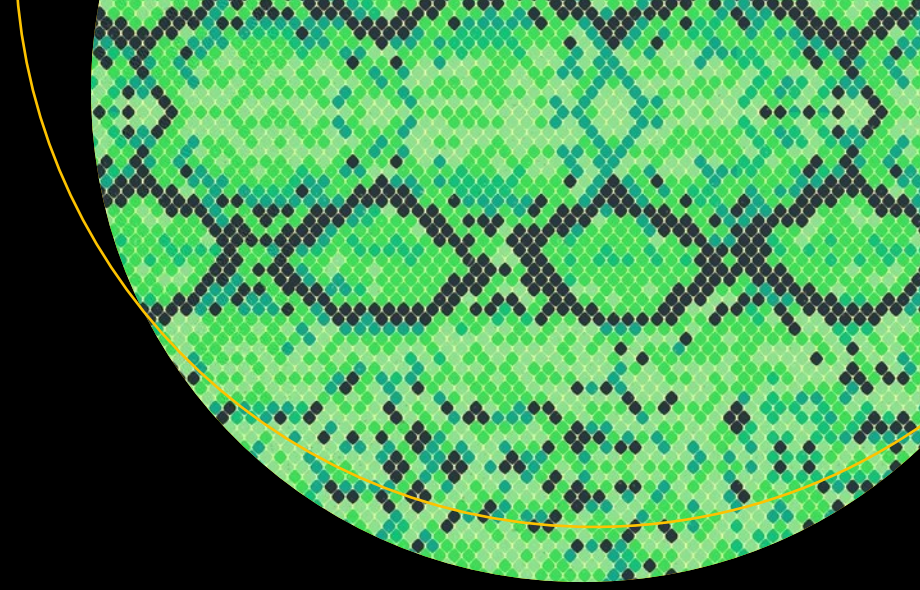


Beginning Python

From Novice to Professional

—

Fourth Edition

—

Magnus Lie Hetland
Fabio Nelli

Apress®

Beginning Python

From Novice to Professional

Fourth Edition

Magnus Lie Hetland
Fabio Nelli

Apress®

Beginning Python: From Novice to Professional, Fourth Edition

Magnus Lie Hetland
Trondheim, Norway

Fabio Nelli
ROMA, Roma, Italy

ISBN-13 (pbk): 979-8-8688-0195-2
ISBN-13 (electronic): 979-8-8688-0196-9
https://doi.org/10.1007/979-8-8688-0196-9

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Suresh John
Development Editor: James Markham
Editorial Assistant: Gryffin Winkler

Cover designed by eStudioCalamar
Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 New York Plaza, Suite 4600, New York, NY 10004-1562, USA. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub. For more detailed information, please visit https://www.apress.com/gp/services/source-code.

If disposing of this product, please recycle the paper

Table of Contents

About the Authors xxi

About the Technical Reviewers xxiii

Preface xxv

Introduction xxvii

Chapter 1: Instant Hacking: The Basics 1

The Interactive Interpreter 2

Algo . . . What? 3

Numbers and Expressions 3

Hexadecimals Octals and Binary 5

Variables 5

Statements 6

Getting Input from the User 7

Functions 8

Modules 9

cmath and Complex Numbers 10

Saving and Executing Your Programs 11

Running Your Python Scripts from a Command Prompt 12

Making Your Scripts Behave Like Normal Programs 13

Comments 14

Strings 14

Single-Quoted Strings and Escaping Quotes 14
Concatenating Strings 16
String Representations, str and repr 16
Long Strings, Raw Strings, and bytes 17
Summary 22
New Functions in This Chapter 23
What Now? 24
Chapter 2: Lists and Tuples 25
Sequence Overview 25
Common Sequence Operations 26
Indexing 26
Slicing 28
Adding Sequences 30
Multiplication 31
Membership 32
Length, Minimum, and Maximum 33
Lists: Python's Workhorse 34
The list Function 34
Basic List Operations 34
List Methods 36
Tuples: Immutable Sequences 43
Summary 44
New Functions in This Chapter 45
What Now? 45
Chapter 3: Working with Strings 47
Basic String Operations 47
String Formatting: The Short Version 47
String Formatting: The Long Version 49
Replacement Field Names 49
Basic Conversions 50

Width, Precision, and Thousands Separators 51
Signs, Alignment, and Zero-Padding 52
String Methods 54
center 55
find 55
lower 57
replace 58
split 58
strip 58
translate 59
Is My String… 60
Summary 60
New Functions in This Chapter 60
What Now? 60
Chapter 4: Dictionaries: When Indices Won't Do 61
Dictionary Uses 61
Creating and Using Dictionaries 62
The dict Function 62
Basic Dictionary Operations 63
String Formatting with Dictionaries 64
Dictionary Methods 65
Summary 71
New Functions in This Chapter 71
What Now? 71
Chapter 5: Conditionals, Loops, and Some Other Statements 73
More About print and import 73
Printing Multiple Arguments 73
Importing Something as Something Else 74
Assignment Magic 75

Sequence Unpacking 75
Chained Assignments 77
Augmented Assignments 77
Blocks: The Joy of Indentation 78
Multiline Editing 78
Conditions and Conditional Statements 80
So *That's* What Those Boolean Values Are For 80
Conditional Execution and the if Statement 81
else Clauses 82
elif Clauses 83
Nesting Blocks 83
More Complex Conditions 83
Assertions 88
Loops 89
while Loops 89
for Loops 90
Iterating Over Dictionaries 91
Some Iteration Utilities 91
Breaking Out of Loops 93
else Clauses in Loops 96
Comprehensions—Slightly Loopy 96
And Three for the Road 98
Nothing Happened! 98
Deleting with del 99
Executing and Evaluating Strings with exec and eval 100
Summary 102
New Functions in This Chapter 104
What Now? 104

Chapter 6: Abstraction 105
Laziness Is a Virtue 105
Abstraction and Structure 106
Creating Your Own Functions 106
Documenting Functions 108
Functions That Aren't Really Functions 108
The Magic of Parameters 109
Where Do the Values Come From? 109
Can I Change a Parameter? 110
Why Would I Want to Modify My Parameters? 111
What If My Parameter Is Immutable? 114
Keyword Parameters and Defaults 114
Collecting Parameters 117
Reversing the Process 119
Parameter Practice 121
Scoping 122
Recursion 125
Two Classics: Factorial and Power 126
Another Classic: Binary Search 127
Summary 130
New Functions in This Chapter 131
What Now? 131
Chapter 7: More Abstraction 133
The Magic of Objects 133
Polymorphism 134
Polymorphism and Methods 135
Polymorphism Comes in Many Forms 136
Encapsulation 137
Inheritance 138

Classes 139
What *Is* a Class, Exactly? 139
Making Your Own Classes 139
Attributes, Functions, and Methods 140
Privacy Revisited 141
The Class Namespace 142
Specifying a Superclass 144
Investigating Inheritance 144
Multiple Superclasses 145
Interfaces and Introspection 146
Some Thoughts on Object-Oriented Design 149
Summary 150
New Functions in This Chapter 151
What Now? 151
Chapter 8: Exceptions 153
What Is an Exception? 153
Making Things Go Wrong . . . Your Way 153
The raise Statement 154
Custom Exception Classes 155
Catching Exceptions 155
Look, Ma, No Arguments! 158
More Than One except Clause 160
Catching Two Exceptions with One Block 161
Catching the Object 162
A Real Catchall 162
When All Is Well 163
And Finally. 165
Exceptions and Functions 165
The Zen of Exceptions 166
Not All That Exceptional 168

A Quick Summary 169
New Functions in This Chapter 170
What Now? 170

Chapter 9: Magic Methods, Properties, and Iterators 171

Constructors 171
Overriding Methods in General, and the Constructor in Particular 172
Calling the Unbound Superclass Constructor 174
Using the super Function 175
Item Access 176
The Basic Sequence and Mapping Protocol 177
Subclassing list, dict, and str 179
More Magic 180
Properties 180
The property Function 181
Static Methods and Class Methods 183
__getattr__, __setattr__, and Friends 184
Iterators 185
The Iterator Protocol 185
Making Sequences from Iterators 186
Generators 187
Making a Generator 187
A Recursive Generator 188
Generators in General 190
Generator Methods 190
Simulating Generators 191
The Eight Queens 192
Generators and Backtracking 192
The Problem 193
State Representation 194
Finding Conflicts 194

The Base Case 194
The Recursive Case 196
Wrapping It Up 197
Summary 198
New Functions in This Chapter 199
What Now? 199
Chapter 10: Batteries Included 201
Modules 201
Modules Are Programs 201
Modules Are Used to Define Things 203
Making Your Modules Available 205
Packages 207
Exploring Modules 208
What's in a Module? 208
Getting Help with help 210
Documentation 211
Use the Source 211
The Standard Library: A Few Favorites 212
sys 212
os 214
fileinput 216
Sets, Heaps, and Deques 218
time 223
random 225
shelve and json 229
re 232
Other Interesting Standard Modules 245
Summary 247
New Functions in This Chapter 248
What Now? 248

Chapter 11: Files and Stuff 249

Opening Files 249

File Modes 249

The Basic File Methods 250

Reading and Writing 251

Piping Output 252

Reading and Writing Lines 253

Closing Files 254

Using the Basic File Methods 255

Iterating Over File Contents 256

One Character (or Byte) at a Time 256

One Line at a Time 259

Reading Everything 259

Lazy Line Iteration with fileinput 261

File Iterators 261

CSV Files 263

XML Files 265

HTML Files 268

JSON Files 270

Apache Parquet 271

Summary 272

New Functions in This Chapter 273

What Now? 273

Chapter 12: Graphical User Interfaces 275

Building a Sample GUI Application 275

Initial Exploration 276

Layout 279

Event Handling 281

The Final Program 282

Using Something Else 284
Summary 284
What Now? 284
Chapter 13: Database Support 285
The Python Database API 285
Global Variables 286
Exceptions 287
Connections and Cursors 287
Types 289
SQLite and PySQLite 290
Getting Started 291
A Sample Database Application 291
Creating and Populating Tables 293
Searching and Dealing with Results 294
SQL Alchemy 295
A Database in a Container with Docker 295
Setting Up a PostgreSQL Database with Docker 296
Using a PostgreSQL Database with Python 301
Using Mongo, a No-SQL Database with Docker and Python 304
Summary 306
New Functions in This Chapter 307
What Now? 307
Chapter 14: Network Programming 309
A Couple of Networking Modules 309
The socket Module 310
The urllib3 Module 312
Other Modules 313
socketserver and http.server 314

Multiple Connections 316
Enhance an HTTP server with socketserver Forking and Threading 317
Asynchronous I/O with asyncio 318
Twisted 320
Downloading and Installing Twisted 320
Writing a Twisted Server 320
Summary 324
What Now? 324
Chapter 15: Python and the Web 325
Screen Scraping 325
Tidy and XHTML Parsing 326
What's Tidy? 326
Getting Tidy 328
But Why XHTML? 328
Using HTMLParser 329
Beautiful Soup 330
Dynamic Web Pages with CGI 331
Step 1: Preparing the Web Server 331
Step 2: Adding the Pound Bang Line 333
Step 3: Setting the File Permissions 333
CGI Security Risks 334
A Simple CGI Script 334
Debugging with cgitb 335
Using the cgi Module 336
A Simple Form 338
Using a Web Framework 339
Other Web Application Frameworks 340
Web Services: Scraping Done Right 341
RSS and Friends 341
Remote Procedure Calls with XML-RPC 342
SOAP 343

Summary 343
New Functions in This Chapter 343
What Now? 343

Chapter 16: Testing, 1-2-3 345

Test First, Code Later 345
Precise Requirement Specification 345
Planning for Change 347
The 1-2-3 (and 4) of Testing 347

Tools for Testing 348
doctest 348
unittest 350

Beyond Unit Tests 354
Source Code Checking with PyLint 354
Profiling 356

Summary 358
New Functions in This Chapter 358
What Now? 358

Chapter 17: Extending Python 359

The Best of Both Worlds 359

The Really Easy Way: Jython and IronPython 360

Writing C Extensions 368
A Swig of … SWIG 369
What Does It Do? 371
I Prefer Pi 371
The Interface File 372
Running SWIG 372
Compiling, Linking, and Using 372
Hacking It on Your Own 375
Reference Counting 375
A Framework for Extensions 376

Summary 377
New Functions in This Chapter 378
What Now? 378

Chapter 18: Packaging and Distributing Your Programs 379
Packages and Packaging 379
setuptools 380
Flit 383
Creating Stand-Alone Applications 385
Virtual Environments and Dependency Management 386
Summary 389
New Functions in This Chapter 389
What Now? 389

Chapter 19: Playful Programming 391
Why Playful? 391
The Jujitsu of Programming 391
Prototyping 392
Developing with an IDE: Spyder 393
Configuration 396
Extracting Constants 396
Configuration Files 396
Logging 400
If You Can't Be Bothered 402
If You Want to Learn More 403
A Quick Summary 403
What Now? 404

Chapter 20: Project 1: Instant Markup 405
What's the Problem? 405
Useful Tools 406
Preparations 406

First Implementation 407
Finding Blocks of Text 407
Adding Some Markup 409
Second Implementation 412
Handlers 413
A Handler Superclass 414
Rules 415
A Rule Superclass 416
Filters 416
The Parser 417
Constructing the Rules and Filters 418
Putting It All Together 421
Further Exploration 425
What Now? 426
Chapter 21: Project 2: XML for All Occasions 427
What's the Problem? 427
Useful Tools 428
Preparations 429
First Implementation 430
Creating a Simple Content Handler 430
Creating HTML Pages 433
Second Implementation 436
A Dispatcher Mix-In Class 436
Factoring Out the Header, Footer, and Default Handling 438
Support for Directories 439
The Event Handlers 439
Further Exploration 442
What Now? 442

Chapter 22: Project 3: File Sharing with XML-RPC 443
What's the Problem? 443
Useful Tools 444
Preparations 445
First Implementation 445
Implementing a Simple Node 446
Trying Out the First Implementation 450
Second Implementation 453
Creating the Client Interface 453
Raising Exceptions 454
Validating Filenames 455
Trying the Second Implementation 458
Further Exploration 460
What Now? 461
Chapter 23: Project 4: File Sharing II—Now with GUI! 463
What's the Problem? 463
Useful Tools 463
Preparations 463
First Implementation 464
Second Implementation 465
Further Exploration 468
What Now? 468
Chapter 24: Project 5: Do-It-Yourself Arcade Game 469
What's the Problem? 469
Useful Tools 470
pygame 470
pygame.locals 470
pygame.display 470
pygame.font 471
pygame.sprite 471

pygame.mouse 471
pygame.event 471
pygame.image 472
Preparations 472
First Implementation 473
Second Implementation 476
Further Exploration 485
What Now? 485
Chapter 25: Activity 1: Data Analysis with Pandas, Matplotlib, and Seaborn 487
Jupyter Notebook 487
The Pandas Library 492
The Matplotlib and Seaborn Libraries 493
Our Data Source: Kaggle 493
Loading the Titanic Data Set 495
Data Analysis: Exploring the Titanic Data Set 497
Further Exploration 504
What Now? 504
Chapter 26: Activity 2: Machine Learning with scikit-learn 505
What Is Machine Learning? 505
The scikit-learn Library 506
The Classification Problem 506
Data Analysis Before the Classification 507
Model Training for Classification 511
The Regression Problem 512
Further Exploration 516
What Now? 516
Chapter 27: Activity 3: Building a Web App with Flask 517
Flask: A Micro-Framework for Web Applications 517
JupyterLab 518

Getting Started with Flask 520
A Few Steps Forward 522
Adding a Database 528
Further Exploration 535
What Now? 535

Chapter 28: Activity 4: Asynchronous Programming with asyncio 537
The asyncio Library 537
Basic Concepts of asyncio 537
PyCharm 538
Getting Started with asyncio 540
Using a Queue in Asynchronous Programming 543
Extending with aiohttp 544
Further Exploration 546
What Now? 546

Chapter 29: Activity 5: Web Scraping with Requests and BeautifulSoup 547
Web Scraping 547
The Requests and BeautifulSoup Libraries 548
Getting Started with Requests and BeautifulSoup 548
Exception Handling and Data Saving 553
Further Exploration 556
What Now? 556

Appendix A: The Short Version 557
Appendix B: Python Reference 565
Appendix C: Development Tools for Python 581
Appendix D: Removing Dead Batteries 591

Index 593

About the Authors

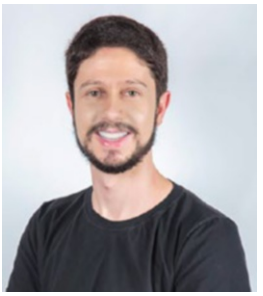
Magnus Lie Hetland is an experienced Python programmer, having used the language since the late 1990s. He is also an associate professor of computer science at the Norwegian University of Science and Technology, where he specializes in algorithm analysis and design. Hetland is the author of *Python Algorithms*, as well as the previous editions of *Beginning Python*.

Fabio Nelli is a data scientist and consultant for companies in the scientific-pharmaceutical field and teaches Python programming and data management. He is the author of several books on programming, such as *Python Data Analytics*, now in its third edition. He gained experience in programming while working for companies such as IBM, EDS, Merck, Hewlett-Packard, and several banks and insurance companies. He has a master's degree in organic chemistry and a bachelor's degree in automation IT engineering.

About the Technical Reviewers

Andrea Gavana has been programming Python for more than 20 years, as well as dabbling with other languages since the late 1990s. He has a master's degree in chemical engineering, and he is now a master development planning architect working for TotalEnergies in Copenhagen, Denmark.

Andrea enjoys programming at work and for fun, and he has been involved in multiple open-source projects, all Python-based. One of his favorite hobbies is Python coding, but he is also fond of cycling, swimming, and cozy dinners with family and friends. This is his fourth book as a technical reviewer.



Vinícius Gubiani Ferreira is an experienced IT professional with more than 15 years of experience in IT areas such as software development, cloud computing, and DevOps. He's been working, learning, and sharing knowledge about Python for 10 years. He currently works as a QA team lead and previously worked as a software engineer in several different industries. He studied electrical engineering at UFRGS and software engineering at PUCRS, and he has an MBA in project management from FGV; he also loves to code, read other people's code, and help others achieve what they want with code, be it directly or by guiding them to figure it out for themselves.

Preface

Hello! Magnus here. Another edition—this time with less involvement from me. This time around, Fabio has entered the scene and done the heavy lifting, updating outdated material and replacing some of the rustier projects. This includes new material on using various coding environments, handling files in several specialized formats, and using Docker with databases and `asyncio` for network programming, for example. He also replaced essentially all of the (really outdated) material on packaging in Chapter 18, swapped out five of the projects with new activities (Chapters 25–29), and added a couple of appendixes (C and D). And he added lots of screenshots! Thanks to him, to the technical reviewers, and to the Apress staff for making this new edition a reality!

Introduction

A C program is like a fast dance on a newly waxed dance floor by people carrying razors.

—Waldi Ravens

C++: Hard to learn and built to stay that way.

—Anonymous

Java is, in many ways, C++ – –.

—Michael Feldman

And now for something completely different . . .

—Monty Python's Flying Circus

I've started this introduction with a few quotes to set the tone for the book, which is rather informal. In the hope of making it an easy read, I've tried to approach the topic of Python programming with a healthy dose of humor, and true to the traditions of the Python community, much of this humor is related to Monty Python sketches. As a consequence, some of my examples may seem a bit silly; I hope you will bear with me. (And, yes, the name Python is derived from Monty Python, not from snakes belonging to the family Pythonidae.) In this introduction, I give you a quick look at what Python is, why you should use it, who uses it, who this book's intended audience is, and how the book is organized.

So, what is Python, and why should you use it? To quote an old official blurb, it is "an interpreted, object-oriented, high-level programming language with dynamic semantics." Many of these terms will become clear as you read this book, but the gist is that Python is a programming language that knows how to stay out of your way when you write your programs. It enables you to implement the functionality you want without any hassle and lets you write programs that are clear and readable (much more so than programs in most other currently popular programming languages).

Even though Python might not be as fast as compiled languages such as C or C++, what you save in programming time will probably make it worth using it, and in most programs, the speed difference won't be noticeable anyway. If you are a C programmer, you can easily implement the critical parts of your program in C at a later date and have them interoperate with the Python parts. If you haven't done any programming before (and perhaps are a bit confused by my references to C and C++), Python's combination of simplicity and power makes it an ideal choice as a place to start.

So, who uses Python? Since Guido van Rossum created the language in the early 1990s, its following has grown steadily, and interest has increased markedly in the past few years. Python is used extensively for system administration tasks (it is, for example, a vital component of several Linux distributions), but it is also used to teach programming to complete beginners. The US National Aeronautics and Space Administration (NASA) uses Python both for development and as a scripting language in several of its systems. Industrial Light & Magic uses Python in its production of special effects for large-budget feature films. Yahoo! uses it (among other things) to manage its discussion groups. Google has used it to implement many components of its web crawler and search engine. Python is being used in such diverse areas as computer games and bioinformatics. Soon one might as well ask, "Who isn't using Python?"

This book is for those of you who want to learn how to program in Python. It is intended to suit a wide audience, from neophyte programmers to advanced computer whizzes. If you have never programmed before, you should start by reading Chapter 1 and continue until you find that things get too advanced for you (if, indeed, they do). Then you should start practicing and write some programs of your own. When the time is right, you can return to the book and proceed with the more intricate stuff.

If you already know how to program, some of the introductory material might not be new to you (although there will probably be some surprising details here and there). You could skim through the early chapters to get an idea of how Python works or perhaps read Appendix A, which is based on my online Python tutorial "Instant Python." It will get you up to speed on the most important Python concepts. After getting the big picture, you could jump straight to Chapter 10 (which describes the Python standard libraries).

The last 10 chapters present 10 programming projects and activities, which show off various capabilities of the Python language. These projects and activities should be of interest to beginners and experts alike. They touch upon a wide range of topics, most of which will be useful to you when writing programs of your own. You will learn how to do things that may seem completely out of reach to you at this point, such as creating a peer-to-peer file-sharing system or a full-fledged graphical computer game. Although much of the material may seem hard at first glance, I think you will be surprised by how easy most of it really is.

Well, that's it. I always find long introductions a bit boring myself, so I'll let you continue with your Pythoneering, either in Chapter 1 or in Appendix A. Good luck, and happy hacking.

CHAPTER 1

Instant Hacking: The Basics

It's time to start hacking.[1] In this chapter, you learn how to take control of your computer by speaking a language it understands: Python. Nothing here is particularly difficult, so if you know the basic principles of how your computer works, you should be able to follow the examples and try them out yourself. I'll go through the basics, starting with the excruciatingly simple, but because Python is such a powerful language, you'll soon be able to do pretty advanced things.

To begin, you need to install Python, or verify that you already have it installed. If you're running macOS, Windows, or Linux/UNIX, open a terminal (the Terminal app on a Mac, or Command on Windows), type in `python`, and press Enter. You should get a welcome message describing the current version and operating system in which it runs. Furthermore, some commands are suggested that could be useful, such as `help`, with which you can obtain information on other commands. Finally, you'll see a prompt consisting of the three characters `>>>` as follows:

```
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If everything went correctly as described, then it means you have just opened an **interactive interpreter session** and the system is ready to accept any line of code in Python and execute it after pressing the Enter key.

But before we start working with it, you can immediately check the currently installed version on your system. If it is too outdated, you should update it to the latest version released. A quicker way to know the Python version, without opening an interactive interpreter session, is to write the following command in the terminal:

```
python --version
```

The details of the installation process will of course vary with your OS and preferred installation mechanism, but the most straightforward approach is to visit `https://www.python.org/downloads/`, where all versions of Python are listed, including the latest release, each with a download link. For Windows and Mac, you'll download an installer that you can run to actually install Python. For Linux/UNIX, Python is generally installed by default, but if this is not the case, it is always possible to install it via APT or other advanced package tools (depending on the distributions).

[1] *Hacking* is not the same as *cracking*, which is a term describing computer crime. The two are often confused, and the usage is gradually changing. *Hacking*, as I'm using it here, basically means "having fun while programming."

M. L. Hetland and F. Nelli, *Beginning Python*, https://doi.org/10.1007/979-8-8688-0196-9_1

■ **Note** The starting point for working with Python is to open an interactive interpreter session from the terminal and start executing commands line by line. There are, however, many other ways to develop and execute code in Python, some simpler and others more complex, which make use of additional applications or web interfaces. We will look at some of these throughout the book, but in the meantime I recommend taking a look at Appendix C, which shows an overview of these methods, with a detailed description of their use.

The Interactive Interpreter

When you start up Python, you get a prompt similar to the following:

```
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The exact appearance of the interpreter and its error messages will depend on which version you are using. This might not seem very interesting, but believe me, it is. This is your gateway to hackerdom—your first step in taking control of your computer. In more pragmatic terms, it's an interactive Python interpreter. Just to see if it's working, try the following:

```
>>> print("Hello, world!")
```

When you press the Enter key, the following output appears:

```
Hello, world!
>>>
```

If you are familiar with other computer languages, you may be used to terminating every line with a semicolon. There is no need to do so in Python. A line is a line, more or less. You can add a semicolon if you like, but it won't have any effect (unless more code follows on the same line), and it is not a common thing to do.

So what happened here? The `>>>` thingy is the prompt. You can write something in this space, like `print("Hello, world!")`. If you press Enter, the Python interpreter prints out the string "Hello, world!" and you get a new prompt below that.

What if you write something completely different? Try it:

```
>>> The Spanish Inquisition
SyntaxError: invalid syntax
>>>
```

Obviously, the interpreter didn't understand that.[2] (If you are running an interpreter other than IDLE, such as the command-line version for Linux, the error message will be slightly different.) The interpreter also indicates what's wrong: it will emphasize the word *Spanish* by giving it a red background (or, in the command-line version, by using a caret, ^).

[2] After all, no one expects the Spanish Inquisition . . .

If you feel like it, play around with the interpreter some more. For some guidance, try entering the command `help()` at the prompt and pressing Enter. You can press F1 for help about IDLE. Otherwise, let's press on. After all, the interpreter isn't much fun when you don't know what to tell it.

Algo . . . What?

Before we start programming in earnest, I'll try to give you an idea of what computer programming is. Simply put, it's telling a computer what to do. Computers can do a lot of things, but they aren't very good at thinking for themselves. They really need to be spoon-fed the details. You need to feed the computer an algorithm in some language it understands. *Algorithm* is just a fancy word for a procedure or recipe—a detailed description of how to do something. Consider the following:

```
SPAM with SPAM, SPAM, Eggs, and SPAM: First, take some SPAM.
Then add some SPAM, SPAM, and eggs.
If a particularly spicy SPAM is desired, add some SPAM.
Cook until done -- Check every 10 minutes.
```

Not the fanciest of recipes, but its structure can be quite illuminating. It consists of a series of instructions to be followed in order. Some of the instructions may be done directly ("take some SPAM"), while some require some deliberation ("If a particularly spicy SPAM is desired"), and others must be repeated several times ("Check every 10 minutes.")

Recipes and algorithms consist of ingredients (objects, things) and instructions (statements). In this example, SPAM and eggs are the ingredients, while the instructions consist of adding SPAM, cooking for a given length of time, and so on. Let's start with some reasonably simple Python ingredients and see what you can do with them.

Numbers and Expressions

The interactive Python interpreter can be used as a powerful calculator. Try the following:

```
>>> 2 + 2
```

This should give you the answer 4. That wasn't too hard. Well, what about this:

```
>>> 53672 + 235253
288925
```

Still not impressed? Admittedly, this is pretty standard stuff. (I'll assume that you've used a calculator enough to know the difference between `1 + 2 * 3` and `(1 + 2) * 3`.) All the usual arithmetic operators work as expected. Division produces decimal numbers, called *floats* (or *floating-point numbers*).

```
>>> 1 / 2
0.5
>>> 1 / 1
1.0
```

If you'd rather discard the fractional part and do integer division, you can use a double slash.

```
>>> 1 // 2
0
>>> 1 // 1
1
>>> 5.0 // 2.4
2.0
```

Now you've seen the basic arithmetic operators (addition, subtraction, multiplication, and division), but I've left out a close relative of integer division.

```
>>> 1 % 2
1
```

This is the remainder (modulus) operator. `x % y` gives the remainder of `x` divided by `y`. In other words, it's the part that's left over when you use integer division. That is, `x % y` is the same as `x - ((x // y) * y)`.

```
>>> 10 // 3
3
>>> 10 % 3
1
>>> 9 // 3
3
>>> 9 % 3
0
>>> 2.75 % 0.5
0.25
```

Here `10 // 3` is 3 because the result is rounded down. But `3 × 3` is 9, so you get a remainder of 1. When you divide 9 by 3, the result is exactly 3, with no rounding. Therefore, the remainder is 0. This may be useful if you want to check something "every 10 minutes" as in the recipe earlier in the chapter. You can simply check whether `minute % 10` is 0. (For a description on how to do this, see the sidebar "Sneak Peek: The if Statement" later in this chapter.) As you can see from the final example, the remainder operator works just fine with floats as well. It even works with negative numbers, and this can be a little confusing.

```
>>> 10 % 3
1
>>> 10 % -3
-2
>>> -10 % 3
2
>>> -10 % -3
-1
```

Looking at these examples, it might not be immediately obvious how it works. It's probably easier to understand if you look at the companion operation of integer division.

```
>>> 10 // 3
3
>>> 10 // -3
-4
```

```
>>> -10 // 3
-4
>>> -10 // -3
3
```

Given how the division works, it's not that hard to understand what the remainder must be. The important thing to understand about integer division is that it is rounded *down*, which for negative numbers is *away from zero*. That means `-10 // 3` is rounded *down* to `-4`, not *up* to `-3`.

The last operator we'll look at is the exponentiation (or power) operator.

```
>>> 2 ** 3
8
>>> -3 ** 2
-9
>>> (-3) ** 2
9
```

Note that the exponentiation operator binds tighter than the negation (unary minus), so `-3**2` is in fact the same as `-(3**2)`. If you want to calculate `(-3)**2`, you must say so explicitly.

Hexadecimals Octals and Binary

To conclude this section, I should mention that hexadecimal, octal, and binary numbers are written like this:

```
>>> 0xAF
175
>>> 0b10
2
>>> 0b1011010010
722
```

The first digit in both of these is zero. (If you don't know what this is all about, you probably don't need this quite yet. Just file it away for later use.)

Variables

Another concept that might be familiar to you is *variables*. If algebra is but a distant memory, don't worry: variables in Python are easy to understand. A variable is a name that represents (or refers to) some value. For example, you might want the name `x` to represent `3`. To make it so, simply execute the following:

```
>>> x = 3
```

This is called an *assignment*. We assign the value `3` to the variable `x`. Another way of putting this is to say that we *bind* the variable `x` to the value (or object) `3`. After you've assigned a value to a variable, you can use the variable in expressions.

```
>>> x * 2
6
```