else` statements

Learning JavaScript isn’t only about learning how to write the language. It’s also about accessing the tools and the community that has been built around the language. JavaScript programmers have refined the tools and techniques used to write JavaScript over the language’s long and exciting history. Throughout this book, we mention important techniques and tools for testing, documenting, and writing better code!

To make this book easier to read, you’ll want to keep in mind a few tips. First, all JavaScript code and all HTML and CSS markup appears in monospaced type like this:

```javascript
document.write("Hi!");
```

The margins on a book page don’t have the same room as your monitor likely does, so long lines of HTML, CSS, and JavaScript may break across multiple lines. Remember that your computer sees such lines as single lines of HTML, CSS, or JavaScript. We indicate that everything should be on one line by breaking it at a punctuation character or space and then indenting any overage, like so:

```javascript
document.getElementById("thisIsAnElementInTheDocument").
    addEventListener("click",doSomething,false);
```

HTML and CSS don’t care very much about whether you use uppercase or lowercase letters or a combination of the two. But, JavaScript cares a lot! In order to make sure that you get the correct results from the code examples in the book, always stick to the same capitalizations that we use.
Foolish Assumptions

You don’t need to be a “programming ninja” or a “hacker” to understand programming. You don’t need to understand how the guts of your computer work. You don’t even need to know how to count in binary.

However, we do need to make a couple of assumptions about you. We assume that you can turn your computer on, that you know how to use a mouse and a keyboard, and that you have a working Internet connection and web browser. If you already know something about how to make web pages (it doesn’t take much!), you’ll have a jumpstart on the material.

The other things you need to know to write and run JavaScript code are details we cover in this book, and the one thing you’ll find to be true is that programming requires attention to details.

Icons Used In This Book

Here’s a list of the icons we use in this book to flag text and information that’s especially noteworthy.

This icon highlights technical details that you may or may not find interesting. Feel free to skip this information, but if you’re the techie type, you might enjoy reading it.

This icon highlights helpful tips that show you easy ways or shortcuts that will save you time or effort.

Whenever you see this icon, pay close attention. You won’t want to forget the information you’re about to read — or, in some cases, we’ll remind you about something that you’ve already learned that you may have forgotten.

Be careful. This icon warns you of pitfalls to avoid.
Beyond the Book

We’ve put together a lot of extra content that you won’t find in this book. Go online to find the following:

👍 Cheat Sheet: An online Cheat Sheet is available at www.dummies.com/cheatsheet/javascriptforkids. Here, you find information on converting CSS property names to JavaScript; a list of common web browser events that JavaScript can respond to; and a list of words that can’t be used as JavaScript variables, functions, methods, loop labels, or object names.

👍 Web Extras: Online articles covering additional topics are available at www.dummies.com/extras/javascriptforkids. In these articles, we cover things like HTML5 form input tricks, how to name JavaScript variables, JavaScript troubleshooting tips, and more.

Where to Go from Here

Coding with JavaScript is fun, and when you get a little knowledge under your belt, the world of interactive web applications is your oyster! So buckle up! We hope you enjoy the book and our occasional pearls of wisdom.

If you want to show us changes and improvements you make to our games, or programs you come up with on your own, you can do so on Facebook (www.facebook.com/watzthisco), Twitter (www.twitter.com/watzthisco), or via email at info@watzthis.com. We’re excited to see what you come up with!
Part I

What Is JavaScript?

Alert! JavaScript Is Awesome!
In this part . . .

- Programming the Web ........................................... 7
- Understanding Syntax ............................................. 22
- Giving and Receiving Data ....................................... 33
- Fiddling with Web Applications ................................. 51

For Dummies can help you get started with lots of subjects. Visit www.dummies.com to learn more and do more with For Dummies!
Programming the Web

JavaScript is a powerful language that’s easy to learn! In this chapter, we explain the basics of programming, tell you what JavaScript is, and get you started with writing your first JavaScript commands.

One of the most important parts of starting any new project is to make sure you have your workshop stocked with all the correct tools. In this chapter, you install and configure all the programs you need and start experimenting with some real JavaScript programs!
What Is Programming?

A computer program is a series of instructions that can be understood and followed by a computer. Computer programming, also known as coding, is what we call it when we write these instructions. Computers can’t do things on their own. They need a computer program to tell them what to do. Computer programmers write code to make computers do all sort of things.

Another name for a computer program is software.

The women who invented programming

Electronic computers as we know them were first invented in the 1930s. But it was the middle of the 1800s when the first computer program — a set of instructions designed to be carried out by a machine — was written.

The author of the first computer program — and, therefore, the world’s first computer programmer — was a woman named Ada Lovelace. A mathematician in England, she was the first person to envision computers that could do much more than just crunch numbers. She foresaw computers being able to do all the things we use computers for today: including working with words, displaying pictures, and playing music. Her unique insights earned her the nickname “The Enchantress of Numbers.”

Compilers are programs for converting programming languages into machine language. The first compiler was created by Grace Murray Hopper in 1944. This invention led to computer programs that could run on different types of computers, and eventually to JavaScript. Hopper is also credited with being the inventor of the term debugging for fixing problems in computer programs. The term was inspired by the removal of an actual moth from an early computer. Hopper became known as “The Queen of Software” or “Amazing Grace” for her contributions to modern computing.
Computer programs help people to do many thousands of things, including the following:

- Playing music and videos
- Performing scientific experiments
- Designing cars
- Inventing medicines
- Playing games
- Controlling robots
- Guiding satellites and spaceships
- Creating magazines
- Teaching people new skills

Can you think of more examples of things that computers can do?

**Talking to Computers**

At the heart of every computer is a central processing unit (CPU). This CPU is made up of millions of tiny, very fast switches (called *transistors*) that can be either on or off. The position of each of these switches at any time determines what the computer will do.

Software written by programmers tells these switches when to turn on or off and in what combination by using *binary codes*. Binary codes use zeros and ones to form letters, numbers, and symbols that can be put together in order to perform tasks.

Every single thing that a computer does is the result of a different combination of many zeros and ones. For example, to represent a lowercase letter *a*, computers use the following binary code:

0110 0001
Each zero or one in a binary number is called a *bit*, and a combination of eight bits is called a *byte*. When you hear the words *kilobyte, megabyte,* and *gigabyte* used to tell how big a file is, what it’s talking about is the number of eight-bit binary codes it takes to store the file.

Table 1-1 lists the most commonly used storage sizes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of Bytes</th>
<th>What It Can Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kilobyte (KB)</td>
<td>1,024</td>
<td>Two to three paragraphs of text</td>
</tr>
<tr>
<td>Megabyte (MB)</td>
<td>1,048,576</td>
<td>800 pages of text</td>
</tr>
<tr>
<td>Gigabyte (GB)</td>
<td>1,073,741,824</td>
<td>250 songs (as MP3s)</td>
</tr>
<tr>
<td>Terabyte (TB)</td>
<td>1,099,511,627,776</td>
<td>350,000 digital pictures</td>
</tr>
<tr>
<td>Petabyte (PB)</td>
<td>1,125,899,906,842,624</td>
<td>41,943 Blu-ray discs</td>
</tr>
</tbody>
</table>

A typical small computer program might contain anywhere from a couple kilobytes to a couple megabytes of instructions, images, and other data. Because it’s unlikely that you have enough time in your busy day to type out thousands, or even millions, of ones and zeros, if you want to tell a computer what to do, you need a translator who speaks both human languages and computer (or *machine*) language. Computer programming languages are this translator.

Every computer program is written using a computer programming language. Programming languages allow you to write complex series of instructions that can be translated (also known as *compiled*) into machine language. Through compilation, these instructions are eventually turned into binary codes that a computer can understand.
Choosing a Language

People have created hundreds of different computer programming languages. You might ask yourself why there are so many programming languages, if they all essentially do the same thing: translate human language into machine language. That’s an excellent question!

There are a few main reasons why there are so many different programming languages. New programming languages are written to allow programmers to

- Write programs in new and better ways than were previously available.
- Write programs for new or specialized types of computers.
- Create new kinds of software.

Examples of computer programming languages include the following:

- C
- Java
- JavaScript
- Logo
- Objective C
- Perl
- Python
- Ruby
- Scratch
- Swift
- Visual Basic
Our short list of programming languages only scratches the surface. For a more complete list of programming languages, visit http://en.wikipedia.org/wiki/List_of_programming_languages.

With so many programming languages to choose from, how do you know which one to use? In many cases, the answer is determined by what you want to do with the languages. For example, if you want to program apps for the iPhone, you have three choices: Objective C, JavaScript, or Swift. If you want to program games to run on Mac or Windows, you have more choices, including C, Java, or JavaScript. If you want to make an interactive website, you need to use JavaScript.

Are you seeing a pattern here? JavaScript is everywhere.

What Is JavaScript?

In the early days of the web, every web page consisted of nothing but plain text in different sizes with links between pages. There were no web forms, there certainly wasn’t any animation, and there weren’t even different styles of text or pictures!

We’re not complaining! When the web was new, it was exciting to click from page to page and discover new things. Even more exciting was how easy the web made it for anyone to be able to publish anything at all and have the potential for anyone else on the Internet to read it.

But when people got a taste of what the web could do, they wanted more features! Graphics, text colors, forms, and many other features were introduced very quickly.

Of all the things that were invented in the earliest days of the web, the thing that has had the biggest impact over the longest time was JavaScript.

JavaScript was created in order to make it possible for web browsers to be interactive. Interactive web pages can range from simple
forms that provide feedback when you make a mistake, to 3D games that run in your web browser. Whenever you visit a website and see something moving, or you see data appearing and changing on the page, or you see interactive maps or browser-based games, chances are, it’s JavaScript at work.

To see some examples of websites that are made possible by JavaScript, open up your web browser and visit the following sites:

- **ShinyText** ([http://cabbi.bo/ShinyText](http://cabbi.bo/ShinyText)): ShinyText is an experimental website that uses JavaScript to display a word. You can adjust different properties of the word, such as Reflection Power and Repulsion Power to see what effect these changes have on how the letters in the word react when you move them around with your mouse. Figure 1-1 shows ShinyText in action.

Even if you don’t understand how it works (we sure don’t!), ShinyText is fun to play with, and it’s a great example of what’s possible with JavaScript.

![Figure 1-1: ShinyText uses JavaScript to produce a 3D physics simulation.](shinytext.png)
Interactive Sock Puppet (www.mediosyproyectos.com/puppetic): Interactive Sock Puppet is another 3D animation. This time, you can control the movements and facial expressions of a JavaScript puppet. Figure 1-2 shows the Interactive Sock Puppet looking quite happy.

Facebook (www.facebook.com): Facebook uses a lot of JavaScript (see Figure 1-3). When you see a smooth animation or video playback, or when a list of posts updates by itself, that’s JavaScript at work!

Some of these examples use some very advanced features of web browsers. We recommend that you use the latest version of Google Chrome to view these. The examples may not work in older web browsers.
Chapter 1: Programming the Web

Get Your Browser Ready

The one essential tool that you need for working with JavaScript is a web browser. You have many different web browsers to choose from, and nearly all of them will do a great job running JavaScript. Odds are, you already have a web browser on your computer.

The most widely used web browsers today are Firefox, Safari, Chrome, Internet Explorer, and Opera. For this book, we’ll be using Chrome. Google Chrome is currently the most popular web browser. It has a number of great tools for working with JavaScript.
If you don’t already have Chrome installed, you’ll need to download and install it. You can install Chrome by opening any web browser and going to www.google.com/chrome/browser/desktop. Follow the instructions found on that page to install Chrome on your computer. When you have Chrome installed, start it up.

In the next section, we show you the Chrome Developer Tools, which help website designers and JavaScript programmers to see exactly what’s going on inside the browser so they can write better web pages and programs.

## Opening the Web Developer Tools

After you have Chrome installed and launched, look at the top of the browser window. In the upper-right corner, you see three lines. This is the icon for the Chrome menu. If you expand the Chrome menu, you see a list of options similar to those shown in Figure 1-4.

![Figure 1-4: The Chrome menu.](image)
If you scroll down to the bottom of this menu and select More Tools, a new menu of options appears, as shown in Figure 1-5. These secret tools are the JavaScript coder’s best friends.

Select Developer Tools from the More Tools menu. A new panel opens at the bottom of your browser window that looks like Figure 1-6.

The Developer Tools give you all the information you need for finding out how any web page works, for testing and improving your own web pages and JavaScript programs, and much more.

Notice that the there’s a menu at the top of the Developer Tools with different options, including Elements, Network, Sources, Timeline, Profiles, Resources, Audits, and Console. If you click each of these, you’ll see a different set of options and data in the Developer Tools panel.

We describe the different components of the Developer Tools as they become necessary throughout this book, but for now, the most important part of the Developer Tools is the one labeled Console. Click the Console tab now.
Introducing the JavaScript Console

The Developer Tools Console, also known as the JavaScript Console, shown in Figure 1-7, gives you information about the JavaScript that’s currently running in the browser window.

If there are errors in the JavaScript code of a web page, you see information about the errors in the console. This is a very helpful tool and one of the main features of the JavaScript Console.

Another very cool capability of the console is that you can type JavaScript into the console panel and it will run. In the next section, you learn why this is useful and how to do it.

The JavaScript Console is a useful tool for JavaScript programmers, but it also has the potential to be misused. If someone you don’t know or trust asks you to paste code into the JavaScript Console, make sure you understand what that code does first.
Chapter 1: Programming the Web

Now it’s time to start experimenting with some real JavaScript code! If you don’t already have it open, open the JavaScript Console by selecting it from the Other Tools menu under the Chrome menu, or by clicking the Console tab in the Developer Tools.

Follow these steps to run your first JavaScript commands:

1. Click inside the JavaScript console, near the >, to start inserting code.

2. Type 1 + 1 and then press Return (Mac) or Enter (Windows).

   The browser gives you the answer on the next line.

Notice that when the answer is returned to you, it has an arrow on the left side of it that points to the left. This arrow indicates that the value came from JavaScript rather than from your input. Any value that comes from JavaScript is called a return value. Every command that you run in JavaScript produces some sort of return value.

Simple math is one thing, but JavaScript can do much, much more. Let’s try out some other commands and see just how quickly we can get some answers around here.
Before we get started, let’s clean up the console and remove any previous commands, errors, and return values in there. To clear the console, look at the upper-left corner and click the circle with the line through it. Everything inside the console will be erased, and now you’ve got a clean slate.

Click your mouse next to the > and try out the following JavaScript commands. Make sure to press Return (Mac) or Enter (Windows) after each one to see the results.

<table>
<thead>
<tr>
<th>JavaScript Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000 - 37</td>
<td>This is a simple math problem, but this time we’re using the minus sign to subtract the number on the right from the number on the left.</td>
</tr>
<tr>
<td>30 * 27</td>
<td>The asterisk (*) is how you tell JavaScript to multiply numbers.</td>
</tr>
<tr>
<td>120 / 20</td>
<td>The forward slash (/) tells JavaScript to divide the number on the left by the number on the right.</td>
</tr>
<tr>
<td>&quot;Your name&quot; + &quot; is learning JavaScript!&quot;</td>
<td>Yes, you can add words together with JavaScript! When you run a command that adds words together, it’s called <em>concatenation</em>. The result will be that the words are combined into a single word. Notice that the words in the above JavaScript command are inside quotes. These quotes are very important. We tell you exactly why they’re important in Chapter 2.</td>
</tr>
<tr>
<td>Your name + is learning JavaScript!</td>
<td>When you don’t use quotes, JavaScript doesn’t like that one bit. It returns an error message containing the keyword <em>SyntaxError</em>. A syntax error means that you’ve written something that isn’t valid JavaScript. Any time you see a syntax error, it means that you’ve goofed. Take a close look at your code and look for typos, missing punctuation, or missing quotes.</td>
</tr>
</tbody>
</table>
Having Fun with Math

Now it’s your turn to try out some math problems on your own! Clear out your commands and the return values and errors from the previous section and experiment with the console.

Here are some ideas to get you started:

▫ Multiply together two decimal numbers.

▫ Run multiple commands in one line (for example, \(1 + 1 \times \frac{4}{8}\)).

▫ Type a number without any symbols at all and then run it.

▫ Add a word (remember to use quotes!) to a number (without quotes).

▫ Add a number (without quotes) to a word (with quotes).

▫ Combine your first name with the last name of your celebrity crush. Remember to add a space between the first and last name! For example, "Eva" + " " + "Harry Styles".

▫ Try to produce extremely large return values.

▫ Try to produce extremely small return values.

▫ Try to do an impossible math problem, such as dividing a number by zero.

▫ Try multiplying a number by a word (in quotes). For example, 343 \(* "hi!"\). The result of this will be NaN, which stands for “not a number.”
CHAPTER 2

Understanding Syntax

Just as spoken languages have rules (called grammar), computer programming languages have rules (called syntax). When you understand the basic rules of speaking JavaScript, it actually looks similar to English.

If you thought that your teacher correcting you when you say “ain’t” was strict, wait until you see how strict JavaScript is! It won’t even listen to a thing you say if you make certain kinds of syntax errors.

In this chapter, you learn the basics of JavaScript syntax and how to avoid being scolded by the syntax police!
Saying Precisely What You Mean

In order to be compiled correctly into machine language instructions, programs need to be written very precisely.

Chapter 1 explains what a program is and how programs are translated into machine language using the process called **compilation**.

As a programmer, your job is to think about the big picture of what you want the program to do, and then break it down into bite-size steps that can be accomplished by the computer without errors. For example, if you wanted to ask a robot to go downstairs and get you a sandwich, you might start your instructions like this:

1. Rotate head toward stairs.
2. Use visual sensors to look for obstacles.
3. If an obstacle is found, determine what it is.
4. If the obstacle is a cat, try to lure the cat away from the top of the stairs by:
   - Throwing a toy down the hall
   - Speaking the cat’s name
   - Gently nudging the cat with your hand until it walks away
5. If there is no obstacle, rotate left foot in the direction of the stairs.
6. Place left foot in front of right foot.
7. Look for an obstacle.
8. Determine whether you’re at the top of the stairs.
9. If you’re not at the top of the stairs, rotate right foot in the direction of the stairs.

10. Place right foot in front of left foot.

11. Repeat steps 1 through 10 until you’re at the top of the stairs.

You’ve written 11 instructions already and the robot hasn’t even started walking down the stairs, much less making a sandwich!

A real computer program to tell a robot to go downstairs and make a sandwich would need to contain far more detailed instructions than the ones shown here. At each step along the way, each motor would need to be told precisely how long to turn on, and each possible condition and obstacle would need to be described and dealt with in detail.

All these instructions need to be written as individual JavaScript commands, or *statements*.

You can find out more about how to control robots with JavaScript by visiting [http://nodebots.io](http://nodebots.io)!

### Making a Statement

In English, we talk in sentences. In JavaScript, a single instruction to the computer is called a *statement*. Like a sentence, statements are made up of different parts and have certain rules that they must follow in order to be understood.

Listing 2-1 shows an example of a statement.

**Listing 2-1  A JavaScript Statement**

```javascript
alert("Coding is fun!");
```

This statement causes a web browser to open up a popup alert window with the sentence “Coding is fun!” If you type this
statement into the JavaScript Console in Chrome, you’ll see something like what’s shown in Figure 2-1.

![Figure 2-1: The output of a JavaScript alert statement.](image)

Notice that the statement in Listing 2-1 contains a keyword, some symbols (parentheses and quotes), and some text (Coding is fun!), and it ends with a semicolon.

Just as an infinite number of sentences can be written using English, an infinite number of statements can be written with JavaScript.

The word `alert` is an example of a JavaScript keyword. Many JavaScript statements begin with keywords, but not all of them do.

The semicolon is what separates one statement from another, just as a period separates one sentence from another. Every statement should end with a semicolon.

### Following the Rules

JavaScript has several rules that must be obeyed if you want your computer to understand you. The first two rules are:

- Spelling counts.
- Spacing doesn’t count.

Let’s take a look at each of these rules in more detail. We’ll write a new message printer program to serve as an example. Listing 2-2 is a JavaScript program that prints out the words “Coding is fun!” 300 times.
Listing 2-2 A Program to Print a Message 300 Times

```javascript
for (var i = 0; i < 300; i++) { document.write ("Coding is fun"); }
```

Follow these steps to test this program:

1. Open the Chrome web browser.
2. Open the JavaScript Console from the More Tools menu under the Chrome menu.
   
   You can also use the keyboard combination to open the JavaScript Console. Press ⌘+Option+J (Mac) or Ctrl+Shift+J (Windows).
3. Type the program in Listing 2-2 onto one line in the JavaScript Console and press Return (Mac) or Enter (Windows).

   If you entered everything correctly, you’ll see the message appear in your browser window 300 times, as shown in Figure 2-2.

![Figure 2-2: The result of running the program in Listing 2-2.](image-url)
This “Coding is fun!” program uses a technique called a for loop in order to do something many times with only a little bit of code. We talk more about for loops in Chapters 17 and 18.

Take a close look at the code in Listing 2-2. Notice that the text that gets written to the browser window is enclosed in quotes. The quotes indicate that this text is to be treated as words, rather than as JavaScript code.

**Using text in strings**

In programming, we call a piece of text inside of quotes a *string*. You can remember this name by thinking of text inside quotes like a piece of string with letters, numbers, and symbols tied to it. These letters stay in the same order and each one takes up a certain amount of space on the string.

For example, try typing the code from Listing 2-2 into your JavaScript console again, but change *Coding is fun!* to another message, such as what you want for lunch or dinner.

Figure 2-3 shows the output of the program from Listing 2-2 when the message is changed to “I want pizza for lunch!”

Any character you can type can be put into a string. However, there’s one important exception that you need to remember: If you want to use quotation marks inside a string, you have to tell JavaScript that the quotation marks are part of the string, rather than the end of the string.

The way to put quotation marks inside a string is by using a backslash (\) before the quotation marks. Using the backslash in a string tells JavaScript that the next character is something special and doesn’t mean what it normally would mean. When you add a backslash before a quotation mark in a string, it’s called *escaping* the quotation mark.

For example, if you want to change the string to:

```
Joe said, "Hi!"
```
You would need to write the string as:

"Joe said, \"Hi!\""

Listing 2-3 shows our message printer program with escaped quotation marks in the message.

Listing 2-3   Escaping Quotation Marks

    for (var i = 0; i < 300; i++) { document.write ("Joe said, \"Hi!\"\"); }

You might be asking yourself now, “If the backslash is used to tell JavaScript that the next character is special, how do I print out a backslash?” Great question! The answer is just to use two backslashes (\\) for each backslash that you need to print out.

As with most things in JavaScript, there is another way to use quotes inside a string: by surrounding the string with different quotes. JavaScript doesn’t care whether you use single quotes (‘) or double quotes (“) to mark text as a string, as long as you use the same type of quotes at the beginning and end of the string.
If you surround your string with single quotes, you can use all the
double quotes that you want inside the string, without escaping
them. But single quotes must be escaped.

If you surround your string with double quotes, you can use all
the single quotes you want inside the string, but double quotes
must be escaped.

Listing 2-4 shows the message printer program with the string in
single quotes and double quotes inside the string.

### Listing 2-4 Double Quotes within Single Quotes

```javascript
for (var i = 0; i < 300; i++) { document.write (' Joe
said, "Hi!" '); }
```

**Using text in code**

Unlike in strings, the contents and spelling of text outside of
quotes matters a lot. When text isn’t surrounded by quotes (single
or double) in JavaScript, it’s considered part of the code of the
JavaScript program.

JavaScript code is very picky about spelling and capitalization.
In JavaScript code, the following words are completely different:

```
FOR
for
For
```

Only the one in the middle means anything special to JavaScript.
If you try to use the other two in the message printer program,
you’ll get an error, as shown in Figure 2-4.

The special meaning of `for` is explained in Chapter 17.

JavaScript is also very picky about spelling. Many times, when
we’re coding and something just isn’t working right, the problem
turns out to be that we accidentally left out a letter or mixed up
the order of two letters.

Just as typos in writing often go unnoticed, these types of errors can be very difficult to track down, so get into the habit early on of typing slowly and carefully and you’ll save yourself a lot of time in the long run!

Paying attention to white space

White space is all the spaces, tabs, and line breaks in your program. JavaScript ignores white space between words and between words and symbols in code. For example, in our message printer program, we could make the whole thing easier for people to read by spacing it out over multiple lines, as shown in Listing 2-5.

Listing 2-5 White Space Makes Programs Easier to Read

```javascript
for (var i = 0; i < 300; i++) {
    document.write("Coding is fun!");
}
```

Listing 2-5 shows the way that we would recommend spacing out this program.

Notice that we’ve inserted line breaks after the opening curly bracket (`{`) and before the ending curly bracket (`}`). Curly brackets are used for grouping pieces of code (also called statements) together into what’s called a block. In this program, they mark the part of the program that should be repeated 300 times — namely, printing out a message.

Curly brackets are a good spot to put some white space to help you read the code more easily. Another great spot to put a line
break is after each semicolon (;). In JavaScript, the semicolon is used to mark the end of a statement, much as a period is used to mark the end of a sentence in English.

If you try to run the program split over three lines in the JavaScript Console in Chrome, you’ll get an error message when you press Return (Mac) or Enter (Windows) after the first line. This is because the console tries to run your code every time you press Return or Enter, and the first line (ending with {) isn’t a complete JavaScript statement. To enter this code into the console with line breaks, hold down the Shift key while pressing Return or Enter after each of the first two lines.

Notice that the statement between the curly brackets is indented. The indentation helps people reading the code to see that this statement is happening inside another statement — namely, the for statement that creates the loop.

We recommend using either two spaces or four spaces to indent statements. Some people use tabs to indent statements. Which one you use is up to you. Once you decide, however, stick with it. If you use two spaces to indent code inside of a block, you shouldn’t sometimes use four spaces or a tab. Neatness counts!

**Making comments**

JavaScript comments are a way that you can put text into a program that isn’t a string or a statement. This may not sound so great, but the thing that makes comments so important and useful is precisely that they don’t cause JavaScript to do anything at all.

Programmers use comments within their code for several reasons:

- To tell their future selves, and anyone else who works on the program in the future, why they wrote something in the particular way they did

- To describe how the code they wrote works
To leave themselves a note telling what they still need to do, or to list improvements that they intend to make at a later date.

To prevent JavaScript statements from running.

JavaScript has two different kinds of comments: single-line and multi-line.

**Single-line comments:** Single-line comments cause everything following them on the same line to be a comment. To create a single-line comment, use two slashes (//) back to back. For example, in Listing 2-6, the first three lines are single-line comments and the fourth line contains a statement that will be executed, followed by a comment.

**Multi-line comments:** Multi-line comments are comments that can be more than one line long. To create a multi-line comment, start with /* and end the comment with the exact reverse, */. Listing 2-7 shows an example of a multi-line comment.

### Listing 2-6  Single-Line Comments

```javascript
// The following code won’t run.
// alert("Watch out!");
// The next statement will run.
alert("Have a nice day!"); // pops up a nice message
```

### Listing 2-7  A Multi-Line Comment

```javascript
/*
   AlertMe, by Chris Minnick and Eva Holland

   A program to alert users that they are using a JavaScript program called AlertMe, which was written by Chris Minnick and Eva Holland.
*/
```
Giving and Receiving Data

Programs come in many different sizes and have many different purposes. Here are three things all programs have in common:

✓ A way to receive information from the user
✓ A way to give information back to the user
✓ A way to store and work with information in between giving and receiving

Information, or data, that a program receives from a user is called *input*. What the program gives back to the user is called *output*. In the time between when a program receives input and produces output, it needs some way to store and work with the various types of data that has been inputted, so it can produce output.

The question of whether it’s better to give or receive isn’t important! It’s all good. In this chapter, you learn how JavaScript can help you to get, receive, and just plain have data!

Dear Eva,

We are pleased to inform you that your song, ‘Can’t Stop Coding’, has been voted the Best Song of All Time by the awarding committee.

Sincerely,
The Grammy Awards
Mastering Variables

In the real world, when you want to store something, give something away (as a gift, for example), move something, or organize something, you often put it in a box.

JavaScript doesn’t care about heart-shaped boxes of chocolates or shoeboxes with the latest sneakers. What JavaScript loves is data. To store and move around data, JavaScript uses a special kind of box called a variable. A variable is a box you can assign a name to. This name will represent all the data contained in that box, or variable.

Variables make it possible for the same program to work with different input to produce different output.

Creating variables

Creating a variable in JavaScript is pretty simple. To create a variable, you use the `var` keyword, followed by the name of the variable, and then a semicolon, like this:

```javascript
var book;
```

As a programmer, you have a lot of flexibility when naming your variables. You can be very creative in naming your variables, but you don’t want to get too crazy. Most important, your variable names should accurately describe the data that you store inside them.

Each of the following variable declarations creates a variable with a good and descriptive name. By looking at them, you can probably guess what the data stored inside them looks like.

```javascript
var myFirstName;
var favoriteFood;
var birthday;
var timeOfDay;
```

Notice how we separate words in variable names by using capital letters for every word after the first one. Variable names can’t
contain spaces, so programmers have created several other ways to separate words. This particular style is called **camelCase**. Can you guess why it has that name?

After looking at these examples, what would you name variables for storing the following pieces of data?

✓ Your pet’s name
✓ Your favorite school subject
✓ The age of your best friend
✓ Your street address

In addition to the rule that variable names must not contain spaces, there are several other rules that you must follow:

✓ Variable names must begin with a letter, an underscore (_), or a dollar sign ($).

✓ Variable names can only contain letters, numbers, underscores, or dollar signs.

✓ Variable names are case sensitive.

✓ Certain words may not be used as variable names, because they have other meanings within JavaScript. These so-called **reserved words** are as follows:

```
break    case    class    catch
const    continue    debugger    default
delete    do    else    export
extends    finally    for    function
if    import    in    instanceof
let    new    return    super
switch    this    throw    try
typeof    var    void    while
with    yield
```
Storing data in variables
After you’ve created a variable, you can store any sort of data inside it. When the data is in there, you can recall it at any time. Let’s try it out!

1. Open the JavaScript Console in Chrome.

2. Create a new variable named `book` by typing the following and then pressing Return (Mac) or Enter (Windows):
   
   ```javascript
   var book;
   ```

   You’ve created your container, or variable, and named it “book.”

   When you press Return or Enter, the JavaScript Console displays the word `undefined`. This is exactly what you want to happen. JavaScript is just telling you that your code ran correctly and that it doesn’t have anything to tell you.

   It may seem funny that JavaScript tells you that it has nothing to tell you. But, trust us, it’s way better that it says something, even if it’s just `undefined` than if it were to give you the cold shoulder and say nothing at all.

3. Put a value into your new variable by typing the following code.

   ```javascript
   book = "JavaScript For Kids For Dummies";
   ```

   You’ve now put data inside your variable, where it will be stored.

   When you press Return or Enter, JavaScript responds with the name of the book.

   You only need to type `var` when you first create and name your variable. When you want to change the data inside your variable, you only need to use the variable’s name.
4. Now, temporarily forget the name of this book. Got it? Now, imagine that you need to recall the name of this book so that you can tell your friend about it! To recall the data, or value, in a variable, you can just type the name of the variable in the console. So, type the following:

```
book
```

The console recalls the string that was assigned to the book variable and prints it out, as shown in Figure 3-1.

![Figure 3-1: Printing out the value assigned to a variable.](image)

Notice that we didn’t use a semicolon ( ; ) when typing — we just used a variable name in the JavaScript Console. The name of a variable isn’t a full JavaScript statement, so it doesn’t require a semicolon. We’re just asking JavaScript for the value of the variable, just as if we had asked it `1 + 1`.

5. Now try changing the value of the book variable by typing the following statement into the JavaScript Console:

```
book = "The Call of the Wild";
```

6. Type `book` into the JavaScript Console to retrieve its new value.

The console prints out “The Call of the Wild” (or whatever you entered as the new value of `book`).
In addition to text, variables can also hold several other different types of data. In the next section, we show you each of the basic (also known as *primitive*) data types that JavaScript understands.

The data inside a variable can also be called the *value* of a variable.

### Understanding Data Types

JavaScript variables have just one job — to hold and store data — and they do this job quite well. Using and creating variables are easy. There are many different types of data in the world, such as numbers, letters, and dates. JavaScript makes some important distinctions between these and other different kinds of data that you, as a coder, need to be aware of.

Data types are how a program knows whether 03-20-2017 is a date (March 20, 2017) or a math problem (subtract 20 from 3 and then subtract 2017 from the result).

JavaScript recognizes three basic data types: string, number, and Boolean.

#### The string data type

The string data type holds text. We explain the basics of how strings work in Chapter 2, but there are a few other cool tricks that you can do with strings besides just storing and printing them.

One cool string trick is to count how many characters the string is made up of. The way you do that is to use `.length` after the string, or after a variable holding the string.

For example, to find out the length of the string held inside the `book` variable you create in the previous section, type `book.length` into the console. The console responds right away with a number, as shown in Figure 3-2.
Chapter 3: Giving and Receiving Data

Every string, even an empty string, has a length. The length of an empty string, of course, is 0. Because it’s something that describes a string, we call length a property of a string.

You see the word property used a lot when people talk about JavaScript. A property is something that describes or is a part of something. For example, a car’s color is a property of the car, a person’s name is a property of the person, and a string’s length is a property of the string.

In addition to finding out the length of a string stored in a variable, you can also just attach the length property to a string in quotes to find out its length:

```
"I am a string.".length
```

Count the letters in this sentence. There are 10 — 11 if you count the period at the end of the sentence. But when you enter this command into the JavaScript console, you get 14. Do you know why?

The spaces in a string count just as much as the letters, punctuation, symbols, and numbers in the string. To use the analogy we make in Chapter 2, it’s all just knots on the string (14 of them to be precise) to JavaScript.

In addition to properties, strings also have things that they can do, or that can be done to them. In programming, we call these things that can be done with or to something its methods.
The most commonly used string method is `indexOf`. The job of `indexOf` is to look at your string, find a certain character or group of characters inside it, and tell you what position they’re at. In the following statement, we look for the position of the word `am` in a string:

```
"I am a string.".indexOf("am");
```

When you run this statement in the console, the result is 2. Try retyping the command, but this time look for the capital `I`.

```
"I am a string.".indexOf("I");
```

The result is 0.

This brings us to a very important concept in JavaScript called zero-based numbering. Unlike people, who have ten fingers and generally start counting at the number one, JavaScript starts counting at zero. So, in the previous example, when JavaScript wants to tell you that `I` is the first character in the string, it says that `I` is at position 0.

If JavaScript were on a sports team, it would proudly wear a shirt that read “We’re number 0!”

**The number data type**

Another type of data that JavaScript understands is the number data type. Numbers can be positive or negative, as well as whole numbers or decimal numbers. Numbers are stored in variables without using quotation marks.

The range of possible numbers that can be used in JavaScript goes from very, very small to very, very large. We won’t bore you with a bunch of zeros right now, but the biggest number that you can use in JavaScript is far greater than the number of stars in the universe. It’s even bigger than the number of atoms in the universe! JavaScript can do any math problem or counting problem that you would want it to do.
One thing to watch out for, however, is what happens when you try to combine two different data types, such as strings and numbers.

JavaScript generally tries to be pretty clever. If you open the console and type “10” + 10, JavaScript will assume that you meant for both pieces of data to be strings, and will put them together and give you the result 1010.

On the other hand, if you type 10 * “10”, JavaScript will assume that you meant for the string "10" to actually be the number 10, and it will give you the result 100. JavaScript does this because it knows there is no way to multiply two strings together.

**The Boolean data type**

The Boolean data type can store one of two possible values: true or false.

Boolean values are the result when you do comparisons in JavaScript, which we cover in more detail in Part V of this book. If you ask JavaScript something like: “Is 3 equal to 30?”, it will respond with a Boolean value of false.

The Boolean data type is named after the mathematician George Boole, so it’s always capitalized.

Let’s do a few experiments with Booleans. Open the JavaScript Console and try typing each of the following statements, pressing Return or Enter after each one to see the result. Note that we’ve used a single-line comment after each statement to explain what it means. You don’t need to type these comments into the console, but you can if you want.

```
1 < 10    // Is 1 less than 10?
100 > 2000 // Is 100 greater than 2000?
2 === 2    // Is 2 exactly equal to 2?
false === false // Is false exactly equal to false?
40 >= 40   // Is 40 greater than or equal to 40?
```

- Boolean (0) // What is the Boolean value of 0?
- Boolean (false) // What is the Boolean value of false?
- "apples" === "oranges" // Is "apples" exactly equal to "oranges"?
- "apples" === "apples" // Is "apples" exactly equal to "apples"?

In addition to the statements you would expect to be false, JavaScript also considers the following values to be false:
- 0
- null
- undefined
- " " (an empty string)
- false

Prompting the User for Input

Now that you know how variables can hold different types of data, let’s explore the process of getting data from a user and storing it inside your variables.

One way to ask a user for data is by using the `prompt` command. To try out the prompt command, open the JavaScript console and type the following:

```
prompt("What is your name?");
```

After you press Return or Enter, a pop-up window appears in your browser window with a text field, as shown in Figure 3-3.

After you enter your name and click OK, the pop-up window disappears, and the value that you entered in the pop-up displays in the console, as shown in Figure 3-4.

That’s all well and good if all you want to do is capture data and immediately repeat it back like a parrot. But what if you want to do something with the user-entered data? To do that, you need to store it in a variable.
Storing user input

To store user-entered data in a variable, you create a new variable and then follow it with =. You then follow it with the prompt statement.

```javascript
var username = prompt("What is your name?");
```

It’s important to note that a single equal sign (=) in JavaScript is called the assignment operator. Its job is to put the value on the right into the variable on the left. We talk more about operators in Chapter 9.
When you press Return or Enter, a pop-up window appears in your browser, just as before.

When you enter your name in the pop-up window and click OK, the JavaScript Console prints out `undefined`, indicating that the statement is finished and there’s nothing else for it to do.

To see the value you just entered, you can type the variable name into the console. JavaScript responds with the value of the variable, as shown in Figure 3-5.

![Figure 3-5: Getting the value of a variable from a prompt.](image)

**Responding to Input**

Now that you know how to get data from the user, and how to store that data, let’s take a look at two of the ways that you can use JavaScript to respond to the user.

**Using alert()**

The `alert()` command pops up a notification box in the user’s browser containing whatever data is between the parentheses.
If you want to display an alert with a simple string message, you can do so by enclosing a message within quotes between the ( and ) after alert. For example, type the following statement into your JavaScript Console:

    alert("Good job!");

When you press Return or Enter, the browser displays an alert message containing the message “Good job!”

You display numbers in alerts by putting numbers without quotes between the parentheses. For example, try this statement:

    alert(300);

The alert pop-up displays the number 300. You can even do math inside an alert. For example, try this one:

    alert(37*37);

The alert displays the result of multiplying 37 and 37.

If you put a word between the parentheses in the alert statement without quotes, JavaScript treats the word as a variable. Try running the following two statements:

    var myNameIs = "your name";
    alert(myNameIs);

The browser pops up a window containing your name.

By combining different data types into one alert statement, you can start to do some really interesting and useful things. For example, try typing each of the following statements into the JavaScript Console, one at a time:

    var firstName = "your name";
    var yourScore = 30;
    alert("Hi, " + firstName + ". Your current score is: " + yourScore);
As you can see, by using `alert()`, you can create all sorts of fun and interesting pop-ups to entertain and inform the user, such as the alert in Figure 3-6.

![Figure 3-6: Creating interesting pop-ups.](image)

Objects are a special data type in JavaScript, like numbers and strings. However, objects are flexible and can store data about anything using properties and methods.

You can picture JavaScript objects as being like objects in the real world. For example, in the real world, you can have a yellow truck. In JavaScript, this yellow truck object would have a color property of yellow and we would write it like this:

```javascript
truck.color="yellow";
```

The truck would also have a method called `drive`, and we would write that like this:

```javascript
truck.drive();
```
Using `document.write()`

In JavaScript, a web page is called a `document`. When you change something on the current web page using JavaScript, you do so by telling JavaScript to change the document object.

One way to make changes to the current web page is by using the `write` method.

A method is something that can be done or that something can do.

Every document (or web page) has a `write` method that causes whatever you put between the parentheses after the method name to be inserted into the web page. You can use `document.write()` in the same ways that you used `alert()`. For example, open a new, blank browser window and try out the following statements in your JavaScript Console:

```javascript
document.write("Hi, Mom!");
document.write(333 + 100);
```

Notice that statements after the first one are added right after the first statement, without a line break or space. You can add space after or before writing text with `document.write` by using the characters `<br>`. For example:

```javascript
document.write("How are you?<br>");
document.write("I'm great! Thanks!<br>");
document.write("That's awesome!");
```

You can clear out the current contents of the browser window by typing `chrome://newtab` into the browser address bar or by opening a new browser tab.

The result of entering these three lines into the JavaScript Console is shown in Figure 3-7.

`<br>` is an HTML tag. We talk much more about HTML in Chapter 5.
Combining Input and Output

Now, let’s combine input and output to display customized output, based on input from a user. This is really the heart of what JavaScript can do for web pages!

Follow these steps in the JavaScript Console to create a letter to yourself in your web browser. Make sure to press Return or Enter after the end of each statement (after each semicolon).

1. Type the following to create a variable containing your first name.

   ```javascript
   var toName = "your name";
   ```

2. Type the following to create a variable containing the person the letter is from:

   ```javascript
   var fromName = "The Grammy Awards";
   ```

   You can change The Grammy Awards to anyone you’d like to get a letter from.
3. Type the contents of your letter into a variable.

Use `<br>` to insert line breaks and don’t press Return or Enter until after you type the semicolon.

Here’s the letter we came up with:

```javascript
var letterBody = "We are pleased to inform you that your song, 'Can\'t Stop Coding!, has been voted the Best Song of All Time by the awarding committee."
```

4. Write `document.write()` statements to output each of the three parts of your letter.

For example:

```javascript
document.write("Dear " + toName + ",
letterBody + "\nSincerely,
fromName);
```

When your letter is done, it should resemble ours, shown in Figure 3-8.

Figure 3-8: A fully customized letter displayed in the browser.
In Chapter 1, we explain and demonstrate the JavaScript Console. In Chapters 2 and 3, we show you how to put multiple statements together to form a program. In this chapter, we kick things up a notch and introduce you to our favorite JavaScript playground: JSFiddle. Instead of swings and slides, you’ll be playing with JavaScript statements, HTML tags, and CSS styles.

JSFiddle lets you write and experiment with JavaScript code from within your web browser. You can use it to try out code, get feedback on your code, share your code, and even work on programs with your friends! You’ll learn how to use JSFiddle to view, modify, save, and share JavaScript web applications, too.

You may be wondering what we mean by web application. A web application (or web app) is software that runs in a browser and is usually powered by JavaScript. Google Earth, for example, is a popular web app you may be familiar with. It can look up and show you nearly any place on Earth in high-resolution photos. Google Earth is also a website because you can access it using a web address, or URL. Do you think JSFiddle is a web app, a website, or both? It’s actually both. In fact, every web application is a website. Not all websites are web applications, however.

In this chapter, you use JSFiddle to experiment with some animations. In the end, you have a JavaScript bubble machine that you can customize as much as you want! It’s called JSFiddle because you can use it to “fiddle” with JavaScript. So, let the fiddling begin!
Introducing JSFiddle

To get started with JSFiddle, open your web browser and type http://jsfiddle.net into the address bar. You’ll see the JSFiddle website, shown in Figure 4-1.

JSFiddle’s user interface window consists of three panes where you can enter different types of code including HTML, CSS, and JavaScript. You see the results of what you type inside these boxes in the Result pane. The toolbar on the left lets you configure additional options, and the top toolbar has buttons for running, saving, and cleaning up your code.

You can resize any of the panes in JSFiddle by clicking and dragging the border that separates them.

For now, we’re mostly concerned with the JavaScript pane. The JavaScript pane works in much the same way as the JavaScript Console. With JSFiddle, the code you enter won’t execute until you tell it to run.

Figure 4-1: JSFiddle’s clean and compartmentalized user interface.
Follow these steps to run your first JSFiddle program:

1. Click inside the JavaScript pane.

2. Type the following JavaScript statement:

   ```javascript
   alert("Hi, everyone!");
   ```

3. Click the Run button on the top toolbar.

   A pop-up window containing the message “Hi, everyone!” appears.

4. Close the pop-up window by clicking OK.

There’s nothing surprising in the behavior of that simple JavaScript program. If you’ve read the first three chapters of this book, you’re already familiar with how the `alert` statement works.

Running JavaScript isn’t the only great thing JSFiddle can do. With JSFiddle, you can also use the HTML and CSS panes to run code that works together with your JavaScript code! In the next few sections, we cover each of these panes in more detail and demonstrate their use. But first, we give a quick demonstration of what JSFiddle is capable of.

**Viewing our fiddles**

We’re going to let you in on a secret. Every program in this book is available for you to view, run, copy, and play around with at [http://jsfiddle.net/user/forkids/fiddles](http://jsfiddle.net/user/forkids/fiddles). That’s right! We’ve done your homework! We’ve formatted it nicely for you and tested it out.

This is our own JSFiddle public dashboard. The public dashboard is where any JSFiddle user can share programs (known as “fiddles” in JSFiddle) with the world.
Although we’ve typed up every project for you already, it’s important that you go through each step of every project for yourself so that you really understand. To get the most out of this book, feel free to copy, modify, completely change, and rewrite our code to see what it does and make it your own! Keep on fiddlin’!

**Playing with fiddles**

Before you get too carried away with viewing all the cool projects from other parts and chapters, take a look at some programs that aren’t part of this book. JSFiddle lets anyone create an account and share their programs in a public dashboard — and many excellent and very experienced JavaScript programmers do!

When programmers share their programs on JSFiddle, they agree that anyone who wants to can make a copy of their work, change it, and republish it. However, it’s always polite to give the original author credit when you borrow code. We’ve made copies of each of the programs below so that we can be sure that they’ll be the same when you view them. If you want to find out who the original author of a program is, open Fiddle Options from the left navigation bar.

Follow these steps to view and run some of the programs in our list of amazing JSFiddle demos:

1. Go to our public dashboard at [http://jsfiddle.net/user/forkids/fiddles](http://jsfiddle.net/user/forkids/fiddles).

   You see a list of all the examples and projects from the entire book.

   You may need to use the page navigation at the bottom of the list to see additional pages of results.

2. Find a demo that sounds interesting to you and open it.

   When the program opens, it will automatically start running.

   If you find a program that you like, try figuring out how it works! Change some values to see what happens.
Anything that you do to a program in JSFiddle won’t overwrite the original. You can try changing things all you want, and no harm will come of it. The worst that can happen is that the program won’t run.

**Fiddling with CSS**

The CSS pane in JSFiddle is located in the upper-right corner. Besides working with JavaScript in JSFiddle, we can also fiddle with the Cascading Style Sheets (CSS) in our web application. CSS allows you to change how elements such as text and graphics appear. If you want to change the color of the text on your page, you use CSS.

We cover CSS in much more detail in Chapter 6. For now, follow these steps to try out changes to one of our programs:

1. Go to [http://jsfiddle.net/forkids/vaj023L5](http://jsfiddle.net/forkids/vaj023L5).

   You see the Bubbles demo, shown in Figure 4-2.

![Figure 4-2: The Bubbles demo.](image)
Listing 14-2 expands the program from Listing 14-1 to create a variable called `speaksJavaScript` when you enter `JavaScript` into the prompt.

If you type in `JavaScript` correctly, the statement within the `if` block will execute, displaying a special message for JavaScript speakers only. If you enter anything other than `JavaScript`, the statement within the `else` block will execute, so that a different message will display.

Listing 14-2  Using Single-Word Operators

```javascript
var language = prompt("What language do you speak?");

if (language === "JavaScript") {
    alert("Great! Let's talk JavaScript!");
    var speaksJavaScript = true;
} else {
    alert("I don't know what you're saying.");
}

if (speaksJavaScript) {
    alert("It's great to meet you.");
}
```

Combining Comparisons with Logical Operators

Logical operators allow you to combine more than one comparison operation. For example, let’s say you own a pizza parlor. Your policy is that if a customer’s order is more than $10 and they live within the city limits, they get free delivery.

In JavaScript, this rule requires two comparisons:

uderline.Is the order over $10?

uderline.Is the customer located within the city limits?
In order for the customer to get free delivery, both of these conditions have to be true. If one of these conditions is not true, the delivery charge is $5.

In JavaScript, you can specify that two conditions both need to be true by using the **and operator** (`&&`). To use the and operator in an if...else statement, you put it between two comparison expressions. Then you surround the whole combination expression with parentheses.

Listing 14-3 shows how you might write your pizza parlor’s delivery rule in JavaScript.

**Listing 14-3  Pizza Parlor Free Delivery Rule in JavaScript**

```javascript
if ((deliveryCity === "Anytown") && (orderPrice > 10)) {
    var deliveryPrice = 0;
} else {
    var deliveryPrice = 5;
}
```

As a special deal, you might decide to offer free delivery to people when it’s their birthday, no matter how far away they live or the size of their order. In order to do this, you need to use the **or operator** (`||`). You type the or operator by holding down the Shift key and pressing the backslash (`\`) character on your keyboard twice.

Listing 14-4 shows how to write the new free delivery policy in JavaScript.

**Listing 14-4  Free Delivery on Your Birthday**

```javascript
if (((deliveryCity === "Anytown") && (orderPrice > 10)) ||
    (birthday === "yes")) {
    var deliveryPrice = 0;
} else {
    var deliveryPrice = 5;
}
```

In the next section, we start with this free delivery policy and create a program for managing several different parts of your pizza parlor.
Freshening Up the JavaScript Pizzeria

The JavaScript Pizzeria is a little mom-and-pop place in Anytown, USA. They pride themselves on making good pizzas at a good price and keeping things simple.

Currently, they have a web page where you can order one of their two kinds of pizza — cheese or pepperoni — and have it delivered to you for free if you live inside the city limits of Anytown, USA.

Customers are demanding more, though! They want additional pizza options. And people in other cities have been hearing about JavaScript Pizzeria, and they want pizza delivered, too! Some people have even asked for a special deal on their birthday!

As the JavaScript programmer for the JavaScript Pizzeria, it’s your job to whip up these new features so that the business continues to thrive! Don’t worry, we’re here to help.

Running the app

To test out the current version of the JavaScript Pizzeria website, follow these steps:

1. Go to our public dashboard on JSFiddle at http://jsfiddle.net/user/forkids/fiddles.

2. Find the program named “Chapter 14 – JavaScript Pizzeria – Start” and click its title to open it.

3. Enter a number of pizzas, select a pizza type, and press the Place Order button.

   The total (at $10 per pizza) will display below the form.

That’s all there is to it! Move on to the next section to create your own version of the JavaScript Pizzeria that you can add new features to.
Forking the code (or just using your hands)

Follow these steps to create your own copy of the JavaScript Pizzeria program that you can add new features to:

1. Open the program named “Chapter 14 – JavaScript Pizzeria – Start.”

2. Click the Fork button in the top menu bar.

3. Change the name of the program in the Fiddle Options on the left menu.

4. Click Update to save your changes, and then click Set as Base.

Great! You’re ready to get started!

Planning the pizza parlor program improvements

Here are the three changes that we’ll make to the JavaScript Pizzeria program:

✓ Add a new kind of pizza and charge extra for it.

✓ Add new cities and calculate delivery charges for them.

✓ Display the delivery charge.

✓ Add a birthday special.

Each of these changes requires an `if...else` statement, as well as some small changes to the HTML.

Adding the new item to the menu

The most important new feature at this point is to spruce up the menu. The cook has invented a new kind of pizza that has bacon, arugula, apples, 14 different kinds of cheese, and a corn dog on top. He calls it the Supreme pizza.
The problem is, the Supreme pizza is very expensive to make — mostly because of that corn dog! It’s so difficult to find a gourmet corn dog in Anytown! So, the owner has decided to charge an extra $2 for each Supreme pizza.

Your job is to add the Supreme pizza to the menu and update the price when it gets ordered. Follow these steps to get started:

1. Look in the HTML pane to find the place where the list of pizzas is created.

   It currently looks like this:

   ```html
   <label>What kind of pizzas?
   <select id="typePizza">
     <option value="cheese">Cheese</option>
     <option value="pepperoni">Pepperoni</option>
   </select>
   </label>
   ```

2. Add a new `option` element inside the `select` element to create the Supreme pizza option.

   It should have a value of "supreme", and the label (between `<option>` and `</option>`) should read Supreme.

3. Click Update to save your work, and then test to make sure that Supreme shows up as a new option in the pizza type dropdown list, as shown in Figure 14-2.

4. Find the `calculatePrice()` function.

   It looks like this:

   ```javascript
   function calculatePrice(numPizzas, typePizza) {
       var orderPrice = Number(numPizzas) * 10;
       var extraCharge = 0;

       // calculate extraCharge, if there is one

       orderPrice += extraCharge;
       return orderPrice;
   }
   ```
5. Right below the comment that reads calculate extraCharge, if there is one, type the following if...else statement:

```javascript
if (typePizza === "supreme") {
    extraCharge = Number(numPizzas) * 2;
}
```

This statement checks the typePizza variable to see if the Supreme was selected. If so, it will multiply the number of pizzas by two in order to get the number of dollars to add to the price.

6. Save your work by clicking Update, and then try it out!

If you select the Supreme pizza, you should now see that the total will be equal to $12 times the number of pizzas your ordered, as shown in Figure 14-3.

**Delivering to other cities**
The pizzeria has to grow! But the population of Anytown can only eat so many pizzas, so management has decided to start delivery service to other, carefully chosen, cities.
There’s a catch, though! It’s not profitable to deliver just a single pizza or to deliver to Beverly Hills for free. We’ll need to charge $5 for delivery of orders less than or equal to $10 and for out-of-town delivery.

Follow these steps to put the new rules into place!

1. In the HTML pane, locate the drop-down menu for the delivery city.

   It currently only has one option, Anytown.

2. Add at least two more options to the drop-down.

   When it’s finished, it should look like this:

   ```html
   <label>Where do you live?
   <select id="deliveryCity">
     <option value="Anytown">Anytown</option>
     <option value="Sacramento">Sacramento</option>
     <option value="Your Town">Your Town</option>
   </select>
   </label>
   
   You can replace Your Town with anything you like.
3. Click Update to save your work and see your changes in the Result pane.

4. In the JavaScript pane, find the `calculateDelivery()` function.

   It currently just sets everyone’s `deliveryPrice` to 0.

5. Under the comment that reads `calculate delivery price, if there is one`, insert the following `if...else` statement.

   ```javascript
   if ((deliveryCity === "Anytown") && (orderPrice > 10))
   {
       deliveryPrice = 0;
   } else {
       deliveryPrice = 5;
   }
   ```

6. Save your work by clicking the Update button, and try out the form in the Result pane.

   If you select a city other than Anytown, or your order price is $10, a delivery fee of $5 will now be added to the total.

### Displaying the delivery fee

Next, we need to display the delivery fee above the total, so that people know what they’re getting into.

To display the delivery fee, follow these steps:

1. In the `placeOrder()` function, find the comment that reads `todo: output the delivery price, if there is one`.

2. Below that comment, type the following `if...else` statements:

   ```javascript
   if (deliveryPrice === 0) {
       theOutput += "<p>You get free delivery!</p>";
   } else {
       theOutput += "<p>Your delivery cost is: $" + deliveryPrice;
   }
   ```
This if...else prints out a free delivery message if the deliveryPrice is 0. Otherwise, it prints out the delivery charge.

3. Click Update to save your changes. Then try out the form in the Result pane.

The new free delivery message is shown in Figure 14-4.

Figure 14-4: Telling the customer that they get free delivery is great for marketing!

**Programming the birthday special**

The final change that we’ll make to the program is to give people free delivery on their birthdays.

To program this change, follow these steps:

1. In the HTML pane, add the birthday question to the form by typing this markup after the delivery city question:

   ```html
   <label>Is it your birthday?</label>
   <select id="birthday">
     <option value="yes">Yes</option>
   </select>
   ```
247

Chapter 14: Making Decisions with the If...Else Statement

2. Click Update to save your work and to see your changes in the Result pane.

If your Result pane doesn’t look like Figure 14-5, check your code carefully. You may also need to insert <br/> tags in order to put in the right amount of spacing between questions.

![JavaScript Pizzeria](image)

Figure 14-5: The Result pane with the new birthday question.

3. Add the following to the `placeOrder()` function, below the other `getElementById` statements to get the value of the birthday form field:

   ```javascript
   var birthday = document.getElementById("birthday").value;
   ```

4. Add a third parameter to the `calculateDelivery` function definition for the birthday variable.

   ```javascript
   function calculateDelivery(orderPrice, deliveryCity, birthday)
   ```
5. Add an or operator and a new expression to the if...else statement in the calculateDelivery function to test whether the value of birthday is yes.

```javascript
if (((deliveryCity === "Anytown") && (orderPrice > 10)) ||
(birthday === "yes")) {
```

6. Modify the statement in the placeOrder() function that calls calculateDelivery, to pass birthday as an argument:

```javascript
var deliveryPrice = calculateDelivery(orderPrice, deliveryCity, birthday);
```

7. Click Update to save your work.

Listing 14-5 shows the completed JavaScript code for the JavaScript Pizzeria program.

**Listing 14-5  The Completed JavaScript Pizzeria Program**

```javascript
// listen for button clicks
document.getElementById("placeOrder").addEventListener("click", placeOrder);

/**
 * gets form values
 * calculates prices
 * produces output
 */
function placeOrder() {
  // get form values
  var numPizzas =
    document.getElementById("numPizzas").value;
  var typePizza =
    document.getElementById("typePizza").value;
  var deliveryCity =
    document.getElementById("deliveryCity").value;
  var birthday =
    document.getElementById("birthday").value;

  // get the pizza price
  var orderPrice = calculatePrice(numPizzas, typePizza);
```
// get the delivery price
var deliveryPrice = calculateDelivery(orderPrice,
    deliveryCity, birthday);

// create the output
var theOutput = "<p>Thank you for your order.</p>";

// output the delivery price, if there is one
if (deliveryPrice === 0) {
    theOutput += "<p>You get free delivery!</p>";
} else {
    theOutput += "<p>Your delivery cost is: $" +
        deliveryPrice;
}

theOutput += "<p>Your total is: $" + (orderPrice +
    deliveryPrice);

// display the output
document.getElementById("displayTotal").innerHTML =
    theOutput;
}

/**
 * calculates pizza price
 */
function calculatePrice(numPizzas, typePizza) {
    var orderPrice = Number(numPizzas) * 10;
    var extraCharge = 0;

    // calculate extraCharge, if there is one
    if (typePizza === "supreme") {
        extraCharge = Number(numPizzas) * 2;
    }

    orderPrice += extraCharge;
    return orderPrice;
}

/** (continued) */
Listing 14-5 (continued)

```javascript
* calculates delivery price
*/
function calculateDelivery(orderPrice, deliveryCity, birthday) {
    var deliveryPrice = 0;

    // calculate delivery price, if there is one
    if (((deliveryCity === "Anytown") && (orderPrice > 10)) || (birthday === "yes")) {
        deliveryPrice = 0;
    } else {
        deliveryPrice = 5;
    }
    return deliveryPrice;
}
```

Listing 14-6 shows the completed HTML markup for the JavaScript Pizzeria.

**Listing 14-6   The Final HTML**

```html
<h1>JavaScript Pizzeria</h1>

<div id="orderForm">
    <label>How many pizzas do you want?</label>
    <input type="number" id="numPizzas" />
<br />
    <br />
    <label>What kind of pizzas?</label>
    <select id="typePizza">
        <option value="cheese">Cheese</option>
        <option value="pepperoni">Pepperoni</option>
        <option value="supreme">Supreme</option>
    </select>
<br />
    <br />
    <label>Where do you live?</label>
    <select id="deliveryCity">
```
<option value="Anytown">Anytown</option>
<option value="Sacramento">Sacramento</option>
<option value="Beverly Hills">Beverly Hills</option>
</select>
</label>
<br />
<br />
<label>Is it your birthday?</label>
<select id="birthday">
<option value="yes">Yes</option>
<option value="no">No</option>
</select>
</label>
<br />
<br />
<button type="button" id="placeOrder">Place Order</button>
</div>
<div id="displayTotal"></div>

Figure 14-6 shows the final program’s Result pane after placing our lunch order today.

**JavaScript Pizzeria**

How many pizzas do you want? 5

What kind of pizzas? Cheese

Where do you live? Sacramento

Is it your birthday? Yes

Place Order

Thank you for your order.
You get free delivery!
Your total is: $500

**Figure 14-6**: Our lunch order.
Switch statements are like highways with many different exits. The switch statement chooses among multiple cases by evaluating an expression. These values are like the exits. Each of these values in a switch statement is called a case.

In this chapter, we use a switch statement to write a calendar program that gives you suggestions for things to do, based on what day of the week it is.
Chapter 15: Doing Different Things with Switch

Writing a Switch

The `switch` statement starts with the `switch` keyword, followed by an expression in parentheses and then a series of different options (called cases).

The syntax for the `switch` statement looks like this:

```java
switch (expression) {
    case value1:
        //statements to execute
        break;
    case value2:
        //statements to execute
        break;
    case default:
        //statements to execute
        break;
}
```

You can have as many cases inside a `switch` statement as you’d like. The `switch` statement will try to match the expression to each case until it finds one that matches. Then it runs the statements within that case until it gets to the `break` statement, which causes it to exit the `switch` statement. Each case must end with a `break` statement or semicolon (;). This tells the program to do everything inside the case up until the `break` statement and then stop.

A default case will run if no case matches the result of the expression.

Let’s take a look at an example! The code in Listing 15-1 asks the user to enter his favorite day of the week. The program then uses a `switch` statement to produce a different output based on possible values that the user might enter. If the user enters anything other than a day of the week, the default `switch` statement will run.
Listing 15-1  Produce Different Results for Different Input

```javascript
var myNumber = prompt("Enter your favorite day of the week!");
var theResponse;

switch (myNumber) {
    case "Monday":
        theResponse = "Ack!";
        break;
    case "Tuesday":
        theResponse = "Taco day!";
        break;
    case "Wednesday":
        theResponse = "Halfway there!";
        break;
    case "Thursday":
        theResponse = "It’s the new Friday!";
        break;
    case "Friday":
        theResponse = "TGIF! Yeah!";
        break;
    case "Saturday":
        theResponse = "What a day!";
        break;
    case "Sunday":
        theResponse = "Sunday = Funday!";
        break;
    default:
        theResponse = "I haven’t heard of that one!";
        break;
}
alert (theResponse);
```

Follow these steps to try out this program in JSFiddle:

1. Open JSFiddle and create a new blank project by clicking the JSFiddle logo in the upper left.
2. Type the code from Listing 15-1 into the JavaScript pane.

3. Click the Run link in the top menu.

   A JavaScript prompt appears, asking you to enter your favorite day of the week.

4. Enter a day of the week and click OK.

   The `switch` statement runs. You should see a result based on the value that you entered, as shown in Figure 15-1.

![Figure 15-1: Determining a response by evaluating different cases.](image)

**Building the Activity-of-the-Day Calendar**

If you’re like most people, you sometimes wake up thinking, “What day is it?” The next thing you may think is, “Of all the great things that I could be doing today, what is the one thing that I’m going to do first?” Here’s where most people’s days go wrong. They start off on the wrong foot, or get up on the wrong side of the bed, or set off on the wrong track.

Don’t you wish you had a web page or mobile app that would tell you what day it is and exactly one thing that you should do on that day. Well, wish no more, because you’re about to build it! If you use this program first thing in the morning, your odds of hitting the ground running and having a real whiz-bang kind of a day will be 110 percent greater! Guaranteed!
Using the Activity Calendar program

Before we start building it, let's check out the finished Activity Calendar and see what it does. Follow these steps to run it:


2. Find the program named “Chapter 15 – Activity of the Day” and click its title to open it.

You see the standard JSFiddle editor with the date and time and a button in the Result pane, as shown in Figure 15-2.

3. Click the button labeled “What should I do today?”

A message appears below the button, telling you what you should do, as shown in Figure 15-3. The message is different for every day of the week.

Forking the Activity Calendar program

To get started with the Activity Calendar, follow these steps:

1. Go to our JSFiddle public dashboard at http://jsfiddle.net/user/forkids/fiddles and locate the program named “Chapter 15 – Activity of the Day – Start.”

2. Click the title of the program to open it in the editor.
3. Open the Fiddle Options in the left toolbar and change the name of the program to *Your Name’s Activity Calendar* (replacing *Your Name* with — you guessed it! — your name).

4. Click Update and Set as Base to save your work.

5. Test out the program by pressing the button in the Result pane.

   Nothing happens because the JavaScript hasn’t been completed yet.

Before we show you how to complete the Activity Calendar, let’s talk about an important built-in JavaScript object that we use in this chapter, the `Date` object.

### Using the Date object

The JavaScript `Date` object represents a single moment in time within a JavaScript program. To create an instance of the `Date`
object, use the new keyword and assign the result to a variable name, like this:

```javascript
var myDate = new Date();
```

Creating a new `Date` object in this way will assign the current date to the variable.

To test this out, follow these steps:

1. Open the JavaScript Console in Google Chrome.

2. Type the following into the console, and then press Return (Mac) or Enter (Windows).

```javascript
var myDate = new Date();
```

The console prints out `undefined` to acknowledge that the command has been run.

3. Type the following, and then press Return or Enter.

```javascript
myDate
```

The console prints out the exact date and time that your `Date` object was created.

Like other JavaScript objects we talk about in this book, the `Date` object has a bunch of built-in functions (also known as methods) that you can use to do different things with the `Date` object.

Table 15-1 lists the methods that can be used to get information from the `Date` object. When you use a method to get information from an object, it’s called a **getter method**.

<table>
<thead>
<tr>
<th>Method</th>
<th>What It Does</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDate()</code></td>
<td>Gets the day of the month (1–31).</td>
</tr>
<tr>
<td><code>getDay()</code></td>
<td>Gets the day of the week as a number (0–6).</td>
</tr>
<tr>
<td><code>getFullYear()</code></td>
<td>Gets the year (yyyy).</td>
</tr>
</tbody>
</table>

Table 15-1 Getter Methods of the Date Object
To use the getter methods of the `Date` object, attach them to an instance of the object using a period (or dot).

For example, after you’ve created a variable to hold a `Date` object in the Chrome Developer Console, follow these steps to use some of the getter methods.

1. Get the day of the week, as a number, with this statement:

   ```javascript
   myDate.getDay()
   ```

   The JavaScript Console responds with a number from 0 to 6, where 0 is equal to Sunday and 6 is equal to Saturday.

2. Get the day of the month, as a number, with this statement:

   ```javascript
   myDate.getDate();
   ```

3. Get the month of the year, as a number, with this statement:

   ```javascript
   myDate.getMonth();
   ```

Notice that both `getMonth` and `getDay` start with 0. In JavaScript, the number for January is 0.

The numbers for both `getDate` and `getFullYear`, on the other hand, are returned how you would expect them. The second day of May is returned as the number 2, and the year 2020 is returned as 2020.

<table>
<thead>
<tr>
<th>Method</th>
<th>What It Does</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getHours()</code></td>
<td>Gets the hour (0–23).</td>
</tr>
<tr>
<td><code>getMilliseconds()</code></td>
<td>Gets the fraction of a second (0–999).</td>
</tr>
<tr>
<td><code>getMonth()</code></td>
<td>Gets the month (0–11).</td>
</tr>
<tr>
<td><code>getSeconds()</code></td>
<td>Gets the seconds (0–59).</td>
</tr>
<tr>
<td><code>getTime()</code></td>
<td>Gets the time, in Unix time (milliseconds since January 1, 1970).</td>
</tr>
</tbody>
</table>
In addition to being able to get values from `Date` objects, JavaScript also allows you to set values. Table 15-2 lists the methods that can be used to set information in a `Date` object. When you use a method to set information in an object, it’s called a *setter method*.

<table>
<thead>
<tr>
<th>Method</th>
<th>What It Does</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setDate()</code></td>
<td>Sets the day of the month (1–31).</td>
</tr>
<tr>
<td><code>setDay()</code></td>
<td>Sets the day of the week as a number (0–6).</td>
</tr>
<tr>
<td><code>setFullYear()</code></td>
<td>Sets the year (yyyy).</td>
</tr>
<tr>
<td><code>setHours()</code></td>
<td>Sets the hour (0–23).</td>
</tr>
<tr>
<td><code>setMilliseconds()</code></td>
<td>Sets the fraction of a second (0–999).</td>
</tr>
<tr>
<td><code>setMonth()</code></td>
<td>Sets the month (0–11).</td>
</tr>
<tr>
<td><code>setSeconds()</code></td>
<td>Sets the seconds (0–59).</td>
</tr>
<tr>
<td><code>setTime()</code></td>
<td>Sets the time, in Unix time (milliseconds since January 1, 1970).</td>
</tr>
</tbody>
</table>

To try using some of the setter methods in the JavaScript Console, follow these steps:

1. Create a new `Date` object using this statement:
   ```javascript
   var myNewDate = new Date();
   ```

2. Find out the initial value of the `Date` object by typing its name into the console:
   ```javascript
   myNewDate
   ```
   The console prints out the current value of the `myNewDate` object as a string.

3. Change the month to August using this statement:
   ```javascript
   myNewDate.setMonth(7);
   ```
The console returns a giant number. This number is the new value of the `myNewDate` object in Unix time. Unix time is how JavaScript stores dates internally. It’s equal to the number of milliseconds (thousandths of a second) since January 1, 1970.

4. Type the name of the object to see the new date as a human-readable string.

```javascript
myNewDate
```

Now that you understand how to use the `Date` object, let’s combine it with a `switch` statement to build the Activity Calendar program.

**Building the Activity Calendar program**

When you first load the starting point program for this chapter, the JavaScript pane contains the starting code and the comments that describe what the program will do. Listing 15-2 shows what our starting point code should look like.

**Listing 15-2  The Starting JavaScript for the Activity Calendar**

```javascript
var todayDate = document.getElementById("todaysdate");
var todoButton = document.getElementById("whattodo");

// add a listener to the whattodo button
todoButton.addEventListener("click", displayActivity);

// create a new Date object
var d = new Date();

// call the displayDate() function
displayDate();

function displayDate() {
    // todo: display the current date in the todaysdate div
}

function displayActivity() {
    // todo: find out the day of the week
}(continued)
```
Listing 15-2 *(continued)*

```javascript
/* todo: set a variable, called youShould, with a different string based on what day of the week it is. */

// todo: output the value of youShould into the thingToDo div
```

Let’s go over what the program does so far. Try following along in the code and pick out which statements do each of the following items:

- Define two new variables to hold references to HTML elements we’ll be working with in the program.
- Create an event listener to handle clicks on the button.
- Create an instance of the **Date** object to hold the current date.
- Call a function that will display the current date.

After these things have been done, the program just sits and waits for someone to click the What To Do button. When it detects a click of the button, it runs the function associated with the event listener, `displayActivity()`.

Your job is to finish the two functions in this program.

Before moving on to the step-by-step instructions, can you figure out how to do them yourself? Give it a try and when you’re ready, move on and we’ll walk you through how it works!

1. Find the `displayDate()` function and add this statement just below the comment:

   ```javascript
todayDate.innerHTML = d;
```

   This statement sets the **innerHTML** property of the **div** element referenced by the `todayDate` variable to the value of `d` (which we created as a **Date** object).
2. Click Update to see the date displayed in the Result pane.

3. To make the date displayed in the Result pane easier to read, change it to the following:

   ```javascript
   todayDate.innerHTML = d.toDateString();
   ```

   Now when you run it, it will display a shorter date, with just the day of the week, the month, the date, and the year.

4. Find the function called `displayActivity()` and add a statement inside of it to get the current day of the week from the `d` variable.

   ```javascript
   var dayOfWeek = d.getDay();
   ```

5. Initialize a variable to hold the string that will contain the message for each day.

   ```javascript
   var youShould;
   ```

6. Write the condition part of a `switch` statement that will evaluate the value of the `dayOfWeek` variable, followed by an opening curly bracket:

   ```javascript
   switch (dayOfWeek) {
   ```

7. Write the first case, which will be for the value 0, or Sunday:

   ```javascript
   case 0:
   ```

8. Write a statement to set the value of `youShould` when it’s Sunday, for example:

   ```javascript
   youShould = "Take it easy. You’ve earned it!";
   ```

9. Write a break statement to end the `switch` statement when this case is true.

   ```javascript
   break;
   ```

10. Write a case for each of the other days of the week.
11. After you’ve done the case for day 6, write a default case that should run in the (very remote) chance that the day of the week is something other than a number from 0 to 6.

    default:
        youShould = "Hmm. Something has gone wrong.";
        break;

12. Finish the switch statement with a closed curly bracket on a line by itself.

    }

13. Under the switch statement, write a statement to output the youShould string into the div with an ID of thingToDo.

    document.getElementById("thingToDo").innerHTML = youShould

When all the statements are written, the JavaScript pane should look like Listing 15-3.

**Listing 15-3  The Finished Program**

```javascript
var todayDate = document.getElementById("todaysdate");
var todoButton = document.getElementById("whattodo");

// add a listener to the whattodo button
todoButton.addEventListener("click", displayActivity);

// create a new Date object
var d = new Date();

// call the displayDate() function
displayDate();

function displayDate() {
    todayDate.innerHTML = d.toDateString();
}

function displayActivity() {
    // find out the day of the week
    var dayOfWeek = d.getDay();
```
Chapter 15: Doing Different Things with Switch

/* set a variable, called youShould, with a different
string based on what day of the week it is */

var youShould;

switch (dayOfWeek) {
    case 0:
        youShould = "Take it easy. You’ve earned it."
        break;
    case 1:
        youShould = "Gotta do what ya gotta do!"
        break;
    case 2:
        youShould = "Take time to smell the roses!"
        break;
    case 3:
        youShould = "Don’t forget to eat breakfast!"
        break;
    case 4:
        youShould = "Learn something new today!"
        break;
    case 5:
        youShould = "Make a list of things you like to do."
        break;
    case 6:
        youShould = "Do one thing from your list of things
        you like to do."
        break;
    default:
        youShould = "Hmm. Something has gone wrong."
        break;
}

// output the value of youShould into the thingToDo div
document.getElementById("thingToDo").innerHTML =
    youShould;
When it’s done, try running it and pressing the button. The output in the Result pane should look like Figure 15-4.

![JS Activity of the Day](image)

Figure 15-4: The output of the Activity Calendar program.

Now that you have your basic Activity Calendar, here are some ideas for making it even more awesome:

- Write your own activities, if you haven’t already!
- Make it have a different activity for every day of the month instead of every day of the week.
- Have multiple messages — one for the day of the week, one for the day of the month, one for the month, and one for the year.
- Write CSS styles to customize the look of your Activity Calendar.

Can you think of other ideas for improving the Activity Calendar?
Imagine you’re climbing a tree. If you climb up one branch, you’ll see certain things, such as a bird’s nest or a balloon that got stuck up there during your birthday party. If you climb a different branch, you might see other things, like the neighbor’s garage. In JavaScript, the technique of using `if...else` or `switch` statements to choose between two or more paths is called *branching*.

In this chapter, we use branching to write a choose-your-own-adventure game that asks for user input at key moments to change the story.

You are the captain of a spaceship named "The Flying Hippo." One day, you're working on tuning up your ship’s engines when you get an urgent message on your space phone:

"Captain, one of our Mars robots is sick. We need you to go to Mars immediately and retrieve it so that we can fix it and download the results of its important experiments."

You remember that you’re supposed to go to a meeting of the Space Scouts tonight, and you were really looking forward to it. But, on the other hand, the other Space Scouts would understand that this mission is very important.

What do you do? Go to Mars, or stay home?

Go to Mars, or stay home?

Enter your answer:  [Input field]  Go!
Planning the Story

Any good story needs a plot. The plot is the outline of events that happen over the course of the story. When writing a story where the user’s input influences the plot, the writer needs to pay close attention to managing the different plot lines. Each plot line has the same beginning, but the middle and ending are different based on input from the user.

Creating a flow chart

Considering all of the different options — and planning for each possibility — in a branching program is a valuable skill to have as a programmer.

We’ll begin by creating a simple story that poses a question. This creates two branches. Each of these branches will have a question that creates two more branches. Eventually, every choice will lead back to one of two possible endings.

A useful tool for visualizing branches of a story or of a program is a flowchart. Figure 16-1 shows a flow chart for our interactive story.

The next step in developing our story and program is to fill in the plot with some details.

Figure 16-1: A flow chart showing the basic outline of the story.
Writing the story

Our story takes place on a spaceship in the not-too-distant future. You’re the captain of the ship, and your mission is to fly to Mars to pick up an old robot that has stopped functioning so that it, and its valuable experiments, can be returned to Earth and studied.

The launch of the ship goes perfectly, but one week into the 260-day voyage, you discover that your cat has stowed away on your ship. Unless something changes, you might not have enough food for the both of you. You need to decide whether to turn your ship around, or keep going for Mars and hope for the best.

If you turn the ship around, the mission ends with the cat being returned to Earth and your boss yelling at you because you don’t have the Mars robot.

If you continue toward Mars, you get very nervous about your food situation, but you feed the cat half of your meals each day, because you’re a good person. You reach Mars and discover that the last person to visit the Mars robot left a large cooler full of delicious sandwiches. You pack them into the spaceship along with the robot and head home, where you’re greeted as a hero.

Playing the Game

The most fun part about a game like this one, in which your choices decide how the story unfolds, is in exploring all the different possibilities. Interactive stories tend to be short, because reading and writing each of the different possibilities takes much more effort and time than writing or reading a story with just a single storyline.

To see how the Martian Rescue! game works, follow these steps:

1. Go to our public dashboard in JSFiddle, at http://jsfiddle.net/user/forkids/fiddles.

You see a list of all the projects that we’ve created for this book.
2. Open the “Chapter 16 – Martian Rescue!” project by clicking its title in the public dashboard.

The project opens, as shown in Figure 16-2, and the Result pane asks you to answer the first question.

3. Enter your answer to the first question into the input field, and click the Go button.

Figure 16-2: The Martian Rescue! program.

Depending on how you respond, new text displays in the Result pane and you’re asked another question.

4. Respond to the new question.

Once again, the program will respond to your question by continuing the story.
5. Answer questions and view the results until the game ends.

6. Click Run in the JSFiddle top menu to start the program over.

   The text from the first time you ran the program will be removed from the Result pane and you see the first question again.

7. Play the game again, but answer questions differently this time in order to see the alternate messages and story ending.

Now that you’ve seen how the game works, move on to the next section, where we show you how to program it, and how you can customize it with your own stories!

**Forking the Code**

We’ve created a starting point for the program, with all the HTML and CSS necessary, but only part of the JavaScript written. Follow these steps to make a copy of the starter program in your own JSFiddle account.

1. Log into JSFiddle if you aren’t already logged in.

2. Go to our JSFiddle Public Dashboard at http://jsfiddle.net/user/forkids/fiddles and find the “Chapter 16 – Martian Rescue – Start” project.

3. Open the starter project by clicking the title.

4. Click the Fork link in the top menu to save a copy in your own JSFiddle account.

5. Change the name of the project to “(Your Name)’s Martian Rescue.”

6. Click Update and Set as Base in the top menu to save your work.
Tiptoeing through the HTML and CSS

The HTML and CSS for Martian Rescue! are finished in the starter program. Let’s take a look at them now before we move on to finishing the JavaScript. We’ll start with the HTML pane.

The HTML is made up of two parts, separated by an HTML comment. The first part of the HTML creates the top section, where the story will be displayed, and the bottom section, where questions and answers will be displayed.

Listing 16-1 shows this top section of the HTML.

Listing 16-1  The Beginning of the HTML

```html
<div id="story"></div>
<div id="siteFooter">
  <div id="question"></div>
  <div id="answer">Enter your answer:
    <input type="text" id="yourAnswer" />
    <button type="button" id="submit">Go!</button>
  </div>
</div>
</div>
```

Everything that you see in the Result pane when the program first starts is the result of these lines of HTML, combined with the CSS.

Figure 16-3 shows the Result pane for the Martian Rescue! project before the JavaScript has been completed.

Notice that there are three differently colored sections:

- The top, dark gray part where the story will display
- The light gray part, where the question will display
- The white part, where the form and user input area will display
If you look at the HTML and compare it with the Result pane, however, you’ll notice that something’s not right. The HTML clearly contains an input field and button, but those aren’t displaying in the Result pane. Why?

**Turning off elements with display:none**

Because we only want to show the input field and button when we’re asking a question, we’ve hidden the user input field and button using CSS.

Listing 16-2 shows the complete CSS for Martian Rescue!

**Listing 16-2  The CSS for Martian Rescue!**

```css
* {
    margin: 0px;
}
html, body {
    font-family: Arial, sans-serif;
    height: 100%;
    overflow:hidden;
}
```

(continued)
If you look again at the HTML, you’ll see that the input field and the button are inside of a div element with an ID of answer.

To see the styles applied to this div element, find the style rules with the ID selector of answer in the CSS pane.
The first five properties in this style rule set the background color, the text color, the padding, and the text alignment of the element. The last one, however, sets the display property, like this:

```html
display: none;
```

When `display` is set to `none`, it turns off the display of the element — in other words, the element just doesn't display.

Often, programmers will use `display: none` in order to hide elements that they want to hide or show using JavaScript. When you want to display an element that’s been hidden with CSS you can use JavaScript to change the value of the display property to any of its visible values, like this:

```javascript
document.getElementById("answer").style.display = "block";
```

## Looking at (or not looking at) the story parts

Underneath the basic HTML that creates the three sections of the Martian Rescue! program, you’ll see several more `div` elements. Each of these contains text that may become part of the story, based on the choices that you make within the program.

Listing 16-3 shows the first one of these `div` elements.

### Listing 16-3  The First `storyPart` div

```html
<div class="storyPart" id="answer01">
  <p>You are the captain of a spaceship named "The Flying Hippo." One day, you’re working on tuning up your ship’s engines when you get an urgent message on your space phone:</p>
  <p>"Captain, one of our Mars robots is sick. We need you to go to Mars immediately and retrieve it so that we can fix it and download the results of its important experiments."</p>
</div>
```

(continued)
Listing 16-3 (continued)

<p>You remember that you’re supposed to go to a meeting of the Space Scouts tonight, and you were really looking forward to it. But, on the other hand, the other Space Scouts would understand that this mission is very important.</p>

<p>What do you do? Go to Mars, or stay home?</p>

Each of the parts of the story has a class attribute set to storyPart and a unique ID attribute.

The same class attribute value can be applied to multiple elements in an HTML document, but each ID attribute must be unique.

Can you guess why none of the div elements with class attribute values of storyPart are displaying in the Result pane when you open the program? If you guessed that it’s because their CSS display properties are set to none, you’re right!

Take a look at the CSS pane. Find the .storyPart selector and notice that it has only one style rule inside of it: display: none;.

By setting the display property to none for every element with the storyPart class, we’ve hidden them all. Then, when the time comes, we can display the correct part of the story using JavaScript.

That just about covers everything that you need to understand about the CSS and HTML. Now let’s talk about the JavaScript.

Writing the Martian Rescue! JavaScript

When you first open the starter program for Martian Rescue!, the JavaScript pane contains the code shown in Listing 16-4.

Let’s step through this skeletal code and finish it up!
Listing 16-4  The Starter JavaScript for Martian Rescue!

    // declare variables
    var story = document.getElementById("story");
    var siteFooter = document.getElementById("siteFooter");
    var question = document.getElementById("question");
    var answer = document.getElementById("answer");
    var yourAnswer = document.getElementById("yourAnswer");
    var submit = document.getElementById("submit");

    // todo: make an empty array called answers
    /* todo: listen for clicks on the submit button and call
     the getAnswer() function when they happen. */

    // todo: call the function to ask the first question

    /* askQuestion() asks a question, based on the number
     passed to it */
    function askQuestion(questionNumber) {
    }

    /* getAnswer() gets the answer from the text field and
     pushes it into the answers array, then calls
     the continueStory function */
    function getAnswer() {
    }

    /* continueStory() displays part of the story or an error
     based on the value of an item in the answers
     array */
    function continueStory(answerNumber) {
    }

    /* theEnd() ends the story and hides the input field */
    function theEnd() {
    }
Creating element shortcuts
The first section of the code defines some global variables that we need to use throughout the rest of the program. The ones that are already finished for you are variables that create references to HTML elements. You’ll use these variables as shortcuts to save you from having to type `document.getElementById` over and over again in the program.

When you use the following statement, it makes it possible for you to use `myElement` in place of `document.getElementById("myElement")`:

```javascript
var myElement = document.getElementById("myElement");
```

This can make your code much easier to type, and must easier to read later on, too.

Creating an empty array
After the element shortcuts is a comment telling you to create an empty array.

Recall from Chapter 11 that the way to create an empty array (one with no values stored in it) is to set the value of a variable to square brackets with nothing between them. To create an empty array called `answers`, type the following code on the next line after the comment telling you to create it.

```javascript
var answers = [];
```

Now you have an array with no elements. Because you created this array outside of all the functions in your program, this array will be usable anywhere in the program.

A variable that can be used anywhere inside a program is called a `global variable`. 
Creating an event listener

The next to-do item in the JavaScript pane says to listen for clicks on the submit button. The word *listen* is a clue for how to write this code. Can you guess what JavaScript method we’ll use to listen for clicks? If you guessed that we’ll use `addEventListener`, that’s correct!

To write the event handler, follow these steps:

1. Under the comment telling you to listen for clicks on the submit button, first type the shortcut to the submit button, followed by a period:

   ```javascript
   submit.
   ```

2. Right after the period, type the `addEventListener` keyword, followed by parentheses.

   ```javascript
   submit.addEventListener()
   ```

3. Inside the parentheses after `addEventListener()`, pass the two arguments: the event you want to listen for and the function that will be called when the event happens.

   ```javascript
   submit.addEventListener("click",getAnswer);
   ```

Great! Now you have the array that will be used to store the user’s responses, and you have an event handler set up for the button. But, if you run the program now, you’ll see that it doesn’t appear to do anything that it couldn’t do before. It just shows the same three blank sections in the Result pane.

In order to make this program do something useful, we need to kick off some sort of action. In the Martian Rescue! program, we start the action with a call to the `askQuestion()` function, as indicated in the next to-do item in the JavaScript pane.
Calling the askQuestion() function

The askQuestion() function takes a single parameter, the questionNumber. The questionNumber is the number of the question to ask the user. We’ll call the first question question #0.

To call the function and ask the first question, type this after the comment asking you to:

```javascript
askQuestion(0);
```

As you complete to-do items, it’s helpful to remove the word todo so that you know that the item is done.

Congratulations, you’ve now completed the parts of the program that aren’t inside of functions. The beginning of your JavaScript should now look like Listing 16-5.

**Listing 16-5   The Beginning of the JavaScript**

```javascript
// declare variables
var story = document.getElementById("story");
var siteFooter = document.getElementById("siteFooter");
var question = document.getElementById("question");
var answer = document.getElementById("answer");
var yourAnswer = document.getElementById("yourAnswer");
var submit = document.getElementById("submit");
var answers = [];

/* listen for clicks on the submit button and call the getAnswer() function when they happen */

submit.addEventListener("click", getAnswer);

// call the function to ask the first question
askQuestion(0);
```

If you run the program now, you’ll see that it still doesn’t do anything in the Result window.
To make the program actually do something, we need to finish the functions.

**Writing the functions**

The first function that we’ll work on is the one that prompts users to answer questions, the `askQuestion()` function.

To complete the `askQuestion()` function, follow these steps.

1. Change the value of the display property of the answer div so that the input field and button appear, using this code:

   ```javascript
   answer.style.display = "block";
   ```

   This statement causes the form to show up in the Result pane.

2. Change the length of the answers array to match the number of the question being asked, using this code:

   ```javascript
   answers.length = questionNumber;
   ```

   This statement uses the argument passed to the function to set the `length` property of the `answers` array. We do this so that answers are always stored within their question in the array. In the event that a user enters an invalid value, such as “Maybe,” when a question asks for a “Yes” or a “No,” setting the length of an array to the number of the question will cause the invalid values to be overridden when the question is asked again.

   When you set the length property of an array to a number that’s less than the actual length of the array, elements after the new length will be deleted.

3. Write a `switch` statement that will use the argument passed into the function to determine which question to answer.
Here’s the code for the switch statement:

```javascript
switch (questionNumber) {
  case 0:
    question.innerHTML = "Are you ready to play?";
    break;
  case 1:
    question.innerHTML = "Go to Mars, or stay home?";
    break;
  case 2:
    question.innerHTML = "Risk it, or go home.";
    break;
  default:
    break;
}
```

It’s not technically necessary to use a break statement after the default clause of a switch statement, since the switch will exit after the default clause anyway. It’s also not necessary to specify a default clause at all if it doesn’t do anything, as in this case. But we think it’s still a good idea to do both of these things for consistency.

4. After the switch statement, end the function with a closing curly brace, like this:

```javascript
}
```

5. Save your work by clicking the Update link.

The finished askQuestion() function is shown in Listing 16-6.

**Listing 16-6  The Finished askQuestion() Function**

```javascript
/* askQuestion() asks a question, based on the number passed to it. */
function askQuestion(questionNumber) {
  answer.style.display = "block";

  //make sure the array is the right length
  answers.length = questionNumber;
```
Chapter 16: Choose Your Own Adventure

With the `askQuestion()` function finishes, the Result pane now does something. You’ll see that the first question displays, and the input field and button display beneath it, as shown in Figure 16-4.

At this point, however, you can put any value into the input field and press the Go button and nothing will happen. In order to make the game work, we need to write the next two functions.

```javascript
switch (questionNumber) {
    case 0:
        question.innerHTML = "Are you ready to play?";
        break;
    case 1:
        question.innerHTML = "Go to Mars, or stay home?";
        break;
    case 2:
        question.innerHTML = "Risk it, or go home.";
        break;
    default:
        break;
}
```

With the `askQuestion()` function finishes, the Result pane now does something. You’ll see that the first question displays, and the input field and button display beneath it, as shown in Figure 16-4.

Follow these steps to write the `getAnswer()` function.

1. Get the value from the input field and convert it to uppercase letters with this statement:

   ```javascript
   cleanInput = yourAnswer.value.toUpperCase();
   ```
2. Use the `push` array method to add the user’s answer as a new element at the end of the `answers` array, like this:

```javascript
answers.push(cleanInput);
```

3. Reset the input field, clearing the current value out of it, like this:

```javascript
yourAnswer.value = "";
```

4. Call the `continueStory()` function, passing it the number of the last element in the `answers` array, using this code:

```javascript
continueStory(answers.length - 1);
```

Because arrays start counting at 0, the `length` (number of elements in the array) will always be one more than the number of the last element, which is why we subtract 1 from the length above.

5. Finish the `getAnswer()` function with a closing curly bracket.

```javascript
}
```

The finished `getAnswer()` function is shown in Listing 16-7.

Move on to the next section to write the `continueStory()` function.

**Listing 16-7  The Finished getAnswer() Function**

```javascript
/* getAnswer() gets the answer from the text field and pushes it into the answers array, then calls the continueStory function */
function getAnswer() {
    cleanInput = yourAnswer.value.toUpperCase();
    answers.push(cleanInput);
    yourAnswer.value = "";
    continueStory(answers.length - 1);
}
```


Writing `continueStory()`

The `continueStory()` function uses `if...else` statements to determine whether the user entered a valid value and then to show the correct part of the story based on that input.

Follow these steps to write `continueStory()`:

1. Write a `switch` statement to use the value of the argument to find out what question is being asked.

The basic switch statement, without the `if...else` statements for each question, looks like this:

```javascript
switch (answerNumber) {
    case 0:
        // insert statements
        break;
    case 1:
        // insert statements
        break;
    case 2:
        // insert statements
        break;
    default:
        // insert statements
        break;
}
```

2. Write `if...else` statements for the first question in the game: “Are you ready to play?”

When it’s finished, the first case in the `switch` statement should look like this:

```javascript
case 0:

    if (answers[0] === "YES") {
        story.innerHTML = document.getElementById("answer01").innerHTML;
        askQuestion(1);
    }
```

Let's step through this code line-by-line:

`case 0:`

This line says that if the user responded to the first question, run the following statements.

```javascript
if (answers[0] === "YES") {

    story.innerHTML = document.getElementById("answer01").innerHTML;

    askQuestion(0);

} else if (answers[0] === "NO") {

    story.innerHTML = document.getElementById("answer02").innerHTML;
    askQuestion(0);

} else {

    story.innerHTML = document.getElementById("err0").innerHTML;
    askQuestion(0);

}

break;
```

This line is saying that if the first element in the array (which corresponds to the first question) is set to "YES", run the following statements. Remember that in the `getAnswer()` function, we converted the user's input to uppercase before pushing it into the array. So, the user can enter `yes`, `Yes`, or even `yeS` and this statement will still be true.

```javascript
story.innerHTML = document.getElementById("answer01").innerHTML;
```

This statement gets the HTML from inside the `div` element with an ID of `answer01` and overwrites the contents of the `div` with the ID of `story`. If you locate the `div` with the ID equal to `answer01` in the HTML pane, you'll see that it's the beginning of the story.
When you answer “Yes” to the question “Are you ready to play?,” the first part of the story will display.

```
    askQuestion(1);
```

This statement calls the `askQuestion` function and tells it to ask question #1. This causes the `askQuestion` function to ask “Go to Mars, or stay home?”

```
} else if (answers[0] === "NO") {

    if the user didn’t answer “yes,” the else clause will run. But here we put another if statement inside of the else clause so that we can test for a value of "NO", but only if the answer wasn’t "YES".

    story.innerHTML = document.getElementById("answer02").innerHTML;

    If the answer is "NO", set the story div’s `innerHTML` equal to the appropriate message.

    askQuestion(0);

    Because they said they aren’t ready to play, ask them the first question again until they are ready.

} else {

    Do the following if the user didn’t enter Yes or No.

    story.innerHTML = document.getElementById("err0").innerHTML;

    Set the value of the story div to an error message, telling them to enter either Yes or No.

    askQuestion(0);

    Ask the first question again, and hopefully they’ll provide a good answer this time!
3. Write the cases for the other two questions in the game, like this:

```javascript
    case 1:
        if (answers[1] === "GO TO MARS") {
            story.innerHTML = document.getElementById("answer11").innerHTML;
            askQuestion(2);
        } else if (answers[1] === "STAY HOME") {
            story.innerHTML = document.getElementById("answer12").innerHTML;
            theEnd();
        } else {
            story.innerHTML = document.getElementById("err1").innerHTML;
            askQuestion(1);
        }
        break;
    case 2:
        if (answers[2] === "RISK IT") {
            story.innerHTML = document.getElementById("answer21").innerHTML;
            theEnd();
        } else if (answers[2] === "GO HOME") {
            story.innerHTML = document.getElementById("answer22").innerHTML;
            theEnd();
        } else {
            story.innerHTML = document.getElementById("err2").innerHTML;
            askQuestion(2);
        }
        break;
    default:
        story.innerHTML = "The story is over!";
        break;
```

4. Finish the function with a closing curly bracket.

```javascript
    }
```

5. Save your work by clicking the Update link.

The completed `continueStory()` function is shown in Listing 16-8.
/* continueStory() displays part of the story or an error based on the value of an item in the answers array. */

function continueStory(answerNumber) {
    switch (answerNumber) {
        case 0:
            if (answers[0] === "YES") {
                story.innerHTML = document.getElementById("answer01").innerHTML;
                askQuestion(1);
            } else if (answers[0] === "NO") {
                story.innerHTML = document.getElementById("answer02").innerHTML;
                askQuestion(0);
            } else {
                story.innerHTML = document.getElementById("err0").innerHTML;
                askQuestion(0);
            }
            break;
        case 1:
            if (answers[1] === "GO TO MARS") {
                story.innerHTML = document.getElementById("answer11").innerHTML;
                askQuestion(2);
            } else if (answers[1] === "STAY HOME") {
                story.innerHTML = document.getElementById("answer12").innerHTML;
                theEnd();
            } else {
                story.innerHTML = document.getElementById("err1").innerHTML;
                askQuestion(1);
            }
            break;
    }
}(continued)
The final function we need to write is the function that runs when the story comes to the end.

**Writing theEnd()**

The function called theEnd() prints out the final line of the story and hides the contents of the answer div — including the question as well as the input field and button. To write the theEnd() function, follow these steps:

1. Type the following statement in the function body of theEnd() to print out “The End” after the last text in the story div:

   ```javascript
   story.innerHTML += "<p>The End.</p>";
   ```
2. Erase the last question asked from the question div, using this statement:

```javascript
question.innerHTML = "";
```

3. Hide the input field and button with this statement:

```javascript
answer.style.display = "none";
```

4. Click Update to save your work.

The final `theEnd()` function is shown in Listing 16-9.

**Listing 16-9  The theEnd() Function**

```javascript
/* theEnd() ends the story and hides the input field */
function theEnd() {
  story.innerHTML += "<p>The End.</p>";
  question.innerHTML = "";
  answer.style.display = "none";
}
```

That completes the Martian Rescue! program. Click Update and Set as Base and then try it out!

If you did everything correctly, you should be able to play through the game in any way that you want. Figure 16-5 shows the Result pane of a game in progress.

Do you have ideas for other interactive stories? Can you think of other ways to modify our story to make it longer, more exciting, or funnier? Experiment with the program and share your work with your friends or with us online! We’re looking forward to seeing what you come up with!
You are the captain of a spaceship named "The Flying Hippo." One day, you're working on tuning up your ship's engines when you get an urgent message on your space phone:

"Captain, one of our Mars robots is sick. We need you to go to Mars immediately and retrieve it so that we can fix it and download the results of its important experiments."

You remember that you're supposed to go to a meeting of the Space Scouts tonight, and you were really looking forward to it. But, on the other hand, the other Space Scouts would understand that this mission is very important.

What do you do? Go to Mars, or stay home?

Go to Mars, or stay home?

Enter your answer: Go!

Figure 16-5: Playing Martian Rescue!
Part VI

Loops

Lunch Game!

You get a weekly allowance of $20 to buy lunch. Sandwiches are always between $1 and $5, but you never know the price until you get to school.

Your goal is to be able to buy lunch every day of the week.

How many sandwiches do you want per day?

2  Place Order

On day 1, sandwiches are: $2.89. You have $14.22 left.
On day 2, sandwiches are: $2.12. You have $9.98 left.
On day 3, sandwiches are: $3.94. You have $2.10 left.
Today, sandwiches are: $2.65. You don’t have enough money. Maybe your sister will give you some of her sandwich.
You bought 3 lunches this week.
In this part . . .

- What’s This Loop For? ............................................. 295
- Using While Loops .................................................. 309
- Building a Lemonade Stand ...................................... 326

For information on advanced looping with JavaScript, go to www.dummies.com/extras/javascriptforkids.
What’s This Loop For?

*for loops are useful* for when you know in advance how many times you need to do something. You can use a *for* loop to count to 10, or to count to 1,000,000. It’s all the same to JavaScript!

In this chapter, we look at one of the most popular types of loops in JavaScript: the *for* loop. We use *for* loops to create our own weather forecasting app.

<table>
<thead>
<tr>
<th>Forecast for Monday:</th>
<th>Forecast for Tuesday:</th>
<th>Forecast for Wednesday:</th>
<th>Forecast for Thursday:</th>
<th>Forecast for Friday:</th>
</tr>
</thead>
</table>
Introducing the for Loop

The **for** loop is the most commonly used type of loop in JavaScript. Here's a sample **for** loop that prints out the words **Hello, JavaScript!** 500 times to the JavaScript console.

```javascript
for (var counter = 0; counter < 500; counter++) {
    console.log(counter + ": Hello, JavaScript!");
}
```

Figure 17-1 shows what this code looks like when it’s run in the JavaScript console.

![Figure 17-1: Saying “Hello, JavaScript!” 500 times.](image)

This isn’t the most exciting use for a loop, but you can certainly see that it’s easier to use a loop than it would be to type out 500 `console.log` statements!

Let’s take a closer look at how to write **for** loops.

**The three parts of the for loop**

The **for** loop is made up of three different statements:

- **Initialization**: The initialization statement declares a variable that the loop will use to keep track of how long it has been looping.

- **Condition**: A Boolean expression to be evaluated with each iteration of the loop.
Final expression: An expression to be evaluated after each loop iteration.

Here’s how our Hello, JavaScript loop works:

1. A new variable — in this case, `counter` — is initiated with the value of 0.

2. A test is done to check whether `counter` is less than 500.

   If it is, then the statements inside the loop are run. In this case, the `console.log` statement will print out Hello, JavaScript!

3. The final expression increments (adds 1 to) the counter variable.

4. The condition statement is evaluated again to determine whether `counter` is still less than 500.

   If so, the statements inside the loop are executed again.

5. The final expression increments the counter again.

6. Steps 2 and 3 keep running until the condition (`counter < 500`) is no longer true.

Writing and using for loops

One very useful thing about `for` loops is that you can use the counter inside the `for` loop to change the output of the statements inside the loop.

The most basic example of this technique is to use the `for` loop to count. Listing 17-1 shows an app that displays a countdown in alert statements.

Listing 17-1  JavaScript Countdown

```javascript
for (var i = 10; i > 0; i--) {
    alert (i);
}
alert ("Blast Off!");
```
Follow these steps to test out this program:

1. Go to http://jsfiddle.net and log in if you’re not already.

2. Open a new program by clicking the JSFiddle logo.

3. Type the code in Listing 17-1 into the JavaScript pane.

4. Click the Run button to run the program.

An alert box appears with the number 10 in it. When you click OK in the alert box, a new alert with the number 9 in it appears. The alerts appear like this until the value of the counter variable (i) is no longer greater than 0. At that point, the loop will exit and a final alert will appear, containing the phrase “Blast Off!”

Counting is a great use for for loops, but there’s an even better and more useful thing that you can do with for loops: looping through arrays.

Listing 17-2 shows a program that creates an array containing names of people. The for loop outputs the same sentence with each of the names inserted into it.

Listing 17-2  Outputting Array Values with for

```javascript
var myFriends = ["Agatha", "Agnes", "Jermaine", "Jack"];

for (var i = 0; i < myFriends.length; i++) {
    alert(myFriends[i] + " is my friend.");
}
```

To use a for loop to output all the values in an array, you just use the length property of the array to find out how many elements are in the array and you use that to perform the loop that same number of times.

Then, inside the loop, you use the counter variable (i, in this case) to output the corresponding array element.
Chapter 17: What’s This Loop For?

When you know how to output array elements, you can do all sorts of cool things with `for` loops. For example, in the next section, we use a `for` loop to provide a randomized five-day weather forecast!

Random Weather Forecasting

Welcome to Anytown, USA! We have a saying here: “If you don’t like the weather, wait five minutes!” And we mean it! It seems like the weather here is completely random. One day it snows, the next day it’s hot and humid. There’s really no predicting — which is why we’ve hired you.

Your job as our new meteorologist is to come up with totally random weather forecasts so that we can print them in the newspaper and talk about them on the TV.

Ready to get started? Okay, let’s forecast!

The first thing we need to do is understand how to get random values in JavaScript. Move on to the next section to find out!

Using `Math.random()`

JavaScript has a built-in function that’s used for creating random numbers. This function is called `Math.random()`.

Every time you run the `Math.random()` function, it creates a random decimal number between 0 and 1. Using this random value, you can do all sorts of things that are necessary for game programming, including adding an element of surprise to the movement of monsters or randomly selecting elements from arrays to create crazy weather forecasts.

Listing 17-3 shows a simple program that pops up a random value every time it’s run. Try running the program several times (in the JavaScript console or in JSFiddle) to verify that you don’t get the same value twice.
Listing 17-3  A Random Number Alert

    alert(Math.random());

Figure 17-2 shows the random number that we got when we ran this statement in JSFiddle.

The page at fiddle.jsfiddle.net says:  
0.4251535323167443

OK

Figure 17-2: A random number.

What programmers usually do with these long decimal numbers, is to use operators and other functions to create the values or the range of random values that they need.

If you want a random number between 0 and 10, you can multiply the random number by 11, like this:

    alert(Math.random() * 11);

If you want to remove the decimal numbers from the result, you can use the Math.floor() function, like this:

    alert(Math.floor(Math.random() * 11));

If you want a random number between 10 and 1,000, you can multiply the random value by the result of subtracting the smallest number from the largest number and then adding the smaller number to that result, like this:

    alert(Math.floor(Math.random() * (1000 - 100) + 100));

If you want to choose a random element from an array, it works the same way as picking a random number from a range starting with 0. Just multiply the random number by the length of the array.
For example, Listing 17-4 creates an array called myFriends and then uses Math.random() to choose one element from that array and alert the value of it.

**Listing 17-4  Finding a Random Friend**

```javascript
var myFriends = ["Agatha", "Agnes", "Jermaine", "Jack"];
var randomFriend = Math.floor(Math.random() * myFriends.length);
alert(myFriends[randomFriend]);
```

When you run this program in JSFiddle, the result will be that an alert with a random friend name will appear, as shown in Figure 17-3.

![Figure 17-3: Choose a random friend.](image)

Now that you understand how to get random data using JavaScript, let’s move on to writing the app!

**Writing the app**

To write the random weather forecast, follow these steps.

1. Go to http://jsfiddle.net and log in if you’re not already logged in.

2. Create a new program by clicking the JSFiddle icon.

3. Open the Fiddle Options panel on the left and enter a name for your program, such as Random Weather.

4. Click Save in the top menu to save your work and publish it to your Public Dashboard.
5. In the HTML pane, create a `div` element with an `id` of `5DayWeather`, like this:

```html
<div id="5DayWeather"></div>
```

6. In the JavaScript pane, start out by creating an array of the days of the week:

```javascript
var days = ["Monday","Tuesday","Wednesday","Thursday","Friday"];
```

7. Create a second array called `weather`.

The elements in this array should be different types of weather. Feel free to put in as many different types of weather as you can think of. Here’s our list:

```javascript
var weather = ["Sunny", "Partly Sunny", "Partly Cloudy", "Cloudy", "Raining", "Snowing", "Thunderstorm", "Foggy"];
```

8. Create two variables — `minTemp` and `maxTemp` — to hold the minimum and maximum temperatures that you want the random weather program to output.

Here are our numbers (in degrees Fahrenheit):

```javascript
minTemp = 0;
maxTemp = 100;
```

9. Start a new function, called `generateWeather()`.

```javascript
function generateWeather() {
```

10. The first line in the body of the function will start a `for` loop that will loop through each of the days of the week.

```javascript
for (var i = 0; i < days.length; i++) {
```

11. Declare a new variable, `weatherToday`, that will get a random element from the `weather` array.

```javascript
var weatherToday = weather[Math.floor(Math.random() * weather.length)];
```
12. Declare a new variable, `tempToday`, that will get a random temperature between the values of your `minTemp` and `maxTemp` variables.

```javascript
var tempToday = Math.floor(Math.random() * (maxTemp - minTemp) + minTemp);
```

13. Use `innerHTML` to output the values of `weatherToday` and `tempToday` inside the div element by adding the following code to the JavaScript pane.

```javascript
document.getElementById("5DayWeather").innerHTML += "<div id='" + days[i] + "' class='" + weatherToday + ">
"+<b>Forecast for " + days[i] + ":<br>" + weatherToday + " and " + tempToday + " degrees.</div>";
```

Notice that the above code adds the name of the day of the week as an ID attribute and the type of weather as a class attribute. We’ll use these later on to style the elements using CSS.

14. Close the loop and the function with closing curly brackets.

```javascript
}
}
```

15. Finally, insert a call to the `generateWeather` function after the variable declarations and above the function.

```javascript
generateWeather();
```

16. Click Update and then Set as Base in the top menu to save your work.

The finished JavaScript code should look like Listing 17-5.

**Listing 17-5  The Finished JavaScript Code**

```javascript
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]; var weather = ["Sunny", "Partly Sunny", "Partly Cloudy", "Cloudy", "Raining", "Snowing", "Thunderstorm", "Foggy"]; (continued)
```
Listing 17-5 (continued)

```javascript
var maxTemp = 100;
var minTemp = 0;

generateWeather();

function generateWeather() {
    for (var i = 0; i < days.length; i++) {
        var weatherToday = weather[Math.floor(Math.random() * weather.length)];
        var tempToday = Math.floor(Math.random() * (maxTemp - minTemp) + minTemp);

        document.getElementById("5DayWeather").innerHTML += 
            "<div id='" + days[i] + "' class='" + weatherToday + ">
                Forecast for " + days[i] + ":
            </b>
            " + weatherToday + " and " + tempToday + " degrees.</div>";
    }
}
```

When you run this program (by clicking Run in the top menu of JSFiddle) the result is the five weekdays followed by a weather prediction for each, as shown in Figure 17-4.

---

Figure 17-4: The result of running Listing 17-5.
Inspecting the results

Now we have a basic weather forecast, but it’s not very visually appealing. Fortunately, we had the foresight to add id and class attributes to each div in the output.

Follow these steps to inspect the output in the Result pane and see the HTML elements and their attributes that have been added by the JavaScript look:

1. Click Update or Run in the top menu bar in JSFiddle.

   The Result pane updates with a new list of forecasts.


   The Chrome Developer Tools open.

3. Click the Elements tab in the Chrome Developer tools.

   The Elements panel, shown in Figure 17-5, appears.

![Figure 17-5: The Elements panel in the Developer Tools.](image-url)
4. Click the Inspector tool (which looks like a magnifying glass) in the upper left of the Elements panel.

5. Move your mouse over the Results pane.

Elements within the pane will become highlighted as your mouse moves over them, as shown in Figure 17-6.

6. While your mouse is hovered over one of the days of the week in the Result pane, click it.

The Elements panel updates to highlight the code that created the element you clicked, as shown in Figure 17-7.
Chapter 17: What’s This Loop For?

7. Click some of the other elements in the Result pane and see how the id attributes, classes, and content of each element are different.

In the next section, we use the id and class attributes to apply styles to the output of the program.

Styling the app

By selecting elements by their id and class attributes, we can style each day, and customize the look of the days based on the weather predicted for that day.

Follow these steps to apply some style to the app:

1. Create a rule in the CSS pane to style each of the days the same using this code.

```css
#Monday, #Tuesday, #Wednesday, #Thursday, #Friday {
  width: 18%;
  height: 200px;
  float: left;
  border: 1px solid black;
  padding: 2px;
  font-family: sans-serif;
  font-size: 12px;
}
```

This rule creates a border, a width and height, a font family, and some padding on each of the days of the week. We’ve also set the float property equal to left in order to make all the days be side-by-side rather than stacked.

2. Create a rule for several of the different types of weather using this code:

```css
.Sunny {
  background-color: skyblue;
}
.Raining {
  background-color: lightgrey;
}
```
When you have spaces in a class attribute (such as in `Partly Sunny` and `Partly Cloudy`), the two words are treated as separate class attributes. So, an element with a class value of `Partly Cloudy` will be styled using the CSS associated with `.Cloudy`, and an element with a class value of `Partly Sunny` will be styled using the CSS associated with `.Sunny`.

3. Click Update and Set as Base to save your work.

The Result pane updates and displays your forecast in a new, more attractive format, as shown in Figure 17-8.

Figure 17-8: The finished Weather Forecast app.
The while loop will continue to perform its loop as long as its conditions are met. The while loop will do the job until it’s done — no questions asked!

In this chapter, we use a while loop to write a game that will keep buying you sandwiches until you run out of money. The object of the game is to make your lunch money last all week.
Writing a while Loop

Compared to for loops, while loops are pretty simple. They only have one part — a Boolean expression — that determines whether the loop will run and continue to run.

Here’s an example of a while loop:

```java
while (money > 0) {
    buyThings();
    saveMoney();
    payTaxes();
}
```

This loop executes the three functions — buyThings(), saveMoney(), and payTaxes() — as long as the value of the money variable is greater than 0.

The for loop has a final expression that changes the value of the counter. The while loop requires you to have an expression or expressions inside the loop that can change the result of its condition.

The three function calls we created inside the while loop are just made up names. If we were to actually write these functions, they would need to update the value of the money variable so that the loop stops at some point (but, of course, this is one loop we hope doesn’t stop!).

If you don’t modify the value of the variable in a while loop’s condition, you may create what’s called an infinite loop. An infinite loop won’t damage your computer, but it will likely cause your web browser to freeze up and cause you to have to force it to quit — risking losing any unsaved changes. So, make sure to check your while loops carefully to make sure they’re not infinite!

A while loop can do everything that a for loop can do, but the coding is just a bit different. Let’s take a look at the three uses for for loops that we talk about in Chapter 17 and show how to do them with while.
Looping a certain number of times
Listing 18-1 shows how you can use a while loop to log Hello, JavaScript! to the console window 500 times.

Listing 18-1  Logging Hello, JavaScript

```javascript
var i = 0;
while (i < 500) {
    console.log(i + ": Hello, JavaScript!");
    i++;
}
```

Notice that the program in Listing 18-1 contains all the same three parts that are in a for loop (initialization, condition, and final expression), but only the condition is inside the parentheses. The initialization (`var i = 0;`) is before the while loop, and the final expression (`i++`) is inside the while loop.

Counting with while
To create a loop that counts, you can just modify a variable inside every pass through the loop and use that variable inside other statements in the loop.

Listing 18-2 shows a countdown like the one from Chapter 17, but using a while loop.

Listing 18-2  Count Down with while

```javascript
var count = 10;
while (count > 0) {
    alert(count);
    count--;
}
alert("Blast Off!");
```

Looping through an array with while
Looping through arrays with while is easier than it is with for. To loop through an array with while, change the condition in the loop to test whether an array element has been declared.
To test whether an element has been declared, just put the name of the array with a counter variable inside the parentheses after the `while` keyword.

For example, Listing 18-3 shows an example that loops through a list of people’s names.

**Listing 18-3  Looping through a List of Names**

```javascript
var people = ["Deborah","Carla","Mary","Suzen"];
var i = 0;
while (people[i]) {
    alert(people[i]);
    i++;
}
```

The condition between the parentheses in a `for` loop or a `while` loop is a Boolean expression, which means it evaluates to either `true` or `false`. When you use an array element, such as `people[5]` as a Boolean expression, it will be true as long as there is an element in the array at that array position.

**Coding the Lunch Game**

The Lunch Game is a unique combination of a game of chance and a game of math. The object of the game is to try to budget so that you have sandwiches for every day of the week.

But, here’s the catch: You go to the strangest school in the world, and you don’t know how much sandwiches will cost until the sandwiches are made — but you have to buy all your sandwiches for the week before the week starts!

You do know that sandwiches will always cost between $1 and $5. So, depending on your luck, you’ll be able to buy somewhere between 4 and 20 sandwiches.

How much risk are you willing to take? Will someone come to your aid and give you part of their sandwich if you run out before the week ends? How many sandwiches can you eat?
All these questions, and more, will be answered in the Lunch Game.

**Forking the code**

To get started with writing the Lunch Game, follow these steps:

1. Go to our JSFiddle Public Dashboard at http://jsfiddle.net/user/forkids/fiddles.
   
   You see the list of all our public programs.

2. Find the program named “Chapter 18 – Lunch Game – Start,” and click the title to open it.
   
   The starter program opens, as shown in Figure 18-1.

   We’ve written the HTML, CSS, and most of the JavaScript for you. The only thing left for you to do is to write the `buyLunches()` function.

   In the next section, we show you how to do it!
Writing buyLunches()

Listing 18-4 shows the starter code and comments for the buyLunches() function.

Listing 18-4 The Starting Point for buyLunches()

```javascript
/*
   buys specified number of sandwiches per day at current prices
*/
function buyLunches() {
    /*
    todo:
    * reset the form
    * start a loop
    * get daily sandwich order
    * calculate total price
    * figure out if there's enough money
    * if so: subtract money, increment number of lunches, and start loop over
    * if not: display 'out of money' message
    * display total lunches after loop exits
    */
}
```

Follow these steps to write the body of the function to match these instructions:

1. Inside the body of the buyLunches() function, make a call to the resetForm() function and initialize a variable for tracking the current day, like this:

   ```javascript
   resetForm();
   var day = 0;
   ```

2. Create a loop that will buy sandwiches until you’re out of money.

   ```javascript
   while (money > 0) {
   ```
3. Get the current price of sandwiches by making a call to the `getSandwichPrice()` function and assigning the return value to a variable.

   ```javascript
   var priceToday = getSandwichPrice();
   ```

   At this point, take a look at the `getSandwichPrice` function. Its purpose is to randomly generate a number between 1 and 5 and return that value.

4. Get the number of sandwiches that the user entered into the form field.

   ```javascript
   var numberOfSandwiches = document.getElementById("numSandwiches").value;
   ```

5. Calculate the total price by multiplying the number of sandwiches that you want by the current sandwich price.

   ```javascript
   var totalPrice = priceToday * numberOfSandwiches;
   ```

6. Find out whether there’s enough money to buy the sandwiches.

   ```javascript
   if(money >= totalPrice) {
   ```

7. If there is enough, subtract the total price from the current money balance.

   ```javascript
   money = money - totalPrice;
   ```

   Congratulations! You’ve successfully purchased a lunch!

8. Increment the `lunches` variable, which keeps track of how many lunches were purchased.

   ```javascript
   lunches++;
   ```

9. Output a message to tell the user the price of the sandwiches he just bought and how much money he has left.
Notice that we’ve attached the `toFixed()` method to the `money` variable. The `toFixed()` method converts a number to a string, while keeping the number of decimals specified within the parentheses. In this case, because we’re printing out a currency value, we use two decimal places.

10. Next, start the `else` clause of the `if...else` to handle cases where the amount of money left isn’t enough to buy the specified number of sandwiches.

```javascript
} else {
```

11. When the `else` clause runs, output a message that’s special for when the user doesn’t have enough money for another lunch.

```javascript
    document.getElementById("receipt").innerHTML += "<p>Today, sandwiches are: $" + priceToday + ". You don't have enough money. Maybe your sister will give you some of her sandwich.</p>";
```

12. Still within the `else` clause, set the value of `money` equal to 0 in order to prevent the loop from running again.

```javascript
    money = 0;
```

13. Finish the `if...else` statement and the `while` loop with curly brackets.

```javascript
} }
```

14. When the loop completes, output the total number of lunches that the user was able to buy.

```javascript
    document.getElementById("receipt").innerHTML += "<p>You bought " + lunches + " lunches this week.</p>";
```
15. Close the function with a curly bracket.

    }  

16. Click Update and Set as Base in the top menu to save your work.

**Trying it out**

The finished Lunch Game is shown in Figure 18-2.

![Figure 18-2: The finished Lunch Game.](image)

If you enter a number into the text field and press the Place Order button, the program calculates how many lunches you can buy, using random sandwich prices. **Remember:** A lunch consists of one or more sandwiches, according to your input.

Try out the program several times by entering new numbers into the text input field and pressing the Place Order button. You see that the random numbers and the number of lunches you can buy per week vary quite a bit.

Figure 18-3 shows one possible outcome of running the program.
Moving to Your Own Website

When you have a game that you’re proud of and you want to share with the world on your own website, you need to move beyond the walls of JSFiddle. In this section, we show you how to do that!

Understanding web hosting

Every website has a unique address that people can use to visit it. In order to get your own address on the Internet, you need to sign up with some sort of web hosting company. JSFiddle is a web hosting company that provides a free testing area for people to make programs with JavaScript, HTML, and CSS.

JSFiddle is great, but it has its limits, such as the fact that it lets anyone copy and modify your code, and it doesn’t give you the option of having your own domain name (such as www.mywebsite.com).
Most web hosting companies charge a monthly fee for uploading your web pages to the Internet. However, there are some that give out free trial accounts. In this section, we show you how to set up and use a free trial account with x10Hosting (www.x10hosting.com).

It’s possible that x10Hosting may choose to start charging for trial accounts or change in some way before we have a chance to update this book. If this happens, you can find different free hosting options by searching the web for “free web hosting.”

**Getting started with x10Hosting**

Follow these steps to create an account and a website at x10Hosting:

1. Open your web browser and go to www.x10hosting.com. You see the home page, featuring a button labeled Sign Up Now, as shown in Figure 18-4.

   ![Figure 18-4: The home page of x10Hosting.](image)
2. Click the Create My Account button.

   A form where you can enter a name for your custom web address appears.

3. Choose a name for your hosting account, as shown in Figure 18-5, and click Continue.

4. On the next screen, enter your email address and click Continue.

5. Choose a password for your account, and click Continue.

6. Click to agree to the terms of service, and click Submit to finish signing up.

   An email confirmation is sent to you.

7. Click the link in the email to confirm your account.

   If you don’t get the email within a few minutes, check your spam folder.

8. When your account is confirmed, click Continue to log in.

   It may take minute for your account to be ready. If you see a message telling you to wait, take a break and then come back and click the Continue button when it becomes available.

9. Enter your name on the next page to personalize your account, and then click Continue.
10. When your domain is set up, you see a page with a help window, a link to your domain, and a link that says Open cPanel.

11. Click the link that says Open cPanel.

Your control panel opens.

12. Click the Add Website link.

13. Give your site a name, leave the default domain selected, and leave the address path text input blank, as shown in Figure 18-6.

![Add Website](image)

Figure 18-6: Creating a new website.

14. Click Add Website.

Your new site is created and you see the unique website address.
Part VI: Loops

Make a note of this website address. You'll be using it later!

15. Click Continue to My Websites.

16. In the control panel for your website, click the File Manager link.

A window opens, showing you the files and directories in your web hosting account (see Figure 18-7).

17. Click New File at the top of the screen.

18. Name the new file lunchgame.html and click Create New File.

19. Highlight the new file by clicking it, and then click Code Editor in the top menu.

20. If this is your first time using the code editor, a window opens asking you to choose an “encoding”; click the Disable Encoding Check link.

A blank page opens in the code editor.

21. Type the HTML from Listing 18-5 into this blank page.
In another browser tab, go back to your Lunch Game in JSFiddle.

Copy everything inside the HTML pane, and paste it between the opening and closing body tags in the code editor for your lunchgame.html file.

Copy everything inside the CSS pane in JSFiddle and paste it between <style> and </style> in the lunchgame.html file.

Copy the first lines from the JavaScript pane, up to the function declaration for the buyLunches() function, and paste it in the function body for the init() function in the lunchgame.html file, as shown in Listing 18-6.

Check your code carefully after you paste, to make sure that it matches Listing 18-6 exactly.
The `init()` function runs as soon as the web page is loaded.

26. Paste the rest of the JavaScript from the JavaScript pane in JSFiddle below the `init()` function, but still between the `<script>` and `</script>` tags.

27. Click Save in the upper-right corner of the screen.

28. Click Close, just to the left of the Save button.

If you get a message regarding the character encoding, you can just close it by clicking OK and you’re returned to the File Manager.

29. Go to your website address in a new browser tab.

You see a list of the files in your website. Currently, you should only have a folder called `cgi-bin` and your `lunchgame.html` file.

If you don’t want to see this list of web pages, you can create a new HTML file called `index.html`, and it will appear when you visit your website instead.
30. Click `lunchgame.html` to open the Lunch Game.

The Lunch Game appears in your browser window, as shown in Figure 18-8.

![Image of Lunch Game](image.png)

**Figure 18-8**: The finished Lunch Game, hosted on your own website!
While working on your random weather forecaster and your lunch app, you came up with a genius business idea: You could combine the two programs to open a lemonade stand!

As the local weather forecaster for Anytown, you should have an advantage over every other lemonade stand in the area. Here’s how it works: People buy more lemonade when it’s hot out. They’re also more willing to pay more money to buy lemonade when it’s hot out. By setting your lemonade price and deciding how much lemonade to make based on the weather, you can maximize your profit and minimize wasted lemonade.

In this chapter, you learn how to build a lemonade stand game.
Playing the Game

Before we get started building the lemonade stand game, let’s try it out and see how it works!

Follow these steps to open and play the game:

1. Go to our JSFiddle Public Dashboard at http://jsfiddle.net/user/forkids/fiddles.

2. Find the program titled “Chapter 19 – Lemonade Stand” and open it by clicking the title.

The game will open and run. You see the Lemonade Stand game, as shown in Figure 19-1.

3. Take a look at the weather forecasts for the week, in the top part of the Result pane.

These are the daily weather forecasts that the game will use to help calculate how many glasses of lemonade your stand sells.
4. Enter a number in the field labeled “How many glasses of lemonade do you want to make this week?”

Remember that you’re making lemonade for the whole week, so the number should be large enough that you don’t run out of lemonade before Friday! **Hint:** Try different quantities, including some values in the hundreds.

5. Enter the price you want to charge per glass.

Your cost (how much you pay to make it) per glass is $0.50, so make sure to price your lemonade higher than what it costs you to make it.

6. Click the Open the Stand button.

A report of your daily and weekly sales will be displayed. Note the last line of the report, which tells you how much profit you made. Is this number greater than 0, or is it negative? If it’s greater than 0, good job!

7. Try changing the price per glass or the number of glasses you make based on the results you got and click Open the Stand again.

Do you notice any patterns in how the profit increases or decreases based on the price of lemonade? Can you figure out how to maximize the profit and minimize the number of glasses of lemonade you have left over?

8. Click Run in the top menu of JSFiddle to generate a new week’s worth of weather and then try the game again.

9. Compare the number of glasses sold to the temperature each day.

Notice that the daily temperature changes how many glasses of lemonade you sell.
Now that you’ve seen the lemonade stand program in action, let’s back up and talk about a few math and business ideas that the game is based on.

Whether you’re running a lemonade stand or just managing your own allowance and how much you spend on comic books and candy, these basic principles of economics apply.

A Lesson in Business

When you open a lemonade stand, you’re running a business. As a new business owner, your primary goal is to make enough profit to be able to continue running the lemonade stand.

You may have other goals for running a lemonade stand as well, such as to spend your days outside in the sun, or to have fun talking with customers, or to learn to make the very best lemonade in the world. But if you don’t make enough profit to be able to keep the lemonade stand running, you can’t enjoy the other benefits of having a lemonade stand.

In order to make a profit selling lemonade, you need to understand your customers and why they buy lemonade from you. Just as you may have many different reasons for running your lemonade stand, customers may have many different reasons for buying lemonade from you and many different factors influence their decision. A few of the factors may include the weather, the price, how much money they have, where your lemonade stand is located, and how your lemonade tastes. Something as simple as buying and selling lemonade can actually be very complicated!

In order to make a game out of a lemonade stand, we need to focus on just a few of the many factors that are involved in the process.

Making a profit

Profit is what’s left over from the total revenue of a business (all the money that comes in) after expenses (everything that the business spends money on).
In a lemonade stand, you may have all the following expenses: lemons, sugar, ice, cups, and stand maintenance (things like paint, repairs, and so on). You’ve done the math, and calculated that when you combine all your expenses, the cost for you to make a cup of lemonade is about $0.50. In order to make back your investment in the lemonade stand, you need to earn at least $0.50 for each glass of lemonade that you make.

**Understanding your customers**

As you know, the temperature in Anytown changes all the time, but one thing is for sure: The hotter it gets, the more lemonade people buy. But if the price of lemonade is too high, people won’t buy it.

As a lemonade stand owner, your goal is to figure out how much lemonade to make and how much to charge for it in order to make the greatest profit.

**Understanding the math**

Here’s the basic formula that our game uses to calculate how much lemonade is sold each day:

\[
\text{Glasses Sold} = \frac{\text{Temperature}}{\text{Price}}
\]

For example, if the temperature is 100 degrees, and the price of lemonade is $2, the math looks like this:

\[
\text{Glasses Sold} = \frac{100}{2}
\]

The result is that you sold 50 glasses of lemonade.

However, if the temperature is lower, say 50 degrees, the formula looks like this:

\[
\text{Glasses Sold} = \frac{50}{2}
\]

The result is that you only sold 25 glasses of lemonade.
However, if you lower the price of lemonade to $1, the math looks like this:

\[
\text{Glasses Sold} = 50 \div 1
\]

The result is that you can sell 50 glasses of lemonade at the lower price when the temperature is lower.

**Graphing sales, temperature, and price**

Understanding the relationship among glasses sold, temperature, and price is important to master the game. Follow these steps to visualize this relationship using a 3D graph:


   You’ll see the homepage of WolframAlpha, as shown in Figure 19-2.

![Figure 19-2: The WolframAlpha home page.](image)

2. In the search form, type **3D plot**.

   You see the search results with a Function to Plot field, as shown in Figure 19-3.
3. Below the Function to Plot field, click the Variables and Ranges link.

Additional fields appear, as shown in Figure 19-4.

4. In the Function to Plot field, enter $z = x/y$.

The letter $z$ represents the number of glasses sold, the letter $x$ represents the temperature, and the letter $y$ represents the price.
5. In the Lower Limit 1 field, enter 0.

The Lower Limit 1 field represents the lowest value we want to graph for the variable \( x \), which corresponds to the temperature value in our lemonade stand.

6. In the Upper Limit 1 field, enter 100.

This represents the maximum temperature value that we’ll graph.

7. In the Lower Limit 2 field, enter 0.

This represents the lowest value for price in the lemonade stand.

At $0 per glass, you’re sure to sell a lot of lemonade, but we don’t recommend this strategy in the long run if you’re trying to run a business!

8. In the Upper Limit 2 field, enter 10.

It’s unlikely that you’ll want to charge anyone more than $10 for a glass of lemonade, so we’ll set the upper limit to 10.

9. Click one of the orange equal sign buttons next to the input fields to graph your function.

The results appear, and you see a graph similar to the one shown in Figure 19-5.

Notice that on the graph in Figure 19-5, the highest possible number of glasses of lemonade sold happens when the temperature is at the maximum and the price is at the minimum.

WolframAlpha can do a lot of really interesting things! Feel free to try out different values and try making different graphs.
Building the Game

Now that you have a better understanding of the math behind the lemonade stand, let’s build the game!

We’ve already started building it for you, so the first step is to fork our code to make your own copy.

Forking the code

Follow these steps to make a copy of the starter app:

1. Log in to JSFiddle and go to our public dashboard at http://jsfiddle.net/user/forkids/fiddles.

2. Find the program named “Chapter 19 – Lemonade Stand – Start” and click its title to open it.

3. Click the Fork link in the top menu to create your own copy of the program.

Figure 19-5: Graphing the relationship between glasses sold, temperature, and price.
4. Open the Fiddle Options in the left navigation and change the title to “(Your Name)’s Lemonade Stand.”

5. Click Update and then Set as Base to save your own copy of the Lemonade Stand game.

**Writing the JavaScript**

Take a look at the starter program for the Lemonade Stand game. We’ve written enough HTML and CSS to get you started, but the JavaScript pane is completely blank.

When you run the program, the HTML shows up in the Results pane, but clicking the button doesn’t do anything.

Let’s walk through each thing that the Lemonade Stand needs to do and write code and comments as we go.

**Creating globals**

The first thing we’ll do is to define some global variables that will be used in the program. We’ll need the following:

- An array of days of the week
- An array of weather types
- Minimum and maximum temperature values
- The cost of making a cup of lemonade
- An array to hold actual daily temperatures

Before writing any code, create comments in the JavaScript window for each of these items, as shown in Listing 19-1.

**Listing 19-1  Create Comments for Variables**

```javascript
// create days of week array

// define types of weather
```

(continued)
Listing 19-1 (continued)

// set min and max temperatures

// cost (to you) of a cup of lemonade

// array for storing daily temps

Now that we have comments, follow these steps to fill in the actual variable declarations.

1. Under the first comment (create days of week array), type the following:

   var days = ["Monday", "Tuesday", "Wednesday", "Thursday", 
               "Friday"];

2. Under the next comment (define types of weather), create an array of descriptions of weather.

   Here's our array as an example:

   var weather = ["Sunny", "Partly Sunny", "Partly Cloudy", 
                   "Cloudy", "Raining", "Snowing", "Thunderstorm", 
                   "Foggy"];

   Feel free to add or delete any types of weather you like.

3. Under the next comment (set min and max temperatures), create two new variables to hold the coldest and hottest temperatures that you’d like to use in your game.

   Here’s our example code:

   var maxTemp = 100;
   var minTemp = 0;

4. Under the next comment (cost of a cup of lemonade), declare a variable named lemonadeCost and give it a numeric value of your cost to make a cup of lemonade, in dollars.

   var lemonadeCost = 0.5;
5. Create an empty array, called `dailyTemp`, to hold the daily temperature values.

   ```javascript
   var dailyTemp = [];
   ```

6. Click Update to save your work.

7. Your JavaScript pane should now look like Listing 19-2.

Listing 19-2  The Globals Have Been Created

```javascript
// create days of week array
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];

// define types of weather
var weather = ["Sunny", "Partly Sunny", "Partly Cloudy", "Cloudy", "Raining", "Snowing", "Thunderstorm", "Foggy"];

// set min and max temps
var maxTemp = 100;
var minTemp = 0;

// cost (to you) of a cup of lemonade
var lemonadeCost = 0.5;

// array for storing daily temps
var dailyTemp = [];
```

**Generating weather**

The next step in writing our program is to generate the weather. Fortunately, we already have a function for generating random weather — namely, the random weather app that we wrote in Chapter 17.

We’re going to make one addition to the `generateWeather` function from our random weather app created in Chapter 17. We’ll store the daily weather in a global array called `dailyTemp`. 
Follow these steps to write the `generateWeather` function:

1. Write a comment describing the purpose of the function.

```javascript
/**
generates weather for the week
**/
```

2. Write the function head.

```javascript
function generateWeather() {
```

3. Create two function variables to hold the current weather and temperature.

```javascript
var weatherToday;
var tempToday;
```

4. Start a for loop to cycle through each day of the week.

```javascript
for (var i = 0; i < days.length; i++) {
```

5. Get a random element from the weather array and assign it to `weatherToday`.

```javascript
weatherToday = weather[Math.floor(Math.random() * weather.length)];
```

6. Get a random temperature between the values of `minTemp` and `maxTemp`.

```javascript
tempToday = Math.floor(Math.random() * (maxTemp - minTemp) + minTemp);
```

7. Store the temperature in the `dailyTemp` array.

```javascript
dailyTemp[i] = tempToday;
```

8. Output a message describing the day’s weather.

```javascript
document.getElementById("5DayWeather").innerHTML += "<div id='" +
days[i] + "' class='" + weatherToday + 
"'>\<b>\Forecast\ for \ " + days[i] + ":\</b><br>" +
weatherToday + " and " + tempToday + " degrees.";
```
9. Close the loop and the function.

}

}

10. Call the function when the program loads, by typing the following below the global variable declarations.

    generateWeather();

11. Click Update to save your work.

That completes the weather generation function. If you did everything correctly, a table of the week’s weather should display in the Result pane now, above the input fields, as shown in Figure 19-6.

Figure 19-6: The Result pane, containing weather and input fields.

Compare your JavaScript with the code in Listing 19-3 and make sure that they match before moving on.
Listing 19-3 The Completed Globals and the generateWeather Function

// create days of week array
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];  

// define types of weather
var weather = ["Sunny", "Partly Sunny", "Partly Cloudy", "Cloudy", "Raining", "Snowing", "Thunderstorm", "Foggy"];  

// set min and max temps
var maxTemp = 100;
var minTemp = 0;

// cost (to you) of a cup of lemonade
var lemonadeCost = 0.5;

// array for storing daily temps
var dailyTemp = [];

// make the week's weather
generateWeather();

/**
generates weather for the week
**/
function generateWeather() {
    var weatherToday;
    var tempToday;
    for (var i = 0; i < days.length; i++) {
        weatherToday = weather[Math.floor(Math.random() * weather.length)];
        tempToday = Math.floor(Math.random() * (maxTemp - minTemp) + minTemp);
        dailyTemp[i] = tempToday;
    }
}
Opening the stand

The next function we’ll create is the one that opens the stand and calculates the number of glasses sold for the week.

Follow these steps to write the `openTheStand` function.

1. Write a comment describing the function and then write the function head.

   ```javascript
   /**
    * calculates glasses of lemonade sold
   * @function openTheStand
   */
   function openTheStand() {

2. Create three new variables — one to hold the daily number of glasses sold, one to hold the weekly total, and one to hold the number of glasses we have left to sell — and initialize all three with 0.

   ```javascript
   var glassesSold = 0; // daily
   var totalGlasses = 0; // weekly
   var glassesLeft = 0; // left to sell
   ```

3. Call a function named `resetForm()`, which resets the report area of the program so that it can be run multiple times without restarting the game.

   ```javascript
   // clear previous results
   resetForm();
   ```

   We’ll write the `resetForm` function after we finish `openTheStand()`.
4. Get the values from the form fields.

    // get input
    var numGlasses = Number(document.getElementById("numGlasses").value);
    var glassPrice = Number(document.getElementById("glassPrice").value);

5. Create a new loop to cycle through each day of the week.

    for (var i = 0; i < days.length; i++) {

6. Calculate the number of glasses sold.

    // glasses sold depends on temp and price
    glassesSold = Math.floor(dailyTemp[i] / glassPrice);

7. Calculate how many glasses are left.

    // how many glasses do we have now?
    glassesLeft = numGlasses - totalGlasses;

8. Write an `if...else` statement that checks whether you’re out of lemonade.

    // we can't sell more than we have
    if (glassesSold > glassesLeft) {
        glassesSold = glassesLeft;
    }

    If `glassesSold` is greater than what we have left, just sell what we have left by setting the `glassesSold` variable equal to what’s left.

9. Increase the weekly total of glasses sold.

    // increase the weekly total
    totalGlasses = glassesSold + totalGlasses;

10. Display the daily totals.

    // display daily total
    document.getElementById("result").innerHTML += "<p>" + days[i] + ", you sold " + glassesSold + " glasses of lemonade.";
    

11. Finish the loop with a curly bracket.
}

12. Make a call to the function that will display the weekly results, passing it three arguments: `numGlasses`, `glassPrice`, and `totalGlasses`.

```
displayResults(numGlasses, glassPrice, totalGlasses);
```

13. Finish the function by typing a closing curly bracket.
}

14. Save your work by clicking Update.

If you did everything right, your `openTheStand` function should match the code in Listing 19-4.

**Listing 19-4  The openTheStand Function**

```javascript
/**
calculates glasses of lemonade sold
**/
function openTheStand() {
    var glassesSold = 0; // daily
    var totalGlasses = 0; // weekly
    var glassesLeft = 0; // left to sell

    // clear out previous results
    resetForm();

    // get input
    var numGlasses = Number(document.
        getElementById("numGlasses").value);
    var glassPrice = Number(document.
        getElementById("glassPrice").value);

(continued)
```
Listing 19-4 (continued)

    for (var i = 0; i < days.length; i++) {

        // glasses sold depends on temp and price
glassesSold = Math.floor(dailyTemp[i] /
            glassPrice);

        // how many glasses do we have now?
glassesLeft = numGlasses - totalGlasses;

        // we can't sell more than we have
        if (glassesSold > glassesLeft) {
            glassesSold = glassesLeft;
        }

        // increase the weekly total
totalGlasses = glassesSold + totalGlasses;

        // display daily total
document.getElementById("result").innerHTML +=
            "<p>" + days[i] + ", you sold " + glassesSold +
            " glasses of lemonade.";</p>"

    }

displayResults(numGlasses, glassPrice, totalGlasses);

Resetting the program
One of the first things that the openTheStand() function does is to make a call to a function called resetForm(). This function is very simple. Its sole purpose is to clear out the content from the report area of the program so that you can run the program again without the results being added to the bottom of the previous output.

Listing 19-5 shows the complete code for resetForm(). Type this function into the JavaScript pane, underneath the openTheStand() function (at the very end of the code in the JavaScript pane).
After you’ve written the resetForm() function, click the Update link to save your work.

**Displaying a report**

The final function in the Lemonade Stand game is the displayResults() function. This function calculates weekly results using arguments supplied to it by the openTheStand() function and outputs a report about how you did.

Follow these steps to write displayResults().

1. Write a comment describing the function and the function header, with three parameters: weeklyInventory, glassPrice, and weeklySales.

   /**
    * calculates results and displays a report
    */
   function displayResults(weeklyInventory, glassPrice, weeklySales) {

2. Calculate your total revenue by multiplying the total number of glasses sold times the price that was paid for each glass.

   var revenue = weeklySales * glassPrice;

3. Calculate your expenses by multiplying the number of glasses of lemonade you made times the cost (to you) of each glass.

   var expense = weeklyInventory * lemonadeCost;
4. Calculate how many glasses are left over by subtracting the total sales from the number of glasses you made.

   var leftOver = weeklyInventory - weeklySales;

5. Calculate your profit by subtracting expenses from the total revenue.

   var profit = revenue - expense;

6. Write out the final report using the following four statements:

   // print out the weekly report
   document.getElementById("result").innerHTML += "<p>You sold a total of " + weeklySales + " glasses of lemonade this week.</p>";
   document.getElementById("result").innerHTML += "<p>Total revenue: $" + revenue + ".</p>";
   document.getElementById("result").innerHTML += "<p>You have " + leftOver + " glasses of lemonade left over.</p>";
   document.getElementById("result").innerHTML += "<p>Each glass costs you $" + lemonadeCost + ". Your profit was $" + profit + ".";

7. Finish the function with a closing curly bracket.

   }

8. Click Update to save your work.

The final function should match Listing 19-6.

**Listing 19-6  The Final displayResults Function**

```javascript
/**
calculates results and displays a report
**/
function displayResults(weeklyInventory, glassPrice, weeklySales) {
    // calculate results
    var revenue = weeklySales * glassPrice;
    var expense = weeklyInventory * lemonadeCost;
}```
var leftOver = weeklyInventory - weeklySales;
var profit = revenue - expense;

// print out the weekly report
document.getElementById("result").innerHTML += "<p>You sold a total of " + weeklySales + " glasses of lemonade this week.</p>";
document.getElementById("result").innerHTML += "<p>Total revenue: $" + income + ".</p>";
document.getElementById("result").innerHTML += "<p>You have " + leftOver + " glasses of lemonade left over.</p>";
document.getElementById("result").innerHTML += "<p>Each glass costs you $" + lemonadeCost + ". Your profit was $" + profit + ".";

---

**Finishing and testing the program**

If you try out the program now, you’ll discover that it doesn’t do anything except print out the random weather forecast.

There’s one thing left that we need to do. Do you know what it is?

If you said that we need to listen for the click event on the button, you’re exactly right. The click event is the switch that makes the lemonade stand work.

Follow these steps to finish the program and test it out.

1. Type the following code before the function declarations in the JavaScript pane:

```javascript
// listen for order
document.getElementById("OpenTheStand").addEventListener("click", openTheStand);
```

2. Click Update and Set as Base to save your work.

The final code in the JavaScript pane should match Listing 19-7.
Listing 19-7  The Lemonade Stand Program

// create days of week array
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];

// define types of weather
var weather = ["Sunny", "Partly Sunny", "Partly Cloudy", "Cloudy", "Raining", "Snowing", "Thunderstorm", "Foggy"];

// set min and max temps
var maxTemp = 110;
var minTemp = 32;

// cost (to you) of a cup of lemonade
var lemonadeCost = 0.5;

// array for storing daily temps
var dailyTemp = [];

// listen for order
document.getElementById("OpenTheStand").addEventListener("click", openTheStand);

generateWeather();

/**
generates weather for the week
**/
function generateWeather() {
    var weatherToday;
    var tempToday;
    for (var i = 0; i < days.length; i++) {
        weatherToday = weather[Math.floor(Math.random() * weather.length)];
        tempToday = Math.floor(Math.random() * (maxTemp - minTemp) + minTemp);
        dailyTemp[i] = tempToday;
    }
}
```
document.getElementById("5DayWeather").innerHTML
  += "<div id='" + days[i] + "' class='" + weatherToday + '">
    b>Forecast for " + days[i] + "</b><br>
    " + weatherToday + " and " + tempToday + " degrees."</div>";

/**
calculates glasses of lemonade sold
**/
function openTheStand() {
  var glassesSold = 0; // daily
  var totalGlasses = 0; // weekly
  var glassesLeft = 0; // left to sell

  // clear previous results
  resetForm();

  // get input
  var numGlasses = Number(document.
    getElementsById("numGlasses").value);
  var glassPrice = Number(document.
    getElementsById("glassPrice").value);

  for (var i = 0; i < days.length; i++) {
    // glasses sold depends on temp and price
    glassesSold = Math.floor(dailyTemp[i] /
      glassPrice);

    // how many glasses do we have now?
    glassesLeft = numGlasses - totalGlasses;

    // we can't sell more than we have
    if (glassesSold > glassesLeft) {
      glassesSold = glassesLeft;
    }
  }
```

(continued)
Listing 19-7 (continued)

    // increase the weekly total
    totalGlasses = glassesSold + totalGlasses;

    // display daily total
    document.getElementById("result").innerHTML +=
        "<p>" + days[i] + ", you sold " + glassesSold + 
        " glasses of lemonade.</p>";

} }

displayResults(numGlasses, glassPrice, totalGlasses);

/**
calculates results and displays a report
**/
function displayResults(weeklyInventory, glassPrice,
    weeklySales) {
    // calculate results
    var revenue = weeklySales * glassPrice;
    var expense = weeklyInventory * lemonadeCost;
    var leftOver = weeklyInventory - weeklySales;
    var profit = revenue - expense;

    // print out the weekly report
    document.getElementById("result").innerHTML += 
        "<p>You sold a total of " + weeklySales + " glasses of lemonade this week.</p>";
    document.getElementById("result").innerHTML +=
        "<p>Total revenue: $" + revenue + ".</p>";
    document.getElementById("result").innerHTML +=
        "<p>You have " + leftOver + " glasses of lemonade left over.</p>";
    document.getElementById("result").innerHTML +=
        "<p>Each glass costs you $" + lemonadeCost + ".
Your profit was $" + profit + ".";
3. Enter a value into the form field labeled “How many glasses of lemonade do you want to make for the week?”

4. Enter a value into the form field labeled “How much will you charge for a glass of lemonade this week?”

5. Press the Open the Stand button.

You see how many glasses of lemonade you sold each day, followed by the weekly totals and the profit, as shown in Figure 19-7.

Monday, you sold 17 glasses of lemonade.
Tuesday, you sold 53 glasses of lemonade.
Wednesday, you sold 50 glasses of lemonade.
Thursday, you sold 38 glasses of lemonade.
Friday, you sold 39 glasses of lemonade.
You sold a total of 197 glasses of lemonade this week.
Total revenue: $394.
You have 3 glasses of lemonade left over.
Each glass costs you $0.5. Your profit was $294.

Figure 19-7: The final Lemonade Stand game.
How did you do? Did you make a profit? Can you increase your profit by changing the price or number of glasses? Does one of the methods of increasing profit seem to work better than the other? What happens when you set either the price or the glasses of lemonade to a very large number? What happens when either one is set to a very small number?

When you’re ready, move on to the next section to get some ideas for improvements you may want to make to the Lemonade Game!

### Improving the Lemonade Game

The Lemonade Game is interesting and demonstrates a number of important JavaScript principles. By now, however, you likely have ideas for how it could be improved to be more of a challenge, more fun, or more realistic.

If you’ve made it this far in the book, you have a good understanding of JavaScript and you’re ready to head off on your own and start modifying and building programs by yourself. Excellent work!

Here are a few ideas to get you started with making modifications to the Lemonade Stand game:

- Allow the user to control the price and how much lemonade is made on a daily basis, rather than weekly.
- Factor the type of weather (rainy, snowy, and so on) into the calculation of how many glasses were sold, instead of just using the temperature.
- Randomize the cost (the price you pay) per glass of lemonade.
- Write more HTML and CSS to improve, or just change, the look of the game.
- Create a button that generates new random weather for a new week, instead of making the player start the game over when they want a new week of weather.
✓ Save the user’s high score in a variable and let them know if they improve from game to game.

✓ Calculate the cost of lemonade based on values for the price of lemons and the price of sugar, as well as how many lemons and how much sugar it takes to make how many glasses of lemonade.

✓ Create random events in the game, such as blizzards or dogs that knock over the stand, which sometimes cause no lemonade at all to be sold on a day.

These are just a few of the hundreds of different improvements that could be made to the Lemonade Stand game. If you make an improvement that you want to share, please show it to us on Facebook, Twitter, or via email at info@watzthis.com. We’re excited to see what you come up with!
Symbols and Numerics

+ (addition), 150–151
&& (and operator), 239
== (equality operator), 145–148, 234
> (greater than operator), 149, 151
>= (greater than or equal to operator), 149–150
!= (inequality operator), 148
|| (or operator), 239
=== (strict equality operator), 147, 234
!== (strict inequality operator), 148–149
; (semicolon)
in CSS declarations, 93
to separate statements, 25, 31

A

Activity Calendar program,
   255–266
   building, 261–266
Date object, 257–261
forking, 256–257
introduction to, 255
using, 256
addEventListener method,
   111–113, 116, 163
addition (+), 150–151
addTheThing() function, 216, 222
addToTheList() function, 216, 219–222
<a> element, 79
alert() command, 44–46
alert() function, 193
alert keyword, 25
alt attribute, 82
and operator (&&), 239
animation
   creating with JavaScript, 115–117
Douglas the JavaScript Robot,
   104–122
   adding a second animation function, 118–122
   animating another element, 118
   arm sweep animation, 118–120
   changing CSS with JavaScript, 105–106
   creating animations with JavaScript, 115–117
   experimenting with Douglas, 108–109
   handling events, 111–113
   making Douglas dance, 109–111
   modifying Douglas with JavaScript, 106–108
   writing a listener, 113–115
appendChild() method, 222
arguments, passing, 195
arithmetic operators, 140–143
arm sweep animation, 118–120
array methods, 177–178
  in JSFiddle, 178–179
arrays, 173–189
  changing element values, 176–177
  creating and accessing, 175–176
  definition of, 174–175
  empty, 278
  in general, 173
  getting array values, 176
  storing different data types in, 175
  variables inside, 176
askQuestion( ) function,
  280–283, 287
assignment operator, 43
asterisk (*), to multiply numbers, 20
<audio> tag, 73

backslash ( )
  printing a, 28
  in a string, 27
Beginning HTML5 and CSS3 For Dummies, 78
beginning tags, 73
binary codes, 9–10
bit, 10
body element, 76, 91, 94, 160, 161
Boolean data type, 41–42
branching, defined, 267
bubbles, size of, 62
Bubbles demo, 55–57, 60, 64
  sharing your fiddle, 65–66
built-in functions, 191
buyLunches( ) function, 313, 314, 323
bytes, 10

calculateDelivery( ) function, 245, 247, 248
calculatePrice( ) function, 242
calling a function, 194
camelCase, 35
capitalization, 29, 30, 106
capital letters, for words in variable names, 34
cascading
  defined, 101
  understanding, 101
Cascading Style Sheets (CSS)
  basics of, 90–91
  changing with JavaScript, 105–109
Function Junction, 197–200
Martian Rescue! game, 272–276
resizing elements with, 98–101
Word Replacement Game, 160
cases, switch statements, 253
central processing unit (CPU), 9
checkPosition( ) function, 203–204
Chrome Developer Tools
class selectors, 92
Clear Console Log button, 218
clearInterval statement, 117
code
  curly brackets used for grouping pieces of, 30
forking
  Activity Calendar program, 256–257
defined, 63
JavaScript Pizzeria, 241
Lemonade Stand game, 334–335
Lunch Game, 313
Martian Rescue! game, 271
Super-Calculator, 138–139
Wish List program, 208–209
JavaScript. See also Martian Rescue! game; Wish List program
array methods, 177–178
changing CSS with, 105–109
changing HTML with, 83–87
creating animations with, 115–117
elements of websites that use,
13–14
Function Junction, 200–204
Lemonade Stand game, 335
overview, 12–13
Word Replacement Game, 163–165
text in, 29–30
coding (computer programming), 8
color names, HTML, 58–59
colors, CSS, 96–97
color swatch, hexadecimal value,
58–59
comments
   making, 31–32
   multi-line, 32
   single-line, 32
comparison operators, 137, 145–150
compilation, defined, 10, 23
compilers, defined, 8
custom functions, 191–193
data. See also input
   prompting the user for input, 42–43
data types, 38–42
concat( ) array method, 178, 181
concatenation, 20
concatenation/assignment operator
   (+=), 164
concatenation operator, 137, 141,
142, 153, 164
condition statement, for loops, 296
continueStory( ) function, 284, 285,
288–290
countdown, 297–298
CPU (central processing unit), 9
createElement( ) method, 220
currentPosition variable, 203, 204
customers, Lemonade Stand game,
330
custom functions, 191–193
data types (continued)
in arrays, 175
Boolean, 41–42
combining two different, 41
definition of, 38
number, 40–41
of operands, 126–129
string, 38–40
dayOfWeek variable, 263
debugging, origin of term, 8
declaration block, CSS rule, 91
defining a function, 193–194
Developer Tools, Chrome, 16–18
clearing, 20
experimenting with, 21
introduction to, 18
overview, 17
potential to be misused, 18
running your first commands, 19
displayActivity() function, 262, 263
displayDate() function, 262
display:none, turning off elements with, 273–275
displayResults() function, 345, 346, 350
<div> elements, 79, 157, 158, 161, 163, 211
document, 47
document.write(), 47, 49
Double Quotes within Single Quotes, 29
Douglas the JavaScript Robot, 90–103
animating
adding a second animation function, 118–122
animating another element, 118
arm sweep animation, 118–120
changing CSS with JavaScript, 105–106
creating animations with JavaScript, 115–117
experimenting with Douglas, 108–109
handling events, 111–113
making Douglas dance, 109–111
modifying Douglas with JavaScript, 106–108
writing a listener, 113–115
cascading, 101
CSS colors, 96–97
CSS declarations, 93
CSS properties, 93–96
CSS selectors, 91–93
customizing your own robot, 103
positioning elements with CSS, 101–102
resizing elements with CSS, 98–101
responsive design, 98
dream car (dreamCar object)
color of the car, 135
configuring, 132–135
creating, 131
CSS pane, 133
customizing, 134–135
generic car, 133–134
HTML pane, 132
JavaScript pane, 134
make and model of the car, 134
model year on the price tag, 135
price tag, 133–135
elements
  adding attributes to, 81–83
  positioning with CSS, 101
  resizing with CSS, 98
  element selectors, 91
  <em> element, 78
empty arrays, 278
ending tags, 73
equality operator (==), 145–148, 234
escaping quotation marks, 27–29
event handler
  addEventListener method, 111–112
defined, 86
event listeners
  Listing 13-2, 214
  Martian Rescue! game, Martian Rescue! game, JavaScript, 279
  Wish List program, 212–214
events
  defined, 111
  examples of, 111
  handling, 111–113
  listening for, 112
  in web browsers, 111–112
  event target, 112
  eye bounce, 110, 113, 115
final expression, for loops, 297
float:left property, 161
flow charts, 268–269
font-family style, 160
font size, 94
<footer> tag, 74
Fork button, 63
forking the code
  Activity Calendar program, 256–257
defined, 63
  JavaScript Pizzeria, 241
  Lemonade Stand game, 334–335
  Lunch Game, 313
  Martian Rescue! game, 271
  Super-Calculator, 138–139
  Wish List program, 208–209
for loops, 295–308
  parts of, 296–297
random weather forecasting
  id and class attributes, 305–307
  Listing 17-5 The Finished JavaScript Code, 303–304
  Math.random() function, 299–301
  styling the app, 307–308
  writing the app, 301–303
sample, 296
  writing and using, 297–298
forward slash (/), 20
frames, in an animation, 115
function body, 194
function head, 194
Function Junction, 196–205
  CSS pane, 197–200
  HTML for, 197
  JavaScript pane, 200–204
function keyword, 193
functions
- built-in, 191
- calling, 194
- custom, 191–193
- defining, 193–194
- defining parameters, 194–195
- example of, 192
- overview, 190–191
- returning a value, 195–196
- Wish List program, 215–227
  - addTheThing(), 216, 222
  - addToTheList(), 216, 219–222
  - printView(), 217–218, 224, 226
  - resetInput(), 222–224
- Function technique, 194

global variables
- Lemonade Stand game, 335
- Martian Rescue! game, 278
- Wish List program, 214–215
- Google Chrome, 14–16
  - Developer Tools, 16–18
  - installing, 16
- graphing sales, temperature, and price, Lemonade Stand game, 331–333
- greater than operator ( > ), 149, 151
- greater than or equal to operator ( >= ), 149–150

hash symbol (#)
- for hexadecimal code, 97
- for ID selectors, 92
- head element, 75
- headers, 76
- <header> tag, 74
- hexadecimal notation, 97
  - for CSS colors, 58
- Hopper, Grace Murray, 8
- <hr> element, 79
- <h1> element, 76
- <h1> through <h6> elements, 78–79
- HTML (Hypertext Markup Language), 72–88
  - Bubbles demo, 60–61
  - changing with JavaScript, 83–87
  - definition of, 72
  - elements of, 78–81
  - for Function Junction, 197
- getElementsByld, 84
Martian Rescue! game, 272–276
tags in, 61, 72–75
text without, 72
for Wish List program, 210–212
for Word Replacement Game, 156
<html> and </html> tags, 75, 77
HTML attributes, 81–83
HTML elements, defined, 73
HTML pane
creating your first web page, 76–77
dream car, 132
JSFiddle, 60

innerHTML method, 84–87
innerHTML property, 220
input
combining output and, 48–50
defined, 33
prompting the user for, 42–43
responding to, 44–46
storing user input, 43–44
inputToReset variable, 224
Interactive Sock Puppet, 14

JavaScript code
array methods, 177–178
changing CSS with, 105–109
changing HTML with, 83–87
creating animations with, 115–117
examples of websites that use, 13–14
Function Junction, 200–204
Lemonade Stand game, 335
Martian Rescue! game
askQuestion() function, 280–283
continueStory() function, 284, 285, 288–290
element shortcuts, 278
empty arrays, 278
getAnswer() function, 283, 284, 286
Listing 16-4 The Starter JavaScript for Martian Rescue!, 277
theEnd() function, 290–291
writing the functions, 281–284
overview, 12–13
Wish List program, 212–229

ID attributes, 82–83
ID selectors and, 92–93
selecting an element with
getElementById and, 84
ID selectors, 92–93
if...else statements
branching, 267
to choose between two paths, 237
introduction to, 236–238
and operator (&&) in, 239
variables without operators,
237–238
<img> tag, 73
increment operator, 117
indentation, 31
indexOf() array method, 178,
181–182
indexOf() method, 40
inequality operator (!=), 148
init() function, 323–324
initialization statement, for loops,
296
JavaScript code (continued)
  declaring global variables, 214–215
  event listeners, 212–214
Word Replacement Game code,
  163–165
JavaScript Console (Developer Tools
  Console)
  clearing, 20
  experimenting with, 21
  introduction to, 18
  potential to be misused, 18
  running your first commands, 19
JavaScript pane, JSFiddle, 61–63
JavaScript Pizzeria, 240–251
  adding the new item to the menu,
    241–243
  delivering to other cities, 243–245
  displaying the delivery fee, 245–246
  forking the code, 241
  planning improvements, 241
  programming the birthday special,
    246–248
  running current version of the
    website, 240
join( ) array method, 178, 182–183
JSFiddle
  Activity Calendar program, 256–257
  array methods, 178–179
  creating an account, 63–65
  CSS pane, 55–60
  HTML pane, 60
  introduction to, 52–53
  JavaScript pane, 61–63
  Log In page, 63–64
  overview, 51
  playing with fiddles, 54–55
  public dashboard, 53
  creating your own public
    dashboard, 67–68
  running your first JSFiddle
    program, 53
  saving your app, 67–68
  user interface, 52
  viewing fiddles, 53–54
  website, 52

k
kilobyte, 10

L
<label> element, 210
languages, programming
  choosing, 11–12
  examples of, 11–12
  function of, 10
lastIndexOf( ) array method,
  183–184
Lemonade Stand game, 326–353
  building the game, 334–351
  creating globals, 335–337
  forking the code, 334–335
  writing the JavaScript, 335
  business lesson, 329
  displaying a report, 345–346
  finishing and testing the program,
    347–351
  generating weather, 337–339
  graphing sales, temperature, and
    price, 331–334
  improving, 352–353
  making a profit, 329–330
open the stand, 341–343
playing the game, 327–329
resetting the program, 344–345
understanding the math, 330–331
understanding your customers, 330
<.>length, 38
length of a string, 38–39
less-than operator (<), 150
less-than-or-equal-to operator (<=), 150
<li> element, 76, 79, 157, 158, 161, 162
line breaks, 30–31
listener
  overview, 112
  writing a, 113–115
Listing 2-1 A JavaScript Statement, 24
Listing 2-2 A Program to Print a Message 300 Times, 26
Listing 2-5 White Space Makes Programs Easier to Read, 30
Listing 2-6 Single-Line Comments, 32
Listing 2-7 Multi-Line Comment, 32
Listing 5-1 A List, 72
Listing 5-2 A Simple HTML Document, 74
Listing 5-3 Updated Home Page, 81
Listing 5-4 Updated Home Page with id Attributes Added, 83
Listing 5-5 Final JavaScript for the HTML Homepage App, 87-88
Listing 7-1 Code Required in the JavaScript Pane to Enable the Eye Bounce, 115
Listing 7-2 Finished moveRightLeft Function, 120
Listing 7-3 JavaScript Required to Implement Both of Douglas’s Dance Moves, 121
Listing 10-1 Completed Markup in the HTML Pane, 168
Listing 10-2 Completed Code in the JavaScript Pane, 168-170
Listing 12-1 Function Junction CSS, 198
Listing 12-2 JavaScript Pane with Comments Indicating What to Do, 200
Listing 13-1 HTML for the Wish List Program, 212
Listing 13-2 Event Listeners, 214
Listing 13-3 Event Listeners and Global Variables, 215
Listing 13-4 Finished addTheThing( ) Function, 216
Listing 13-5 Finished addToTheList( ) Function, 222
Listing 13-6 Finished resetInput( ) Function, 224
Listing 13-7 Finished printView( ) Function, 226
Listing 14-1 Using if. . .else, 236
Listing 14-2 Using Single-Word Operators, 238
Listing 14-3 Pizza Parlor Free Delivery Rule in JavaScript, 239
Listing 14-4 Free Delivery on Your Birthday, 239
Listing 14-5 The Completed JavaScript Pizzeria Program, 248-250
Listing 14-6 The Final HTML, 250-251
Listing 15-1 Produce Different Results for Different Input, 254
Listing 15-2 The Starting JavaScript for the Activity Calendar, 261-262
Listing 15-3 The Finished Program, 264-266
Listing 16-1 The Beginning of the HTML, 272
Listing 16-2 The CSS for Martian Rescue!, 273-274
Listing 16-3 The First storyPart div, 275-276
Listing 16-4 The Starter JavaScript for Martian Rescue!, 277
Listing 16-5 The Beginning of the JavaScript, 280
Listing 16-6 The Finished askQuestion( ) Function, 282-283
Listing 16-7 The Finished getAnswer( ) Function, 284
Listing 16-9 The theEnd( ) Function, 291
Listing 17-1 JavaScript Countdown, 297
Listing 17-2 Outputting Array Values with for, 298
Listing 17-3 A Random Number Alert, 299-300
Listing 17-4 Finding a Random Friend, 301
Listing 17-5 The Finished JavaScript Code, 303-304
Listing 18-1 Logging Hello, JavaScript, 311
Listing 18-3 Looping through a List of Names, 312
Listing 18-4 The Starting Point for buyLunches, 314
Listing 18-5 A Standard HTML Template, 323
Listing 18-6 Finishing the init( ) Function, 324
Listing 19-1 Create Comments for Variables, 335-336
Listing 19-2 The Globals Have Been Created, 337
Listing 19-3 The Completed Globals and the generateWeather Function, 340-341
Listing 19-4 The openTheStand Function, 343-344
Listing 19-5 The resetForm( ) Function, 345
Listing 19-6 The Final displayResults Function, 346-347
Listing 19-7 The Lemonade Stand Program, 348-351
lists changing, 85–88
ordered, 76
unordered, 76
local variables, 214
logical operators, combining comparisons with, 238–239
loops, 295–308
for. See for loops
while
example of, 310
introduction to, 309
Lunch Game, 312–317
writing, 310–311
Lovelace, Ada, 8
Lunch Game, 312–317
  buyLunches() function, 313, 314, 323
coding, 312–317
forking the code, 313
trying it out, 317
web hosting, 318–325

machine language
  compilation into, 10
  purpose of, 10
Martian Rescue! game, 269–292
  forking the code, 271
HTML and CSS, 272–276
JavaScript code
  askQuestion() function, 280–283
  continueStory() function, 284, 285, 288–290
element shortcuts, 278
empty arrays, 278
getAnswer() function, 283, 284, 286
Listing 16-4 The Starter JavaScript for Martian Rescue!, 277
theEnd() function, 290–291
writing the functions, 281–284
looking at (or not looking at) the story parts, 275–276
playing the game, 269–271
turning off elements with display:none, 273–275
writing the story, 269
Math.floor() function, 300
Math.random() function, 299–301
megabyte, 10
methods
  array, 177–178
defined, 39
getter, 258–259
minus sign, 20
modulo operation, 137, 143
moveRightLeft function, 119, 120
moveUpDown listener function, 113, 118, 119
multi-line comments, 32
myElement variable, 278
myListArea variable, 215, 222

name/value pair, 82
newList function, 86
newListItem variable, 220, 221
number data type, 40–41

objects
  creating, 130–131
  overview, 46
  working with, 130–132
<ol> and </ol> tags, 75, 79
ol element, 76
openTheStand() function, 341, 343–345
operands, 126
data types of, 126–129
operands (continued)
  object type, 130–131
  overview, 126
Super-Calculator, 139, 140
variables as, 127–128
Operate button, 140–144, 146–151
operator
  addition, 140
  assignment, 43
  concatenation, 137, 141, 142, 153, 164
  increment, 117
operators, 136–152
  arithmetic, 140–143
  comparison, 137, 145–150
  defined, 136
  greater than (>), 149
  greater than or equal to (>=), 149–150
  inequality (!=), 148
  less-than (<), 150
  less-than-or-equal-to (<=), 150
  logical, combining comparisons with, 238–239
  strict equality (===), 147, 234
  strict inequality (!=), 148–149
  variables without, 237–238
ordered lists, 76
or operator (||), 239
output
  combining input and, 48–50
  defined, 33

<p>and </p> tags (<p> element), 61, 91
percent, as units of measurement, 98–100
period (.), for class selectors, 92
pixels, as units of measurement, 98
placeOrder() function, 245, 247, 248
pop() array method, 178, 184–185
programming languages
  choosing, 11–12
  examples of, 11–12
  function of, 10
prompt command, 42
property, 39
  in CSS declarations, 93
public dashboard, 53
  creating your own, 67–68
push() array method, 178, 185–186, 220
print dialog box, browser’s, 227–228
printing, program that prints out the words “Coding is fun!” 300 times, 25–26
print() method, 227
printView() function, 217–218, 224, 226, 227
Profit, Lemonade Stand game, 329–330
program(s)
  that prints out the words “Coding is fun!” 300 times, 25–26
  things all programs have in common, 33
programming (coding), 8
passing an argument, 195
parameters, 194–195

### Q

quotes (quotation marks)
- Double Quotes within Single Quotes, 29
- escaping, 27–29
- inside a string, 27
- single, 28–29
- types of, 28–29
- use of, 20

### R

radio buttons, Super-Calculator, 139
random weather forecasting, 299–308
id and class attributes, 305–307
Listing 17-5 The Finished JavaScript Code, 303–304
Math.random() function, 299–301
styling the app, 307–308
writing the app, 301–303
rectangles, elements in an HTML document as, 98
remainder (modulo operation), 143, 144
replaceButton variable, 163
Replace It button, 155–157, 165
reserved words, 35
resetForm() function, 314, 341, 344, 345
resetInput() function, 222–224
resetting the program, Lemonade Stand game, 344–345

### S

scientific notation, 144–145
selecting an element with
- getElementById, 84
selectors
- class, 92
- CSS, 91–93
- CSS rule, 91
- element, 91
- ID, 92–93
- semicolon (;)
  - in CSS declarations, 93
  - to separate statements, 25, 31
- setDate() method, 260
- setDay() method, 260
- setFullYear() method, 260
- setHours() method, 260
- setInterval command, 116, 117, 203
- setMilliseconds() method, 260
- setMonth() method, 260
- setSeconds() method, 260
- setter methods, 260

resizing elements with CSS, 98–101
responding to input, 44–46
responsive design, 98–99
return statement, 195
return value, 195
defined, 19
reverse() array method, 178, 186
robotPart variable, 116
rules, 25–27
setTime() method, 260
shift() array method, 178, 187
ShinyText, 13
single-line comments, 32
slice() array method, 178, 188
software, defined, 8
sort() array method, 178, 188–189
spaces, in strings, 39
spacing, 25
<span> element, 164
speaksJavaScript variable, 238
speedUp() function, 202
spelling, 25, 29
splice() array method, 178, 189
src attribute, 82
statements
curly brackets for, 30
defined, 24
example of, 24–25
indenting, 31
semicolon (;) for marking the end of, 31
semicolon used to separate, 25
writing, 125
stopTrain() function, 204
storyDiv variable, 164
.<storyPart selector, 276
strict equality operator (===), 147, 234
strict inequality operator (!=), 148–149
string data type, 38–40
strings
length of, 38
text in, 27–29
<strong> element, 79
Super-Calculator
comparison operators, 145–150
forking, 138
introduction to, 137
name of, 138
strings and arithmetic operators, 141–145
using, 139–141
switches, overview, 9
switch statements, 252–266
branching, 267
defined, 252
syntax for, 253
writing, 253–255
syntax, 22–32
comments, 31–32
definition of, 22
following rules, 25–27
making a statement, 24–25
text in code, 29–30
text in strings, 27–29
white space, 30–31
syntax error, defined, 20
tags
beginning, 73
ending, 73
HTML, 61, 72
order in which tags are opened and closed, 75
tempToday() variable, 303

text
in code, 29–30
in strings, 27–29
text sizes, specifying, 94
theEnd() function, 290–291
theStory variable, 164
TidyUp button, 165, 211
timerDelay, 62, 63
toFixed() method, 316
top variable, 116, 117
toString() array method,
   178, 180–181
trainPosition variable, 203–204
trainSpeed variable, 202–203
transistors, in general, 9
Twitter, sharing your programs
   on, 65
type conversion, 146–147
typeface, changing, 160
typeof command, 129–130

variables
   creating, 34–35
definition of, 34
global, 278
   Lemonade Stand game, 335
   Martian Rescue! game, 278
   Wish List program, 214–215
inside arrays, 176
local, 214
naming, 34–35
storing data in, 36–38
without operators, 237–238
var keyword, 34
<video> tag, 73

weather, generating, Lemonade
   Stand game, 337–341
weather forecast app
   id and class attributes, 305–307
   Listing 17-5 The Finished JavaScript
      Code, 303–304
   Math.random() function, 299–301
   styling the app, 307–308
   writing the app, 301–303
weather forecasts, Lemonade Stand
game, 327
weatherToday() variable, 302
web application (near web app),
   definition of, 51
web browsers
   events in, 111–112
   overview, 14–16
web hosting, 318–325

U
<ul> element, 76, 79
unordered lists, 76
unshift() array method,
   178, 187
updateList function, 86, 87

V
valueOf() array method, 180–181
values
   of array elements, 176
   changing, 176–177
   in CSS declarations, 93
   variable declarations, 34
web pages
  basic structure of, 75–76
  body element, 76
  creating your first, 76–78
  as documents, 47
  head element, 75
  h1 element, 76
  li element, 76
  ol element, 76
websites, overview, 51
while loops
  example of, 310
  introduction to, 309
  Lunch Game, 312–317
  writing, 310–311
white space, 30–31
Wish List program, 206–229
  browser’s Print dialog box, 227–228
  enhancing the Wish List, 228–229
  Forking the code, 208–209
functions
  addTheThing(), 216, 222
  addToTheList(), 216, 219–222
  printView(), 217–218, 224, 226
  resetInput(), 222–224
HTML for, 210–212
  introduction to, 207–208
JavaScript code for, 212–229
  declaring global variables, 214–215
  event listeners, 212–214
  running with incomplete functions, 217
  viewing the finished program, 207–208
women programmers, 8
Word Replacement Game, 153–170
  button area, 156
  creating a story for, 154
  creating the game, 154–160
  writing the HTML, 156
  finishing the program, 166–170
JavaScript code, 163–165
question area, 156
story area, 157
styling, 160–163
words, reserved, 35
writing statements, 125

X

x10Hosting, 319–321

Y

youShould variable, 263

Z

zero-based numbering, in general, 40
About the Authors

**Chris Minnick:** Chris is a JavaScript superhero who is known for being able to solve any problem. He enjoys swimming, writing, and playing guitar.

**Eva Holland:** Eva’s superpower is her ability to get things done. She is known throughout the land as the Facilitator. She enjoys dancing, writing, and dressing up for parties.

Dedication

This book is dedicated to kids from 0 to 1100100.

Authors’ Acknowledgments

We’d like to give special thanks to everyone who advised us and helped with testing and suggestions, including: Camille McCue, Ivy Jackson, Beth Burkhart, Marek Belski, Stephen Tow, Carole Jelen, our readers and students, our families and friends, our social media followers, our awesome team at Wiley, and kids everywhere who do cool things and inspire us to keep learning.
Math & Science
Algebra I For Dummies, 2nd Edition
978-0-470-55964-2
Anatomy and Physiology For Dummies, 2nd Edition
978-0-470-92326-9
Astronomy For Dummies, 3rd Edition
978-1-118-37697-3
Biology For Dummies, 2nd Edition
978-0-470-59875-7
Chemistry For Dummies, 2nd Edition
978-1-118-00730-3
1001 Algebra II Practice Problems For Dummies
978-1-118-44662-1

Microsoft Office
Excel 2013 For Dummies
978-1-118-51012-4
Office 2013 All-in-One For Dummies
978-1-118-51636-2
PowerPoint 2013 For Dummies
978-1-118-50253-2
Word 2013 For Dummies
978-1-118-49123-2

Music
Blues Harmonica For Dummies
978-1-118-25269-7
Guitar For Dummies, 3rd Edition
978-1-118-11554-1
iPod & iTunes For Dummies, 10th Edition
978-1-118-50864-0

Programming
Beginning Programming with C For Dummies
978-1-118-73763-7
Excel VBA Programming For Dummies, 3rd Edition
978-1-118-49037-2
Java For Dummies, 6th Edition
978-1-118-40780-6

Religion & Inspiration
The Bible For Dummies
978-0-7645-5296-0
Buddhism For Dummies, 2nd Edition
978-1-118-02379-2
Catholicism For Dummies, 2nd Edition
978-1-118-07778-8

Self-Help & Relationships
Beating Sugar Addiction For Dummies
978-1-118-54645-1
Meditation For Dummies, 3rd Edition
978-1-118-29144-3

Seniors
Laptops For Seniors For Dummies, 3rd Edition
978-1-118-71105-7
Computers For Seniors For Dummies, 3rd Edition
978-1-118-11553-4
iPad For Seniors For Dummies, 6th Edition
978-1-118-72826-0
Social Security For Dummies
978-1-118-20573-0

Smartphones & Tablets
Android Phones For Dummies, 2nd Edition
978-1-118-72030-1
Nexus Tablets For Dummies
978-1-118-77243-0
Samsung Galaxy S 4 For Dummies
978-1-118-64222-1

Samsung Galaxy Tabs For Dummies
978-1-118-77294-2

Test Prep
ACT For Dummies, 5th Edition
978-1-118-01259-8
ASVAB For Dummies, 3rd Edition
978-0-470-63760-9
GRE For Dummies, 7th Edition
978-0-470-88921-3
Officer Candidate Tests For Dummies
978-0-470-59876-4
Physician’s Assistant Exam For Dummies
978-1-118-11556-5
Series 7 Exam For Dummies
978-0-470-09932-2

Windows 8
Windows 8.1 All-in-One For Dummies
978-1-118-82087-2
Windows 8.1 For Dummies
978-1-118-82121-3
Windows 8.1 For Dummies, Book + DVD Bundle
978-1-118-82107-7

Available in print and e-book formats.
Take Dummies with you everywhere you go!

Whether you are excited about e-books, want more from the web, must have your mobile apps, or are swept up in social media, Dummies makes everything easier.
Leverage the Power

For Dummies is the global leader in the reference category and one of the most trusted and highly regarded brands in the world. No longer just focused on books, customers now have access to the For Dummies content they need in the format they want. Let us help you develop a solution that will fit your brand and help you connect with your customers.

Advertising & Sponsorships

Connect with an engaged audience on a powerful multimedia site, and position your message alongside expert how-to content.

Targeted ads • Video • Email marketing • Microsites • Sweepstakes sponsorship

21 Million Monthly Page Views & 13 Million Unique Visitors
WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.