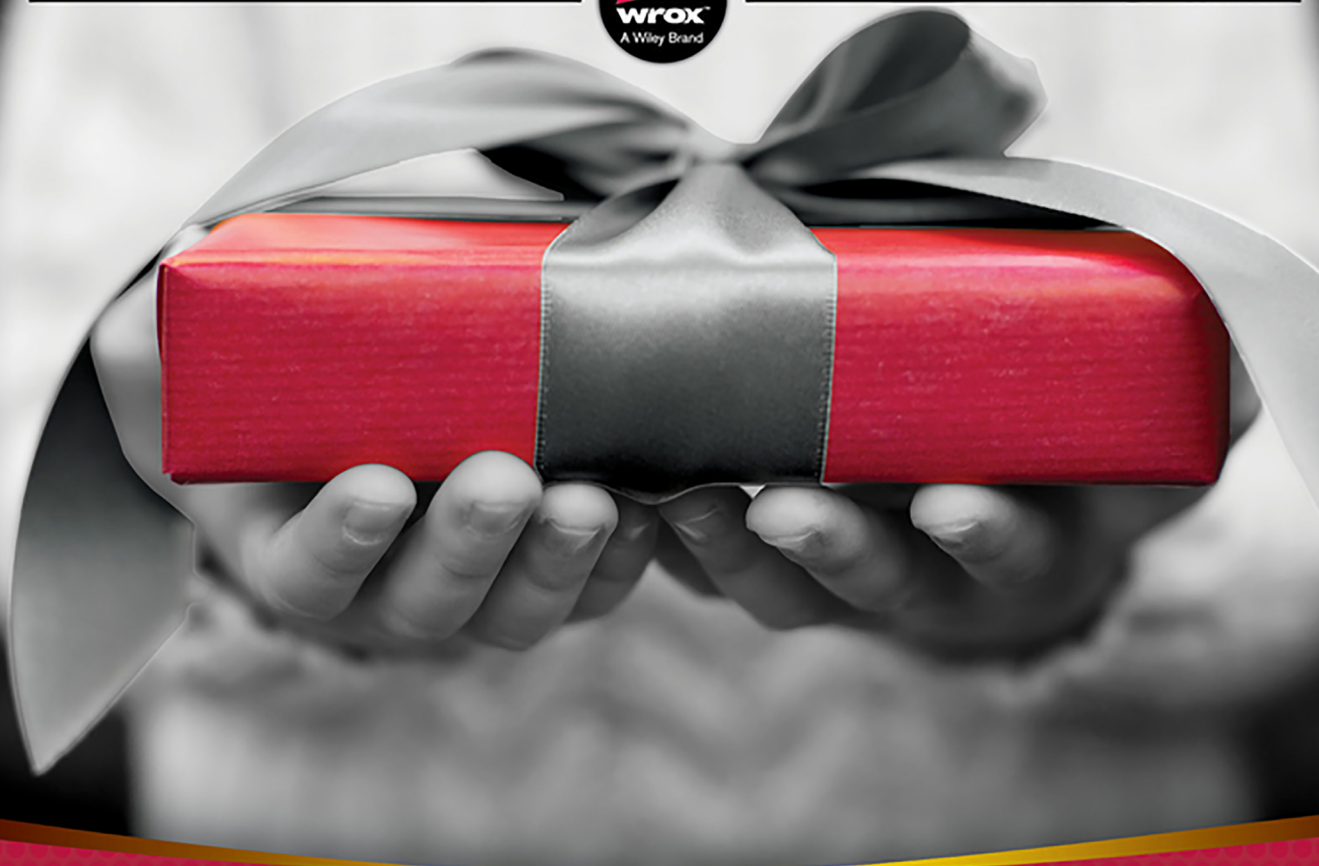


Covers ES2020



# JavaScript<sup>®</sup>

## The New Toys

T.J. Crowder



# JAVASCRIPT<sup>®</sup>

## THE NEW TOYS

---

INTRODUCTION .....	xxxi
CHAPTER 1 The New Toys in ES2015–ES2020, and Beyond .....	1
CHAPTER 2 Block-Scoped Declarations: let and const .....	17
CHAPTER 3 New Function Features .....	39
CHAPTER 4 Classes .....	65
CHAPTER 5 New Object Features .....	105
CHAPTER 6 Iterables, Iterators, for-of, Iterable Spread, Generators .....	131
CHAPTER 7 Destructuring .....	165
CHAPTER 8 Promises .....	181
CHAPTER 9 Asynchronous Functions, Iterators, and Generators .....	221
CHAPTER 10 Templates, Tag Functions, and New String Features .....	241
CHAPTER 11 New Array Features, Typed Arrays .....	263
CHAPTER 12 Maps and Sets .....	293
CHAPTER 13 Modules .....	319
CHAPTER 14 Reflection—Reflect and Proxy .....	365
CHAPTER 15 Regular Expression Updates .....	397
CHAPTER 16 Shared Memory .....	417
CHAPTER 17 Miscellany .....	461
CHAPTER 18 Upcoming Class Features .....	493
CHAPTER 19 A Look Ahead ... ..	517
APPENDIX Fantastic Features and Where to Find Them .....	539
INDEX .....	557



# **JavaScript®**

## **The New Toys**



# JavaScript®

THE NEW TOYS

T.J. Crowder



A Wiley Brand

# JavaScript®: The New Toys

Copyright © 2020 by Thomas Scott “T.J.” Crowder

Published simultaneously in Canada

ISBN: 978-1-119-36797-6

ISBN: 978-1-119-36797-0 (ebk)

ISBN: 978-1-119-36796-3 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

**Limit of Liability/Disclaimer of Warranty:** The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommendations it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

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

**Library of Congress Control Number:** 2018965303

**Trademarks:** Wiley, the Wiley logo, Wrox, the Wrox logo, Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. JavaScript is a registered trademark of Oracle America, Inc. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book.

*To Wendy and James, who've met late nights and  
working weekends with unstinting support and  
loving encouragement.*



# ABOUT THE AUTHOR

**T.J. CROWDER** is a software engineer with 30 years of professional experience, at least half that time working with JavaScript. He runs Farsight Software, a UK software consulting and product company. As one of the top 10 contributors on Stack Overflow and *the* top contributor in the JavaScript tag, he likes to use what he's learned to help others with the technical challenges they're facing, with an emphasis not just on imparting knowledge, but on helping with the process of solving problems.

T.J. started programming in his early teens in California, playing around with the Apple II and Sinclair ZX-80 and -81 using BASIC and assembly language. He got his first proper job with computers many years later, working a technical support position for a company with a PC (DOS) product for court reporters. While working support, he taught himself C from the TurboC manuals in the office, reverse-engineered the company's undocumented, compressed binary file format, and used that knowledge to create a much-requested feature for the product in his spare time (one the company soon started shipping). Before long the development department took him in and over the next several years gave him the opportunity, resources, and responsibility to grow into a programmer and, eventually, into their lead engineer, creating various products including their first Windows product.

At his next company he took on a professional services and developer education role that saw him customizing the company's enterprise product on-site for clients (using VB6, JavaScript, and HTML) and teaching training classes to the clients' developers; eventually he was writing the training classes. A move from the U.S. to London put him back in a straight development role where he was able to increase his software design, SQL, Java, and JavaScript skills before branching out into independent contracting.

Since then through circumstance he's been doing primarily closed-source remote development work for a variety of companies and organizations (a NATO agency, a UK local government authority, and various private firms) working primarily in JavaScript, SQL, C#, and (recently) TypeScript. The desire for community led him first to the PrototypeJS mailing list back in the day, then to Stack Overflow, and now to various platforms.

English and American by birth, American by upbringing, T.J. lives in a village in central England with his wife and son.



# ABOUT THE TECHNICAL EDITOR

**CHAIM KRAUSE** is an expert computer programmer with over thirty years of experience to prove it. He has worked as a lead tech support engineer for ISPs as early as 1995, as a senior developer support engineer with Borland for Delphi, and has worked in Silicon Valley for over a decade in various roles, including technical support engineer and developer support engineer. He is currently a military simulation specialist for the US Army's Command and General Staff College, working on projects such as developing serious games for use in training exercises. He has also authored several video training courses on Linux topics, and has been a technical reviewer for over two dozen books.



# ABOUT THE TECHNICAL PROOFREADER

**MARCIA K. WILBUR** is a technical communicator consulting in the semiconductor field, focusing on Industrial IoT (IIoT) and AI. Marcia holds degrees in computer science, technical communication, and information technology. As Copper Linux User Group president, she is heavily involved with the maker community leading West Side Linux + Pi and the East Valley leading regular Raspberry Pi, Beaglebone, Banana Pi/Pro, and ESP8266 Projects including home automation, gaming consoles, surveillance, network, multimedia, and other “pi fun”.



# ACKNOWLEDGMENTS

“I wrote a book” is almost never an accurate statement.

Sure, all the unquoted words in the book are mine. But none of them would be here if it weren't for others—others offering support, offering perspective, offering encouragement; others reviewing drafts to help cut the chaff and reveal the nugget of usefulness buried within; others asking questions in a variety of forums over the years, giving me the opportunity to practice the art and craft of explaining what meager knowledge I've been able to pick up; others answering *my* questions, directly and by pointing to resources; others helping hone my language and improve my wordcraft.

All of which waffle means: I owe thanks to a lot of people. This is my first proper book, and I made just about every mistake there is to make. The people below caught and corrected as many of them as they could. Any that remain are mine.

Primarily, thanks to my wife Wendy for her amazing support, for being an endless and willing source of strength and font of encouragement, for bringing me back when (as James Taylor puts it) I find myself careening in places where I should not let me go.

Thanks to my son James for putting up with Dad being squirrelled away in the office so much.

Thanks to my mother Virginia and father Norman for instilling in me a love of language, learning, reading, and writing. These are gifts that last a lifetime.

Thanks to my best friend Jock for being an ever-patient sounding board and source of encouragement—and, for that matter, for having been instrumental in getting me interested in programming in the first place.

Thanks to all of my editors and reviewers at Wiley & Sons: Jim Minatel and Pete Gaughan for supporting and maintaining the project even in the face of a frustrating author; David Clark and Chaim Krause for editorial and technical review; Kim Cofer for her excellent copyediting and help with grammar, syntax, and clarity; to Nancy Bell for her proofreading; to the artists and composers in the production department; and to all the others I've failed to mention by name for supporting all aspects of the project.

Thanks to Andreas Bergmaier for helping me keep my technical T's crossed and I's dotted—Andreas' keen eye and deep understanding were a huge help throughout the book.

Thanks to TC39 member Daniel Ehrenberg of Igalia for helping me better understand how TC39 works, and for kindly reviewing and helping me refine Chapter 1. His gentle corrections, input, and insight dramatically improved that chapter.

Thanks to TC39 member Lars T. Hansen of Mozilla (co-author of the shared memory and atomics JavaScript proposal) for his kind and invaluable help with getting the details and scope right in Chapter 16. His deep knowledge and perspective made all the difference there.

And finally, thanks to you, dear reader, for giving your time and attention to my efforts here. I hope it serves you well.



# CONTENTS

INTRODUCTION

xxxi

---

## CHAPTER 1: THE NEW TOYS IN ES2015–ES2020, AND BEYOND 1

---

Definitions, Who’s Who, and Terminology	2
Ecma? ECMAScript? TC39?	2
ES6? ES7? ES2015? ES2020?	2
JavaScript “Engines,” Browsers, and Others	3
What Are the “New Toys”?	4
How Do New Toys Get Created?	6
Who’s in Charge	6
The Process	7
Getting Involved	8
Keeping Up with the New Toys	9
Using Today’s Toys in Yesterday’s Environments, and Tomorrow’s Toys Today	10
Transpiling an Example with Babel	11
Review	15

---

## CHAPTER 2: BLOCK-SCOPED DECLARATIONS: LET AND CONST 17

---

An Introduction to let and const	18
True Block Scope	18
Repeated Declarations Are an Error	19
Hoisting and the Temporal Dead Zone	20
A New Kind of Global	22
const: Constants for JavaScript	24
const Basics	24
Objects Referenced by a const Are Still Mutable	25
Block Scope in Loops	26
The “Closures in Loops” Problem	26
Bindings: How Variables, Constants, and Other Identifiers Work	28
while and do-while Loops	33
Performance Implications	34
const in Loop Blocks	35
const in for-in Loops	36

---

Old Habits to New	36
Use const or let Instead of var	36
Keep Variables Narrowly Scoped	37
Use Block Scope Instead of Inline Anonymous Functions	37
<b>CHAPTER 3: NEW FUNCTION FEATURES</b>	<b>39</b>
<hr/>	
Arrow Functions and Lexical this, super, etc.	40
Arrow Function Syntax	40
Arrow Functions and Lexical this	44
Arrow Functions Cannot Be Constructors	45
Default Parameter Values	45
Defaults Are Expressions	46
Defaults Are Evaluated in Their Own Scope	47
Defaults Don't Add to the Arity of the Function	49
"Rest" Parameters	50
Trailing Commas in Parameter Lists and Function Calls	52
The Function name Property	53
Function Declarations in Blocks	55
Function Declarations in Blocks: Standard Semantics	57
Function Declarations in Blocks: Legacy Web Semantics	58
Old Habits to New	60
Use Arrow Functions Instead of Various this Value Workarounds	60
Use Arrow Functions for Callbacks When Not Using this or arguments	61
Consider Arrow Functions Elsewhere As Well	61
Don't Use Arrow Functions When the Caller Needs to Control the Value of this	62
Use Default Parameter Values Rather Than Code Providing Defaults	62
Use a Rest Parameter Instead of the arguments Keyword	63
Consider Trailing Commas If Warranted	63
<b>CHAPTER 4: CLASSES</b>	<b>65</b>
<hr/>	
What Is a Class?	66
Introducing the New class Syntax	66
Adding a Constructor	68
Adding Instance Properties	70
Adding a Prototype Method	70
Adding a Static Method	72
Adding an Accessor Property	73
Computed Method Names	74
Comparing with the Older Syntax	75

---

<b>Creating Subclasses</b>	<b>77</b>
The super Keyword	81
Writing Subclass Constructors	81
Inheriting and Accessing Superclass Prototype Properties and Methods	83
Inheriting Static Methods	86
super in Static Methods	88
Methods Returning New Instances	88
Subclassing Built-ins	93
Where super Is Available	94
<b>Leaving Off Object.prototype</b>	<b>97</b>
<b>new.target</b>	<b>98</b>
<b>class Declarations vs. class Expressions</b>	<b>101</b>
class Declarations	101
class Expressions	102
<b>More to Come</b>	<b>103</b>
<b>Old Habits to New</b>	<b>104</b>
Use class When Creating Constructor Functions	104
<b>CHAPTER 5: NEW OBJECT FEATURES</b>	<b>105</b>
<hr/>	
<b>Computed Property Names</b>	<b>106</b>
<b>Shorthand Properties</b>	<b>107</b>
<b>Getting and Setting an Object's Prototype</b>	<b>107</b>
Object.setPrototypeOf	107
The __proto__ Property on Browsers	108
The __proto__ Literal Property Name on Browsers	109
<b>Method Syntax, and super Outside Classes</b>	<b>109</b>
<b>Symbol</b>	<b>112</b>
Why Symbols?	112
Creating and Using Symbols	114
Symbols Are Not for Privacy	115
Global Symbols	115
Well-Known Symbols	119
<b>New Object Functions</b>	<b>120</b>
Object.assign	120
Object.is	121
Object.values	122
Object.entries	122
Object.fromEntries	122
Object.getOwnPropertySymbols	122
Object.getOwnPropertyDescriptors	123
<b>Symbol.toPrimitive</b>	<b>123</b>
<b>Property Order</b>	<b>125</b>

<b>Property Spread Syntax</b>	<b>127</b>
<b>Old Habits to New</b>	<b>128</b>
Use Computed Syntax When Creating Properties with Dynamic Names	128
Use Shorthand Syntax When Initializing a Property from a Variable with the Same Name	128
Use Object.assign instead of Custom “Extend” Functions or Copying All Properties Explicitly	129
Use Spread Syntax When Creating a New Object Based on an Existing Object’s Properties	129
Use Symbol to Avoid Name Collision	129
Use Object.getPrototypeOf/setPrototypeOf Instead of __proto__	129
Use Method Syntax for Methods	130

---

## **CHAPTER 6: ITERABLES, ITERATORS, FOR-OF, ITERABLE SPREAD, GENERATORS**

---

**131**

<b>Iterators, Iterables, the for-of Loop, and Iterable Spread Syntax</b>	<b>131</b>
Iterators and Iterables	132
The for-of Loop: Using an Iterator Implicitly	132
Using an Iterator Explicitly	133
Stopping Iteration Early	135
Iterator Prototype Objects	136
Making Something Iterable	138
Iterable Iterators	142
Iterable Spread Syntax	143
Iterators, for-of, and the DOM	144
<b>Generator Functions</b>	<b>146</b>
A Basic Generator Function Just Producing Values	147
Using Generator Functions to Create Iterators	148
Generator Functions As Methods	149
Using a Generator Directly	150
Consuming Values with Generators	151
Using return in a Generator Function	155
Precedence of the yield Operator	155
The return and throw Methods: Terminating a Generator	157
Yielding a Generator or Iterable: yield*	158
<b>Old Habits to New</b>	<b>163</b>
Use Constructs That Consume Iterables	163
Use DOM Collection Iteration Features	163
Use the Iterable and Iterator Interfaces	164
Use Iterable Spread Syntax in Most Places You Used to Use Function.prototype.apply	164
Use Generators	164

---

<b>CHAPTER 7: DESTRUCTURING</b>	<b>165</b>
Overview	165
Basic Object Destructuring	166
Basic Array (and Iterable) Destructuring	169
Defaults	170
Rest Syntax in Destructuring Patterns	172
Using Different Names	173
Computed Property Names	174
Nested Destructuring	174
Parameter Destructuring	175
Destructuring in Loops	178
Old Habits to New	179
Use Destructuring When Getting Only Some Properties from an Object	179
Use Destructuring for Options Objects	179
<b>CHAPTER 8: PROMISES</b>	<b>181</b>
Why Promises?	182
Promise Fundamentals	182
Overview	182
Example	184
Promises and “Thenables”	186
Using an Existing Promise	186
The then Method	187
Chaining Promises	187
Comparison with Callbacks	191
The catch Method	192
The finally Method	194
throw in then, catch, and finally Handlers	198
The then Method with Two Arguments	199
Adding Handlers to Already Settled Promises	201
Creating Promises	202
The Promise Constructor	203
Promise.resolve	205
Promise.reject	206
Other Promise Utility Methods	207
Promise.all	207
Promise.race	209
Promise.allSettled	209
Promise.any	210

<b>Promise Patterns</b>	<b>210</b>
Handle Errors or Return the Promise	210
Promises in Series	211
Promises in Parallel	213
<b>Promise Anti-Patterns</b>	<b>214</b>
Unnecessary new Promise(/*...*/)	214
Not Handling Errors (or Not Properly)	214
Letting Errors Go Unnoticed When Converting a Callback API	214
Implicitly Converting Rejection to Fulfillment	215
Trying to Use Results Outside the Chain	216
Using Do-Nothing Handlers	216
Branching the Chain Incorrectly	217
<b>Promise Subclasses</b>	<b>218</b>
<b>Old Habits to New</b>	<b>219</b>
Use Promises Instead of Success/Failure Callbacks	219
<b>CHAPTER 9: ASYNCHRONOUS FUNCTIONS, ITERATORS, AND GENERATORS</b>	<b>221</b>
<hr/>	
<b>async Functions</b>	<b>222</b>
async Functions Create Promises	224
await Consumes Promises	225
Standard Logic Is Asynchronous When await Is Used	225
Rejections Are Exceptions, Exceptions Are Rejections; Fulfillments Are Results, Returns Are Resolutions	227
Parallel Operations in async Functions	229
You Don't Need return await	230
Pitfall: Using an async Function in an Unexpected Place	231
<b>async Iterators, Iterables, and Generators</b>	<b>232</b>
Asynchronous Iterators	233
Asynchronous Generators	236
for-await-of	238
<b>Old Habits to New</b>	<b>238</b>
Use async Functions and await Instead of Explicit Promises and then/catch	238
<b>CHAPTER 10: TEMPLATES, TAG FUNCTIONS, AND NEW STRING FEATURES</b>	<b>241</b>
<hr/>	
<b>Template Literals</b>	<b>241</b>
Basic Functionality (Untagged Template Literals)	242
Template Tag Functions (Tagged Template Literals)	243
String.raw	248

---

Reusing Template Literals	249
Template Literals and Automatic Semicolon Insertion	250
<b>Improved Unicode Support</b>	<b>250</b>
Unicode, and What Is a JavaScript String?	250
Code Point Escape Sequence	252
String.fromCodePoint	252
String.prototype.codePointAt	252
String.prototype.normalize	253
<b>Iteration</b>	<b>255</b>
<b>New String Methods</b>	<b>256</b>
String.prototype.repeat	256
String.prototype.startsWith, endsWith	256
String.prototype.includes	257
String.prototype.padStart, padEnd	257
String.prototype.trimStart, trimEnd	258
<b>Updates to the match, split, search, and replace Methods</b>	<b>259</b>
<b>Old Habits to New</b>	<b>260</b>
Use Template Literals Instead of String Concatenation (Where Appropriate)	260
Use Tag Functions and Template Literals for DSLs Instead of Custom Placeholder Mechanisms	261
Use String Iterators	261
<b>CHAPTER 11: NEW ARRAY FEATURES, TYPED ARRAYS</b>	<b>263</b>
<b>New Array Methods</b>	<b>264</b>
Array.of	264
Array.from	264
Array.prototype.keys	266
Array.prototype.values	267
Array.prototype.entries	268
Array.prototype.copyWithWithin	269
Array.prototype.find	271
Array.prototype.findIndex	273
Array.prototype.fill	273
Common Pitfall: Using an Object As the Fill Value	273
Array.prototype.includes	274
Array.prototype.flat	275
Array.prototype.flatMap	276
<b>Iteration, Spread, Destructuring</b>	<b>276</b>
<b>Stable Array Sort</b>	<b>276</b>

---

<b>Typed Arrays</b>	<b>277</b>
Overview	277
Basic Use	279
Value Conversion Details	280
ArrayBuffer: The Storage Used by Typed Arrays	282
Endianness (Byte Order)	284
DataView: Raw Access to the Buffer	286
Sharing an ArrayBuffer Between Arrays	287
Sharing Without Overlap	287
Sharing with Overlap	288
Subclassing Typed Arrays	289
Typed Array Methods	289
Standard Array Methods	289
%TypedArray%.prototype.set	290
%TypedArray%.prototype.subarray	291
<b>Old Habits to New</b>	<b>292</b>
Use find and findIndex to Search Arrays Instead of Loops (Where Appropriate)	292
Use Array.fill to Fill Arrays Rather Than Loops	292
Use readAsArrayBuffer Instead of readAsBinaryString	292
 <b>CHAPTER 12: MAPS AND SETS</b>	 <b>293</b>
<hr/>	
<b>Maps</b>	<b>293</b>
Basic Map Operations	294
Key Equality	296
Creating Maps from Iterables	297
Iterating the Map Contents	297
Subclassing Map	299
Performance	300
<b>Sets</b>	<b>300</b>
Basic Set Operations	301
Creating Sets from Iterables	302
Iterating the Set Contents	302
Subclassing Set	303
Performance	304
<b>WeakMaps</b>	<b>304</b>
WeakMaps Are Not Iterable	305
Use Cases and Examples	305
Use Case: Private Information	305
Use Case: Storing Information for Objects Outside Your Control	307
Values Referring Back to the Key	308

---

<b>WeakSets</b>	<b>314</b>
Use Case: Tracking	314
Use Case: Branding	315
<b>Old Habits to New</b>	<b>316</b>
Use Maps Instead of Objects for General-Purpose Maps	316
Use Sets Instead of Objects for Sets	316
Use WeakMaps for Storing Private Data Instead of Public Properties	317
<b>CHAPTER 13: MODULES</b>	<b>319</b>
<hr/>	
<b>Introduction to Modules</b>	<b>319</b>
<b>Module Fundamentals</b>	<b>320</b>
The Module Specifier	322
Basic Named Exports	322
Default Export	324
Using Modules in Browsers	325
Module Scripts Don't Delay Parsing	326
The nomodule Attribute	327
Module Specifiers on the Web	328
Using Modules in Node.js	328
Module Specifiers in Node.js	330
Node.js is Adding More Module Features	331
<b>Renaming Exports</b>	<b>331</b>
<b>Re-Exporting Exports from Another Module</b>	<b>332</b>
<b>Renaming Imports</b>	<b>333</b>
<b>Importing a Module's Namespace Object</b>	<b>333</b>
<b>Exporting Another Module's Namespace Object</b>	<b>334</b>
<b>Importing a Module Just for Side Effects</b>	<b>335</b>
<b>Import and Export Entries</b>	<b>335</b>
Import Entries	335
Export Entries	336
<b>Imports Are Live and Read-Only</b>	<b>338</b>
<b>Module Instances Are Realm-Specific</b>	<b>340</b>
<b>How Modules Are Loaded</b>	<b>341</b>
Fetching and Parsing	342
Instantiation	344
Evaluation	346
Temporal Dead Zone (TDZ) Review	346
Cyclic Dependencies and the TDZ	347
<b>Import/Export Syntax Review</b>	<b>348</b>
Export Varieties	348
Import Varieties	350

---

<b>Dynamic Import</b>	<b>350</b>
Importing a Module Dynamically	351
Dynamic Module Example	352
Dynamic Import in Non-Module Scripts	356
<b>Tree Shaking</b>	<b>357</b>
<b>Bundling</b>	<b>359</b>
<b>Import Metadata</b>	<b>360</b>
<b>Worker Modules</b>	<b>360</b>
Loading a Web Worker as a Module	360
Loading a Node.js Worker as a Module	361
A Worker Is in Its Own Realm	361
<b>Old Habits to New</b>	<b>362</b>
Use Modules Instead of Pseudo-Namespaces	362
Use Modules Instead of Wrapping Code in Scoping Functions	363
Use Modules to Avoid Creating Megalithic Code Files	363
Convert CJS, AMD, and Other Modules to ESM	363
Use a Well-Maintained Bundler Rather Than Going Homebrew	363
<b>CHAPTER 14: REFLECTION—REFLECT AND PROXY</b>	<b>365</b>
<hr/>	
<b>Reflect</b>	<b>365</b>
Reflect.apply	367
Reflect.construct	367
Reflect.ownKeys	368
Reflect.get, Reflect.set	369
Other Reflect Functions	370
<b>Proxy</b>	<b>371</b>
Example: Logging Proxy	373
Proxy Traps	381
Common Features	381
The apply Trap	381
The construct Trap	382
The defineProperty Trap	382
The deleteProperty Trap	384
The get Trap	385
The getOwnPropertyDescriptor Trap	386
The getPrototypeOf Trap	387
The has Trap	388
The isExtensible Trap	388
The ownKeys Trap	388

---

The preventExtensions Trap	389
The set Trap	389
The setPrototypeOf Trap	390
Example: Hiding Properties	391
Revocable Proxies	394
<b>Old Habits to New</b>	<b>395</b>
Use Proxies Rather Than Relying on Consumer Code Not to Modify API Objects	395
Use Proxies to Separate Implementation Code from Instrumenting Code	395
<b>CHAPTER 15: REGULAR EXPRESSION UPDATES</b>	<b>397</b>
<hr/>	
<b>The Flags Property</b>	<b>398</b>
<b>New Flags</b>	<b>398</b>
The Sticky Flag (y)	398
The Unicode Flag (u)	399
The "Dot All" Flag (s)	400
<b>Named Capture Groups</b>	<b>400</b>
Basic Functionality	400
Backreferences	404
Replacement Tokens	405
<b>Lookbehind Assertions</b>	<b>405</b>
Positive Lookbehind	405
Negative Lookbehind	406
Greediness Is Right-to-Left in Lookbehinds	407
Capture Group Numbering and References	407
<b>Unicode Features</b>	<b>408</b>
Code Point Escapes	408
Unicode Property Escapes	409
<b>Old Habits to New</b>	<b>413</b>
Use the Sticky Flag (y) Instead of Creating Substrings and Using ^ When Parsing	413
Use the Dot All Flag (s) Instead of Using Workarounds to Match All Characters (Including Line Breaks)	414
Use Named Capture Groups Instead of Anonymous Ones	414
Use Lookbehinds Instead of Various Workarounds	415
Use Code Point Escapes Instead of Surrogate Pairs in Regular Expressions	415
Use Unicode Patterns Instead of Workarounds	415

---

<b>CHAPTER 16: SHARED MEMORY</b>	<b>417</b>
<b>Introduction</b>	<b>417</b>
<b>Here There Be Dragons!</b>	<b>418</b>
<b>Browser Support</b>	<b>418</b>
<b>Shared Memory Basics</b>	<b>420</b>
Critical Sections, Locks, and Condition Variables	420
Creating Shared Memory	422
<b>Memory Is Shared, Not Objects</b>	<b>426</b>
<b>Race Conditions, Out-of-Order Stores, Stale Values, Tearing, and More</b>	<b>427</b>
<b>The Atomics Object</b>	<b>429</b>
Low-Level Atomics Features	432
Using Atomics to Suspend and Resume Threads	433
<b>Shared Memory Example</b>	<b>434</b>
<b>Here There Be Dragons! (Again)</b>	<b>455</b>
<b>Old Habits to New</b>	<b>460</b>
Use Shared Blocks Rather Than Exchanging Large Data Blocks Repeatedly	460
Use Atomics.wait and Atomics.notify Instead of Breaking Up Worker Jobs to Support the Event Loop (Where Appropriate)	460
<b>CHAPTER 17: MISCELLANY</b>	<b>461</b>
<b>BigInt</b>	<b>462</b>
Creating a BigInt	462
Explicit and Implicit Conversion	463
Performance	464
BigInt64Array and BigUint64Array	465
Utility Functions	465
<b>New Integer Literals</b>	<b>465</b>
Binary Integer Literals	465
Octal Integer Literals, Take II	466
<b>New Math Methods</b>	<b>467</b>
General Math Functions	467
Low-Level Math Support Functions	468
<b>Exponentiation Operator (**)</b>	<b>468</b>
<b>Date.prototype.toString Change</b>	<b>470</b>
<b>Function.prototype.toString Change</b>	<b>471</b>
<b>Number Additions</b>	<b>471</b>
“Safe” Integers	471
Number.MAX_SAFE_INTEGER, Number.MIN_SAFE_INTEGER	472
Number.isSafeInteger	472