Index

Symbols

< > angle brackets, 328

(=) assignment operator, 41, 102, 141, 184

(\) backslash, path separator, 58, 228, 229

($) command-line prompt, Linux, 19

(%) command-line prompt, macOS Terminal, 19

(>) command-line prompt, Windows Terminal, 19

*complement dereferences, 209

(/) division sign operator, 62

(") double quotation marks, 55, 228

(==) equality operators, 308, 335

(=) equals sign operator, 62, 65, 74, 184

(?) error propagation operator, 193

(\") escape sequence, 229

(\n) escape sequence, 229

\t escape sequence, 241, 245

(>–greater than symbol) thin arrow return type, 114

(#) hashtag, 219

(–) minus sign, 55

(*/) multiple-line comments, 46

(*) multiplication operator, 62

(!=) nonequality operators, 308

:: operator, 58

πr^2 formula, 68

(|) pipe symbol, 97

(+) plus sign, 55

({:p}) pointer formatter, 150

(%) remainder operator, 64–65, 89

(;)semicolon, 41

(//) single-line comments, 46

(') single quotation marks, 55, 228

#[...] syntax, 309

(_) underscore separator, 52

(~) user account directory, 28

(_) wildcard character, 94, 98, 102, 189

A

a.cmp(b) method, 258, 259

adapting iterators

definition, 269

enumerate() method, 270–271

filter() method, 269

map() method, 270

COPYRIGHTED MATERIAL

add_numbers() function, 115
add_one()s function, 123
Adobe Photoshop, 506
analyze_password() function, 523–524
AND (&&) operators, 81–82, 148
anonymous loop, 88
ANSI escape sequences, 493
AppConfig struct, 357
application programming interface (API), definition of, 594
APP_NAME variable, 119
Arc (atomic reference counter), 411, 559, 602, 626, 627
  sharing ownership, 407–409
archive subcommand, 500–501
area variable, 75, 99
arguments, 255
  function of, 115, 117, 142, 158, 205, 210, 228, 239, 348, 376, 534
    definition of, 110
    multiple arguments, 112–113
    as mutable, 113–114
    read-only text arguments, 113
    single arguments, 111–112
  text argument, 156
arithmetic assignment operators, 65–66
arithmetic operators, 62–64
  division sign (/) operator, 62
  multiplication (*) operator, 62
  remainder operator (%), 64–65
array_contains() function, 334–336
arrays, 529
  bounds checking in, 203
  definition of, 201
  element in, 203, 204
  functions of, 204–206
  index, 202
  iteration, 203–204
  JavaScript, 532–535
  numeric, 530
  repeat expression syntax, 202
  types of, 204–206
  vector creation, 212–213
Askama, 621–625, 639
as keyword, 64
assertion macros, 427
assignment operator (=), 41, 102, 141, 184
associated functions, 168–169, 171
  constructor, 169

asynchronous programming, 414
async keyword, 414–416, 513, 578
atomic reference counter (Arc), 411, 559, 602, 626, 627
  sharing ownership, 407–409
authentication, for web application, 628
  session management, 629–630
  updating the home page with login status, 637
  user login, 634–636
  user logout, 636–637
  user registration, 630–634
authentication flow modelling project
  error propagation operator (?), 193
  example of, 195
  main() function, 194
  ResultString, io:Error return type, 193
  unwrap_or_default() method, 194
  user input, 193
author: author function, 164
AutoCAD, 507
await keyword, 414–416, 541, 628
Axum, 571, 575–579, 618, 625–628, 631, 633, 642
  routing works in, 580

B

backslash (\) path separator, 58, 228, 229
Bank Account Manager project
  adding implementation block, 171–173
  adding user input functions, 173
  check_balance() method, 172
  creating instance of struct, 173–174
  deposit() method, 172
  display_account_info() method, 173, 174
  error handling, 191
  get_amount() function, 173
  get_user_input() function, 173
  main menu loop implementation, 175
  setting up the struct, 171
  unwrap_or_default() method, 173
  withdraw() method, 172, 191
BankAccount struct, 173, 174
behavioral trait, 305
binary (base 2), 52
binary crate, 280
binary size, 507
binding generator, 515
blank placeholder, 47–48

blocks, expressions, 74–75, 255
block scope, 117
  definition of, 99
  example, 99–100
block syntax, 78
book library API, 600–604
  database migration with SQLx, 609–614
  deployment considerations, 615
  error handling, 607–609
  testing, 614–615
  validation, 607–609
bookmarks, in database
  adding, 639–641
  deletion, 643
  displaying, 637–639
  editing, 641–642
`BookStore` type, 603
`Book` struct, 161, 162, 169, 355, 601
`bool` data type, 54–55, 329
borrow checker, 14, 146–149, 152, 153, 156, 158, 264, 345, 350
borrowed string. *See* `(&str)` string slice type
borrowing, 140, 144
  and function calls, 145
  golden rule of, 146–149
  immutable borrowing, 144–149, 154
  mutable borrowing, 146–149, 154
  non-owning variable, 144
  references, 149–153
  sharing XOR mutating, 148
  string parameters, 145
bounds checking in arrays, 203
`break` keyword, 85
break mode, 436
`break` statement, 84, 89, 90, 189
B-Tree map, 221
buffer overflows, 135
`BufReader`, 473
`BufRead` trait, 458

C

C++, 13, 14
`calculate_tip()` function, 116, 121
Cargo
  capabilities of, 25
  `Cargo.toml` file (*see* `Cargo.toml` file)
  commands, 26
  `rand` version `0.9.2`, 296
  `src` subdirectory, 29–30
`cargo build` command, 26, 34, 280, 282, 361, 449

`cargo check` command, 26, 34
`cargo clean` command, 26
`cargo: command not found` error, 23
`cargo doc` commands, 26
`Cargo.lock` file, 31
`cargo new` command, 282, 431
`cargo new` *`project`* commands, 26
`cargo run` command, 26, 34
`cargo test` command, 26, 424, 426, 429, 433, 445
`Cargo.toml` file, 280, 296, 297, 302, 415, 461, 494, 495, 521, 530, 539, 548, 552, 609, 614
  dependencies, 31, 297
  edition, 31
  name, 31
  version, 31
`cargo --version` commands, 23
case-conversion methods
  example, 243–244
  output, 244
  `string.make_ascii_lowercase()` method, 244
  `string.make_ascii_uppercase()` method, 244
  `string.to_lowercase()` method, 243
  `string.to_uppercase()` method, 243
`cd` *`path`* (change directory), 20
`cd` *`project`* command, 35
`cedilla` variable, 54
change directory (`cd` *`path`*), 20
channel, message passing, 403–404
  multiple producers, single consumer, 406–407
  sender and receiver, 404–406
`char` data type, 54, 237
`char.is_ascii``_alphanumeric()` method, 233
`char.len_utf8()` method, 237
`chars()` method, 185, 232–233
`check_balance()` method, 172
`choice` variable, 102, 103, 189
clamp, 337
cleaning service of code, 439
  Clippy, 441–443
  `rustfmt`, 439–441
Clippy, 441–443
`clone()` method, 141, 142, 308, 312
`Clone` trait, 308–310, 312
cloning, 141–142, 154, 235, 309
closures
  capturing local variables, 256–257
  definition of, 255
  general syntax for, 255
  `sort_by_key()` method, 257–258

closures *(continued)*
`sort_by()` method, 258–259
to write, 259–260
code
cleaning service, 439
Clippy, 441–443
`rustfmt`, 439–441
debugging, 433
VS Code user, 435–437
`dbg!` macro, 434–435
`println!` macro, 438–439
documentation, 443
adding comments, 443–445
building your program, 446
running documentation examples as tests, 445–446
testing, 422–423
assertion macros, 427
code panics, 428
integration tests, 432–433
library, 431
module, 429–431
returns `Result`, 427–428
writing, 423–426
CodeLLDB, 435
code organization
access control
encapsulation, 292–293
private by default, 290–292
crate
binary crate, 280
definition of, 280, 296
documentation, 297
library crate, 280
as project dependency, 296
`rand` rate, 296
dependency, 296–297
module
crate root module, 280
definition of, 280
and files, 281
inline module, 283–284
shortening the module paths, 294–295
tree of, 280, 281
in multiple files
creating own module files, 288–290
mapping rules, 287–288
using `main.rs` and `lib.rs`, 287
in one file
creating modules in `main.rs`, 283–284
`main.rs vs lib.rs` decision, 282–283
nesting modules, 284–287
package, 280

real-world organization patterns, 298–302
`use` keyword, code importing, 293–294
coding process
abstraction, 14
repeat, 12
revise, 12
run, 12
write, 12
Collatz Conjecture, 416–420
collection, definition of, 87
`collect()` method, 245, 268, 273, 275
color scheme, 65
command line, 21
basic command-line survival skills, 20
definition of, 18
keyboard shortcuts, 19–20
launching terminal, 18–19
shell, 18
terminal, 18
command-line interface (CLI) tools, 449–451, 465, 487–489, 491, 493, 494, 496, 498
arguments, 451–452
clarity, 450
composability, 450
courtesy, 450
file operations, 459–460
parsing arguments, 452–457
standard input (`stdin`), 457–458
with subcommands
adding, 488–490
working with nested, 491–493
`word-replacer` tool, 461–464
comma-separated values (CSV)
reading files, 477–479
writing files, 477–479
comment, definition of, 46
comparison expressions, 54–55, 78, 86
comparison operators, 68
definition of, 68
example, 69
comparison operators, 68
compiler, definition of, 11
computer, description, 8, 254
concatenation operator, 234
concurrency safety of Rust, 13
concurrency vs parallelism, 396–397, 403
`ConfigError` enum, 378
`config.rs` file, 299
constants, 56–57, 119, 272
constructor, 169, 171, 341

consuming iterators
collect() method, 268
example, 265–266
find() method, 266–268
iterator.count() method, 265
iterator.last() method, 265
iterator.max() method, 265
iterator.min() method, 265
iterator.product() method, 265
iterator.sum() method, 265
output, 266
contains() method, 242, 243, 273, 324
continue statement, 89
control block, 151
conversion operator, 64
copied() method, 359
Copy trait, 137–138, 140, 166, 212, 219, 337
cost_per_splice() function, 187
counts.get_mut(w) method, 274
C programming language, 10, 13, 14
CPU-bound operation, 397
crate
binary crate, 280
definition of, 280, 296
documentation, 297
library crate, 280
multiple crates, 301–302
as project dependency, 296
root module, 280
CreateBook structs, 601
Cross-Site Request Forgery (CSRF), 628
CRUD operations, 600
book update, 605–606
deletion, 606–607
single book with path parameters, 604–605
CSS files, 571, 575, 589, 590, 625–626
curl command, 22

D

dangling pointer, 135, 158, 344
database
bookmarks
adding, 639–641
deletion, 643
displaying, 637–639
editing, 641–642
connection pool, 626–628
design for LinkLocker, 619–620
initialization, 620–621
database.rs file, 299
data cleanup procedure
garbage collection, 134, 135
manual memory management, 134, 135

data storage
  arrays, 201–206
  hash maps, 217–222
  slices, 206–211
  tuple, 198–201
  vectors, 211–217
`days_in_ month()` function, 122
`days_in_month` variable, 96
`dbg!` macro, 434–435, 438
debugging of code, 433, 591–592
  VS Code user, 435–437
  `dbg!` macro, 434–435
  `println!` macro, 438–439
`Debug` trait, 206, 310, 316
declarative macro, 384, 387
deep copy, 141
deep-copying heap data, 141–142
`Default` trait, 318–319
`DELETE` (removing resources), 600
`demonstrate_parse_error()` function, 373–374
dependency
  adding a crate, 296
  definition of, 296
  external dependency, 296
  importing to `Cargo.toml`, 297
`deposit()` method, 172
deref coercion, 239–240, 548
dereferencing, 150
  with mutable slice, 209–210
`derive` attribute, 309
deserializing data, 480–482
destructuring, tuple, 200
directory (`dir`) on Windows, 20
`dir` (directory) on Windows, 20
`display_account_info()` method, 173, 174
`display_app_title()` function, 285
`Display` trait, 306–307, 310, 316–318, 324, 341, 377, 379
`display_welcome()` function, 286
division sign (`/`) operator, 62
doctest, 446
Document Object Model (DOM), 506, 514, 520
Don't repeat yourself (DRY). *See also* generics
  definition, 325
  violations, 327
dot notation, 162–163
double-free, 135
double quotation marks ("), 55, 228

E

elision rules, for lifetimes, 354–356
empty tuple, 199

empty vector, 211, 264, 363, 364, 411, 639
enum (enumerations), 159, 161. *See also* generics
adding data, 180–181
adding trait to, 309
creating an instance of, 178–179
definition of, 177, 180
error handling
assigning default value, 189–190
example, 188
`if let` pattern-matching, 192
return functions, 190–192
`(_)` wildcard character, 189
general syntax of, 178
`if let` pattern-matching, 179–180, 184
`LoginError` enum, 281
`Option` enum (*see* `Option` enum)
`Ordering` enum, 258–259
PascalCase, 178
`Result`, 93, 188–190
`TrafficLight` enum, 178, 179
variants, 178
`enumerate()` method, 270–271
equality `(==)` operators, 308, 335
equals sign operator (=), 62, 65, 74, 184
error, location of, 53
error handling
pattern matching with `match` keyword, 365–366
problems with, 372–373
`Result` type, 364–365
error propagation operator (`?`), 193
error-related helper methods
`unwrap()` and `expect()`, 366–367
`unwrap_or_default()` method, 370–371
`unwrap_or_else()` method, 369–370
`unwrap_or()` method, 367–369
error types, 373–374
`Err(String)`, 377–380
`Err` variant, 189–191, 365, 366, 371
escape sequence `(\")`, 229
escape sequence `(\n)`, 229
escape sequence `(\t)`, 229
Escape the Dungeon project (text-based adventure game)
setting up the game, 102–103
user input, 101–102
pattern-matching, 104–106
variable shadowing, 103
`excerpt` variable, 156
`expect()`method, 366–369

explicit lifetimes, 346
expression-based return, 115
expressions, 61. *See also specific expressions*
  blocks, 74–75, 255
  comparison, 54–55, 78, 86
  definition of, 62
  logical, 78, 79, 86
  numeric, 62–68
  operands, 62
  operator, 62
  remainder, 65
  structure, 62
external data, 480

F

`factorial()` function, 123
fat pointer, 151–152
`favorite_number` variable, 41, 42, 44, 45, 51
fearless concurrency, 14, 396
`fetch_and_render()` function, 541
field init shorthand, 164–165
Figma, 507
file-handling techniques, 482–485
file operations, for command-line interface (CLI) tools, 459–460
files vs modules, 281
file system
  fiddling, 470–471
  manipulating paths, 466–470
    building, 467–468
    components, 468–470
    new, 467
`filter()` method, 269
`filter(|w| !STOPWORDS.contains(&w.as_str()))` method, 273
`filter(|w| !w.is_empty())` method, 273
`find()` method, 182, 183, 185, 186, 243, 266–268
fixed-sized storage. *See* arrays
`Float32Array`, 530, 531
floating-point constants, 66–67
floating-point expressions, 64
floating-point methods, 67–68
floating-point number
  `bool`, 138, 161
  `char`, 138, 161
  definition of, 53
  `f32`, 53, 205, 329
  `f64`, 53, 173, 329
flow control
  `if...else` statement, 79–80
  `if` statement, 78–79

flow control *(continued)*
making multiple decisions
`if` expression, 82–83
multiple `if/else` blocks, 80–81
using AND (`&&`) and OR (`||`) operators, 81–82
using loops, 83–91
`fmt` module, 307, 316
`fn main(){ }` program, 197
`for` loops, 87–89, 249, 251, 262–264, 381, 393, 405, 419, 439
`format_currency()` function, 116, 121
`format!` macro, 235, 323, 324
foundation
database design for LinkLocker, 619–620
database initialization, 620–621
project structure, 618–619
`from()` method, 212, 218, 219, 541
function. *See also* generics
advantages, 120
arguments, 115, 117, 142, 158, 205, 210, 228, 239, 348, 376, 534
definition of, 110
multiple arguments, 112–113
as mutable, 113–114
read-only text arguments, 113
single argument, 110–111
basic structure of, 109
definition of, 108
helper function, 121–122
importance of, 108–109
with no arguments, 110
recursive function, 122–124
return values, 114–116
separating responsibilities, 120–121
functional programming
closures
capturing local variables, 256–257
definition of, 255
general syntax for, 255
`sort_by_key()` method, 257–258
`sort_by()` method, 258–259
to write, 259–260
definition of, 254
iterator
adapting iterators, 269–271
consuming iterators, 264–268
creation, 261–264
definition of, 260
uses of, 260
`function_name` function, 110
function scope
advantage, 118
code, 117

definition of, 117
local_variable variable, 118
output, 118
function signature, 112, 115, 240, 327, 329, 335, 338, 349

G

Games, 507
garbage collection, 13, 134
garbage collector, 100, 135, 507
generate_wave() function, 531–532
generic lifetime parameter declaration, 348, 349, 353
generics, 340–342
<T> (type parameter), 327–335
<U> (type parameter), 333
<V> (type parameter), 333
description of, 327
making enum generic, 331–332
making function generic, 328–329
making struct generic, 329–330
trait bounds, 335–338
GET (reading resources), 600
get_amount() function, 173
get_character_count() function, 156
get_duration() method, 321, 323
get_first_element() function, 363
get_length() function, 142, 143, 156
get_length(&msg) function, 145
get_longest_word() function, 156
get_menu_choice() function, 125, 126, 129
get_mut() method, 274
get_rating() method, 323
get_review_count() method, 323
get_searchable_text() method, 321, 324
get_temp() function, 126, 129
get_user_input() function, 173, 175
get_value() method, 359
get_word_count() function, 156
.gitignore file, 32
global scope, 119, 518
golden rule of borrowing, 146–149
Google Earth, 507
greet() function, 598

H

handler functions, 580–581, 583, 586, 587, 631
handlers.rs file, 299
hashing algorithm, 619, 630
HashMap data type, 217

hash_map.get() method, 220
hash_map.insert(*key, value*) method, 220
HashMap<K, V>, 218
hash_map.remove(*key*) method, 221
hash maps
accessing values, 219–220
creation
from array of tuples, 218–219
new hash map, 218
definition of, 217
HashMap<K, V>, 218
iteration, 221–222
key-value pairs, 217
adding and removing, 220–221
mutable hash map, 220, 221
hashtag (#), 219
heap, 138–141
deep-copying heap data, 141–142
heap-allocated variable, 141
height variable, 99
helper function, 126
days_in_ month() function, 122
definition of, 121
examples, 121–122
is_leap_year() function, 122
is_valid_date() function, 122
output, 122
hexadecimal (base 16), 52
Hoare, Graydon (programmer), 12
HTML, 446, 512–514, 518, 523, 573, 575, 579, 589, 590, 617
reading HTML from files, 582–584
return HTML responses, 581–582
with superpowers, 621–625
human language, 9
Hypertext Transfer Protocol (HTTP), 575–576, 618, 628, 629, 631
request, 572–573
response, 574–575

I

i32 integers, 50, 125, 126, 204, 209–211, 218, 227, 328, 329
idiomatic Rust, 441
if...else statement, 79–80
if expression, 78, 79, 82–83, 86
if let pattern-matching, 192
if statement, 78–82, 97, 104, 254
immutable borrowing, 144–149, 154, 209
immutable variable, 44, 56
imperative programming, 254

implementation (`impl`) block, 167, 169, 171–173, 314
implicit lifetimes, 345
`impl` keyword, 167
inbugging, 433
index, arrays, 202
indexing operator method, 213–214
infinite recursion, 123
`infinity_symbol` variable, 54
`initialize()` function, 119
inline bounds, 338
inline module, 283–286
inner documentation, 443
installation, Rust
  components installed, 22
  on Linux, 22
  on macOS, 22
  verification, 22–23
  on Windows, 21
`instance` syntax, 165, 166
`Int32Array`, 530
integers, 43, 63, 137, 161
  with floating-point type, 64
  `i32`, 50, 125, 126, 204, 210, 211, 218
  platform-dependent, 51
  types of, 49–53
  `u8`, 51, 189, 327, 329
integration tests, 432–433
internal data, 480
interpreter, definition of, 10
`into_iter()` iterator, 263–264, 267
`IntoResponse`, 580, 581, 618, 625
I/O-bound operation, 397
`io` module, 101, 285, 289
`is_empty()` method, 233
`isize` type, 51
`is_leap_year()` function, 122
`is_prime` vector, 224
`is_valid_date()` function, 122
`item_average` variable, 62
`item_count` variable, 62
item-level documentation, 444
`item_total` variable, 62
iterator, 157
  adapting iterators
    definition, 269
    `enumerate()` method, 270–271
    `filter()` method, 269
    `map()` method, 270
  consuming iterators
    `collect()` method, 268
    example, 265–266
    `find()` method, 266–268
    `iterator.count()` method, 265

iterator *(continued)*
`iterator.last()` method, 265
`iterator.max()` method, 265
`iterator.min()` method, 265
`iterator.product()` method, 265
`iterator.sum()` method, 265
output, 266
creation
borrowing immutably with `iter()`, 261–262
borrowing mutably with `iter_mut()`, 262–263
owning with `into_iter()`, 263–264
definition of, 260
uses of, 260
`iterator.any(closure)` method, 268
`iterator.count()` method, 265
`iterator.``for_each`*`(closure)`* method, 271
`iterator.last()` method, 265
`iterator.max()` method, 265
`iterator.min()` method, 265
`iterator.product()` method, 265
`iterator.sum()` method, 265
`iter()` method, 261, 264
`iter_mut()` method, 262–263

J

JavaScript, 33, 225, 506, 571, 575, 589, 591
array of strings to Rust, 534–535
array to Rust, 532–533
functions, 515–519
namespaces, 515–517
Rust user profile struct to, 525, 535
Rust vector to, 530
string, 527–529
vector of strings to, 533–534
JavaScript Object Notation (JSON), 477, 480–483, 597
`Greeting` struct, 598
vs HTML, 595–596

K

key, definition of, 257
keyboard shortcuts, command line, 19–20

L

labeling loops, 90–91
layout template, 622–624
`len()` method, 142, 232
`let` statement, 51, 345
lexical scope. *See* block scope
library crate, 280

`lib.rs` decision, 282–283, 287, 298
lifetimes, 343–346
  annotation, 346–347
  definition, 343
  elision rules, 354–356
  facts about, 344
  function, 347–348
    multiple lifetime annotations, 350–351
    return reference, 350
    signature, 349
    unsafe designs, 351
  `static` keyword, 356–357
  structs, 352–353
  zero-copy parser approach, 357–360
linear memory, 527
LinkLocker
  database design for, 619–620
  extending instructions, 643
`linklocker.db` file, 621
Linux
  Rust installation, 22
  Terminal app launching, 19
Linux kernel, 14
`load_game()` function, 482
`load_server_config()` function, 380
local scope. *See* function scope
`local_variable` variable, 118
`lock()` method, 458
logging, in web server, 591–592
logical expressions, 78, 79, 86
  definition of, 69
  example of, 70
  logical operators, 69
logical operators, 69
`LoginError` enum, 281
`login_screen` module, 281
`longest_word()` function, 275–277
`longest_word` variable, 157, 158
loops, 203, 216
  anonymous loop, 88
  basic loop, 84–85
  bypassing loop statements, 89
  conditional repetition, 86–87
  `continue` statement, 89
  definition of, 84
  labeling loops, 90–91
  `for` loops, 87–89
  main menu loop, 175
  nesting loops, 89–90
  using loop as expression, 85
  `while` loops, 86–87, 224

`lowercase()` method, 103
`ls` (list), Mac and Linux, 20

M

machine language, 10
macOS
  Rust installation, 22
  Terminal app launching, 18–19
`macro_rules!`, 384–387, 389, 390, 392
macros, 381
  code, 383–384
  `format!` macro, 235, 323, 324
  `println!` (*see* `println!` macro)
  syntax, 384–387
  timer macro, 391–393
  `vec!` macro, 213, 382–383
  writing, 387–391
    converting of key-value pairs to vector, 390–391
    sums any number of inputs, 389–390
`main()` function, 40, 74, 77, 108, 111, 118–120, 127, 145, 191, 194, 199, 252, 272, 276–277, 282, 283, 292, 463–464, 484–485, 498–499
`main.rs` file, 29, 30, 40, 111, 155, 160, 171, 223, 279, 281–283, 287
`make_ascii_lowercase()` method, 263
`make_cookies()` function, 292
make directory (`mkdir` *name*), 20
`make_string_tuple()` function, 326, 328, 329
manual memory management, 134
`map()` method, 270
mapping rules, module files
  submodules, 288
  top-level modules, 287
`map(|w| w.to_lowercase())` method, 273
Markdown, 444–446, 538–543
`matches_query()` method, 321, 324
`match` expression, 129
  arm of, 94
  example, 95–96
  general syntax, 94
  matching one value or another, 97
  matching range of values, 96
  matching with a condition, 97–98
  pattern alternative syntax, 97
`match` keyword, 179, 365–366
`match` structure, 100, 104, 129, 179, 253
`max_by_key()` method, 276
`MAX_PLAYERS` variable, 119
Mcnamara, Tim, 617
`Measurement` enum, 332

media library building project
`Display` trait, 324
`Movie` struct, 321–322
`Playable` trait, 321
`Rateable` trait, 320, 322–323
`Searchable` trait, 321, 323–324
memory, facts of, 33
memory leak, 134
memory management, 100, 507
failure of
buffer overflows, 135
dangling pointer, 135
double-free, 135
memory leak, 134
memory safety, 13, 135, 141, 143, 148–149, 507
memory storage, variables, 42–43
message passing, 403–404
multiple producers, single consumer, 406–407
sender and receiver, 404–406
metaprogramming, 382
method, definition of, 166
method chaining, 266
mise en place principle, 40
missing command-line argument, 451
`mkdir` *name* (make directory), 20
`mod` block, 283, 284
module-level documentation, 443
modules, 58, 280
`config.rs` file, 299
crate root module, 280
`database.rs` file, 299
definition of, 280
and files, 281
`fmt` module, 316
`handlers.rs` file, 299
inline module, 283–284
`io` module, 101, 285, 289
`login_screen` module, 281
in `main.rs`, 283–284
mapping rules, 287–288
nesting modules, 284–287
shortening the module paths, 294–295
`std::io` module, 125, 129, 171
tree of, 280, 281
`user_interface` module, 285, 286, 288–290, 294
`utils.rs` file, 299
monomorphization, 329
`msg` variable, 142, 143
multiblock statement, 80

multicommand file organizer, 496–501
archive subcommand, 500–501
command-line interface structure, 498
main() function, 498–499
scan subcommand, 499
stats subcommand, 501
multiple arguments, function
examples, 112, 113
output, 113
&str, 113
multiple crates, 301–302
multiple if/else blocks, 80–81
multiple lifetime annotations, 350–351
multiple-line comments (*/), 46
multiplication (*) operator, 62
multi-threaded code, 395
mutable borrowing, 146–149, 154
Mutex, 410–411
mut keyword, 45, 114, 146, 163–164, 290, 291

N

named placeholders, 47
nesting loops, 89–90
nesting modules, 284–287
nesting parentheses, 73
newline character, 47
next() method, 454–455
Node.js installation, 509
nonequality (!=) operators, 308
non-lexical lifetime, 149
note-taking tool, 489–490
numbered placeholders, 48
numeric expressions
arithmetic operators in, 62–64
definition of, 62
floating-point constants in, 66–67
floating-point methods, 67–68
remainder operator (%) in, 64–65
using arithmetic assignment operators, 65–66
numeric type conversion, 64

O

octal (base 8), 52
ok_or_else() method, 376–377
ok_or() method, 375–376
open() method, 476
operand, definition of, 62
operator, definition of, 62
Option enum, 93, 181–183, 331, 374–375
ok_or_else() method, 376–377
ok_or() method, 375–376

None arm, 184
assigning a default value to, 185–186
return functions, 186–187
Some arm, 184
Optionf64 enum, 187
Option<usize> method, 182
Ordering enum, 258–259
Equal, 258
Greater, 258
Less, 258
order of precedence
control of, 72–73
for operators, 70, 71
OR (||) operators, 81–82, 148
owned string. *See* String type
ownership
borrowing values, 143–149
cloning, 141–142
copying owned values, 137–138
Copy trait, 137–138, 140, 166, 212, 219
example of, 135
passing values into functions, 142–143
of password, 148
String type, 138–140, 142–145, 150, 157, 158
three rules of
each piece of data can have only one owner, 136, 137
every value has an owner, 136
when the owner goes out of scope, the value is dropped, 136
transferring ownership to new variable, 141

P

package, definition of, 280
package manager
definition of, 24
Maven (Java), 24
npm (Node.js), 24
pip (Python), 24
pages directory, 582
parallelism vs concurrency, 396–397, 403
parse_age() function, 372
ParseIntError, 367, 373–374, 379
parse() method, 125, 189
parsing arguments, 452–457
PartialEq trait, 307–308, 310, 311, 335, 336
PartialOrd trait, 337
PascalCase, 161, 178
password analyzer, 520–524
password sharing, 148

pattern alternative syntax, 97
pattern-matching
  `if let` pattern-matching, 192
  `match` expression, 94–98
  with `match` keyword, 365–366
  `Option` enum, 179–180, 184
  and scope, 98–100
  user input, 104–106
pi (`PI`) constant, 58, 59, 68
pipe symbol (|), 97
platform-dependent, integer, 51
`Playable` trait, 321, 322
`play()` method, 321, 323
pointer, 135, 139, 144, 149, 151–152, 170, 230, 239, 408, 409, 478–479
pointer formatter (`{:p}`), 150
positional parameters, 112
`POST` (creating resources), 600
PowerShell, 18
prelude, standard library, 58
`prev_hyphen` Boolean, 248, 249
`price_per_unit()` function, 364–365
prime-explorer project, 222–224
`primes.iter()` method, 262
`print_converted_temp()` function, 127
`println!` macro, 29, 30, 46–48, 58, 74, 79, 99, 101, 137, 143–145, 152, 155, 205, 209, 219, 256, 264, 292, 382–383, 431, 434, 438–439
print working directory (`pwd`), 20
`process_file()` function, 463
programming language. *See also* Rust
  C programming language, 10, 13, 14
  definition of, 9
  role of, 9–11
progress bars implementation, 487, 494–496
`prompt()` function, 194
`ProvidesEnergy` trait, 339, 340
`pub` keyword, 284, 291, 292
`pulldown-cmark`, 539, 540
`push_hyphen()` function, 248
`push()` method, 138
`push_str()` method, 138
`PUT` (updating resources), 600
`pwd` (print working directory), 20
Python, 33, 225

Q

`question.find("nothinf")`, 183
`question` variable, 183

R

range syntax, 96, 237–238

`Rateable` trait, 320, 322–323

`rate()` method, 320, 323

raw strings, 229, 582

Raymond, Eric (programmer), 325

Rayon, parallelizing iterations with, 413

reading files, 471–472

- appending text, 475–476
- comma-separated values, 477–479
- increment, 472–473
- one fell swoop, 472

`read_line()` function, 101–103

read-only data segment, 230

read-only text arguments, 113

`read_shopping_list()` function, 472

`read_stock_count()` function, 371

`read_to_string()` function, 472–474, 481, 482, 484

real-world organization patterns

- organizing larger applications, 298–300
- re-exports, 300–301
- splitting project into multiple crates, 301–302

recoverable error vs unrecoverable error, 362–364

recursion, 122–124

recursive function

- `add_one()` function, 123
- examples of, 122–123
- `factorial()` function, 123

re-exports, 300–301

references, borrowing, 144, 149, 156

- dereferencing, 150
- fat pointer, 151–152
- lifetimes, 153
- memory address, 150
- pointer formatter (`{:p}`), 150
- and `println!` macro, 152

`RegisterTemplate`, 631–633

release version, 34, 35

remainder expression, 65

remainder operator (%), 64–65, 89

rendering templates, 538–543

- from routes, 624–625

repeat expression syntax, 202

`replace()` function, 157, 158

`replace_in_file()` function, 474–475

`replacen()` method, 236

representational state transfer (REST), 594–595
reserved words/keywords, 9, 41, 110
REST API, 571, 593
  vs HTML web server, 595
`Result` enum, 93, 188–190, 331, 364–365, 427–428
`result.is_err()` method, 189
`result.is_none()` method, 183
`result.is_ok()` method, 189
`result.is_some()` method, 183
`ResultString, io:Error` return type, 193
`return` keyword, 115
return values
  `add_numbers()` function, 115
  example, 115
  expression-based return, 115
  function signature, 115
  practical use case for, 116
  thin arrow (`>`–greater than symbol) return type, 114
  unit type return, 116
revised signature, 349
`Rng` trait, 297
`.rs` file extension, 26, 432
rules and words, human language, 9
rules and words, programming language, 9
`running_total` variable, 65
Rust, 361, 465, 505. *See also specific entries*
  advantages of, 13, 14
  vs C, 13, 14
  vs C++, 13, 14
  code cleaning service, 439
    Clippy, 441–443
    `rustfmt`, 439–441
  code debugging, 433
    VS Code user, 435–437
    `dbg!` macro, 434–435
    `println!` macro, 438–439
  code documentation, 443
    adding comments, 443–445
    building your program, 446
    running documentation examples as tests, 445–446
  code testing, 422–423
    assertion macros, 427
    code panics, 428
    integration tests, 432–433
    library, 431
    module, 429–431
    returns `Result`, 427–428
    writing, 423–426
  command-line tools, 487
    multicommand file organizer, 496–501

progress bars implementation, 487, 494–496
with subcommands, 488–493
compiling, 32–36
error handling
pattern matching with `match` keyword, 365–366
problems with, 372–373
`Result` type, 364–365
error-related helper methods
`unwrap()` and `expect()`, 366–367
`unwrap_or_default()` method, 370–371
`unwrap_or_else()` method, 369–370
`unwrap_or()` method, 367–369
error types, 373–374
`Err(String)`, 377–380
file-handling techniques, 482–485
file system
fiddling, 470–471
manipulating paths, 466–470
functions in web browser, 510
building your project, 512
creating, 512–513
projects directory, 511
spinning up a local server for, 513–514
WebAssembly development, 514–515
installation
components installed, 22
on Linux, 22
on macOS, 22
verification, 22–23
on Windows, 21
JavaScript functions, 515–519
`Option`, 374–375
`ok_or()` method, 375–376
`ok_or_else()` method, 376–377
projects directory, 618
reading files, 471–472
appending text, 475–476
comma-separated values, 477–479
increment, 472–473
one fell swoop, 472
recoverable error vs unrecoverable error, 362–364
rust-powered password analyzer, 520–524
Rust-to-WASM toolchain
Node.js, 509
test project, 510
`wasm32-unknown-unknown`, 508
`wasm-pack`, 509
serializing and deserializing data, 480–482
update, 23–24
`wasm-pack` commands, 519–520

Rust *(continued)*
WebAssembly, 506–508
web servers, 571
Axum, 571, 575–579
code, 586–589
HTTP, 572–576
logging and debugging, 591–592
reading HTML from files, 582–584
return HTML responses, 581–582
routing works in Axum, 580
starting the project, 576
static files, 589–591
testing, 578–580
URL query strings, 585–586
writing handler functions, 580–581
writing files, 471–472
appending text, 475–476
comma-separated values, 477–479
safety, 474–475
rust-analyzer extension installation, 27–28
`rustc: command not found` error, 23
`rustc --version` commands, 23
Rust-flavored templates, 621–622
`rustfmt`, 439–441
Rust multitasking, 395–396
Arc, sharing ownership, 407–409
`async`/`await` keywords, 414–416
Collatz Conjecture, 416–420
message passing, 403–404
multiple producers, single consumer, 406–407
sender and receiver, 404–406
Mutex, modifying shared data safely with, 410–411
parallelism vs concurrency, 396–397
Rayon, parallelizing iterations with, 413
scoped threads, borrowing data with, 411–413
spawning threads
creating and joining, 397–400
moving data, 400–402
Rust-powered password analyzer, 520–524
Rust Rover, 26
`Rust statements`, 110
Rust-to-WASM toolchain
Node.js, 509
test project, 510
`wasm32-unknown-unknown`, 508
`wasm-pack`, 509
Rust WASM community, 508
browser functions, 518–519
project building, 512
projects directory, 511
`pub fn hello()`, 511
`wasm-bindgen` library, 511

S

`save_game()` function, 482
`say_hello()` function, 283
scalar value, 49
`scan` subcommand, 499
scope
  block scope, 99, 100, 117
  definition of, 98
  function scope, 117–118
  global scope, 119
  memory management, 100
scoped threads, 411–413
`Searchable` trait, 321, 323–324
`secret_recipe()` function, 292
`self` keyword, 167, 168
semicolon (`;`), 41
Serde, 530
`serde` crate, 480–483, 585, 596
`serde_json`, 597
serializing data, 480–482
server-side web app, 617
  authentication, 628
    session management, 629–630
    updating the home page with login status, 637
    user login, 634–636
    user logout, 636–637
    user registration, 630–634
  bookmarks
    adding, 639–641
    deletion, 643
    displaying, 637–639
    editing, 641–642
  database connection pool, 626–628
  foundation
    database design for LinkLocker, 619–620
    database initialization, 620–621
    project structure, 618–619
  instructions for extending LinkLocker app, 643
  static file server, 625–626
  templates
    layout template, 622–624
    rendering templates from routes, 624–625
    Rust-flavored templates, 621–622
  session management, 619, 629–630
`SessionManagerLayer` middleware, 629
sharing. *See* immutable borrowing
shell, command line, 18
`show_menu()` function, 124, 129
Sieve of Eratosthenes (mathematical algorithm), 222

signed integer data type, 51
single arguments, function
  example of, 111
  function set up, 112
  output, 112
single-line comments (//), 46
single-line syntax, 78
single quotation marks ('), 55, 228
sleep apnea, 361
slice.replace(from, to) method, 157
slices
  creation of, 207–208
  definition of, 206
  to make functions, 210
  mutable slice
    data modification, 208–209
    dereferencing, 209–210
slug, definition of, 246
slugifying a string project
  appending hyphen to slug, 247–248
  avoiding consecutive hyphens addition, 248
  checking for ASCII alphanumeric character, 249
  checking for case of characters, 250
  checking for common punctuation characters, 251
  checking for common separators, 250
  checking for whitespace character, 249
  example, 246
  main() function, 252
  prev_hyphen Boolean, 248, 249
  process steps, 247
  push_hyphen() function, 248
  slugify() function, 247, 248, 251, 252
  substitutions of accented characters, 248
  transliterate() function, 248, 251
  trimmed.chars() method, 249
smart pointer, 383, 408, 478–479, 559, 562
sort_by_key() method, 257–258
sort_by() method, 258–259, 275
spawning threads, 403
  creating and joining, 397–400
  moving data, 400–402
speak() method, 313–315
Speak trait, 313
split(|c: char| !c.is_alphanumeric() && c != ’’’) method, 273

splitting, string
on substring, 246
on whitespace, 244–245
`split_whitespace()` method, 157, 244–245
`SqlitePool`, 626–628
SQLx, 609–614, 627
`sqlx-cli` command-line tool, 620
standard input (`stdin`), 193, 451, 457–458
standard library
definition of, 57
modules, 58
prelude, 58
standard output (`stdout`), 46–49, 101, 306, 382, 451
`statements-if-false` statement, 79, 81
`statements-if-true` statement, 79, 81
static file server, 625–626
`static` keyword, 356–357
`stats` subcommand, 501
`std::f32::consts::PI` constant, 58
`stdin()` function, 101
`std::io` module, 125, 129, 171
`STOPWORDS` constant, 272–273
`string``.clear()` method, 236
`string.contains(substring)` method, 242
`string.ends_with(substring)` method, 242
`string.find(substring)` method, 242
string literals, 55–56, 138, 228–229, 272
lifetime, 229
`&'static str`, 229
storage, 230
`&str` type, 229
`string.make_ascii_lowercase()` method, 244
`string.make_ascii_uppercase()` method, 244
`string.pop()` method, 236
`string``.push(`*`char`*`)` method, 233
`string``.push_str(`*`slice`*`)` method, 234
`(&str)` string slice type, 145, 151, 156–158, 170, 227, 229
adding to end of string, 234
concatenation operation, 235
deref coercion, 239–240
`get()` method, 238
string slicing with range syntax, 237–238
usage rules, 227–228

`string.split(substring)` method, 246
`string.split_whitespace()` method, 244–245
`string.starts_with(substring)` method, 242
`strings_you_bow` variable, 245
`string.to_lowercase()` method, 243
`string.to_uppercase()` method, 243
`string.trim_end()` method, 241
`string.trim_start()` method, 241
`String` type, 138–140, 142–145, 150, 157–159, 161, 170, 182, 183, 185, 226, 328–330
adding character to end of string, 233
adding `&str` slice to end of string, 234
case-conversion methods, 243–244
`chars()`, 232–233
clearing a string, 236
concatenation operation, 234–235
conversion to `&str`, 231
creation, 230–231
`from(str)` method, 231
`is_empty()` method, 233
`len()` method, 232
`new()` method, 230
replacing text in, 236
searching and testing, 242–243
splitting
on substring, 246
on whitespace, 244–245
`str.to_string()` method, 231
trimming whitespace from, 240–241
usage rules, 227–228
`with_capacity(bytes)` method, 231
structs, 306, 352–353. *See also* generics
adding trait to, 309
associated functions, 168–169
`Book` struct, 161, 162, 169
creating an instance of, 161–162
data fields, 161
definition of, 160
dot notation, 162–163
general syntax, 160–161
method, 166–167
mutable, 163–164
`mut` keyword, 163–164
PascalCase, 161
shortcuts
field init shorthand, 164–165
struct update syntax, 165–166

teaching new behaviors, 166–168
`Use &mut self`, 167
`Use &self`, 167
struct update syntax, 165–166
subcommands, command-line tools with, 487
adding, 488–490
`archive` subcommand, 500–501
`scan` subcommand, 499
`stats` subcommand, 501
working with nested, 491–493
`substring`, 182
`sum()` function, 211

T

temperature converter project
`get_menu_choice()` function, 125, 126, 129
`get_temp()` function, 126, 129
helper functions, 126
`main()` function, 127–129
`match` structure, 129
`parse()` method, 125
`print_converted_temp()` function, 127
`show_menu()` function, 124, 129
standard formulas, 127
`std::io` module, 129
`trim()` method, 125
`unwrap_or()` method, 125, 126
template inheritance pattern, 622
templates
layout template, 622–624
rendering templates from routes, 624–625
Rust-flavored templates, 621–622
terminal, command line, 18
Terminal app launching
on Linux, 19
on macOS, 18–19
on Windows, 18
testing of code, 422–423
assertion macros, 427
code panics, 428
integration tests, 432–433
library, 431
module, 429–431
returns `Result`, 427–428
writing, 423–426
`tests` directory, 432
text analyzer project, 155–158
`text` argument, 156
text types
owned string, 226
string slice (*see* `(&str)` string slice type)

`text_without_dashes` local variable, 157, 158
thin arrow (`>`–greater than symbol) return type, 114
thread, 404–414, 419
  creating and joining, 397–400
  moving data, 400–402
  spawning, 403
`thread` module, 397, 411
`tidy`, 496–498
timer macro, 391–393
`title: title` function, 164
`to_digit()` method, 185
tokenization, 272–274
`tokenize()` function, 273, 277
tokens, definition of, 272
"Tom's Obvious, Minimal Language" (TOML), 31, 477, 480, 483–485
`top_words()` function, 275, 277
`to_vec()` method, 212, 213
T programming Truth, 422
`TrafficLight` enum, 178, 179
trait bounds
  adding to generic parameter, 336
  clamp, 337
  definition of, 335
  inline bounds, 338
  multiple trait bounds, 336–337
traits
  behavioral trait, 305
  `Copy` trait, 137–138, 140, 166, 212, 219
  custom trait, 313
  `Debug` trait, 206, 310, 316
  `Default` trait, 318–319
  definition of, 305
  extensibility, 314
  implementing `Clone` trait, 308–310, 312
  implementing `Display` trait, 306–307, 310, 316–318
  implementing `PartialEq` trait, 307–308, 310, 311
  polymorphism, 313
  `Rng` trait, 297
  shared contracts, 314
  `speak()` method, 313–315
  `Speak` trait, 313
`transliterate()` function, 251
`trimmed.chars()` method, 249
`trim()` method, 103, 125
tuple
  definition of, 198
  destructuring, 200
  empty tuple, 199
  limitations, 200–201

tuple.i syntax, 198
unit tuple, 199
uses, 199
tuple.i syntax, 198
type inference, 56
type-safe language, 57

U

u8 integers, 51, 189, 327, 329
u32 type, 57, 166, 423, 432, 586
Uint8Array, 530
underscore (_) separator, 52
Unicode, 54
Unicode Transformation Format-8-bit (UTF-8), 226, 227, 230, 232, 237, 238
unit tuple, 199
unit type return, 116
unrecoverable error vs recoverable error, 362–364
unsigned integer data type, 51
unwrap()method, 366–367
unwrap_or_default() method, 370–371
unwrap_or_else() method, 369–370
unwrap_or() method/unwrap_or_default() method, 125, 126, 173, 190, 194, 367–369
unwrap_or() Option method, 186
UpdateBook, 601
UpperCamelCase. See PascalCase
URL query strings, 585–586
use keyword, 59, 290, 293–294, 297
user account directory (~), 28
user_interface module, 285, 286, 288–290, 294
user login, 634–636
user logout, 636–637
user registration, 630–634
usize type, 51
utils.rs file, 299

V

variable-length encoding, 226
variables, 61, 135, 141, 144. *See also specific variables*
choice variable, 189
declarations, 41, 279
definition of, 40
immutable variable, 44, 56
memory locations, 42–43
mutable variable, 45
naming rules, 41, 42
question variable, 183
scope of, 98

variable shadowing, 103
vec! macro, 213, 382–383
vector (Vec<T>), 170, 529
 adding and removing elements, 214–216
 advantages of, 214
 creation
  from array, 212–213
  new vector, 211
  using vec! macro, 213
 definition of, 211
 elements, accessing of, 213–214
 empty vector, 211
 indexing operator, 213–214
 iteration, 216–217
 mutable vector, 214–215
 numeric, 530
vector.get() method, 214
version
 1.89.0, 23–24
 1.90, 24
v.insert(value, index) method, 215
v.is_empty() method, 215
Visual Studio, 21
Visual Studio Code
 compiling and running from, 35
 installation, 27–28
v.push(value) method, 215
v.remove(index) method, 215
VS Code user, 424–426, 433, 435–437

W

walkdir, 497–501
WASM. *See* WebAssembly
wasm32-unknown-unknown, 508
wasm-bindgen-futures, 539, 541
wasm-bindgen library, 507, 526–527
wasm-pack, 507, 509
 commands, 519–520
 installation, 509
WebAssembly (WASM), 506–508
 browser integration, 520
 development, 513–515
 module size, 507
 wasm32 target, 510
 wasm-bindgen, 510
 web page creation, 512–513
web servers, 571
 Axum, 571, 575–579
 code, 586–589
 HTTP, 572–576
 logging and debugging, 591–592
 reading HTML from files, 582–584
 return HTML responses, 581–582
 routing works in Axum, 580
 starting the project, 576

static files, 589–591
testing, 578–580
URL query strings, 585–586
writing handler functions, 580–581
`web_sys`, 539
`while` loops, 86–87, 224
whitespace
definition of, 240
example, 241
output, 241
splitting, string on, 244–245
`string.trim_end()` method, 241
`string.trim()` method, 241
`string.trim_start()` method, 241
`width` variable, 99
wildcard character `(_)`, 94, 98, 102, 189
Windows
Rust installation, 21
Terminal app launching, 18
`withdraw()` method, 172, 191
`word_counts()` function, 274, 275
`word-replacer` tool, 461–464
words and rules, human language, 9
words and rules, programming language, 9
`words.iter_mut()` method, 263
Word Wrangler project
counting word frequencies, 274
`longest_word()` function, 275–276
`main()` function, 276–277
tokenization, 272–274
`top_words()` function, 275
writing files, 471–472
appending text, 475–476
comma-separated values, 477–479
safety, 474–475

X

XOR operator, 148

Y

YAML, 480, 595

Z

zagging, 422
zero-copy parser approach, 357–360
`zero_out()` function, 209
zigging, 422