

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
CHAPTER 3: NEW FUNCTION FEATURES	39
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
CHAPTER 4: CLASSES	65
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

Creating Subclasses	77
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
Leaving Off Object.prototype	97
new.target	98
class Declarations vs. class Expressions	101
class Declarations	101
class Expressions	102
More to Come	103
Old Habits to New	104
Use class When Creating Constructor Functions	104
 CHAPTER 5: NEW OBJECT FEATURES	 105
Computed Property Names	106
Shorthand Properties	107
Getting and Setting an Object's Prototype	107
Object.setPrototypeOf	107
The __proto__ Property on Browsers	108
The __proto__ Literal Property Name on Browsers	109
Method Syntax, and super Outside Classes	109
Symbol	112
Why Symbols?	112
Creating and Using Symbols	114
Symbols Are Not for Privacy	115
Global Symbols	115
Well-Known Symbols	119
New Object Functions	120
Object.assign	120
Object.is	121
Object.values	122
Object.entries	122
Object.fromEntries	122
Object.getOwnPropertySymbols	122
Object.getOwnPropertyDescriptors	123
Symbol.toPrimitive	123
Property Order	125

Property Spread Syntax	127
Old Habits to New	128
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

Iterators, Iterables, the for-of Loop, and Iterable Spread Syntax	131
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
Generator Functions	146
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
Old Habits to New	163
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

CHAPTER 7: DESTRUCTURING	165
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
CHAPTER 8: PROMISES	181
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

Promise Patterns	210
Handle Errors or Return the Promise	210
Promises in Series	211
Promises in Parallel	213
Promise Anti-Patterns	214
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
Promise Subclasses	218
Old Habits to New	219
Use Promises Instead of Success/Failure Callbacks	219
 CHAPTER 9: ASYNCHRONOUS FUNCTIONS, ITERATORS, AND GENERATORS	 221
 async Functions	 222
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
async Iterators, Iterables, and Generators	232
Asynchronous Iterators	233
Asynchronous Generators	236
for-await-of	238
Old Habits to New	238
Use async Functions and await Instead of Explicit Promises and then/catch	238
 CHAPTER 10: TEMPLATES, TAG FUNCTIONS, AND NEW STRING FEATURES	 241
 Template Literals	 241
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
Improved Unicode Support	250
Unicode, and What Is a JavaScript String?	250
Code Point Escape Sequence	252
String.fromCodePoint	252
String.prototype.codePointAt	252
String.prototype.normalize	253
Iteration	255
New String Methods	256
String.prototype.repeat	256
String.prototype.startsWith, endsWith	256
String.prototype.includes	257
String.prototype.padStart, padEnd	257
String.prototype.trimStart, trimEnd	258
Updates to the match, split, search, and replace Methods	259
Old Habits to New	260
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
CHAPTER 11: NEW ARRAY FEATURES, TYPED ARRAYS	263
New Array Methods	264
Array.of	264
Array.from	264
Array.prototype.keys	266
Array.prototype.values	267
Array.prototype.entries	268
Array.prototype.copyWithin	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
Iteration, Spread, Destructuring	276
Stable Array Sort	276

Typed Arrays	277
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
Old Habits to New	292
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
 CHAPTER 12: MAPS AND SETS	 293
 Maps	 293
Basic Map Operations	294
Key Equality	296
Creating Maps from Iterables	297
Iterating the Map Contents	297
Subclassing Map	299
Performance	300
Sets	300
Basic Set Operations	301
Creating Sets from Iterables	302
Iterating the Set Contents	302
Subclassing Set	303
Performance	304
WeakMaps	304
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

WeakSets	314
Use Case: Tracking	314
Use Case: Branding	315
Old Habits to New	316
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
CHAPTER 13: MODULES	319
Introduction to Modules	319
Module Fundamentals	320
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
Renaming Exports	331
Re-Exporting Exports from Another Module	332
Renaming Imports	333
Importing a Module's Namespace Object	333
Exporting Another Module's Namespace Object	334
Importing a Module Just for Side Effects	335
Import and Export Entries	335
Import Entries	335
Export Entries	336
Imports Are Live and Read-Only	338
Module Instances Are Realm-Specific	340
How Modules Are Loaded	341
Fetching and Parsing	342
Instantiation	344
Evaluation	346
Temporal Dead Zone (TDZ) Review	346
Cyclic Dependencies and the TDZ	347
Import/Export Syntax Review	348
Export Varieties	348
Import Varieties	350

Dynamic Import	350
Importing a Module Dynamically	351
Dynamic Module Example	352
Dynamic Import in Non-Module Scripts	356
Tree Shaking	357
Bundling	359
Import Metadata	360
Worker Modules	360
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
Old Habits to New	362
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
CHAPTER 14: REFLECTION—REFLECT AND PROXY	365
Reflect	365
Reflect.apply	367
Reflect.construct	367
Reflect.ownKeys	368
Reflect.get, Reflect.set	369
Other Reflect Functions	370
Proxy	371
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
Old Habits to New	395
Use Proxies Rather Than Relying on Consumer Code Not to Modify API Objects	395
Use Proxies to Separate Implementation Code from Instrumenting Code	395
CHAPTER 15: REGULAR EXPRESSION UPDATES	397
The Flags Property	398
New Flags	398
The Sticky Flag (y)	398
The Unicode Flag (u)	399
The "Dot All" Flag (s)	400
Named Capture Groups	400
Basic Functionality	400
Backreferences	404
Replacement Tokens	405
Lookbehind Assertions	405
Positive Lookbehind	405
Negative Lookbehind	406
Greediness Is Right-to-Left in Lookbehinds	407
Capture Group Numbering and References	407
Unicode Features	408
Code Point Escapes	408
Unicode Property Escapes	409
Old Habits to New	413
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

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

Number.isInteger	473
Number.isFinite, Number.isNaN	473
Number.parseInt, Number.parseFloat	473
Number.EPSILON	473
Symbol.isConcatSpreadable	474
Various Syntax Tweaks	475
Nullish Coalescing	475
Optional Chaining	476
Optional catch Bindings	478
Unicode Line Breaks in JSON	478
Well-Formed JSON from JSON.stringify	479
Various Standard Library / Global Additions	479
Symbol.hasInstance	479
Symbol.unscopables	479
globalThis	480
Symbol description Property	481
String.prototype.matchAll	481
Annex B: Browser-Only Features	482
HTML-Like Comments	483
Regular Expression Tweaks	483
Control Character Escape (\cX) Extension	483
Tolerating Invalid Sequences	484
RegExp.prototype.compile	484
Additional Built-In Properties	484
Additional Object Properties	484
Additional String Methods	485
Various Bits of Loosened or Obscure Syntax	486
When document.all Isn't There . . . or Is It?	487
Tail Call Optimization	488
Old Habits to New	491
Use Binary Literals	491
Use New Math Functions Instead of Various Math Workarounds	491
Use Nullish Coalescing for Defaults	491
Use Optional Chaining Instead of && Checks	491
Leave the Error Binding (e) Off of "catch (e)" When Not Using It	492
Use the Exponentiation Operator (**) Rather Than Math.pow	492
CHAPTER 18: UPCOMING CLASS FEATURES	493
Public and Private Class Fields, Methods, and Accessors	493
Public Field (Property) Definitions	494

Private Fields	499
Private Instance Methods and Accessors	507
Private Methods	507
Private Accessors	511
Public Static Fields, Private Static Fields, and Private Static Methods	511
Public Static Fields	511
Private Static Fields	513
Private Static Methods	513
Old Habits to New	514
Use Property Definitions Instead of Creating Properties in the Constructor (Where Appropriate)	514
Use Private Fields Instead of Prefixes (Where Appropriate)	514
Use Private Methods Instead of Functions Outside the Class for Private Operations	514
CHAPTER 19: A LOOK AHEAD . . .	517
Top-Level await	518
Overview and Use Cases	518
Example	520
Error Handling	524
WeakRefs and Cleanup Callbacks	525
WeakRefs	525
Cleanup Callbacks	528
RegExp Match Indices	533
String.prototype.replaceAll	535
Atomics asyncWait	535
Various Syntax Tweaks	536
Numeric Separators	536
Hashbang Support	537
Legacy Deprecated RegExp Features	537
Thank You for Reading!	538
APPENDIX: FANTASTIC FEATURES AND WHERE TO FIND THEM	539
 <i>INDEX</i>	 557