

INDEX

A

- abstract classes, 171–172
- abstract keyword, 171–172
- abstract methods, 172
- addClickListener function, 71–72
- allowJs setting, 308
- allowUnreachableCode option, 314
- ambient context, 291, 302–303
- angle brackets (<>), 48, 56, 230, 333
- annotating
 - empty objects, 269–270
 - empty parameters, 39–40
 - with Equal type, 38
 - function parameters, 34, 36–37
 - types
 - any, 41–42
 - basic, 34, 40–41
 - excess property checking, 271
 - type inference, 35–37
 - values, 225–229
 - variables, 35–36
 - vs. as const, 151
 - vs. as and satisfies, 242–245
 - vs. values, 225–229
- any type, 37–38
 - annotations with, 41–42
 - assigning to never, 280
 - evolving, 249–250
 - vs. unknown, 91
- application development, 359
- arguments
 - adding to constructors, 161
 - in generic types, 336
 - types, 333
- array methods, 295
- arrays, 49
 - creating, 48–49
 - mutation of, 148–149
 - named tuples, 50–55
 - of objects
 - creating, 52–54
 - inferring, 146–148, 153–156
 - read-only and mutable, 144–145
 - tuples, 50
- Array type, 48, 51–52
- arrow functions
 - to define class methods, 165–166
 - implementing class methods, 174–175
 - and this keyword, 265
- as any tool, 234
- as const assertion, 150–151
 - arrays, 210–212
 - creating deeply read-only objects, 246–247
 - deep immutability with, 150–153
 - inferring literal values in arrays, 153–156
 - for JavaScript-style enums, 203–205
 - vs. Object.freeze, 151–153
 - vs. variable annotation, 151
- as keyword
 - for assertions, 229–231
 - providing additional information, 236–238
 - remapping keys, 343–344, 356–357
 - vs. satisfies and variable annotations, 242–245
- assertion functions, 368–369, 384–386
- assertions
 - as keyword, 229–231
 - forcing type of values, 229–232
 - never type, 280
 - non-null, 231–232
 - providing additional information, 236–238
 - solving issues with, 238–240
- asserts keyword, 368–369, 385–386

- assignability of types, 90, 93, 257
- asynchronous (async) functions
 - Awaited utility type, 213
 - expecting certain properties, 135–136
 - and generic types, 379–381
 - typing, 64–65, 73–75
 - unwrapping promises, 217–218
- attribute getters, 354–355
- augmenting modules, 294–295
- autocomplete, 9–11
- automated testing, 5
- Awaited utility type, 213

B

- babel tool, 311
- basic types, 34, 38–39, 40–41
- bigint type, 34
- book repository, xxvi
- boolean type, 34
- browsers, 24–25
- Bun, 28
- bundlers, 314–316

C

- C#, xxiv
- Capitalize type, 339
- classes
 - creating, 159–162, 173–174
 - extending, 167–168, 177–178
 - generic, 363
 - getters, 175–176
 - inheritance
 - abstract classes, 171–172
 - abstract methods, 172
 - implements keyword, 169–171
 - overriding, 168–169
 - protected properties, 168
 - methods, 165–166, 174–175
 - parameter properties, 181–182
 - properties, 162–165
 - property initializers, 162–163
 - setters, 177
 - in types and values, 259
 - using as types, 161–162
- class keyword, 159

- CLI (command line interface), 5, 25, 27–28
- closed object types, 253–254
- colon (:), 33, 61, 62
- comment directives, 232–234
- comments, 16–17
- CommonJS, 310
- composite option, 308, 326
- conditional types, 340–341
- configuring TypeScript
 - base options, 310–312
 - jsx option, 319–320
 - module options, 314–316
 - multiple configurations
 - extending configurations, 323–325
 - finding *tsconfig.json* files, 321–323
 - project, 325
 - project references, 325–326
 - Node support, 320–321
 - noEmit, 316
 - recommended configuration, 307–309
 - source maps, 316
 - strictness options, 312–314
 - transpiling code for library use, 316–319
- console.log function, 64
- constants, 21
- const enums, 187
- const keyword, 140–141
- constructors, 160–161, 167–168
- curly brackets ({})
 - empty object annotation, 269–270
 - empty object types, 256–258
 - object literal types, 43–44

D

- declaration setting, 308, 317
- declaration files, 288–291. *See also* *tsconfig.json* files
 - authoring, 299–302
 - creating, 317–318
 - declare keyword, 291–295
 - from TypeScript and third parties, 295–299

- declaration maps, 318–319
- declarationMap setting, 308, 319
- declaration merging, 116–117, 189–190, 299–300
- declarations, 14–16
- declare keyword, 291
 - const enum, 311
 - global, 292–293
 - global scope, 301
 - global variables, 291–292
 - module, 293–295
- decoupling vs. deriving, 221–223
- deeply read-only objects, 246–247
- deeply Required type, 131
- default parameters, 61–62, 66–67
- DefinitelyTyped GitHub repository, 298–299
- deriving types
 - creating values from types, 200–201
 - vs. decoupling, 221–223
 - from functions, 212–214
 - indexed access types, 201–205
 - from other types, 198–200
 - transforming, 218–220
- diff library, 298
- discriminants, 98
- discriminated tuples, 103–105
- discriminated unions, 98–100
 - bag of optionals problem, 97–98
 - destructuring, 100–102
 - handling defaults with, 105–107
 - narrowing with switch statement, 102–103
- dist* directory, 317
- DistributiveOmit type, 134
- DistributivePick type, 134
- documentation, 16–17
- document.getElementById function, 57
- document global, 296–297
- document.querySelector function, 57–58
- dom.iterable function, 297
- DOM types, 296–297
- .d.ts* files, 288–291

- dynamic keys
 - combining with known keys, 122–123
 - default properties with, 125–126
 - index signatures, 121–122
 - using, 124
 - PropertyKey type, 123–124, 128
 - Record type, 122
 - support for, 127–128

E

- ECMAScript, 191–192, 310
- empty object type (`{}`), 256–258, 269–270
- empty parameters, 39–40
- enum keyword, 182
- enums
 - as const for JavaScript-style, 203–205
 - numeric, 182–187
 - string, 183–187
 - transpiling, 184–185
 - in types and values, 260
 - whether to use, 187
 - workings of, 184–187
- environment variables, 305
- env property, 305
- Equal type annotation, 38
- erasableSyntaxOnly flag, 192, 321
- Error class, 94
- errors
 - checking, 11–14
 - multiline, 13–14
 - narrowing with instanceof, 93–94
 - runtime, 11–12
 - suppressing, 232–236
 - TError, 347–348
 - @ts-expect-error directive, xxvii, 232–233
 - in TypeScript CLI, 27–28
- es5 option, 296
- es2022 option, 296
- ESbuild, 309
- esbuild tool, 311
- esModuleInterop flag, 308, 310–311
- exactOptionalPropertyTypes option, 314
- @example tag, 16

- excess property warnings
 - comparing functions, 252–253
 - fresh and stale objects, 254–255
 - in functions, 272–274
 - in objects, 270–272
 - open vs. closed objects, 253–254
 - on variables, 251–252
- Exclude utility type, 218–219, 220
- exercises in book, xxvi–xxviii
- Expect annotation, 38
- experimental-strip-types flag, 320
- export statement, 285–286
- extends keyword
 - with abstract classes, 172
 - constraining generic type parameters, 379
 - constraining Result types, 349
 - creating new interfaces, 114
 - extending classes, 167–168
 - setting parameter constraints, 337
 - uses of, 338
- Extract utility type, 220

F

- feedback loop, 5
- fetch function, 73–74
- files: [], 325–326
- Flow language, 253–254
- frameworks, 28
- fresh objects, 254–255
- function keyword, 360, 370
- function overloads, 369–372
- functions. *See also* asynchronous functions
 - arrow type, 165–166
 - assignability, 265–267
 - deriving types from, 212–214
 - excess property checks when comparing, 252–253
 - excess property warnings in, 272–274
 - extracting, 21
 - generic, 360–361, 363
 - combining with generic types, 379–381
 - constraining type parameters, 365–366

- debugging inferred type
 - of, 364
- inference in, 375–377
- making, 372–374
- problems solved by, 363–364
- type aliases vs. generic types, 361–363
- type arguments in, 382–384
- type parameter defaults, 364–365

- parameters
 - annotations, 34, 36–37
 - basic types with, 38–39
 - comparing, 276–278
 - default, 61–62
 - implementation, 265–267
 - optional, 61
 - rest, 62–63, 67–68
 - restricting, 81–82
- passing
 - read-only arrays to, 144–145
 - read-only objects to, 145–146
 - types to, 56–60
 - returning void, 71–72
 - return types, 62
 - this keyword in, 263–264
 - types, 63–64, 68–70
 - unions of
 - intersecting parameters, 267–269
 - with object parameters, 278–281
- void type, 64

G

- generic classes, 363
- generic functions, 360–361, 363
 - combining with generic types, 379–381
 - constraining type parameters, 365–366
 - debugging inferred type of, 364
 - inference in, 375–377
 - making, 372–374
 - problems solved by, 363–364
 - type aliases vs. generic types, 361–363

- type arguments in, 382–384
 - type parameter defaults, 364–365
- “generic” term, 362–363
- generic types, 332–334, 363
 - arguments, 336
 - combining with generic functions, 379–381
 - creating, 344–345
 - default type parameters, 336–337
 - vs. generic function type aliases, 361–363
 - multiple type parameters, 334–335
 - passing arguments to, 361
 - type parameter constraints, 337–338
- get keyword, 176
- getters, 175–176
- GitHub repository, xxvii
- global identifiers with multiple *tconfig.json* files, 322–323
- global scope
 - adding types to, 301
 - declaration files, 290, 299
 - scripts, 286
 - using `declare` keyword, 292–293, 301
- global types, 301–306
- global variables, 291–292
- Go to Definition, 17–18
- Go to References, 17–18

H

- Hejlsberg, Anders, xxiv
- hot module replacement (HMR), 28
- hovering, 14–17
- HTMLAudioElement API, 10
- Hungarian notation, 170

I

- IDE (integrated development environment)
 - autocomplete, 9–11
 - automatic imports, 18
 - Go to Definition and Go to References, 17–18
 - introspecting variables and declarations, 14–16
 - JSDoc comments, 16–17

- navigation in, 17–18
 - quick fixes, 19
 - rename symbol, 18
 - restarting VS Code server, 19
 - tconfig.json* files, 322
 - for TypeScript, 5
 - TypeScript error checking, 11–14
 - working in JavaScript, 19–21
- implementations, 290–291
- implementation signature, 370–371
- implements keyword, 169–171
- imports, 18
- import statement, 285–286
- indexed access types, 201–205
 - accessing specific values, 207–208
 - chaining multiple, 202
 - unions with, 208–209
- indexing, 121
- index signatures, 121–122, 124
- inference
 - with array of objects, 146–148, 153–156
 - debugging inferred type of generic functions, 364
 - in generic functions, 375–377
 - of literal values in arrays, 153–156
 - and mutability
 - array mutation, 148–149
 - array of objects, 146–148
 - const keyword, 140–141
 - let keyword, 140
 - of object properties, 141–142
 - readonly object properties, 143–146
 - readonly tuples, 149–150
 - of types, 35–37
- inheritance
 - abstract classes, 171–172
 - abstract methods, 172
 - extending classes, 167–168
 - implements keyword, 169–171
 - overriding, 168–169
 - protected properties, 168
- initializers of class properties, 162–163
- in operator, 88–90, 96
- instanceof operator, 93–94
- integrated development environment. *See* IDE

- interface extends syntax, 115–116
- interfaces
 - in declaration files, 289
 - expecting certain properties, 135–136
 - extending, 114–117, 119–120
 - implements keyword, 169–171
 - merging within namespaces, 189–191
 - vs. types, 116–117
- intersection types, 112–113
 - creating, 117–118
 - vs. interface extends syntax, 115–116
- introspecting variables and declarations, 14–16
- I prefix, 170
- is keyword, 367–369
- isolatedModules setting, 308, 311–312

J

- JavaScript
 - choosing version with lib setting, 296
 - declaration files describing, 288–290
 - files, 316
 - integrated development environment, 19–21
 - leading to TypeScript, xxiii–xxiv
 - Plain Old JavaScript Object, 203
 - template literals in, 338
 - vs. TypeScript, 3–4
 - TypeScript changing, 26–27
 - TypeScript declaration files for, 295–296
 - typing module, 302
 - workflow, 4
 - working in, 19–21
- JavaScript-style enums, 203–205
- JSDoc comments, 16–17
- JSON.parse function, 59–60, 360
- jsx option, 319–320
- JSX syntax, 319–320

K

- keyof operator, 198–199
 - deriving types from values, 206–207
 - extracting union of all values, 209–210

- getting an object's values
 - with, 203
- iterating over objects, 275
- reducing key repetition, 205–206
- and typeof, 200
- keys
 - dynamic
 - combining with known keys, 122–123
 - default properties with, 125–126
 - index signatures, 121–122, 124
 - PropertyKey type, 123–124, 128
 - Record type, 122
 - support for, 127–128
 - iterating over, 255–256, 274–276
 - omitting, 349–351
 - reducing repetition of, 205–206
 - remapping with as, 343–344, 356–357
 - renaming in mapped types, 355–357

L

- language server, 4
- large numbers, 41
- let keyword, 140
- lib.dom.d.ts* file, 296–297
- lib.es5.d.ts* file, 296
- lib.es2021.string.d.ts* file, 296
- libraries
 - building, 308
 - development, 359
 - transpiling code for library use, 316–319
 - types shipping with, 297–298
- lib setting, 296, 297
- linters, 28
- literal types, 79–82
 - inferring values in arrays, 153–156
 - objects, 43–44
 - templates
 - in JavaScript, 338
 - in TypeScript, 338–340, 352–353

- local scope, 286
- Long Term Support (LTS), 6
- Lowercase type, 339

M

- Map feature, 58–59
- `.map` method, 267, 295
- mapped types, 341–343
 - key remapping with `as`, 343–344
 - renaming keys in, 355–357
 - and union types, 344
- members of unions, 78
- merging namespaces, 189–191
- methods, 165–166
 - abstract keyword, 171–172
 - implementing, 174–175
- Microsoft, xxiii–xxiv
 - NodeNext, 308, 314–315
 - Preserve, 309, 315–316
 - settings, 314–316
- `moduleDetection` configuration, 288, 308
- modules
 - augmentation vs. overriding, 294–295
 - and scripts, 285–288
 - sharing types across, 46–48
 - typing JavaScript types, 302
- module setting, 308, 320
- monorepos, 318–319, 325
- multiline errors, 13–14
- mutability
 - with `as const`, 150–153
 - inferring literal values in arrays, 153–156
 - and inference
 - array mutation, 148–149
 - array of objects, 146–148
 - `const` keyword, 140–141
 - `let` keyword, 140
 - of object properties, 141–142
 - readonly object properties, 143–146
 - readonly tuples, 149–150

N

- named tuples, 50–55
- name global variable, 287

- namespaces, 188–191
- naming types and values the same, 261–263

narrowing

- discriminated unions with `switch` statement, 102–103
- never type, 92–93
- process, 84–90
- unions wider than members, 83–84
- unknown type, 95–97
- values with `satisfies`, 228–229
- wide and narrow types, 82–83

- never type, 92–93, 113, 257, 280
- new keyword, 56, 160
- Next.js, 316
- NODE16 version, 315
- node `index.ts` command, 320
- Node.js
 - installing, 6
 - process, 305–306
 - runtime environment, 4, 5
 - support for TypeScript, 320–321
 - type checking, 28
- NodeJS namespace, 305–306
- `node_modules` folder, 6, 297, 299
- `node_modules/typescript/lib`, 296
- NodeNext setting, 308, 314–315
- `noEmit` option, 28, 309, 316, 320
- `noFallthroughCasesInSwitch` option, 314
- `noImplicitAny` setting, 312
- `noImplicitOverride` option, 169, 314
- `noImplicitReturns` option, 314
- `NonNullable` utility type, 219–220
- non-null assertion, 231–232
- non-runtime errors, 12
- `noPropertyAccessFromIndexSignature` option, 314
- `noUncheckedIndexedAccess` option, 308, 312–314
- `noUnusedLocals` option, 314
- `noUnusedParameters` option, 314
- npm package manager, 6, 297
- null type
 - empty object type, 257–258, 269–270
 - handling, 80–81
 - runtime errors, 369

- Number.MAX_SAFE_INTEGER value, 41
- numbers, large, 41
- number type, 34
- numeric enums, 182–183
 - behavior of, 185–187
 - transpiling, 184–185

O

- Object.freeze method, 151–153
- Object.keys() method, 275
- object literal types, 43–44
- objects
 - arrays of, 49
 - closed vs. open types, 253–254
 - dynamic object keys
 - combining with known keys, 122–123
 - index signatures, 121–122
 - PropertyKey type, 123–124, 128
 - Record type, 122
 - excess properties
 - detecting, 270–272
 - warnings, 251–255
 - extending
 - interfaces, 114–117
 - intersection types, 112–113
 - fresh and stale, 254–255
 - getting values with keyof operator, 203
 - inference of properties, 141–142
 - inference with array of, 146–148, 153–156
 - iterating over keys of, 255–256, 274–276
 - optional properties, 43–45
 - restricting keys with Record keyword, 126–127
 - utility types
 - Omit, 131–132
 - Partial, 129
 - Pick, 131
 - Required, 129–131
 - union types, 132–134
- Omit type, 131–132
 - action of, 220
 - DistributiveOmit type, 134
 - strictness of, 349
 - StrictOmit type, 349–351
 - union types with, 132–134
- online resources, xxvi
- open object types, 253–254
- operators
 - ?, 43, 45, 75, 232
 - !, 231, 240
 - &, 112, 118
 - |, 78–79
 - =, 61, 67, 336, 347
 - in, 88–90, 96
 - instanceof, 93–94
 - keyof, 198–199
 - deriving types from values, 206–207
 - extracting union of all values, 209–210
 - getting an object’s values with, 203
 - iterating over objects, 275
 - reducing key repetition, 205–206
 - and typeof, 200
- satisfies
 - annotating values with, 227–228
 - vs. as and variable annotations, 242–245
 - creating deeply read-only objects, 246–247
 - enforcing valid configuration, 240–242
 - excess property checking, 272, 274
 - narrowing values with, 228–229
- ternary, 340
- typeof
 - extracting types from values, 199–200, 206–207
 - extracting union of all values, 209–210
 - for narrowing, 84, 86–87
- optional class properties, 163
- optional modifier (?), 55, 61
- optional parameters, 61
 - function parameters, 65–66
- opts argument, 161

- outDir option, 317
- overload signature, 370
- override keyword, 168–169
- overriding modules, 294–295

P

- package.json*, 6, 315
- parameters. *See also* type parameters
 - annotations on, 34, 36–37
 - comparing, 276–278
 - default, 61–62, 66–67
 - in function callbacks, 265–267
 - in generic types, 334–338
 - optional, 61
 - rest, 62–63
 - type, 333
 - in unions of functions, 267–269, 278–281
- Parameters utility type, 212, 214–216
- @param tag, 16
- Parcel, 314–315
- Partial type, 129
- Pick type, 131
 - action of, 220
 - union types with, 132–134
- Plain Old JavaScript Object (POJO), 203
- PNG files, 300
- pnpm package manager, 6–7, 298
- polyfills, 310
- Preserve module, 309, 315–316, 320
- private class properties, 163–165
- private keyword, 163–165
- process entity, 305–306
- project flag, 325
- project references, 325–326
- projects, 25
- projects* folder, 7
- Promise type, 65, 217–218, 346
- properties
 - in classes, 162–165
 - excess property warnings
 - comparing functions, 252–253
 - fresh and stale objects, 254–255
 - in objects, 270–272
 - open vs. closed objects, 253–254
 - on variables, 251–252

- expecting certain, 135–136
- of objects
 - inferring, 141–142
 - readonly, 143–146
- optional, 43–45
- protected, 168
- PropertyKey type, 123–124, 128
- property types, 44–45
- protected keyword, 168
- public class properties, 163–165
- public keyword, 163–165

Q

- Quick Fixes menu, 19, 20

R

- react-jsx value, 320
- React TypeScript Cheatsheet, 312
- react value, 320
- ReadonlyArray type helper, 144
- readonly keyword
 - class properties, 163
 - object properties, 143–146
 - tuples, 149–150
- Record type, 122, 126–127
- @redux/toolkit library, 312
- Remix, 316
- rename symbol feature, 18
- .replaceAll method, 296
- Required type, 129–131
- resolveJsonModule setting, 308
- response.json() function, 73–74
- Response type, 73–74
- rest parameters, 62–63, 67–68
- Result types, 346–349
- return types, 62
- ReturnType utility type, 213, 216–217
- return value–based typing, 216–217
- rewriteRelativeImportExtensions setting, 320
- Rollup, 315
- root directory, 317
- rootDir option, 317
- Rosenwasser, Daniel, 192
- route matching, 351–352
- runtime errors, 11–12, 369
- runtime tests, xxvii

S

- satisfies operator
 - annotating values with, 227–228
 - vs. `as` and variable annotations, 242–245
 - creating deeply read-only objects, 246–247
 - enforcing valid configuration, 240–242
 - excess property checking, 272, 274
 - narrowing values with, 228–229
- Script# (ScriptSharp), xxiii–xxiv
- scripts, 285–288
- Set data structure, 56
- set keyword, 177
- sets, 56–57
- setters, 177
- setup for TypeScript development, 3–7
- shared utilities
 - assertion functions, 368–369
 - function overloads, 369–372
 - generic functions, 360–361
 - problem solved by generic functions, 363–364
 - generic function type alias vs. generic types
 - constraining type parameters, 365–366
 - debugging inferred type of generic functions, 364
 - “generic” term, 362–363
 - missing or conflicting type arguments, 361–362
 - type parameter defaults, 364–365
 - type predicates, 366–368
- single-file transpilers, 311
- single source of truth, 214–216
- `skipLibCheck` option, 299, 308
- Sorhus, Sindre, 131
- source maps, 316
- square brackets (`[]`), 48
- `src` directory, 6
- StackOverflow, 312
- stale objects, 254–255
- strictness options, 312–314
- `strictNullChecks` setting, 312

- `StrictOmit` type, 349–351
- `strict` option, 308, 312
- string enums, 183–187
- string types, 34, 80–81, 339–340
- subtypes, 83
- `super()` method, 167
- supertypes, 83
- SvelteKit, 316
- `swc` tool, 311
- switch statement, 102–103
- symbol type, 34

T

- `target` option, 308, 310
- TC39 committee, 192
- template literal types
 - combining with union types, 339
 - in JavaScript, 338
 - in TypeScript, 338–340
 - permutations, 352–353
- ternary operator, 340
- `TError` parameter, 347–348
- testing, automated, 5
- tests, runtime, xxvii
- `this` keyword
 - accessing classes, 161
 - and arrow functions, 265
 - in functions, 263–264
 - in types and values, 260–261
- `throw` keyword, 87–88
- tools for TypeScript development, 5–7
- `@totaltypescript/helpers` library, 38
- T prefix, 170, 363
- transforming derived types, 218–220
- transpiling TypeScript, 25–26
 - enums, 184–185
 - for library use, 316–319
 - module settings, 308
 - namespaces, 189
 - `noEmit` option, 316
- `trpc` library, 312
- `try...catch` statement, 93–94
- `tsconfig.json` files. *See also* declaration files
 - base options, 310–312
 - generating, 25
 - `jsx` option, 319–320
 - `lib` setting, 296

- moduleDetection, 288
- module options, 314–316
- multiple configurations
 - extending configurations, 323–325
 - finding *tsconfig.json* files, 321–323
 - project, 325
 - project references, 325–326
- Node's TypeScript support, 320–321
- noEmit, 28, 316
- noImplicitOverride, 169
- recommended configuration, 307–309
- source maps, 316
- strictness options, 312–314
- transpiling code for library use, 316–319
- tsc tool, 4, 25–26, 311, 326
- @ts-expect-error directive, xxvii, 232–233
- .ts files, 4, 46–47, 285
- @ts-ignore directive, 164, 233
- @ts-nocheck directive, 233–234
- .tsx files, 4
- tuples, 50
 - discriminated, 103–105
 - extracting types from, 202
 - in functions, 54–55
 - named, 50–55
 - and readonly, 149–150
- type aliases, 46, 361–363
- type annotations
 - any type, 37–38
 - basic types, 34, 40–41
 - excess property checking, 271
 - function parameter annotations, 34
 - type inference, 35–37
 - variable annotations, 35–36
- type arguments
 - in generic functions, 363, 382–384
 - missing or conflicting, 361–362
 - passing
 - to functions, 57
 - to generic functions, 362
 - to generic types, 333
 - setting defaults, 374–375
- type-fest library (Sorhus), 131
- type inference, 35–37
- type keyword, 46, 47–48
- type="module" attribute, 287
- typeof operator
 - extracting types from values, 199–200, 206–207
 - extracting union of all values, 209–210
 - for narrowing, 84, 86–87
- type-only namespaces, 191
- type parameters
 - constraining
 - in functions, 377–379
 - generic functions, 365–366
 - generic types, 337–338
 - creating generic types, 333, 346
 - defaults for generic types, 336–337
 - in generic functions, 363
 - generic types accepting multiple, 334–335
 - setting defaults, 364–365
- type predicates, 366–368
- types
 - arguments, 333
 - assignability of, 90, 93
 - based on return value, 216–217
 - basic, 34, 38–39, 40–41
 - conditional, 340–341
 - creating values from, 200–201
 - in declaration files, 289
 - deriving
 - creating values from types, 200–201
 - vs. decoupling, 221–223
 - from functions, 212–214
 - indexed access types, 201–205
 - from other types, 198–200
 - transforming derived types, 218–220
 - from values, 206–207
 - DistributiveOmit, 134
 - DistributivePick, 134
 - functions, 63–64
 - generic, 332–334
 - arguments, 336
 - default type parameters, 336–337

- types (*continued*)
 - generic (*continued*)
 - multiple type parameters, 334–335
 - type parameter constraints, 337–338
 - indexed access, 201–205
 - vs. interfaces, 116–117
 - intersection, 112–118
 - from libraries, 297–298
 - literal, 79–82
 - mapped, 341–343
 - key remapping with `as`, 343–344
 - renaming keys in, 355–357
 - and union types, 344
 - naming the same as values, 261–263
 - narrow, 82–83
 - never, 92–93, 113, 257, 280
 - open vs. closed objects, 253–254
 - passing to functions, 56, 57–60
 - passing to sets, 56–57
 - sharing across modules, 46–48
 - storing in declaration files, 301
 - template literals in TypeScript, 338–340, 352–353
 - throw narrowing, 87–88
 - union, 77–78
 - unknown, 90–91
 - using classes as, 161–162
 - and value world, 258–261
 - wide and narrow, 82–83
 - TypeScript
 - birth of, xxiii–xxiv
 - choosing modules and scripts, 287–288
 - configuring
 - base options, 310–312
 - jsx option, 319–320
 - module options, 314–316
 - multiple configurations, 321–326
 - Node support for, 320–321
 - `noEmit`, 316
 - recommended configuration, 307–309
 - source maps, 316
 - strictness options, 312–314
 - transpiling code for library use, 316–319
 - declaration files, 295–296
 - in development pipeline
 - browser support, 24–25
 - changing JavaScript, 26–27
 - errors in TypeScript CLI, 27–28
 - frameworks, 28
 - linter functionality, 28
 - transpiling, 25–26
 - version control, 27
 - watch mode, 27
 - vs. ECMAScript, 191–192
 - error checking, 11–14
 - future of, 192
 - installing, 7
 - vs. JavaScript, 3–4
 - performance, 116
 - project workflow, 5
 - setup, 3–7
 - tools for development, 5–7
 - workflow, 4–5
 - TypeScript command line interface (CLI), 5, 25, 27–28
 - TypeScript Official Handbook, xxvii
 - `typescript` package, 7
 - `@types/node` package, 305
 - `@types/*` package, 298
- ## U
- Uncapitalize type, 340
 - undefined type
 - and empty object type, 257–258, 269–270
 - vs. `void`, 72–73
 - unions
 - combining with other unions, 79–82
 - creating from `as const` array, 210–212
 - discriminated, 98–100
 - bag of optionals problem, 97–98
 - destructuring, 100–102
 - handling defaults with, 105–107

- narrowing with `switch` statement, 102–103
- extracting all values from object, 209–210
- vs. function overloads, 372
- of functions
 - intersecting parameters, 267–269
 - with object parameters, 278–281
- with indexed access types, 208–209
- members of, 78
- passing to indexed access types, 202–203
- types of, 77–78
- wider than members, 83–84
- union types, 77–78
 - combining template literal types with, 339
 - with `Omit` and `Pick`, 132–134
 - using mapped types with, 344
- unknown type, 90–91
 - narrowing to single value, 95–97
 - type assignability chart for, 257
 - type parameter defaults, 365
- Uppercase type, 339
- utility types
 - expecting certain properties, 135–136
 - `Omit`, 131–132
 - `Partial`, 129
 - `Pick`, 131
 - `Required`, 129–131
 - union types, 132–134
 - updating products, 136–137
- `/utils` folder, 359

V

- values
 - accessing specific, 207–208
 - annotating, 225–229

- creating from types, 200–201
- deriving types from, 206–207
- forcing type of, 229–232
- naming the same as types, 261–263
- narrowing with `satisfies`, 228–229
 - and type world, 258–261
- variables
 - annotating, 35–36
 - vs. annotating values, 225–229
 - with `as` and `satisfies`, 242–245
 - vs. `as const`, 151
 - environment, 305
 - excess property checks on, 251–252
 - global, 291–292
 - inlining, 20
 - introspecting, 14–16
- `verbatimModuleSyntax` function, 321
- version control, 27
- Visual Studio Code (VS Code), 5
 - server, 19
- Vite, 28, 316
- Vitest, xxvii–xxviii
- void type, 64, 71–73

W

- warning locations, 12–13
- `--watch` flag, 27
- watch mode, 27
- Webpack, 300, 314–315
- wide types, 82–83
- window object, 304–305

X

- `xstate` library, 312

Z

- `zod` library, 97, 297–298, 312