15 Fun Coding Projects!

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®
A Wiley Brand

JavaScript® For Kids

FOR DUMMIES®

A Wiley Brand

by Chris Minnick and Eva Holland

FOR DUMMIES®

A Wiley Brand

JavaScript® For Kids For Dummies®

Published by: **John Wiley & Sons, Inc.,** 111 River Street, Hoboken, NJ 07030-5774, www.wiley.com

Copyright © 2015 by John Wiley & Sons, Inc., Hoboken, New Jersey

Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the Publisher. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permissions.

Trademarks: Wiley, For Dummies, the Dummies Man logo, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc., and may not be used without written permission. JavaScript is a registered trademark of Oracle, Inc. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: WHILE THE PUBLISHER AND AUTHOR HAVE USED THEIR BEST EFFORTS IN PREPARING THIS BOOK, THEY MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS BOOK AND SPECIFICALLY DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES REPRESENTATIVES OR WRITTEN SALES MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR YOUR SITUATION. YOU SHOULD CONSULT WITH A PROFESSIONAL WHERE APPROPRIATE. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM.

For general information on our other products and services, please contact our Customer Care Department within the U.S. at 877-762-2974, outside the U.S. at 317-572-3993, or fax 317-572-4002. For technical support, please visit www.wiley.com/techsupport.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at http://booksupport.wiley.com. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2015945282

ISBN 978-1-119-11986-9 (pbk); ISBN 978-1-119-11989-0 (ebk); ISBN 978-1-119-11988-3

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents at a Glance

Introduction 1

Part I: What Is JavaScript? Alert! JavaScript Is Awesome! 5

Chapter 1: Programming the Web 7
Chapter 2: Understanding Syntax 22
Chapter 3: Giving and Receiving Data 33
Chapter 4: Fiddling with Web Applications 51

Part II: Animating the Web 69

Chapter 5: JavaScript and HTML 71
Chapter 6: JavaScript and CSS 89
Chapter 7: Building an Animated Robot 104

Part III: Getting Operations 123

Chapter 8: Building Your Dream Car with Operands 125
Chapter 9: Putting It Together with Operators 136
Chapter 10: Creating Your Own JavaScript Word Game 153

Part IV: Arrays and Functions 171

Chapter 11: Creating and Changing Arrays 173
Chapter 12: Making It Functional 190
Chapter 13: Creating a Wish List Program 206

Part V: Freedom of Choice 231

Chapter 14: Making Decisions with the If...Else Statement 233
Chapter 15: Doing Different Things with Switch 252
Chapter 16: Choose Your Own Adventure 267

Part VI: Loops 293
Chapter 17: What's This Loop For? 295
Chapter 18: Using While Loops 309
Chapter 19: Building a Lemonade Stand 326

Index 355

Table of Contents

Introduction 1

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

Part I: What Is JavaScript? Alert! JavaScript Is Awesome! 5

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

Part II: Animating the Web 69

Chapter 5: JavaScript and HTML 71

Writing HTML 72
Knowing Your HTML Elements 78
Adding Attributes to Elements 81
Changing HTML with JavaScript 83

Chapter 6: JavaScript and CSS 89

Meet Douglas the JavaScript Robot 90
CSS Basics 90
CSS Properties Give You Style 93
Customize Your Own JavaScript Robot! 103

Chapter 7: Building an Animated Robot 104

Changing CSS with JavaScript 105
Make Douglas Dance! 109

Part III: Getting Operations 123

Chapter 8: Building Your Dream Car with Operands . . . 125

Knowing Your Operands 126
Working with Objects 130
Configuring Your Dream Car 132

Chapter 9: Putting It Together with Operators 136

Introducing the Super-Calculator 137
Super-Calculator Tricks 150

Chapter 10: Creating Your Own JavaScript Word Game 153

Creating a Variable Story 154
Creating the Word Replacement Game 154

Part IV: Arrays and Functions 171

Chapter 11: Creating and Changing Arrays 173

What Are Arrays? 174
Creating and Accessing Arrays 175
Changing Array Element Values 176
Working with Array Methods 177
Learning the Ways of Arrays 178

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

Chapter 19: Building a Lemonade Stand 326

Playing the Game 327
A Lesson in Business 329
Building the Game 334
Improving the Lemonade Game 352

Index 355

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:

```
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:

```
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.

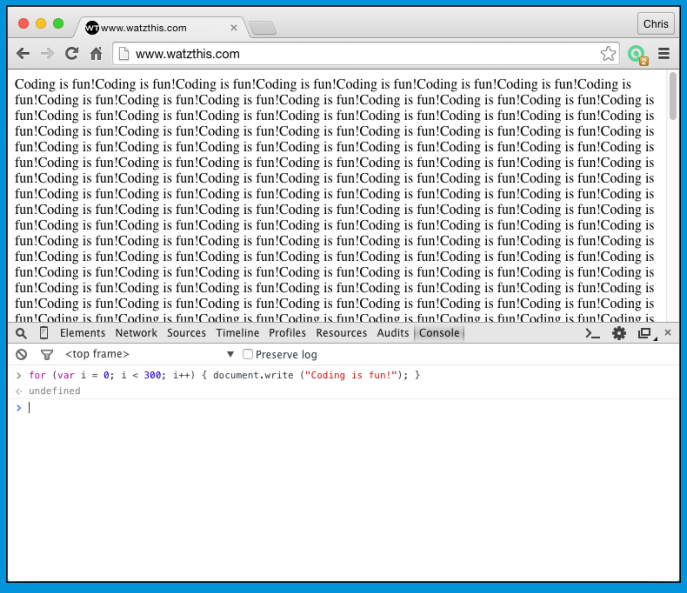
Where to Go from Here

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!*

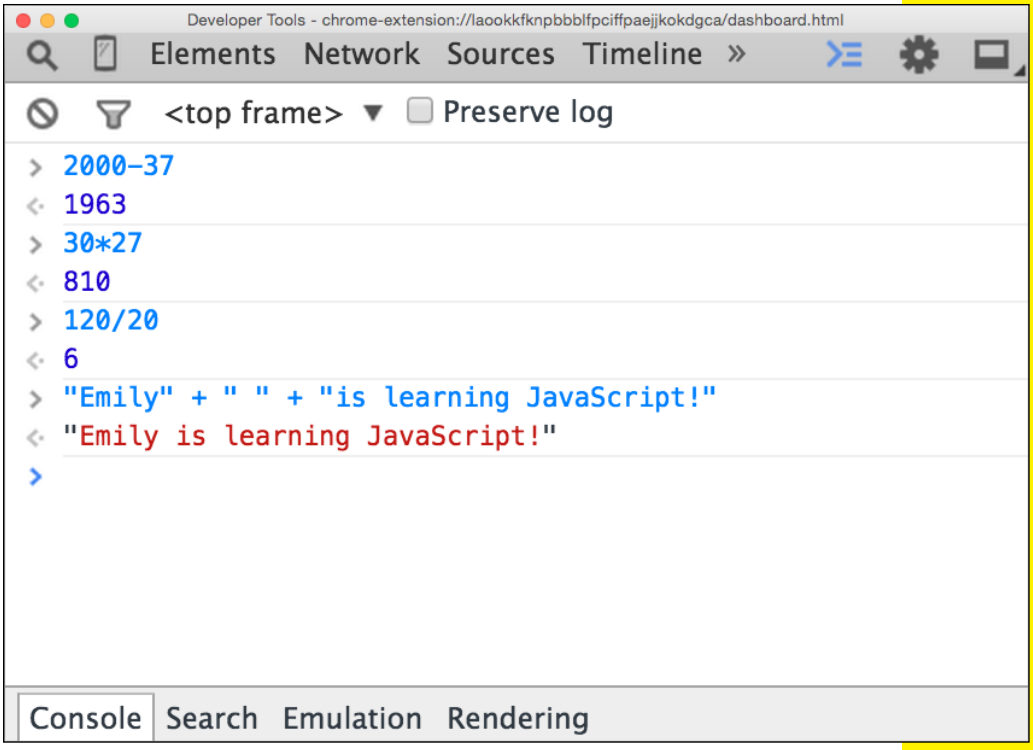
CHAPTER

1

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 1-1 How Many Bytes Is That?

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

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`):** 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.

- **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.

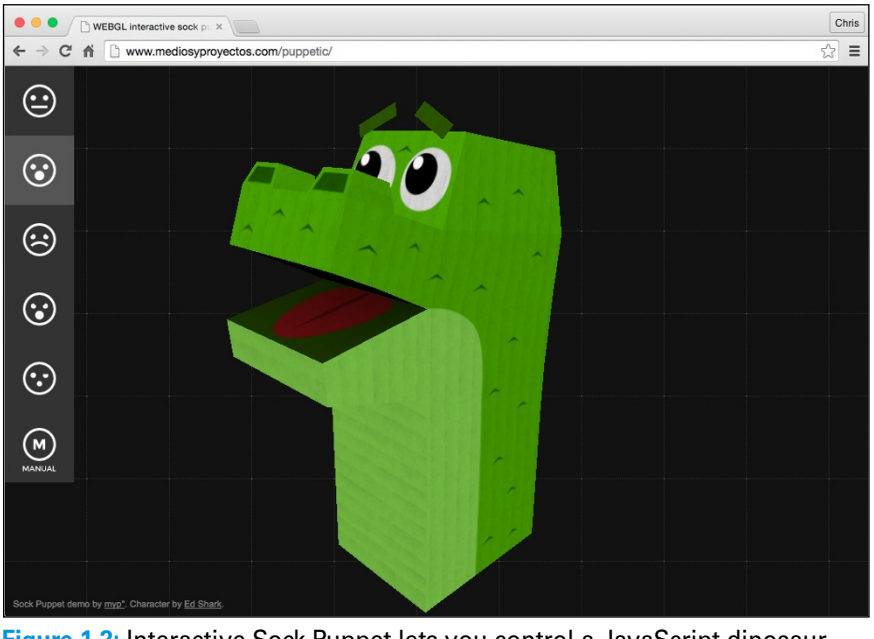
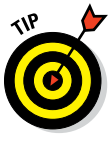


Figure 1-2: Interactive Sock Puppet lets you control a JavaScript dinosaur sock puppet.

- **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.

Figure 1-3: Facebook uses JavaScript to do everything.

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.

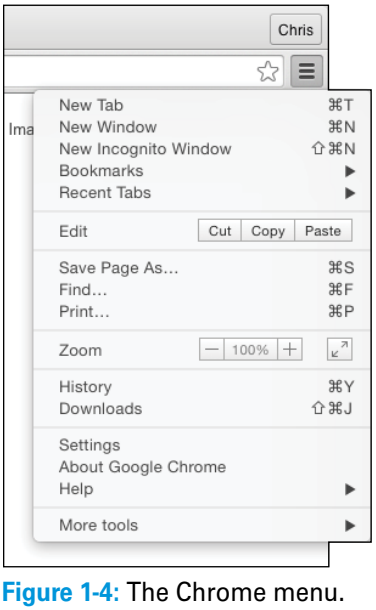


Figure 1-4: The Chrome menu.

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.

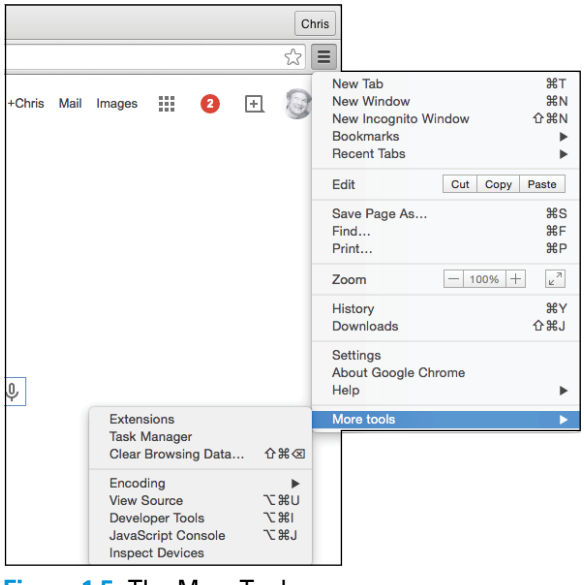


Figure 1-5: The More Tools menu.

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.

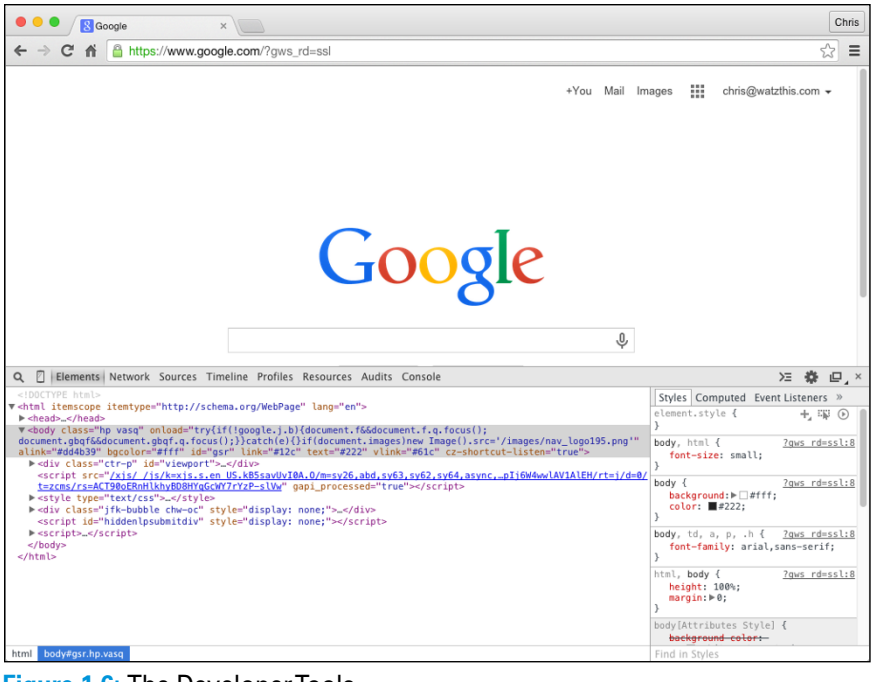


Figure 1-6: The Developer Tools.

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.

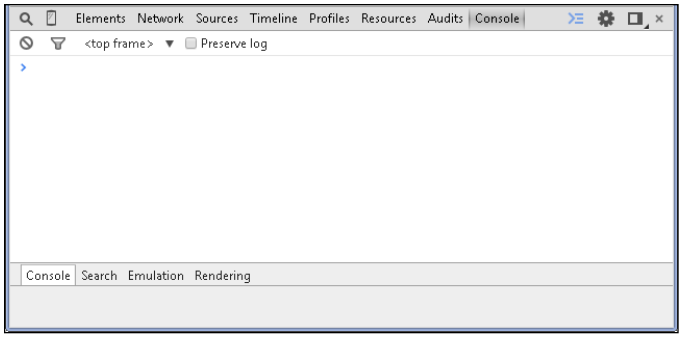


Figure 1-7: The JavaScript Console.

Running Your First JavaScript Commands

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.

JavaScript Command	Description
`2000 - 37`	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.
`30 * 27`	The asterisk (*) is how you tell JavaScript to multiply numbers.
`120 / 20`	The forward slash (/) tells JavaScript to divide the number on the left by the number on the right.
`"Your name" + " " + "is learning JavaScript!"`	Yes, you can add words together with JavaScript! When you run a command that adds words together, it's called *concatenation.* 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.
`Your name + + is learning JavaScript!`	When you don't use quotes, JavaScript doesn't like that one bit. It returns an error message containing the keyword `SyntaxError`. 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.

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 * 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

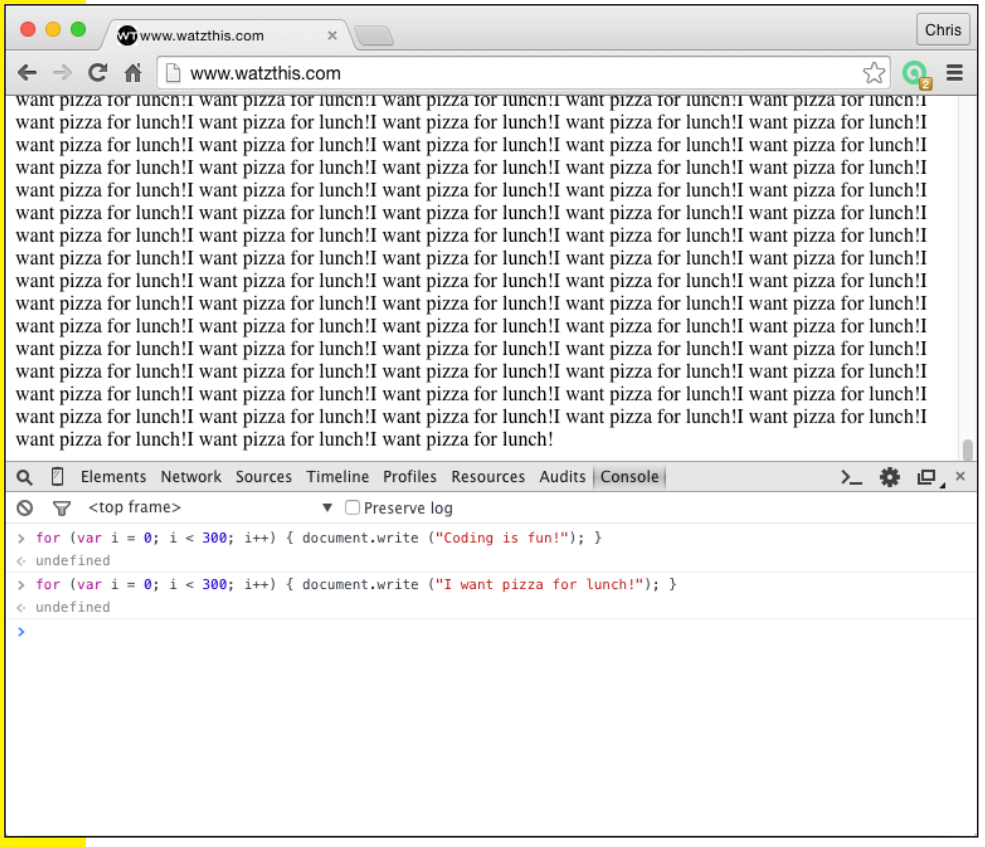
2

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`!

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

```
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.

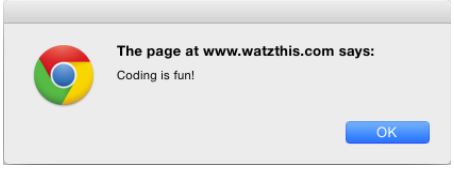


Figure 2-1: The output of a JavaScript alert statement.

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

```
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.

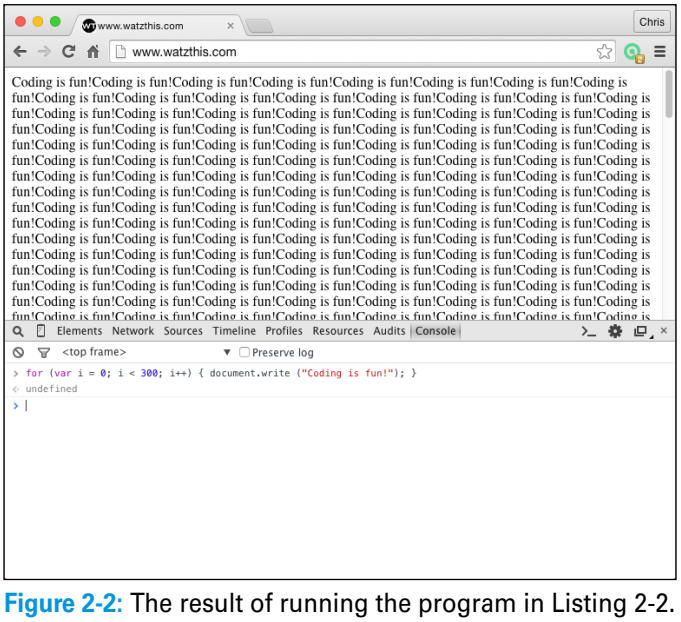


Figure 2-2: The result of running the program in Listing 2-2.

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.

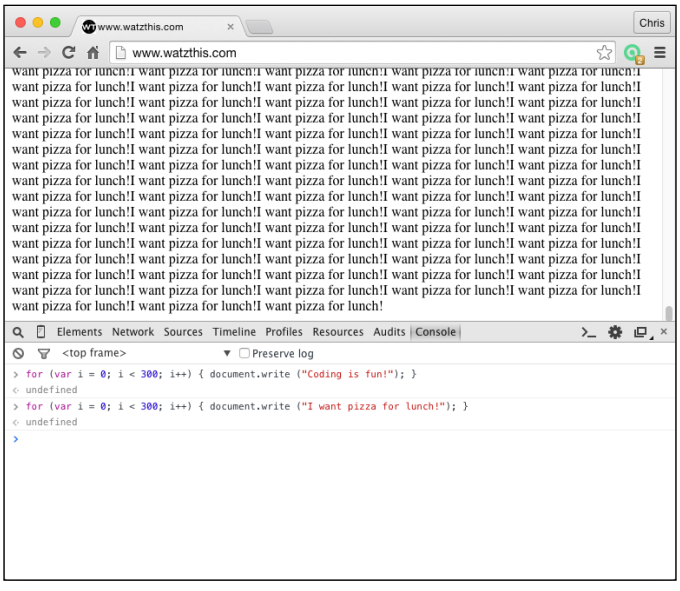
Figure 2-3 shows the output of the program from Listing 2-2 when the message is changed to "I want pizza for lunch!"

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!"
```


www.watzthis.com
Chris
www.watzthis.com
want pizza for lunch!I want pizza for lunch!I want pizza for lunch!I

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

```
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.

Figure 2-4: Capitalizing a JavaScript keyword wrong produces errors.

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

```
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

```
// 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

```
/*
   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.
*/
```

CHAPTER

3

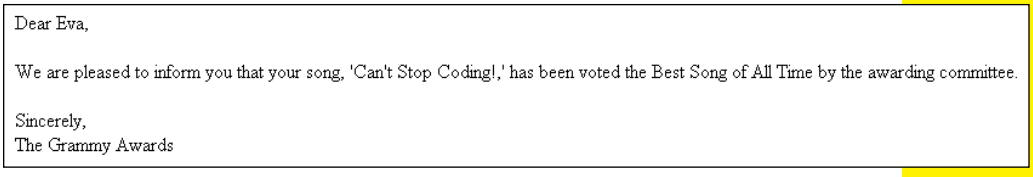
Giving and Receiving Data

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:

```
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.

```
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):

```
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.

```
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.

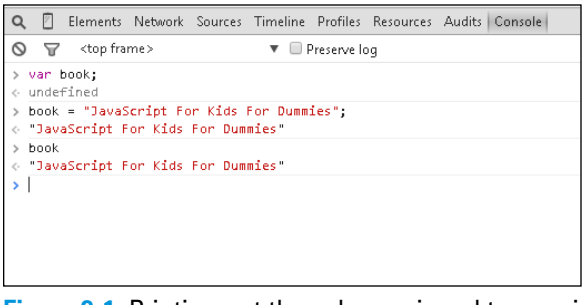


Figure 3-1: Printing out the value assigned to a variable.

   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.

Figure 3-2: Getting the length of a string.

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.

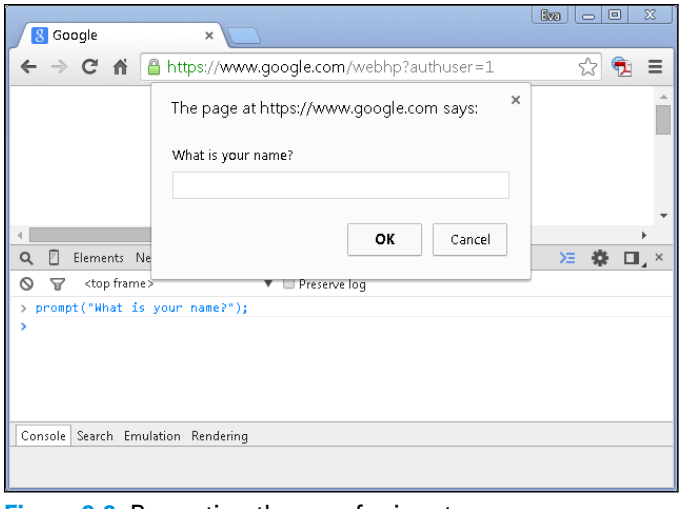


Figure 3-3: Prompting the user for input.

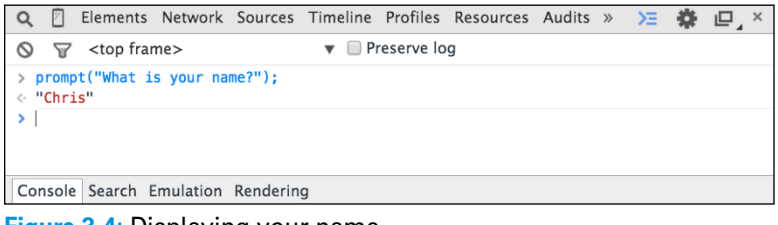


Figure 3-4: Displaying your name.

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.

```
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.

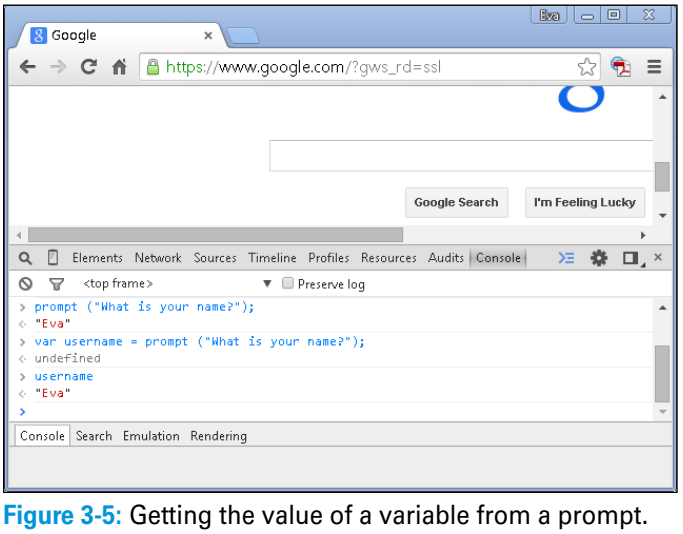


Figure 3-5: Getting the value of a variable from a prompt.

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.

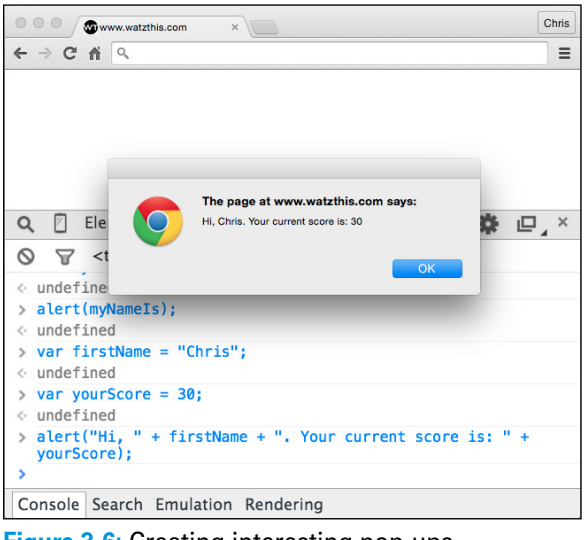


Figure 3-6: Creating interesting pop-ups.

The special life of objects

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:

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

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

```
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:

```
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:

```
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.

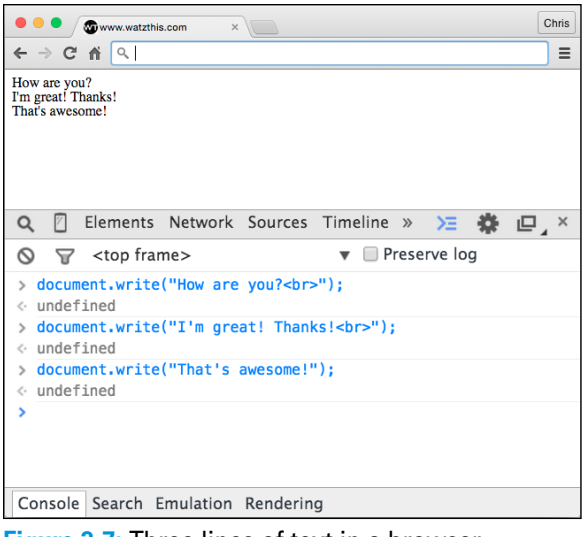


Figure 3-7: Three lines of text in a browser.

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.

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

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

   ```
   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:

```
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:

```
document.write("Dear " + toName + ",<br><br>");
document.write(letterBody + "<br><br>");
document.write("Sincerely,<br>");
document.write(fromName);
```

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

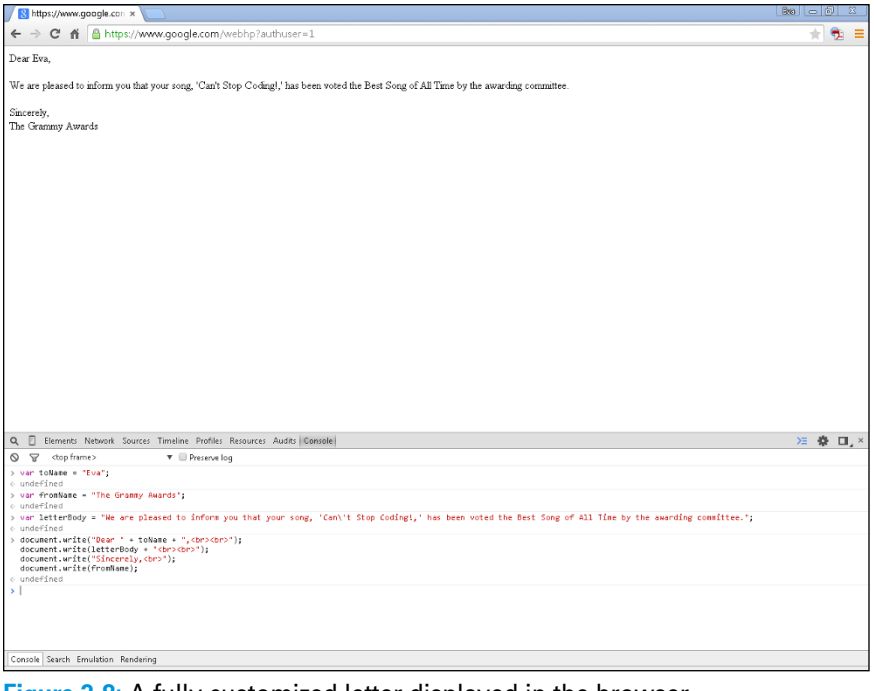


Figure 3-8: A fully customized letter displayed in the browser.

CHAPTER

4

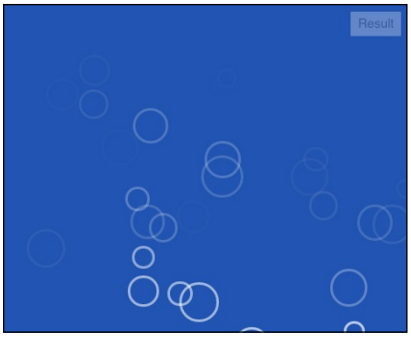
Fiddling with Web Applications

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.

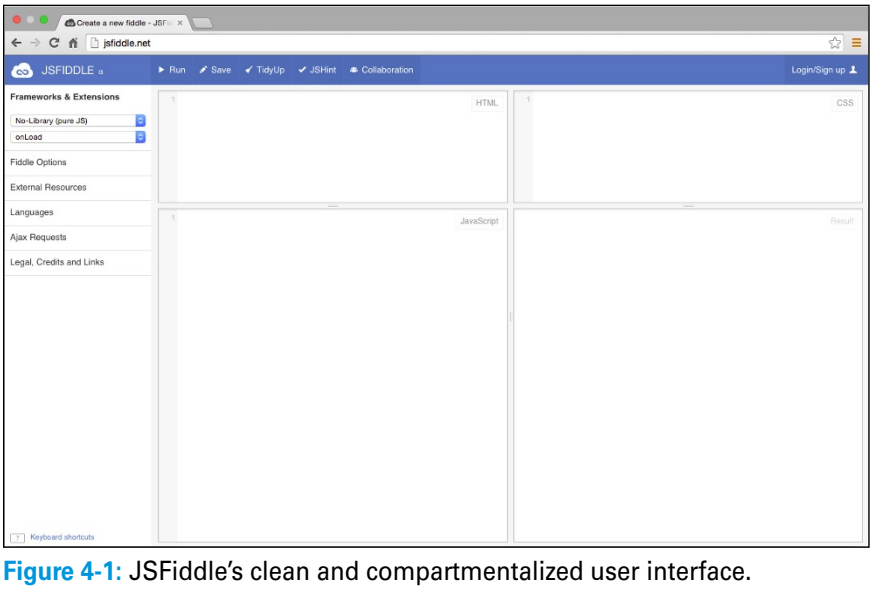


Figure 4-1: JSFiddle's clean and compartmentalized user interface.

Follow these steps to run your first JSFiddle program:

1. Click inside the JavaScript pane.
2. Type the following JavaScript statement:

```
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`. 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

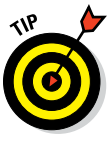
Before you get too carried away with viewing all the cool projects from other parts and chapters, take a look at some programs that aren't part of this book. JSFiddle lets anyone create an account and share their programs in a public dashboard — and many excellent and very experienced JavaScript programmers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

   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`.

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

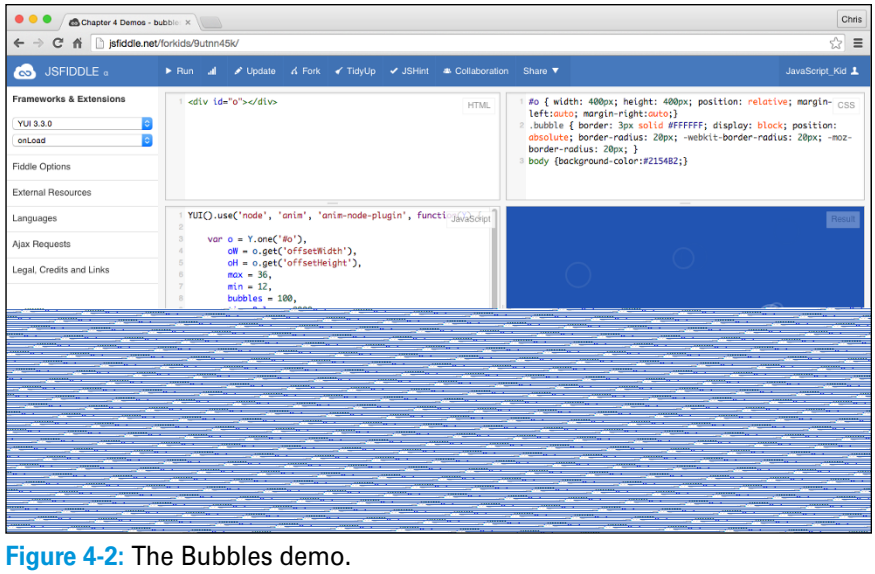


Figure 4-2: The Bubbles demo.