JavaScript

ABSOLUTE BEGINNER’S GUIDE

Kirupa Chinnathambi
This page intentionally left blank
## Contents at a Glance

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Hello, World!</td>
<td>5</td>
</tr>
<tr>
<td><strong>Part I The Basic Stuff</strong></td>
<td></td>
</tr>
<tr>
<td>2 Values and Variables</td>
<td>13</td>
</tr>
<tr>
<td>3 Functions</td>
<td>19</td>
</tr>
<tr>
<td>4 Conditional Statements: If, Else, and Switch</td>
<td>31</td>
</tr>
<tr>
<td>5 Meet the Loops: For, While, and Do...While!</td>
<td>47</td>
</tr>
<tr>
<td>6 Timers</td>
<td>59</td>
</tr>
<tr>
<td>7 Variable Scope</td>
<td>67</td>
</tr>
<tr>
<td>8 Closures</td>
<td>77</td>
</tr>
<tr>
<td>9 Where Should Your Code Live?</td>
<td>89</td>
</tr>
<tr>
<td>10 Commenting Your Code</td>
<td>101</td>
</tr>
<tr>
<td><strong>Part II It’s an Object-Oriented World</strong></td>
<td></td>
</tr>
<tr>
<td>11 Of Pizza, Types, Primitives, and Objects</td>
<td>109</td>
</tr>
<tr>
<td>12 Strings</td>
<td>121</td>
</tr>
<tr>
<td>13 When Primitives Behave Like Objects</td>
<td>133</td>
</tr>
<tr>
<td>14 Arrays</td>
<td>139</td>
</tr>
<tr>
<td>15 Numbers</td>
<td>149</td>
</tr>
<tr>
<td>16 A Deeper Look at Objects</td>
<td>161</td>
</tr>
<tr>
<td>17 Extending Built-in Objects</td>
<td>179</td>
</tr>
<tr>
<td>18 Booleans and the Stricter === and !== Operators</td>
<td>189</td>
</tr>
<tr>
<td>19 Null and Undefined</td>
<td>195</td>
</tr>
<tr>
<td>20 Immediately Invoked Function Expressions</td>
<td>201</td>
</tr>
<tr>
<td><strong>Part III Working with the DOM</strong></td>
<td></td>
</tr>
<tr>
<td>21 JS, The Browser, and The DOM</td>
<td>219</td>
</tr>
<tr>
<td>22 Finding Elements in the DOM</td>
<td>231</td>
</tr>
<tr>
<td>23 Modifying DOM Elements</td>
<td>237</td>
</tr>
<tr>
<td>24 Styling Your Content</td>
<td>247</td>
</tr>
<tr>
<td>25 Traversing the DOM</td>
<td>255</td>
</tr>
<tr>
<td>26 Creating and Removing DOM Elements</td>
<td>265</td>
</tr>
<tr>
<td>27 In-Browser Developer Tools</td>
<td>279</td>
</tr>
</tbody>
</table>
Part IV  Dealing with Events

28  Events .............................................................. 299
29  Event Bubbling and Capturing ........................................ 311
30  Mouse Events ...................................................... 325
31  Keyboard Events ................................................... 339
32  Page Load Events and Other Stuff .................................. 349
33  Handling Events for Multiple Elements ............................ 363
34  Conclusion .......................................................... 373

Glossary ................................................................. 377
Index .................................................................... 381
# Table of Contents

**Introduction** ............................................................................................................ 1

Parlez-vous JavaScript? ............................................................................................ 2

Contacting Me/Getting Help ................................................................................... 4

1 **Hello, World!** ........................................................................................................ 5

What Is JavaScript? .................................................................................................... 7

A Simple Example .................................................................................................... 8

  Code Editing Tools .............................................................................................. 8

  The HTML Document .......................................................................................... 9

Looking at the Code: Statements and Functions .................................................. 10

1 **The Basic Stuff**

2 **Values and Variables** .......................................................................................... 13

  Using Variables .................................................................................................... 14

  More Variable Stuff ............................................................................................ 15

    Naming Variables ............................................................................................ 15

    More on Declaring and Initializing Variables ................................................ 16

3 **Functions** ............................................................................................................ 19

  What Is a Function? ............................................................................................ 22

    A Simple Function .......................................................................................... 22

  Creating a Function That Takes Arguments ..................................................... 24

  Creating a Function That Returns Data .............................................................. 27

    The Return Keyword ...................................................................................... 27

    Exiting the Function Early ............................................................................. 28

4 **Conditional Statements: If, Else, and Switch** ................................................. 31

  The If/Else Statement ......................................................................................... 32

    Meet the Conditional Operators ..................................................................... 34

    Creating More Complex Expressions ............................................................... 36

    Variations on the If/Else Statement ................................................................. 38

    Phew ................................................................................................................ 39
Switch Statements ................................................................. 39
Using a Switch Statement .................................................. 39
Similarity to an If/Else Statement ..................................... 42
Deciding Which to Use ......................................................... 44

5 Meet the Loops: For, While, and Do...While! ............... 47
The for Loop ........................................................................... 49
The Starting Condition .......................................................... 51
Terminating Condition (aka Are we done yet?) .................. 51
Reaching the End ................................................................. 51
Putting It All Together .......................................................... 52
Some for Loop Examples ...................................................... 52
Stopping a Loop Early .......................................................... 53
Skipping an Iteration ............................................................. 53
Going Backwards .................................................................. 54
You Don’t Have to Use Numbers ......................................... 54
Array! Array! Array! ............................................................. 54
Oh No He Didn’t! ................................................................. 55
The Other Loops ................................................................... 55
The while Loop .................................................................... 56
The do...while Loop .............................................................. 56

6 Timers .................................................................................. 59
Meet the Three Timers .......................................................... 60
Delaying with setTimeout ..................................................... 60
Looping with setInterval ....................................................... 61
Animating Smoothly with requestAnimationFrame ............ 62

7 Variable Scope ................................................................. 67
Global Scope .......................................................................... 68
Local Scope ........................................................................... 70
Miscellaneous Scoping Shenanigans ................................. 71
Declarations Using var Do Not Support Block Scoping ....... 71
How JavaScript Processes Variables ................................. 72
Closures ............................................................................... 74

8 Closures ............................................................................. 77
Functions within Functions ................................................. 78
When the Inner Functions Aren’t Self-Contained ................. 81
9 Where Should Your Code Live? ................................................................. 89
  The Options on the Table .................................................................... 90
  Approach #1: All the Code Lives in Your HTML Document .............. 92
  Approach #2: The Code Lives in a Separate File ................................. 93
    The JavaScript File ........................................................................ 93
    Referencing the JavaScript File .................................................... 94
  So...Which Approach to Use? .......................................................... 97
    Yes, My Code Will Be Used on Multiple Documents! .................... 97
    No, My Code Is Used Only Once, on a Single HTML Document! ...... 99

10 Commenting Your Code ...................................................................... 101
  What Are Comments? ......................................................................... 102
    Single Line Comments ................................................................ 103
    Multi-line Comments .................................................................. 104
  Commenting Best Practices .................................................................. 106

II It's an Object-Oriented World ............................................................ 109
  11 Of Pizza, Types, Primitives, and Objects ...................................... 109
    Let's First Talk About Pizza .......................................................... 110
    From Pizza to JavaScript .............................................................. 113
    What Are Objects? ...................................................................... 115
    The Predefined Objects Roaming Around .................................... 117

  12 Strings ............................................................................................. 121
    The Basics ..................................................................................... 122
    String Properties and Methods ....................................................... 124
      Accessing Individual Characters ................................................. 124
      Combining (aka Concatenating) Strings .................................... 125
      Making Substrings out of Strings .............................................. 126
      Splitting a String/split .............................................................. 128
      Finding Something Inside a String .......................................... 129
      Upper and Lower Casing Strings ............................................ 130
22 Finding Elements in the DOM .......................................................... 231
    Meet the querySelector Family ........................................... 232
    querySelector .............................................................. 233
    querySelectorAll ......................................................... 233
    It Really Is the CSS Selector Syntax .................................. 234
23 Modifying DOM Elements ............................................................. 237
    DOM Elements Are Objects—Sort of! .......................... 238
    Let's Actually Modify DOM Elements ............................... 240
    Changing an Element's Text Value .................................. 242
    Attribute Values .......................................................... 242
24 Styling Your Content ................................................................. 247
    Why Would You Set Styles Using JavaScript? ............. 248
    A Tale of Two Styling Approaches ................................ 248
    Setting the Style Directly ............................................... 249
    Adding and Removing Classes Using classList ............... 250
    Adding Class Values ..................................................... 250
    Removing Class Values .................................................. 251
    Toggling Class Values .................................................... 251
    Checking Whether a Class Value Exists ..................... 252
    Going Further ............................................................. 252
25 Traversing the DOM ................................................................. 255
    Finding Your Way Around ................................................. 256
    Dealing with Siblings and Parents ................................. 259
    Let's Have Some Kids! ................................................... 259
    Putting It All Together .................................................. 261
    Checking Whether a Child Exists ................................... 261
    Accessing All the Child Elements ................................. 261
    Walking the DOM .......................................................... 262
26 Creating and Removing DOM Elements ....................................... 265
    Creating Elements ......................................................... 266
    Removing Elements ........................................................ 271
    Cloning Elements .......................................................... 273
## In-Browser Developer Tools

- Meet the Developer Tools ................................................................. 280
- Inspecting the DOM ........................................................................ 282
- Debugging JavaScript ..................................................................... 287
- Meet the Console ............................................................................. 293
- Inspecting Objects ........................................................................... 294
- Logging Messages ........................................................................... 296

## Dealing with Events

### 28 Events

- What Are Events? ............................................................................ 300
- Events and JavaScript ..................................................................... 302
  1. Listening for Events .................................................................... 302
  2. Reacting to Events ...................................................................... 304
- A Simple Example ........................................................................... 305
- The Event Arguments and the Event Type ..................................... 307

### 29 Event Bubbling and Capturing

- Event Goes Down. Event Goes Up .................................................. 312
- Meet the Phases ............................................................................. 316
- Who Cares? ..................................................................................... 319
- Event, Interrupted ........................................................................... 319

### 30 Mouse Events

- Meet the Mouse Events ................................................................. 326
  - Clicking Once and Clicking Twice .............................................. 326
  - Mousing Over and Mousing Out ............................................... 328
  - The Very Click-like Mousing Down and Mousing Up Events .... 330
  - The Event Heard Again…and Again…and Again! ...................... 331
  - The Context Menu ...................................................................... 332
- The MouseEvent Properties ......................................................... 333
  - The Global Mouse Position ....................................................... 333
  - The Mouse Position Inside the Browser .................................... 334
  - Detecting Which Button Was Clicked ....................................... 335
- Dealing with the Mouse Wheel ...................................................... 336
31 Keyboard Events ................................................................. 339
  Meet the Keyboard Events .................................................. 340
  Using These Events ........................................................... 341
  The Keyboard Event Properties ....................................... 342
  Some Examples ................................................................................................................................. 343
    Checking That a Particular Key Was Pressed .................... 343
    Doing Something When the Arrow Keys Are Pressed ........... 344
    Detecting Multiple Key Presses ..................................... 345

32 Page Load Events and Other Stuff .................................. 349
  The Things That Happen During Page Load ......................... 350
    Stage Numero Uno .......................................................... 351
    Stage Numero Dos ........................................................ 352
    Stage Numero Three ...................................................... 352
  The DOMContentLoaded and load Events ......................... 353
  Scripts and Their Location in the DOM ............................ 355
    Script Elements—Async and Defer ................................. 358
      async .................................................................................. 358
      defer ................................................................................. 359

33 Handling Events for Multiple Elements ......................... 363
  How to Do All of This ....................................................... 365
    A Terrible Solution .......................................................... 366
    A Good Solution ............................................................. 367
    Putting It All Together .................................................... 370

34 Conclusion ........................................................................ 373

Glossary .................................................................................. 377

Index ....................................................................................... 381
Dedication

To Meena!
(Who still laughs at the jokes found in these pages despite having read them a bazillion times!)

Acknowledgments

As I found out, getting a book like this out the door is no small feat. It involves a bunch of people in front of (and behind) the camera who work tirelessly to turn my ramblings into the beautiful pages that you are about see. To everyone at Pearson who made this possible, thank you!

With that said, there are a few people I’d like to explicitly call out. First, I’d like to thank Mark Taber for giving me this opportunity, Chris Zahn for patiently answering my numerous questions, and Loretta Yates for helping make the connections that made all of this happen. The technical content of this book has been reviewed in great detail by my long-time friends and online collaborators, Kyle Murray and Trevor McCauley. I can’t thank them enough for their thorough (and occasionally, humorous!) feedback.

Lastly, I’d like to thank my parents for having always encouraged me to pursue creative hobbies like painting, writing, playing video games, and writing code. I wouldn’t be half the rugged indoorsman I am today without you both.😊
About the Author

Kirupa Chinnathambi has spent most of his life trying to teach others to love web development as much as he does.

In 1999, before blogging was even a word, he started posting tutorials on kirupa.com. In the years since then, he has written hundreds of articles, written a few books (none as good as this one, of course!), and recorded a bunch of videos you can find on YouTube. When he isn’t writing or talking about web development, he spends his waking hours helping make the Web more awesome as a Program Manager in Microsoft. In his non-waking hours, he is probably sleeping...or writing about himself in the third person.

You can find him on Twitter (twitter.com/kirupa), Facebook (facebook.com/kirupa), or e-mail (kirupa@kirupa.com). Feel free to contact him anytime.
Have you ever tried learning to read, speak, or write in a language different from the one you grew up with? If you were anything like me, your early attempts probably looked something like the following:

*Sono qui per uccidere il grande drago rosso?*
Unless you are Jason Bourne (or Roger Federer), you barely survived learning your first language. This is because learning languages is hard. It doesn’t matter if you are learning your first language or a second or third. Being good at a language to a point where you are useful in a non-comical way takes a whole lotta time and effort.

It requires starting with the basics:

It requires a boatload of practice and patience. It’s one of those few areas where there really aren’t any shortcuts for becoming proficient.

**Parlez-vous JavaScript?**

Successfully learning a *programming* language is very similar to how you would approach learning a *real world* language. You start off with the basics. Once you’ve gotten good at that, you move on to something a bit more advanced. This whole process just keeps repeating itself, and it never really ends. None of us ever truly stop learning. It just requires starting somewhere. To help you with the
“starting somewhere” part is where this book comes in. This book is filled from beginning to end with all sorts of good (and hilarious—I hope!) stuff to help you learn JavaScript.

Now, I hate to say anything bad about a programming language behind its back, but JavaScript is pretty dull and boring:

```javascript
var count = 0;

function doingSomethingBoring() {
    count++;

    if (count > 10) {
        alert("Yaaaaaaaaawwwnnnnnnnn!");
    } else {
        alert("This one time, at band camp....");
    }
}
```

There is no other way to describe it. Despite how boring JavaScript might most certainly be,¹ it doesn’t mean that learning it has to be boring as well.

As you make your way through the book, hopefully you will find the very casual language and illustrations both informative as well as entertaining (infotaining!). All of this casualness and fun is balanced out by deep coverage of all the interesting things you need to know about JavaScript to become better at using it. By the time you reach the last chapter, you will be prepared to face almost any JavaScript-related challenge head-on without breaking a sweat.

¹. FYI. All grammatical snafus are carefully and deliberately placed—most of the time!
Contacting Me/Getting Help

If you ever get stuck at any point or just want to contact me, post in the forums at: forum.kirupa.com.

For non-technical questions, you can also send e-mail to kirupa@kirupa.com, tweet to @kirupa, or message me on Facebook (facebook.com/kirupa). I love hearing from readers like you, and I make it a point to personally respond to every message I receive.

And with that, flip the page—it’s time to get started!
IN THIS CHAPTER

• Learn how functions help you better organize and group your code
• Understand how functions make your code reusable
• Discover the importance of function arguments and how to use them

FUNCTIONS

So far, all of the code we’ve written contained virtually no structure. It was just...there:

```javascript
alert("hello, world!");
```

There is nothing wrong with having code like this. This is especially true if your code is made up of a single statement. Most of the time, though, that will never be the case. Your code will rarely be this simple when you are using JavaScript in the real world for real-worldy things.
To highlight this, let’s say we want to display the distance something has traveled (see Figure 3.1).

![A realistic-looking spaceship!]

**FIGURE 3.1**
The distance something has traveled.

If you remember from school, distance is calculated by multiplying the speed something has traveled by how long it took as shown in Figure 3.2.

**FIGURE 3.2**
The formula for calculating distance.

The JavaScript version of that will look as follows:

```javascript
var speed = 10;
var time = 5;
alert(speed * time);
```

We have two variables—`speed` and `time`—and they each store a number. The `alert` function displays the result of multiplying the values stored by the `speed` and `time` variables. Quick note: The `*` character (which I threw in there without warning) between two numbers indicates that a multiplication needs to take place. Anyway, as you can see, our JavaScript is a pretty literal translation of the distance equation you just saw.

Let’s say we want to calculate the distance for more values. Using only what we’ve seen so far, our code would look as follows:

```javascript
var speed = 10;
var time = 5;
alert(speed * time);
```
var speed1 = 85;
var time1 = 1.5;
alert(speed1 * time1);

var speed2 = 12;
var time2 = 9;
alert(speed2 * time2);

var speed3 = 42;
var time3 = 21;
alert(speed3 * time3);

I don’t know about you, but this just looks (as Frank Caliendo impersonating Charles Barkley would say) turrible.¹ Our code is unnecessarily verbose and repetitive. As I mentioned earlier, when we looked at variables in the previous chapter, repetition makes your code harder to maintain. It also wastes your time. This entire problem can be solved very easily by using what you’ll be seeing a lot of here—functions:

```
function showDistance(speed, time) {
    alert(speed * time);
}
```

showDistance(10, 5);
showDistance(85, 1.5);
showDistance(12, 9);
showDistance(42, 21);

Don’t worry too much about what this code does just yet. Just know that this smaller chunk of code does everything all those many lines of code did earlier without all of the negative side effects and calories. We’ll learn all about functions, and how they do all the sweet things that they do, starting...right...now!

---

What Is a Function?

At a very basic level, a function is nothing more than a wrapper for some code. A function basically

- Groups statements together
- Makes your code reusable

You will rarely write or use code that doesn’t involve functions, so it’s important that you become familiar with them and learn all about how well they work.

A Simple Function

The best way to learn about functions is to just dive right in and start using them, so let’s start off by creating a very simple function. Creating a function is pretty easy and only requires understanding some little syntactical quirks like using weird parentheses and brackets.

The following is an example of what a very simple function looks like:

```javascript
function sayHello() {
    alert("hello!");
}
```

Just having your function isn’t enough, though. Your function needs to actually be called, and you can do that by adding the following line at the end of your code block:

```javascript
function sayHello() {
    alert("hello!");
}
sayHello();
```

If you type all this in your favorite code editor and preview your page in your browser, you will see hello! displayed. The only thing that you need to know right now is that your code works. Let’s look at why the code works by breaking it up into individual chunks and looking at them in greater detail.

First, you see the `function` keyword leading things off:

```javascript
function sayHello() {
    alert("hello!");
}
```
This keyword tells the JavaScript engine that lives deep inside your browser to treat this entire block of code as something to do with functions.

After the `function` keyword, you specify the actual name of the function followed by some opening and closing parentheses, ( ):  
```javascript
function sayHello() {

    alert("hello!");
}
```

Rounding out your function declaration are the opening and closing brackets that enclose any statements that you may have inside:
```javascript
function sayHello() {
    alert("hello!");
}
```

The final thing is the contents of your function—the statements that make your function actually...functional:
```javascript
function sayHello() {
    alert("hello!");
}
```

In our case, the content is the alert function that displays a dialog box with the word `hello!` displayed.

The last thing to look at is the function call:
```javascript
function sayHello() {
    alert("hello!");
}
sayHello();
```

The function call is typically the name of the function you want to call (or invoke) followed again by parentheses. Without your function call, the function you created doesn’t do anything. It is the function call that wakes your function up and makes it do things.

Now, what you have just seen is a very simple function. In the next couple of sections, we are going to build on what you’ve just learned and look at increasingly more realistic examples of functions.
Creating a Function That Takes Arguments

Like I mentioned earlier, the previous `sayHello` example was quite simple:

```javascript
function sayHello() {
    alert("hello!");  
}
```

You call a function, and the function does something. That simplification by itself is not out of the ordinary. All functions work just like that. There are differences, however, in the details on how functions get invoked, where they get their data from, and so on. The first such detail we are going to look at involves functions that take arguments.

Let’s start with a simple example:

```javascript
alert("my argument");
```

What we have here is your `alert` function. You’ve probably seen it a few (or a few dozen) times already. As you know, this function simply displays some text that you tell it to show (see Figure 3.3).

![Figure 3.3](image)

*The text “my argument” is displayed as a result of the alert function.*

Let’s look at this a little closer. Between your opening and closing parentheses when calling the alert function, you specify the stuff that needs to be displayed. This “stuff” is more formally known as an argument. The `alert` function is just one of many functions that take arguments, and many functions you create will take arguments as well.
To stay local, within this chapter itself, another function that we briefly looked at that takes arguments is our `showDistance` function:

```javascript
function showDistance(speed, time) {
    alert(speed * time);
}
```

So, you can tell when a function takes arguments by looking at the function declaration itself:

```javascript
function showDistance(speed, time) {
    ...
}
```

Functions that don’t take arguments are easy to identify. They typically show up with empty parentheses following their name. Functions that take arguments aren’t like that. Following their name and between the parentheses, these functions will contain some information about the quantity of arguments they need, along with some hints about what values your arguments will take.

For `showDistance`, you can infer that this function takes two arguments: the first corresponds to the `speed` and the second corresponds to the `time`.

You specify your arguments to the function as part of the function call:

```javascript
function showDistance(speed, time) {
    alert(speed * time);
}
showDistance(10, 5);
```

In our case, we call `showDistance` and specify the values we want to pass to the function inside the parentheses.

```javascript
showDistance(10, 5);
```

Functions that take arguments, however, contain some information about the quantity of arguments they need in the parentheses following their name, along with some hints about what values your arguments will take. To emphasize this, let’s look at Figure 3.4.
When the `showDistance` function gets called, the 10 corresponds to the `speed` argument, and the 5 corresponds to the `distance` argument. That mapping is entirely based on order.

Once the values you pass in as arguments reach your function, the names you specified for the arguments are treated just like variable names (see Figure 3.5).

You can use these variable names to easily reference the values stored by the arguments inside your function.
NOTE If a function happens to take arguments and you don’t provide any arguments as part of your function call, provide too few arguments, or provide too many arguments, things can still work. You can code your function defensively against these cases.

In general, to make the code you are writing clear, just provide the required number of arguments for the function you are calling. Don’t complicate things unnecessarily.

Creating a Function That Returns Data

The last function variant we will look at is one that returns some data back to whatever called it. Here is what we want to do. We have our `showDistance` function, and we know that it looks as follows:

```javascript
function showDistance(speed, time) {
    alert(speed * time);
}
```

Instead of having our `showDistance` function calculate the distance and display it as an `alert`, we actually want to store that value for some future use. We want to do something like this:

```javascript
var myDistance = showDistance(10, 5);
```

The `myDistance` variable will store the results of the calculation done by the `showDistance` function. There are just a few things you need to know about being able to do something like this.

The Return Keyword

The way you return data from a function is by using the `return` keyword. Let’s create a new function called `getDistance` that looks identical to `showDistance` with the only difference being what happens when the function runs to completion:

```javascript
function getDistance(speed, time) {
    var distance = speed * time;
    return distance;
}
```

Notice that we are still calculating the distance by multiplying `speed` and `time`. Instead of displaying an alert, we return the distance (as stored by the `distance` variable).
To call the `getDistance` function, you can just call it as part of initializing a variable:

```javascript
var myDistance = showDistance(10, 5);
```

When the `getDistance` function gets called, it gets evaluated and returns a numerical value that then becomes assigned to the `myDistance` variable. That’s all there is to it.

### Exiting the Function Early

Once your function hits the `return` keyword, it stops everything it is doing at that point, returns whatever value you specified to whatever called it, and exits:

```javascript
function getDistance(speed, time) {
    var distance = speed * time;
    return distance;

    if (speed < 0) {
        distance *= -1;
    }
}
```

Any code that exists after your `return` statement will not be reached, such as the following highlighted lines:

```javascript
function getDistance(speed, time) {
    var distance = speed * time;
    return distance;

    if (speed < 0) {
        distance *= -1;
    }
}
```

It will be as if that chunk of code never even existed. In practice, you will use the `return` statement to terminate a function after it has done what you wanted it to do. That function could return a value to the caller like you saw in the previous examples, or that function could simply exit:

```javascript
function doSomething() {
    // do something
    return;
}
```
Using the `return` keyword to return a value is optional. The `return` keyword can be used by itself as you see here to just exit the function.

---

**THE ABSOLUTE MINIMUM**

Functions are among a handful of things that you will use in almost every single JavaScript application. They provide the much sought-after capability to help make your code reusable. Whether you are creating your own functions or using the many functions that are built into the JavaScript language, you will simply not be able to live without them.

What you have seen so far are examples of how functions are commonly used. There are some advanced traits that functions possess that I did not cover here. Those uses will be covered in the future…a distant future. For now, everything you’ve learned will take you quite far when it comes to understanding how functions are used in the real world.

---

**TIP** Just a quick reminder for those of you reading these words in the print or e-book edition of this book: If you go to www.quepublishing.com and register this book, you can receive free access to an online Web Edition that not only contains the complete text of this book but also features a short, fun interactive quiz to test your understanding of the chapter you just read.

If you’re reading these words in the Web Edition already and want to try your hand at the quiz, then you’re in luck – all you need to do is scroll down!
This page intentionally left blank
Symbols

* (asterisk), multiplication operator, 20, 151
[] (brackets)
array declaration, 140
property definition, 167
|| conditional operator, 35
&& (conditional operator, 35
< conditional operator, 35
<= conditional operator, 35
> conditional operator, 35
>= conditional operator, 35
{ } (curly braces) in functions, 23
-- (decrement operator), 152, 153
-= (decrement operator), 152
/ (division operator), 152
== (equality operator), 35
comparison with === strict equality operator, 192-194, 198
/ (forward slash), division operator, 151
++ (increment operator), 152, 153
!= (inequality operator), 35
comparison with !== strict inequality operator, 192-194
- (minus sign), subtraction operator, 151
%== (modulus operator), 152
/*,*/ (multi-line comments), 104
== (multiplication operator), 152
() (parentheses)
    in functions, 23
    in IIFE, 203-204
    in mathematical expressions, 151
% (percent sign), modulus operator, 151
+ (plus sign)
    addition operator, 151
    concatenation operator, 125-126
    increment operator, 152
"" (quotation marks) in strings, 11, 123
; (semicolon) in statements, 11
// (single line comments), 103-104
== (strict equality operator), 35
comparison with === strict equality operator, 192-194, 198
!= (strict inequality operator), 35

comparison with != inequality operator, 192-194

A
abs() function (Math object), 159
absolute paths, referencing JavaScript files, 94-96
absolute value, 159
accessibility tools, DOM and, 278
accessing
    all children, 261
    array values, 141-142
    characters in strings, 124-125
    Developer Tools, 280-282
acos() function (Math object), 158
add method (classList API), 250-251
addEventListener function, 302-304, 341-342
adding
    class values, 250-251
    items to arrays, 143-144
addition operator, 151
alert function, 11
arguments, 24
console.log function versus, 296
altKey property (keyboard events), 342
animation, requestAnimationFrame function, 62-64
anonymous functions, 202
appendChild function, 267-268
arguments
  for event handlers, 307-308
  for functions, 24-27
  in IIFE, 204
Array object, 117, 134
  concat method, 147
  indexOf method, 146-147
  lastIndexOf method, 146-147
  length property, 142
  pop method, 145
  push method, 143
  shift method, 145
  shuffling arrays example (extending objects), 179-180
  slice method, 146
  unshift method, 143-144
arrays
  accessing values, 141-142
  adding items to, 143-144
  declaring, 140-141
  finding items in, 146-147
  in for loops, 54-55
  merging, 147
  removing items from, 145-146
  shuffling arrays example (extending objects), 179-180
arrow key actions, 344-345
asin() function (Math object), 158
asterisk (∗), multiplication operator, 20, 151
async attribute (script tag), 358-359
atan() function (Math object), 158
attribute selectors, 234
attribute values, modifying in DOM elements, 242-244

B
block scope, 71-72
Boolean function, 190-192
Boolean Logic, 37
Boolean object, 117, 134, 190
Boolean type, 114
booleans
  initializing variables as, 189
  as primitives, 190
brackets ([])
  array declaration, 140
  property definition, 167
break keyword
  in for loops, 53
  in switch statements, 41-42
breakpoints
  removing, 293
  setting, 288
browser Developer Tools.
  See Developer Tools
browser-specific mouse properties
  See primitives
ceil() function (Math object), 157
characters in strings
  accessing, 124-125
  finding, 129-130
charAt method
  (String object), 125
charCodeAt property
  (keyboard events), 342, 343-344
child objects, 170, 258
  accessing all, 261
  checking existence, 261
  event listening on, 367-370
  mouse events and, 329
  properties, 259-260

C
calling functions, 22, 23
  within functions, 78-81
cancelAnimationFrame function, 63
canceling
  animation frames, 63
  looping timers, 62
timers, 60
capturing events phase, 316
  usage examples, 322-324
  when to use, 319
Cascading Style Sheets.
  See CSS
case (of strings), changing, 130
case statements in switch statements, 39-42
ceil() function (Math object), 157
characters in strings
  accessing, 124-125
  finding, 129-130
charAt method
  (String object), 125
charCodeAt property
  (keyboard events), 342, 343-344
child objects, 170, 258
  accessing all, 261
  checking existence, 261
  event listening on, 367-370
  mouse events and, 329
  properties, 259-260
children property, 259-260
class values
  adding, 250-251
  checking existence, 252
  removing, 251
toggling, 251
classList API, 250-252
className property, 244
clearInterval function, 62
clearTimeout function, 60
click event, 303, 326-327
clientX property
  (MouseEvent object), 334-335
clientY property
  (MouseEvent object), 334-335
cloneNode function, 273-276
cloning DOM elements, 273-276
closures, 77-78, 81-86
  outer variable references, 207-212
code collision avoidance, 206-207
Code Convention, 16
code editors, 8
code placement options, 90-92, 355-358, 360-361
  in HTML document, 92-93
  in separate file, 93-96
  which to use, 97-99
code privacy, providing, 213-216
combining strings, 125-126
comments
  best practices, 106-107
  JSDoc comments, 105
  multi-line comments, 104
single line comments, 103-104
  when to use, 102-103
complex expressions in
  if/else statements, 36-37
concat method
  Array object, 147
  String object, 126
concatenating strings, 125-126
conditional operators, 34-36
conditional statements
  if/else statements, 32-34
    complex expressions, 36-37
    conditional operators in, 34-36
    switch statement
      comparison, 42-44
      when to use, 44-45
  if/else-if/else statements, 38-39
  if-only statements, 38
  switch statements, 39-42
    if/else statement
      comparison, 42-44
      when to use, 44-45
console (Developer Tools), 293
canvasArea property (webGL)
  shader program
    uniform variables, 145-146
coreObject property (GLFW)
  global variables, 141-142
credentials property (navigator)
cssRules property (CSSRule)
  basic properties, 237
  reference, 237

css (Cascading Style Sheets)
  purpose of, 222-223
  selector syntax, 234-235
  styling directly, 249-250
CSS Zen Garden, 5-6
ctrlKey property (keyboard events), 342
curly braces ({}) in functions, 23
currentTarget property
  (Event type), 308
cursor. See mouse
custom objects, creating, 169-174

cos() function (Math object), 158
create method (Object type), 171-174
createElement method (DOM), 266
Crockford, Douglas, 16

css (Cascading Style Sheets)
  purpose of, 222-223
  selector syntax, 234-235
  styling directly, 249-250
CSS Zen Garden, 5-6
currentTarget property
  (Event type), 308
cursor. See mouse
custom objects, creating, 169-174

cos() function (Math object), 158
create method (Object type), 171-174
createElement method (DOM), 266
Crockford, Douglas, 16

D

Date object, 117, 134
Date.now() function, 83
dblclick event, 303, 327-328
debugging JavaScript, 287-293
decisions. See conditional statements
declaring
  arrays, 140-141
  functions, 22-23
  variables, 14-15, 16-17
decrementing
  for loops, 54
    with operators, 152-153
default statements, 42

How can we make this index more useful? Email us at indexes@quepublishing.com
defer attribute (script tag), 358, 359-360
defining
   methods, 168-169
   properties, 167-169
delays. See timers
detail property (DOMMouseScroll event), 336-337
Developer Tools, 279
   accessing, 280-282
   console, 293
document object, 228-229
   event paths, 312-316
   JavaScript objects
      modifying elements, 240-241
      attribute values, 242-244
text values, 242
   navigating, 256-258
   accessing all children, 261
   checking child existence, 261
   child relationships, 259-260
   parent and sibling relationships, 259
   usage examples, 262
   nodes, 225-226
   removing elements, 271-273
searching in
   CSS selector syntax, 234-235
   querySelector function, 233
   querySelectorAll function, 233-234
window object, 227
DOMContentLoaded event, 303, 353-355, 358
DOMContentLoaded event,
   303, 353-355, 358
DOMMouseScroll event, 303, 336-337
document object, 228-229
dot notation, 167
do...while loops, 56-57
declaration statements, 38-39
equality operator (==), 35
comparison with ===
strict equality operator, 192-194, 198
Event Bubbling Phase, 318
   usage examples, 322-324
   when to use, 319
Event Capturing Phase, 316
   usage examples, 322-324
   when to use, 319
event handlers, 304
   arguments, 307-308
   as functions, 304-305
   usage examples, 305-307
event targets, 314
Event type, 308
events
   definition, 300-302
   DOM paths of, 312-316
   handling. See event handlers
   interrupting, 319-321
   keyboard events
      arrow key actions, 344-345
detecting particular keys, 343-344
   list of, 340-341
   listening for, 341-342
   multiple key presses, 345-347
   properties, 342-343
   list of, 303
   listening for
      addEventListener function, 302-304
      in document object, 354
      interrupting, 319-321
      keyboard events, 341-342
      on multiple elements, 365-371
calling, 22, 23
cancelAnimationFrame, 63
ceil(), 157
clearInterval, 62
clearTimeout, 60
cloneNode, 273-276
closures, 77-78, 81-86
cos(), 158
Date.now(), 83
declaring, 22-23
definition, 22
distance calculation
event handlers, 304-305
excerpt, 21
document.writeln, 50
document.getElementById, 235
document.getElementsByTagName
DOMMouseScroll, 336-337
document classes, 61
innerHTML, 11
elements, 20
exit, 49-50
exp(), 158
files, JavaScript in, 91, 93-96
for loops, 49-50, 52
accessing array values, 142
arrays, 54-55
decrementing, 54
iterations, 51-52
missing sections, 55
non-numerical iterations, 54
skipping iterations, 53
starting condition, 51
stopping early, 53
terminating condition, 51
forward slash (/), division
operator, 151
function keyword, 22
getDistance, 27-28
getElementsByClassName, 235
getElementsByClassName
getElementsByTagName, 235
IIFE
arguments in, 204
call collision
avoidance, 206-207
creating, 203-204
hidden code, 213-216
outer variable
references, 207-212
when to use, 205-206
initializing variables in,
72-74
insertAfter, 271
insertBefore, 268-270
pow(), 158
purpose of, 201

How can we make this index more useful? Email us at indexes@quepublishing.com
querySelector, 233, 234-235, 257
querySelectorAll, 233-235, 257
random(), 159
removeChild, 271-273
removeEventListener, 309
requestAnimationFrame, 62-64
returning data, 27-28
round(), 157
scope and, 70-71, 205
setInterval, 61-62
setTimeout, 60-61
showDistance
arguments, 25-26
returning data, 27
sin(), 158
sqrt(), 158
tan(), 158

G
getAttribute method, 242-244
getDistance function, 27-28
getElementById function, 235
getElementsByClassName function, 235
getElementsByTagName function, 235
global mouse position, 333-334
global scope, 16-17, 68-69

H
handling events. See event handlers
Hello, World! example, 9-10
hidden code with IIFE, 213-216
hoisting variables, 72-74, 205
hovering. See mouseover event
HTML (Hypertext Markup Language)
purpose of, 220-222
styling, 247-248
with classList API, 250-252
directly with style object, 249-250
with JavaScript, 248
HTML documents
code placement within, 90, 92-93
CSS, purpose of, 222-223
Hello, World! example, 9-10
HTML, purpose of, 220-222
JavaScript, purpose of, 223-224
page loads
DOMContentLoaded and load events, 353-355
steps in, 350-353
parsing, 96
script tag placement, 96, 355-358, 360-361
html object, 256-257
hybrid code placement option, 91-92
Hypertext Markup Language. See HTML conditional operators in, 34-36
switch statement comparison, 42-44
when to use, 44-45
if/else-if/else statements, 38-39
if-only statements, 38
IIFE (Immediately Invoked Function Expression)
arguments in, 204
code collision avoidance, 206-207
creating, 203-204
hidden code, 213-216
outer variable references, 207-212
when to use, 205-206
incrementing with operators, 152-153
index positions, 124
index values of arrays, 142
indexOf method
Array object, 146-147
String object, 129-130
inequality operator (!=), 35
comparison with !== strict inequality operator, 192-194
Infinity keyword, 153
inheritance, 166, 170-171
initializing variables, 14-15, 16-17
as booleans, 189
in functions, 72-74
inner functions, shared variables, 81-86
insertAfter function, 271
insertBefore function, 268-270
inspecting objects, 294-295
interrupting events, 319-321

iterations
  in for loops, 51-52
  non-numerical, 54
  skipping, 53

J

JavaScript
code placement options, 90-92, 355-358, 360-361
  in HTML document, 92-93
  in separate file, 93-96
  which to use, 97-99
debugging, 287-293
definition, 7-8
  purpose of, 223-224
JavaScript Variable Name Validator, 16
.js file extension, 93-94
JSDoc comments, 105

K

keyboard events
  arrow key actions, 344-345
detecting particular keys, 343-344
  list of, 340-341
  listening for, 341-342
multiple key presses, 345-347
  properties, 342-343
keyboards, 339
keyCode property (keyboard events), 342, 343-344
keydown event, 303, 340-341
  listening for, 341-342
keypress event, 340-341
  listening for, 341-342
keyup event, 303, 340-341
  listening for, 341-342
keywords
  break
    in for loops, 53
    in switch statements, 41-42
  continue, 53
  function, 22
  Infinity, 153
  let, 72
  NaN, 154
  null, 196-197
  return
    exiting functions early, 28-29
    returning data, 27-28
    this, 175-177
typeof, 137
  var, 14, 70-71

L

lastChild property, 259-260
lastIndexOf method
  Array object, 146-147
  String object, 129-130
length property
  Array object, 142
  String object, 124
let keyword, 72
lexical scope, 205
listening for events
  addEventListener function, 302-304
  in document object, 354
interrupting, 319-321
keyboard events,
  341-342
on multiple elements,
  365-371
phases, 316-319
removeEventListener function, 309

literal notation, 167
literals, string, 123
LN2 property
  (Math object), 155
LN10 property
  (Math object), 155
load event, 303, 353-355
local scope, 70-71
locking in state, 207-212
LOG2E property
  (Math object), 155
LOG10E property
  (Math object), 155
logging messages, 296
loops
  do...while loops, 56-57
  for loops, 49-50, 52
    accessing array values, 142
    arrays, 54-55
    decrementing, 54
    iterations, 51-52
    missing sections, 55
    non-numerical
      iterations, 54
    skipping iterations, 53
    starting iterations, 51
    stopping early, 53
    terminating condition, 51
types of, 48
  while loops, 56
lowercase, changing strings to, 130

M

match method
  (String object), 130
Math object, 117, 134,
  154-155. See also numbers
  absolute value, 159
constants, 155-157
powers and square roots, 158-159
random numbers, 159
rounding numbers, 157
trigonometric functions, 158
merging arrays, 147
metaKey property (keyboard events), 343
methods. See also functions
add (classList API), 250-251
charAt (String object), 125
concat
    Array object, 147
    String object, 126
contains (classList API), 252
create (Object type), 171-174
createElement (DOM), 266
defining, 168-169
getAttribute, 242-244
indexOf
    Array object, 146-147
    String object, 129-130
lastIndexOf
    Array object, 146-147
    String object, 129-130
match (String object), 130
pop (Array object), 145
preventDefault (Event type), 308, 321, 333, 347
push (Array object), 143
remove (classList API), 251
setAttribute, 242-244
shift (Array object), 145
slice
    Array object, 146
    String object, 127
split (String object), 128-129
stopPropagation (Event type), 308, 320-321, 370
substr (String object), 127-128
toggle (classList API), 251
toLowerCase
    (String object), 130
toUpperCase
    (String object), 130
unshift (Array object), 143-144
minus sign (−), subtraction operator, 151
modifying DOM elements, 240-241
    attribute values, 242-244
text values, 242
modulus operator, 151
mouse, 325
    browser-specific mouse position, 334-335
    button detection, 335-336
global mouse position, 333-334
    scroll wheel movement, 336-337
mouseleave event, 329
mousemove event, 303, 331
mouseout event, 303, 328-329
mouseover event, 303, 328-329
mouseup event, 330-331
mousewheel event, 330-331
    336-337
multi-line comments, 104
multiple elements, events on, 365-371
multiple key presses, 345-347
multiplication operator, 20, 151
naming variables, 15-16
NaN keyword, 154
navigating DOM, 256-258
    accessing all children, 261
    checking child existence, 261
child relationships, 259-260
    parent and sibling relationships, 259
usage examples, 262
nextSibling property, 259
nodes, 225-226
non-numerical iterations in for loops, 54
null keyword, 196-197
Null type, 114
Number object, 117, 134
Number type, 114
numbers. See also Math object
  absolute value, 159
  converting strings to, 154
  Infinity keyword, 153
  NaN keyword, 154
  operators
    incrementing and decrementing, 152-153
    simple math, 151
  powers and square roots, 158-159
  random numbers, 159
  rounding, 157
  types, 150
  usage examples, 150

O

object literal syntax, 163
Object type, 114, 162
  create method, 171-174
  creating objects, 163-166
DOM element
  comparison, 238-240
  [[Prototype]] property, 166
objects, 115-116
  child objects, 170, 258
  accessing all, 261
  checking existence, 261
  event listening on, 367-370
  properties, 259-260
  constructor functions, 135
  creating, 163-166
  custom objects, creating, 169-174
  document, 228-229, 256-257, 354
DOM element comparison, 238-240
  extending
    array shuffling example, 179-180
    prototype property, 181-185
    reasons against, 185-186
  html, 256-257
  inspecting, 294-295
  list of, 117-118
  methods, defining, 168-169
  parent objects, 170, 258
  event listening on, 367-370
  properties, 259
  properties, defining, 167-169
  prototype, 165
  prototypical inheritance model, 178
  sibling objects, 258, 259
  style, 249-250
  temporarily converting primitives to, 133-138
  this keyword, 175-177
  window, 69, 227, 256-257
operators
  conditional, 34-36
  incrementing and decrementing, 152-153
  simple math, 151
  outer functions, shared variables, 81-86
  outer variables, referencing, 207-212
  preventDefault method (Event type), 308, 321, 333, 347

P

page loads
  DOMContentLoaded and load events, 353-355
  steps in, 350-353
parent objects, 170, 258
  event listening on, 367-370
  properties, 259
parentheses (( ), 203-204
in functions, 23
in IIFE, 203-204
in mathematical expressions, 151
parentNode property, 259
parsing HTML documents, 96
percent sign (%), modulus operator, 151
phases (event listening), 316-318
  usage examples, 322-324
  when to use, 319
PI property (Math object), 155, 156-157
pizza metaphor (types), 110-113
plus sign (+)
  addition operator, 151
  concatenation operator, 125-126
pop method (Array object), 145
pow() function (Math object), 158
powers (Math object), 158-159
preventDefault method (Event type), 308, 321, 333, 347

How can we make this index more useful? Email us at indexes@quepublishing.com
previousSibling property, 259
primitives, 115
  booleans as, 190
null, 196-197
object usage versus, 117-118
temporary conversion to objects, 133-138
undefined, 197-198
private code with IIFE, 213-216
properties, 162
  child objects, 259-260
className, 244
CSS, styling directly, 249-250
for custom objects, 170-171
defining, 167-169
Event type, 308
id, 244
keyboard events, 342-343
MouseEvent object
  browser-specific mouse position, 334-335
  button detection, 335-336
global mouse position, 333-334
  scroll wheel movement, 336-337
parent and sibling objects, 259
prototype, extending objects, 181-185
textContent, 242
this keyword, 175-177
__proto__ property, 164
prototype chains, 166, 170-171
prototype objects, 165
prototype property, extending objects, 181-185
[[Prototype]] property, 166
prototypical inheritance model, 178
push method (Array object), 143
querySelector function, 233, 234-235, 257
querySelectorAll function, 233-235, 257
quotation marks (") in strings, 11, 123
random() function
  (Math object), 159
random numbers, 159
referencing
  JavaScript files, 94-96
  outer variables, 207-212
RegExp object, 117, 134
relative paths, referencing JavaScript files, 94-96
remove method
  (classList API), 251
removeChild function, 271-273
removeEventListener function, 309
removing
  breakpoints, 293
class values, 251
  DOM elements, 271-273
  items from arrays, 145-146
repeats. See loops
requestAnimationFrame function, 62-64
requestID variable, 64
return keyword
  exiting functions early, 28-29
  returning data, 27-28
returning data from functions, 27-28
  within functions, 78-81
Revealing Module Pattern, 215
right-click menus, disabling, 332-333
round() function
  (Math object), 157
rounding numbers, 157
scope of variables
  block scope, 71-72
  closures, 77-78, 81-86
  global scope, 16-17, 68-69
  initializing variables in functions, 72-74
  inspecting objects, 295
  lexical scope, 205
  local scope, 70-71
  var keyword, 70-71
screenX property
  (MouseEvent object), 333-334
screenY property
  (MouseEvent object), 333-334
script tag, 9-10
  async attribute, 358-359
code placement within, 92-93
defer attribute, 358, 359-360
placement in HTML document, 96, 355-358, 360-361
referencing JavaScript files, 94-96
scroll event, 303
scroll wheel movement, 336-337
searching in DOM
CSS selector syntax, 234-235
querySelector function, 233
querySelectorAll function, 233-234
semicolon (;) in statements, 11
setAttribute method, 242-244
setInterval function, 61-62
setTimeout function, 60-61
shared variables in inner functions, 81-86
shift method (Array object), 145
shiftKey property (keyboard events), 342
showDistance function
arguments, 25-26
returning data, 27
shuffling arrays example (extending objects), 179-180
sibling objects, 258, 259
sin() function (Math object), 158
single line comments, 103-104
slice method
Array object, 146
String object, 127
split method (String object), 128-129
sqrt() function (Math object), 158
SQR1_2 property
(Math object), 155
SQR2 property
(Math object), 155
square brackets ([])
array declaration, 140
property definition, 167
square roots, 158-159
src attribute (script tag), 94-95
starting condition in for loops, 51
state, locking in, 207-212
statements
case, 39-42
conditional
if/else statements, 32-34
if/else-if/else statements, 38-39
if-only statements, 38
switch statements, 39-42
default, 42
definition, 10
loops
do...while loops, 56-57
for loops, 49-50, 52
types of, 48
while loops, 56
semicolon (;) in, 11
stepping through code, 290-292
stopPropagation method
(Event type), 308, 320-321, 370
strict equality operator
(==), 35
comparison with == equality operator, 192-194, 198
strict inequality operator
(!=), 35
comparison with != inequality operator, 192-194
string literals, 123
String object, 117, 134
charAt method, 125
concat method, 126
indexOf method, 129-130
lastIndexOf method, 129-130
length property, 124
match method, 130
slice method, 127
split method, 128-129
substr method, 127-128
toLowerCase method, 130
toUpperCase method, 130
String type, 114
strings, 121-123
accessing characters in, 124-125
changing case, 130
concatenating, 125-126
converting to numbers, 154
finding characters within, 129-130
primitive versus object forms, 134-136
quotation marks ("""" in, 11, 123
substrings, 126
slice method, 127
split method, 128-129
substr method, 127-128

How can we make this index more useful? Email us at indexes@quepublishing.com
style object, 249-250
styling HTML elements, 247-248
with classList API, 250-252
directly with style object, 249-250
with JavaScript, 248
substr method (String object), 127-128
substrings, 126
slice method, 127
split method, 128-129
substr method, 127-128
subtraction operator, 151
switch statements, 39-42
if/else statement
comparison, 42-44
when to use, 44-45

tags. See script tag
tan() function (Math object), 158
target property (Event type), 308
terminating condition in for loops, 51
text values, modifying in DOM elements, 242. See also strings
textContent property, 242
this keyword, 175-177
time ID variable, 60
timers
canceling, 60
requestAnimationFrame function, 62-64
setInterval function, 61-62
setTimeout function, 60-61
toggle method (classList API), 251
toggling class values, 251
toLowerCase method (String object), 130
toUpperCase method (String object), 130
trigonometric functions, 158
true/false. See booleans
type coercion, 193
type property (Event type), 308
typeof keyword, 137
types
list of, 113-115
for numbers, 150
objects, 115-116
list of, 117-118
pizza metaphor, 110-113
primitives, 115
undefined primitive, 197-198
Undefined type, 114
unshift method (Array object), 143-144
uppercase, changing strings to, 130
values
of arrays, accessing, 141-142
definition, 14
null, 196-197
undefined, 197-198
var keyword, 14, 70-71
variables
arguments as, 26
declaring, 14-15, 16-17
definition, 14
hoisting, 72-74, 205
initializing, 14-15, 16-17
as booleans, 189
in functions, 72-74
naming, 15-16
outer variables,
referencing, 207-212
request ID, 64
scope
block scope, 71-72
closures, 77-78, 81-86
global scope, 16-17,
68-69
inspecting objects, 295
lexical scope, 205
local scope, 70-71
var keyword, 70-71
shared variables in inner
functions, 81-86
time ID, 60
View Source command
(Developer Tools), 284-287
viewing DOM in Developer Tools, 282-287
web documents. See HTML documents
wheelDelta property
(mousewheel events), 336-337
which property (MouseEvent object), 336
while loops, 56
whitespace in comments, 106
window object, 69, 227,
256-257