

contents

<i>preface</i>	<i>xv</i>
<i>acknowledgments</i>	<i>xvii</i>
<i>about this book</i>	<i>xix</i>
<i>about the author</i>	<i>xxii</i>
<i>about the cover</i>	<i>xxiii</i>

Getting started: Introduction and installation guide 1

Writing command-line programs	1
Using test-driven development	4
Setting up your environment	4
Code examples	5
Getting the code	8
Installing modules	10
Code formatters	10
Code linters	11
How to start writing new programs	11
Why not Notebooks?	12
The scope of topics we'll cover	12
Why not object-oriented programming?	13
A note about the lingo	13

1	<i>How to write and test a Python program</i>	15
1.1	Creating your first program	15
1.2	Comment lines	16
1.3	Testing your program	17
1.4	Adding the #! (shebang) line	18
1.5	Making a program executable	20
1.6	Understanding \$PATH	20
	<i>Altering your \$PATH</i>	21
1.7	Adding a parameter and help	22
1.8	Making the argument optional	24
1.9	Running our tests	26
1.10	Adding the main() function	26
1.11	Adding the get_args() function	27
	<i>Checking style and errors</i>	28
1.12	Testing hello.py	29
1.13	Starting a new program with new.py	30
1.14	Using template.py as an alternative to new.py	33

2	<i>The crow's nest: Working with strings</i>	35
2.1	Getting started	36
	<i>How to use the tests</i>	36
	<i>Creating programs with new.py</i>	37
	<i>Write, test, repeat</i>	38
	<i>Defining your arguments</i>	39
	<i>Concatenating strings</i>	41
	<i>Variable types</i>	42
	<i>Getting just part of a string</i>	43
	<i>Finding help in the REPL</i>	44
	<i>String methods</i>	44
	<i>String comparisons</i>	45
	<i>Conditional branching</i>	47
	<i>String formatting</i>	48
	<i>Time to write</i>	49
2.2	Solution	49
2.3	Discussion	50
	<i>Defining the arguments with get_args()</i>	50
	<i>The main() thing</i>	51
	<i>Classifying the first character of a word</i>	51
	<i>Printing the results</i>	52
	<i>Running the test suite</i>	52
2.4	Going further	53

3	<i>Going on a picnic: Working with lists</i>	55
3.1	Starting the program	56
3.2	Writing picnic.py	58

- 3.3 Introducing lists 59
 - Adding one element to a list* 60 ▀ *Adding many elements to a list* 61 ▀ *Indexing lists* 63 ▀ *Slicing lists* 64 ▀ *Finding elements in a list* 64 ▀ *Removing elements from a list* 65
 - Sorting and reversing a list* 67 ▀ *Lists are mutable* 69
 - Joining a list* 70
- 3.4 Conditional branching with if/elif/else 70
 - Time to write* 71
- 3.5 Solution 71
- 3.6 Discussion 73
 - Defining the arguments* 73 ▀ *Assigning and sorting the items* 73
 - Formatting the items* 73 ▀ *Printing the items* 74
- 3.7 Going further 75

4 *Jump the Five: Working with dictionaries* 76

- 4.1 Dictionaries 77
 - Creating a dictionary* 78 ▀ *Accessing dictionary values* 80
 - Other dictionary methods* 81
- 4.2 Writing jump.py 82
- 4.3 Solution 84
- 4.4 Discussion 85
 - Defining the parameters* 85 ▀ *Using a dict for encoding* 85
 - Various ways to process items in a series* 86 ▀ *(Not) using str.replace()* 90
- 4.5 Going further 91

5 *Howler: Working with files and STDOUT* 92

- 5.1 Reading files 93
- 5.2 Writing files 97
- 5.3 Writing howler.py 99
- 5.4 Solution 101
- 5.5 Discussion 102
 - Defining the arguments* 102 ▀ *Reading input from a file or the command line* 103 ▀ *Choosing the output file handle* 104
 - Printing the output* 104 ▀ *A low-memory version* 104
- 5.6 Going further 106

6 *Words count: Reading files and STDIN, iterating lists, formatting strings* 107

6.1 Writing wc.py 109

Defining file inputs 110 ■ *Iterating lists* 111 ■ *What you're counting* 111 ■ *Formatting your results* 112

6.2 Solution 114

6.3 Discussion 115

Defining the arguments 115 ■ *Reading a file using a for loop* 115

6.4 Going further 117

7 *Gashlycrumb: Looking items up in a dictionary* 118

7.1 Writing gashlycrumb.py 119

7.2 Solution 122

7.3 Discussion 123

Handling the arguments 123 ■ *Reading the input file* 124
Using a dictionary comprehension 125 ■ *Dictionary lookups* 126

7.4 Going further 126

8 *Apples and Bananas: Find and replace* 128

8.1 Altering strings 130

Using the str.replace() method 131 ■ *Using str.translate()* 131
Other ways to mutate strings 132

8.2 Solution 133

8.3 Discussion 134

Defining the parameters 134 ■ *Eight ways to replace the vowels* 135

8.4 Refactoring with tests 149

8.5 Going further 149

9 *Dial-a-Curse: Generating random insults from lists of words* 150

9.1 Writing abuse.py 151

Validating arguments 153 ■ *Importing and seeding the random module* 154 ■ *Defining the adjectives and nouns* 155 ■ *Taking random samples and choices* 156 ■ *Formatting the output* 156

9.2 Solution 157

9.3 Discussion 159

Defining the arguments 159 ■ *Using parser.error()* 160
Program exit values and STDERR 160 ■ *Controlling randomness
 with random.seed()* 161 ■ *Iterating with range() and using
 throwaway variables* 162 ■ *Constructing the insults* 162

9.4 Going further 163

10 Telephone: Randomly mutating strings 165

10.1 Writing telephone.py 167

Calculating the number of mutations 168 ■ *The mutation
 space* 169 ■ *Selecting the characters to mutate* 169
Mutating a string 172 ■ *Time to write* 173

10.2 Solution 173

10.3 Discussion 175

Mutating a string 175 ■ *Using a list instead of a str* 176

10.4 Going further 177

11 Bottles of Beer Song: Writing and testing functions 178

11.1 Writing bottles.py 179

Counting down 180 ■ *Writing a function* 181 ■ *Writing
 a test for verse()* 182 ■ *Using the verse() function* 186

11.2 Solution 187

11.3 Discussion 189

Counting down 189 ■ *Test-driven development* 189
The verse() function 190 ■ *Iterating through the verses* 191
1,500 other solutions 194

11.4 Going further 194

12 Ransom: Randomly capitalizing text 195

12.1 Writing ransom.py 197

Mutating the text 197 ■ *Flipping a coin* 198 ■ *Creating
 a new string* 198

12.2 Solution 199

12.3 Discussion 200

Iterating through elements in a sequence 200 ■ *Writing a function
 to choose the letter* 202 ■ *Another way to write list.append()* 202
Using a str instead of a list 203 ■ *Using a list comprehension* 203
Using a map() function 204

12.4 Comparing methods 204

12.5 Going further 205

13 *Twelve Days of Christmas: Algorithm design* 207

13.1 Writing `twelve_days.py` 208

Counting 209 ■ *Creating the ordinal value* 211 ■ *Making the verses* 213 ■ *Using the `verse()` function* 215 ■ *Printing* 215
Time to write 215

13.2 Solution 216

13.3 Discussion 218

Making one verse 218 ■ *Generating the verses* 221
Printing the verses 222

13.4 Going further 223

14 *Rhymer: Using regular expressions to create rhyming words* 225

14.1 Writing `rhymer.py` 227

Breaking a word 228 ■ *Using regular expressions* 229
Using capture groups 232 ■ *Truthiness* 236 ■ *Creating the output* 238

14.2 Solution 238

14.3 Discussion 240

Stemming a word 240 ■ *Formatting and commenting the regular expression* 242 ■ *Using the `stemmer()` function outside your program* 243 ■ *Creating rhyming strings* 244 ■ *Writing `stemmer()` without regular expressions* 245

14.4 Going further 246

15 *The Kentucky Friar: More regular expressions* 248

15.1 Writing `friar.py` 250

Splitting text using regular expressions 251 ■ *Shorthand classes* 252 ■ *Negated shorthand classes* 254 ■ *Using `re.split()` with a captured regex* 255 ■ *Writing the `fry()` function* 256
Using the `fry()` function 261

15.2 Solution 262

15.3 Discussion 263

Writing the `fry()` function manually 264 ■ *Writing the `fry()` function with regular expressions* 266

15.4 Going further 266

16 *The Scrambler: Randomly reordering the middles of words* 268

16.1 Writing `scrambler.py` 269

Breaking the text into lines and words 270 ■ *Capturing, non-capturing, and optional groups* 272 ■ *Compiling a regex* 272
Scrambling a word 273 ■ *Scrambling all the words* 275

16.2 Solution 276

16.3 Discussion 277

Processing the text 277 ■ *Scrambling a word* 279

16.4 Going further 280

17 *Mad Libs: Using regular expressions* 281

17.1 Writing `mad.py` 282

Using regular expressions to find the pointy bits 284
Halting and printing errors 287 ■ *Getting the values* 288
Substituting the text 289

17.2 Solution 289

17.3 Discussion 290

Substituting with regular expressions 291 ■ *Finding the placeholders without regular expressions* 291

17.4 Going further 293

18 *Gematria: Numeric encoding of text using ASCII values* 295

18.1 Writing `gematria.py` 296

Cleaning a word 297 ■ *Ordinal character values and ranges* 298 ■ *Summing and reducing* 300 ■ *Using `functools.reduce`* 302 ■ *Encoding the words* 303
Breaking the text 304

18.2 Solution 304

18.3 Discussion 305

Writing `word2num()` 306 ■ *Sorting* 308 ■ *Testing* 309

18.4 Going further 309

19 *Workout of the Day: Parsing CSV files, creating text table output* 311

19.1 Writing `wod.py` 312

Reading delimited text files 313 ■ *Manually reading a CSV file* 315 ■ *Parsing with the `csv` module* 318 ■ *Creating a*

function to read a CSV file 320 ■ *Selecting the exercises* 321
Formatting the output 322 ■ *Handling bad data* 322
Time to write 323

19.2 Solution 323

19.3 Discussion 325

Reading a CSV file 325 ■ *Potential runtime errors* 326
Using pandas.read_csv() to parse the file 327 ■ *Formatting the table* 328

19.4 Going further 330

20 *Password strength: Generating a secure and memorable password* 331

20.1 Writing password.py 334

Creating a unique list of words 335 ■ *Cleaning the text* 337
Using a set 339 ■ *Filtering the words* 340 ■ *Titlecasing the words* 341
 ■ *Sampling and making a password* 341
l33t-ify 342 ■ *Putting it all together* 343

20.2 Solution 343

20.3 Discussion 346

Cleaning the text 346 ■ *A king's ransom* 347 ■ *How to l33t()* 347
 ■ *Processing the files* 347 ■ *Sampling and creating the passwords* 348

20.4 Going further 349

21 *Tic-Tac-Toe: Exploring state* 351

21.1 Writing tictactoe.py 353

Validating user input 355 ■ *Altering the board* 355
Printing the board 356 ■ *Determining a winner* 356
Solution 357 ■ *Validating the arguments and mutating the board* 360
 ■ *Formatting the board* 363 ■ *Finding the winner* 364

21.2 Going further 366

22 *Tic-Tac-Toe redux: An interactive version with type hints* 367

22.1 Writing itictactoe.py 368

Tuple talk 369 ■ *Named tuples* 371 ■ *Adding type hints* 372
 ■ *Type verification with Mypy* 373 ■ *Updating*

	<i>immutable structures</i>	375	■	<i>Adding type hints to function definitions</i>	376
22.2	Solution	377			
	<i>A version using TypedDict</i>	379	■	<i>Thinking about state</i>	381
22.3	Going further	381			
	<i>Epilogue</i>	383			
appendix	<i>Using argparse</i>	385			
	<i>index</i>	405			

