Symbols and Numbers

+ (addition operator), 38, 137–138, 492

& (ampersand), 17, 492

&str (string slice type), 75–78

<> (angle brackets), 494–495

for specifying lifetime parameters, 191

for specifying type parameters, 130, 172, 173

-> (arrow), 46–47, 492

\* (asterisk), 492

dereference operator, 68, 311–315, 416

glob operator, 124–125

multiplication operator, 38

\*const T, 415–417, 492

\*mut T, 415–417, 492

@ (at operator), 410–411, 493

: (colon), 492–493, 495

for trait bounds, 183

for struct fields, 82

{} (curly brackets), 496

for function bodies, 6, 15

as placeholders in the println! macro, 18

scope creation, 45, 71

. (dot), 492

for method syntax, 91

for struct field access, 82

for tuple element access, 40

:: (double colon), 494

for associated functions, 94

for enum variants, 96

for namespacing, 111

" (double quote), 39, 493

/ (division operator), 38, 492

- (hyphen)

for negation, 492

for subtraction, 38, 492

+ (multiple trait bound syntax), 183, 492

! (never type), 443–444, 494

() (parentheses), 496

for function parameters, 6, 15

for tuples, 39–40

% (remainder operator), 38, 492

? (question mark operator), 159–161, 493

?Sized, 446

; (semicolon), 6, 493

' (single quote), 493–494

for characters, 39

for lifetime parameter names, 190

'static lifetime, 198–199, 421, 429, 430, 465

[] (square brackets), 496

for array creation, 41

for element access, 41, 131–132

\_ (underscore), 494

as a catchall pattern, 28, 106–107, 403–405

as a visual separator in integer literals, 37

| (vertical pipe), 493–494

in closure definitions, 261

in patterns, 398

1:1 threading model, 343

A

ABI (application binary interface), 420

abort, 150

addition

of custom types, 432–434

of number types, 38

of strings, 137–138

addition operator (+), 38, 137–138, 492

ahead-of-time compiled, 7

ampersand (&), 17, 492

angle brackets (<>), 494–495

for specifying lifetime parameters, 191

for specifying type parameters, 130, 172, 173

API (Application Programming Interface), iv, 4

application binary interface (ABI), 420

Application Programming Interface (API), iv, 4

Arc<T> type, 361–362, 473–474

arguments, 43

arms

in if expressions, 49

in match expressions, 24, 103

array data type, 40–42

invalid element access, 41–42

iterating over elements of, 54–55

slices of, 78–79

arrow (->), 46–47, 492

as\_bytes method, 73–74

assert\_eq! macro, 208–210

assert! macro, 205–208

assert\_ne! macro, 210

associated function, 16, 93–94

associated types, 431–432

associative array. See HashMap<K, V> type

asterisk (\*), 492

dereference operator, 68, 311–315, 416

glob operator, 124–125

multiplication operator, 38

at operator (@), 410–411, 493

atomically reference counted, 361–362

automatic dereferencing, 92

automatic referencing, 92

B

backtrace, 151–153

binary crate, 8, 19, 233

binary target, 302

blanket implementations, 186

blocking, 345

Boolean data type, 39, 50

borrow checker, 188–189, 190

borrowing, 68–73

Box<T> type, 306–11

break keyword, 27–28

buffer overread, 151

Build Tools for Visual Studio, 3

byte literal syntax, 37, 74

C

Cargo, ii, 7–11

commands

build, 9

check, 10

doc, 22, 287–288

install, 302–303

login, 294

new, 8, 14

publish, 294–296

run, 10, 299

test, 202–205, 215–220, 289, 301–302

update, 21

yank, 296

extending with custom commands, 303

workspaces, 297–302

Cargo.toml

dependencies section, 19

package section, 8, 294–295

profile section, 286–287

updating crate versions in, 21

Cargo.lock, 9, 20–21

carriage return, 454

cfg (configuration) attribute, 221

channels, 349–355, 470–474

character data type, 39

client, 450

clone method

deep copy creation, 65

trade-offs of, 236

Clone trait, 499–500

closed channel, 350

closures, 258–270

capturing the environment with, 268–270, 274–275

returning, 448

running in threads, 344

type inference in, 263–264

cmp method, 23

coherence, 180

collections, 129–147

collect method, 143, 229

colon (:), 492–493

for trait bounds, 183

for struct fields, 82

command line arguments, accepting, 228–231

command line notation, 2

comments, 48

documentation, 287–290

compiler-driven development, 462

compiling

with cargo, 7–11

in release mode, 10–11

with rustc, 6–7

compound data types, 39–42

concurrency, 341–364

concurrent programming, 341

configuration (cfg) attribute, 221

connection, 451–54

cons list, 308–11

constants, 34

vs. static variables, 421–22

vs. variables, 34

constructor, 319

consume, 272–273

consuming adaptors, 272–273

continue keyword, 28–29

contracts, 163

control flow, 48–55

Copy trait, 5, 499–500

ctrl-c, 27, 53, 452, 479

crate, 9

binary vs. library, 8, 19

license of, 295

publishing, 294–296

publishing a new version of, 296

updating versions, 21

using as a dependency, 21–22

yanking, 296

crates.io, 287–96

documentation comments, 287–288

publishing to, 295–296

removing from, 296

setting up an account on, 294

CRLF sequence, 454

curly brackets ({}), 496

creating new scopes with, 45, 71

for function bodies, 6, 15

as placeholders in the println! macro, 18

D

dangling pointer, 72

dangling reference, 72–73, 187–189, 193–194

data race, 70–71, 422

data types, 36–42

annotation of, 25, 36

compound, 39–42

scalar, 36–39

deadlock, 343, 362, 484

Debug trait, 89–90, 498

declarative macros, 502–504

deep copy, 64–65

Default trait, 500

default type parameters, 432–434

dependency, 7, 19

dependencies section in Cargo.toml, 9, 19

deref coercion, 138, 315–317

Deref trait, 311–317, 440

DerefMut trait, 316–317

derive annotation, 88–90, 497–500

destructor, 319

destructuring

of enums, 400–401

of references, 402

of structs, 399–400

of tuples, 40

Dickinson, Emily, 231

Dijkstra, Edsger W., 201

Display trait, 89

diverging functions, 443

division, 38

division operator (/), 38, 492

doc tests, 289

documentation

offline for Rust, 4

tests, 289

viewing a crate’s, 22

writing, 287–290

documentation comments, 287–290, 467

as tests, 289

dot (.), 492

for method syntax, 91

for struct field access, 82

for tuple element access, 40

double colon (::), 494

for associated functions, 94

for enum variants, 96

for namespacing, 111

double free error, 63, 319

double quote ("), 39, 493

Doyle, Sir Arthur Conan, 281

drop function, 62

Drop trait, 65, 317–320, 479–481

DST (dynamically sized type), 445–446

duck typing, 373

dynamically sized type (DST), 445–446

dynamic dispatch, 374

E

else if expression, 50–51

else keyword, 49

empty type. See never type

encapsulation, 366–368

entry method, 145–147

Entry type, 145–147

enumerate method, 74

enums, 95–108

defining, 96

instantiating, 96

variants of, 96

environment, 269

environment variables, 249–254

eprintln! macro, 255–256

Eq trait, 498

error handling, 149–166

executable file, 6–7

executing code, 6–7

exit status code, 239–40

expect method, 17–18, 26, 157–158

expressions, 44–46. See also statements

extern crate, 21–22

extern functions, 420–421

F

fearless concurrency, 342

FFI (Foreign Function Interface), 420

field init shorthand, 83

fields, 82

files, 231–232

floating-point data types, 38

fn keyword, 15

FnMut trait, 265, 269, 447, 465

FnOnce trait, 265, 269, 447, 465

Fn trait, 265, 269, 447, 465

fn type, 446–447

Foreign Function Interface (FFI), 420

for keyword

in trait implementations, 179–180

loop, 54–55

format! macro, 138

from function

on String, 60–61, 136

on the From trait, 160

fully qualified syntax, 434–437, 447

functional programming, 257

function pointers, 446–447

functions, 42–47

arguments to, 43

bodies, statements and expressions in, 44–46

parameters of, 43–44

returning early from, 46

with return values, 46–47

with multiple return values using a tuple, 67–68

G

Gallant, Andrew, 228

Gamma, Erich, 366

garbage collector (GC), 61

GC (garbage collector), 61

generics, 167–178, 199

default types for, 432–434

in enum definitions, 174–175

in function definitions, 170–173

in method definitions, 175–177

in struct definitions, 173–174

performance of, 177–178

get method

on HashMap<K, V>, 144

on Vec<T>, 131–132

getter, 165

Git, 8, 11

global variables, 421–422

grapheme clusters, 140, 142

green threads, 343

grep, 227

guarding, 356

guessing game, 13–30

H

hash. See HashMap<K, V> type

hasher, 147

hashing function, 142, 147

hash map. See HashMap<K, V> type

HashMap<K, V> type, 142–147

entry method on, 145–147

get method on, 144

insert method on, 142–143

iterating over, 144–145

new function on, 142–143

hash table. See HashMap<K, V> type

Hash trait, 500

heap

and the stack, 58–59

allocating on, 58

Helm, Richard, 366

Hoare, Tony, 100

HTTP (Hypertext Transfer Protocol), 450, 454–456

Hypertext Transfer Protocol (HTTP), 450, 454–456

hyphen (-)

for negation, 492

for subtraction, 38, 492

I

IDE (Integrated Development Environment), ii, 4

if keyword, 48–52

if let syntax, 107–108

ignore attribute, 219–220

immutabiity, See Mutability

impl keyword

for defining methods, 91

for defining associated functions, 93–94

for implementing traits, 179–180

indexing syntax, 131–132

indirection, 310–311

inheritance, 368–369

input lifetimes, 196

input/output (io) library, 15, 231–232

installation, 1–4

instance, 82

integer data types, 36–38

numeric operations with, 38

signed, 36–37

type suffixes of, 37

unsigned, 36–37

Integrated Development Environment (IDE), ii, 4

integration tests, 222–225

interfaces. See traits

interior mutability, 323–332, 362

invalidated variable, 64

io (input/output) library, 15, 231–232

IpAddr type, 96–98

irrefutable patterns, 395–396

isize type

architecture dependent size of, 37

indexing collection with, 38

iterator adaptors, 273–274, 280–281

Iterators, 270–277

creating with iter method, 73–74

enumerate method on, 74

next method on, 271–272

performance of, 281–283

iter method, 73–74

J

Johnson, Ralph, 366

JoinHandle type, 345

K

Kay, Alan, 365

keywords, 32, 487–489

L

last in, first out, 58

lazy evaluation, 264, 270

len method, 74

let keyword, 16

library crate, 7, 8, 19, 110

license, 295

license identifier value, 295

lifetime bounds, 428–429

lifetime elision rules, 196–198

lifetimes, 187–199

annotation of, 190–195

elision of, 195–198

inferring for trait objects, 429–430

lifetime subtyping, 423–428

line feed, 454

linker, 2–3

Linux installation, 2–3

Little Book of Rust Macros, The, 504

lock, 356–359

loop keyword, 26–27, 53

M

macOS installation, 2–3

macro\_export annotation, 503

macro\_rules! macro, 502–503

macros, 501–510

declarative, 502–504

procedural, 504–509

macro\_use annotation, 502–503

main function, 5–6

mangling, 421

map. See HashMap<K, V> type

match expression, 102–107

exhaustiveness of, 106

handling comparison results with, 23–24

handling Result values with, 28–29

match guard, 156, 408–410

memoization, 264

memory leak, 332

message passing, 349–355

metaprogramming, 502

methods

defined on structs, 90–93

defined on enums, 99

method syntax, 91

M:N threading model, 343

mock object, 325–330

mod keyword, 110–112

modules, 109–127

moving to other files, 112–118

root, 118

monomorphization, 177–178

move keyword, 269–270, 347–349

moving ownership, 62–64

vs. borrowing, 68–73

with function calls, 66

with function return values, 66–68

mpsc (multiple producer, single consumer), 350, 354, 473

multiple producer, single consumer (mpsc), 350, 354, 473

multiple trait bound syntax (+), 183, 492

multiplication, 38

Mutex<T> type, 356–362, 473–474, 478–479

poisoned, 475

mut keyword

making a variable mutable with, 33

making a reference mutable with, 69–71

mutable, See mutability

mutability

of references, 69–71

of variables, 32–33

mutual exclusion, 356

N

never type (!), 443–444, 494

new function

on HashMap<K, V>, 142–143

on String, 135–136

on Vec<T>, 130

new project setup, using cargo, 14

newtype pattern, 439–441

null, 100–102

numeric operations, 38

O

object, 366, 370. See also HashMap<K, V> type

object-oriented programming (OOP), 365–387

object-safe traits, 374–375, 383

OOP (object-oriented programming), 365–387

operator overloading, 432–434

operators, 491–493

optimizations, 10–11

Option<T> enum, 100–102

Ordering type, 23

Ord trait, 499

orphan rule, 180, 439

output lifetimes, 196

ownership, 57–79

rules, 59

and functions, 66–68

P

package section in Cargo.toml, 8, 294–295

panic, 42

panic! macro, 150–153, 161–165

parallel programming, 341

parameters, 43–44

parentheses, (()), 496

for function parameters, 6, 15

for tuples, 39–40

parse method, 25

PartialEq trait, 498

PartialOrd trait, 499

PATH system variable, 2, 3, 302–303

patterns, 389–411

binding to values with, 104–105

in for loops, 392–393

in function parameters, 394

in if let syntax, 107, 390–391

in let statements, 393–394

in match expressions, 102–103, 390

in while let syntax, 392

refutable vs. irrefutable, 395–396

pointer, 305

dangling, 72

to data on the heap, 58

raw, 415–417

poisoned mutex, 475

polymorphism, 368

prelude, 15

primitive obsession, 235

println! macro, 6, 18

privacy rules, 121

private, 119

procedural macros, 504–509

process, 342

proc\_macro crate, 507

profile section in Cargo.toml, 286–287

profiles, 286–287

propagating errors, 158–161

pub keyword, 119–121

public

API, 120, 290–293

vs. private, 119

pub use, 290–293

push method, 131, 137

push\_str method, 61, 137

Q

question mark operator (?), 159–161, 493

quote crate, 507–509

R

race conditions, 70, 343

RAII (Resource Acquisition Is Initialization), 62

rand crate, 19, 21–23

random number functionality, 19, 21–23

Range type, 55

raw pointers, 415–417

Rc<T> type, 320–323, 330–339

read\_line method, 16–17

receiver, 350

recoverable errors, 149

recursive types, 308–311

re-export, 290–293

RefCell<T> type, 323–339

reference counting, 306, 320–323, 361–362

reference cycles, 332–339

references

for accessing data from multiple places, 17

and borrowing, 68–73

creating in patterns, 407–408

dangling, 72–73

dereferencing, 68mutability of, 69–71, 73

rules of, 73

refutable patterns, 395–396

registry, 20, 287–296

release mode, 10–11

release profiles, 286–287

remainder, 38

remainder operator (%), 38, 492

request line, 454

request-response protocol, 450

Resource Acquisition Is Initialization (RAII), 62

Result<T, E> type, 17, 153–161

expect method on, 17–18, 26, 157–158

vs. panic!, 161–165

type aliases for, 442–443

unwrap method on, 157–158

unwrap\_or\_else method on, 239

return keyword, 46

return values

of functions, 46–47

multiple using a tuple, 67–68

rev method, 55

ripgrep, 228, 302–303

RLS (Rust Language Server), ii

root module, 118

.rs file extension, 5

running code, 6–7

runtime, 343

Rustaceans, 3–4

rustc, 3, 5, 6–7

rustfmt, ii, 6

Rust Language Server (RLS), ii

Rustonomicon, The, 133, 339, 363

rustup, 1–4

commands

doc, 4

uninstall, 3

update, 3

S

scalar data types, 36–39, 65–66

scope, 60

SCREAMING\_SNAKE\_CASE, 421

self parameter, 90

Self keyword, 374–375

Semantic Versioning (SemVer), 19, 296

semicolon (;), 6, 493

SemVer (Semantic Versioning), 19, 296

Send trait, 362–363, 423, 465

sequence, 55

server, 450

shadowing, 25, 34–36

shallow copy, 64

shared-state concurrency, 355–362

should\_panic attribute, 212–215

signed integer types, 36–37

single quote ('), 493–494

for characters, 39

for lifetime parameter names, 190

Sized trait, 445–446, 448

slice type, 73–79

of array, 78–79

string slices, 75–78

smart pointer, 305–339

snake case, 42

Software Package Data Exchange (SPDX), 295

SPDX (Software Package Data Exchange), 295

square brackets ([]), 496

for array creation, 41

for element access, 41, 131–132

stack

and the heap, 58–59

last in, first out ordering, 58

popping off of, 58

pushing onto, 58

standard error (stderr), 254–256

standard output (stdout), 254–256

statements, 44–46. See also expressions

state objects, 376

state pattern, 376–384

statically typed, 36

static dispatch, 374

static method. See associated function

static variables, 421–422

status line, 455

stderr (standard error) , 254–256

stdin function, 16

stdout (standard output) , 254–256

stream, 451–53

stringify! macro, 509

string literal, 60storage in the binary of, 61

of string slice type, 75–78

string slice type (&str), 75–78

String type, 60–61, 135–142

as\_bytes method on, 73–74

bytes method on, 141–142

chars method on, 141

concatenation with +, 137–138

from function on, 60–61, 136

indexing into, 139–140

internal structure of, 62–63, 139–140

iterating over, 141–142

len method on, 74

new function on, 135–136

parse method on, 25

push method on, 137

push\_str method on, 61, 137

slicing, 140–141

trim method on, 25

UTF-8 encoding of, 136

Stroustrup, Bjarne, 282

structs, 81–94

defining, 81–82

field init shorthand, 83

fields, 82

instantiating, 81–82

ownership of data, 85–86

tuple, 84–85, 439–440

unit-like, 85

update syntax, 84

subtraction, 38

super keyword, 125–127

supertraits, 437–439

symbols, 493–496

syn crate, 507–508

Sync trait, 362–363, 423

T

TCP (Transmission Control Protocol), 450–452

TDD (Test-driven development), 244

test attribute, 202–203

test double, 325

Test-driven development (TDD), 244

test functions, 202–205

tests, 201–226

custom failure messages for, 210–212

filtering, 218–219

ignoring, 219–220

integration, 222–225

organizing, 220–225

of private functions, 221–222

running, 215–220

unit, 220–222

writing, 201–215

thread pool, 461–485

threads, 342–349

creating with spawn, 344, 462–463

joining, 345

pausing with sleep, 344

thunk, 442

TOML (Tom’s Obvious, Minimal Language), 8

Tom’s Obvious, Minimal Language (TOML), 8

to\_string method, 136, 186

trait bounds, 182–187, 199

conditionally implementing methods with, 185–187

fixing the largest function with, 183–185

trait objects, 369–375, 448

dynamic dispatch, 374

inferring lifetimes of, 429–430

object safety, 374–375

traits, 178–187

associated types in, 431–432

default implementations of, 181–182

defining, 178–179

implementing, 179–180

unsafe, 422–423

Transmission Control Protocol (TCP), 450–452

transmitter, 350

trim method, 25

tuple data type, 39–40, 67–68

tuple structs, 84–85, 439–440

type alias, 441–443, 474

type annotation, 25, 36

type inference, 24

type suffixes, 37

U

underscore (\_), 494

as a catchall pattern, 28, 106–107, 403–405

as a visual separator in integer literals, 37

Unicode Scalar Value, 39, 139–141

Uniform Resource Identifier (URI), 454

Uniform Resource Locator (URL), 454

unit-like structs, 85

unit tests, 220–222

unrecoverable errors, 149–153

unrolling, 283

unsafe, 414–423

functions, 417–420

superpowers, 414, 423

traits, 422–423

unsigned integer types, 36–37

unsized type. See dynamically sized type

unwinding, 150

unwrap method, 157–158

unwrap\_or\_else method, 239

URI (Uniform Resource Identifier), 454

URL (Uniform Resource Locator), 454

use keyword, 22, 123–127

user input, 15

usize type

architecture dependent size of, 37

indexing collection with, 38

UTF-8 encoding, 136, 139–140

V

variables

vs. constants, 34

mutability, 32–33

global, 421–422

shadowing, 25, 34–36

static, 421–422

storing values in, 15–16

variants, 96

vec! macro, 130

vector. See Vec<T> type.

Vec<T> type, 130–134

get method on, 131–132

iterating over, 133–134

new function on, 130

push method on, 131

vertical pipe (|), 493–494

in closure definitions, 261

in patterns, 398

Visual Studio, 3

Vlissides, John, 366

W

weak reference, 334–335

Weak<T> type, 334–339

where clause, 183

while loop, 53–54

Windows installation, 3

workspaces, 297–302

X

Y

yanking, 296

Z

zero-cost abstractions, iii, 282–283

zero-overhead, 282