

# INDEX

## Symbols

- + (addition operator), 39, 156–157, 560
- & (ampersand), 17, 560
- <> (angle brackets), 562–563
  - for declaring lifetime parameters, 211
  - for specifying type parameters, 148, 192
- > (arrow), 48, 560
- \* (asterisk), 560
  - dereference operator, 74, 330–335, 478
  - glob operator, 142
  - multiplication operator, 39
- @ (at operator), 472, 561
- : (colon), 26, 204, 561, 563
  - for struct fields, 86
  - for trait bounds, 203
- { } (curly brackets), 565
  - for function bodies, 6, 15
  - as placeholders in the `println!` macro, 18
  - scope creation, 47, 77
- / (division operator), 39, 561
- . (dot), 560
  - for method syntax, 99–100
  - for struct field access, 88–89
  - for tuple element access, 41
- :: (double colon), 65, 106, 129, 562–563
- " (double quote), 40, 562
- (hyphen)
  - for negation, 560
  - for subtraction, 39, 560
- + (multiple trait bound syntax), 204, 560
- ! (never type), 501–502, 562
- () (parentheses), 564
  - for function parameters, 6, 15
  - for tuples, 40–51
- ? (question mark operator), 177–181, 561

- % (remainder operator), 39, 560
- ; (semicolon), 6, 561
  - in the array type, 42
  - to end statements, 47
- ' (single quote), 562
  - for characters, 40
  - for lifetime parameter names, 211
  - for loop labels, 56
- [] (square brackets), 560
  - for array creation, 41
  - in the array type, 42
  - for element access, 42, 149–151
- \_ (underscore), 562
  - as a catch-all pattern, 29, 117–118, 465–467
  - in filenames, 5
  - as a visual separator in integer literals, 37–38
- | (vertical pipe)
  - in closure definitions, 284, 562
  - in patterns, 460, 561

## A

- ABI (application binary interface), 483
- abort, 168
- absolute paths, 130
- addition
  - of custom types, 489–491
  - of number types, 39
  - of strings, 155–157
- addition operator (+), 39, 156–157, 560
- ahead-of-time compiled, 7
- ampersand (&), 17, 560
- ancestor modules, 131
- angle brackets (<>), 562–563
  - for declaring lifetime parameters, 211
  - for specifying type parameters, 148, 192

- API (application programming interface), 4, 308–311
- application binary interface (ABI), 483
- `Arc<T>` type, 378–381, 543–544
- arguments, 45
- arms
  - in `if` expressions, 51
  - in `match` expressions, 24, 113–114
- array data type, 41–44
  - accessing elements of, 42
  - invalid element access, 43
  - iterating over elements of, 57–59
  - slices of, 86
- arrow (`->`), 48, 560
- `as_bytes` method, 81
- `as` keyword, 139
- `assert_eq!` macro, 229–231
- `assert!` macro, 226–229
- `assert_ne!` macro, 231
- associated function, 16, 103
- associated types, 488–489
- associative array. *See* `HashMap<K, V>` type
- asterisk (`*`), 560
  - dereference operator, 74, 330–335, 478
  - `glob` operator, 142
  - multiplication operator, 39
- `async` block, 392–406
- `async` function, 392–394
- asynchronous programming, 385–425
- `async` keyword, 389, 393
- `async` runtime. *See* runtime
- atomically reference counted, 378–381
- `at` operator (`@`), 472, 561
- attribute-like procedural macros, 517
- automatic dereferencing, 101
- automatic referencing, 101
- `await` keyword, 389, 391, 393–394
- `await` point, 394, 405–409, 417

## B

- back of house, 127
- backtrace, 169–171
- backward-compatibility guarantees, xxi
- binary crate, 19, 125, 133, 248, 257
- binary literal syntax, 38
- binary target, 320
- blanket implementations, 207

- blocking, 365, 371, 376, 386–401, 409
- Boolean data type, 40
- borrow checker, 209–216
- borrowing, 73–80
- `Box<T>` type, 324–329
- `break` keyword, 28, 55
- buffer overread, 169
- byte literal syntax, 38, 81, 562

## C

- Cargo, 7
  - commands
    - `build`, 9–10, 11
    - `check`, 10–11
    - `clippy`, 573
    - `doc`, 23, 305–307
    - `fix`, 572–573
    - `fmt`, 571–572
    - `install`, 320–321
    - `login`, 312
    - `new`, 8–9, 14, 125, 128
    - `publish`, 312–314
    - `run`, 10, 317
    - `test`, 223–224, 237–243, 306–307, 319–320
    - `update`, 21
    - `yank`, 314–315
  - extending with custom commands, 321
  - workspaces, 315–320
- Cargo.lock*, 10, 21, 318
- Cargo.toml*, 8–9, 19–22, 125
  - dependencies section, 9, 19–22
  - package section, 9, 312–313
  - profile section, 304
  - updating crate versions in, 21–22
- carriage return, 525
- `cfg` (configuration) attribute, 243–244
- channels, 369–375, 400–405, 541–547, 550–553
- character data type, 40
- `checked_*` methods, 38
- child modules, 129, 131
- client, 520
- Clippy, 573
- `clone` method
  - deep copy creation, 69
  - trade-offs of, 260

- Clone trait, 569–570
- closed channel, 369
- closures, 282–292
  - capturing the environment with, 282–284, 286–288, 295–297
  - moving ownership into, 287–288
  - moving ownership out of, 288–292
  - returning, 506
  - running in threads, 366–369
  - type inference in, 284–286
- cmd.exe*, 3, 5, 7
- cmp method, 23–24
- coherence, 201
- collections, 147–165
- colon (:), 26, 204, 561, 563
  - for struct fields, 86
  - for trait bounds, 203
- command line arguments, accepting, 252–255
- command line notation, 2
- comments, 49, 305, 537
- compiler-driven development, 533
- compile-time evaluation, 34
- compiling
  - with cargo, 9–10
  - in release mode, 11
  - with rustc, 5–7
- compound data types, 40–44
- compute-bound operation, 386, 409, 424
- concurrency, 361, 386–389, 397–411, 424–425
- concurrent programming, 361–362
- configuration (cfg) attribute, 243–244
- connection, 521–522
- cons list, 325–329
- constant evaluation, 34
- constants, 34
  - vs. static variables, 484
  - vs. variables, 34
- constructor, 337
- \*const T, 477–479, 560
- consuming adapters, 284
- consuming an iterator, 292–295
- continue keyword, 29, 55
- control flow, 50–59
- conventions
  - Cargo, 11
  - for crate root files, 125
  - naming
    - of constants, 34
    - of files, 5
    - of function and variable names, 44
    - of static variables, 484
    - of type parameters, 191
  - for use paths, 138–139
- cooperative multitasking, 409
- Copy trait, 70, 570
- CPU-bound operation, 386, 409, 424
- crate, 9, 124–125
  - binary, 125–126, 133–134
  - binary vs. library, 19
  - library, 125–126, 133
  - license of, 312
  - metadata, 312–313
  - publishing, 305–314
  - root file of, 125–126
  - root module of, 128
  - updating versions, 21–22
  - using as a dependency, 19–22, 140–141
  - yanking, 314
- crate keyword, 128
- crate root, 125–126, 128, 143
- crates.io*, 305
  - publishing to, 305–314
  - removing from, 314
  - setting up an account on, 311
- CRLF sequence, 525
- CTRL-C, 27, 54
- curly brackets ({}), 565
  - for function bodies, 6, 15
  - as placeholders in the `println!` macro, 18
  - scope creation, 47, 77
- custom derive procedural macros, 511–516

## D

- dangling pointer, 78
- dangling reference, 78–79, 208–209, 214–215
- data race, 76, 484–485

- data types, 36–43
  - annotation of, 26, 36
  - compound, 40–43
  - floating point, 39
  - integer, 37–38
  - scalar, 37–40
- dbg! macro, 98
- deadlock, 381
- Debug trait, 97–99, 231, 568
- decimal literal syntax, 38
- declarative macros, 509–511
- deep copy, 569
- Default trait, 570
- default type parameters, 489–491
- dependencies section in *Cargo.toml*, 9, 19–22
- dependency, 8, 11, 19–22
- deref coercion, 156, 333–335
- dereference operator, 74, 330–335
- DerefMut trait, 335
- Deref trait, 330–335, 498
- derive annotation, 96–99, 511–516, 567–570
- description metadata, 312
- destructor, 337
- destructuring
  - of enums, 463–464
  - in patterns, 461–465
  - of structs, 461–462, 464–465
  - of tuples, 41, 465
- Dickinson, Emily, 255
- dictionary. *See* `HashMap<K, V>` type
- Dijkstra, Edsger W., 221
- Display trait, 96, 154, 206–207, 495–497, 568
- division operator (`/`), 39, 561
- doc tests, 306–307
- documentation
  - comments, 305, 537
  - offline for Rust, 4
  - tests, 306
  - viewing a crate's, 23
  - writing, 305
- dot (`.`), 560
  - for method syntax, 99–100
  - for struct field access, 88–89
  - for tuple element access, 41
- double colon (`::`), 65, 106, 129, 562–563
- double free error, 68, 338
- double quote (`"`), 40, 562
- Doyle, Sir Arthur Conan, 301
- drop function, 66–70, 337–338
- Drop trait, 335–338, 548–553
- dynamically sized type (DST), 502–504
- dynamic dispatch, 436
- dyn compatibility, 436
- dyn keyword, 265, 432

**E**

- editions of Rust, xxi, 9, 558, 573, 575–576
- else if expression, 52
- else keyword, 51
- empty type, 501–502, 562
- encapsulation, 123, 127, 428–430
- entry method, 163–164
- Entry type, 163–164
- enumerate method, 81, 456
- enums, 105–112
  - defining, 106
  - destructuring, 463–464
  - initializer function, 505–506
  - instantiating, 106–108
  - making public, 135–136
  - variants of, 106
- environment, 282–284, 286–288, 295–297
- environment variables, 273–278
- eprintln! macro, 279
- Eq trait, 568–569
- error handling, 167–186
- executable file, 6–7, 10
- executing code, 6–7, 10
- executors, 395, 423
- exit status code, 263
- expect method, 17–18, 26, 175
- expressions, 46–47
- extern functions, 482–483

**F**

- fairness, 400, 410
- fearless concurrency, 362
- FFI (Foreign Function Interface), 482
- field init shorthand, 89–90
- fields, 88
- files, 255–256
  - naming conventions, 5
  - organization, 143–144

- filtered-out tests, 241–242
- Firefox web browser, xxvi
- floating-point data types, 39
- fn keyword, 15
- FnMut trait, 288–291, 505, 536
- FnOnce trait, 288–291, 505, 536
- Fn trait, 288, 505, 536
- fn type, 504–506
- Foreign Function Interface (FFI), 482
- for keyword
  - loop, 57–58
  - patterns in, 456
  - in trait implementations, 200
- format! macro, 157
- from function
  - on the From trait, 177
  - on String, 64, 154
- front of house, 127
- fully qualified syntax, 492–495, 505
- functional programming, 281
- function-like procedural macros, 517
- function pointers, 504–506
- functions, 44–49
  - arguments to, 45
  - bodies, statements, and expressions in, 46–47
  - extern, 482–483
  - vs. macros, 508–509
  - making public, 132–133
  - with multiple return values using a tuple, 73
  - parameters of, 45–46
    - patterns in, 456
  - returning early from, 48
  - with return values, 48–49
- futures, 389, 392
  - joining, 398–400, 404–405
  - laziness of, 391
  - vs. threads, 423–425
- futures crate, 390, 396, 421
- Future trait, 389, 392, 413–415, 416–417

## G

- Gallant, Andrew, 252
- Gamma, Erich, 428
- garbage collector (GC), 61, 65

- generics, 187–198, 219–220
  - default types for, 489–491
  - in enum definitions, 195
  - in function definitions, 191–193
  - in method definitions, 195–197
  - performance of, 197–198
  - in struct definitions, 193–194
- get method
  - on HashMap<K, V>, 161
  - on Vec<T>, 149–151
- getter methods, 101, 185
- Git, 8, 11
- GitHub, 311
- global variables, 484–485
- grapheme clusters, 158–160
- grep, 251
- guard, 376
- guessing game, 13–30

## H

- hash. *See* HashMap<K, V> type
- hasher, 165
- hashing function, 160, 165
- HashMap<K, V> type, 160–165
  - entry method on, 163–164
  - get method on, 161–162
  - insert method on, 161–164
  - iterating over, 162–164
  - new function on, 161–162
    - and ownership, 162
  - updating, 163–165
- Hash trait, 570
- heap
  - allocating on, 62, 325
  - and the stack, 62–63
- “Hello, world!” program, 4–7
- Helm, Richard, 428
- hexadecimal literal syntax, 38
- Hoare, Tony, 110
- HTTP (Hypertext Transfer Protocol), 520, 524–526
- hyphen (-)
  - for negation, 560
  - for subtraction, 39, 560

## I

- IDE (integrated development environment), xxvi, 4, 574

- if keyword, 50–54
- if let syntax, 118–121
  - patterns in, 454–455
- ignore attribute, 241–242
- immutability. *See* mutability
- impl keyword
  - for defining associated functions, 103
  - for defining methods, 99–103
  - for implementing traits, 200
- impl Trait syntax, 203–205, 392
- indexing syntax, 149–151
- indirection, 328–329
- inheritance, 430–431
- input lifetimes, 217
- input/output (io) library, 15
- input/output-bound operation, 386
- installation of Rust, 1–4
  - Linux, 2
  - macOS, 2
  - Windows, 3
- integer data types, 37–38
  - numeric operations with, 39
  - type suffixes of, 37
- integer overflow, 38
- integrated development environment (IDE), xxvi, 4, 574
- integration tests, 243–248
- interfaces. *See* traits
- interior mutability, 342–349, 351, 381
- invalidated variable, 68–69
- io (input/output) library, 15
- I/O-bound operation, 386, 424
- IpAddr type, 107–108
- irrefutable patterns, 457–459
- isize type
  - architecture-dependent size of, 37
  - indexing collection with, 38
- iterator adapters, 294–296
- iterators, 292–302, 391, 411–412
  - adapters for, 294–296
  - consuming adapters for, 294
  - creating with iter method, 81
  - enumerate method on, 81
  - next method on, 293
  - performance of, 301–302
- iter method, 81

**J**

- Johnson, Ralph, 428
- JoinHandle<T> type, 364–366
- joining futures, 398–400, 404–405

**K**

- Kay, Alan, 427
- Keep, Daniel, 511
- keywords, 32, 555–558

**L**

- Language Server Protocol, 574
- last in, first out (LIFO) ordering, 62
- lazy evaluation, 292, 295, 391
- len method, 81
- let...else syntax, 119–121
- let keyword, 15
  - using patterns with, 453–454
- library crate, 8, 19, 125, 133
- license identifier value, 312
- license metadata, 312
- lifetimes, 207–220
  - annotation of, 210–216
  - elision, 216–218
- LIFO (last in, first out) ordering, 62
- line feed, 525
- linker, 2
- lints, 573
- Linux, installing Rust on, 2
- Linux Foundation, 312
- “The Little Book of Rust Macros,” 511
- lock, 375–378
- loop keyword, 27–28, 54–57
- loop labels, 56

**M**

- macOS, installing Rust on, 2
- macro\_export annotation, 510
- macro\_rules! macros, 509–511
- macros, 508–518
  - declarative, 509–511
  - vs. functions, 508–509
  - procedural, 511
- main function, 6, 180, 393–395
- mangling, 483
- map. *See* HashMap<K, V> type
- marker trait, 419

- match expression, 112–118
  - exhaustiveness of, 116–117
  - handling comparison results
    - with, 24
  - handling error values with, 173
  - handling Result values with, 29, 172
  - patterns in, 452–453
- match guard, 470–472
- memory leak, 351–352, 359
- message passing, 369–375, 400–402, 424
- methods
  - defined on enums, 109–110
  - defined on structs, 99–104
  - disambiguating, 492–495
  - getters, 101
- method syntax, 99–100
- minigrep project, 251–280
- Miri, 486–488
- mock object, 345–350
- mod keyword, 128
- modules, 125–128
  - cheat sheet, 125–127
  - filepaths for, 126
  - moving to other files, 143–144
- module system, 124
- module tree, 128–129
- monomorphization, 197–198
- move keyword, 287–288, 366, 403–404
- moving ownership, 66–69
  - into async blocks, 403–404
  - vs. borrowing, 73–75
  - with function calls, 71–72
  - with function return values, 72–73
- multiple producer, single consumer
  - (mpsc), 370, 374, 400–401
- multiple trait bound syntax (+), 204, 560
- multiplication, 39
- mutability
  - of references, 75–78
  - of variables, 32–33
- Mutex<T> type, 375–381, 543–544, 546–548
- mut keyword
  - making a reference mutable with, 75–78
  - making a variable mutable with, 32–33
  - vs. shadowing, 35–36

- \*mut T, 477–479, 560
- mutual exclusion, 375

## N

- namespace, 65, 103, 106
- never type (!), 501–502, 562
- new function
  - on HashMap<K, V>, 161–162
  - on String, 154–155
  - on Vec<T>, 148–149
- new project setup, using cargo, 14
- newtype pattern, 497–498
- null, 110–112
- numeric operations, 39

## O

- object. *See* HashMap<K, V> type
- object-oriented programming (OOP), 427–449
- octal literal syntax, 38
- 1:1 threading model, 363
- operator overloading, 489–491
- operators, 559–561
- optimizations, 11
- Option<T> enum, 110–112, 115–116
- Ordering type, 24
- Ord trait, 569
- orphan rule, 201, 497
- output lifetimes, 217
- overflowing\_\* methods, 38
- overflow of integers, 38
- ownership, 61–86. *See also* moving ownership
  - and async blocks, 403–404, 417
  - and functions, 71–73
  - rules, 63
  - of struct data, 92–93

## P

- package, 125
- package registry, 305–314
- package section in *Cargo.toml*, 9, 312–313
- panicking, 38
- panic! macro, 168–171, 233–236
  - vs. Result, 181–186
- parallelism, 387–389, 424–425
- parallel programming, 361–362

- parameters, 45
  - patterns in, 456
- parentheses (`()`), 564
  - for function parameters, 6, 15
  - for tuples, 40–41
- parent modules, 129, 131
- parse method, 26
- `PartialEq` trait, 231, 568–569
- `PartialOrd` trait, 193, 569
- paths, 129–134
  - absolute, 130
  - nested, 141
  - relative, 129–130, 134
- `%PATH%` system variable, 3, 321
- patterns, 451–473
  - binding to values with, 114–115
  - destructuring in, 461–465
  - ignoring values in, 465–469
  - in for loops, 456
  - in function parameters, 456–457
  - in if `let` syntax, 118–121, 454–455
  - in `let...else` syntax, 119–121
  - in `let` statements, 453–454
  - in match expressions, 112–118, 452–453
  - refutable vs. irrefutable, 457–459
  - in while `let` loops, 455
- `.pdb` file extension, 7
- `pin!` macro, 420–421
- `Pin` type, 415–421
- pointer, 62–63, 323
  - dangling, 78
  - to data on the heap, 62
  - raw, 477–479
  - smart, 323–359
- poisoned mutex, 546
- polling, 389, 410, 413–415, 422
- `Poll<T>` enum, 413
- polymorphism, 430–431
- postfix keyword, 391
- PowerShell, 2–3, 7, 277
- prelude, 15, 142
- `println!` macro, 6, 18
- privacy, 127, 131–133
- procedural macros, 511
  - attribute-like, 517
  - custom `derive`, 511–513
  - function-like, 517
- process, 362
- `proc_macro` crate, 511, 514
- profiles, 304–305
- profile section in *Cargo.toml*, 304
- promises. *See* futures
- propagating errors, 175–181
- `pub` keyword, 126, 131–133
- public, 131–136
  - API, 133, 308–311
  - making items, 132
  - making structs and enums, 134
- `pub` use, 140, 308–311
- push method, 149, 155
- `push_str` method, 65, 155

**Q**

- question mark operator (`?`), 177, 561
- quote crate, 513–514, 516

**R**

- race conditions, 76, 363
- RAII (Resource Acquisition Is Initialization), 66
- `rand` crate, 19–22
- random number functionality, 19, 22–23
- range syntax, 82, 461, 549
- Range type, 58
- raw identifiers, 557–558
- raw pointers, 477–479
- `Rc<T>` type, 339–342, 350–359
- `read_line` method, 16–17
- receiver, 369–374
- recoverable errors, 167–168, 171–181
- recursive type, 325–329
- re-export, 140, 308–311
- `RefCell<T>` type, 342–359
- reference counting, 323, 339–342, 378–381
- reference cycles, 351–359
- references
  - for accessing data from multiple places, 17
  - and borrowing, 73–80
  - dangling, 78–79
  - dereferencing, 74
  - mutability of, 75–78
  - rules of, 80

- refutable patterns, 457–459
- registry, 20, 305–314
- relative paths, 129–130, 134
- release mode, 11, 38
- release profiles, 304–305
- remainder operator (%), 39, 560
- request line, 525–526
- request-response protocol, 520
- Resource Acquisition Is Initialization (RAII), 66
- Result<T, E> type, 17–18, 171–181
  - expect method on, 17–18, 175–186
  - vs. panic!, 181
  - type aliases for, 500–501
  - unwrap method on, 174
  - unwrap\_or\_else method on, 174
- return keyword, 48
- return values
  - of functions, 48–49
  - of loops, 55
  - multiple using a tuple, 73
- rev method, 58–59
- ripgrep, 252, 321
- RLS (Rust Language Server), xxvi
- .rs file extension, 5
- running code, 5–7, 10–11
- runtime, 393–395, 405–409, 414, 423–424
- Rustaceans, 3
- rust-analyzer, 574
- rustc, 3, 5–7
- rustfix, 572–573
- rustfmt, xxvi, 6, 571–572
- Rust Language Server (RLS), xxvi
- Rustonomicon*, 151, 359, 382
- rustup commands, 1
  - doc, 4
  - uninstall, 4
  - update, 4

**S**

- saturating\_\* methods, 38
- scalar data types, 37–40
- scope, 63, 124
- SCREAMING\_SNAKE\_CASE, 484
- Self keyword, 100
- self module, 129
- self parameter, 99
  - type annotations on, 416–417
- self-referential types, 417–421
- Semantic Versioning (SemVer), 19–20, 314
- semicolon (;), 6, 561
  - in the array type, 42
  - to end statements, 47
- Send trait, 382, 486, 536
- serial programming, 388
- server, building a, 520
- shadowing, 34–36
  - vs. mut keyword, 35–36
- shared-state concurrency, 375–381
- should\_panic attribute, 233–236
- sibling modules, 129
- single quote ('), 562
  - for characters, 40
  - for lifetime parameter names, 211
  - for loop labels, 56
- ?Sized, 504, 563
- Sized trait, 503–504, 507
- slice type, 80–86
  - of array, 86
  - string slices, 82–86, 159
- smart pointer, 323–359
- snake case, 44
- Software Package Data Exchange (SPDX), 312
- square brackets ([]), 560
  - for array creation, 41
  - in the array type, 42
  - for element access, 42, 149–151
- stack
  - and the heap, 62–63
  - last in, first out (LIFO) ordering, 62
  - popping off of, 62
  - pushing onto, 62
- standard error (stderr), 278–279
- standard output (stdout), 278–279
- starvation, 406
- state machine, 394–395, 409, 417
- statements, 46–47
- state objects, 437–438
- state pattern, 437–445
- statically typed, 36
- static dispatch, 436

- 'static lifetime, 219, 484, 536
- static variables, 484
- status line, 525
- stderr (standard error), 278–279
- stdin function, 16–17
- stdout (standard output), 278–279
- &str (string slice type), 82–84
- StreamExt trait, 412–413, 422–423
- streams, 411–413, 521
- Stream trait, 412, 421–423
- stringify! macro, 516
- string literal, 64
  - storage in the binary of, 65
  - of string slice type, 85
- string slice type (&str), 82–84
- String type, 64–66, 153–160
  - as\_bytes method on, 81
  - bytes method on, 160
  - chars method on, 159
  - concatenation with +, 156
  - from function on, 64, 154
  - indexing into, 157–159
  - internal structure of, 65–67, 157–159
  - iterating over, 159–160
  - len method on, 81
  - new function on, 154–155
  - parse method on, 26
  - push method on, 155
  - push\_str method on, 65, 155
  - slicing, 159
  - trim method on, 26
  - UTF-8 encoding of, 153–154, 158–160
- Stroustrup, Bjarne, 302
- structs
  - defining, 88
  - destructuring, 461–462, 464–465
  - field init shorthand, 89–90
  - fields, 88
  - instantiating, 88–89
  - making public, 134–136
  - ownership of data, 92–93
  - tuple, 91
  - unit-like, 92
  - update syntax, 90–91
- subtraction, 39
- super keyword, 129, 134
- supertraits, 495–497

- symbols, 562–565
- synchronous programming, 386
- syn crate, 514–515
- Sync trait, 382, 486

## T

- tasks, 397–402, 423–425. *See also*
  - futures
- TCP (Transmission Control Protocol), 520
- test attribute, 223
- test double, 345
- test-driven development (TDD), 268–272
- test functions, 222–226
- tests, 221–249
  - custom failure messages for, 231–233
  - documentation, 306–307
  - filtering, 240–242
  - ignoring, 241–243
  - integration, 243–248
  - organizing, 243–248
  - of private functions, 244
  - running, 237–243
  - unit, 243–244
  - using Result<T, E> in, 236
  - writing, 222–237
- thread pool, 532–553
- threads, 362–382
  - creating with spawn, 363–364, 391, 533–547
  - vs. futures, 423–425
  - joining, 364–366
  - pausing with sleep, 364
  - running closures in, 363–364, 366–369
- thunk, 500
- tokio crate, 390, 393
- Tom’s Obvious, Minimal Language (TOML), 9
- to\_string method, 154, 207
- ToString trait, 207
- trait bounds, 203, 207, 219–220
  - conditionally implementing methods with, 206–207
- trait objects, 415, 431–436, 507
  - dynamic dispatch, 436

- traits, 198–207
  - associated types in, 488–489
  - default implementations of, 201–203
  - defining, 199
  - derived, 96–99
  - implementing, 199–201
  - as parameters, 203–205
  - supertraits, 495–497
  - unsafe, 485–486
- Transmission Control Protocol (TCP), 520
- transmitter, 369–374
- trim method, 26
- trpl crate, 390
- tuple data type, 40–41
- tuples, destructuring, 41, 465
- tuple structs, 91, 496–497
- two’s complement wrapping, 38
- type alias, 499–501, 544
- type annotation, 26, 36
- type inference, 25
- type suffixes, 37

## U

- underscore (`_`), 562
  - as a catch-all pattern, 29, 117–118, 465–467
  - in filenames, 5
  - as a visual separator in integer literals, 37–38
- Unicode scalar value, 40, 158–160
- uniform resource identifier (URI), 525
- uniform resource locator (URL), 525
- unions, 486
- unit-like structs, 92
- unit tests, 243–244
- unit type, 41
- Unpin trait, 415–421
- unrecoverable errors, 168–171
- unsafe, 476–488
  - checking with Miri, 486–488
  - functions, 479–484
  - superpowers, 476–477, 488
  - traits, 485–486
- unsized type, 502–504
- unwinding, 168
- unwrap method, 174

- unwrap\_or\_else method, 263–264
- URI (uniform resource identifier), 525
- URL (uniform resource locator), 525
- use keyword, 136–142
  - and as, 139
  - and external packages, 140–141
  - and the glob operator, 142
  - and nested paths, 141–142
  - and pub, 140
- user input, 16–17
- usize type
  - architecture-dependent size of, 37
  - indexing collection with, 38
- UTF-8 encoding, 153–154, 158–160

## V

- variables, 32–36
  - vs. constants, 34
  - creating with patterns, 453–454
  - global, 484–485
  - mutability, 32–33
  - shadowing, 34–36
  - static, 484–485
  - storing values in, 15–16
- variants, 106
- vec! macro, 148
- Vec<T> type, 148–153
  - get method on, 149–150
  - indexing into, 149–150
  - iterating over, 151–152
  - new function on, 148–149
  - push method on, 149
  - storing multiple types in, 152–153
- vertical pipe (`|`)
  - in closure definitions, 284, 562
  - in patterns, 460, 561
- Visual Studio, 3
- Visual Studio Code, 574
- Vlissides, John, 428

## W

- warnings, 572–573
- Weak<T> type, 356–359
- web scraper project, 390–396
- web server project, 519–553
- where clause, 204–205
- while let loop, 402, 455
- while loop, 56–57

Windows, installing Rust on, 3  
Wirth, Lukas, 511  
workspaces, 315–320  
work stealing, 423  
wrapping\_\* methods, 38

## **Y**

yanking, 314–315  
yielding, 405–409

## **Z**

zero-cost abstractions, 302  
zero-overhead, 302