

INDEX

Symbols and Numbers

- + (addition operator), 39, 139–140, 500
 - & (ampersand), 17, 500
 - <> (angle brackets), 502–503
 - for specifying lifetime parameters, 197
 - for specifying type parameters, 132, 176, 177
 - > (arrow), 47–48, 500
 - * (asterisk), 500
 - dereference operator, 70, 317–321, 420
 - glob operator, 127
 - multiplication operator, 39
 - @ (at operator), 415–416, 501
 - : (colon), 500, 503
 - for struct fields, 84
 - for trait bounds, 187
 - { } (curly brackets), 504
 - for function bodies, 5, 15
 - as placeholders in the `println!` macro, 18
 - scope creation, 46, 73
 - / (division operator), 39, 500
 - . (dot), 500
 - for method syntax, 93
 - for struct field access, 84
 - for tuple element access, 41
 - :: (double colon), 502
 - for associated functions, 96
 - for enum variants, 98
 - for namespacing, 115
 - " (double quote), 39, 501
 - (hyphen)
 - for negation, 500
 - for subtraction, 39, 500
 - + (multiple trait bound syntax), 187, 500
 - ! (never type), 440–441, 502
 - () (parentheses), 504
 - for function parameters, 5, 15
 - for tuples, 40–41
 - ? (question mark operator), 162–164, 501
 - % (remainder operator), 39, 500
 - ; (semicolon), 6, 42, 501
 - ' (single quote), 501–502
 - for characters, 39
 - for lifetime parameter names, 196
 - [] (square brackets), 505
 - for array creation, 41
 - in the array type, 42
 - for element access, 42–43, 133–135
 - _ (underscore), 502
 - as a catchall pattern, 28, 108–109, 409–411
 - as a visual separator in integer literals, 37
 - | (vertical pipe), 501–502
 - in closure definitions, 267
 - in patterns, 404
 - 1:1 threading model, 349
 - 2015 edition. *See* editions
 - 2018 edition. *See* editions
- ## A
- ABI (application binary interface), 424
 - abort, 152
 - absolute path, 115–116, 119
 - addition
 - of custom types, 429–430
 - of number types, 39
 - of strings, 139–140
 - addition operator (+), 39, 139–140, 500
 - ahead-of-time compiled, 7
 - ampersand (&), 17, 500
 - ancestor modules, 117
 - angle brackets (<>), 502–503
 - for specifying lifetime parameters, 197
 - for specifying type parameters, 132, 176, 177
 - API (Application Programming Interface), xxviii, 4
 - application binary interface (ABI), 424
 - `Arc<T>` type, 367–368, 480–481

- arguments, 44
- arms
 - in if expressions, 50
 - in match expressions, 23–24, 105
- array data type, 41–43
 - invalid element access, 42–43
 - iterating over elements of, 56–57
 - slices of, 81
- arrow (->), 47–48, 500
- as_bytes method, 76
- as keyword, 124
- assert_eq! macro, 214–216
- assert! macro, 211–214
- assert_ne! macro, 216
- associated function, 16, 95–96
- associated types, 427–428
- associative array. *See* `HashMap<K, V>` type
- asterisk (*), 500
 - dereference operator, 70, 317–321, 420
 - glob operator, 127
 - multiplication operator, 39
- atomically reference counted, 367–368
- at operator (@), 415–416, 501
- attribute-like procedural macros, 454–455
- automatic dereferencing, 94
- automatic referencing, 94

B

- back of house, 114
- backtrace, 153–155
- binary crate, 19, 111–113, 239
- binary target, 308
- blanket implementations, 192
- blocking, 351
- Boolean data type, 39, 51
- borrow checker, 194–196, 197
- borrowing, 70–75
- `Box<T>` type, 312–317
- break keyword, 27, 54–55
- buffer overread, 153
- Build Tools for Visual Studio, 3
- byte literal syntax, 37, 76, 502

C

- Cargo, xxvi, 7–11
 - commands
 - build, 9, 10
 - check, 10
 - clippy, 513–514
 - doc, 22, 293–294, 296
 - fix, 512–513, 516
 - fmt, 511–512
 - install, 308–309
 - login, 300
 - new, 8, 14
 - publish, 300–302
 - run, 9–10, 305
 - test, 208–211, 221–227, 295, 306–308
 - update, 21
 - yank, 302
 - extending with custom
 - commands, 309
 - workspaces, 303–308
- Cargo.lock*, 9, 20–21
- Cargo.toml*, 19–21, 112–113
 - dependencies section, 8, 19
 - package section, 8, 300–301
 - profile section, 292–293
 - updating crate versions in, 21
- carriage return, 462
- cfg (configuration) attribute, 227–228
- channels, 355–361, 478–483
- character data type, 39–40
- child modules, 115, 117
- client, 458
- Clippy, 513–514
- clone method
 - deep copy creation, 67
 - trade-offs of, 242
- Clone trait, 509–510
- closed channel, 356
- closures, 264–276
 - capturing the environment with, 274–276, 280–281
 - returning, 445
 - running in threads, 350
 - type inference in, 269–270
- cmp method, 23–24
- coherence, 184
- collections, 131–149
- collect method, 145, 235
- colon (:), 500
 - for struct fields, 84
 - for trait bounds, 187
- command line arguments, accepting, 234–237
- command line notation, 2
- comments, 49, 293–296, 475
- compiler-driven development, 469

- compiling
 - with cargo, 7–11
 - in release mode, 10
 - with rustc, 6–7
- compound data types, 40–43
- concurrency, 347–370
- concurrent programming, 347
- configuration (cfg) attribute, 228
- connection, 458–462
- cons list, 314–317
- constants, 34
 - vs. static variables, 425–426
 - vs. variables, 34
- constructor, 325
- *const T, 419–421, 500
- consume, 278–279
- consuming adaptors, 278–279
- continue keyword, 28
- contracts, 166
- control flow, 49–57
- Copy trait, 67, 190–191, 509–510
- crate, 8, 112–113
 - binary vs. library, 19
 - license of, 300–301
 - publishing, 301–302
 - updating versions, 21
 - using as a dependency, 19–21, 125–126
 - yanking, 302
- crate keyword, 115–116
- crate root, 112, 115, 127
- crates.io*, 293–302
 - publishing to, 301–302
 - removing from, 302
 - setting up an account on, 300
- CRLF sequence, 462
- CTRL-C, 26, 54, 460, 487
- curly brackets ({}), 504
 - for function bodies, 5, 15
 - as placeholders in the `println!` macro, 18
 - scope creation, 46, 73
- custom derive procedural macros, 449–454

D

- dangling pointer, 74
- dangling reference, 74–75, 193–195, 199–200
- data race, 73, 425–426

- data types, 36–43
 - annotation of, 25, 36
 - compound, 40–43
 - scalar, 36–40
- deadlock, 349, 368, 491
- Debug trait, 91–92, 508
- declarative macros, 446–448
- deep copy, 66–67, 509
- Default trait, 510
- default type parameters, 429–430
- dependencies section in *Cargo.toml*, 8, 19
- dependency, 7, 8, 19–20
- deref coercion, 140, 321–323
- DerefMut trait, 322–323
- Deref trait, 317–323, 437
- derive annotation, 91–92, 449–454, 507–510
- destructor, 325
- destructuring
 - of enums, 406–407, 408
 - of nested items, 408
 - of structs, 405–406, 408, 409
 - of tuples, 40–41, 398–400, 409
- Dickinson, Emily, 237
- dictionary. *See* `HashMap<K, V>` type
- Dijkstra, Edsger W., 207
- Display trait, 91, 191–192, 205, 434–437
- diverging functions, 440
- division operator (/), 39, 500
- doc tests, 295
- documentation
 - comments, 293–296, 475
 - offline for Rust, 4
 - tests, 295
 - viewing a crate's, 22
 - writing, 293–296
- dot (.), 500
 - for method syntax, 93
 - for struct field access, 84
 - for tuple element access, 41
- double colon (::), 502
 - for associated functions, 96
 - for enum variants, 98
 - for namespacing, 115
- double free error, 65, 325
- double quote ("), 39, 501
- Doyle, Sir Arthur Conan, 287
- drop function, 64, 323–326
- Drop trait, 323–326, 487–489
- dynamically sized type (DST), 441–443
- dynamic dispatch, 380
- dyn keyword, 247, 376

E

- editions, xxi–xxii, 8, 498, 513, 515–516
- else if expression, 51–52
- else keyword, 50
- empty type, 440–441, 502
- encapsulation, 112, 372–374
- entry method, 147–149
- Entry type, 147–149
- enumerate method, 76, 398–399
- enums, 97–104
 - defining, 98–101
 - instantiating, 98
 - making public, 120–121
 - variants of, 98
- environment, 274–276
- environment variables, 255–260
- eprintln! macro, 261–262
- Eq trait, 508
- error handling, 151–169
- executable file, 6–7
- executing code, 6–7
- exit status code, 245–246
- expect method, 17–18, 25, 159–160
- expressions, 45–47
- extern functions, 424–425

F

- fearless concurrency, 348
- FFI (Foreign Function Interface), 424
- field init shorthand, 85
- fields, 84
- files, 237–238
- floating-point data types, 39
- fn keyword, 15
- FnMut trait, 271, 275–276, 443, 473
- FnOnce trait, 271, 275–276, 443, 473
- Fn trait, 271, 275–276, 443, 473
- fn type, 443–445
- Foreign Function Interface (FFI), 424
- for keyword
 - loop, 56–57
 - in trait implementations, 183–184
- format! macro, 139–140
- from function
 - on the From trait, 162
 - on String, 62–63, 138
- front of house, 114
- fully qualified syntax, 431–434, 439
- functional programming, 263–264
- function-like procedural macros, 455
- function pointers, 443–445

- functions, 43–48
 - arguments to, 44
 - bodies, statements and expressions in, 45–47
 - with multiple return values using a tuple, 69–70
 - parameters of, 44–45
 - public vs. private, 117–119
 - returning early from, 47
 - with return values, 47–48

G

- Gallant, Andrew, 234
- Gamma, Erich, 372
- garbage collector (GC), 60, 63
- generics, 171–182, 205
 - default types for, 429–430
 - in enum definitions, 178–179
 - in function definitions, 174–177
 - in method definitions, 179–181
 - performance of, 181–182
 - in struct definitions, 177–178
- get method
 - on HashMap<K, V>, 146
 - on Vec<T>, 133–134
- getter, 168
- Git, 8, 11
- global variables, 425–426
- grapheme clusters, 142–143, 144
- green threads, 349
- grep, 233–234
- guarding, 362
- guessing game, 13–30

H

- hash. *See* HashMap<K, V> type
- hasher, 149
- hashing function, 144, 149
- hash map. *See* HashMap<K, V> type
- HashMap<K, V> type, 144–149
 - entry method on, 147–148
 - get method on, 146
 - insert method on, 144–146
 - iterating over, 146–147
 - new function on, 144–145
- hash table. *See* HashMap<K, V> type
- Hash trait, 510
- heap
 - allocating on, 60–61, 312–313
 - and the stack, 60–61
- Helm, Richard, 372

Hoare, Tony, 102
HTTP (Hypertext Transfer Protocol),
458, 462–464
hyphen (-)
 for negation, 500
 for subtraction, 39, 500

I

IDE (Integrated Development Environment), xxvi, 4, 514
if keyword, 49–53
if let syntax, 109–110
ignore attribute, 226–227
immutability, *See* mutability
impl keyword
 for defining associated functions, 95–96
 for defining methods, 92–94
 for implementing traits, 183–186
impl Trait syntax, 186–187, 188–189
indexing syntax, 133–135
indirection, 316–317
inheritance, 374–375
input lifetimes, 202
input/output (io) library, 15
installation of Rust, 1–4
instance, 16, 84
integer data types, 36–38
 numeric operations with, 39
 signed, 36–37
 type suffixes of, 37
 unsigned, 36–37
integer overflow, 38
Integrated Development Environment (IDE), xxvi, 4, 514
integration tests, 228–232
interfaces. *See* traits
interior mutability, 330–336, 338, 368
invalidated variable, 66
io (input/output) library, 15
IpAddr type, 99–100
irrefutable patterns, 401–402
isize type
 architecture dependent size of, 37
 indexing collection with, 38
iterator adaptors, 279–281, 286–287
iterators, 276–289
 creating with iter method, 76
 enumerate method on, 76
 next method on, 281–283
 performance of, 287–289
iter method, 76

J

Johnson, Ralph, 372
JoinHandle type, 351

K

Kay, Alan, 371
keywords, 32, 495–498

L

Language Server Protocol, 514
last in, first out ordering, 60
lazy evaluation, 270, 276, 279
len method, 76
let keyword, 15–16
library crate, 7, 19, 111–113
license, 300–301
license identifier value, 301
lifetimes, 192–205
 annotation of, 196–204
 elision, 201–203
line feed, 462
linker, 2–3
lints, 513–514
Linux installation of Rust, 2–3
Little Book of Rust Macros, The, 448
lock, 363–365
loop keyword, 26–27, 54–55

M

macOS installation of Rust, 2–3
macro_export annotation, 447
macro_rules! macro, 446–448
macros, 446–455
 declarative, 446–448
 procedural, 449–455
main function, 5, 164
mangling, 425
map. *See* HashMap<K, V> type
match expression, 104–109
 exhaustiveness of, 108
 handling comparison results with, 23–24
 handling error values with, 158
 handling Result values with, 28
match guard, 413–415
memoization, 270
memory leak, 339, 345
message passing, 355–361
metaprogramming, 446

- methods
 - defined on enums, 101
 - defined on structs, 92–95
- method syntax, 93
- M:N threading model, 349
- mock object, 332–336
- mod keyword, 113–115
- modules, 112, 113–120, 127–128
 - moving to other files, 127–128
 - root, 115–116
- module system, 112
- module tree, 115
- monomorphization, 181–182
- move keyword, 275–276, 353–355
- moving ownership, 64–66
 - vs. borrowing, 70–75
 - with function calls, 68
 - with function return values, 68–70
- multiple producer, single consumer (mpsc), 356, 360, 480
- multiple trait bound syntax (+), 187, 500
- multiplication, 39
- mutability
 - of references, 72–74
 - of variables, 32–33
- Mutex<T> type, 362–368, 480–481, 486
- mut keyword
 - making a reference mutable with, 72–74
 - making a variable mutable with, 33
- *mut T, 419–421, 500
- mutual exclusion, 362

N

- namespace, 63, 96, 98, 113
- nested path, 126–127
- never type (!), 440–441, 502
- new function
 - on HashMap<K, V>, 144–145
 - on String, 137–138
 - on Vec<T>, 132
- new project setup, using cargo, 14
- newtype pattern, 436–438
- null, 101–104
- numeric operations, 39

O

- object, 372, 376. *See also* HashMap<K, V> type

- object-oriented programming (OOP), 371–393
- object-safe traits, 380–381, 389
- operator overloading, 429–430
- operators, 499–501
- optimizations, 10
- Option<T> enum, 101–104
- Ordering type, 23–24
- Ord trait, 509
- orphan rule, 184, 436
- output lifetimes, 202
- overflow of integers, 38
- ownership, 59–81
 - and functions, 68–70
 - rules, 61

P

- package, 112–113
- package section in *Cargo.toml*, 8, 300–301
- panicking, 38
- panic! macro, 152–155, 164–169
- parallel programming, 347
- parameters, 44–45
- parentheses, (()), 504
 - for function parameters, 5, 15
 - for tuples, 40–41
- parent modules, 115, 117
- parse method, 25
- PartialEq trait, 508
- PartialOrd trait, 509
- paths, 112, 113, 115–127
 - absolute, 115–116, 119
 - nested, 126–127
 - relative, 115–116, 119–120
- PATH system variable, 2, 3, 308–309
- patterns, 395–416
 - binding to values with, 106–107
 - in for loops, 398–399
 - in function parameters, 400
 - in if let syntax, 109–110, 396–397
 - in let statements, 399–400
 - in match expressions, 104–107, 396
 - refutable vs. irrefutable, 401–402
 - in while let syntax, 398
- .pdb file extension, 7
- pointer, 60, 311
 - dangling, 74
 - to data on the heap, 60–61
 - raw, 419–420
 - smart, 311–346

- poisoned mutex, 483
- polymorphism, 374
- prelude, 15, 127
- primitive obsession, 241
- println! macro, 6, 18
- privacy, 114, 117
- privacy boundary, 117
- privacy rules, 117
- private, 112, 114, 117, 228
 - struct fields, 120–121
- procedural macros, 449–455
 - attribute-like, 454–455
 - custom derive, 449–454
 - function-like, 455
- process, 348
- proc_macro crate, 449
- profiles, 292–293
- profile section in *Cargo.toml*, 292–293
- propagating errors, 160–164
- pub keyword, 116, 117–119
- public, 112, 114
 - API, 116, 296–299
 - making items, 117–119
 - making structs and enums, 120–121
- pub use, 124–125, 296–299
- push method, 132–133, 139
- push_str method, 63, 139

Q

- question mark operator (?), 162–164, 501
- quote crate, 451–454

R

- race conditions, 73, 349
- RAII (Resource Acquisition Is Initialization), 64
- rand crate, 19–23
- random number functionality, 19, 21–23
- Range type, 57
- raw identifiers, 497–498
- raw pointers, 419–421
- Rc<T> type, 326–330, 337–345
- read_line method, 15–18
- receiver, 356
- recoverable errors, 151, 155–164
- recursive types, 314–317
- re-export, 124–125, 296–299
- RefCell<T> type, 330–345

- reference counting, 312, 326–330, 366–368
- reference cycles, 339–345
- references
 - for accessing data from multiple places, 17
 - and borrowing, 70–75
 - dangling, 74–75
 - dereferencing, 70
 - mutability of, 72–74, 75
 - rules of, 75
- refutable patterns, 401–402
- registry, 20, 293–302
- relative path, 115–116, 119–120
- release mode, 10, 38
- release profiles, 292–293
- remainder operator (%), 39, 500
- request line, 462
- request-response protocol, 458
- Resource Acquisition Is Initialization (RAII), 64
- Result<T, E> type, 17–18, 155–164
 - expect method on, 17–18, 25, 159–160
 - vs. panic!, 164–169
 - in tests, 221
 - type aliases for, 439–440
 - unwrap method on, 159–160
 - unwrap_or_else method on, 159, 245–246, 248
- return keyword, 47
- return values
 - of functions, 47–48
 - multiple using a tuple, 69–70
- rev method, 57
- ripgrep, 234, 308–309
- RLS (Rust Language Server), xxvi, 514
- root module, 115–116
- .rs file extension, 5
- running code, 5, 6–7, 9–10
- runtime, 349
- Rustaceans, 3–4
- rustc, 3, 5, 6–7
- rustfix, 512–513
- rustfmt, xxvi, 6, 511–512
- Rust Language Server (RLS), xxvi, 514
- Rustonomicon*, *The*, 135, 346, 369
- rustup commands, 1–4
 - doc, 4
 - uninstall, 3
 - update, 3

S

- scalar data types, 36–40, 67–68
- scope, 62, 112, 113
- SCREAMING_SNAKE_CASE, 425
- Self keyword, 380–381
- self module, 115, 122, 127
- self parameter, 92
- Semantic Versioning (SemVer), 19–20, 302
- semicolon (;), 6, 42, 501
- Send trait, 368–369, 427, 473
- sequence, 57
- server, 458
- shadowing, 25, 34–36
- shallow copy, 66
- shared-state concurrency, 361–368
- should_panic attribute, 218–221
- sibling modules, 115
- signed integer types, 36–37
- single quote ('), 501–502
 - for characters, 39
 - for lifetime parameter names, 196
- ?Sized, 442–443
- Sized trait, 441–443, 445
- slice type, 75–81
 - of array, 81
 - string slices, 77–80
- smart pointer, 311–346
- snake case, 43
- Software Package Data Exchange (SPDX), 301
- square brackets ([]), 505
 - for array creation, 41
 - in the array type, 42
 - for element access, 42–43, 133–135
- stack
 - and the heap, 60–61
 - last in, first out ordering, 60
 - popping off of, 60
 - pushing onto, 60
- standard error (stderr), 260–262
- standard output (stdout), 260–262
- statements, 45–47
- state objects, 382
- state pattern, 382–393
- statically typed, 36
- static dispatch, 380
- 'static lifetime, 204, 425, 473
- static method, 16, 95–96
- static variables, 425–426
- status line, 463
- stderr (standard error), 260–262
- stdin function, 16
- stdout (standard output), 260–262
- &str (string slice type), 77–80
- stream, 459–461
- stringify! macro, 454
- string literal, 62
 - storage in the binary of, 63
 - of string slice type, 80–81
- string slice type (&str), 77–80
- String type, 62–63, 137–144
 - as_bytes method on, 76
 - bytes method on, 143–144
 - chars method on, 143
 - concatenation with +, 139–140
 - from function on, 62–63, 140
 - indexing into, 141–142
 - internal structure of, 64–65, 141–142
 - iterating over, 143–144
 - len method on, 76
 - new function on, 137–138
 - parse method on, 25
 - push method on, 139
 - push_str method on, 63, 139
 - slicing, 142–143
 - trim method on, 25
 - UTF-8 encoding of, 137
- Stroustrup, Bjarne, 288
- structs, 83–96
 - defining, 83–84
 - field init shorthand, 85
 - fields, 84
 - instantiating, 84–85
 - making public, 120–121
 - ownership of data, 87–88
 - tuple, 86–87, 436–437
 - unit-like, 87
 - update syntax, 86
- subtraction, 39
- super keyword, 115, 119–120
- supertraits, 434–436
- symbols, 501–505
- syn crate, 451–453
- Sync trait, 368–369, 427

T

- TCP (Transmission Control Protocol), 458–460
- test attribute, 208–209
- test double, 332
- test-driven development (TDD), 250
- test functions, 208–211
- tests, 207–232
 - custom failure messages for, 216–217
 - filtering, 224–226
 - ignoring, 226–227
 - integration, 228–232
 - organizing, 227–232
 - of private functions, 228
 - running, 221–227
 - unit, 227–228
 - using `Result<T, E> in`, 221
 - writing, 208–221
- thread pool, 469–493
- threads, 348–355
 - creating with `spawn`, 350, 470–473
 - joining, 351
 - pausing with `sleep`, 350
- thunk, 438–439
- Tom’s Obvious, Minimal Language (TOML), 8
- `to_string` method, 138, 192
- trait bounds, 187–192, 205
 - conditionally implementing methods with, 191–192
 - fixing the largest function with, 189–191
- trait objects, 375–381, 445
 - dynamic dispatch, 380
 - object safety, 380–381
- traits, 182–192
 - associated types in, 427–428
 - default implementations of, 185–186
 - defining, 182–183
 - implementing, 183–184
 - unsafe, 426–427
- Transmission Control Protocol (TCP), 458–460
- transmitter, 356
- `trim` method, 25
- tuple data type, 40–41, 69–70
- tuple structs, 86–87, 436–437, 444–445
- two’s complement wrapping, 38
- type alias, 438–440, 481–482

- type annotation, 25, 36
- type inference, 24
- type suffixes, 37

U

- underscore (`_`), 502
 - as a catchall pattern, 28, 108–109, 409–411
 - as a visual separator in integer literals, 37
- Unicode Scalar Value, 40, 141–144
- Uniform Resource Identifier (URI), 462
- Uniform Resource Locator (URL), 462
- unit-like structs, 87
- unit tests, 227–228
- unrecoverable errors, 152–155
- unrolling, 288–289
- unsafe, 418–427
 - functions, 421–424
 - superpowers, 418–419, 427
 - traits, 426–427
- unsigned integer types, 36–37
- unsized type, 441–443
- unwinding, 152
- `unwrap` method, 159–160
- `unwrap_or_else` method, 245–246
- URI (Uniform Resource Identifier), 462
- URL (Uniform Resource Locator), 462
- use keyword, 22, 112, 121–127
 - and `as`, 124
 - and external packages, 125–126
 - and the `glob` operator, 127
 - and nested paths, 126–127
 - and `pub`, 124–125
- user input, 15
- usize type
 - architecture dependent size of, 37
 - indexing collection with, 38
- UTF-8 encoding, 137, 141–142

V

- variables
 - vs. constants, 34
 - global, 425–426
 - mutability, 32–33
 - shadowing, 25, 34–36
 - static, 425–426
 - storing values in, 15–16

- variants, 98
- vec! macro, 132
- vector. *See* Vec<T> type.
- Vec<T> type, 132–136
 - get method on, 133–135
 - iterating over, 135–136
 - new function on, 132
 - push method on, 132–133
- vertical pipe (|), 501–502
 - in closure definitions, 267
 - in patterns, 404
- Visual Studio, 3
- Visual Studio Code, 514
- Vlissides, John, 372

W

- warnings, 512–513
- weak reference, 341–342
- Weak<T> type, 341–345
- where clause, 188
- while loop, 55–56
- Windows installation of Rust, 3
- workspaces, 303–308
- wrapping type, 38

Y

- yanking, 302

Z

- zero-cost abstractions, xxvii, 288–289
- zero-overhead, 288