

CONTENTS

Contents

iv

I Saving time and aggravation

0	Simplifying Repetitive Calculations	5
0.0	<i>Problem</i>	6
0.1	<i>Solution</i>	7
0.1.0	Giving your computer instructions using Python commands	7
0.1.1	Having Python do basic math by evaluating mathematical expressions	8
0.1.2	Accessing capabilities beyond basic math using functions	9
0.1.3	Clarifying how your code works with comments	12
0.1.4	Temporarily storing data using variables	13
0.1.5	Accessing a history of code you have executed in the console	14
0.1.6	Expressing data as readable text using strings	16
0.1.7	Avoiding repetitive typing by making your own functions	18
0.2	<i>Exercises</i>	20
1	Automating Repetitive Calculations	25
1.0	<i>Problem</i>	26
1.1	<i>Solution</i>	26
1.1.0	Importing libraries to add capabilities to Python	27
1.1.1	Operating on collections of data efficiently with Numpy arrays	27
1.1.2	Interpreting Python error messages to fix mistakes	29
1.1.3	Structuring data using arrays of arrays	30
1.1.4	Generating arrays of all ones or zeros Numpy	31
1.1.5	Generating an array of equally spaced numbers using Numpy	32
1.1.6	Planning a multi-step solution to a complex problem	33
1.1.7	Writing scripts to execute complete solutions in one click	34
1.1.8	Keeping track of changes to your code using versioning practices	35
1.2	<i>Code</i>	36
1.2.0	Line-by-line explanation	38
1.3	<i>Exercises</i>	39
2	Automating the Plotting of Data	43
2.0	<i>Problem</i>	44
2.1	<i>Solution</i>	44

2.1.0	Planning	44
2.1.1	Importing subsets of libraries using <code>from</code>	46
2.1.2	Performing multiple assignment operations in one line by unpacking	47
2.1.3	Representing answers to yes/no questions using booleans	47
2.1.4	Exploiting built-in methods to manipulate data stored in objects	47
2.1.5	Describing the location of files or folders on your computer using <code>Pathlib</code>	48
2.1.6	Making paths convenient and portable by specifying relative paths	50
2.1.7	Adding <code>codechembook</code> to your Python installation	51
2.1.8	Locating files and folders graphically with <code>codechembook.quickTools.quickOpenFilename()</code>	51
2.1.9	Accessing data in structured text files using <code>Numpy</code>	52
2.1.10	Making appealing and interactive plots using <code>Plotly</code>	53
2.1.11	Outlining your solution before coding using comments	54
2.2	<i>Code</i>	54
2.2.0	Line-by-line explanation	56
2.3	<i>Exercises</i>	58
3	Plotting Multiple Data Sets	63
3.0	<i>Problem</i>	64
3.1	<i>Solution</i>	64
3.1.0	Planning	64
3.1.1	Making ordered collections of data more flexible using list objects	64
3.1.2	Combining lists by concatenation	65
3.1.3	Adding items to lists using the <code>append()</code> method	65
3.1.4	Determining the number of items in a collection using the <code>len()</code> function	66
3.1.5	Accessing items in lists, arrays, or strings by indexing	66
3.1.6	Access subsets of lists, arrays, or strings by slicing	67
3.1.7	Ordering lists and arrays by sorting	68
3.1.8	Obtaining a list of paths using <code>codechembook.quickTools.quickOpenFileNames()</code>	68
3.1.9	Creating collections of objects with customized organization using dictionaries	69
3.1.10	Accessing information about objects using attributes	69
3.1.11	Performing repetitive operations using <code>for</code> loops	70
3.2	<i>Code</i>	71
3.2.0	Line-by-line explanation	73
3.3	<i>Exercises</i>	74
4	Generating Molecular Structures and Annotating Plots	79
4.0	<i>Problem</i>	80
4.1	<i>Solution</i>	80
4.1.0	Planning	80
4.1.1	Accessing items from dictionaries using keys	80
4.1.2	Describing chemical structures using SMILES	81
4.1.3	Drawing molecules with <code>RDKit</code>	81
4.1.4	Using tuples to create unchangeable lists	82

4.1.5	Specifying colors using the <code>rgb</code> standard	83
4.1.6	Decoding data from binary files	83
4.1.7	Adding images to Plotly figures	84
4.1.8	Adding annotations to Plotly figures	84
4.1.9	Formatting text in Plotly using HTML tags	84
4.1.10	Producing HTML-formatted text quickly using <code>codechembook.symbols</code>	85
4.1.11	Generating sequences of numbers using <code>range()</code>	85
4.1.12	Synchronizing iteration through multiple collections using <code>zip()</code>	85
4.1.13	Specifying the result of function evaluation using the <code>return</code> keyword	86
4.1.14	Describing how to use your functions in docstrings	87
4.2	<i>Code</i>	88
4.2.0	Line-by-line explanation	90
4.3	<i>Exercises</i>	92

II Modeling and fitting data

5	Fitting Equations to Data	97
5.0	<i>Problem</i>	98
5.1	<i>Solution</i>	98
5.1.0	Planning	98
5.1.1	Prompting the user for information using <code>input()</code>	99
5.1.2	Turning numeral characters into numbers	99
5.1.3	Finding the element closest to a given number in a Numpy array	100
5.1.4	Inserting calculation results into strings using f-strings	101
5.1.5	Formatting the representation of numbers in strings	101
5.1.6	Generating symbols that aren't on your keyboard using <code>codechembook.symbols</code>	102
5.1.7	Finding paths to files in folders using <code>glob()</code>	103
5.1.8	Dividing a string into multiple substrings with <code>split()</code>	103
5.1.9	Fitting models to data with <code>lmfit</code>	104
5.1.10	Analyzing fit results in <code>lmfit</code>	104
5.1.11	Building figures with multiple subplots in Plotly	105
5.2	<i>Code</i>	105
5.2.0	Line-by-line explanation	107
5.3	<i>Exercises</i>	110
6	Fitting Data with Multiple Components	113
6.0	<i>Problem</i>	114
6.1	<i>Solution</i>	114
6.1.0	Planning	114
6.1.1	Getting yes/no answers to questions posed by comparison statements	114
6.1.2	Selecting a subset of a Numpy array using comparison expressions	115
6.1.3	Selecting array subsets based on multiple conditions using <code>and</code> and <code>or</code>	116

6.1.4	Accessing the number of times a loop has iterated using <code>enumerate()</code>	117
6.1.5	Visualizing fit results using <code>codechembook.quickPlots.plotFit()</code>	118
6.1.6	Learning to use libraries by reading documentation	118
6.1.7	Combining <code>lmfit</code> models	118
6.2	<i>Code</i>	121
6.2.0	Line-by-line explanation	123
6.3	<i>Exercises</i>	125
7	Fitting Simulations to Data	129
7.0	<i>Problem</i>	130
7.1	<i>Solution</i>	131
7.1.0	Planning	131
7.1.1	Automatically running different code under different conditions using <code>if-then-else</code> statements	132
7.1.2	Setting the working directory using <code>os.chdir()</code>	133
7.1.3	Importing any Python code from anywhere on your computer	134
7.1.4	Handling errors without crashing using <code>try-except-finally</code>	135
7.1.5	Plotting in a single line of code using <code>codechembook.quickPlots.quickScatter()</code>	135
7.2	<i>Code</i>	136
7.2.0	Line-by-line explanation	139
7.2.1	Line-by-line explanation	142
7.3	<i>Exercises</i>	143
8	Simulating Chemical Kinetics	145
8.0	<i>Problem</i>	146
8.1	<i>Solution</i>	147
8.1.0	Simulating the kinetics of one elementary reaction using <code>scipy.solve_ivp()</code>	147
8.1.1	Planning	148
8.1.2	Validating coding approaches with simple test cases	149
8.1.3	Line-by-line explanation	150
8.1.4	Simulating multi-step reaction mechanisms using <code>scipy.solve_ivp()</code>	152
8.2	<i>Code</i>	152
8.2.1	Line-by-line explanation	153
8.3	<i>Exercises</i>	154

III Analyzing Numerical Data

9	Integrating Data Numerically	159
9.0	<i>Problem</i>	160
9.1	<i>Background</i>	160
9.2	<i>Solution</i>	161
9.2.0	Integrating data using <code>scipy.integrate</code>	161
9.2.1	Generating test data to evaluate the accuracy of numerical analysis	162

9.2.2	Line-by-Line Explanation	163
9.2.3	Creating customized versions of existing functions using wrappers	163
9.2.4	Line-by-Line Explanation	164
9.2.5	Simplifying the processing of iterable objects using comprehensions	165
9.3	<i>Code</i>	166
9.3.0	Line-by-line explanation	169
9.4	<i>Exercises</i>	170
10	Smoothing Data Responsibly	173
10.0	<i>Problem</i>	174
10.0.0	Understanding why implementing smoothing requires caution	174
10.0.1	Comparing smoothing approaches using test data	175
10.0.1.1	Understanding the common moving average	176
10.0.1.2	Smoothing by local regression	178
10.0.1.3	Smoothing by more advanced methods	179
10.0.2	Deciding when smoothing is appropriate	180
10.1	<i>Solution</i>	181
10.1.0	Generating noisy test data using random numbers from <code>numpy.random</code>	181
10.1.1	Smoothing by moving averages using <code>scipy.signal.convolve()</code>	182
10.1.2	Smoothing using local regression with <code>scipy.signal.savgol_filter()</code>	182
10.2	<i>Code</i>	183
10.2.0	Line-by-line explanation	184
10.3	<i>Exercises</i>	186
11	Interpolating Between Data Points	189
11.0	<i>Problem</i>	190
11.1	<i>Background</i>	190
11.2	<i>Solution</i>	191
11.2.0	Implementing linear interpolation using <code>numpy.interp</code>	192
11.2.1	Implementing cubic spline interpolation using <code>scipy.interpolate.CubicSpline</code>	192
11.3	<i>Code</i>	194
11.3.0	Line-by-line explanation	196
11.4	<i>Exercises</i>	196
12	Finding Peaks	199
12.0	<i>Problem</i>	200
12.1	<i>Background</i>	200
12.2	<i>Solution</i>	202
12.2.0	Finding local maxima using <code>scipy.signal.find_peaks()</code>	202
12.2.1	Looping until some conditions are met with <code>while</code> loops	202
12.3	<i>Code</i>	203
12.3.0	Line-by-line explanation	205
12.4	<i>Exercises</i>	206

13	Computing Numerical Derivatives	209
13.0	<i>Problem</i>	210
13.1	<i>Solution</i>	212
13.1.0	Refactoring code to turn it into functions you can reuse	213
13.1.1	Importing objects from Python files in arbitrary locations using <code>codechembook.quickTools.importFrom</code>	213
13.1.2	Computing derivatives using <code>numpy.gradient()</code>	217
13.2	<i>Code</i>	218
13.2.0	Line-by-line explanation	224
13.3	<i>Exercises</i>	224
IV Reporting and Sharing Results		
14	Producing Structured Data Files	229
14.0	<i>Problem</i>	230
14.1	<i>Writing a csv file</i>	230
14.1.0	Planning	230
14.1.1	Writing text files with arbitrary formatting using <code>with ... open()</code>	230
14.2	<i>Code to produce a csv file</i>	231
14.2.0	Line-by-line explanation	234
14.3	<i>Writing a Microsoft Word file</i>	235
14.3.0	Making Microsoft word tables using <code>docx</code>	235
14.4	<i>Code to produce a .docx file</i>	236
14.4.0	Line-by-line explanation	237
14.5	<i>Exercises</i>	239
15	Reading, Editing, and Writing Excel Files	241
15.0	<i>Problem</i>	242
15.1	<i>Solution</i>	243
15.1.0	Planning	243
15.1.1	Reading and writing excel files using <code>openpyxl</code>	244
15.1.2	Specifying colors using the hexadecimal standard	244
15.2	<i>Code</i>	245
15.2.0	Line-by-line explanation	248
15.3	<i>Exercises</i>	250
16	Automating Basic Image Processing	253
16.0	<i>Problem</i>	254
16.1	<i>Solution</i>	255
16.1.0	Understanding how computers represent images	255
16.1.1	Importing and manipulating images using the Pillow library	257
16.2	<i>Code</i>	258
16.2.0	Line-by-line explanation	259
16.3	<i>Exercises</i>	261

V Moving beyond this book

17 Thinking about Your Next Steps	265
17.0.0 Implementing a reasonable data management plan	266
17.0.1 Managing coding projects with GitHub	267
17.0.2 Organizing your data in databases	267
17.0.3 Ensuring your data meet FAIR data standards	267
17.0.4 Implementing more advanced data analysis techniques	268
17.0.5 Making your own libraries	268
17.0.6 Incorporating machine learning in your workflow	268
17.0.7 Taking advantage of large language mode based assistants like ChatGPT	268
17.0.8 Interfacing with hardware and software through vendor-provided APIs	269
17.0.9 Collecting data from the internet <i>via</i> scraping	269
17.0.10 Implementing ChemInformatics workflows	269
17.0.11 Understanding the strengths and weaknesses of other IDEs	269
17.0.12 Creating a record of data workup in Python notebooks	270
17.0.13 Simplifying the use of your code using graphical user interfaces or webassembly	270
17.0.14 Formatting scientific communications with Latex	271
17.0.15 Finding instruction not covered in this book	271
17.1 Exercises	272

VI Technical Details

A Computers	275
A.0 <i>A computer is not just a laptop or a rack of servers</i>	275
A.1 <i>Computers convert input data into output data</i>	275
A.1.0 What is data?	276
A.1.1 How computers store and represent data	277
A.1.1.0 Memory Structure	277
A.1.1.1 Storing Data in Memory	278
A.1.1.2 Type of Data Stored in a Variable	278
A.1.2 How computers manipulate data	279
A.2 <i>You are used to working with a “software stack”</i>	279
A.2.0 The Operating System	279
A.2.1 Libraries	280
A.2.1.0 Software Development Kits	280
A.2.2 Applications	280
A.2.3 Scripts	280
B Data Types	281
B.0 <i>Types that deal with numbers</i>	281
B.0.0 Numeric types are used for ‘normal’ math	281
B.0.0.0 int and float	281

B.0.0.1	complex	282
B.0.0.2	using numeric types	283
B.0.1	The boolean type is used when evaluating truth statements	284
B.0.2	The binary types allow direct access of computer memory	285
B.0.3	The <code>NoneType</code> lets us represent the value of nothing	285
B.1	<i>Collections let us hold multiple elements, and have new behaviors</i>	286
B.1.0	The list, and behaviors of collections	286
B.1.1	Iterable	287
B.1.2	Slicable	289
B.1.3	Mutable	290
B.1.3.0	Growing and shrinking lists	294
B.1.4	Moving on to other collection data types	295
B.1.5	sequence types hold things in specific orders (list, tuple, range)	295
B.1.5.0	list	295
B.1.5.1	tuple	295
B.1.5.2	range	296
B.1.5.3	Converting between sequence types	297
B.1.6	Set types are used to ensure we don't duplicate items	298
B.1.7	The text type lets us represent language	299
B.1.8	The dictionary type allows us to map, or connect, values	301
B.1.8.0	Accessing information from dictionaries	302
B.2	<i>Common types added by libraries</i>	303
B.3	<i>Numpy arrays allow for vectorization of data</i>	303
B.4	<i>the Pillow Image type lets you store, manipulate, and display images</i>	303
B.5	<i>the Plotly figure lets you build and display plots of data.</i>	303
C	Working With Text	305
C.0	<i>What is Text Data?</i>	305
C.1	<i>Working with text in Python using strings</i>	306
C.1.0	Creating strings	306
C.1.0.0	Creating strings from other data types using <code>str()</code>	306
C.1.0.1	How strings are interpreted and displayed	306
C.1.0.2	Modifying how strings are interpreted	307
C.1.1	Using format codes to control the display of values in f-strings	308
C.1.2	Using <code>.format()</code> to insert values into strings dynamically	308
C.1.3	Creating strings derived from other strings	309
C.1.3.0	Concatenating two strings into one	309
C.1.3.1	Replacing a subset of a string with a different string using <code>.replace()</code>	309
C.1.3.2	Changing case	309
C.1.4	Taking apart a string to create multiple new strings	310
C.1.4.0	Using a specific string to denote the divisions between substrings with <code>.split()</code>	310

C.1.4.1	<code>.splitlines()</code>	310
C.1.5	Analyzing strings to determine some information about their text	310
C.1.5.0	Finding the first instance of a substring with <code>.index()</code>	310
C.1.5.1	<code>.find()</code>	310
C.1.5.2	Checking if the string can be converted to a different type with <code>.is<type>()</code>	311
C.1.5.3	StringA in StringB	311
D	Files and Folders	313
D.0	<i>Understanding the file system</i>	313
D.0.1	All of the information that you use is stored in files	313
D.0.1.0	Files have names that (hopefully) describe their content and extensions that give clues on what program to use to open them	314
D.0.2	Files are stored in folders	314
D.0.3	Folders are a organizational hierarchy	314
D.0.4	The location of a file or folder in the file system is called a path	315
D.0.5	Absolute paths specify the location of any file or folder in the file system	316
D.0.6	Relative paths specify the location of a file or folder with respect to some other folder	316
D.1	<i>File system metadata provides additional information beyond file and folder contents</i>	317
D.1.0	Metadata is not always reliable	317
D.2	<i>Python provides a built-in library to work with the file system, but you should usually use <code>pathlib</code> instead</i>	317
E	Pathlib	319
E.0	<i>Creating a <code>Path</code> object</i>	319
E.1	<i>Accessing components of <code>Path</code> objects</i>	320
E.2	<i>Modifying existing <code>Path</code> objects</i>	320
F	Control Structures	323
F.0	<i>Getting Yes or No Answers: Comparisons and Boolean Logic</i>	323
F.0.0	Comparison Statements Compare Two Values	323
F.0.1	Complex Comparisons can be Evaluated using Boolean Logic	324
F.1	<i>Conditionals: Choosing Between Two or More Options</i>	326
F.1.0	If-Then: Do Something or Don't Do It	326
F.1.1	If-Then-Else: Do One of Two Things	327
F.1.2	If-Then-Elseif: Do One of Many Things	327
F.1.3	Possible Pitfalls of Conditionals	328
F.1.4	When to use If-Then-Elif vs. several If-Then Statements	330
F.2	<i>Loops: Doing the Same Operations Over and Over</i>	330
F.2.0	General Remarks on Loops	330
F.2.1	For Loops: Do the Same Set of Instructions a Fixed Number of Times	331
F.2.1.0	<code>enumerate</code> : get the iteration number and a list element	332
F.2.1.1	<code>zip</code> : iterate elements from multiple lists together	332
F.2.2	While Loops: Do the Same Set of Instructions Repeatedly until a Condition is Met	332
F.2.3	<code>with</code> : letting Python take care of boring stuff for you	333

F.2.4	<code>try</code> : keep your code executing even after an error occurs	333
G	Functions	337
G.0	<i>Creating and using functions</i>	337
G.1	<i>Function parameters</i>	339
G.1.0	Positional parameters	339
G.1.1	Keyword parameters	339
G.1.2	Arbitrary positional parameters: <code>*args</code>	340
G.1.3	Arbitrary keyword parameters: <code>**kwargs</code>	340
G.1.4	Order of parameters: mixing and matching parameter types	341
G.1.5	Setting default values for parameters	342
G.2	<i>Scope</i>	342
G.3	<i>Use <code>copy.deepcopy()</code> to pass mutable objects to functions</i>	344
G.4	<i>Recursion</i>	345
H	Common Errors	347
H.0	<i>General advice</i>	347
H.1	<i>Syntax Errors</i>	348
H.1.-1.1	Incorrect indentation	348
H.1.-1.2	Unmatched brackets and parentheses	348
H.1.-1.3	Missing colon	348
H.1.-1.4	Missing comma	349
H.2	<i>Segmentation Fault</i>	349
H.3	<i>Overflow</i>	349
H.4	<i>Using mutable objects incorrectly</i>	350
I	Plotly	351
I.0	<i>The basics of plotting using Plotly</i>	351
I.1	<i>Controlling formatting during Plot construction</i>	356
I.2	<i>Understanding the structure of Plotly Figure objects</i>	366
I.3	<i>Other tasks</i>	368
I.3.0	Making insets	369
I.3.1	Controlling margins	369
I.4	<i>The codechembook Plotly wrappers</i>	369
	Glossary	371