

Undergraduate Topics in Computer Science

Series Editor

Ian Mackie, University of Sussex, Brighton, UK

Advisory Editors

Samson Abramsky, Department of Computer Science, University of Oxford, Oxford, UK

Chris Hankin, Department of Computing, Imperial College London, London, UK

Dexter C. Kozen, Computer Science Department, Cornell University, Ithaca, NY, USA

Andrew Pitts, University of Cambridge, Cambridge, UK

Hanne Riis Nielson , Department of Applied Mathematics and Computer Science, Technical University of Denmark, Kongens Lyngby, Denmark

Steven S. Skiena, Department of Computer Science, Stony Brook University, Stony Brook, NY, USA

Iain Stewart, Department of Computer Science, Science Labs, University of Durham, Durham, UK

Mike Hinchey, University of Limerick, Limerick, Ireland

'Undergraduate Topics in Computer Science' (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems, many of which include fully worked solutions.

The UTiCS concept relies on high-quality, concise books in softback format, and generally a maximum of 275-300 pages. For undergraduate textbooks that are likely to be longer, more expository, Springer continues to offer the highly regarded Texts in Computer Science series, to which we refer potential authors.

More information about this series at <http://www.springer.com/series/7592>

John Hunt

A Beginners Guide to Python 3 Programming



Springer

John Hunt
Midmarsh Technology Ltd
Chippenham, Wiltshire, UK

ISSN 1863-7310 ISSN 2197-1781 (electronic)
Undergraduate Topics in Computer Science
ISBN 978-3-030-20289-7 ISBN 978-3-030-20290-3 (eBook)
<https://doi.org/10.1007/978-3-030-20290-3>

© Springer Nature Switzerland AG 2019, corrected publication 2020

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

*This book was written for, and is dedicated
to, my daughter Phoebe and son Adam;
I could not be prouder of either of you.*

Preface

There is currently huge interest in the Python programming language. This is driven by several factors; its use in schools with the Raspberry Pi platform, its ability to be used for DevOps scripts, its use in data science and machine learning and of course the language itself.

There are many books on Python, however, most assume previous programming experience or are focussed on particular aspects of Python use such as data science or machine learning or have a scientific flavor.

The aim of this book is to introduce Python to those with little or very little programming knowledge, and then to take them through to become an experienced Python developer.

As such the earlier parts of the book introduce fundamental concepts such as what a *variable* is and how a *for loop* works. In contrast, the later chapters introduce advanced concepts such as functional programming, object orientation, and exception handling.

In between a wide range of topics are introduced and discussed from a Python point of view including functions, recursion, operators, Python properties, modules and packages, protocols and monkey patching, etc.

After the core elements of Python are established, each new subject area is introduced by way of an introductory chapter presenting the topic in general, providing background on that subject, why it is of importance, etc. These introductions cover Structured Analysis, functional programming, and object orientation.

Some of the key aspects of this book are:

1. It assumes very little knowledge or experience of Python or programming.
2. It provides a basic introduction to Python as well as advanced topics such as generators and coroutines.
3. This book provides extensive coverage of object orientation and the features in Python 3 supporting classes, inheritance, and protocols.
4. Python's support for functional programming is also presented.

5. Following on from introducing the basic ideas behind functional programming, the book presents how advanced functional concepts such as closures, currying, and higher-order functions work in Python.
6. The book includes exercises at the end of most chapters with online solutions.
7. There are several case studies spread through the book that broaden understanding of preceding topics.
8. All code examples (and exercise solutions) are provided online in a GitHub repository.

Chapter Organization

Each chapter has a brief introduction, the main body of the chapter, followed by a list of (typically) online references that can be used for further reading.

Following this, there is typically an *Exercises* section that lists one or more exercises that build on the skills you will have learned in that chapter.

Sample solutions to the exercises are available in a GitHub online repository that supports this book.

What You Need

You can of course just read this book; however, following the examples in this book will ensure that you get as much as possible out of the content.

For this, you will need a computer.

Python is a cross-platform programming language and as such you can use Python on a Windows PC, a Linux box or an Apple Mac, etc. So you are not tied to a particular type of operating system; you can use whatever you have available.

However, you will need to install some software on that computer. At a minimum, you will need Python.

This book focusses on Python 3, so you will need that. Some guidance on this is provided in Chap. 2 on setting up your environment.

You will also need some form of editor in which to write your programs. There are numerous generic programming editors available for different operating systems with Vim on Linux, Notepad++ on Windows and Sublime Text on Windows, and Macs being popular choices.

However, using an integrated development environment (IDE) editor such as PyCharm will make writing and running your programs much easier.

Using an IDE

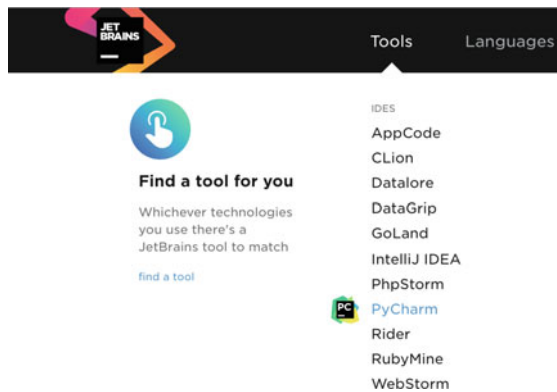
The IDE I prefer for Python is PyCharm, although it is not the only IDE for Python by any means, but it is a very widely used one.

Other IDEs available for Python include:

- Rodeo which is a lightweight, open source, IDE see <https://rodeo.yhat.com>.
- Jupyter Notebook which is a web-based IDE and is particularly good for data scientists <https://jupyter.org>.
- Visual Studio Code. This is a very good free editor from Microsoft that has really useful features <https://code.visualstudio.com>.
- Sublime Text is more of a text editor that color codes Python; however, for a simple project it may be all you need <https://www.sublimetext.com>.

Downloading the PyCharm IDE

PyCharm is provided by JetBrains who make tools for a variety of different languages. The PyCharm IDE can be downloaded from their site—see <https://www.jetbrains.com/>. Look for the menu heading ‘Tools’ and select that. You will see a long list of tools, which should include PyCharm.



Select this option. The resulting page has a lot of information on it; however, you only need to select the ‘DOWNLOAD NOW’. Make sure that you select the operating system you use (there are options for Windows, Mac OS, and Linux).

There are then two download options available: Professional and Community. The Professional version is the charged for option, while the Community version is

free. For most of the work I do in Python, the Community version is more than adequate and it is therefore the version you can download and install (note with the Professional version you do get a free trial but will need to either pay for the full version at the end of the trial or reinstall the Community version at that point).

Assuming you selected the Community edition the installer will now download, and you will be prompted to run it. Note you can ignore the request to subscribe if you want.

You can now run the installer and follow the instructions provided.

Setting Up the IDE

You need to first start the PyCharm IDE. Once started, the first dialog shown to you asks if you want to import any settings you may have had for another version of PyCharm. At this point, select ‘Do not import settings’.

Step through the next set of dialogs selecting the look and feel (I like the light version of the IDE), whether you want to share data with JetBrains, etc. Once you have completed this, click the ‘Start PyCharm’ option.

You should now be presented with the landing screen for PyCharm:

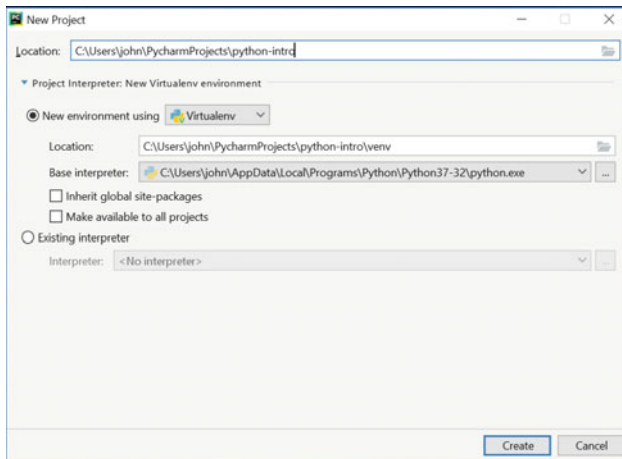


We will now create a project for you to work in. A project in PyCharm is where you write your programs and how you config what version of Python you are using and any libraries that you might need (such as graphics libraries, etc.).

Click on the ‘Create New Project’ option in the landing dialog.

You will now be asked where you want to create this new project. Again you can use the default location, but you will need to give it a name, I will call my project `python-intro`.

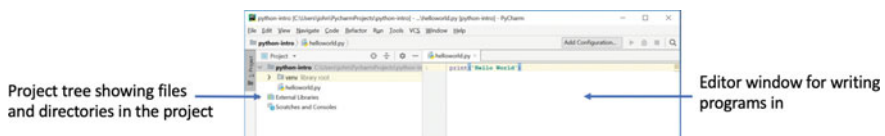
It is also worth making sure that the Python interpreter you installed has been picked up by the IDE. You can do this by opening the ‘Project Interpreter: New Virtualenv environment’ option and making sure that the base interpreter field is populated appropriately. This is shown below:



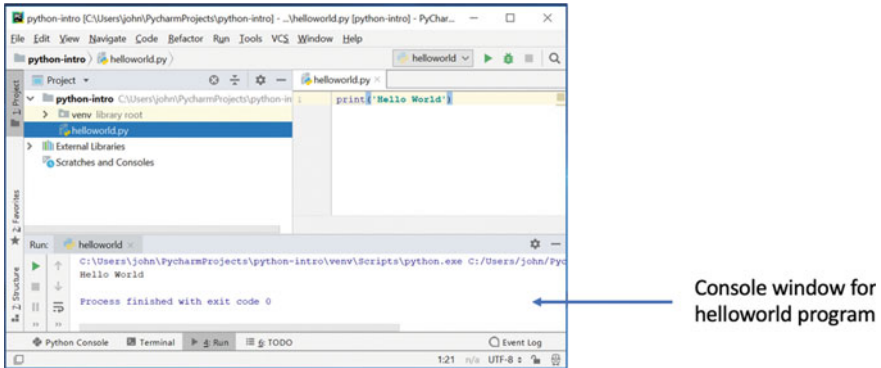
If all is OK, then select ‘Create’; if the base interpreter is not specified or is incorrect, then click on the ‘.’ button to the right of the field and browse to the appropriate location.

On opening the PyCharm project, you should see a Welcome message; click ‘Close’ and the project will be set up for you.

When you open the project, you will be shown at least two views. The left-hand view is the ‘Project’ view which shows you all the directories and files in your project. The right-hand area is where the editor is presented that allows you to type in your program. For example,



The third area that may be shown represents the output from your program. If this is the first time you have opened the project, then it may not yet be visible. However, if you run a program, it will be shown at the bottom of the IDE. For example:



Conventions

Throughout this book, you will find a number of conventions used for text styles. These text styles distinguish different kinds of information.

Code words, variable sand Python values, used within the main body of the text, are shown using a Courier font. For example:

This program creates a top-level window (the `wx.Frame`) and gives it a title. It also creates a label (a `wx.StaticText` object) to be displayed within the frame.

In the above paragraph, `wx.Frame` and `wx.StaticText` are classes available in a Python graphical user interface library.

A block of Python code is set out as shown here:

```
num = int(input('Enter another number: '))
if num > 0:
    print(num, 'is positive')
    print(num, 'squared is ', num * num)

print('Bye')
```

Note that keywords and strings are shown in bold font.

Any command line or user input is shown in *italics*:

```
> python hello.py
```

Or

```
Hello, world  
Enter your name: John  
Hello John
```

Example Code and Sample Solutions

The examples used in this book (along with sample solutions for the exercises at the end of most chapters) are available in a GitHub repository. GitHub provides both a server environment hosting Git and a web based interface to that environment.

Git is a version control system typically used to manage source code files (such as those used to create systems in programming languages such as Python but also Java, C#, C++, Scala, etc). Systems such as Git are very useful for collaborative development as they allow multiple people to work on an implementation and to merge their work together. They also provide a useful historical view of the code (which also allows developers to roll back changes if modifications prove to be unsuitable).

If you already have Git installed on your computer, then you can clone (obtain a copy of) the repository locally using:

```
git clone https://github.com/johnehunt/beginnerspython3.  
git
```

If you do not have Git, then you can obtain a zip file of the examples using <https://github.com/johnehunt/beginnerspython3/archive/master.zip>

You can of course install Git yourself if you wish. To do this, see <https://git-scm.com/downloads>. Versions of the Git client for Mac OS, Windows, and Linux/Unix are available here.

However, many IDEs such as PyCharm come with Git support and so offer another approach to obtaining a Git repository.

For more information on Git, see <http://git-scm.com/doc>. This Git guide provides a very good primer and is highly recommended.

Bath, UK

John Hunt

Contents

- 1 Introduction 1**
 - 1.1 What Is Python? 1
 - 1.2 Python Versions 2
 - 1.3 Python Programming 3
 - 1.4 Python Libraries 3
 - 1.5 Python Execution Model 4
 - 1.6 Running Python Programs 5
 - 1.6.1 Interactively Using the Python Interpreter 5
 - 1.6.2 Running a Python File 6
 - 1.6.3 Executing a Python Script 7
 - 1.6.4 Using Python in an IDE 9
 - 1.7 Useful Resources 10
- 2 Setting Up the Python Environment 13**
 - 2.1 Introduction 13
 - 2.2 Check to See If Python Is Installed 13
 - 2.3 Installing Python on a Windows PC 15
 - 2.4 Setting Up on a Mac 20
 - 2.5 Online Resources 22
- 3 A First Python Program 23**
 - 3.1 Introduction 23
 - 3.2 Hello World 23
 - 3.3 Interactive Hello World 24
 - 3.4 Variables 26
 - 3.5 Naming Conventions 28
 - 3.6 Assignment Operator 29
 - 3.7 Python Statements 29
 - 3.8 Comments in Code 30
 - 3.9 Scripts Versus Programs 30

3.10	Online Resources	31
3.11	Exercises	31
4	Python Strings	33
4.1	Introduction	33
4.2	What Are Strings?	33
4.3	Representing Strings	34
4.4	What Type Is String	35
4.5	What Can You Do with Strings?	35
4.5.1	String Concatenation	36
4.5.2	Length of a String	36
4.5.3	Accessing a Character	36
4.5.4	Accessing a Subset of Characters	37
4.5.5	Repeating Strings	37
4.5.6	Splitting Strings	38
4.5.7	Counting Strings	38
4.5.8	Replacing Strings	39
4.5.9	Finding Sub Strings	39
4.5.10	Converting Other Types into Strings	40
4.5.11	Comparing Strings	40
4.5.12	Other String Operations	40
4.6	Hints on Strings	42
4.6.1	Python Strings Are Case Sensitive	42
4.6.2	Function/Method Names	42
4.6.3	Function/Method Invocations	42
4.7	String Formatting	43
4.8	String Templates	45
4.9	Online Resources	48
4.10	Exercises	49
5	Numbers, Booleans and None	51
5.1	Introduction	51
5.2	Types of Numbers	51
5.3	Integers	52
5.3.1	Converting to Ints	53
5.4	Floating Point Numbers	53
5.4.1	Converting to Floats	54
5.4.2	Converting an Input String into a Floating Point Number	54
5.5	Complex Numbers	55
5.6	Boolean Values	55
5.7	Arithmetic Operators	57
5.7.1	Integer Operations	57
5.7.2	Negative Number Integer Division	59

5.7.3	Floating Point Number Operators	60
5.7.4	Integers and Floating Point Operations	60
5.7.5	Complex Number Operators	61
5.8	Assignment Operators	61
5.9	None Value	62
5.10	Online Resources	63
5.11	Exercises	63
5.11.1	General Exercise	64
5.11.2	Convert Kilometres to Miles	64
6	Flow of Control Using If Statements	65
6.1	Introduction	65
6.2	Comparison Operators	65
6.3	Logical Operators	66
6.4	The If Statement	66
6.4.1	Working with an If Statement	67
6.4.2	Else in an If Statement	68
6.4.3	The Use of elif	68
6.5	Nesting If Statements	69
6.6	If Expressions	70
6.7	A Note on True and False	71
6.8	Hints	72
6.9	Online Resources	72
6.10	Exercises	72
6.10.1	Check Input Is Positive or Negative	72
6.10.2	Test if a Number Is Odd or Even	73
6.10.3	Kilometres to Miles Converter	73
7	Iteration/Looping	75
7.1	Introduction	75
7.2	While Loop	75
7.3	For Loop	77
7.4	Break Loop Statement	79
7.5	Continue Loop Statement	81
7.6	For Loop with Else	82
7.7	A Note on Loop Variable Naming	83
7.8	Dice Roll Game	83
7.9	Online Resources	84
7.10	Exercises	84
7.10.1	Calculate the Factorial of a Number	84
7.10.2	Print All the Prime Numbers in a Range	85

8	Number Guessing Game	87
8.1	Introduction	87
8.2	Setting Up the Program	87
8.2.1	Add a Welcome Message	87
8.2.2	Running the Program	88
8.3	What Will the Program Do?	89
8.4	Creating the Game	90
8.4.1	Generate the Random Number	90
8.4.2	Obtain an Input from the User	91
8.4.3	Check to See If the Player Has Guessed the Number	91
8.4.4	Check They Haven't Exceeded Their Maximum Number of Guess	92
8.4.5	Notify the Player Whether Higher or Lower	93
8.4.6	End of Game Status	94
8.5	The Complete Listing	94
8.6	Hints	96
8.6.1	Initialising Variables	96
8.6.2	Blank Lines Within a Block of Code	96
8.7	Exercises	97
9	Recursion	99
9.1	Introduction	99
9.2	Recursive Behaviour	99
9.3	Benefits of Recursion	100
9.4	Recursively Searching a Tree	100
9.5	Recursion in Python	102
9.6	Calculating Factorial Recursively	102
9.7	Disadvantages of Recursion	104
9.8	Online Resources	105
9.9	Exercises	105
10	Introduction to Structured Analysis	107
10.1	Introduction	107
10.2	Structured Analysis and Function Identification	107
10.3	Functional Decomposition	108
10.3.1	Functional Decomposition Terminology	109
10.3.2	Functional Decomposition Process	110
10.3.3	Calculator Functional Decomposition Example	110
10.4	Functional Flow	112
10.5	Data Flow Diagrams	112
10.6	Flowcharts	113
10.7	Data Dictionary	115
10.8	Online Resources	116

11	Functions in Python	117
11.1	Introduction	117
11.2	What Are Functions?	117
11.3	How Functions Work	118
11.4	Types of Functions	118
11.5	Defining Functions	119
11.5.1	An Example Function	120
11.6	Returning Values from Functions	121
11.7	Docstring	123
11.8	Function Parameters	124
11.8.1	Multiple Parameter Functions	124
11.8.2	Default Parameter Values	125
11.8.3	Named Arguments	126
11.8.4	Arbitrary Arguments	127
11.8.5	Positional and Keyword Arguments	128
11.9	Anonymous Functions	129
11.10	Online Resources	131
11.11	Exercises	131
12	Scope and Lifetime of Variables	133
12.1	Introduction	133
12.2	Local Variables	133
12.3	The Global Keyword	135
12.4	Nonlocal Variables	136
12.5	Hints	137
12.6	Online Resources	138
12.7	Exercise	138
13	Implementing a Calculator Using Functions	139
13.1	Introduction	139
13.2	What the Calculator Will Do	139
13.3	Getting Started	140
13.4	The Calculator Operations	140
13.5	Behaviour of the Calculator	141
13.6	Identifying Whether the User Has Finished	142
13.7	Selecting the Operation	144
13.8	Obtaining the Input Numbers	146
13.9	Determining the Operation to Execute	147
13.10	Running the Calculator	147
13.11	Exercises	148
14	Introduction to Functional Programming	149
14.1	Introduction	149
14.2	What Is Functional Programming?	149

14.3	Advantages to Functional Programming	151
14.4	Disadvantages of Functional Programming	153
14.5	Referential Transparency	153
14.6	Further Reading	155
15	Higher Order Functions	157
15.1	Introduction	157
15.2	Recap on Functions in Python	157
15.3	Functions as Objects	158
15.4	Higher Order Function Concepts	160
15.4.1	Higher Order Function Example	161
15.5	Python Higher Order Functions	162
15.5.1	Using Higher Order Functions	163
15.5.2	Functions Returning Functions	164
15.6	Online Resources	165
15.7	Exercises	165
16	Curried Functions	167
16.1	Introduction	167
16.2	Currying Concepts	167
16.3	Python and Curried Functions	168
16.4	Closures	170
16.5	Online Resources	172
16.6	Exercises	173
17	Introduction to Object Orientation	175
17.1	Introduction	175
17.2	Classes	175
17.3	What Are Classes for?	176
17.3.1	What Should a Class Do?	177
17.3.2	Class Terminology	177
17.4	How Is an OO System Constructed?	178
17.4.1	Where Do We Start?	179
17.4.2	Identifying the Objects	180
17.4.3	Identifying the Services or Methods	181
17.4.4	Refining the Objects	182
17.4.5	Bringing It All Together	183
17.5	Where Is the Structure in an OO Program?	186
17.6	Further Reading	188
18	Python Classes	189
18.1	Introduction	189
18.2	Class Definitions	190
18.3	Creating Examples of the Class Person	191
18.4	Be Careful with Assignment	192

18.5	Printing Out Objects	194
18.5.1	Accessing Object Attributes	194
18.5.2	Defining a Default String Representation	195
18.6	Providing a Class Comment	196
18.7	Adding a Birthday Method	197
18.8	Defining Instance Methods	198
18.9	Person Class Recap	199
18.10	The del Keyword	200
18.11	Automatic Memory Management	201
18.12	Intrinsic Attributes	202
18.13	Online Resources	203
18.14	Exercises	203
19	Class Side and Static Behaviour	205
19.1	Introduction	205
19.2	Class Side Data	205
19.3	Class Side Methods	206
19.3.1	Why Class-Side Methods?	207
19.4	Static Methods	207
19.5	Hints	208
19.6	Online Resources	208
19.7	Exercises	209
20	Class Inheritance	211
20.1	Introduction	211
20.2	What Is Inheritance?	211
20.3	Terminology Around Inheritance	215
20.4	The Class Object and Inheritance	217
20.5	The Built-in Object Class	218
20.6	Purpose of Subclasses	218
20.7	Overriding Methods	219
20.8	Extending Superclass Methods	221
20.9	Inheritance Oriented Naming Conventions	222
20.10	Python and Multiple Inheritance	223
20.11	Multiple Inheritance Considered Harmful	225
20.12	Summary	230
20.13	Online Resources	230
20.14	Exercises	231
21	Why Bother with Object Orientation?	233
21.1	Introduction	233
21.2	The Procedural Approach	233
21.2.1	Procedures for the Data Structure	234
21.2.2	Packages	235

21.3	Does Object Orientation Do Any Better?	235
21.3.1	Packages Versus Classes	235
21.3.2	Inheritance	237
21.4	Summary	239
22	Operator Overloading	241
22.1	Introduction	241
22.2	Operator Overloading	241
22.2.1	Why Have Operator Overloading?	241
22.2.2	Why Not Have Operator Overloading?	242
22.2.3	Implementing Operator Overloading	242
22.3	Numerical Operators	244
22.4	Comparison Operators	247
22.5	Logical Operators	249
22.6	Summary	249
22.7	Online Resources	250
22.8	Exercises	250
23	Python Properties	253
23.1	Introduction	253
23.2	Python Attributes	253
23.3	Setter and Getter Style Methods	255
23.4	Public Interface to Properties	256
23.5	More Concise Property Definitions	258
23.6	Online Resources	260
23.7	Exercises	260
24	Error and Exception Handling	263
24.1	Introduction	263
24.2	Errors and Exceptions	263
24.3	What Is an Exception?	264
24.4	What Is Exception Handling?	265
24.5	Handling an Exception	267
24.5.1	Accessing the Exception Object	269
24.5.2	Jumping to Exception Handlers	270
24.5.3	Catch Any Exception	272
24.5.4	The Else Clause	272
24.5.5	The Finally Clause	273
24.6	Raising an Exception	274
24.7	Defining an Custom Exception	275
24.8	Chaining Exceptions	277
24.9	Online Resources	279
24.10	Exercises	279

25	Python Modules and Packages	281
25.1	Introduction	281
25.2	Modules	281
25.3	Python Modules	282
25.4	Importing Python Modules	284
25.4.1	Importing a Module	284
25.4.2	Importing from a Module	285
25.4.3	Hiding Some Elements of a Module	287
25.4.4	Importing Within a Function	288
25.5	Module Properties	288
25.6	Standard Modules	289
25.7	Python Module Search Path	291
25.8	Modules as Scripts	292
25.9	Python Packages	294
25.9.1	Package Organisation	294
25.9.2	Sub Packages	296
25.10	Online Resources	296
25.11	Exercise	297
26	Abstract Base Classes	299
26.1	Introduction	299
26.2	Abstract Classes as a Concept	299
26.3	Abstract Base Classes in Python	300
26.3.1	Subclassing an ABC	300
26.3.2	Defining an Abstract Base Class	302
26.4	Defining an Interface	304
26.5	Virtual Subclasses	305
26.6	Mixins	306
26.7	Online Resources	308
26.8	Exercises	308
27	Protocols, Polymorphism and Descriptors	311
27.1	Introduction	311
27.2	Implicit Contracts	311
27.3	Duck Typing	313
27.4	Protocols	314
27.5	An Protocol Example	315
27.6	The Context Manager Protocol	315
27.7	Polymorphism	317
27.8	The Descriptor Protocol	319
27.9	Online Resources	322
27.10	Exercises	322

28	Monkey Patching and Attribute Lookup	325
28.1	Introduction	325
28.2	What Is Monkey Patching?	325
28.2.1	How Does Monkey Patching Work?	326
28.2.2	Monkey Patching Example	326
28.2.3	The Self Parameter	327
28.2.4	Adding New Data to a Class	328
28.3	Attribute Lookup	328
28.4	Handling Unknown Attribute Access	331
28.5	Handling Unknown Method Invocations	332
28.6	Intercepting Attribute Lookup	333
28.7	Intercepting Setting an Attribute	334
28.8	Online Resources	335
28.9	Exercises	335
29	Decorators	337
29.1	Introduction	337
29.2	What Are Decorators?	337
29.3	Defining a Decorator	338
29.4	Using Decorators	339
29.5	Functions with Parameters	340
29.6	Stacked Decorators	340
29.7	Parameterised Decorators	342
29.8	Method Decorators	343
29.8.1	Methods Without Parameters	343
29.8.2	Methods with Parameters	344
29.9	Class Decorators	345
29.10	When Is a Decorator Executed?	347
29.11	Built-in Decorators	348
29.12	FuncTools Wrap	348
29.13	Online Resources	349
29.14	Book Reference	350
29.15	Exercises	350
30	Iterables, Iterators, Generators and Coroutines	353
30.1	Introduction	353
30.2	Iteration	353
30.2.1	Iterables	353
30.2.2	Iterators	354
30.2.3	The Iteration Related Methods	354
30.2.4	The Iterable Evens Class	354
30.2.5	Using the Evens Class with a for Loop	355
30.3	The Itertools Module	356
30.4	Generators	356

30.4.1	Defining a Generator Function	356
30.4.2	Using a Generator Function in a for Loop	357
30.4.3	When Do the Yield Statements Execute?	357
30.4.4	An Even Number Generator	358
30.4.5	Nesting Generator Functions	358
30.4.6	Using Generators Outside a for Loop	359
30.5	Coroutines	360
30.6	Online Resources	361
30.7	Exercises	361
31	Collections, Tuples and Lists	363
31.1	Introduction	363
31.2	Python Collection Types	363
31.3	Tuples	364
31.3.1	Creating Tuples	364
31.3.2	The tuple() Constructor Function	364
31.3.3	Accessing Elements of a Tuple	365
31.3.4	Creating New Tuples from Existing Tuples	365
31.3.5	Tuples Can Hold Different Types	366
31.3.6	Iterating Over Tuples	367
31.3.7	Tuple Related Functions	367
31.3.8	Checking if an Element Exists	368
31.3.9	Nested Tuples	368
31.3.10	Things You Can't Do with Tuples	369
31.4	Lists	369
31.4.1	Creating Lists	369
31.4.2	List Constructor Function	371
31.4.3	Accessing Elements from a List	372
31.4.4	Adding to a List	373
31.4.5	Inserting into a List	374
31.4.6	List Concatenation	374
31.4.7	Removing from a List	375
31.4.8	The pop() Method	375
31.4.9	Deleting from a List	376
31.4.10	List Methods	377
31.5	Online Resources	377
31.6	Exercises	378
32	Sets	379
32.1	Introduction	379
32.2	Creating a Set	379
32.3	The Set() Constructor Function	380
32.4	Accessing Elements in a Set	380
32.5	Working with Sets	380

32.5.1	Checking for Presence of an Element	380
32.5.2	Adding Items to a Set	381
32.5.3	Changing Items in a Set	381
32.5.4	Obtaining the Length of a Set	382
32.5.5	Obtaining the Max and Min Values in a Set	382
32.5.6	Removing an Item	382
32.5.7	Nesting Sets	383
32.6	Set Operations	384
32.7	Set Methods	386
32.8	Online Resources	386
32.9	Exercises	387
33	Dictionaries	389
33.1	Introduction	389
33.2	Creating a Dictionary	389
33.2.1	The dict() Constructor Function	390
33.3	Working with Dictionaries	391
33.3.1	Accessing Items via Keys	391
33.3.2	Adding a New Entry	391
33.3.3	Changing a Keys Value	391
33.3.4	Removing an Entry	392
33.3.5	Iterating over Keys	393
33.3.6	Values, Keys and Items	394
33.3.7	Checking Key Membership	394
33.3.8	Obtaining the Length of a Dictionary	395
33.3.9	Nesting Dictionaries	395
33.4	A Note on Dictionary Key Objects	396
33.5	Dictionary Methods	397
33.6	Online Resources	397
33.7	Exercises	398
34	Collection Related Modules	401
34.1	Introduction	401
34.2	List Comprehension	401
34.3	The Collections Module	402
34.4	The Itertools Module	405
34.5	Online Resources	406
34.6	Exercises	406
35	ADTs, Queues and Stacks	407
35.1	Introduction	407
35.2	Abstract Data Types	407
35.3	Data Structures	408

35.4	Queues	408
35.4.1	Python List as a Queue	409
35.4.2	Defining a Queue Class	409
35.5	Stacks	411
35.5.1	Python List as a Stack	412
35.6	Online Resources	413
35.7	Exercises	413
36	Map, Filter and Reduce	415
36.1	Introduction	415
36.2	Filter	415
36.3	Map	417
36.4	Reduce	419
36.5	Online Resources	420
36.6	Exercises	420
37	TicTacToe Game	423
37.1	Introduction	423
37.2	Classes in the Game	423
37.3	Counter Class	426
37.4	Move Class	427
37.5	The Player Class	427
37.6	The HumanPlayer Class	428
37.7	The ComputerPlayer Class	429
37.8	The Board Class	430
37.9	The Game Class	431
37.10	Running the Game	432
	Correction to: Functions in Python	C1