

Contents

1 Introduction 33

1.1	Why Did We Write This Book?	33
1.2	What Does This Book Provide?	34
1.3	Structure of the Book	34
1.4	How Should You Read This Book?	35
1.5	Sample Programs	36
1.6	Preface To the First English Edition (2022)	36
1.7	Acknowledgments	37

2 The Python Programming Language 39

2.1	History, Concepts, and Areas of Application	39
2.1.1	History and Origin	39
2.1.2	Basic Concepts	40
2.1.3	Possible Areas of Use and Strengths	41
2.1.4	Examples of Use	42
2.2	Installing Python	42
2.2.1	Installing Anaconda on Windows	43
2.2.2	Installing Anaconda on Linux	43
2.2.3	Installing Anaconda on macOS	44
2.3	Installing Third-Party Modules	45
2.4	Using Python	45

Part I Getting Started with Python

3 Getting Started with the Interactive Mode 49

3.1	Integers	49
3.2	Floats	51
3.3	Character Strings	51

3.4	Lists	52
3.5	Dictionaries	53
3.6	Variables	54
3.6.1	The Special Meaning of the Underscore	54
3.6.2	Identifiers	55
3.7	Logical Expressions	55
3.8	Functions and Methods	57
3.8.1	Functions	57
3.8.2	Methods	58
3.9	Screen Outputs	59
3.10	Modules	60

4 The Path to the First Program 63

4.1	Typing, Compiling, and Testing	63
4.1.1	Windows	63
4.1.2	Linux and macOS	64
4.1.3	Shebang	65
4.1.4	Internal Processes	65
4.2	Basic Structure of a Python Program	66
4.2.1	Wrapping Long Lines	68
4.2.2	Joining Multiple Lines	69
4.3	The First Program	70
4.3.1	Initialization	71
4.3.2	Loop Header	71
4.3.3	Loop Body	71
4.3.4	Screen Output	72
4.4	Comments	72
4.5	In Case of Error	72

5 Control Structures 75

5.1	Conditionals	75
5.1.1	The if Statement	75
5.1.2	Conditional Expressions	78

5.2	Loops	79
5.2.1	The while Loop	79
5.2.2	Termination of a Loop	80
5.2.3	Detecting a Loop Break	81
5.2.4	Aborting the Current Iteration	82
5.2.5	The for Loop	84
5.2.6	The for Loop as a Counting Loop	85
5.3	The pass Statement	87
5.4	Assignment Expressions	87
5.4.1	The Guessing Numbers Game with Assignment Expressions	89

6 Files 91

6.1	Data Streams	91
6.2	Reading Data from a File	92
6.2.1	Opening and Closing a File	92
6.2.2	The with Statement	93
6.2.3	Reading the File Content	94
6.3	Writing Data to a File	96
6.4	Generating the File Object	97
6.4.1	The Built-In open Function	97
6.4.2	Attributes and Methods of a File Object	99
6.4.3	Changing the Write/Read Position	100

7 The Data Model 103

7.1	The Structure of Instances	105
7.1.1	Data Type	106
7.1.2	Value	107
7.1.3	Identity	108
7.2	Deleting References	109
7.3	Mutable versus Immutable Data Types	111

8 Functions, Methods, and Attributes 115

8.1 Parameters of Functions and Methods	115
8.1.1 Positional Parameters	116
8.1.2 Keyword Arguments	116
8.1.3 Optional Parameters	117
8.1.4 Keyword-Only Parameters	117
8.2 Attributes	118

9 Sources of Information on Python 119

9.1 The Built-In Help Function	119
9.2 The Online Documentation	120
9.3 PEPs	120

Part II Data Types

10 Basic Data Types: An Overview 125

10.1 Nothingness: NoneType	126
10.2 Operators	127
10.2.1 Operator Precedence	127
10.2.2 Evaluation Order	129
10.2.3 Concatenating Comparisons	129

11 Numeric Data Types 131

11.1 Arithmetic Operators	131
11.2 Comparison Operators	133
11.3 Conversion between Numeric Data Types	134
11.4 Integers: int	135
11.4.1 Numeral Systems	135

11.4.2	Bit Operations	137
11.4.3	Methods	141
11.5	Floats: float	141
11.5.1	Exponential Notation	142
11.5.2	Precision	142
11.5.3	Infinite and Not a Number	143
11.6	Boolean Values: bool	144
11.6.1	Logical Operators	144
11.6.2	Truth Values of Non-Boolean Data Types	146
11.6.3	Evaluating Logical Operators	148
11.7	Complex Numbers: complex	149

12 Sequential Data Types 153

12.1	The Difference between Text and Binary Data	153
12.2	Operations on Instances of Sequential Data Types	154
12.2.1	Checking for Elements	155
12.2.2	Concatenation	157
12.2.3	Repetition	158
12.2.4	Indexing	159
12.2.5	Slicing	160
12.2.6	Length of a Sequence	164
12.2.7	The Smallest and the Largest Element	164
12.2.8	Searching for an Element	165
12.2.9	Counting Elements	166
12.3	The list Data Type	166
12.3.1	Changing a Value within the List: Assignment via []	167
12.3.2	Replacing Sublists and Inserting New Elements: Assignment via []	167
12.3.3	Deleting Elements and Sublists: del in Combination with []	168
12.3.4	Methods of list Instances	169
12.3.5	Sorting Lists: s.sort([key, reverse])	171
12.3.6	Side Effects	174
12.3.7	List Comprehensions	177
12.4	Immutable Lists: tuple	179
12.4.1	Packing and Unpacking	179
12.4.2	Immutable Doesn't Necessarily Mean Unchangeable!	181

12.5	Strings: str, bytes, bytearray	182
12.5.1	Control Characters	185
12.5.2	String Methods	187
12.5.3	Formatting Strings	196
12.5.4	Character Sets and Special Characters	207

13 Mappings and Sets 215

13.1	Dictionary: dict	215
13.1.1	Creating a Dictionary	215
13.1.2	Keys and Values	216
13.1.3	Iteration	218
13.1.4	Operators	219
13.1.5	Methods	221
13.1.6	Dict Comprehensions	227
13.2	Sets: set and frozenset	227
13.2.1	Creating a Set	228
13.2.2	Iteration	229
13.2.3	Operators	230
13.2.4	Methods	235
13.2.5	Mutable Sets: set	236
13.2.6	Immutable Sets: frozenset	237

14 Collections 239

14.1	Chained Dictionaries	239
14.2	Counting Frequencies	240
14.3	Dictionaries with Default Values	242
14.4	Doubly Linked Lists	243
14.5	Named Tuples	245

15 Date and Time 247

15.1 Elementary Time Functions—time	247
15.1.1 The struct_time Data Type	248
15.1.2 Constants	249
15.1.3 Functions	250
15.2 Object-Oriented Date Management: datetime	254
15.2.1 datetime.date	255
15.2.2 datetime.time	256
15.2.3 datetime.datetime	257
15.2.4 datetime.timedelta	259
15.2.5 Operations for datetime.datetime and datetime.date	262
15.3 Time Zones: zoneinfo	263
15.3.1 The IANA Time Zone Database	263
15.3.2 Specifying the Time in Local Time Zones	265
15.3.3 Calculating with Time Indications in Local Time Zones	265

16 Enumerations and Flags 269

16.1 Enumeration Types: enum	269
16.2 Enumeration Types for Bit Patterns: flag	271
16.3 Integer Enumeration Types: IntEnum	272

Part III Advanced Programming Techniques

17 Functions 277

17.1 Defining a Function	278
17.2 Return Values	280
17.3 Function Objects	282
17.4 Optional Parameters	282
17.5 Keyword Arguments	283
17.6 Arbitrarily Many Parameters	284

17.7	Keyword-Only Parameters	286
17.8	Positional-Only Parameters	287
17.9	Unpacking When Calling a Function	288
17.10	Side Effects	290
17.11	Namespaces	293
17.11.1	Accessing Global Variables: global	293
17.11.2	Accessing the Global Namespace	294
17.11.3	Local Functions	295
17.11.4	Accessing Parent Namespaces: nonlocal	296
17.11.5	Unbound Local Variables: A Stumbling Block	297
17.12	Anonymous Functions	299
17.13	Recursion	300
17.14	Built-In Functions	300
17.14.1	abs(x)	304
17.14.2	all(iterable)	304
17.14.3	any(iterable)	305
17.14.4	ascii(object)	305
17.14.5	bin(x)	305
17.14.6	bool([x])	306
17.14.7	bytearray([source, encoding, errors])	306
17.14.8	bytes([source, encoding, errors])	307
17.14.9	chr(i)	307
17.14.10	complex([real, imag])	307
17.14.11	dict([source])	308
17.14.12	divmod(a, b)	309
17.14.13	enumerate(iterable, [start])	309
17.14.14	eval(expression, [globals, locals])	309
17.14.15	exec(object, [globals, locals])	310
17.14.16	filter(function, iterable)	310
17.14.17	float([x])	311
17.14.18	format(value, [format_spec])	311
17.14.19	frozenset([iterable])	311
17.14.20	globals()	312
17.14.21	hash(object)	312
17.14.22	help([object])	313
17.14.23	hex(x)	313
17.14.24	id(object)	313
17.14.25	input([prompt])	314
17.14.26	int([x, base])	314

17.14.27	len(s)	315
17.14.28	list([sequence])	315
17.14.29	locals()	315
17.14.30	map(function, [*iterable])	316
17.14.31	max(iterable, {default, key}), max(arg1, arg2, [*args], {key})	317
17.14.32	min(iterable, {default, key}), min(arg1, arg2, [*args], {key})	318
17.14.33	oct(x)	318
17.14.34	ord(c)	318
17.14.35	pow(x, y, [z])	318
17.14.36	print([*objects], {sep, end, file, flush})	318
17.14.37	range([start], stop, [step])	319
17.14.38	repr(object)	320
17.14.39	reversed(sequence)	320
17.14.40	round(x, [n])	321
17.14.41	set([iterable])	321
17.14.42	sorted(iterable, {key, reverse})	321
17.14.43	str([object, encoding, errors])	322
17.14.44	sum(iterable, [start])	323
17.14.45	tuple([iterable])	323
17.14.46	type(object)	323
17.14.47	zip([*iterables], {strict})	324

18 Modules and Packages 325

18.1	Importing Global Modules	326
18.2	Local Modules	328
18.2.1	Name Conflicts	329
18.2.2	Module-Internal References	330
18.2.3	Executing Modules	330
18.3	Packages	331
18.3.1	Importing All Modules of a Package	333
18.3.2	Namespace Packages	333
18.3.3	Relative Import Statements	334
18.4	The importlib Package	335
18.4.1	Importing Modules and Packages	335
18.4.2	Changing the Import Behavior	335
18.5	Planned Language Elements	338

19 Object-Oriented Programming 341

19.1 Example: A Non-Object-Oriented Account	341
19.1.1 Creating a New Account	342
19.1.2 Transferring Money	342
19.1.3 Depositing and Withdrawing Money	343
19.1.4 Viewing the Account Balance	344
19.2 Classes	346
19.2.1 Defining Methods	347
19.2.2 The Constructor	348
19.2.3 Attributes	349
19.2.4 Example: An Object-Oriented Account	349
19.3 Inheritance	351
19.3.1 A Simple Example	352
19.3.2 Overriding Methods	353
19.3.3 Example: Checking Account with Daily Turnover	355
19.3.4 Outlook	363
19.4 Multiple Inheritance	363
19.5 Property Attributes	365
19.5.1 Setters and Getters	365
19.5.2 Defining Property Attributes	366
19.6 Static Methods	367
19.7 Class Methods	369
19.8 Class Attributes	370
19.9 Built-in Functions for Object-Oriented Programming	370
19.9.1 Functions for Managing the Attributes of an Instance	371
19.9.2 Functions for Information about the Class Hierarchy	372
19.10 Inheriting Built-In Data Types	373
19.11 Magic Methods and Magic Attributes	375
19.11.1 General Magic Methods	375
19.11.2 Overloading Operators	382
19.11.3 Emulating Data Types: Duck Typing	388
19.12 Data Classes	393
19.12.1 Tuples and Lists	393
19.12.2 Dictionaries	394
19.12.3 Named Tuples	394
19.12.4 Mutable Data Classes	395

19.12.5	Immutable Data Classes	396
19.12.6	Default Values in Data Classes	396

20 Exception Handling 399

20.1	Exceptions	399
20.1.1	Built-In Exceptions	400
20.1.2	Raising an Exception	401
20.1.3	Handling an Exception	401
20.1.4	Custom Exceptions	406
20.1.5	Re-Raising an Exception	408
20.1.6	Exception Chaining	410
20.2	Assertions	411
20.3	Warnings	412

21 Generators and Iterators 415

21.1	Generators	415
21.1.1	Subgenerators	418
21.1.2	Generator Expressions	421
21.2	Iterators	422
21.2.1	The Iterator Protocol	422
21.2.2	Example: The Fibonacci Sequence	423
21.2.3	Example: The Golden Ratio	424
21.2.4	A Generator for the Implementation of <code>__iter__</code>	424
21.2.5	Using Iterators	425
21.2.6	Multiple Iterators for the Same Instance	428
21.2.7	Disadvantages of Iterators Compared to Direct Access via Indexes	430
21.2.8	Alternative Definition for Iterable Objects	430
21.2.9	Function Iterators	431
21.3	Special Generators: <code>itertools</code>	432
21.3.1	<code>accumulate(iterable, [func])</code>	433
21.3.2	<code>chain([*iterables])</code>	433
21.3.3	<code>combinations(iterable, r)</code>	434
21.3.4	<code>combinations_with_replacement(iterable, r)</code>	434
21.3.5	<code>compress(data, selectors)</code>	435
21.3.6	<code>count([start, step])</code>	435

21.3.7	cycle(iterable)	436
21.3.8	dropwhile(predicate, iterable)	436
21.3.9	filterfalse(predicate, iterable)	436
21.3.10	groupby(iterable, [key])	437
21.3.11	islice(iterable, [start], stop, [step])	437
21.3.12	permutations(iterable, [r])	438
21.3.13	product([*iterables], [repeat])	438
21.3.14	repeat(object, [times])	439
21.3.15	starmap(function, iterable)	439
21.3.16	takewhile(predicate, iterable)	439
21.3.17	tee(iterable, [n])	439
21.3.18	zip_longest([*iterables], {fillvalue})	440

22 Context Manager 441

22.1	The with Statement	441
22.1.1	__enter__(self)	443
22.1.2	__exit__(self, exc_type, exc_value, traceback)	444
22.2	Helper Functions for with Contexts: contextlib	444
22.2.1	Dynamically Assembled Context Combinations - ExitStack	444
22.2.2	Suppressing Certain Exception Types	445
22.2.3	Redirecting the Standard Output Stream	445
22.2.4	Optional Contexts	446
22.2.5	Simple Functions as Context Manager	447

23 Decorators 449

23.1	Function Decorators	449
23.1.1	Decorating Functions and Methods	451
23.1.2	Name and Docstring after Applying a Decorator	451
23.1.3	Nested Decorators	452
23.1.4	Example: A Cache Decorator	452
23.2	Class Decorators	454
23.3	The functools Module	455
23.3.1	Simplifying Function Interfaces	455
23.3.2	Simplifying Method Interfaces	457

23.3.3	Caches	457
23.3.4	Completing Orderings of Custom Classes	459
23.3.5	Overloading Functions	459

24 Annotations for Static Type Checking 463

24.1	Annotations	464
24.1.1	Annotating Functions and Methods	465
24.1.2	Annotating Variables and Attributes	466
24.1.3	Accessing Annotations at Runtime	468
24.1.4	When are Annotations Evaluated?	470
24.2	Type Hints: The typing Module	471
24.2.1	Valid Type Hints	472
24.2.2	Container Types	472
24.2.3	Abstract Container Types	473
24.2.4	Type Aliases	474
24.2.5	Type Unions and Optional Values	474
24.2.6	Type Variables	475
24.3	Static Type Checking in Python: mypy	476
24.3.1	Installation	476
24.3.2	Example	477

25 Structural Pattern Matching 479

25.1	The match Statement	479
25.2	Pattern Types in the case Statement	480
25.2.1	Literal and Value Patterns	481
25.2.2	OR Pattern	481
25.2.3	Patterns with Type Checking	482
25.2.4	Specifying Conditions for Matches	483
25.2.5	Grouping Subpatterns	483
25.2.6	Capture and Wildcard Patterns	484
25.2.7	Sequence Patterns	486
25.2.8	Mapping Patterns	488
25.2.9	Patterns for Objects and Their Attribute Values	490

Part IV The Standard Library

26 Mathematics 497

- 26.1 Mathematical Functions: `math`, `cmath`** 497
 - 26.1.1 General Mathematical Functions 498
 - 26.1.2 Exponential and Logarithm Functions 501
 - 26.1.3 Trigonometric and Hyperbolic Functions 501
 - 26.1.4 Distances and Norms 502
 - 26.1.5 Converting Angles 502
 - 26.1.6 Representations of Complex Numbers 502
- 26.2 Random Number Generator: `random`** 503
 - 26.2.1 Saving and Loading the Random State 504
 - 26.2.2 Generating Random Integers 504
 - 26.2.3 Generating Random Floats 505
 - 26.2.4 Random Operations on Sequences 505
 - 26.2.5 `SystemRandom([seed])` 507
- 26.3 Statistical Calculations: `statistics`** 507
- 26.4 Intuitive Decimal Numbers: `decimal`** 509
 - 26.4.1 Using the Data Type 509
 - 26.4.2 Nonnumeric Values 512
 - 26.4.3 The Context Object 513
- 26.5 Hash Functions: `hashlib`** 514
 - 26.5.1 Using the Module 516
 - 26.5.2 Other Hash Algorithms 517
 - 26.5.3 Comparing Large Files 517
 - 26.5.4 Passwords 518

27 Screen Outputs and Logging 521

- 27.1 Formatted Output of Complex Objects: `pprint`** 521
- 27.2 Log Files: `logging`** 523
 - 27.2.1 Customizing the Message Format 525
 - 27.2.2 Logging Handlers 527

28 Regular Expressions 529

28.1 Syntax of Regular Expressions	529
28.1.1 Any Character	530
28.1.2 Character Classes	530
28.1.3 Quantifiers	531
28.1.4 Predefined Character Classes	533
28.1.5 Other Special Characters	534
28.1.6 Nongreedy Quantifiers	535
28.1.7 Groups	536
28.1.8 Alternatives	536
28.1.9 Extensions	537
28.2 Using the re Module	539
28.2.1 Searching	540
28.2.2 Matching	540
28.2.3 Splitting a String	541
28.2.4 Replacing Parts of a String	541
28.2.5 Replacing Problem Characters	542
28.2.6 Compiling a Regular Expression	542
28.2.7 Flags	543
28.2.8 The Match Object	544
28.3 A Simple Sample Program: Searching	546
28.4 A More Complex Sample Program: Matching	547
28.5 Comments in Regular Expressions	550

29 Interface to Operating System and Runtime Environment 553

29.1 Operating System Functionality: os	553
29.1.1 environ	554
29.1.2 getpid()	554
29.1.3 cpu_count()	554
29.1.4 system(cmd)	554
29.1.5 popen(command, [mode, buffering])	555
29.2 Accessing the Runtime Environment: sys	555
29.2.1 Command Line Parameters	556
29.2.2 Default Paths	556
29.2.3 Standard Input/Output Streams	556

29.2.4	Exiting the Program	556
29.2.5	Details of the Python Version	557
29.2.6	Operating System Details	557
29.2.7	Hooks	559
29.3	Command Line Parameters: argparse	561
29.3.1	Calculator: A Simple Example	562
29.3.2	A More Complex Example	566

30 File System 569

30.1	Accessing the File System: os	569
30.1.1	access(path, mode)	570
30.1.2	chmod(path, mode)	571
30.1.3	listdir([path])	571
30.1.4	mkdir(path, [mode]) and makedirs(path, [mode])	572
30.1.5	remove(path)	572
30.1.6	removedirs(path)	572
30.1.7	rename(src, dst) and renames(old, new)	573
30.1.8	walk(top, [topdown, onerror])	573
30.2	File Paths: os.path	575
30.2.1	abspath(path)	576
30.2.2	basename(path)	577
30.2.3	commonprefix(list)	577
30.2.4	dirname(path)	577
30.2.5	join(path, *paths)	578
30.2.6	normcase(path)	578
30.2.7	split(path)	578
30.2.8	splitdrive(path)	579
30.2.9	splittext(path)	579
30.3	Accessing the File System: shutil	579
30.3.1	Directory and File Operations	581
30.3.2	Archive Operations	582
30.4	Temporary Files: tempfile	585
30.4.1	TemporaryFile([mode, [bufsize, suffix, prefix, dir])	585
30.4.2	tempfile.TemporaryDirectory([suffix, prefix, dir])	586

31 Parallel Programming 587

31.1 Processes, Multitasking, and Threads	587
31.1.1 The Lightweights among the Processes: Threads	588
31.1.2 Threads or Processes?	590
31.1.3 Cooperative Multitasking: A Third Way	590
31.2 Python's Interfaces for Parallelization	591
31.3 The Abstract Interface: concurrent.futures	592
31.3.1 An Example with a futures.ThreadPoolExecutor	593
31.3.2 Executor Instances as Context Managers	595
31.3.3 Using futures.ProcessPoolExecutor	595
31.3.4 Managing the Tasks of an Executor	596
31.4 The Flexible Interface: threading and multiprocessing	602
31.4.1 Threads in Python: threading	603
31.4.2 Processes in Python: multiprocessing	611
31.5 Cooperative Multitasking	613
31.5.1 Cooperative Functions: Coroutines	613
31.5.2 Awaitable Objects	614
31.5.3 The Cooperation of Coroutines: Tasks	615
31.5.4 A Cooperative Web Crawler	618
31.5.5 Blocking Operations in Coroutines	625
31.5.6 Other Asynchronous Language Features	627
31.6 Conclusion: Which Interface Is the Right One?	629
31.6.1 Is Cooperative Multitasking an Option?	629
31.6.2 Abstraction or Flexibility?	630
31.6.3 Threads or Processes?	630

32 Data Storage 631

32.1 XML	631
32.1.1 ElementTree	633
32.1.2 Simple API for XML	640
32.2 Databases	643
32.2.1 The Built-In Database in Python: sqlite3	646
32.3 Compressed Files and Archives	661
32.3.1 gzip.open(filename, [mode, compresslevel])	661
32.3.2 Other Modules for Accessing Compressed Data	662

32.4	Serializing Instances: pickle	662
32.4.1	Functional Interface	663
32.4.2	Object-Oriented Interface	665
32.5	The JSON Data Exchange Format: json	665
32.6	The CSV Table Format: csv	667
32.6.1	Reading Data from a CSV File with reader Objects	668
32.6.2	Using Custom Dialects: Dialect Objects	670

33 Network Communication 673

33.1	Socket API	674
33.1.1	Client-Server Systems	675
33.1.2	UDP	677
33.1.3	TCP	678
33.1.4	Blocking and Nonblocking Sockets	680
33.1.5	Creating a Socket	681
33.1.6	The Socket Class	682
33.1.7	Network Byte Order	685
33.1.8	Multiplexing Servers: selectors	686
33.1.9	Object-Oriented Server Development: socketserver	688
33.2	XML-RPC	690
33.2.1	The Server	691
33.2.2	The Client	694
33.2.3	Multicall	696
33.2.4	Limitations	697

34 Accessing Resources on the Internet 701

34.1	Protocols	701
34.1.1	Hypertext Transfer Protocol	701
34.1.2	File Transfer Protocol	701
34.2	Solutions	702
34.2.1	Outdated Solutions for Python 2	702
34.2.2	Solutions in the Standard Library	702
34.2.3	Third-Party Solutions	702

34.3	The Easy Way: requests	703
34.3.1	Simple Requests via GET and POST	703
34.3.2	Web APIs	704
34.4	URLs: urllib	705
34.4.1	Accessing Remote Resources: urllib.request	706
34.4.2	Reading and Processing URLs: urllib.parse	710
34.5	FTP: ftplib	713
34.5.1	Connecting to an FTP Server	714
34.5.2	Executing FTP commands	715
34.5.3	Working with Files and Directories	715
34.5.4	Transferring Files	716

35 Email 721

35.1	SMTP: smtplib	721
35.1.1	SMTP([host, port, local_hostname, timeout, source_address])	722
35.1.2	Establishing and Terminating a Connection	722
35.1.3	Sending an Email	723
35.1.4	Example	724
35.2	POP3: poplib	724
35.2.1	POP3(host, [port, timeout])	725
35.2.2	Establishing and Terminating a Connection	725
35.2.3	Listing Existing Emails	726
35.2.4	Retrieving and Deleting Emails	727
35.2.5	Example	727
35.3	IMAP4: imaplib	728
35.3.1	IMAP4([host, port, timeout])	729
35.3.2	Establishing and Terminating a Connection	730
35.3.3	Finding and Selecting a Mailbox	730
35.3.4	Operations with Mailboxes	731
35.3.5	Searching Emails	731
35.3.6	Retrieving Emails	732
35.3.7	Example	733
35.4	Creating Complex Emails: email	734
35.4.1	Creating a Simple Email	734
35.4.2	Creating an Email with Attachments	735
35.4.3	Reading an Email	737

36 Debugging and Quality Assurance 739

36.1 The Debugger	739
36.2 Automated Testing	741
36.2.1 Test Cases in Docstrings: doctest	742
36.2.2 Unit Tests: unittest	746
36.3 Analyzing the Runtime Performance	749
36.3.1 Runtime Measurement: timeit	749
36.3.2 Profiling: cProfile	752
36.3.3 Tracing: trace	756

37 Documentation 759

37.1 Docstrings	759
37.2 Automatically Generated Documentation: pydoc	761

Part V Advanced Topics

38 Distributing Python Projects 765

38.1 A History of Distributions in Python	765
38.1.1 The Classic Approach: distutils	766
38.1.2 The New Standard: setuptools	766
38.1.3 The Package Index: PyPI	766
38.2 Creating Distributions: setuptools	767
38.2.1 Installation	767
38.2.2 Writing the Module	767
38.2.3 The Installation Script	768
38.2.4 Creating a Source Distribution	773
38.2.5 Creating a Binary Distribution	773
38.2.6 Installing Distributions	774
38.3 Creating EXE files: cx_Freeze	775
38.3.1 Installation	775
38.3.2 Usage	775

38.4	Package Manager	776
38.4.1	The Python Package Manager: pip	777
38.4.2	The conda Package Manager	778
38.5	Localizing Programs: gettext	781
38.5.1	Example of Using gettext	781
38.5.2	Creating the Language Compilation	783

39 Virtual Environments 785

39.1	Using Virtual Environments: venv	786
39.1.1	Activating a Virtual Environment	786
39.1.2	Working in a Virtual Environment	786
39.1.3	Deactivating a Virtual Environment	787
39.2	Virtual Environments in Anaconda	787

40 Alternative Interpreters and Compilers 789

40.1	Just-in-Time Compilation: PyPy	789
40.1.1	Installation and Use	789
40.1.2	Example	790
40.2	Numba	790
40.2.1	Installation	791
40.2.2	Example	791
40.3	Connecting to C and C++: Cython	793
40.3.1	Installation	793
40.3.2	The Functionality of Cython	794
40.3.3	Compiling a Cython Program	794
40.3.4	A Cython Program with Static Typing	796
40.3.5	Using a C Library	797
40.4	The Interactive Python Shell: IPython	799
40.4.1	Installation	799
40.4.2	The Interactive Shell	799
40.4.3	The Jupyter Notebook	802

41 Graphical User Interfaces

805

41.1	Toolkits	805
41.1.1	Tkinter (Tk)	805
41.1.2	PyGObject (GTK)	806
41.1.3	Qt for Python (Qt)	806
41.1.4	wxPython (wxWidgets)	807
41.2	Introduction to tkinter	807
41.2.1	A Simple Example	807
41.2.2	Control Variables	810
41.2.3	The Packer	811
41.2.4	Events	815
41.2.5	Widgets	821
41.2.6	Drawings: The Canvas Widget	839
41.2.7	Other Modules	846
41.3	Introduction to PySide6	850
41.3.1	Installation	850
41.3.2	Basic Concepts of Qt	850
41.3.3	Development Process	852
41.4	Signals and Slots	859
41.5	Important Widgets	861
41.5.1	QCheckBox	862
41.5.2	QComboBox	862
41.5.3	QDateEdit, QTimeEdit, and QDateTimeEdit	863
41.5.4	QDialog	863
41.5.5	QLineEdit	864
41.5.6	QListWidget and QListView	864
41.5.7	QProgressBar	865
41.5.8	QPushButton	865
41.5.9	QRadioButton	865
41.5.10	QSlider and QDial	866
41.5.11	QTextEdit	866
41.5.12	QWidget	867
41.6	Drawing Functionality	868
41.6.1	Tools	868
41.6.2	Coordinate System	870
41.6.3	Simple Shapes	871
41.6.4	Images	873
41.6.5	Text	874
41.6.6	Eye Candy	876

41.7	Model-View Architecture	879
41.7.1	Sample Project: An Address Book	880
41.7.2	Selecting Entries	888
41.7.3	Editing Entries	890

42 Python as a Server-Side Programming Language on the Web: An Introduction to Django 893

42.1	Concepts and Features of Django	894
42.2	Installing Django	895
42.3	Creating a New Django Project	896
42.3.1	The Development Web Server	897
42.3.2	Configuring the Project	898
42.4	Creating an Application	900
42.4.1	Importing the Application into the Project	901
42.4.2	Defining a Model	901
42.4.3	Relationships between Models	902
42.4.4	Transferring the Model to the Database	903
42.4.5	The Model API	904
42.4.6	The Project Gets a Face	909
42.4.7	Django's Template System	915
42.4.8	Processing Form Data	926
42.4.9	Django's Admin Control Panel	930

43 Scientific Computing and Data Science 935

43.1	Installation	936
43.2	The Model Program	936
43.2.1	Importing numpy, scipy, and matplotlib	937
43.2.2	Vectorization and the numpy.ndarray Data Type	938
43.2.3	Visualizing Data Using matplotlib.pyplot	942
43.3	Overview of the numpy and scipy Modules	944
43.3.1	Overview of the numpy.ndarray Data Type	944
43.3.2	Overview of scipy	952
43.4	An Introduction to Data Analysis with pandas	953
43.4.1	The DataFrame Object	954

43.4.2	Selective Data Access	955
43.4.3	Deleting Rows and Columns	961
43.4.4	Inserting Rows and Columns	961
43.4.5	Logical Expressions on Data Records	962
43.4.6	Manipulating Data Records	963
43.4.7	Input and Output	965
43.4.8	Visualization	966

44 Inside Knowledge 969

44.1	Opening URLs in the Default Browser: webbrowser	969
44.2	Interpreting Binary Data: struct	969
44.3	Hidden Password Entry	971
44.3.1	getpass([prompt, stream])	971
44.3.2	getpass.getuser()	972
44.4	Command Line Interpreter	972
44.5	File Interface for Strings: io.StringIO	975
44.6	Generators as Consumers	976
44.6.1	A Decorator for Consuming Generator Functions	978
44.6.2	Triggering Exceptions in a Generator	978
44.6.3	A Pipeline as a Chain of Consuming Generator Functions	979
44.7	Copying Instances: copy	981
44.8	Image Processing: Pillow	984
44.8.1	Installation	984
44.8.2	Loading and Saving Image Files	984
44.8.3	Accessing Individual Pixels	985
44.8.4	Manipulating Images	986
44.8.5	Interoperability	992

45 From Python 2 to Python 3 993

45.1	The Main Differences	996
45.1.1	Input/Output	996
45.1.2	Iterators	997
45.1.3	Strings	998
45.1.4	Integers	999

45.1.5	Exception Handling	999
45.1.6	Standard Library	1000
45.2	Automatic Conversion	1001

Appendices 1005

A	Appendix	1005
B	The Authors	1017

Index	1019
-------------	------