<table>
<thead>
<tr>
<th>Symbols</th>
<th>Page References</th>
</tr>
</thead>
<tbody>
<tr>
<td>* (asterisk), 139–142, 157, 159, 175</td>
<td></td>
</tr>
<tr>
<td>** (exponentiation operator), 306–307</td>
<td></td>
</tr>
<tr>
<td>\ (backslash), 26</td>
<td></td>
</tr>
<tr>
<td>` (backtick), 26</td>
<td></td>
</tr>
<tr>
<td>: (colon), 69, 88</td>
<td></td>
</tr>
<tr>
<td>{ } (curly braces), 56–57, 88–89, 285</td>
<td></td>
</tr>
<tr>
<td>${ } (substitution delimiters), 28–29</td>
<td></td>
</tr>
<tr>
<td>... (rest items), 92–93</td>
<td></td>
</tr>
<tr>
<td>... (rest parameters), 44–46</td>
<td></td>
</tr>
<tr>
<td>... (spread operator), 47–48</td>
<td></td>
</tr>
<tr>
<td>converting sets to arrays, 126</td>
<td></td>
</tr>
<tr>
<td>nonarray iterables and, 151–152</td>
<td></td>
</tr>
<tr>
<td>= (equal sign), 87</td>
<td></td>
</tr>
<tr>
<td>== (equals operator), 72</td>
<td></td>
</tr>
<tr>
<td>=== (identically equals operator), 72</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>[] (square brackets), 70–71</td>
<td></td>
</tr>
</tbody>
</table>

**array buffers**

199–206

- creating, 199–200
- manipulating with views, 200–206
- reading and writing data, 202–203
- retrieving view information, 201
- type-specific views, 203–206

**arguments object**

- arrow functions and, 55, 60
- effect of default parameter values on, 38–39
- effect of rest parameters on, 45–46

- ArrayBuffer constructor, 200

**Arrow constructor**, 191–192

**Arrow.isArray() method**, 209

**Arrow.of() method**, 192–193

**Arrow.prototype.includes() method**, 307–308

**arrays**, 191–212

- arrow functions and, 60
- converting sets to, 126
- creating, 191–195
  - **Array.from() method**, 193–196
  - **Array.of() method**, 192–193
- creating objects that behave as, 244, 269–275
- deleting elements when reducing length, 272–273
- detecting array indexes, 270
- implementing MyArray class, 273–275
- increasing length when adding new elements, 270–271
- destructuring for, 90–93
- default values, 92
- destructuring assignment, 90–92
- mixed with object destructuring, 93–94
- nested array destructuring, 92
- rest items, 92–93
- methods for, 196–198
  - **copyWithin() method**, 197–198
  - **fill() method**, 197
  - **findIndex() method**, 196–197
  - **find() method**, 196–197
- typed, 198–206
  - array buffers, 199–206
  - element size, 206
  - numeric data types, 199
  - regular arrays versus, 207–211
  - views, 200–206

**arrow functions**, 54–61

- arguments binding, 60
- arrays and, 60
- compared to other functions, 54–55
- identifying, 61
arrow functions, continued
  immediately invoked function expressions, 57–58
  syntax for, 55–57
  this binding, 58–60
assignments, destructuring
  arrays, 90–92
  objects, 85–86
asterisk (*), 139–142, 157, 159, 175
asynchronous module loading, 295–296
asynchronous programming, 213–216.
  See also promises
callbacks, 215–216
events, 214–215
asynchronous task running, 159–164, 237–240
asynchronous task runner example, 161–164
await syntax, 240
  simple task runner example, 159–160
task running with data, 160–161
await syntax, 240

B
backslash (\), 26
backtick (‘), 26
Basic Multilingual Plane (BMP), 14, 16
bind() method, 49
block bindings, 1–12
declarations, 2–7
  const declarations, 4–6
  let declarations, 3–4
  redeclaration, 4
temporal dead zone, 6–7
  var declarations, 2–3
emerging best practices for, 12
global, 11–12
hoisting, 2–3
in loops, 7–11
  const declarations, 10–11
  functions, 8
  let declarations, 9–10
block-level functions, 52–54
in non-strict mode, 54
when to use, 53
block scopes (lexical scopes), 3
BMP (Basic Multilingual Plane), 14, 16
buffer property, 201, 205
built-in iterators, 145–151
collection iterators, 145–149
default for collection types, 148–149
  entries() iterator, 146
  keys() iterator, 147–148
  values() iterator, 146–147
  NodeList iterators, 151
string iterators, 149–150
built-in objects, 68
byteLength property, 201, 205
byteOffset property, 201, 205
BYTES_PER_ELEMENT property, 206

C
callbacks, 215–216
call() function, 61
[[Call]] method, 50–51
canonical equivalence, 16
catch() method, 218–219, 221, 225,
  228–229, 239–240
charAt() method, 13–14
charCodeAt() method, 13–15
checkArgs() method, 46
classes, 165–190
  accessor properties, 173–174
  class constructors
    callable, 267–268
    new.target in, 188–189
    overriding abstract, 266–267
class declarations, 166–169
  basic, 166–167
  syntax for, 167–169
class expressions, 169–171
  basic, 169–170
  named, 170–171
class-like structures in
  ECMAScript 5, 166
computed member names, 174–175
constant class names, 169
derived, 178–188
  from expressions, 181–183
  inherited static members, 181
inheriting from built-ins, 184–185
shadowing class methods, 180–181
Symbol.species property, 185–188
as first-class citizens, 172–173
generator methods, 175–176
Index

static members, 176–177, 181
using proxies as prototypes on, 279–282
class keyword, 166
clear() method
  for maps, 130
  for sets, 123–124
clone() method, 186–187
codePointAt() method, 15–16
code points, 14–15
code units, 13–14
collection iterators, 145–149
  default for collection types, 148–149
  entries() iterator, 146
  keys() iterator, 147–148
  values() iterator, 146–147
colon (:) 69, 88
computed member names, 174–175
computed property names, 70–71
concat() method, 93, 107–108, 210–211
concise method syntax, 69–70
console.log() method, 61, 103
const declarations, 4–5
  in global scope, 11
  let declarations versus, 5
  in loops, 10–11
  for objects, 6
  temporal dead zone, 6
  using by default, 11
[[Construct]] method, 50–51
constructors, 50–52
construct trap, 245, 262–267
copyWithin() method, 197–198, 207
CORS (Cross-Origin Resource Sharing), 296
count variable, 4
creator() function, 139–142
create() method, 181
Crockford, Douglas, 113
Cross-Origin Resource Sharing (CORS), 296
curly braces ({}), 56–57, 88–89, 285

D

DataView type, 200–201
default keyword, 289–290
default parameter values, 36–43
  default parameter expressions, 40–41
  for destructured parameters, 96–97
  in ECMAScript 5, 36
  in ECMAScript 6, 37–38
  effect on arguments object, 38–39
  object destructuring, 86–87
  temporal dead zone, 41–43
defineProperty trap, 245, 257–261
deflect() method
  for maps, 130
  for sets, 123–124
  for weak maps, 133–134
  for weak sets, 127–128
delete operator, 250
deleteProperty trap, 245, 250–252
derived classes, 178–188
  from expressions, 181–183
  inherited static members, 181
  inheriting from built-ins, 184–185
  shadowing class methods, 180–181
  Symbol.species property, 185–188
destructuring, 83–97
  for arrays, 90–93
    default values, 92
    destructuring assignment, 90–92
    nested array destructuring, 92
    rest items, 92–93
  for-of loops and, 149
  initializers and, 85
  mixed, 93–94
  for objects, 84–89
    assigning to different local variable names, 87–88
    default values, 86–87
    destructuring assignment, 85–86
    nested object destructuring, 88–89
    for parameters, 94–97
    default values, 96–97
    required, 95–96
    usefulness of, 84
domain-specific languages (DSLs), 25

E

ECMAScript 2016 (ECMAScript 7), 305
Array.prototype.includes() method, 307–308
exponentiation operator, 306–307
function-scoped strict mode, 308–309
endsWith() method, 19–20
entries() iterator, 145–146, 148, 207–208
enumerate trap, 245
equal sign (\*), 87
equals operator (==), 72
error-first callback style, 215
events, 214–215
evec() method, 23
exotic objects, 68
exponentiation operator (\*\*), 306–307
exporting to modules, 284–285
re-exporting bindings, 291–292
renaming exports, 288–289
export keyword, 284–285
extends keyword, 178
functions, 35–65. See also names of specific functions
arrow, 54–61
arguments binding, 60
arrays and, 60
compared to other functions, 54–55
identifying, 61
immediately invoked function expressions, 57–58
syntax for, 55–57
this binding, 58–60
block-level, 52–54
in non-strict mode, 54
when to use, 53
constructors, 50–52
with default parameter values, 36–43
default parameter expressions, 40–41
in ECMAScript 5, 36
in ECMAScript 6, 37–38
effect on arguments object, 38–39
temporal dead zone, 41–43
Function constructor, 46–47
in loops, 8
name property, 48–49
spread operator, 47–48
tail call optimization, 61–64
in ECMAScript 6, 62–64
making use of, 63–64
unnamed parameters, 43–44
in ECMAScript 5, 43–46
rest parameters, 44–46
generators, 139–142, 155–164
asynchronous task running, 159–164
asynchronous task runner example, 161–164
simple task runner example, 159–160
task running with data, 160–161
deleaguing, 156–159
function expressions for, 141
loops and complexity, 138
methods for, 175–176
object methods for, 142
return statements, 155–156
getFloat32() method, 202

F
factorial() method, 64
failure() method, 236–237
fill() method, 197–198, 207
filter() method, 207
findIndex() method, 196–197, 207
find() method, 196–197, 207
first-class citizens, classes as, 172–173
flags property, 24–25
Float32Array constructor, 204
Float64Array constructor, 204
forEach() method, 207
for maps, 131–132
for sets, 124–126
for-in loops
cost declarations in, 10–11
let declarations in, 9–10
own property enumeration order, 76
for loops, 7–9
cost declarations in, 10
let declarations in, 9–10
var declarations in, 7–8
for-of loops
cost declarations in, 10–11
destructuring and, 149
iterables and, 142–145
accessing default iterator, 143–144
creating iterables, 144–145
let declarations in, 9–10
from() method, 208–209
func() method, 63
Function constructor, 46–47
getFloat64() method, 202
getInt() method, 202–203
getInt16() method, 203
get() method
  for maps, 129
  for weak maps, 132–134
getOwnPropertyDescriptor trap, 245, 257–261
getPrototypeOf trap, 245, 252–254
get trap, 245
  object shape validation using, 247–249
  using on prototypes, 276–277
getInt8() method, 202
getValue() function, 2–3, 40–41
g flag, 22, 24
global block bindings, 11–12

H
handlers, 245
has() method
  for maps, 130
  for sets, 123
  for weak maps, 133–134
  for weak sets, 127–128
has trap, 245
  hiding property existence using, 249–250
  using on prototypes, 278–279
hoisting, 2–3, 53
[[HomeObject]] property, 80–81
HTML escaping, 25

I
identically equals operator (===), 72
i flag, 24
if statements, 121–122
immediately invoked function
  expressions (IIFEs), 8–9, 57–58, 135, 168
importing modules, 285–288
  entire module, 286–287
  multiple bindings, 286
  quirk of imported bindings, 288
  renaming imports, 288–289
  single binding, 286
  without bindings, 292
import keyword, 285
includes() method, 19–20, 307–308
indentLevel variable, 21
indexOf() method, 19–20, 196–197, 207
inheritance. See derived classes
  initializers, destructuring and, 85
init() method, 58–60
in operator, 122
instanceof, 50
Int8Array constructor, 204–205
Int16Array constructor, 204–207
Int32Array constructor, 204, 206
integers, 299–301
is extensible trap, 245, 255–257
iterables, 142
  creating typed arrays, 205
    for-of loops and, 142–145
      accessing default iterator, 143–144
      creating iterables, 144–145
      spread operator and nonarray, 151–152
      using Array.from() method on, 195–196
    iterators, 137–139, 142–155, 159–164
      asynchronous task running, 159–164
      asynchronous task runner example, 161–164
      simple task runner example, 159–160
      task running with data, 160–161
    built-in, 145–151
      collection iterators, 145–149
      NodeList iterators, 151
      string iterators, 149–150
    for-of loops and iterables, 142–145
      accessing default iterator, 143–144
      creating iterables, 144–145
      loops and complexity, 138
      passing arguments to, 152–154
      spread operator and nonarray iterables, 151–152
      throwing errors in, 154–155
i variable, 7–9

J
job queues, 214
job scheduling, 220
join() method, 207
json2.js, 113
JSON global object, 113
JSON.stringify() method, 76
K
keys() iterator, 145, 147–148, 207–208

L
lastIndexOf() method, 19–20, 196–197, 207
let declarations
  const declarations versus, 5
  in global scope, 11
  in loops, 9–10
  no redeclaration, 4
  syntax for, 3–4
  temporal dead zone, 6
lexical scopes (block scopes), 3
little-endian, 202
loading modules, 293–297
  asynchronous module loading, 295–296
  browser module specifier resolution, 297
  loading sequence, 294–295
<script> element, 293–294
  as workers, 296
localName variable, 87–88
localType variable, 87
logical OR operator (||), 36
loops. See also names of specific loops
  block bindings in, 7–11
    const declarations in loops, 10–11
    functions in loops, 8
    let declarations in loops, 9–10
  complexity and, 138

M
makeRequest() method, 37–38
Map constructor, 131
map() method, 207–208
maps, 119–122, 129–136
  array conversion with mapping functions, 194–195
  in ECMAScript 5, 120–122
forEach() method for, 131–132
initializing, 131
methods for, 130
rejection handling, 226–227
weak, 132–136
  initializing, 133
  limitations of, 136
methods for, 133–134
private object data, 134–135
using, 132–133
match() method, 18, 109
Math.max() method, 47–48
Math object methods, 301–302
MAX_SAFE_INTEGER property, 300
memory leaks, 127
metaproperties, 51
methods. See also names of specific methods
  for arrays, 196–198
    copyWithin() method, 197–198
    fill() method, 197
    findIndex() method, 196–197
    find() method, 196–197
    typed versus regular, 207–211
  formal definition, 80–81
  for generators, 142, 175–176
  for identifying substrings, 19–21
  for maps, 130
  math, 301–302
  on Object global, 71–74
    accessor properties, 74
    Object.assign() method, 72–74
    Object.is() method, 72
  for weak maps, 133–134
MIN_SAFE_INTEGER property, 300
mixArgs() function, 38–39
mixin() function, 73, 183
mixins, 72
modules, 283–298
  default values, 289–291
  exporting, 289–290
  importing, 290–291
exporting, 284–285
  re-exporting bindings, 291–292
  renaming exports and imports, 288–289
importing, 285–288
  entire module, 286–287
  multiple bindings, 286
quirk of imported bindings, 288
re-importing, 288–289
  single binding, 286
  without bindings, 292
loading, 293–297
  asynchronous module loading, 295–296
browser module specifier resolution, 297
loading sequence, 294–295
<script> element, 293–294
as workers, 296
syntax limitations, 287
module specifiers, 285, 297
multiline strings, 26–28

N
name property, 48–49, 55
choosing appropriate names, 48
concise method syntax, 70
special cases of, 49
nested array destructuring, 92
nested object destructuring, 88–89
new.target metaproperty, 51–52,
188–189, 265–266

next() method
for iterators, 138–140, 143
passing arguments to iterators,
152–154
return statements for generators,
155–156
simple task runners, 160
task running with data, 160–161
throwing errors in iterators,
154–155


NodeList iterators, 151
normalization forms, 16–17
normalize() method, 16–17
Number.isInteger() method, 300–301
Number.isSafeInteger() method, 300–301
numeric data types, for typed
arrays, 199

O
Object.assign() method, 72–75, 261–262
Object.create() method, 76, 116, 279
Object.defineProperties() method,
101–102
Object.defineProperty() method, 101,
106, 167, 243, 257–261, 275
object extensibility traps, 255–257
duplicate extensibility methods,
256–257
examples of, 255–256
Object.freeze() method, 248
Object.getOwnPropertyDescriptor() method,
257–261
Object.getOwnPropertyNames() method,
75–76, 104, 261–262
Object.getOwnPropertySymbols() method,
104, 261–262
Object.getPrototypeOf() method, 76,
78–79, 252–254, 303–304
object literals
duplicate properties, 75
syntax extensions, 68–71
computed property names,
70–71
concise method syntax, 69–70
property initializer shorthand,
68–69
Object.isExtensible() method, 255–256
Object.is() method, 72, 122
Object.keys() method, 76, 104, 261–262
Object.preventExtensions() method, 248,
255–257
Object.prototype.toString() method,
114–115

objects, 67–81
categories of, 68
destructuring for, 84–89
assigning to different local
variable names, 87–88
default values, 86–87
destructuring assignment,
85–86
mixed with array destructuring,
93–94
nested object destructuring,
88–89
duplicate object literal
properties, 75
method definition, 80–81
methods on Object global, 71–74
accessor properties, 74
Object.assign() method, 72–74
Object.is() method, 72
object literal syntax extensions,
68–71
computed property names,
70–71
concise method syntax, 69–70
property initializer shorthand,
68–69
own property enumeration order,
75–76
prototypes, 76–80
accessing with super references,
77–80
changing, 76–77
Object.seal() method, 248
Object.setPrototypeOf() method, 76–77,
252, 254, 303–304
of() method, 208–209
onclick event, 214–215
ordinary objects, 68
outputInfo() function, 86
ownKeys trap, 245, 261–262
own properties
   creating, 167
   enumeration order, 75–76

P
parameters
   default values, 36–43
      default parameter expressions, 40–41
destructured parameters, 96–97
in ECMAScript 5, 36
in ECMAScript 6, 37–38
   effect on arguments object, 38–39
doctor destructuring, 86–87
temporal dead zone, 41–43
destructured, 94–97
default values, 96–97
required, 95–96
unnamed, 43–46
   in ECMAScript 5, 43–44
   rest parameters, 44–46
validating, 264–265
pick() method, 44
pop() method, 210
p parameter, 64
preventExtensions trap, 245, 255–257
Promise constructor, 219
Promise.reject() method, 222–223
Promise.resolve() method, 222–223, 239–240
promises, 217–241
   asynchronous task running, 237–240
   chaining, 228–233
catching errors, 229–230
returning promises in promise
   chain, 231–233
   returning values in promise
   chain, 230–231
global rejection handling, 224–228
browser rejection handling, 227–228
Node.js rejection handling, 225–227
inheriting from, 236–237
life cycle of, 217–219
responding to multiple, 233–236
   Promise.all() method, 234–235
   Promise.race() method, 235–236
settled, 221–224
executor errors, 224
unsettled, 219–221
property descriptor traps, 257–261
blocking Object.defineProperty(), 258–259
defineProperty() methods, 260–261
descriptor object restrictions, 259–260
duplicate descriptor methods, 260
getOwnPropertyDescriptor() methods, 261
property initializer shorthand syntax, 68–69
[Proto__ property, 303–304
prototype property, 54
[[Prototype]] property, 77
prototypes, 76–80
   accessing with super references, 77–80
   changing, 76–77
   prototype proxy traps, 252–255
      function of, 252–253
      purpose of two sets of methods, 254–255
using proxies as, 275–282
   on classes, 279–282
   using get trap on prototypes, 276–277
   using has trap on prototypes, 278–279
   using set trap on prototypes, 277–278
proxies, 243–282
   creating, 244, 245–246
function proxies, 262–268
   callable class constructors, 267–268
calling constructors without
   new, 265–266
overriding abstract base class
   constructors, 266–267
validating function parameters, 264–265
hiding property existence, 249–250
mimicking array behavior, 244, 269–275
deleting elements when reducing length, 272–273
detecting array indexes, 270
implementing the MyArray class, 273–275
increasing length when adding new elements, 270–271
object extensibility traps, 255–257
duplicate extensibility methods, 256–257
examples of, 255–256
object shape validation, 247–249
ownKeys trap, 261–262
preventing property deletion, 250–252
property descriptor traps, 257–261
blocking Object.defineProperty(), 258–259
defineProperty() methods, 260–261
descriptor object restrictions, 259–260
duplicate descriptor methods, 260
getOwnPropertyDescriptor() methods, 261
prototype proxy traps, 252–255
function of, 252–253
purpose of two sets of methods, 254–255
revocable proxies, 268–269
using as prototypes, 275–282
on classes, 279–282
using set trap on prototypes, 277–278
validating properties, 246–247
Proxy.revocable() method, 268–269
push() method, 210
Reflect.apply() method, 263
Reflect.construct() method, 263, 265–267
Reflect.defineProperty() method, 257, 259–261
Reflect.deleteProperty() method, 251
reflect() function, 56
Reflect.get() method, 248, 273
Reflect.getOwnPropertyDescriptor() method, 257, 260–261
Reflect.getPrototypeOf() method, 252, 254
Reflect.has() method, 249–250
reflection API, 244. See also proxies
Reflect.isExtensible() method, 255–256
Reflect object, 244
Reflect.ownKeys() method, 75–76
Reflect.preventExtensions() method, 255–257
Reflect.set() method, 246–247, 271, 278
Reflect.setPrototypeOf() method, 252, 254
RegExp variable, 11, 19, 23–24
regular expressions
  duplicating, 23–24
  flags property, 24–25
  importance of, 13
  u flag, 18–19
  y flag, 21–23
rejectionhandled event, 227–228
rejectionHandled event, 225–226
reject() method, 219–220, 231
repeat() method, 20–21
replace() method, 109
resolve() method, 219–221, 230
rest items (…), 92–93
rest parameters (…), 44–46
return statements, for generators, 155–156
reverse() method, 207
revoke() function, 269
run() function, 160, 239–240
S
<script> element, 293–294
scripts, 284
search() method, 109
Set constructor, 122–123
setCookie() function, 95–96
setFloat32() method, 202
setFloat64() method, 202
setInt8() method, 202–203
setInterval() function, 220, 226
set keyword, 173
set() method, 132, 211
setPrototypeOf trap, 245, 252–254
sets, 119–129, 136
  adding items, 122–123
  converting to arrays, 126
  creating, 122
  in ECMAScript 5, 120–122
  forEach() method for, 124–126
  removing items, 123–124
  weak, 127–129
  creating, 127–128
  regular versus, 128–129
setTimeout() function, 220–221
settled promises, 217, 221–224
  executor errors, 224
set trap, 245
  using on prototypes, 277–278
  validating properties using, 246–247
setInt8() method, 202
shift() method, 210
single-threaded event loops, 214
slice() method, 185, 194, 200, 207
some() method, 207
sort() method, 207
splice() method, 210
split() method, 109
spread operator (...), 47–48
  converting sets to arrays, 126
  nonarray iterables and, 151–152
square brackets ([]), 70–71
standard objects, 68
startsWith() method, 19–20
static keyword, 177
static members, 176–177, 181
step() method, 160, 239–240
sticky regular expressions, 21–23
strict mode, changes to, 308–309
String.fromCharCode() method, 16
String() function, 103–104
string iterators, 149–150
String.raw() tag, 31–32
strings, 13–33
  importance of, 13
  methods for identifying substrings, 19–21
repeat() method, 20–21
template literals, 25–32
  making substitutions, 28–29
  multiline strings, 26–28
  syntax for, 26
  tagged templates, 29–32
Unicode support, 13–19
  codePointAt() method, 15–16
  normalize() method, 16–17
  String.fromCharCode() method, 16
  UTF-16 code points, 14–15, 18
subarray() method, 211
substitutions, 28–29
substrings, methods for identifying, 19–21
success() method, 236–237
sum() function, 56, 61, 264
super() method, 77–80, 179, 180, 183, 189
supplementary planes, 14
surrogate pairs, 14
Symbol.for() method, 102–103
Symbol function, 100
Symbol.hasInstance() method, 105–107
Symbol.isConcatSpreadable property, 105, 107–108
Symbol.iterator() method, 105, 142–145, 176
Symbol.keyFor() method, 103
Symbol.match property, 105, 109–110
Symbol.replace property, 105, 109–110
symbols, 99–117
  coercion of, 103
  creating, 100
  exposing internal operations with, 105–116
  identifying, 101
  retrieving properties of, 104–105
  sharing, 102–103
  using, 101–102
Symbol.search property, 105, 109–110
Symbol.species property, 105, 185–188
Symbol.split property, 105, 109–110
Symbol.toPrimitive() method, 105, 111–112
Symbol.toStringTag property, 105, 112–115
  defining object string tags, 114–115
  identification problem
  workaround, 113
Symbol.unscopables property, 105, 115–116
tagged templates, 29–32
  defining tags, 30–31
  using raw values in template literals, 31–32
tail call optimization, 61–64
  in ECMAScript 6, 62–63
  making use of, 63–64
targets, 244–246
TDZ (temporal dead zone), 6–7, 41–43
template literals, 25–32
  multiline strings, 26–28
  substitutions, 28–29
  syntax for, 26
tagged templates, 29–32
  defining tags, 30–31
  raw values, 31–32
temporal dead zone (TDZ), 6–7, 41–43
test() method, 23
then() method, 217–219, 221–223, 225, 228–229
this binding, 54, 58–60
throw() method, 154–155
toString() method, 100, 111, 113–114
traps, 244–245
  function proxies with, 262–268
    callable class constructors, 267–268
    calling constructors without new, 265–266
    overriding abstract base class constructors, 266–267
    validating function parameters, 264–265
hiding property existence using, 249–250
object extensibility, 255–257
  duplicate extensibility methods, 256–257
  examples of, 255–256
object shape validation using, 247–249
ownKeys, 261–262
preventing property deletion with, 250–252
property descriptor, 257–261
  blocking Object.defineProperty(), 258–259
  defineProperty() methods, 260–261
  descriptor object restrictions, 259–260
  duplicate descriptor methods, 260
  getOwnPropertyDescriptor() methods, 261
  purpose of, 254–255
  validating properties using, 246–247
trim() method, 28
type coercion, 103–104
typed arrays, 198–206
  array buffers, 199–206
  creating, 199–200
  manipulating with views, 200–206
  element size, 206
  numeric data types, 199
  regular arrays versus, 207
  behavioral differences between, 209–210
  iterators, 208
  methods in common, 207–208
  methods missing from typed arrays, 210–211
  methods present in typed arrays, 211
  of() and from() methods, 208–209
typeof operator, 6–7, 36, 101
u flag, 18–19
Uint8Array constructor, 204
Uint8ClampedArray constructor, 204
Uint16Array constructor, 204
Uint32Array constructor, 204
unhandledrejection event, 227–228
unhandledRejection event, 225–226
Unicode support, 13–19
  codePointAt() method, 15–16
  identifiers, 302–303
  normalize() method, 16–17
  String.fromCharCode() method, 16
  u flag, 18–19
  UTF-16 code points, 14–15, 18
unnamed parameters, 43–46
  in ECMAScript 5, 43–44
  rest parameters, 44–46
unsettled promises, 217, 219–221
unshift() method, 210
UTF-16 code points, 14–15, 18

V
valueOf() method, 111
values() iterator, 145–148, 176, 207–208
value variable, 2–4, 6
var declarations, 2–3
in global scope, 11–12
in loops, 7–8
views, manipulating array buffers with, 200–206

W
WeakMap constructor, 133
weak maps, 132–136
initializing, 133
limitations of, 136
methods for, 133–134
private object data, 134–135
using, 132–133
weak references, 127
WeakSet constructor, 127–128
weak sets, 127–129
creating, 127–128
regular sets versus, 128–129
WebGL, 198–199
well-known symbols, 105
window object, 11
with statements, 115–116
workers, 296
writeFile() function, 216

Y
y flag, 21–23
yield keyword, 139–141, 157, 159