

## Symbols

\$ character 250, 361  
: (colon) 113  
" (single quotes) 42, 157  
"" (double quotes) 42, 157  
[] (square brackets) 43, 59, 78, 80, 89, 125, 293, 385  
[<>] character class 285  
{ (curly brackets) 48, 52, 78–79, 113, 124–125, 293, 339  
\* (asterisk) character 108, 363  
\* (multiplication operator) 358  
. (period) character 235, 284, 353, 361  
^ (caret) character 254, 285, 297, 346  
- (hyphen character) 40  
+ operator 176  
+= sign 88–89, 203  
= (equal sign) 41, 46, 57  
>>> prompt 41, 87  
| (pipe operator) 108, 356

## A

abuse.py (Dial-a-Curse program) 150–164  
  constructing insults 162–163  
  defining adjectives and nouns 155–156  
  defining arguments 159  
  exit values 160–161  
  final program 157–159  
  formatting output 156–157

importing and seeding random module 154–155  
parser.error() function 160  
random.seed() function 161  
range() function 162  
STDERR (standard error) 160–161  
taking random samples and choices 156  
throwaway variables 162  
validating arguments 153–154  
writing program 151–157  
add1() function 145, 189–190  
affordances 403  
AI (artificial intelligence) 351  
algorithm design 207–224  
  counting 209–210  
  creating ordinal value 211–212  
  final program 216–217  
  generating verses 221–222  
  making verses 213–215, 218–221  
  printing verses 215, 222  
  verse() function 215  
  writing program 208–215  
all() function 358, 360, 365  
alpha characters 169  
anonymous function 144, 241  
any() function 358, 360  
append() function 61  
apples.py (Apples and Bananas program) 128–149  
  altering strings 130–132  
  str.replace() method 131  
  str.translate() method 131–132

defining parameters 134–135  
final program 133–134  
refactoring with tests 149  
replacing vowels 135–149  
  iterating through every character 135–136  
  list comprehensions 140–142  
  list comprehensions with functions 142–144  
  map() function 144–147  
  map() function with named functions 147  
  regular expressions 148–149  
  str.replace() method 136  
  str.translate() method 137–139  
argparse module 385–404  
  creating flag option 391–392  
  creating optional file parameter 390–391  
  creating optional numeric parameter 390  
  creating optional string parameter 390  
  creating parser 389  
  creating positional parameter 389  
  returning from get\_args() function 392  
  types of arguments 386–387  
  use cases 392–404  
    automatic help 403–404  
    file arguments 400–402  
    manually checking arguments 402–403

argparse module (*continued*)

- one or more of the same positional arguments 399–400
- restricting values using choices option 396–398
- single positional argument 393–394
- two different positional arguments 394–396
- two of the same positional arguments 398–399
- using template.py to start programs 387–388

argparse.ArgumentParser() function 389

arguments

- abuse.py 153–154, 159
- crowsnest.py 39–40, 50–51
- file 400–402
- gashlycrumb.py 123–124
- hello.py 24–25
- howler.py 102
- manually checking 402–403
- picnic.py 73
- positional
  - one or more of the same 399–400
  - single 393–394
  - two different 394–396
  - two of the same 398–399
- tictactoe.py 360–363
- types of 386–387
- wc.py 115

artificial intelligence (AI) 351

ASCII values 295–310

- breaking text 304
- cleaning words 297–298
- encoding words 303
- final program 304–305
- functools.reduce() function 302–303
- ordinal character values and ranges 298–300
- sorting 308–309
- summing and reducing 300–301
- testing 309
- word2num() function 306–307
- writing program 296–304

assert statements 60, 83

asterisk (\*) character 108, 363

## B

binary decisions 191, 198

Black tool 28–29

Boolean values 202

bottles.py (Bottles of Beer program) 178–194

- counting down 180–181, 189
- final program 187–188
- iterating through verses 191–193
- other solutions 194
- test-driven development 189–190
- verse() function 190–191
  - using 186–187
  - writing 181–182
  - writing tests for 182–186
  - writing program 179–187

breaking text 270–271, 304

## C

capture groups 232–236

caret (^) character 254, 285, 297, 346

chmod (change mode) command 20

choices option 132, 134, 360, 386, 396–398

choose() function 197–199, 202, 204

chr() function 296, 298

class, defined 42

clean() function 337–338, 348

cleaning text 297–298, 337–339, 346–347

colon (:) 113

comma-separated values files. *See* CSV (comma-separated values) files

comment lines 16

- adding shebang line 18–19
- regular expressions 242–243

conditional branching 47–48, 70–71

consonants pattern 233

copy (cp) command 33

counting down 180–181, 189

cp (copy) command 33

crowsnest.py (Crow's Nest program) 35–54

- classifying first character of words 51–52
- concatenating strings 41–42

conditional branching 47–48

defining arguments 39–40, 50–51

final program 49–50

getting individual characters of strings 43–44

main() function 51

printing results 52

REPL (Read-Evaluate-Print-Loop) 44

starting with new.py 37–38

string comparisons 45–47

string formatting 48

string methods 44–45

test-driven development 52–53

variable types 42

writing and testing little by little 38–39

writing program 49

crowsnest.py file 38

CSV (comma-separated values) files 311–330

final program 323–325

formatting output 322

formatting table 328–330

handling bad data 322–323

parsing
 

- with csv module 318–319
- with pandas.read\_csv() function 327–328

potential runtime errors 326–327

reading 313–314, 325–326

- creating function to read 320–321
- manually 315–318
- selecting exercises 321–322
- writing program 312–323

csv module 312, 315, 318–319

csv.DictReader() function 319–320, 323, 325

csvkit module 313

curly brackets (|) 48, 52, 78–79, 113, 124–125, 293, 339

## D

declarative programming 149

def (defines) 28, 181

delimited text file 312

deterministic approach 169

Dial-a-Curse program. *See* abuse.py (Dial-a-Curse program)

dict() function 78–79, 124  
 dict.get() method 80, 84, 87, 126, 327  
 dictionaries 77–82  
   accessing values 80  
   creating 78–79  
   defining parameters 85  
   dictionary  
     comprehensions 125  
   looking up items in 124–126  
   methods 81–82  
   processing items in series 86–90  
   turning for loop into  
     list comprehension 89–90  
   using for loop to build new list 89  
   using for loop to build new string 88  
   using for loop to print()  
     each character 86–88  
   using str.translate()  
     function 90  
   using for encoding 85  
 dict.items() method 81, 318  
 dict.keys() method 81  
 dict.pop() function 82  
 dict.values() method 81  
 domain-specific language (DSL) 148  
 double quotes (") 42, 157  
 double-under methods 60  
 DSL (domain-specific language) 148  
 dynamically typed language 48

## E

enumerate() function 352, 363–364  
 env command 18–19  
 equal sign (=) 46, 57  
 error checking 287–288  
   hello.py 28–29  
 executability, making programs executable 20  
 exit values 160–161  
 extend() function 62

## F

fh (file handle) 94–95  
 fh.read() method 94–95, 103, 111

fh.readline() function 315–316  
 fh.seek(0) method 95  
 fh.write() method 223  
 file arguments 400–402  
 file handle (fh) 94–95  
 file parameter (argparse) 390–391  
 files  
   choosing output file handle 104  
   printing output 104  
   reading 93–97  
   reading input from file or command line 103–104  
   reading using for loops 115–117  
   writing 97–99  
 FileType 401–402  
 filter() function 45, 147, 227, 240–241, 339–340, 348  
 find\_brackets() function 292–293  
 find\_winner() function 356–357, 359  
 flag option 31, 391–392  
 flags 387  
 Flake8 program 28  
 float type 82, 167  
 floating-point values 237  
 foo() function 143  
 foo.py program 391  
 for loops  
   building new lists with 89  
   building new strings with 88  
   printing each character with 86–88  
   reading files using 115–117  
   turning into list comprehensions 89–90  
 format\_board() function 356  
 formatting  
   CSV output 322  
   game board 363–364  
   list items 73–74  
   output 156–157  
   regular expressions 242–243  
   strings 48, 112–114  
   text tables 328–330  
 friar.py (Kentucky Friar program) 248–267  
 final program 262–263  
 fry() function  
   using 261–262  
   writing manually 264–265

  writing with regular expressions 256–261, 266  
 negated shorthand classes 254–255  
 re.split() with captured regex 255–256  
 shorthand classes 252  
 splitting text 251  
 writing program 250–262  
 fry() function  
   using 261–262  
   writing manually 264–265  
   writing with regular expressions 256–261, 266  
 f-strings 40, 49, 52, 113, 159  
 functions  
   adding type hints to definition 376  
   using 186–187  
   writing 181–182  
   writing tests for 182–186  
 functools module 302  
 functools.reduce()  
   function 302–303

## G

gashlycrumb.py (Gashlycrumb Tinies program) 118–127  
 dictionary comprehensions 125  
 dictionary lookups 126  
 final program 122–123  
 handling arguments 123–124  
 reading input file 124–125  
 writing program 119–122  
 gematria.py (Gematria program) 295–310  
   breaking text 304  
   cleaning words 297–298  
   encoding words 303  
   final program 304–305  
   functools.reduce()  
     function 302–303  
   ordinal character values and ranges 298–300  
   sorting 308–309  
   summing and reducing 300–301  
   testing 309  
 word2num() function 306–307  
 writing program 296–304

`get_args()` function 39, 49–50, 57, 71, 84–85, 100, 102, 104, 119, 121, 123, 134–135, 154, 157, 167–168, 179, 189, 200, 209, 263, 277, 305, 321, 325, 346, 355, 359, 388–389, 402  
 adding 27–29  
 defining arguments with 50–51  
 returning from 392  
`getoutput()` function 40  
 global changes 198  
 guard statement 240

## H

hashable values 161  
`hello.py` (Hello, World! program) 15–16  
 adding `get_args()` function 27–29  
 adding help messages 22–24  
 adding `main()` function 26–27  
 adding parameters 22–24  
 comment lines 16  
 error checking 28–29  
 making argument optional 24–25  
 making program executable 20  
`$PATH` variable 20–22  
 shebang line 18–19  
 style checking 28–29  
 testing 17–18, 26, 29–30  
 help messages  
 adding 22–24  
 higher-order function. *See* HOF (higher-order function)  
 HOF (higher-order function) 147, 240, 302  
`$HOME` variable 18, 21  
`howler.py` (Howler program) 92  
 defining arguments 102  
 files  
 choosing output file handle 104  
 printing output 104  
 reading 93–97  
 reading input from file or command line 103–104  
 writing 97–99  
 final program 101–102  
 low-memory version 104–106  
 writing program 99–101  
 hyphen character (-) 40

## I

if expressions 47, 52, 142, 202  
 if/elif/else statements, conditional branching with 70–71  
 if/else statements 47, 70  
 immutable strings 130, 165  
 immutable values 275  
 imperative methods 225  
 import statements 154  
 import `sys` command 101  
 in-dels (insertion-deletions) 177  
 index method 64  
 indexing 63  
 indexing lists 63–64  
 input handles 94  
`input()` function 127, 288  
 input-output (io) module 105  
 int type 82, 167  
 int values 153, 166  
`int()` function 322, 326–327  
 integer values 153, 365  
 integration tests 186, 383  
 io (input-output) module 105  
`io.StringIO()` function 105–106, 320  
 IPython 41, 59, 78  
 iterator object 68  
`itictactoe.py` (Interactive Tic-Tac-Toe program) 367–382  
 final program 377–381  
 state 381  
 tuples 369–372  
 type hints 372–373  
 adding to function definitions 376  
 type verification with Mypy 373–375  
 updating immutable structures 375–376  
 version using `TypedDict` 379–381  
 writing program 368–376

## J

joining  
 lists 70  
 strings 41  
`jump.py` (Jump the Five program) 76–91  
 dictionaries 77–82  
 accessing values 80  
 creating 78–79

defining parameters 85  
 methods 81–82  
 processing items in series 86–90  
 using for encoding 85  
 final program 84–85  
 writing program 82–84  
 Jupyter Notebook 41, 59, 78

## K

Kentucky Friar program. *See* `friar.py` (Kentucky Friar program)

## L

`l33t()` function 342–343, 345, 347, 349  
`lambda` keyword 144, 241  
 lazy functions 140, 180, 317  
`len()` function 45, 60, 63, 72, 81, 112, 116  
 lines of code (LOC) 189  
`line.split()` method 114, 116  
 linters 28  
 list comprehensions  
 replacing vowels 140–142  
 turning for loops into 89–90  
 with functions 142–144  
 list context 136  
 list variable 55  
`list()` function 59, 68, 140, 171, 180, 220, 245, 319, 363  
`list.append()` function 60, 140, 142, 202–203, 220  
`list.extend()` method 62, 217, 220  
`list.index()` method 64  
`list.insert()` method 62  
`list.pop()` method 65, 289  
`list.remove()` method 66  
`list.reverse()` method 67, 69, 214, 274  
 lists 59–70  
 adding many elements to 61–63  
 adding one element to 60–61  
 building with for loops 89  
 finding elements in 64–65  
 formatting items 73–74  
 indexing 63–64  
 iterating 111  
 joining 70  
 mutability of 69–70

lists (*continued*)  
 mutating 176–177  
 printing items 74  
 removing elements from 65–67  
 reversing 67–69  
 slicing 64  
 sorting items 67–69, 73  
 list.sort() method 67–69, 274, 308  
 LOC (lines of code) 189

## M

MAD (mutually assured destruction) doctrine 351  
 mad.py (Mad Libs program) 281–294  
 final program 289–290  
 finding placeholders 284–287  
 finding placeholders without regular expressions 291–293  
 getting values 288–289  
 halting and printing errors 287–288  
 limiting replacements 291  
 substituting text 289  
 writing program 282–289  
 main namespace 50  
 main() function 26–27, 32, 49–51, 72, 85, 104, 120, 136, 143, 158, 160, 168, 181–182, 189, 244, 270, 293, 297, 328  
 make directory (mkdir) command 21  
 make program 29  
 make test command 29, 36, 49, 56, 119, 186  
 Makefile 29  
 map() function 45, 144, 146, 179, 186–188, 192, 196, 199, 222, 262, 265, 276, 296, 301, 306, 318, 337, 340, 347, 365  
 ransom.py 204  
 replacing vowels 144–147  
 with named functions 147  
 MapReduce 205  
 match.groups() function 234, 255  
 Mispar hebrechi 295  
 mkdir (make directory) command 21

multiplication operator (\*) 358  
 mutable elements 221  
 mutating  
 game board 360–363  
 lists 69–70, 176–177  
 strings 165–177  
 calculating number of mutations 168–169  
 final program 173–174  
 mutation space 169  
 selecting characters to mutate 169–172  
 writing program 167–173  
 text 197–198  
 updating immutable structures 375–376  
 --mutations option 166–167  
 --mutations parameter 174–175  
 mutually assured destruction (MAD) doctrine 351  
 Mypy 373–375

## N

--name option 24–25, 31  
 name parameter 23  
 --name value 385  
 named options 386  
 nargs option 71, 398  
 natural language processing (NLP) 304  
 negative index numbers 43  
 new\_char() function 142–144, 147  
 newline character 98–99  
 new.py file 30, 38, 56  
 new.py, starting new program with 30–33, 37–38  
 NLP (natural language processing) 304  
 non-capturing groups 272  
 non-deterministic selection 170  
 non-greedy regex 285  
 no-op (no operation) 273  
 numeric encoding of text 295–310  
 breaking text 304  
 cleaning words 297–298  
 encoding words 303  
 final program 304–305  
 functools.reduce 302–303  
 ordinal character values and ranges 298–300

sorting 308–309  
 summing and reducing 300–301  
 testing 309  
 word2num() function 306–307  
 writing program 296–304  
 numeric parameter (argparse) 390

## O

open file handles 110, 115, 392  
 open() method 94, 104–105, 135, 215, 401  
 operating system (os) module 93  
 operator.mul function 302  
 optional arguments 23, 25  
 optional parameters 25  
 ord() function 138, 212, 296, 298–299, 301, 307  
 OrderedDict 319  
 ordinal value 138  
 os (operating system) module 93  
 os.path.basename() method 94  
 os.path.dirname() method 93  
 os.path.isfile() function 98, 103, 110, 168  
 output handles 94  
 overwritten files 98

## P

pandas.read\_csv() function 327–328  
 parallel operations 205  
 parameters  
 apples.py 134–135  
 argparse  
 file 390–391  
 numeric 390  
 positional 389  
 string 390  
 jump.py 85  
 parser.add\_argument() function 50  
 parser.error() function 151, 154, 157–158, 160, 167, 179, 187, 189, 209, 216, 283, 355, 402–403  
 pass statement 182

password.py (Secure Password Generator program) 331–350  
 cleaning text 337–339, 346–347  
 creating unique list of words 335–337  
 filtering words 340–341  
 final program 343–346  
 l33t() function 342, 347  
 processing files 347–348  
 ransom() function 347  
 sampling and creating passwords 341–342, 348–349  
 title-casing words 341  
 using sets 339  
 writing program 334–343  
 \$PATH variable 20–22  
 PCRE (Perl-Compatible Regular Expressions) 252  
 period (.) character 232, 235, 284, 353, 361  
 Perl-Compatible Regular Expressions (PCRE) 252  
 picnic.py (Picnic List program) 55–75  
 defining arguments 73  
 final program 71–72  
 lists 59–70  
   adding many elements to 61–63  
   adding one element to 60–61  
   conditional branching with if/elif/else statements 70–71  
   finding elements in 64–65  
   formatting items 73–74  
   indexing 63–64  
   joining 70  
   mutability of 69–70  
   printing items 74  
   removing elements from 65–67  
   slicing 64  
   sorting items 67–69, 73  
   starting new program 56–58  
   writing program 58–59  
 pip module 28  
 pipe operator (|) 108, 356  
 placeholders  
   finding with regular expressions 284–287

finding without regular expressions 291–293  
 positional arguments 23, 25, 32, 39, 50, 386  
   one or more of the same 399–400  
   single 393–394  
   two different 394–396  
   two of the same 398–399  
 positional parameters 25, 389  
 pprint module 124  
 pprint() function 124–125  
 pprint.pprint() function 138  
 print() function 42, 59, 74, 84, 86, 88, 98, 116, 121, 124, 140, 157, 159, 210, 222, 278, 287  
 printf() function 113  
 printing  
   characters with for loops 86–88  
   errors 287–288  
   game board 356  
   list items 74  
   output 52, 104, 215, 222  
   REPL (Read-Evaluate-Print-Loop) 44  
 product() function 302  
 pseudo-random events 150  
 PyCharm 15, 29  
 python3 18–19, 243

## R

random events  
   capitalizing text 195  
   comparing methods 204–205  
   creating new strings 198–199  
   final program 199–200  
   flipping coin 198  
   iterating through elements in sequence 200–202  
   list.append() function 202–203  
   map() function 204  
   mutating text 197–198  
   using list comprehensions 203  
   using strings instead of lists 203  
   writing function to choose letter 202  
   writing program 197–199

mutating strings 165–177  
   calculating number of mutations 168–169  
   final program 173–174  
   mutation space 169  
   selecting characters to mutate 169–172  
   using lists instead of strings 176–177  
   writing program 167–173  
 reordering middles of words 268–280  
   breaking text into lines and words 270–271  
   capturing, non-capturing, and optional groups 271–272  
   compiling regexes 272  
   final program 276–277  
   processing text 277–279  
   scrambling all words 275  
   scrambling words 273–275, 279–280  
   writing program 269–275  
 word generation 150–164  
   constructing insults 162–163  
   defining adjectives and nouns 155–156  
   defining arguments 159  
   exit values 160–161  
   final program 157–159  
   formatting output 156–157  
   importing and seeding random module 154–155  
   parser.error() function 160  
   random.seed() function 161  
   range() function 162  
   STDERR (standard error) 160–161  
   taking random samples and choices 156  
   throwaway variables 162  
   validating arguments 153–154  
   writing program 151–157  
 random module 161, 166, 313, 343, 352  
 random seeds 198  
 random.choice() function 153, 156–157, 163, 170, 174–175, 335  
 random.choices() function 156  
 random.getstate() function 273  
 random.randint() function 314, 321, 323



`random.random()` function 170  
`random.sample()` function  
 156–157, 159, 163, 171, 174,  
 176, 321, 323, 329, 341–342  
`random.seed()` function 154,  
 156–158, 161, 168, 175, 197,  
 200, 268, 273, 276, 313, 321,  
 335, 342, 345  
`random.shuffle()` function 269,  
 273–275  
`range()` function 140, 146, 153,  
 156–157, 159, 162, 180,  
 186–187, 193, 208–209, 211,  
 349  
`ransom()` function 335, 342,  
 345–347  
`ransom.py` (Ransom Note  
 program) 195  
 comparing methods 204–205  
 creating new strings 198–199  
 final program 199–200  
 flipping coin 198  
 iterating through elements in  
 sequence 200–202  
`list.append()` function 202–203  
`map()` function 204  
 mutating text 197–198  
 using list  
   comprehensions 203  
 using strings instead of  
   lists 203  
 writing function to choose  
   letter 202  
 writing program 197–199  
 Read the Fine Manual  
   (RTFM) 44  
`read()` method 94, 96, 135  
`read_csv()` function 320–321,  
 325, 327, 329  
`re.compile()` function 264, 272  
`reduce()` function 296, 302–303  
 refactoring 149  
`re.findall()` function 282, 287,  
 297  
 regular expressions 148–149  
   captured 255–256  
   compiling 272  
   `friar.py` 248–267  
   final program 262–263  
   `fry()` function 256–262,  
   264–266  
   negated shorthand  
   classes 254–255  
   `re.split()` with captured  
   regex 255–256

shorthand classes 252  
 splitting text 251  
 writing program 250–262  
`mad.py` 281–294  
   final program 289–290  
   finding placeholders  
   284–287  
   finding placeholders with-  
   out regular  
   expressions 291–293  
   getting values 288–289  
   halting and printing  
   errors 287–288  
   substituting text 289  
   substituting with regular  
   expressions 291  
   writing program 282–289  
`rhymmer.py` 225–247  
   commenting 242–243  
   creating output 238  
   creating rhyming  
   strings 244–245  
   final program 238–239  
   formatting 242–243  
   stemmer() function 228–  
   229, 240–241, 243, 245  
   truthiness 236–237  
   using 229–232  
   using capture groups 232–  
   236  
   writing `rhymmer.py` 227–238  
`re.Match` object 260  
`re.match()` function 230, 232,  
 234, 239, 250–251, 266, 272,  
 327  
`re.Match.groups()` method 232  
`Repl.it` 20, 29  
 requirements.txt file 313  
`re.search()` function 230, 232,  
 250, 252, 260, 266, 272  
`re.split()` function 250, 255–256,  
 263–264, 270  
`re.sub()` function 148, 282, 291,  
 296, 298, 305, 307, 346  
 return statement 182  
`reversed()` function 68, 180,  
 214, 220  
 reversing lists 67–69  
`rhymmer.py` (Rhym-  
 er program) 225–247  
   commenting 242–243  
   creating output 238  
   creating rhyming strings  
   244–245  
   final program 238–239

formatting 242–243  
`stemmer()` function 240–241  
   `rhymmer.py` 228–229  
   using outside `rhymmer.py` 243  
   writing without regular  
   expressions 245  
 truthiness 236–237  
 using capture groups 232–236  
 using regular  
   expressions 229–232  
   writing program 227–238  
`round()` function 168  
 round-tripping 91  
`rstrip()` method 135  
 RTFM (Read the Fine  
   Manual) 44

## S

`scramble()` function 273, 275,  
 277–278  
`scrambler.py` (Scrambler  
 program) 268–280  
   breaking text into lines and  
   words 270–271  
   capturing, non-capturing, and  
   optional groups 271–272  
   compiling regexes 272  
   final program 276–277  
   processing text 277–279  
   scrambling all words 275  
   scrambling words 273–275,  
   279–280  
   writing program 269–275  
 secure password  
   generation 331–350  
   cleaning text 337–339,  
   346–347  
   creating unique list of  
   words 335–337  
   filtering words 340–341  
   final program 343–346  
   `l33t()` function 342, 347  
   processing files 347–348  
   `ransom()` function 347  
   sampling and creating  
   passwords 341–342,  
   348–349  
   title-casing words 341  
   using sets 339  
   writing program 334–343  
 Secure Password Generator pro-  
   gram. *See* `password.py`  
   (Secure Password Genera-  
   tor program)

sep argument 86  
 set() function 339  
 set.add() function 339  
 set.intersection() function 339  
 sets 339  
 set.sort() function 349  
 shebang (#!) line 18–19  
 shorthand classes 252, 254–255  
 shuffled value 274  
 single quotes (") 42, 157  
 slice notation 43  
 slices 173  
 slicing lists 64  
 SNP (single nucleotide polymorphisms) 177  
 SNV (single nucleotide variations) 177  
 sorted() function 68–69, 174, 245, 345, 349  
 sorting  
   list items 67–69, 73  
   strings 308–309  
 spaCy library 332  
 square brackets ([]) 43, 59, 78, 80, 89, 125, 293, 385  
 state 351–366, 381  
   altering board 355  
   determining winner 356–357, 364–365  
   final program 357–365  
   mutating board 360–363  
   printing board 356  
   validating arguments 360–363  
   validating user input 355  
   writing tictactoe.py 353–357  
 static typing 48  
 STDERR (standard error) 160–161, 287, 290, 293  
 STDIN (standard in) 110  
 STDOUT (standard out) 97, 101, 108, 110, 130, 157, 160, 208, 246, 290  
   choosing output file handle 104  
   writing files 97–99  
 stemmer() function 228–229, 236, 238, 244  
 rhymer.py 228–229, 240–241  
   using outside rhymer.py 243  
   writing without regular expressions 245  
 step value 181  
 stop value 43, 180, 292

str class 42, 44–45, 60, 63, 85, 90, 96, 134, 172, 183, 197, 303, 320, 394  
 str.casefold() function 308  
 str.endswith() function 259  
 str.format() method 48, 52, 72, 112–113, 157, 363  
 string module 169  
 string parameter (argparse) 390  
 string slices 173  
 string.ascii\_letters 169  
 string.ascii\_lowercase 241  
 string.punctuation 169, 335  
 strings  
   altering 130–132  
     str.replace() method 131  
     str.translate() method 131–132  
   building with for loops 88  
   comparisons of 45–47  
   concatenating 41–42  
   creating new 198–199  
   formatting 48, 112–114  
   getting individual characters of strings 43–44  
   methods for 44–45  
   mutating 172–173, 175–176  
   using instead of lists 203  
 str.isupper() function 45  
 str.join() method 70, 89, 136, 147, 157, 163, 192, 210, 215, 222, 230, 278, 303, 349  
 str.lower() method 49  
 str.maketrans() function 132, 137  
 str.replace() method 84, 90–91, 131, 171, 175  
   altering strings 131  
   replacing vowels 136  
 str.rstrip() method 97, 103, 124, 133, 315–316  
 str.split() method 112, 155, 157, 162, 251, 270, 316, 348  
 str.splitlines() function 261, 263, 270, 276  
 str.splitlines() method 251, 262  
 str.strip() function 356  
 str.title() method 334  
 str.translate() function 90–91, 139, 345, 347  
 str.translate() method 90  
   altering strings 131–132  
   replacing vowels 137–139  
 str.upper() function 44, 49, 97

sum() function 296, 300, 302, 307  
 sys.exit() function 160, 282, 288, 329  
 sys.stderr 287  
 sys.stdout file 101–102, 208

## T

tab character 323  
 tabulate module 312–313, 320, 323, 330  
 tabulate() function 322, 325  
 TDD (test-driven development) 53, 182  
 telephone.py (Telephone program) 165–177  
   calculating number of mutations 168–169  
   final program 173–174  
   mutating strings 172–173, 175–176  
   mutation space 169  
   selecting characters to mutate 169–172  
     non-deterministic selection 170  
     randomly sampling characters 170–172  
   using lists instead of strings 176–177  
   writing program 167–173  
 template.py, starting new programs with 387–388  
 test-driven development 52–53, 189–190  
 testing  
   apples.py 149  
   gematria.py 309  
   hello.py 17–18, 26, 29–30  
   writing and testing little by little 38–39  
   writing tests for functions 182–186  
 test.py 17–18, 30  
 TextIOWrapper class 96  
 Tic-Tac-Toe program. *See* itictactoe.py (Interactive Tic-Tac-Toe program)  
 tictactoe.py (Tic-Tac-Toe program) 351–366  
   altering board 355  
   determining winner 356–357, 364–365  
   final program 357–365



tictactoe.py (Tic-Tac-Toe program) (*continued*)  
 formatting board 363–364  
 mutating board 360–363  
 printing board 356  
 validating arguments 360–363  
 validating user input 355  
 writing program 353–357

TIMTOWTDI (There Is More Than One Way To Do It) 135

title-casing 341

truthiness 236–237

tuples 369–372

twelve\_days.py (Twelve Days of Christmas program) 207–224  
 counting 209–210  
 creating ordinal value 211–212  
 final program 216–217  
 generating verses 221–222  
 making verses 213–215, 218–221  
 printing verses 215, 222  
 verse() function 215  
 writing program 208–215

two equal signs (==) 46

type hints 372–373  
 adding to function definitions 376  
 type verification with Mypy 373–375

type() function 42, 45, 60, 78, 80, 97

TypedDict 379–381

## U

unbalanced brackets 285

unit tests 183, 186, 383

unit.py file 356

usage statement 23

user input validation 355

## V

variables  
 \$PATH variable 20–22  
 throwaway 162  
 types of 42

verbose output 17

verse() function 187–189, 193, 210–211, 217, 221  
 twelve\_days.py 215  
 using 186–187  
 writing 181–182  
 writing tests for 182–186

VS Code 15, 29

## W

wc.py (Word Count program) 107–117  
 defining arguments 115  
 defining file inputs 110  
 final program 114–115  
 formatting results 112–114  
 iterating lists 111  
 reading files using for loops 115–117  
 what's being counted 111–112  
 writing program 109–114

which command 19–20

wildcard 108

wod.py (Workout of the Day program) 311–330  
 final program 323–325  
 formatting text table  
   output 322, 328–330  
 handling bad data 322–323  
 parsing delimited text files  
   with csv module 318–319  
   with pandas.read\_csv() function 327–328

potential runtime errors 326–327

reading delimited text files 313–314, 325–326  
 creating function to read 320–321  
 manually 315–318

selecting exercises 321–322  
 writing program 312–323

wod.read\_csv() function 329

Word Count program. *See* wc.py (Word Count program)

word variable 41–44, 48, 51, 116

word2num() function 303–307, 309

write() method 96, 98

writing programs 15–34, 353–357  
 abuse.py 151–157  
 bottles.py 179–187  
 crownsnest.py 49  
 friar.py 250–262  
 gashlycrumb.py 119–122  
 gematria.py 296–304  
 hello.py 15–30  
 howler.py 99–101  
 itictactoe.py 368–376  
 jump.py 82–84  
 mad.py 282–289  
 password.py 334–343  
 picnic.py 58–59, 71  
 ransom.py 197–199  
 rhymmer.py 227–238  
 scrambler.py 269–275  
 starting new program with new.py 30–33  
 starting new program with template.py 33  
 telephone.py 167–173  
 twelve\_days.py 208–215  
 wc.py 109–114  
 wod.py 312–323  
 writing and testing little by little 38–39

wt (writing text) 104

## Y

YAPF 28–29

## Z

zero-offset indexing 63

zip() function 316–317

# Tiny Python Projects

Ken Youens-Clark



**W**ho says learning to program has to be boring? The 21 activities in this book teach Python fundamentals through puzzles and games. Not only will you be entertained with every exercise, but you'll learn about text manipulation, basic algorithms, and lists and dictionaries as you go. It's the ideal way for any Python newbie to gain confidence and experience.

The projects are tiny, but the rewards are big: each chapter in **Tiny Python Projects** challenges you with a new Python program, including a password creator, a word rhymers, and a Shakespearean insult generator. As you complete these entertaining exercises, you'll graduate from a Python beginner to a confident programmer—and you'll have a good time doing it!

## What's Inside

- Write command-line Python programs
- Manipulate Python data structures
- Use and control randomness
- Write and run tests for programs and functions
- Download testing suites for each project

For readers with beginner programming skills.

**Ken Youens-Clark** is a Senior Scientific Programmer at the University of Arizona. He has an MS in Biosystems Engineering and has been programming for over 20 years.

“*Tiny Python Projects* is a gentle, amusing introduction to Python that will firm up several key concepts while occasionally making you snicker.”

—Amanda Debler  
Schaeffler Technologies

“Knowledge meets humor meets succinctness. A gem.”

—Mafinar Khan, theScore

“Learning based on doing small projects is effective, and that's why this book is perfect.”

—Marcin Sęk, e-Xim IT

“Excellent pick for those who want to improve coding skills with Python.”

—José Apablaza, Steadfast

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit  
[www.manning.com/books/tiny-python-projects](http://www.manning.com/books/tiny-python-projects)