

# JavaScript™

## ABSOLUTE BEGINNER'S GUIDE

No experience necessary!



Third Edition



Kirupa Chinnathambi

FREE SAMPLE CHAPTER |



# JavaScript™

Third Edition

**ABSOLUTE  
BEGINNER'S  
GUIDE**



Kirupa Chinnathambi

# JavaScript™ Absolute Beginner's Guide, Third Edition

Copyright © 2023 by Pearson Education, Inc.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit [www.pearson.com/permissions](http://www.pearson.com/permissions).

No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-13-795916-7

ISBN-10: 0-13-795916-8

Library of Congress Control Number: 2022914516

ScoutAutomatedPrintCode

## Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Pearson cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

## Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

## Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [intlcs@pearson.com](mailto:intlcs@pearson.com).

### Editor-in-Chief

Mark Taub

### Director, ITP Product Management

Brett Bartow

### Acquisitions Editor

Kim Spenceley

### Development Editor

Chris Zahn

### Managing Editor

Sandra Schroeder

### Project Editor

Mandie Frank

### Copy Editor

Bart Reed

### Indexer

Ken Johnson

### Proofreader

Barbara Mack

### Technical Editor

Trevor McCauley

### Editorial Assistant

Cindy Teeters

### Designer

Chuti Prasertsith

### Composer

codeMantra

### Graphics

Vived Graphics

# Pearson's Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity of all learners. We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, socioeconomic status, ability, age, sexual orientation, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world's leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

Our ambition is to purposefully contribute to a world where

- Everyone has an equitable and lifelong opportunity to succeed through learning
- Our educational products and services are inclusive and represent the rich diversity of learners
- Our educational content accurately reflects the histories and experiences of the learners we serve
- Our educational content prompts deeper discussions with learners and motivates them to expand their own learning (and worldview)

While we work hard to present unbiased content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

- Please contact us with concerns about any potential bias at <https://www.pearson.com/report-bias.html>.

## Credits

Figures 1.2a-c, 11.2-11.8, Chapter 35 – Screenshots of Chrome browser: Google LLC

Figures 1.2d, Chapter 43 - Screenshot of smileys: Twitter, Inc.

Figure 1.2e: GitHub, Inc.

Figures 1.2f, 34.1: Netflix, Inc.

Figures 1.5-1.8, Chapter 36 - Screenshot of an Excel sheet: Microsoft

Figures 5.1, 9.7, Chapter 41 - Screenshot of JavaScript file: Dropbox, Inc.

Figure 11.1: Randall Munroe

Chapter 43 – Screenshots of using an emoji and Character Viewer on Mac: Apple Inc

Many illustrations and screenshots use emojis from Twitter’s Twemoji set: <https://twemoji.twitter.com/>

Cover Image: rozdesign/Shutterstock

# Contents at a Glance

	Introduction .....	1
1	Hello, World!.....	5
<b>Part I</b>	<b>The Basic Stuff</b>	
2	Values and Variables .....	15
3	Functions.....	23
4	Conditional Statements: if, else, and switch.....	39
5	Looping with for, while, and do...while!.....	57
6	Commenting Your Code...FTW!.....	71
7	Timers .....	79
8	Variable Scope .....	85
9	Closures .....	95
10	Where Should Your Code Live?.....	109
11	Console Logging Basics .....	123
<b>Part II</b>	<b>It's an Object-Oriented World</b>	
12	Of Pizza, Types, Primitives, and Objects.....	135
13	Arrays .....	145
14	Strings .....	161
15	Combining Strings and Variables .....	173
16	When Primitives Behave Like Objects .....	179
17	Numbers .....	185
18	Getters and Setters.....	201
19	A Deeper Look at Objects.....	211
20	Using Classes .....	231
21	Extending Built-in Objects.....	247
22	Arrow Functions .....	259
23	Making Sense of this and More.....	265
24	Booleans and the Stricter === and !== Operators.....	277
25	Null and Undefined.....	283
26	All About JSON (JavaScript Object Notation).....	287
<b>Part III</b>	<b>Working with the DOM</b>	
27	JS, the Browser, and the DOM .....	303
28	Finding Elements in the DOM.....	315
29	Modifying DOM Elements .....	321
30	Styling Our Content.....	337
31	Using CSS Custom Properties.....	345

32	Traversing the DOM.....	353
33	Creating and Removing DOM Elements.....	363
34	Quickly Adding Many Elements into the DOM.....	381
35	In-Browser Developer Tools.....	397
<b>Part IV Dealing with Events</b>		
36	Events.....	417
37	Event Bubbling and Capturing.....	429
38	Mouse Events.....	443
39	Keyboard Events.....	457
40	Page Load Events and Other Stuff.....	467
41	Loading Script Files Dynamically.....	481
42	Handling Events for Multiple Elements.....	491
<b>Part V Totally Useful Topics that Only Make Sense Now</b>		
43	Using Emojis in HTML, CSS, and JavaScript.....	501
44	Making HTTP/Web Requests in JavaScript.....	511
45	Accessing the Webcam.....	529
46	Array and Object Destructuring.....	539
47	Storing Data Using Web Storage.....	549
48	Variable and Function Hoisting.....	559
49	Working with Sets.....	565
50	Conclusion.....	577
<b>Glossary.....</b>		<b>581</b>
<b>Index.....</b>		<b>585</b>

## Reader Services

Register your copy of *JavaScript™ Absolute Beginner's Guide, Third Edition* at [informit.com](http://informit.com) for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to [informit.com/register](http://informit.com/register) and log in or create an account\*. Enter the product ISBN, **9780137959167**, and click Submit. Once the process is complete, you will find any available bonus content under Registered Products.

\*Be sure to check the box that you would like to hear from us in order to receive exclusive discounts on future editions of this product.

# Table of Contents

<b>Introduction</b> .....	1
Parlez-Vous JavaScript?.....	2
Contacting Me/Getting Help .....	2
<b>1 Hello, World!</b> .....	<b>5</b>
What Is JavaScript? .....	7
What JavaScript Looks Like .....	8
Hello, World!.....	9
The HTML Document .....	9
Statements, Expressions, and Functions.....	12
<b>I The Basic Stuff</b>	
<hr/>	
<b>2 Values and Variables</b> .....	<b>15</b>
Using Variables .....	16
More Variable Stuff.....	18
Naming Variables .....	18
More on Declaring and Initializing Variables.....	19
<b>3 Functions</b> .....	<b>23</b>
What Is a Function?.....	26
A Simple Function .....	26
Creating a Function That Takes Arguments .....	30
Creating a Function That Returns Data.....	35
The Return Keyword .....	35
Exiting the Function Early.....	36
Function Expressions .....	36
<b>4 Conditional Statements: if, else, and switch</b> .....	<b>39</b>
The If/Else Statement.....	40
Meet the Conditional Operators .....	43
Creating More Complex Expressions.....	46
Variations on the If/Else Statement.....	47



Switch Statements .....	49
Using a Switch Statement .....	49
Similarity to an If/Else Statement .....	53
Deciding Which to Use .....	55
<b>5 Looping with for, while, and do...while!.....</b>	<b>57</b>
The for Loop .....	59
The Starting Point.....	62
The Step .....	62
The Condition (aka How Long to Keep Looping) .....	63
Putting It All Together.....	64
Some for Loop Examples .....	64
Breaking a Loop .....	65
Skipping an Iteration.....	65
Going Backwards.....	66
You Don't Have to Use Numbers .....	66
Oh No He Didn't!.....	66
The Other Loops .....	67
The while Loop .....	67
The do...while Loop.....	68
<b>6 Commenting Your Code...FTW!.....</b>	<b>71</b>
What Are Comments?.....	72
Single-Line Comments.....	73
Multiline Comments.....	74
Commenting Best Practices .....	76
<b>7 Timers.....</b>	<b>79</b>
Delaying with setTimeout.....	80
Looping with setInterval .....	81
Animating Smoothly with requestAnimationFrame.....	83
<b>8 Variable Scope.....</b>	<b>85</b>
Global Scope .....	86
Local Scope.....	88
Miscellaneous Scoping Shenanigans .....	89
Block Scoping .....	89
How JavaScript Processes Variables.....	93
Closures.....	94

<b>9</b>	<b>Closures</b> .....	<b>95</b>
	Functions Within Functions .....	96
	When the Inner Functions Aren't Self-Contained .....	100
<b>10</b>	<b>Where Should Your Code Live?</b> .....	<b>109</b>
	Approach #1: All the Code Lives in Your HTML Document .....	113
	Approach #2: The Code Lives in a Separate File .....	114
	The JavaScript File .....	114
	Referencing the JavaScript File .....	115
	So, Which Approach to Use? .....	118
	Yes, My Code Will Be Used on Multiple Documents! .....	118
	No, My Code Is Used Only Once on a Single HTML Document! .....	120
<b>11</b>	<b>Console Logging Basics</b> .....	<b>123</b>
	Meet the Console .....	124
	Displaying the Console .....	126
	If You Want to Follow Along .....	127
	Console Logging 101 .....	128
	Meet the log Method .....	128
	Going Beyond Predefined Text .....	130
	Displaying Warnings and Errors .....	131
 <b>II It's an Object-Oriented World</b>		
<b>12</b>	<b>Of Pizza, Types, Primitives, and Objects</b> .....	<b>135</b>
	Let's First Talk About Pizza .....	136
	From Pizza to JavaScript! .....	139
	What Are Objects? .....	141
	The Predefined Objects Roaming Around in JavaScript .....	142
<b>13</b>	<b>Arrays</b> .....	<b>145</b>
	Creating an Array .....	146
	Accessing Array Values .....	147
	Adding Items .....	149
	Removing Items .....	151

Finding Items .....	152
Merging Arrays .....	152
Mapping, Filtering, and Reducing Arrays .....	153
The Old School Way .....	153
Modifying Each Array Item with map .....	154
Filtering Items .....	156
Getting One Value from an Array of Items .....	157
More on the Callback Function Arguments .....	159
A Short Foray into Functional Programming .....	160
<b>14 Strings .....</b>	<b>161</b>
The Basics .....	162
String Properties and Methods .....	163
Accessing Individual Characters .....	163
Combining (aka Concatenating) Strings .....	165
Getting Substrings Out of Strings .....	166
Splitting a String with split .....	168
Finding Something Inside a String .....	169
Uppercasing and Lowercasing Strings .....	171
<b>15 Combining Strings and Variables .....</b>	<b>173</b>
Our Setup .....	174
Using the + Operator (aka String Concatenation) .....	175
Template Literals (aka String Interpolation) .....	175
<b>16 When Primitives Behave Like Objects .....</b>	<b>179</b>
Strings Aren't the Only Problem .....	180
Let's Pick on Strings Anyway .....	180
Why This Matters .....	182
<b>17 Numbers .....</b>	<b>185</b>
Using a Number .....	186
Operators .....	187
Doing Simple Math .....	187
Incrementing and Decrementing .....	188
Hexadecimal and Octal Values .....	190

Special Values—Infinity and NaN .....	190
Infinity .....	190
NaN.....	191
The Math Object .....	191
The Constants.....	192
Rounding Numbers.....	193
Trigonometric Functions .....	194
Powers and Square Roots.....	195
Getting the Absolute Value.....	196
Random Numbers .....	196
<b>18 Getters and Setters.....</b>	<b>201</b>
A Tale of Two Properties.....	202
Meet Getters and Setters .....	205
Shout Generator.....	206
Logging Activity.....	206
Property Value Validation .....	207
<b>19 A Deeper Look at Objects.....</b>	<b>211</b>
Meet the Object.....	212
Creating Objects .....	213
Adding Properties .....	213
Removing Properties.....	217
What Is Going on Behind the Scenes? .....	218
Creating Custom Objects.....	222
The this Keyword.....	226
<b>20 Using Classes.....</b>	<b>231</b>
The Class Syntax and Object Creation .....	232
Creating an Object.....	232
Meet the Constructor.....	234
What Goes Inside the Class .....	236
Extending Objects.....	240
<b>21 Extending Built-in Objects.....</b>	<b>247</b>
Say Hello to prototype Again, Sort Of!.....	249
Using a Subclassing Approach.....	253
Extending Built-in Objects Is Controversial.....	255
You Don't Control the Built-in Object's Future .....	256
Some Functionality Should Not Be Extended or Overridden .....	256

<b>22 Arrow Functions .....</b>	<b>259</b>
What Are Arrow Functions?.....	260
Starting with the Basics.....	260
Of Arguments and Parenthesis.....	261
To Curly Bracket or Not to Curly Bracket .....	261
Putting It All Together.....	263
<b>23 Making Sense of this and More.....</b>	<b>265</b>
The this Keyword 101 .....	266
When this Just Ain't Right.....	268
Using a Redefined Version of the this Keyword.....	271
Arrow Functions and Their Lexical Scope .....	273
One Method to Bind Them All.....	274
<b>24 Booleans and the Stricter === and !== Operators .....</b>	<b>277</b>
The Boolean Object.....	278
The Boolean Function.....	278
Strict Equality and Inequality Operators.....	281
<b>25 Null and Undefined .....</b>	<b>283</b>
Null.....	284
Undefined.....	284
<b>26 All About JSON (JavaScript Object Notation).....</b>	<b>287</b>
What Is JSON?.....	288
Looking Inside a JSON Object .....	292
Property Names.....	292
The Values.....	293
Reading JSON Data .....	297
Parsing JSON-Looking Data into Actual JSON .....	299
Writing JSON Data?.....	300
 <b>III Working with the DOM</b>	
<b>27 JS, the Browser, and the DOM.....</b>	<b>303</b>
What HTML, CSS, and JavaScript Do .....	304
HTML Defines the Structure .....	304

Prettify My World, CSS! .....	306
It's JavaScript Time! .....	307
Meet the Document Object Model .....	309
The window Object .....	311
The Document Object .....	312
<b>28 Finding Elements in the DOM .....</b>	<b>315</b>
Meet the querySelector Family .....	316
querySelector .....	317
querySelectorAll .....	317
It Really Is the CSS Selector Syntax .....	318
<b>29 Modifying DOM Elements .....</b>	<b>321</b>
DOM Elements Are Objects, Sort Of! .....	322
Let's Actually Modify DOM Elements .....	324
Changing an Element's Text Value .....	326
Attribute Values .....	328
Basics of Attribute Access .....	328
Custom Attributes .....	330
<b>30 Styling Our Content .....</b>	<b>337</b>
Why Would We Set Styles Using JavaScript? .....	338
A Tale of Two Styling Approaches .....	338
Setting the Style Directly .....	339
Adding and Removing Classes Using JavaScript .....	340
Going Further .....	343
<b>31 Using CSS Custom Properties .....</b>	<b>345</b>
What Are CSS Custom Properties/Variables? .....	346
Setting Complex Values Easily .....	348
<b>32 Traversing the DOM .....</b>	<b>353</b>
Finding Your Way Around .....	354
Dealing with Siblings and Parents .....	356
Let's Have Some Kids! .....	357
Putting It All Together .....	358
Checking If a Child Exists .....	359
Accessing All the Child Elements .....	359
Walking the DOM .....	360

<b>33</b>	<b>Creating and Removing DOM Elements.....</b>	<b>363</b>
	Creating Elements .....	364
	Removing Elements .....	372
	Cloning Elements .....	374
<b>34</b>	<b>Quickly Adding Many Elements into the DOM .....</b>	<b>381</b>
	General Approach .....	383
	Example.....	383
	Getting Started .....	384
	The innerHTML Approach .....	388
	The DocumentFragment Approach .....	391
	Removing Elements (Emptying an Entire Subtree).....	395
<b>35</b>	<b>In-Browser Developer Tools.....</b>	<b>397</b>
	Meet the Developer Tools.....	398
	Inspecting the DOM .....	400
	Debugging JavaScript .....	405
	Meet the Console.....	411
	Inspecting Objects .....	412
	Logging Messages .....	414

## IV Dealing with Events

---

<b>36</b>	<b>Events.....</b>	<b>417</b>
	What Are Events? .....	418
	Events and JavaScript .....	420
	Listening for Events .....	420
	Reacting to Events .....	422
	A Simple Example .....	423
	The Event Arguments and the Event Type.....	426
<b>37</b>	<b>Event Bubbling and Capturing .....</b>	<b>429</b>
	Event Goes Down, Event Goes Up .....	430
	Meet the Phases .....	434

Who Cares? .....	437
Event, Interrupted .....	438
<b>38 Mouse Events .....</b>	<b>443</b>
Meet the Mouse Events .....	444
Clicking Once and Clicking Twice .....	444
Mousing Over and Mousing Out .....	446
The Very Click-Like Mousing Down and Mousing Up Events .....	448
The Event Heard Again...and Again...and Again! .....	449
The Context Menu .....	450
The MouseEvent Properties .....	451
The Global Mouse Position .....	451
The Mouse Position Inside the Browser .....	452
Detecting Which Button Was Clicked .....	453
Dealing with the Mouse Wheel .....	454
<b>39 Keyboard Events .....</b>	<b>457</b>
Meet the Keyboard Events .....	458
Using These Events .....	459
The Keyboard Event Properties .....	460
Some Examples .....	461
Checking That a Particular Key Was Pressed .....	461
Doing Something When the Arrow Keys Are Pressed .....	462
Detecting Multiple Key Presses .....	462
<b>40 Page Load Events and Other Stuff .....</b>	<b>467</b>
The Things That Happen During Page Load .....	468
Stage Numero Uno .....	469
Stage Numero Dos .....	469
Stage Numero Three .....	470
The DOMContentLoaded and load Events .....	471
Scripts and Their Location in the DOM .....	473
Script Elements: async and defer .....	477
async .....	477
defer .....	477



<b>41 Loading Script Files Dynamically .....</b>	<b>481</b>
The Basic Technique .....	482
Running Our Dynamically Loaded Script First .....	486
Running Dependent Code After Our Script File Has Loaded.....	488
<b>42 Handling Events for Multiple Elements .....</b>	<b>491</b>
How to Do All This .....	493
A Terrible Solution.....	494
A Good Solution.....	495
Putting It All Together.....	498
 <b>V Totally Useful Topics that Only Make Sense Now</b>	
<hr/>	
<b>43 Using Emojis in HTML, CSS, and JavaScript .....</b>	<b>501</b>
What Are Emojis Exactly? .....	502
Emojis in HTML.....	503
Using the Emoji Directly .....	504
Specifying the Emoji Codepoint.....	505
<b>44 Making HTTP/Web Requests in JavaScript.....</b>	<b>511</b>
The Example .....	513
Meet Fetch .....	514
Diving into the Code .....	514
Wrapping Up the Example.....	518
Meet XMLHttpRequest .....	520
Creating the Request .....	521
Sending the Request.....	522
Asynchronous Stuff and Events.....	523
Processing the Request .....	523
Processing the Request...for Realz!.....	526
<b>45 Accessing the Webcam.....</b>	<b>529</b>
The Example .....	530
Overview of How This Works .....	531
Adding the Code.....	532
Examining the Code .....	535

<b>46</b>	<b>Array and Object Destructuring .....</b>	<b>539</b>
	Destructuring Examples .....	541
	General Overview Using Arrays .....	541
	Destructuring with Objects .....	544
<b>47</b>	<b>Storing Data Using Web Storage .....</b>	<b>549</b>
	How Web Storage Works .....	550
	What Exactly Goes on Inside .....	550
	Web Storage Data Is Tied to Your Domain .....	552
	Getting Your Code On .....	552
	Adding Data .....	552
	Retrieving Data .....	554
	Removing Data .....	555
	Dealing with File Size .....	556
	Detecting Support for Web Storage .....	556
	What About Session Storage? .....	557
<b>48</b>	<b>Variable and Function Hoisting .....</b>	<b>559</b>
	JavaScript and Compiler Behavior .....	560
	Variable Declarations .....	561
	Function Declarations .....	562
	Some Hoisting Quirks .....	562
<b>49</b>	<b>Working with Sets .....</b>	<b>565</b>
	Creating a Set, Part I .....	566
	Adding Items to a Set .....	567
	How Checking for Duplicates Works .....	567
	Creating a Set, Part 2 .....	569
	Checking the Size of Our Set .....	570
	Deleting Items from a Set .....	571
	Checking If an Item Exists .....	572
	Looping Through Items in a Set .....	572
	Entries, Keys, and Values .....	573
<b>50</b>	<b>Conclusion .....</b>	<b>577</b>
	<b>Glossary .....</b>	<b>581</b>
	<b>Index .....</b>	<b>585</b>

## 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 developers happy and productive as a Product Manager at Google. In his non-waking hours, he is probably sleeping, joining Meena in running after their daughter Akira, protecting himself from Pixel (aka a T-rex in an unassuming cat's body)...or writing about himself in the third person.

You can find him on Twitter, Facebook, LinkedIn, and the interwebs at large. Just search for his name in your favorite search engine.

## About the Technical Editor

**Trevor McCauley:** friend.

## 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 you are about to 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 so many years ago, Kim Spenceley for carrying forward Mark's work in the second and third editions, Chris Zahn for meticulously ensuring everything is human-readable, Bart Reed for his excellent copyediting, Mandie Frank for keeping the project on track, 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 (1<sup>st</sup> edition), Trevor McCauley (1<sup>st</sup>, 2<sup>nd</sup>, and 3<sup>rd</sup> editions), Steve Mills (3<sup>rd</sup> edition), and Dillion Megida (3<sup>rd</sup> edition). I can't thank them enough for their thorough (and frequently, 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 their support. 😊

## We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

*Please note that we cannot help you with technical problems related to the topic of this book.*

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: [community@informit.com](mailto:community@informit.com)

*This page intentionally left blank*

## IN THIS CHAPTER

- Learn how to use values to store data
- Organize your code with variables
- Get a brief look at variable naming conventions



# 2

## VALUES AND VARIABLES

In JavaScript, every piece of data we provide or use is considered to contain a value. In our example from the previous chapter, we might think of **hello, world!** as just some words we pass in to the `alert` function:

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

To JavaScript, however, these words have a specific representation under the covers. They are considered **values**. We may not have thought much about that when we were typing those words, but when we are in JavaScript Country, every piece of data we touch is considered a value.

Now, why is knowing this important? It is important because we will be working with values a whole lot. Working with them in a way that doesn't drive you insane is a good thing. There are just two things we need to simplify our life working with values:

- We need to identify them easily.
- We need to reuse them throughout our application without unnecessarily duplicating them.

Those two things are provided by what we are going to be spending the rest of our time on: **variables**. Let's learn all about them here.

## Using Variables

A variable is an identifier for a value. Instead of typing **hello, world!**, every time we want to use that phrase in our application, we can assign that phrase to a variable and use that variable whenever we need to use **hello, world!** again. This will make more sense in a few moments—I promise!

There are several ways to use variables. For most cases, the best way is by relying on the `let` keyword followed by the name you want to give your variable, like so:

```
let myText
```

In this line of code, we declare a variable called `myText`. Right now, our variable has simply been **declared**. It doesn't contain anything of value. It is merely an empty shell.

Let's fix that by **initializing** our variable to a value like, say, **hello, world!**, as shown here:

```
let myText = "hello, world!";
```

At this point, when this code runs, our `myText` variable will have the value **hello, world!** associated with it. Let's put all of this together as part of a full example. If you still have **hello\_world.htm** open from earlier, replace the contents of your

`<script>` tag with the following, or you can create a new HTML file and add the following contents into it:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>An Interesting Title Goes Here</title>

  <style>

</style>
</head>

<body>
  <script>
    let myText = "hello, world!";
    alert(myText);
  </script>
</body>

</html>
```

Notice that we are no longer passing in the **hello, world!** text to the `alert` function directly. Instead, we are now passing in the variable name `myText` instead. The end result is the same. When this script runs, an alert with **hello, world!** will be shown. What this change allows us to do is have one place in our code where **hello, world!** is being specified. If we wanted to change **hello, world!** to **The dog ate my homework!**, all we would have to do is just make one change to the phrase specified by the `myText` variable:

```
let myText = "The dog ate my homework!";
alert(myText);
```



Throughout our code, wherever we reference the `myText` variable, we will now see the new text appear. Although this is hard to imagine as being useful for something as simple as what we have right now, for larger applications, the convenience of having just one location where we can make a change that gets reflected everywhere is a major time-saver. You'll see more less-trivial cases of the value variables provide in subsequent examples.

## More Variable Stuff

What we learned in the previous section will take us far in life. At least, it will in the parts of our life that involve getting familiar with JavaScript. We won't dive too much further into variables here—we'll do all of that as part of future chapters where the code is more complex and the importance of variables is more obvious. With that said, there are a few odds and ends we should cover before calling it a day.

### Naming Variables

We have a lot of freedom in naming our variables however we see fit. Ignoring what names we should give things based on philosophical/cultural/stylistic preferences, from a technical point of view, JavaScript is very lenient on what characters can go into a variable name.

This leniency isn't infinite, so we should keep the following points in mind when naming our variables:

- Variables can be as short as one character, or they can be as long as you want—think thousands and thousands of characters.
- Variables can start with a letter, underscore, or dollar sign (\$). They can't start with a number.
- Outside of the first character, our variables can be made up of any combination of letters, underscores, numbers, and \$ characters. We can also mix and match lowercase and uppercase letters to our heart's content.
- Spaces are not allowed.

Here are some examples of valid variable names:

```
let myText;  
let $;  
let r8;  
let _counter;  
let $field;
```

```
let thisIsALongVariableName_butItCouldBeLonger;
let __$abc;
let OldSchoolNamingScheme;
```

To see if a variable name is valid, check out the really awesome and simple **JavaScript Variable Name Validator** at <https://bit.ly/namevalidator>.

Outside of valid names, there are other things to focus on as well, such as naming conventions, how many people commonly name variables, and other things you identify with a name. We will touch on these items in other chapters.

## More on Declaring and Initializing Variables

One of the things you will learn about JavaScript is that it is a very forgiving and easy-to-work-with language.

### Declaring a Variable Is Optional

For example, we don't have to use the `let` keyword to declare a variable. We could just do something like the following:

```
myText = "hello, world!";
alert(myText);
```

Notice the `myText` variable is being used without formally being declared with the `let` keyword. While not recommended, this is completely fine. The end result is that we have a variable called `myText`. The only thing is that by declaring a variable this way, we are declaring it globally. Don't worry if the last sentence makes no sense. We'll look at what *globally* means when talking about variable scope later.

### Declaring and Initializing on Separate Lines Is Cool

There is one more thing to call out, and that is this: The declaration and initialization of a variable do not have to be part of the same statement. We can break them up across multiple statements:

```
let myText;
myText = "hello, world!";
alert(myText);
```

In practice, we will find ourselves breaking up our declaration and initialization of variables all the time.

## Changing Variable Values and the const Keyword

Lastly, we can change the value of a variable declared via `let` to whatever we want, whenever we want:

```
let myText;  
myText = "hello, world!";  
myText = 99;  
myText = 4 * 10;  
myText = true;  
myText = undefined;  
alert(myText);
```

If you have experience working with languages that are more strict and don't allow variables to store a variety of data types, this leniency is one of the features people both love and hate about JavaScript. With that said, JavaScript does provide a way for you to restrict the value of a variable from being changed after you initialize it. That restriction comes in the form of the `const` keyword, which we can declare and initialize our variables with:

```
const siteURL = "https://www.google.com";  
alert(siteURL);
```

By relying on `const`, we can't change the value of `siteURL` to something other than <https://www.google.com>. JavaScript will complain if we try to do that. There are some gotchas with using the `const` keyword, but it does a great job overall in preventing accidental modifications of a variable. We'll cover those pesky gotchas in bits and pieces when the time is right.



### TIP Jump Ahead—Variable Scoping

Now that you know how to declare and initialize variables, a very important topic is that of **visibility**. You need to know when and where a variable you declared can actually be used in your code. The catch-all phrase for this is **variable scope**. If you are curious to know more about it, you can jump ahead and read Chapter 8, "Variable Scope."

## THE ABSOLUTE MINIMUM

Values store data, and variables act as an easy way to refer to that data. There are a lot of interesting details about values, but those are details you do not need to learn right now. Just know that JavaScript enables you to represent a variety of values such as text and numbers without a lot of fuss.

To make your values more memorable and reusable, you declare variables. You declare variables using the `let` keyword and a **variable name**. If you want to initialize the variable to a default value, you follow all of that up with an equal sign (=) and the value you want to initialize your variable with.

? Ask a question: <https://forum.kirupa.com>

✓ Practice by building real apps: [https://bit.ly/coding\\_exercises](https://bit.ly/coding_exercises)

🐞 Errors/known issues: [https://bit.ly/javascript\\_errata](https://bit.ly/javascript_errata)



*This page intentionally left blank*

# Index

## Symbols

---

&& operators, 44  
\* (multiplication) operators, 187–188  
/ (division) operators, 187–188  
' (backtick) character, string interpolation (template literals), 175–177  
, (commas), destructuring arrays, 542–543  
{ } (curly brackets), arguments, 261–262  
== (equality) operators, 44, 281–282, 286  
=== operators  
  null primitives, 284  
  undefined primitives, 285–286  
!= (inequality) operators, 44, 281–282, 286  
- (minus sign) operator, 186, 187–188  
( ) (parentheses), arguments, 261  
% (percentage) operators, 187–188  
|| operators, 44  
+ operators, 162, 165–166, 175, 187–188  
' (single quotation mark), strings, 162, 163  
" (double quotation marks), strings, 162, 163  
> operators, 44  
>= operators, 44  
< operators, 44  
<= operators, 44

## A

---

absolute values, 196  
accessing  
  array values, 147–148  
  child elements, DOM, 359  
  HTML element attributes, 328–330  
  individual characters in strings, 163–165  
  webcams, 529–530  
    adding code, 532–535  
    constraints, 532–536  
    examining code, 535–537  
    example of, 530–531  
    overview, 531–532  
    stopping streams, 537  
accessor properties, 202–204  
addEventListener function, 420, 422  
  capturing events, 422  
  event handler, 421  
  event names, 421  
  sources, 420  
adding  
  classes, 340–342  
  data to Web Storage, 552–555  
  elements, DOM  
    DocumentFragment objects, 391–395  
    general approach, 383–388  
    innerHTML, 388–390  
  items to  
    arrays, 149–150  
    sets, 567  
  properties, to objects, 213–217  
  values in classes, 342  
Ajax (Asynchronous JavaScript and XML), HTTP requests, 512–513  
alert function, 124  
altKey property, keyboard events, 460  
animation, requestAnimationFrame function, 83–84  
arguments  
  arrow functions, 261, 263  
  curly brackets ( { } ), 261–262  
  events, 426–427  
  functions, creating in, 30–34  
  mismatched number of, 34  
  parentheses ( ( ) ), 261  
arrays, 153–154  
  adding items, 149–150  
  callback functions, 156, 159  
  creating, 146–147  
  destructuring, 541–542, 546  
    commas ( , ), 542–543  
    declarations, 543–544  
    destructuring arrays, 543–544  
    variables, 543  
  filtering items, 156–157  
  finding items, 152  
  JSON objects, 296–297  
  mapping items, 154–156  
  merging, 152–153  
  objects, 143, 180  
  reducing item values, 157–159  
  removing items, 151–152  
  values, accessing, 147–148  
arrow functions, 259, 263, 274  
  defined, 260  
  lexical scope, 273

arrow keys, keyboard events, 462

assignments, destructuring arrays, 543–544

async script element, 477

author's website, 579

---

## B

backtick (') character, string interpolation (template literals), 175–177

backwards in loops, going, 66

best practices, comments, 76–77

bind method, 274–275

block scoping, 89–92

Boolean functions, 278–280

Boolean logic, 47

Boolean objects, 143, 180, 277–278

Boolean values, JSON objects, 295

bracket notation, object properties, 214–215

breaking loops, 65

browsers

- developer tools, 397
- debugging JavaScript, 405–412
- displaying, 398–400
- DOM inspection, 400–405
- logging messages, 414–415
- object inspection, 412–413
- variable scope, 413–414
- View Source command, 402–405
- mouse positioning, 452–453
- webcam access, 529–530
  - adding code, 532–535
  - constraints, 532–536
  - examining code, 535–537
  - example of, 530–531

- overview, 531–532
- stopping streams, 537

bubbling events, 435–436, 437–438

built-in objects, extending, 247–248

- controversy, 255–257
- functionality, 256
- future of, 256
- online resources, 257
- prototype inheritance, 249–253
- subclasses, 247–248

buttons

- id values, 130–131
- mouse events, 453–454

---

## C

callback functions, 156, 159

calling functions, 26, 29–30

cameras (web), accessing, 529–530

- adding code, 532–535
- constraints, 532–536
- examining code, 535–537
- example of, 530–531
- overview, 531–532
- stopping streams, 537

capturing events, 422, 434–435, 437–438

cascading rules, CSS, 347

changing

- text values in DOM elements, 326–328
- variable values, 20

CharCode property, keyboard events, 460–462

checking

- for duplicates in sets, 567–569
- existence of
  - children, DOM, 357–358
  - class values, 343
- items in sets, 572
- size of sets, 570–571

- children, DOM, 355–356, 357–358
  - accessing elements of, 359
  - checking existence of, 357–358
  - null properties, 357–358

choosing event phases, 436–437

Chrome developer tools (Google), 398

Console

- debugging JavaScript, 405–412
- logging messages, 414–415
- variable scope, 413–414

debugging JavaScript, 405–412

displaying, 398–400

DOM inspections, 400–405

object inspections, 412–413

View Source command, 402–405

classes, 231–232

- adding/removing, 340–342
- components of, 236–239
- constructors, 234–236
- functions, appearance of, 239
- inside of, 236–239
- objects, creating, 232–234
- objects, extending, 240–244, 247–248
  - controversy, 255–257
  - functionality, 256
  - future of, 256
  - online resources, 257
  - prototype inheritance, 249–253
  - subclasses, 247–248
- subclasses, extending objects, 247–248
- syntax, 236–239
- values
  - adding/removing, 340–342
  - checking existence of class values, 343
  - online resources, 343
  - toggling, 342–343

click events, 421, 444–445, 446

cloning elements, DOM, 374–378

closures, 95–100

codepoints, emojis, 505–508

coding
 

- duplicate code, 118–120
- emojis, 174
- location/placement of code, 109–112
  - HTML documents, 113–114
  - in separate files, 114–116
- in multiple documents, 118–120
- in a single document, 120

combining (concatenating)
 

- strings, 162, 165–166, 173–174
  - interpolation (template literals), 175–177
  - plus sign (+) operators, 175
- variables, 173–174
  - interpolation (template literals), 175–177
  - plus sign (+) operators, 175

commas (,), destructuring
 

- arrays, 542–543

comments
 

- best practices, 76–77
- defined, 72–73
- JSDoc-style comments, 75
- multiline comments, 74–75
- single-line comments, 73–74
- whitespace, 76–77

compilers, behavior of, 560–561

complex expressions, if/else statements, 46–47

concatenating (combining)
 

- strings, 162, 165–166, 173–174
  - interpolation (template literals), 175–177
  - plus sign (+) operators, 175

variables, 173–174
 

- interpolation (template literals), 175–177
- plus sign (+) operators, 175

conditional operators, 43–46

conditional statements, 39–40
 

- if statements, 40–43, 46–47
- if/else statements, 40–43
  - complex expressions, 46–47
  - switch statement
    - similarities, 53–55
  - using, 55–56
- if/else-if/else statements, 47–48
- if-only statements, 47
- switch statements, 49–53
  - if/else statement
    - similarities, 53–55
    - using, 55–56
  - true/false evaluations, 42–43, 46–48, 53–54, 56

conditions, for loops, 63

configuring buttons, mouse events, 453–454

Console, 124–125, 127–128
 

- debugging JavaScript, 411–412
- displaying, 126, 412
- logging, 128
  - displaying warnings/errors, 131–134
  - id values of buttons, 130–131
  - log method, 128–130
- messages, 414–415
- variable scope, 413–414

const Keyword, 20

constants, math objects, 192–193

constraints, 532–536

constructors, 234–236

content, styling, 337–339
 

- classes
  - adding/removing, 340–342
  - tooggling values, 342–343

setting styles directly, 339–340

contextmenu events, 450–451

CSS (Cascading Style Sheets), 2–3
 

- cascading rules, 347
- custom properties/variables
  - defined, 346–348
  - setting complex values, 348–351
  - updating, 349
- emojis, 506–507
- selector syntax, 318–319
- styling web pages, 306–307

ctrlKey property, keyboard events, 460

curly brackets ({}), arguments, 261–262

custom HTML element attributes, 330–334

custom objects, creating, 222–226

custom properties/variables, CSS
 

- defined, 346–348
- setting complex values, 348–351
- updating, 349

---

## D

data properties, 201–204

data storage, Web Storage, 550
 

- adding data, 552–555
- coding, 552
- domains, 552
- file sizes, 556
- operation of, 550–551
- removing data, 555
- retrieving data, 554–555
- session storage, 557–558
- support, 556–557

data-\* attributes, HTML, 333–334

dataset property, custom HTML element attributes, 332–333



- date objects, 143, 180
- dblclick events, 421, 445–446
- dead zones, hoisting, 563
- debugging JavaScript
  - Console, 411–412
  - developer tools, 405–412
- declarations, destructuring arrays, 543–544
- declaring
  - numbers, 186–187
  - variables, 16, 19–20
    - hoisting, 561
    - using variables without declaring, 88–89
- decrementing
  - for loops, 66
  - variables, 188–189
- defer sync element, 477–478
- defining web page structures with HTML, 304–306
- delays, setTimeout function, 80–81
- deleting items from sets, 571–572
- destructuring, 539–541
  - arrays, 541–542, 546
    - commas (,), 542–543
    - declarations, 543–544
    - destructuring arrays, 543–544
    - variables, 543
  - objects, 544–547
- developer tools, 397
  - Console
    - debugging JavaScript, 405–412
    - logging messages, 414–415
    - variable scope, 413–414
  - debugging JavaScript, 405–412
  - displaying, 398–400
  - inspecting
    - DOM, 400–405
    - objects, 412–413
  - View Source command, 402–405
- displaying
  - Console, 412
  - console, 126
  - developer tools, 398–400
  - errors, 131–134
  - warnings, 131–134
- division (/) operators, 187–188
- document objects, 312–313
- DocumentFragment objects, adding DOM elements, 391–395
- documents
  - coding in
    - multiple documents, 118–120
    - a single document, 120
  - HTML documents, location/placement of code, 113–114
- document.write function, 59
- DOM (Document Object Model), 309, 360, 364
  - children, 355–356, 357–358
    - accessing elements of, 359
    - checking existence of, 357–358
    - null properties, 357–358
  - document objects, 312–313
  - elements
    - adding large amounts of elements, DocumentFragment objects, 391–395
    - adding large amounts of elements, general approach, 383–388
    - adding large amounts of elements, innerHTML, 388–390
    - changing text values, 326–328
    - cloning, 374–378
    - creating, 364–370
    - emptying subtrees, 395–396
    - events for multiple elements, 492–498
    - inserting, 368–372
    - modifying, 324–326
      - as objects, 322–324
      - removing, 372–373, 395–396
    - finding elements in, 316
      - CSS selector syntax, 318–319
      - querySelector function, 317
      - querySelectorAll function, 317–318
    - hierarchy of, 353–356
    - inspecting, 400–405
    - navigating, 354–356
    - nodes, 309–311
    - parents, 355–357
    - querySelector function, 317
    - querySelectorAll function, 317–318
    - scripts, locating, 473–476
    - siblings, 355–357
    - subtrees, emptying, 395–396
    - window objects, 311
  - domains, Web Storage, 552
  - DOMContentLoaded events, 421, 471–473
  - DOMMouseScroll events, 421, 454–455
  - dot notation, object properties, 214
  - double quotation marks ("), strings, 162, 163
  - do.while loops, 68–69
  - down/up, events, 430–434
  - duplicate code, 118–120
  - duplicates, checking for in sets, 567–569
  - dynamically loading scripts, 482–486
    - running dependent code, 488–489
    - running scripts, 486–488

---

## E

- element attributes, HTML
  - accessing, 328–330
  - custom attributes, 330–334

- data-\* attributes, 333–334
  - reading, 329
  - removing, 330
  - setting, 329–330
  - values, 328
- elements, DOM
  - adding large amounts of elements
    - DocumentFragment objects, 391–395
    - general approach, 383–388
    - innerHTML, 388–390
  - changing text values, 326–328
  - cloning, 374–378
  - creating, 364–370
  - emptying subtrees, 395–396
  - events for multiple elements, 492–498
  - inserting, 368–372
  - modifying, 324–326
    - as objects, 322–324
  - removing, 372–373, 395–396
- emojis
  - codepoints, specifying, 505–508
  - in coding, 174
  - CSS, 506–507
  - defined, 501–503
  - direct usage of, 504–505
  - HTML, 503–506
  - JavaScript, 507
- emptying subtrees, DOM, 395–396
- entries, sets, 574
- equality (==) operators, 44, 281–282, 286
- errors, displaying, 131–134
- events
  - addEventListener function, 420, 422
    - capturing events, 422
    - event handler, 421
    - event names, 421
    - sources, 420
  - arguments, 426–427
  - bubbling events, 435–436, 437–438
  - capturing, 422, 434–435, 437–438
  - contextmenu events, 450–451
  - defined, 418–419
  - DOMContentLoaded events, 421, 471–473
  - event handler, 421
  - example of, 423–425
  - going up/down, 430–434
  - HTTP requests, 523
  - interrupting, 438–441
  - JavaScript, 420
  - keyboard events, 458
    - arrow keys, 462
    - keydown events, 421, 458
    - keypress events, 458–459
    - keyup events, 421, 458
    - multiple key presses, 462–466
    - particular key presses, 461–462
    - properties, 460–466
  - listening for, 420–422, 427, 437
  - load events, 421
  - mouse events, 444
    - browser positioning, 452–453
    - button configurations, 453–454
    - click events, 421, 444–445, 446
    - contextmenu events, 450–451
    - dblclick events, 421, 445–446
    - DOMMouseScroll events, 421, 454–455
    - global mouse position, 451–452
    - mousedown events, 448–449
    - mouseenter events, 447
    - mouseleave events, 447
    - mousemove events, 421, 449
    - mouseout events, 421, 446–447
    - mouseover events, 421, 446–447
    - mouseup events, 448–449
    - mousewheel events, 421, 454–455
  - for multiple elements, 492–498
  - phases of, 434–437
    - choosing, 436–437
    - not specifying, 437
  - preventDefault function, 440–441
  - reacting to, 422–423
  - removing event listeners, 427
  - scroll events, 421
  - stopping, 438–441
  - stopPropagation function, 438–440
  - types of, 426–427
- existence of items in sets, checking, 572
- exiting functions early, 36
- expressions
  - complex expressions, if/else statements, 46–47
  - defined, 14
  - evaluation order, 187–188
  - functions, 36–38, 562–563
  - hoisting, 562–563
- extending objects, 240–244, 247–248
  - controversy, 255–257
  - functionality, 256
  - future of, 256
  - online resources, 257
  - prototype inheritance, 249–253
  - subclasses, 247–248

---

## F

false evaluations, conditional statements, 42–43, 46–48, 53–54, 56

false/true values  
 Boolean functions, 278–280  
 Boolean objects, 277–278  
 equality (==) operators, 281–282  
 inequality (!=) operators, 281–282

fetch API, HTTP requests, 514–520

files  
 JavaScript files, referencing, 115–116  
 sizes, Web Storage, 556

filling loops  
 incompletely, 66–67  
 without numbers, 66

filtering array items, 156–157

finding  
 elements in DOM, 316  
 CSS selector syntax, 318–319  
 querySelector function, 317  
 querySelectorAll function, 317–318  
 items in arrays, 152  
 scripts in DOM, 473–476  
 something inside strings, 169–171

for loops, 59–62, 64  
 backwards, going, 66  
 breaking, 65  
 conditions, 63  
 decrementing, 66  
 examples of, 65–67  
 filling  
 incompletely, 66–67  
 without numbers, 66  
 skipping iterations, 65–66  
 starting points, 62  
 steps, 62

function objects, 143, 180

functional programming, 160

functions  
 addEventListener function, 420, 422  
 capturing events, 422  
 event handler, 421  
 event names, 421  
 sources, 420  
 alert function, 124  
 appearance in classes, 239  
 arguments, creating  
 functions with, 30–34  
 arrow functions, 259, 263, 274  
 defined, 260  
 lexical scope, 273  
 bind method, 274–275  
 Boolean functions, 278–280  
 callback functions, 156, 159  
 calling, 26, 29–30  
 classes, appearance of  
 functions, 239  
 closures, 95–100  
 declaring, hoisting, 562  
 defined, 13, 26  
 document.write function, 59  
 expressions, 36–38, 562–563  
 “hello world” example, 13, 25  
 calling functions, 29–30  
 functions with arguments, 30–34  
 returning data, 35–38  
 simple functions, 26–30  
 hoisting  
 declaring functions, 562  
 expressions, 562–563  
 IIFE, 37–38  
 inner functions, functions  
 that aren’t self-contained, 100–106  
 preventDefault function, 440–441  
 querySelector function, 317  
 querySelectorAll function, 317–318  
 requestAnimationFrame  
 function, 83–84  
 returning data, 35  
 exiting early, 36  
 expressions, 35  
 return keyword, 35–36  
 self-contained, functions that  
 aren’t, 100–106  
 setInterval function, 81–83  
 setTimeout function, 80–81

simple functions, 26–30  
 stopPropagation function, 438–440  
 trigonometric functions, 194.0345  
 within functions, 96–100

---

## G

getters/setters, 205–206  
 logging activity, 206–207  
 property values validation, 207–208  
 shout generators, 206

global mouse position, mouse  
 events, 451–452

global scope, 86–88

Google Chrome developer  
 tools, 398  
 Console  
 debugging JavaScript, 405–412  
 logging messages, 414–415  
 variable scope, 413–414  
 debugging JavaScript, 405–412  
 displaying, 398–400  
 DOM inspections, 400–405  
 object inspections, 412–413  
 View Source command, 402–405

---

## H

handling events, 421  
 “hello world” example, 9, 13–14, 15–16  
 functions, 13, 25, 26–27  
 arguments, 30–34  
 calling, 29–30  
 returning data, 35–38  
 simple functions, 26–30  
 HTML document, 9–12  
 statements, 12  
 strings, 13  
 variables, 16–18, 24–25  
 changing values, 20

- declaring, 16, 19–20
  - initializing, 16, 19–20
  - naming, 18–19
- help, online resources, 579, 101.0025
- hexadecimal numbers, 190
- hoisting, 94
  - compiler behavior, 560–561
  - declaring
    - functions, 562
    - variables, 561
  - defined, 559–560
  - functions
    - declaring, 562
    - expressions, 562–563
  - ReferenceErrors, 563
  - temporal dead zones, 563
- HTML (HyperText Markup Language), 2–3, 304
  - defining web page structures, 304–306
  - designing for data, 390–391
  - element attributes
    - accessing, 328–330
    - custom attributes, 330–334
    - data-\* attributes, 333–334
    - reading, 329
    - removing, 330
    - setting, 329–330
    - values, 328
  - emojis, 503–506
  - “hello world” example, 9–12
  - innerHTML, adding DOM elements, 388–390
  - location/placement of code, 113–114
  - styling web pages with CSS, 306–307
- HTTP requests, 512–513
  - example of, 513
  - fetch API, 514–520
  - XMLHttpRequest objects, 520–521
    - creating requests, 521–522
    - events, 523

- processing requests, 523–527
  - running asynchronously, 523
  - sending requests, 522–523

---

**I**

---

- id values of buttons, 130–131
- if statements, 40–43, 46–47
- if/else statements, 40–43
  - complex expressions, 46–47
  - switch statement similarities, 53–55
  - using, 55–56
- if/else-if/else statements, 47–48
- if-only statements, 47
- IIFE (Immediately Invoked Function Expressions), 37–38
- in-browser developer tools, 397
  - Console
    - debugging JavaScript, 405–412
    - logging messages, 414–415
    - variable scope, 413–414
  - debugging JavaScript, 405–412
  - displaying, 398–400
  - inspecting
    - DOM, 400–405
    - objects, 412–413
  - View Source command, 402–405
- incompletely filling loops, 66–67
- incrementing/decrementing variables, 188–189
- indexing, strings, 169–171
- indexOf method, strings, 169–170
- individual characters, accessing in strings, 163–165
- inequality (!=) operators, 44, 281–282, 286

- Infinity values, 190
- inheritance, prototype, 229, 249–253
- initializing variables, 16, 19–20
- inner functions, functions that aren’t self-contained, 100–106
- innerHTML, adding DOM elements, 388–390
- inserting elements, DOM, 368–372
- inspecting
  - DOM, 400–405
  - objects, 412–413
- interpolation (template literals), strings, 175–177
- interrupting events, 438–441
- intervals, setInterval function, 81–83
- iterations (loops), skipping, 65–66
- iterators, keys, 574

## J

---

- JavaScript
  - appearance of, 8–9
  - debugging
    - Console, 411–412
    - developer tools, 405–412
  - defined, 7–8
  - emojis, 507
  - events, 420
  - expression evaluation order, 187–188
  - flexibility of, 7
  - popularity of, 7–8
  - predefined objects, 142–144
  - referencing files, 115–116
  - types
    - overview, 139–141
    - pizza example, 136–139
    - variables, processing, 93–94
- JSDoc-style comments, 75
- JSON (JavaScript Object Notation)
  - arrays, 296–297

Boolean values, 295  
 defined, 287–292  
 null values, 297  
 numbers, 294  
 object values, 295–296  
 objects, property names, 292–293  
 parsing JSON-looking data into actual JSON, 299–300  
 reading data, 297–299  
 strings, 293–294  
 syntax of, 287–292  
 values (overview), 293  
 writing data, 300

---

## K

keyboard events, 458  
 arrow keys, 462  
 keydown events, 458  
 keypress events, 458–459  
 keyup events, 458  
 multiple key presses, 462–466  
 particular key presses, 461–462  
 properties, 460–461  
 KeyCode property, keyboard events, 460–462  
 keydown events, 421  
 keys, sets, 573–574  
 keyup events, 421

---

## L

lastIndexOf method, strings, 170  
 lexical scope, arrow functions, 273  
 listening for events, 420–422, 427, 437  
 literal syntax, object, 213  
 load events, 421  
 loading  
 scripts dynamically, 482–486

running dependent code, 488–489  
 running scripts, 486–488  
 web pages, 468–469  
 async script element, 477  
 defer sync element, 482–486  
 DOMContentLoaded events, 471–473  
 script location in DOM, 473–476  
 stages of, 469–471  
 local scope, 88  
 locating scripts in DOM, 473–476  
 location/placement of code, 109–112  
 HTML documents, 113–114  
 <script> tags, 117  
 in separate files, 114–116  
 log method, 128–130  
 logging  
 activity, 206–207  
 Console, 128  
 displaying warnings/errors, 131–134  
 id values of buttons, 130–131  
 log method, 128–130  
 'messages, 414–415  
 loops  
 defined, 58  
 do.while loops, 68–69  
 for loops, 59–62, 64  
 breaking, 65  
 conditions, 63  
 decrementing, 66  
 examples of, 65–67  
 filling incompletely, 66–67  
 filling without numbers, 66  
 going backwards, 66  
 skipping iterations, 65–66  
 starting points, 62  
 steps, 62  
 sets, 572–573  
 while loops, 67–68  
 lowercasing strings, 171

---

## M

mapping array items, 154–156  
 match method, strings, 170–171  
 math objects, 143, 180, 191–193  
 merging arrays, 152–153  
 messages, logging, 414–415  
 MetaKey property, keyboard events, 460  
 minus sign (-) operator, 186, 187–188  
 modifying DOM elements, 324–326  
 mouse events, 444  
 browser positioning, 452–453  
 button configurations, 453–454  
 click events, 421, 444–445, 446  
 contextmenu events, 450–451  
 dblclick events, 421, 445–446  
 DOMMouseScroll events, 421, 454–455  
 global mouse position, 451–452  
 mousedown events, 448–449  
 mouseenter events, 447  
 mouseleave events, 447  
 mousemove events, 421, 449  
 mouseout events, 421, 446–447  
 mouseover events, 421, 446–447  
 mouseup events, 448–449  
 mousewheel events, 421, 454–455  
 multiline comments, 74–75  
 multiple documents, coding in, 118–120

multiple elements, events for, 492–498  
 multiple key presses, keyboard events, 462–466  
 multiplication (\*) operators, 187–188

---

## N

naming variables, 18–19  
 NaN (Not a Number) values, 191  
 navigating DOM, 354–356  
 negative numbers, 186  
 nested objects, properties, 216–217  
 nodes, DOM, 309–311  
 null primitives, 284  
 null properties, children (DOM), 357–358  
 null values, JSON objects, 297  
 number objects, 143, 180  
 numbers  
   absolute values, 196  
   declaring, 186–187  
   division (/) operators, 187–188  
   expression evaluation order, 187–188  
   filling loops without numbers, 66  
   hexadecimal numbers, 190  
   incrementing/decrementing variables, 188–189  
   Infinity values, 190  
   JSON objects, 294  
   math objects, 191–193  
   minus sign (-) operator, 186, 187–188  
   multiplication (\*) operators, 187–188  
   NaN values, 191  
   negative numbers, 186  
   Number method, 191  
   octal numbers, 190

percentage (%) operators, 187–188  
 plus sign (+) operators, 187–188  
 powers, 195–196  
 random numbers, 196–198  
 rounding numbers, 193–194  
 square roots, 195–196  
 strings, going to numbers, 191  
 trigonometric functions, 194.0345  
 using, 186

---

## O

objects, 212  
   array objects, 143, 180  
   behind the scenes  
     operations, 218–221  
   Boolean objects, 143, 180, 277–278  
   built-in objects, extending, 247–248  
     controversy, 255–257  
     functionality, 256  
     future of, 256  
     online resources, 257  
     prototype inheritance, 249–253  
     subclasses, 247–248  
   classes, creating objects, 232–234  
   creating, 213, 232–234  
   custom objects, creating, 222–226  
   date objects, 143, 180  
   destructuring, 544–547  
   document objects, 312–313  
   DocumentFragment objects, adding DOM elements, 391–395  
   DOM elements as objects, 322–324  
   extending, 240–244, 247–248  
     controversy, 255–257  
     functionality, 256  
     future of, 256  
     online resources, 257  
   prototype inheritance, 249–253  
   subclasses, 247–248  
   function objects, 143, 180  
   inspecting, 412–413  
   JSON objects  
     arrays, 296–297  
     Boolean values, 295  
     null values, 297  
     numbers, 294  
     property names, 292–293  
     strings, 293–294  
     values, 295–296  
     values (overview), 293  
   literal syntax, 213  
   math objects, 143, 180, 191–193  
   nested objects, properties, 216–217  
   number objects, 143, 180  
   predefined objects, 142–144  
   primitives  
     converting to objects, 182–183  
     object behavior as, 180–183  
   properties  
     adding to objects, 213–217  
     bracket notation, 214–215  
     dot notation, 214  
     nested objects, 216–217  
     removing, 217–218  
     this keyword, 226–229  
     undefined properties, 218  
   prototype chains, 220–221  
   prototype inheritance, 229, 249–253  
   RegExp objects, 143, 180  
   string objects, 143, 180  
   this keyword, 226–229  
   window objects, 311  
   XMLHttpRequest objects, 520–521  
     creating requests, 521–522  
     events, 523  
     processing requests, 523–527

- running asynchronously, 523
- sending requests, 522–523
- octal numbers, 190
- online resources
  - author's website, 579
  - built-in objects, 257
  - class values, 343
  - help, I01.0025
- operators
  - === operators
    - null primitives, 284
    - undefined primitives, 285–286
  - conditional operators, 43–46
  - equality (==) operators, 281–282, 286
  - incrementing/decrementing variables, 188–189
  - inequality (!=) operators, 281–282, 286

---

## P

- parentheses ( ( ) ), arguments, 261
- parents, DOM, 355–357
- parsing, 117, 299–300
- particular key presses, keyboard events, 461–462
- percentage (%) operators, 187–188
- pizza example, types, 136–139
- placement/location of code, 109–112
  - HTML documents, 113–114
  - <script> tags, 117
  - in separate files, 114–116
- plus sign (+) operators, 162, 165–166, 175, 187–188
- positioning mouse
  - browser positioning, 452–453
  - global positioning, 451–452
- powers/square roots, 195–196
- predefined objects, 142–144

- preventDefault function, 440–441
- primitives
  - converting to objects, 182–183
  - null primitives, 284
  - object behavior as, 180–183
  - undefined primitives, 284–286
- properties
  - accessor properties, 202–204
  - CSS custom properties/variables
    - defined, 346–348
    - setting complex values, 348–351
    - updating, 349
  - data properties, 201–204
  - dataset property, custom HTML element attributes, 332–333
  - JSON objects, 292–293
  - keyboard events, 460–461
  - objects
    - adding properties to objects, 213–217
    - bracket notation, 214–215
    - dot notation, 214
    - nested objects, 216–217
    - removing properties, 217–218
    - this keyword, 226–229
    - undefined properties, 218
    - removing, 217–218
    - undefined properties, 218
    - value validation, 207–208
- prototype chains, objects, 220–221
- prototype inheritance, 229, 249–253

---

## Q

- querySelector function, 317
- querySelectorAll function, 317–318
- quotation marks, strings, 162, 163

---

## R

- random numbers, 196–198
- reacting to events, 422–423
- reading
  - HTML element attributes, 329
  - JSON data, 297–299
  - values, data properties, 202
- reducing item values, arrays, 157–159
- ReferenceErrors, 563
- referencing JavaScript files, 115–116
- RegExp objects, 143, 180
- removing
  - classes, 340–342
  - data from Web Storage, 555
  - elements, DOM, 372–373, 395–396
  - event listeners, 427
  - HTML element attributes, 330
  - items from arrays, 151–152
  - properties, 217–218
  - values from classes, 342
- requestAnimationFrame function, 83–84
- requests, HTTP, 512–513
  - example of, 513
  - fetch API, 514–520
  - XMLHttpRequest objects, 520–521
    - creating requests, 521–522
    - events, 523
    - processing requests, 523–527
    - running asynchronously, 523
    - sending requests, 522–523
- resources, online
  - author's website, 579
  - built-in objects, 257
  - class values, 343
- retrieving data from Web Storage, 554–555



return keyword, 35–36  
 returning data with functions, 35  
   exiting early, 36  
   expressions, 35  
   return keyword, 35–36  
 rounding numbers, 193–194

## S

scope  
   block scoping, 89–92  
   global scope, 86–88  
   local scope, 88  
   variable scope, 413–414  
     block scoping, 89–92  
     global scope, 86–88  
     local scope, 88  
 scoping variables, 20  
 <script> tags  
   location/placement of code, 117  
   parsing, 117  
 scripts  
   async script element, 477  
   defer sync element, 477–478  
   loading dynamically, 482–486  
   running dependent code, 488–489  
   running scripts, 486–488  
   locating in DOM, 473–476  
 scroll events, 421  
 self-contained functions that aren't, 100–106  
 separate files, location/placement of code in, 114–116  
 session storage, Web Storage, 557–558  
 setInterval function, 81–83  
 sets, 565–566  
   adding items, 567  
   checking  
     for duplicates, 567–569  
     existence of items, 572  
     size of sets, 570–571  
   creating, 566, 569–570  
   deleting items, 571–572  
   entries, 574  
   existence of items in sets, checking, 572  
   iterators, 574  
   keys, 573–574  
   loops, 572–573  
   size of, 570–571  
   values, 573–574  
 setters/getters, 205–206  
   logging activity, 206–207  
   property values validation, 207–208  
   shout generators, 206  
 setTimeout function, 80–81  
 setting HTML element attributes, 329–330  
 shiftKey property, keyboard events, 460  
 shout generators, 206  
 siblings, DOM, 355–357  
 simple functions, 26–30  
 single documents, coding in, 120  
 single quotation marks ('), strings, 162, 163  
 single-line comments, 73–74  
 skipping iterations, loops, 65–66  
 slice method, strings, 167  
 smooth animation, request-Animation Frame function, 83–84  
 sources, addEventListener function, 420  
 split method, 168–169  
 splitting, strings, 168–169  
 square roots/powers, 195–196  
 starting points, for loops, 62  
 statements  
   conditional statements, 39–40  
   if statements, 40–48  
   if/else statements, 40–48  
   true/false evaluations, 42–43, 46–48, 53–54, 56  
   defined, 12  
   "hello world" example, 12  
   if statements, 40–43, 46–47  
   if/else statements, 40–43  
     complex expressions, 46–47  
     switch statement similarities, 53–55  
     using, 55–56  
   if/else-if/else statements, 47–48  
   if-only statements, 47  
   switch statements, 49–53  
     if/else statement similarities, 53–55  
     using, 55–56  
 steps for loops, 62  
 stopping  
   events, 438–441  
   webcam streams, 537  
 stopPropagation function, events, 438–440  
 storing data, Web Storage, 550  
   adding data, 552–555  
   coding, 552  
   domains, 552  
   file sizes, 556  
   operation of, 550–551  
   removing data, 555  
   retrieving data, 554–555  
   session storage, 557–558  
   support, 556–557  
 string objects, 143, 180  
 strings  
   backtick (') character, 175–177  
   combining (concatenating), 162, 165–166, 173–174  
   interpolation (template literals), 175–177  
   plus sign (+) operators, 175  
   defined, 13  
   finding something inside strings, 169–171



- “hello world” example, 13
- indexing, 169–171
- indexOf method, 169–170
- individual characters,
  - accessing in strings, 163–165
- interpolation (template literals), 175–177
- JSON objects, 293–294
- lastIndexOf method, 170
- lowercasing, 171
- match method, 170–171
- numbers, going to, 191
- plus sign (+) operators, 162, 165–166, 175
- quotation marks, 162, 163
- splitting, 168–169
- substrings, 166–167
  - slice method, 167
  - substr method, 167–168
- template literals (string interpolation), 175–177
- uppercasing, 171
- visualizing, 162–163

styling

- content, 337–339
  - adding/removing classes, 340–342
  - setting class styles directly, 339–340
  - toggling class values, 342–343
- web pages with CSS, 306–307

subclasses, extending objects, 253–255

substr method, strings, 167–168

substrings, getting out of strings, 166–167
 

- slice method, 167
- substr method, 167–168

subtrees (DOM), emptying, 395–396

support, Web Storage, 556–557

switch statements, 49–53
 

- if/else statement similarities, 53–55

using, 55–56

syntax

- CSS selector syntax, 318–319
- objects, 213

---

## T

template literals (string interpolation), 175–177

temporal dead zones, hoisting, 563

text values, changing in DOM elements, 326–328

this keyword, 226–229, 266–267
 

- redefined, 271–273
- variable scope, 268–271

timers, 79–80
 

- requestAnimationFrame function, 83–84
- setInterval function, 81–83
- setTimeout function, 80–81

toggling class values, 342–343

trigonometric functions, 194.0345

true/false evaluations
 

- conditional statements, 42–43, 46–48, 53–54, 56
- equality (==) operators, 281–282
- inequality (!=) operators, 281–282

true/false values
 

- Boolean functions, 278–280
- Boolean objects, 277–278

types
 

- defined, 141–142
- overview, 139–141
- pizza example, 136–139

---

## U

UI (User Interface), developing, 81

undefined primitives, 284–286

undefined properties, 218

updating, CSS custom properties/variables, 349

up/down, events, 430–434

uppercasing strings, 171

---

## V

validation, property values, 207–208

values
 

- absolute values, 196
- array values, accessing, 147–148
- class values
  - adding/removing, 340–342
  - toggling, 342–343
- classes
  - checking existence of class values, 343
  - online resources, 343
- HTML element attributes, 328
- JSON objects, 295–296
  - arrays, 296–297
  - Boolean values, 295
  - null values, 297
  - numbers, 294
  - overview, 293
  - strings, 293–294
- property values validation, 207–208
- reading/writing values, data properties, 202
- reducing item values, arrays, 157–159
- sets, 573–574
- text values, changing in DOM elements, 326–328
- variable scope, 413–414
  - block scoping, 89–92
  - closures, 95–100
  - global scope, 86–88
  - local scope, 88
  - this keyword, 268–271

variables
 

- changing values, 20

- combining (concatenating), 173–174
  - interpolation (template literals), 175–177
  - plus sign (+) operators, 175
- CSS custom properties/variables
  - defined, 346–348
  - setting complex values, 348–351
  - updating, 349
- declaring, 16, 19–20
  - hoisting, 561
  - using variables without declaring, 88–89
- destructuring
  - arrays, 543
  - objects, 546–547
- “hello world” example, 16–18, 24–25
  - changing values, 20
  - declaring, 16, 19–20
  - initializing, 16, 19–20
  - naming, 18–19
- hoisting, 94
- initializing, 16, 19–20
- interpolation (template literals), 175–177
- naming, 18–19
- processing in JavaScript, 93–94
- scoping, 20
- template literals (string interpolation), 175–177
- using without declaring, 88–89

View Source command, 402–405

visualizing, strings, 162–163

---

## W

- warnings, displaying, 131–134
- web browsers
  - developer tools, 397
  - debugging JavaScript, 405–412
  - displaying, 398–400
  - DOM inspection, 400–405
  - logging messages, 414–415
  - object inspection, 412–413
  - variable scope, 413–414
  - View Source command, 402–405
- mouse positioning, 452–453
- webcam access, 529–530
  - adding code, 532–535
  - constraints, 532–536
  - examining code, 535–537
  - example of, 530–531
  - overview, 531–532
  - stopping streams, 537
- web pages
  - building, 9
  - defining structures with HTML, 304–306
  - loading, 468–469
    - async script element, 477
    - defer sync element, 477–478
    - DOMContentLoaded events, 471–473
    - script location in DOM, 473–476
    - stages of, 469–471

- Web Storage, 550
  - adding data, 552–555
  - coding, 552
  - domains, 552
  - file sizes, 556
  - operation of, 550–551
  - removing data, 555
  - retrieving data, 554–555
  - session storage, 557–558
  - support, 556–557
- webcams, accessing, 529–530
  - adding code, 532–535
  - constraints, 532–536
  - examining code, 535–537
  - example of, 530–531
  - overview, 531–532
  - stopping streams, 537
- websites, author’s website, 579
- while loops, 67–68
- whitespace in comments, 76–77
- window objects, 311
- writing
  - JSON data, 300
  - values, data properties, 202

---

## X - Y - Z

- XMLHttpRequest objects, 520–521
  - creating requests, 521–522
  - events, 523
  - processing requests, 523–527
  - running asynchronously, 523
  - sending requests, 522–523