

# Python®

## Notes for Professionals

### Chapter 2: Python Data Types

Data types are nothing but variable you used to reserve some space in memory. Python variable explicit declaration to reserve memory space. The declaration happens automatically when you create a variable.

#### Section 2.1: String Data Type

String are identified as a contiguous set of characters represented in the quotation marks. Pairs of single or double quotes. Strings are immutable sequence data type, i.e. each time you create a string, completely new string object is created.

```
s_str = "Hello World"
print(s_str) # output will be whole string: Hello World
print(s_str[0]) # output will be first character: H
print(s_str[0:5]) # output will be first five characters: Hello
```

#### Section 2.2: Set Data Types

Sets are unordered collections of unique objects. There are two types of sets:

1. Sets - They are mutable and new elements can be added once sets are defined.

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket) # output: {'apple', 'orange', 'pear', 'banana'}
# 'apple' and 'orange' are removed
# 'banana' and 'pear' are added
a = set('abracadabra') # a set with 11 distinct elements
print(a)
# {'a', 'b', 'r', 'c', 'd'}
a.add('x')
print(a)
# {'a', 'b', 'r', 'c', 'd', 'x'}
```

2. Frozen Sets - They are immutable and new elements cannot be added after.

```
b = frozenset(['a', 'b', 'c', 'd', 'e', 'f'])
print(b)
# frozenset({'a', 'b', 'c', 'd', 'e', 'f'})
cities = frozenset(['Frankfurt', 'Basel', 'Freiburg'])
print(cities)
# frozenset({'Frankfurt', 'Basel', 'Freiburg'})
```

#### Section 2.3: Numbers data type

Numbers have four types in Python: int, float, complex, and long.

```
int_num = 10 # int value
float_num = 10.2 # float value
complex_num = 1+1j # complex value
long_num = 1234567L # long value
```

### Chapter 30: String Formatting

When storing and transforming data for humans to see, string formatting can become very important. Python offers a wide variety of string formatting methods which are outlined in this topic.

#### Section 30.1: Basics of String Formatting

```
foo = 1
bar = 'bar'
baz = 3.14
```

You can use `str.format` to format output. Bracket pairs are replaced with arguments in the order in which the arguments are passed:

```
print('{}, {} and {}'.format(foo, bar, baz))
# Out: '1, bar and 3.14'
```

Indices can also be specified inside the brackets. The numbers correspond to indexes of the arguments passed to the `str.format` function (0-based).

```
print('{0}, {1}, {2}, and {3}'.format(foo, bar, baz))
# Out: '1, bar, 3.14, and bar'
print('{0}, {1}, {2}, and {3}'.format(foo, bar, baz))
# Out: '1, bar, 3.14, and bar'
```

Named arguments can be also used:

```
print('X value is: {x_val}, Y value is: {y_val}'.format(x_val=2, y_val=3))
# Out: 'X value is: 2, Y value is: 3'
```

Object attributes can be referenced when passed into `str.format`:

```
class AssignValue(object):
    def __init__(self, value):
        self.value = value
my_value = AssignValue(5)
print('My value is: {0.value}'.format(my_value)) # '0' is optional
# Out: 'My value is: 5'
```

Dictionary keys can be used as well:

```
my_dict = {'key': 5, 'other_key': 7}
print('My other key is: {0[other_key]}'.format(my_dict)) # '0' is optional
# Out: 'My other key is: 7'
```

Same applies to list and tuple indices:

```
my_list = ['zero', 'one', 'two']
print('2nd element is: {0[2]}'.format(my_list)) # '0' is optional
# Out: '2nd element is: two'
```

Note: In addition to `str.format`, Python also provides the modulo operator `%` also known as the `printf` formatting or interpolation operator (see [PEP 3101](#)) for formatting strings. `str.format` is a success.

Python® Notes for Professionals

### Chapter 46: Random module

#### Section 46.1: Creating a random user password

In order to create a random user password we can use the symbols provided in the `string` module. Specifically, punctuation for punctuation symbols, `ascii_letters` for letters and digits for digits.

We can then combine all these symbols in a name named `symbols`:

```
symbols = ascii_letters + digits + punctuation
```

Remove either of these to create a pool of symbols with fewer elements.

After this, we can use `random.SystemRandom` to generate a password. For a 10 length password:

```
secure_random = random.SystemRandom()
password = ''.join(secure_random.choice(symbols) for i in range(10))
print(password) # '4qj07fme'
```

Note that other routines made immediately available by the `random` module — such as `random.choice`, `random.randrange`, etc. — are unsuitable for cryptographic purposes.

Behind the curtains, these routines use the `Mersenne Twister PRNG`, which does not satisfy the requirements of a `CSPRNG`. Thus, in particular, you should not use any of them to generate passwords you plan to use. Always use an instance of `SystemRandom` as shown above.

Python 3.x version = 3.6

Starting from Python 3.6, the `secrets` module is available, which exposes cryptographically safe functionality.

Quoting the [official documentation](#), to generate 'a ten-character alphanumeric password with at least one lowercase character, at least one uppercase character, and at least three digits,' you could:

```
import string
alphabet = string.ascii_letters + string.digits
while True:
    password = ''.join(choice(alphabet) for i in range(10))
    if any(c.islower() for c in password) and any(c.isupper() for c in password) and sum(c.isdigit() for c in password) > 3:
        break
```

#### Section 46.2: Create cryptographically secure random numbers

By default the Python random module use the `Mersenne Twister PRNG` to generate random numbers, which, although suitable in domains like simulations, fails to meet security requirements in more demanding environments.

In order to create a cryptographically secure pseudorandom number, one can use `SystemRandom` which, by using `os.urandom`, is able to act as a cryptographically secure pseudorandom number generator. [CSPRNGs](#).

Python® Notes for Professionals

**700+ pages**  
of professional hints and tricks

# Contents

<b>About</b>	1
<b>Chapter 1: Getting started with Python Language</b>	2
<a href="#">Section 1.1: Getting Started</a>	2
<a href="#">Section 1.2: Creating variables and assigning values</a>	6
<a href="#">Section 1.3: Block Indentation</a>	10
<a href="#">Section 1.4: Datatypes</a>	11
<a href="#">Section 1.5: Collection Types</a>	16
<a href="#">Section 1.6: IDLE - Python GUI</a>	20
<a href="#">Section 1.7: User Input</a>	21
<a href="#">Section 1.8: Built in Modules and Functions</a>	22
<a href="#">Section 1.9: Creating a module</a>	25
<a href="#">Section 1.10: Installation of Python 2.7.x and 3.x</a>	26
<a href="#">Section 1.11: String function - str() and repr()</a>	29
<a href="#">Section 1.12: Installing external modules using pip</a>	30
<a href="#">Section 1.13: Help Utility</a>	31
<b>Chapter 2: Python Data Types</b>	33
<a href="#">Section 2.1: String Data Type</a>	33
<a href="#">Section 2.2: Set Data Types</a>	33
<a href="#">Section 2.3: Numbers data type</a>	33
<a href="#">Section 2.4: List Data Type</a>	34
<a href="#">Section 2.5: Dictionary Data Type</a>	34
<a href="#">Section 2.6: Tuple Data Type</a>	34
<b>Chapter 3: Indentation</b>	35
<a href="#">Section 3.1: Simple example</a>	35
<a href="#">Section 3.2: How Indentation is Parsed</a>	35
<a href="#">Section 3.3: Indentation Errors</a>	36
<b>Chapter 4: Comments and Documentation</b>	37
<a href="#">Section 4.1: Single line, inline and multiline comments</a>	37
<a href="#">Section 4.2: Programmatically accessing docstrings</a>	37
<a href="#">Section 4.3: Write documentation using docstrings</a>	38
<b>Chapter 5: Date and Time</b>	41
<a href="#">Section 5.1: Parsing a string into a timezone aware datetime object</a>	41
<a href="#">Section 5.2: Constructing timezone-aware datetimes</a>	41
<a href="#">Section 5.3: Computing time differences</a>	43
<a href="#">Section 5.4: Basic datetime objects usage</a>	43
<a href="#">Section 5.5: Switching between time zones</a>	44
<a href="#">Section 5.6: Simple date arithmetic</a>	44
<a href="#">Section 5.7: Converting timestamp to datetime</a>	45
<a href="#">Section 5.8: Subtracting months from a date accurately</a>	45
<a href="#">Section 5.9: Parsing an arbitrary ISO 8601 timestamp with minimal libraries</a>	45
<a href="#">Section 5.10: Get an ISO 8601 timestamp</a>	46
<a href="#">Section 5.11: Parsing a string with a short time zone name into a timezone aware datetime object</a>	46
<a href="#">Section 5.12: Fuzzy datetime parsing (extracting datetime out of a text)</a>	47
<a href="#">Section 5.13: Iterate over dates</a>	48
<b>Chapter 6: Date Formatting</b>	49
<a href="#">Section 6.1: Time between two date-times</a>	49
<a href="#">Section 6.2: Outputting datetime object to string</a>	49

Section 6.3: <a href="#">Parsing string to datetime object</a>	49
<b>Chapter 7: Enum</b>	50
Section 7.1: <a href="#">Creating an enum (Python 2.4 through 3.3)</a>	50
Section 7.2: <a href="#">Iteration</a>	50
<b>Chapter 8: Set</b>	51
Section 8.1: <a href="#">Operations on sets</a>	51
Section 8.2: <a href="#">Get the unique elements of a list</a>	52
Section 8.3: <a href="#">Set of Sets</a>	52
Section 8.4: <a href="#">Set Operations using Methods and Builtins</a>	52
Section 8.5: <a href="#">Sets versus multisets</a>	54
<b>Chapter 9: List comprehensions</b>	56
Section 9.1: <a href="#">List Comprehensions</a>	56
Section 9.2: <a href="#">Avoid repetitive and expensive operations using conditional clause</a>	58
Section 9.3: <a href="#">Dictionary Comprehensions</a>	60
Section 9.4: <a href="#">Generator Expressions</a>	61
Section 9.5: <a href="#">Set Comprehensions</a>	63
Section 9.6: <a href="#">Comprehensions involving tuples</a>	63
Section 9.7: <a href="#">Counting Occurrences Using Comprehension</a>	64
Section 9.8: <a href="#">Changing Types in a List</a>	64
<b>Chapter 10: Simple Mathematical Operators</b>	66
Section 10.1: <a href="#">Division</a>	66
Section 10.2: <a href="#">Addition</a>	67
Section 10.3: <a href="#">Exponentiation</a>	68
Section 10.4: <a href="#">Trigonometric Functions</a>	69
Section 10.5: <a href="#">Inplace Operations</a>	69
Section 10.6: <a href="#">Subtraction</a>	70
Section 10.7: <a href="#">Multiplication</a>	70
Section 10.8: <a href="#">Logarithms</a>	71
Section 10.9: <a href="#">Modulus</a>	71
<b>Chapter 11: Bitwise Operators</b>	72
Section 11.1: <a href="#">Bitwise NOT</a>	72
Section 11.2: <a href="#">Bitwise XOR (Exclusive OR)</a>	73
Section 11.3: <a href="#">Bitwise AND</a>	74
Section 11.4: <a href="#">Bitwise OR</a>	74
Section 11.5: <a href="#">Bitwise Left Shift</a>	74
Section 11.6: <a href="#">Bitwise Right Shift</a>	75
Section 11.7: <a href="#">Inplace Operations</a>	75
<b>Chapter 12: Boolean Operators</b>	76
Section 12.1: <a href="#">`and` and `or` are not guaranteed to return a boolean</a>	76
Section 12.2: <a href="#">A simple example</a>	76
Section 12.3: <a href="#">Short-circuit evaluation</a>	76
Section 12.4: <a href="#">and</a>	77
Section 12.5: <a href="#">or</a>	77
Section 12.6: <a href="#">not</a>	78
<b>Chapter 13: Operator Precedence</b>	79
Section 13.1: <a href="#">Simple Operator Precedence Examples in python</a>	79
<b>Chapter 14: Filter</b>	80
Section 14.1: <a href="#">Basic use of filter</a>	80
Section 14.2: <a href="#">Filter without function</a>	80

Section 14.3: Filter as short-circuit check .....	81
Section 14.4: Complementary function: filterfalse, ifilterfalse .....	81
<b>Chapter 15: Arrays</b> .....	83
Section 15.1: Access individual elements through indexes .....	83
Section 15.2: Basic Introduction to Arrays .....	83
Section 15.3: Append any value to the array using append() method .....	84
Section 15.4: Insert value in an array using insert() method .....	84
Section 15.5: Extend python array using extend() method .....	84
Section 15.6: Add items from list into array using fromlist() method .....	84
Section 15.7: Remove any array element using remove() method .....	85
Section 15.8: Remove last array element using pop() method .....	85
Section 15.9: Fetch any element through its index using index() method .....	85
Section 15.10: Reverse a python array using reverse() method .....	85
Section 15.11: Get array buffer information through buffer_info() method .....	86
Section 15.12: Check for number of occurrences of an element using count() method .....	86
Section 15.13: Convert array to string using tostring() method .....	86
Section 15.14: Convert array to a python list with same elements using tolist() method .....	86
Section 15.15: Append a string to char array using fromstring() method .....	86
<b>Chapter 16: Dictionary</b> .....	87
Section 16.1: Introduction to Dictionary .....	87
Section 16.2: Avoiding KeyError Exceptions .....	88
Section 16.3: Iterating Over a Dictionary .....	88
Section 16.4: Dictionary with default values .....	89
Section 16.5: Merging dictionaries .....	90
Section 16.6: Accessing keys and values .....	90
Section 16.7: Accessing values of a dictionary .....	91
Section 16.8: Creating a dictionary .....	91
Section 16.9: Creating an ordered dictionary .....	92
Section 16.10: Unpacking dictionaries using the ** operator .....	92
Section 16.11: The trailing comma .....	93
Section 16.12: The dict() constructor .....	93
Section 16.13: Dictionaries Example .....	93
Section 16.14: All combinations of dictionary values .....	94
<b>Chapter 17: List</b> .....	95
Section 17.1: List methods and supported operators .....	95
Section 17.2: Accessing list values .....	100
Section 17.3: Checking if list is empty .....	101
Section 17.4: Iterating over a list .....	101
Section 17.5: Checking whether an item is in a list .....	102
Section 17.6: Any and All .....	102
Section 17.7: Reversing list elements .....	103
Section 17.8: Concatenate and Merge lists .....	103
Section 17.9: Length of a list .....	104
Section 17.10: Remove duplicate values in list .....	104
Section 17.11: Comparison of lists .....	105
Section 17.12: Accessing values in nested list .....	105
Section 17.13: Initializing a List to a Fixed Number of Elements .....	106
<b>Chapter 18: List slicing (selecting parts of lists)</b> .....	108
Section 18.1: Using the third "step" argument .....	108
Section 18.2: Selecting a sublist from a list .....	108

Section 18.3: Reversing a list with slicing .....	108
Section 18.4: Shifting a list using slicing .....	108
<b>Chapter 19: Linked lists</b> .....	110
Section 19.1: Single linked list example .....	110
<b>Chapter 20: Linked List Node</b> .....	114
Section 20.1: Write a simple Linked List Node in python .....	114
<b>Chapter 21: Tuple</b> .....	115
Section 21.1: Tuple .....	115
Section 21.2: Tuples are immutable .....	116
Section 21.3: Packing and Unpacking Tuples .....	116
Section 21.4: Built-in Tuple Functions .....	117
Section 21.5: Tuple Are Element-wise Hashable and Equatable .....	118
Section 21.6: Indexing Tuples .....	119
Section 21.7: Reversing Elements .....	119
<b>Chapter 22: Functions</b> .....	120
Section 22.1: Defining and calling simple functions .....	120
Section 22.2: Defining a function with an arbitrary number of arguments .....	121
Section 22.3: Lambda (Inline/Anonymous) Functions .....	124
Section 22.4: Defining a function with optional arguments .....	126
Section 22.5: Defining a function with optional mutable arguments .....	127
Section 22.6: Argument passing and mutability .....	128
Section 22.7: Returning values from functions .....	129
Section 22.8: Closure .....	129
Section 22.9: Forcing the use of named parameters .....	130
Section 22.10: Nested functions .....	131
Section 22.11: Recursion limit .....	131
Section 22.12: Recursive Lambda using assigned variable .....	132
Section 22.13: Recursive functions .....	132
Section 22.14: Defining a function with arguments .....	133
Section 22.15: Iterable and dictionary unpacking .....	133
Section 22.16: Defining a function with multiple arguments .....	135
<b>Chapter 23: Defining functions with list arguments</b> .....	136
Section 23.1: Function and Call .....	136
<b>Chapter 24: Functional Programming in Python</b> .....	137
Section 24.1: Lambda Function .....	137
Section 24.2: Map Function .....	137
Section 24.3: Reduce Function .....	137
Section 24.4: Filter Function .....	137
<b>Chapter 25: Partial functions</b> .....	138
Section 25.1: Raise the power .....	138
<b>Chapter 26: Decorators</b> .....	139
Section 26.1: Decorator function .....	139
Section 26.2: Decorator class .....	140
Section 26.3: Decorator with arguments (decorator factory) .....	141
Section 26.4: Making a decorator look like the decorated function .....	142
Section 26.5: Using a decorator to time a function .....	143
Section 26.6: Create singleton class with a decorator .....	144
<b>Chapter 27: Classes</b> .....	145
Section 27.1: Introduction to classes .....	145

<a href="#">Section 27.2: Bound, unbound, and static methods</a>	146
<a href="#">Section 27.3: Basic inheritance</a>	148
<a href="#">Section 27.4: Monkey Patching</a>	150
<a href="#">Section 27.5: New-style vs. old-style classes</a>	150
<a href="#">Section 27.6: Class methods: alternate initializers</a>	151
<a href="#">Section 27.7: Multiple Inheritance</a>	153
<a href="#">Section 27.8: Properties</a>	155
<a href="#">Section 27.9: Default values for instance variables</a>	156
<a href="#">Section 27.10: Class and instance variables</a>	157
<a href="#">Section 27.11: Class composition</a>	158
<a href="#">Section 27.12: Listing All Class Members</a>	159
<a href="#">Section 27.13: Singleton class</a>	160
<a href="#">Section 27.14: Descriptors and Dotted Lookups</a>	161
<b>Chapter 28: Metaclasses</b>	162
<a href="#">Section 28.1: Basic Metaclasses</a>	162
<a href="#">Section 28.2: Singletons using metaclasses</a>	163
<a href="#">Section 28.3: Using a metaclass</a>	163
<a href="#">Section 28.4: Introduction to Metaclasses</a>	163
<a href="#">Section 28.5: Custom functionality with metaclasses</a>	164
<a href="#">Section 28.6: The default metaclass</a>	165
<b>Chapter 29: String Methods</b>	167
<a href="#">Section 29.1: Changing the capitalization of a string</a>	167
<a href="#">Section 29.2: str.translate: Translating characters in a string</a>	168
<a href="#">Section 29.3: str.format and f-strings: Format values into a string</a>	168
<a href="#">Section 29.4: String module's useful constants</a>	169
<a href="#">Section 29.5: Stripping unwanted leading/trailing characters from a string</a>	171
<a href="#">Section 29.6: Reversing a string</a>	171
<a href="#">Section 29.7: Split a string based on a delimiter into a list of strings</a>	172
<a href="#">Section 29.8: Replace all occurrences of one substring with another substring</a>	173
<a href="#">Section 29.9: Testing what a string is composed of</a>	173
<a href="#">Section 29.10: String Contains</a>	176
<a href="#">Section 29.11: Join a list of strings into one string</a>	176
<a href="#">Section 29.12: Counting number of times a substring appears in a string</a>	176
<a href="#">Section 29.13: Case insensitive string comparisons</a>	177
<a href="#">Section 29.14: Justify strings</a>	178
<a href="#">Section 29.15: Test the starting and ending characters of a string</a>	178
<a href="#">Section 29.16: Conversion between str or bytes data and unicode characters</a>	179
<b>Chapter 30: String Formatting</b>	181
<a href="#">Section 30.1: Basics of String Formatting</a>	181
<a href="#">Section 30.2: Alignment and padding</a>	182
<a href="#">Section 30.3: Format literals (f-string)</a>	183
<a href="#">Section 30.4: Float formatting</a>	183
<a href="#">Section 30.5: Named placeholders</a>	184
<a href="#">Section 30.6: String formatting with datetime</a>	185
<a href="#">Section 30.7: Formatting Numerical Values</a>	185
<a href="#">Section 30.8: Nested formatting</a>	186
<a href="#">Section 30.9: Format using Getitem and Getattr</a>	186
<a href="#">Section 30.10: Padding and truncating strings combined</a>	186
<a href="#">Section 30.11: Custom formatting for a class</a>	187
<b>Chapter 31: Conditionals</b>	189



Section 31.1: Conditional Expression (or "The Ternary Operator")	189
Section 31.2: if, elif, and else	189
Section 31.3: Truth Values	189
Section 31.4: Boolean Logic Expressions	190
Section 31.5: Using the cmp function to get the comparison result of two objects	192
Section 31.6: Else statement	192
Section 31.7: Testing if an object is None and assigning it	192
Section 31.8: If statement	193
<b>Chapter 32: Loops</b>	194
Section 32.1: Break and Continue in Loops	194
Section 32.2: For loops	196
Section 32.3: Iterating over lists	196
Section 32.4: Loops with an "else" clause	197
Section 32.5: The Pass Statement	199
Section 32.6: Iterating over dictionaries	200
Section 32.7: The "half loop" do-while	201
Section 32.8: Looping and Unpacking	201
Section 32.9: Iterating different portion of a list with different step size	202
Section 32.10: While Loop	203
<b>Chapter 33: Using loops within functions</b>	204
Section 33.1: Return statement inside loop in a function	204
<b>Chapter 34: Importing modules</b>	205
Section 34.1: Importing a module	205
Section 34.2: The <code>__all__</code> special variable	206
Section 34.3: Import modules from an arbitrary filesystem location	207
Section 34.4: Importing all names from a module	207
Section 34.5: Programmatic importing	208
Section 34.6: PEP8 rules for Imports	208
Section 34.7: Importing specific names from a module	209
Section 34.8: Importing submodules	209
Section 34.9: Re-importing a module	209
Section 34.10: <code>import __()</code> function	210
<b>Chapter 35: Difference between Module and Package</b>	211
Section 35.1: Modules	211
Section 35.2: Packages	211
<b>Chapter 36: Math Module</b>	212
Section 36.1: Rounding: round, floor, ceil, trunc	212
Section 36.2: Trigonometry	213
Section 36.3: Pow for faster exponentiation	214
Section 36.4: Infinity and NaN ("not a number")	214
Section 36.5: Logarithms	217
Section 36.6: Constants	217
Section 36.7: Imaginary Numbers	218
Section 36.8: Copying signs	218
Section 36.9: Complex numbers and the cmath module	218
<b>Chapter 37: Complex math</b>	221
Section 37.1: Advanced complex arithmetic	221
Section 37.2: Basic complex arithmetic	222
<b>Chapter 38: Collections module</b>	223

<a href="#">Section 38.1: collections.Counter</a>	223
<a href="#">Section 38.2: collections.OrderedDict</a>	224
<a href="#">Section 38.3: collections.defaultdict</a>	225
<a href="#">Section 38.4: collections.namedtuple</a>	226
<a href="#">Section 38.5: collections.deque</a>	227
<a href="#">Section 38.6: collections.ChainMap</a>	228
<b>Chapter 39: Operator module</b>	230
<a href="#">Section 39.1: Itemgetter</a>	230
<a href="#">Section 39.2: Operators as alternative to an infix operator</a>	230
<a href="#">Section 39.3: Methodcaller</a>	230
<b>Chapter 40: JSON Module</b>	232
<a href="#">Section 40.1: Storing data in a file</a>	232
<a href="#">Section 40.2: Retrieving data from a file</a>	232
<a href="#">Section 40.3: Formatting JSON output</a>	232
<a href="#">Section 40.4: `load` vs `loads`, `dump` vs `dumps`</a>	233
<a href="#">Section 40.5: Calling `json.tool` from the command line to pretty-print JSON output</a>	234
<a href="#">Section 40.6: JSON encoding custom objects</a>	234
<a href="#">Section 40.7: Creating JSON from Python dict</a>	235
<a href="#">Section 40.8: Creating Python dict from JSON</a>	235
<b>Chapter 41: Sqlite3 Module</b>	236
<a href="#">Section 41.1: Sqlite3 - Not require separate server process</a>	236
<a href="#">Section 41.2: Getting the values from the database and Error handling</a>	236
<b>Chapter 42: The os Module</b>	238
<a href="#">Section 42.1: makedirs - recursive directory creation</a>	238
<a href="#">Section 42.2: Create a directory</a>	239
<a href="#">Section 42.3: Get current directory</a>	239
<a href="#">Section 42.4: Determine the name of the operating system</a>	239
<a href="#">Section 42.5: Remove a directory</a>	239
<a href="#">Section 42.6: Follow a symlink (POSIX)</a>	239
<a href="#">Section 42.7: Change permissions on a file</a>	239
<b>Chapter 43: The locale Module</b>	240
<a href="#">Section 43.1: Currency Formatting US Dollars Using the locale Module</a>	240
<b>Chapter 44: Itertools Module</b>	241
<a href="#">Section 44.1: Combinations method in Itertools Module</a>	241
<a href="#">Section 44.2: itertools.dropwhile</a>	241
<a href="#">Section 44.3: Zipping two iterators until they are both exhausted</a>	242
<a href="#">Section 44.4: Take a slice of a generator</a>	242
<a href="#">Section 44.5: Grouping items from an iterable object using a function</a>	243
<a href="#">Section 44.6: itertools.takewhile</a>	244
<a href="#">Section 44.7: itertools.permutations</a>	244
<a href="#">Section 44.8: itertools.repeat</a>	245
<a href="#">Section 44.9: Get an accumulated sum of numbers in an iterable</a>	245
<a href="#">Section 44.10: Cycle through elements in an iterator</a>	245
<a href="#">Section 44.11: itertools.product</a>	245
<a href="#">Section 44.12: itertools.count</a>	246
<a href="#">Section 44.13: Chaining multiple iterators together</a>	247
<b>Chapter 45: Asyncio Module</b>	248
<a href="#">Section 45.1: Coroutine and Delegation Syntax</a>	248
<a href="#">Section 45.2: Asynchronous Executors</a>	249



<a href="#">Section 45.3: Using UVLoop</a>	250
<a href="#">Section 45.4: Synchronization Primitive: Event</a>	250
<a href="#">Section 45.5: A Simple Websocket</a>	251
<a href="#">Section 45.6: Common Misconception about asyncio</a>	251
<b>Chapter 46: Random module</b>	253
<a href="#">Section 46.1: Creating a random user password</a>	253
<a href="#">Section 46.2: Create cryptographically secure random numbers</a>	253
<a href="#">Section 46.3: Random and sequences: shuffle, choice and sample</a>	254
<a href="#">Section 46.4: Creating random integers and floats: randint, randrange, random, and uniform</a>	255
<a href="#">Section 46.5: Reproducible random numbers: Seed and State</a>	256
<a href="#">Section 46.6: Random Binary Decision</a>	257
<b>Chapter 47: Functools Module</b>	258
<a href="#">Section 47.1: partial</a>	258
<a href="#">Section 47.2: cmp_to_key</a>	258
<a href="#">Section 47.3: lru_cache</a>	258
<a href="#">Section 47.4: total_ordering</a>	259
<a href="#">Section 47.5: reduce</a>	260
<b>Chapter 48: The dis module</b>	261
<a href="#">Section 48.1: What is Python bytecode?</a>	261
<a href="#">Section 48.2: Constants in the dis module</a>	261
<a href="#">Section 48.3: Disassembling modules</a>	261
<b>Chapter 49: The base64 Module</b>	263
<a href="#">Section 49.1: Encoding and Decoding Base64</a>	264
<a href="#">Section 49.2: Encoding and Decoding Base32</a>	265
<a href="#">Section 49.3: Encoding and Decoding Base16</a>	265
<a href="#">Section 49.4: Encoding and Decoding ASCII85</a>	266
<a href="#">Section 49.5: Encoding and Decoding Base85</a>	266
<b>Chapter 50: Queue Module</b>	268
<a href="#">Section 50.1: Simple example</a>	268
<b>Chapter 51: Deque Module</b>	269
<a href="#">Section 51.1: Basic deque using</a>	269
<a href="#">Section 51.2: Available methods in deque</a>	269
<a href="#">Section 51.3: limit deque size</a>	270
<a href="#">Section 51.4: Breadth First Search</a>	270
<b>Chapter 52: Usage of "pip" module: PyPI Package Manager</b>	271
<a href="#">Section 52.1: Example use of commands</a>	271
<a href="#">Section 52.2: Handling ImportError Exception</a>	271
<a href="#">Section 52.3: Force install</a>	272
<b>Chapter 53: Webbrowser Module</b>	273
<a href="#">Section 53.1: Opening a URL with Default Browser</a>	273
<a href="#">Section 53.2: Opening a URL with Different Browsers</a>	274
<b>Chapter 54: pyautogui module</b>	275
<a href="#">Section 54.1: Mouse Functions</a>	275
<a href="#">Section 54.2: Keyboard Functions</a>	275
<a href="#">Section 54.3: ScreenShot And Image Recognition</a>	275
<b>Chapter 55: Plotting with Matplotlib</b>	276
<a href="#">Section 55.1: Plots with Common X-axis but different Y-axis : Using twinx()</a>	276
<a href="#">Section 55.2: Plots with common Y-axis and different X-axis using twiny()</a>	277
<a href="#">Section 55.3: A Simple Plot in Matplotlib</a>	279

<a href="#">Section 55.4: Adding more features to a simple plot : axis labels, title, axis ticks, grid, and legend</a>	280
<a href="#">Section 55.5: Making multiple plots in the same figure by superimposition similar to MATLAB</a>	281
<a href="#">Section 55.6: Making multiple Plots in the same figure using plot superimposition with separate plot commands</a>	282
<b>Chapter 56: Comparisons</b>	284
<a href="#">Section 56.1: Chain Comparisons</a>	284
<a href="#">Section 56.2: Comparison by `is` vs `==`</a>	285
<a href="#">Section 56.3: Greater than or less than</a>	286
<a href="#">Section 56.4: Not equal to</a>	286
<a href="#">Section 56.5: Equal To</a>	287
<a href="#">Section 56.6: Comparing Objects</a>	287
<b>Chapter 57: Sorting, Minimum and Maximum</b>	289
<a href="#">Section 57.1: Make custom classes orderable</a>	289
<a href="#">Section 57.2: Special case: dictionaries</a>	291
<a href="#">Section 57.3: Using the key argument</a>	292
<a href="#">Section 57.4: Default Argument to max, min</a>	292
<a href="#">Section 57.5: Getting a sorted sequence</a>	293
<a href="#">Section 57.6: Extracting N largest or N smallest items from an iterable</a>	293
<a href="#">Section 57.7: Getting the minimum or maximum of several values</a>	294
<a href="#">Section 57.8: Minimum and Maximum of a sequence</a>	294
<b>Chapter 58: Variable Scope and Binding</b>	295
<a href="#">Section 58.1: Nonlocal Variables</a>	295
<a href="#">Section 58.2: Global Variables</a>	295
<a href="#">Section 58.3: Local Variables</a>	296
<a href="#">Section 58.4: The del command</a>	297
<a href="#">Section 58.5: Functions skip class scope when looking up names</a>	298
<a href="#">Section 58.6: Local vs Global Scope</a>	299
<a href="#">Section 58.7: Binding Occurrence</a>	301
<b>Chapter 59: Basic Input and Output</b>	302
<a href="#">Section 59.1: Using the print function</a>	302
<a href="#">Section 59.2: Input from a File</a>	302
<a href="#">Section 59.3: Read from stdin</a>	304
<a href="#">Section 59.4: Using input() and raw_input()</a>	304
<a href="#">Section 59.5: Function to prompt user for a number</a>	304
<a href="#">Section 59.6: Printing a string without a newline at the end</a>	305
<b>Chapter 60: Files &amp; Folders I/O</b>	307
<a href="#">Section 60.1: File modes</a>	307
<a href="#">Section 60.2: Reading a file line-by-line</a>	308
<a href="#">Section 60.3: Iterate files (recursively)</a>	309
<a href="#">Section 60.4: Getting the full contents of a file</a>	309
<a href="#">Section 60.5: Writing to a file</a>	310
<a href="#">Section 60.6: Check whether a file or path exists</a>	311
<a href="#">Section 60.7: Random File Access Using mmap</a>	312
<a href="#">Section 60.8: Replacing text in a file</a>	312
<a href="#">Section 60.9: Checking if a file is empty</a>	312
<a href="#">Section 60.10: Read a file between a range of lines</a>	313
<a href="#">Section 60.11: Copy a directory tree</a>	313
<a href="#">Section 60.12: Copying contents of one file to a different file</a>	313
<b>Chapter 61: Indexing and Slicing</b>	314
<a href="#">Section 61.1: Basic Slicing</a>	314

Section 61.2: Reversing an object .....	315
Section 61.3: Slice assignment .....	315
Section 61.4: Making a shallow copy of an array .....	315
Section 61.5: Indexing custom classes: <code>getitem</code> , <code>setitem</code> and <code>delitem</code> .....	316
Section 61.6: Basic Indexing .....	317
<b>Chapter 62: Generators</b> .....	318
Section 62.1: Introduction .....	318
Section 62.2: Infinite sequences .....	320
Section 62.3: Sending objects to a generator .....	321
Section 62.4: Yielding all values from another iterable .....	322
Section 62.5: Iteration .....	322
Section 62.6: The <code>next()</code> function .....	322
Section 62.7: Coroutines .....	323
Section 62.8: Refactoring list-building code .....	323
Section 62.9: Yield with recursion: recursively listing all files in a directory .....	324
Section 62.10: Generator expressions .....	325
Section 62.11: Using a generator to find Fibonacci Numbers .....	325
Section 62.12: Searching .....	325
Section 62.13: Iterating over generators in parallel .....	326
<b>Chapter 63: Reduce</b> .....	327
Section 63.1: Overview .....	327
Section 63.2: Using <code>reduce</code> .....	327
Section 63.3: Cumulative product .....	328
Section 63.4: Non short-circuit variant of <code>any/all</code> .....	328
<b>Chapter 64: Map Function</b> .....	329
Section 64.1: Basic use of <code>map</code> , <code>itertools.imap</code> and <code>future_builtins.map</code> .....	329
Section 64.2: Mapping each value in an iterable .....	329
Section 64.3: Mapping values of different iterables .....	330
Section 64.4: Transposing with Map: Using "None" as function argument (python 2.x only) .....	332
Section 64.5: Series and Parallel Mapping .....	332
<b>Chapter 65: Exponentiation</b> .....	335
Section 65.1: Exponentiation using builtins: <code>**</code> and <code>pow()</code> .....	335
Section 65.2: Square root: <code>math.sqrt()</code> and <code>cmath.sqrt</code> .....	335
Section 65.3: Modular exponentiation: <code>pow()</code> with 3 arguments .....	336
Section 65.4: Computing large integer roots .....	336
Section 65.5: Exponentiation using the <code>math</code> module: <code>math.pow()</code> .....	337
Section 65.6: Exponential function: <code>math.exp()</code> and <code>cmath.exp()</code> .....	338
Section 65.7: Exponential function minus 1: <code>math.expm1()</code> .....	338
Section 65.8: Magic methods and exponentiation: <code>builtin</code> , <code>math</code> and <code>cmath</code> .....	339
Section 65.9: Roots: nth-root with fractional exponents .....	340
<b>Chapter 66: Searching</b> .....	341
Section 66.1: Searching for an element .....	341
Section 66.2: Searching in custom classes: <code>contains</code> and <code>iter</code> .....	341
Section 66.3: Getting the index for strings: <code>str.index()</code> , <code>str.rindex()</code> and <code>str.find()</code> , <code>str.rfind()</code> .....	342
Section 66.4: Getting the index list and tuples: <code>list.index()</code> , <code>tuple.index()</code> .....	343
Section 66.5: Searching key(s) for a value in dict .....	343
Section 66.6: Getting the index for sorted sequences: <code>bisect.bisect_left()</code> .....	344
Section 66.7: Searching nested sequences .....	344
<b>Chapter 67: Counting</b> .....	346
Section 67.1: Counting all occurrence of all items in an iterable: <code>collections.Counter</code> .....	346

<a href="#">Section 67.2: Getting the most common value(-s): collections.Counter.most_common()</a>	346
<a href="#">Section 67.3: Counting the occurrences of one item in a sequence: list.count() and tuple.count()</a>	346
<a href="#">Section 67.4: Counting the occurrences of a substring in a string: str.count()</a>	347
<a href="#">Section 67.5: Counting occurrences in numpy array</a>	347
<b>Chapter 68: Manipulating XML</b>	348
<a href="#">Section 68.1: Opening and reading using an ElementTree</a>	348
<a href="#">Section 68.2: Create and Build XML Documents</a>	348
<a href="#">Section 68.3: Modifying an XML File</a>	349
<a href="#">Section 68.4: Searching the XML with XPath</a>	349
<a href="#">Section 68.5: Opening and reading large XML files using iterparse (incremental parsing)</a>	350
<b>Chapter 69: Parallel computation</b>	351
<a href="#">Section 69.1: Using the multiprocessing module to parallelise tasks</a>	351
<a href="#">Section 69.2: Using a C-extension to parallelize tasks</a>	351
<a href="#">Section 69.3: Using Parent and Children scripts to execute code in parallel</a>	351
<a href="#">Section 69.4: Using PyPar module to parallelize</a>	352
<b>Chapter 70: Processes and Threads</b>	353
<a href="#">Section 70.1: Global Interpreter Lock</a>	353
<a href="#">Section 70.2: Running in Multiple Threads</a>	354
<a href="#">Section 70.3: Running in Multiple Processes</a>	355
<a href="#">Section 70.4: Sharing State Between Threads</a>	355
<a href="#">Section 70.5: Sharing State Between Processes</a>	356
<b>Chapter 71: Multithreading</b>	357
<a href="#">Section 71.1: Basics of multithreading</a>	357
<a href="#">Section 71.2: Communicating between threads</a>	358
<a href="#">Section 71.3: Creating a worker pool</a>	359
<a href="#">Section 71.4: Advanced use of multithreads</a>	359
<a href="#">Section 71.5: Stoppable Thread with a while Loop</a>	361
<b>Chapter 72: Writing extensions</b>	362
<a href="#">Section 72.1: Hello World with C Extension</a>	362
<a href="#">Section 72.2: C Extension Using c++ and Boost</a>	362
<a href="#">Section 72.3: Passing an open file to C Extensions</a>	364
<b>Chapter 73: Unit Testing</b>	365
<a href="#">Section 73.1: Test Setup and Teardown within a unittest.TestCase</a>	365
<a href="#">Section 73.2: Asserting on Exceptions</a>	365
<a href="#">Section 73.3: Testing Exceptions</a>	366
<a href="#">Section 73.4: Choosing Assertions Within Unittests</a>	367
<a href="#">Section 73.5: Unit tests with pytest</a>	368
<a href="#">Section 73.6: Mocking functions with unittest.mock.create_autospec</a>	371
<b>Chapter 74: Regular Expressions (Regex)</b>	373
<a href="#">Section 74.1: Matching the beginning of a string</a>	373
<a href="#">Section 74.2: Searching</a>	374
<a href="#">Section 74.3: Precompiled patterns</a>	374
<a href="#">Section 74.4: Flags</a>	375
<a href="#">Section 74.5: Replacing</a>	376
<a href="#">Section 74.6: Find All Non-Overlapping Matches</a>	376
<a href="#">Section 74.7: Checking for allowed characters</a>	377
<a href="#">Section 74.8: Splitting a string using regular expressions</a>	377
<a href="#">Section 74.9: Grouping</a>	377
<a href="#">Section 74.10: Escaping Special Characters</a>	378

Section 74.11: Match an expression only in specific locations .....	379
Section 74.12: Iterating over matches using <code>`re.finditer`</code> .....	380
<b>Chapter 75: Incompatibilities moving from Python 2 to Python 3</b> .....	381
Section 75.1: Integer Division .....	381
Section 75.2: Unpacking Iterables .....	382
Section 75.3: Strings: Bytes versus Unicode .....	384
Section 75.4: Print statement vs. Print function .....	386
Section 75.5: Differences between range and xrange functions .....	387
Section 75.6: Raising and handling Exceptions .....	388
Section 75.7: Leaked variables in list comprehension .....	390
Section 75.8: True, False and None .....	391
Section 75.9: User Input .....	391
Section 75.10: Comparison of different types .....	391
Section 75.11: <code>.next()</code> method on iterators renamed .....	392
Section 75.12: <code>filter()</code> , <code>map()</code> and <code>zip()</code> return iterators instead of sequences .....	393
Section 75.13: Renamed modules .....	393
Section 75.14: Removed operators <code>&lt;&gt;</code> and <code>`</code> , synonymous with <code>!=</code> and <code>repr()</code> .....	394
Section 75.15: long vs. int .....	394
Section 75.16: All classes are "new-style classes" in Python 3 .....	395
Section 75.17: Reduce is no longer a built-in .....	396
Section 75.18: Absolute/Relative Imports .....	396
Section 75.19: <code>map()</code> .....	398
Section 75.20: The <code>round()</code> function tie-breaking and return type .....	399
Section 75.21: File I/O .....	400
Section 75.22: <code>cmp</code> function removed in Python 3 .....	400
Section 75.23: Octal Constants .....	401
Section 75.24: Return value when writing to a file object .....	401
Section 75.25: <code>exec</code> statement is a function in Python 3 .....	401
Section 75.26: encode/decode to hex no longer available .....	402
Section 75.27: Dictionary method changes .....	402
Section 75.28: Class Boolean Value .....	403
Section 75.29: <code>hasattr</code> function bug in Python 2 .....	404
<b>Chapter 76: Virtual environments</b> .....	405
Section 76.1: Creating and using a virtual environment .....	405
Section 76.2: Specifying specific python version to use in script on Unix/Linux .....	407
Section 76.3: Creating a virtual environment for a different version of python .....	407
Section 76.4: Making virtual environments using Anaconda .....	407
Section 76.5: Managing multiple virtual enviroments with <code>virtualenvwrapper</code> .....	408
Section 76.6: Installing packages in a virtual environment .....	409
Section 76.7: Discovering which virtual environment you are using .....	410
Section 76.8: Checking if running inside a virtual environment .....	411
Section 76.9: Using <code>virtualenv</code> with fish shell .....	411
<b>Chapter 77: Copying data</b> .....	413
Section 77.1: Copy a dictionary .....	413
Section 77.2: Performing a shallow copy .....	413
Section 77.3: Performing a deep copy .....	413
Section 77.4: Performing a shallow copy of a list .....	413
Section 77.5: Copy a set .....	413
<b>Chapter 78: Context Managers ("with" Statement)</b> .....	415
Section 78.1: Introduction to context managers and the with statement .....	415

Section 78.2: Writing your own context manager .....	415
Section 78.3: Writing your own contextmanager using generator syntax .....	416
Section 78.4: Multiple context managers .....	417
Section 78.5: Assigning to a target .....	417
Section 78.6: Manage Resources .....	418
<b>Chapter 79: Hidden Features</b> .....	419
Section 79.1: Operator Overloading .....	419
<b>Chapter 80: Unicode and bytes</b> .....	420
Section 80.1: Encoding/decoding error handling .....	420
Section 80.2: File I/O .....	420
Section 80.3: Basics .....	421
<b>Chapter 81: The <code>_name_</code> special variable</b> .....	423
Section 81.1: <code>_name_ == 'main'</code> .....	423
Section 81.2: Use in logging .....	423
Section 81.3: <code>function class or module._name_</code> .....	423
<b>Chapter 82: Checking Path Existence and Permissions</b> .....	425
Section 82.1: Perform checks using <code>os.access</code> .....	425
<b>Chapter 83: Python Networking</b> .....	426
Section 83.1: Creating a Simple Http Server .....	426
Section 83.2: Creating a TCP server .....	426
Section 83.3: Creating a UDP Server .....	427
Section 83.4: Start Simple HttpServer in a thread and open the browser .....	427
Section 83.5: The simplest Python socket client-server example .....	428
<b>Chapter 84: The Print Function</b> .....	429
Section 84.1: Print basics .....	429
Section 84.2: Print parameters .....	430
<b>Chapter 85: <code>os.path</code></b> .....	432
Section 85.1: Join Paths .....	432
Section 85.2: Path Component Manipulation .....	432
Section 85.3: Get the parent directory .....	432
Section 85.4: If the given path exists .....	432
Section 85.5: check if the given path is a directory, file, symbolic link, mount point etc .....	433
Section 85.6: Absolute Path from Relative Path .....	433
<b>Chapter 86: Creating Python packages</b> .....	434
Section 86.1: Introduction .....	434
Section 86.2: Uploading to PyPI .....	434
Section 86.3: Making package executable .....	436
<b>Chapter 87: Parsing Command Line arguments</b> .....	438
Section 87.1: Hello world in <code>argparse</code> .....	438
Section 87.2: Using command line arguments with <code>argv</code> .....	438
Section 87.3: Setting mutually exclusive arguments with <code>argparse</code> .....	439
Section 87.4: Basic example with <code>docopt</code> .....	440
Section 87.5: Custom parser error message with <code>argparse</code> .....	440
Section 87.6: Conceptual grouping of arguments with <code>argparse.add_argument_group()</code> .....	441
Section 87.7: Advanced example with <code>docopt</code> and <code>docopt_dispatch</code> .....	442
<b>Chapter 88: HTML Parsing</b> .....	444
Section 88.1: Using CSS selectors in BeautifulSoup .....	444
Section 88.2: PyQuery .....	444
Section 88.3: Locate a text after an element in BeautifulSoup .....	445



<b>Chapter 89: Subprocess Library</b>	446
<a href="#">Section 89.1: More flexibility with Popen</a>	446
<a href="#">Section 89.2: Calling External Commands</a>	447
<a href="#">Section 89.3: How to create the command list argument</a>	447
<b>Chapter 90: setup.py</b>	448
<a href="#">Section 90.1: Purpose of setup.py</a>	448
<a href="#">Section 90.2: Using source control metadata in setup.py</a>	448
<a href="#">Section 90.3: Adding command line scripts to your python package</a>	449
<a href="#">Section 90.4: Adding installation options</a>	449
<b>Chapter 91: Sockets</b>	451
<a href="#">Section 91.1: Raw Sockets on Linux</a>	451
<a href="#">Section 91.2: Sending data via UDP</a>	451
<a href="#">Section 91.3: Receiving data via UDP</a>	452
<a href="#">Section 91.4: Sending data via TCP</a>	452
<a href="#">Section 91.5: Multi-threaded TCP Socket Server</a>	452
<b>Chapter 92: Recursion</b>	455
<a href="#">Section 92.1: The What, How, and When of Recursion</a>	455
<a href="#">Section 92.2: Tree exploration with recursion</a>	458
<a href="#">Section 92.3: Sum of numbers from 1 to n</a>	459
<a href="#">Section 92.4: Increasing the Maximum Recursion Depth</a>	459
<a href="#">Section 92.5: Tail Recursion - Bad Practice</a>	460
<a href="#">Section 92.6: Tail Recursion Optimization Through Stack Introspection</a>	460
<b>Chapter 93: Type Hints</b>	462
<a href="#">Section 93.1: Adding types to a function</a>	462
<a href="#">Section 93.2: NamedTuple</a>	463
<a href="#">Section 93.3: Generic Types</a>	463
<a href="#">Section 93.4: Variables and Attributes</a>	463
<a href="#">Section 93.5: Class Members and Methods</a>	464
<a href="#">Section 93.6: Type hints for keyword arguments</a>	464
<b>Chapter 94: pip: PyPI Package Manager</b>	465
<a href="#">Section 94.1: Install Packages</a>	465
<a href="#">Section 94.2: To list all packages installed using `pip`</a>	465
<a href="#">Section 94.3: Upgrade Packages</a>	465
<a href="#">Section 94.4: Uninstall Packages</a>	466
<a href="#">Section 94.5: Updating all outdated packages on Linux</a>	466
<a href="#">Section 94.6: Updating all outdated packages on Windows</a>	466
<a href="#">Section 94.7: Create a requirements.txt file of all packages on the system</a>	466
<a href="#">Section 94.8: Using a certain Python version with pip</a>	467
<a href="#">Section 94.9: Create a requirements.txt file of packages only in the current virtualenv</a>	467
<a href="#">Section 94.10: Installing packages not yet on pip as wheels</a>	468
<b>Chapter 95: Exceptions</b>	471
<a href="#">Section 95.1: Catching Exceptions</a>	471
<a href="#">Section 95.2: Do not catch everything!</a>	471
<a href="#">Section 95.3: Re-raising exceptions</a>	472
<a href="#">Section 95.4: Catching multiple exceptions</a>	472
<a href="#">Section 95.5: Exception Hierarchy</a>	473
<a href="#">Section 95.6: Else</a>	475
<a href="#">Section 95.7: Raising Exceptions</a>	475
<a href="#">Section 95.8: Creating custom exception types</a>	476

Section 95.9: Practical examples of exception handling .....	476
Section 95.10: Exceptions are Objects too .....	477
Section 95.11: Running clean-up code with finally .....	477
Section 95.12: Chain exceptions with raise from .....	478
<b>Chapter 96: Web scraping with Python</b> .....	479
Section 96.1: Scraping using the Scrapy framework .....	479
Section 96.2: Scraping using Selenium WebDriver .....	479
Section 96.3: Basic example of using requests and lxml to scrape some data .....	480
Section 96.4: Maintaining web-scraping session with requests .....	480
Section 96.5: Scraping using BeautifulSoup4 .....	481
Section 96.6: Simple web content download with urllib.request .....	481
Section 96.7: Modify Scrapy user agent .....	481
Section 96.8: Scraping with curl .....	481
<b>Chapter 97: Distribution</b> .....	483
Section 97.1: py2app .....	483
Section 97.2: cx_Freeze .....	484
<b>Chapter 98: Property Objects</b> .....	485
Section 98.1: Using the @property decorator for read-write properties .....	485
Section 98.2: Using the @property decorator .....	485
Section 98.3: Overriding just a getter, setter or a deleter of a property object .....	485
Section 98.4: Using properties without decorators .....	486
<b>Chapter 99: Overloading</b> .....	488
Section 99.1: Operator overloading .....	488
Section 99.2: Magic/Dunder Methods .....	489
Section 99.3: Container and sequence types .....	490
Section 99.4: Callable types .....	491
Section 99.5: Handling unimplemented behaviour .....	491
<b>Chapter 100: Debugging</b> .....	493
Section 100.1: Via IPython and ipdb .....	493
Section 100.2: The Python Debugger: Step-through Debugging with pdb .....	493
Section 100.3: Remote debugger .....	495
<b>Chapter 101: Reading and Writing CSV</b> .....	496
Section 101.1: Using pandas .....	496
Section 101.2: Writing a TSV file .....	496
<b>Chapter 102: Dynamic code execution with `exec` and `eval`</b> .....	497
Section 102.1: Executing code provided by untrusted user using exec, eval, or ast.literal_eval .....	497
Section 102.2: Evaluating a string containing a Python literal with ast.literal_eval .....	497
Section 102.3: Evaluating statements with exec .....	497
Section 102.4: Evaluating an expression with eval .....	498
Section 102.5: Precompiling an expression to evaluate it multiple times .....	498
Section 102.6: Evaluating an expression with eval using custom globals .....	498
<b>Chapter 103: PyInstaller - Distributing Python Code</b> .....	499
Section 103.1: Installation and Setup .....	499
Section 103.2: Using Pyinstaller .....	499
Section 103.3: Bundling to One Folder .....	500
Section 103.4: Bundling to a Single File .....	500
<b>Chapter 104: Iterables and Iterators</b> .....	501
Section 104.1: Iterator vs Iterable vs Generator .....	501
Section 104.2: Extract values one by one .....	502

Section 104.3: Iterating over entire iterable .....	502
Section 104.4: Verify only one element in iterable .....	502
Section 104.5: What can be iterable .....	503
Section 104.6: Iterator isn't reentrant! .....	503
<b>Chapter 105: Data Visualization with Python</b> .....	504
Section 105.1: Seaborn .....	504
Section 105.2: Matplotlib .....	506
Section 105.3: Plotly .....	507
Section 105.4: MayaVI .....	509
<b>Chapter 106: The Interpreter (Command Line Console)</b> .....	511
Section 106.1: Getting general help .....	511
Section 106.2: Referring to the last expression .....	511
Section 106.3: Opening the Python console .....	512
Section 106.4: The PYTHONSTARTUP variable .....	512
Section 106.5: Command line arguments .....	512
Section 106.6: Getting help about an object .....	513
<b>Chapter 107: *args and **kwargs</b> .....	515
Section 107.1: Using **kwargs when writing functions .....	515
Section 107.2: Using *args when writing functions .....	515
Section 107.3: Populating kwarg values with a dictionary .....	516
Section 107.4: Keyword-only and Keyword-required arguments .....	516
Section 107.5: Using **kwargs when calling functions .....	516
Section 107.6: **kwargs and default values .....	516
Section 107.7: Using *args when calling functions .....	517
<b>Chapter 108: Garbage Collection</b> .....	518
Section 108.1: Reuse of primitive objects .....	518
Section 108.2: Effects of the del command .....	518
Section 108.3: Reference Counting .....	519
Section 108.4: Garbage Collector for Reference Cycles .....	519
Section 108.5: Forcefully deallocating objects .....	520
Section 108.6: Viewing the refcount of an object .....	521
Section 108.7: Do not wait for the garbage collection to clean up .....	521
Section 108.8: Managing garbage collection .....	521
<b>Chapter 109: Pickle data serialisation</b> .....	523
Section 109.1: Using Pickle to serialize and deserialize an object .....	523
Section 109.2: Customize Pickled Data .....	523
<b>Chapter 110: urllib</b> .....	525
Section 110.1: HTTP GET .....	525
Section 110.2: HTTP POST .....	525
Section 110.3: Decode received bytes according to content type encoding .....	526
<b>Chapter 111: Binary Data</b> .....	527
Section 111.1: Format a list of values into a byte object .....	527
Section 111.2: Unpack a byte object according to a format string .....	527
Section 111.3: Packing a structure .....	527
<b>Chapter 112: Python and Excel</b> .....	529
Section 112.1: Read the excel data using xlrd module .....	529
Section 112.2: Format Excel files with xlswriter .....	529
Section 112.3: Put list data into a Excel's file .....	530
Section 112.4: OpenPyXL .....	531

<a href="#">Section 112.5: Create excel charts with <code>xlsxwriter</code></a>	531
<b><a href="#">Chapter 113: Idioms</a></b>	534
<a href="#">Section 113.1: Dictionary key initializations</a>	534
<a href="#">Section 113.2: Switching variables</a>	534
<a href="#">Section 113.3: Use truth value testing</a>	534
<a href="#">Section 113.4: Test for "<code>__main__</code>" to avoid unexpected code execution</a>	535
<b><a href="#">Chapter 114: Method Overriding</a></b>	536
<a href="#">Section 114.1: Basic method overriding</a>	536
<b><a href="#">Chapter 115: Data Serialization</a></b>	537
<a href="#">Section 115.1: Serialization using JSON</a>	537
<a href="#">Section 115.2: Serialization using Pickle</a>	537
<b><a href="#">Chapter 116: Python concurrency</a></b>	539
<a href="#">Section 116.1: The multiprocessing module</a>	539
<a href="#">Section 116.2: The threading module</a>	540
<a href="#">Section 116.3: Passing data between multiprocessing processes</a>	540
<b><a href="#">Chapter 117: Introduction to RabbitMQ using AMQPStorm</a></b>	542
<a href="#">Section 117.1: How to consume messages from RabbitMQ</a>	542
<a href="#">Section 117.2: How to publish messages to RabbitMQ</a>	543
<a href="#">Section 117.3: How to create a delayed queue in RabbitMQ</a>	543
<b><a href="#">Chapter 118: Descriptor</a></b>	546
<a href="#">Section 118.1: Simple descriptor</a>	546
<a href="#">Section 118.2: Two-way conversions</a>	547
<b><a href="#">Chapter 119: Multiprocessing</a></b>	548
<a href="#">Section 119.1: Running Two Simple Processes</a>	548
<a href="#">Section 119.2: Using Pool and Map</a>	548
<b><a href="#">Chapter 120: <code>tempfile</code> <code>NamedTemporaryFile</code></a></b>	550
<a href="#">Section 120.1: Create (and write to a) known, persistant temporary file</a>	550
<b><a href="#">Chapter 121: Input, Subset and Output External Data Files using Pandas</a></b>	551
<a href="#">Section 121.1: Basic Code to Import, Subset and Write External Data Files Using Pandas</a>	551
<b><a href="#">Chapter 122: Writing to CSV from String or List</a></b>	553
<a href="#">Section 122.1: Basic Write Example</a>	553
<a href="#">Section 122.2: Appending a String as a newline in a CSV file</a>	553
<b><a href="#">Chapter 123: Unzipping Files</a></b>	554
<a href="#">Section 123.1: Using Python <code>ZipFile.extractall()</code> to decompress a ZIP file</a>	554
<a href="#">Section 123.2: Using Python <code>TarFile.extractall()</code> to decompress a tarball</a>	554
<b><a href="#">Chapter 124: Working with ZIP archives</a></b>	555
<a href="#">Section 124.1: Examining Zipfile Contents</a>	555
<a href="#">Section 124.2: Opening Zip Files</a>	555
<a href="#">Section 124.3: Extracting zip file contents to a directory</a>	556
<a href="#">Section 124.4: Creating new archives</a>	556
<b><a href="#">Chapter 125: Stack</a></b>	557
<a href="#">Section 125.1: Creating a Stack class with a List Object</a>	557
<a href="#">Section 125.2: Parsing Parentheses</a>	558
<b><a href="#">Chapter 126: Profiling</a></b>	559
<a href="#">Section 126.1: <code>%%timeit</code> and <code>%timeit</code> in IPython</a>	559
<a href="#">Section 126.2: Using cProfile (Preferred Profiler)</a>	559
<a href="#">Section 126.3: <code>timeit()</code> function</a>	559
<a href="#">Section 126.4: <code>timeit</code> command line</a>	560

Section 126.5: <a href="#">line_profiler in command line</a>	560
<b>Chapter 127: User-Defined Methods</b>	561
Section 127.1: <a href="#">Creating user-defined method objects</a>	561
Section 127.2: <a href="#">Turtle example</a>	562
<b>Chapter 128: Working around the Global Interpreter Lock (GIL)</b>	563
Section 128.1: <a href="#">Multiprocessing.Pool</a>	563
Section 128.2: <a href="#">Cython nogil:</a>	564
<b>Chapter 129: Deployment</b>	565
Section 129.1: <a href="#">Uploading a Conda Package</a>	565
<b>Chapter 130: Logging</b>	567
Section 130.1: <a href="#">Introduction to Python Logging</a>	567
Section 130.2: <a href="#">Logging exceptions</a>	568
<b>Chapter 131: Database Access</b>	571
Section 131.1: <a href="#">SQLite</a>	571
Section 131.2: <a href="#">Accessing MySQL database using MySQLdb</a>	576
Section 131.3: <a href="#">Connection</a>	577
Section 131.4: <a href="#">PostgreSQL Database access using psycopg2</a>	578
Section 131.5: <a href="#">Oracle database</a>	579
Section 131.6: <a href="#">Using sqlalchemy</a>	580
<b>Chapter 132: Python HTTP Server</b>	582
Section 132.1: <a href="#">Running a simple HTTP server</a>	582
Section 132.2: <a href="#">Serving files</a>	582
Section 132.3: <a href="#">Basic handling of GET, POST, PUT using BaseHTTPRequestHandler</a>	583
Section 132.4: <a href="#">Programmatic API of SimpleHTTPServer</a>	584
<b>Chapter 133: Web Server Gateway Interface (WSGI)</b>	586
Section 133.1: <a href="#">Server Object (Method)</a>	586
<b>Chapter 134: Python Server Sent Events</b>	587
Section 134.1: <a href="#">Flask SSE</a>	587
Section 134.2: <a href="#">Asyncio SSE</a>	587
<b>Chapter 135: Connecting Python to SQL Server</b>	588
Section 135.1: <a href="#">Connect to Server, Create Table, Query Data</a>	588
<b>Chapter 136: Sockets And Message Encryption/Decryption Between Client and Server</b>	589
Section 136.1: <a href="#">Server side Implementation</a>	589
Section 136.2: <a href="#">Client side Implementation</a>	591
<b>Chapter 137: Alternatives to switch statement from other languages</b>	593
Section 137.1: <a href="#">Use what the language offers: the if/else construct</a>	593
Section 137.2: <a href="#">Use a dict of functions</a>	593
Section 137.3: <a href="#">Use class introspection</a>	594
Section 137.4: <a href="#">Using a context manager</a>	595
<b>Chapter 138: List Comprehensions</b>	596
Section 138.1: <a href="#">Conditional List Comprehensions</a>	596
Section 138.2: <a href="#">List Comprehensions with Nested Loops</a>	597
Section 138.3: <a href="#">Refactoring filter and map to list comprehensions</a>	598
Section 138.4: <a href="#">Nested List Comprehensions</a>	599
Section 138.5: <a href="#">Iterate two or more list simultaneously within list comprehension</a>	600
<b>Chapter 139: List destructuring (aka packing and unpacking)</b>	601
Section 139.1: <a href="#">Destructuring assignment</a>	601

Section 139.2: Packing function arguments .....	602
Section 139.3: Unpacking function arguments .....	604
<b>Chapter 140: Accessing Python source code and bytecode</b> .....	605
Section 140.1: Display the bytecode of a function .....	605
Section 140.2: Display the source code of an object .....	605
Section 140.3: Exploring the code object of a function .....	606
<b>Chapter 141: Mixins</b> .....	607
Section 141.1: Mixin .....	607
Section 141.2: Overriding Methods in Mixins .....	608
<b>Chapter 142: Attribute Access</b> .....	609
Section 142.1: Basic Attribute Access using the Dot Notation .....	609
Section 142.2: Setters, Getters & Properties .....	609
<b>Chapter 143: ArcPy</b> .....	611
Section 143.1: Printing one field's value for all rows of feature class in file geodatabase using Search Cursor .....	611
Section 143.2: createDissolvedGDB to create a file gdb on the workspace .....	611
<b>Chapter 144: Abstract Base Classes (abc)</b> .....	612
Section 144.1: Setting the ABCMeta metaclass .....	612
Section 144.2: Why/How to use ABCMeta and @abstractmethod .....	612
<b>Chapter 145: Plugin and Extension Classes</b> .....	614
Section 145.1: Mixins .....	614
Section 145.2: Plugins with Customized Classes .....	615
<b>Chapter 146: Websockets</b> .....	617
Section 146.1: Simple Echo with aiohttp .....	617
Section 146.2: Wrapper Class with aiohttp .....	617
Section 146.3: Using Autobahn as a Websocket Factory .....	618
<b>Chapter 147: Immutable datatypes(int, float, str, tuple and frozensets)</b> .....	620
Section 147.1: Individual characters of strings are not assignable .....	620
Section 147.2: Tuple's individual members aren't assignable .....	620
Section 147.3: Frozenset's are immutable and not assignable .....	620
<b>Chapter 148: String representations of class instances: __str__ and __repr__ methods</b> .....	621
Section 148.1: Motivation .....	621
Section 148.2: Both methods implemented, eval-round-trip style __repr__() .....	625
<b>Chapter 149: Polymorphism</b> .....	626
Section 149.1: Duck Typing .....	626
Section 149.2: Basic Polymorphism .....	626
<b>Chapter 150: Non-official Python implementations</b> .....	629
Section 150.1: IronPython .....	629
Section 150.2: Jython .....	629
Section 150.3: Transcrypt .....	630
<b>Chapter 151: 2to3 tool</b> .....	633
Section 151.1: Basic Usage .....	633
<b>Chapter 152: Abstract syntax tree</b> .....	635
Section 152.1: Analyze functions in a python script .....	635
<b>Chapter 153: Unicode</b> .....	637
Section 153.1: Encoding and decoding .....	637
<b>Chapter 154: Python Serial Communication (pyserial)</b> .....	638



Section 154.1: Initialize serial device .....	638
Section 154.2: Read from serial port .....	638
Section 154.3: Check what serial ports are available on your machine .....	638
<b>Chapter 155: Neo4j and Cypher using Py2Neo</b> .....	640
Section 155.1: Adding Nodes to Neo4j Graph .....	640
Section 155.2: Importing and Authenticating .....	640
Section 155.3: Adding Relationships to Neo4j Graph .....	640
Section 155.4: Query 1 : Autocomplete on News Titles .....	640
Section 155.5: Query 2 : Get News Articles by Location on a particular date .....	641
Section 155.6: Cypher Query Samples .....	641
<b>Chapter 156: Basic Curses with Python</b> .....	642
Section 156.1: The wrapper() helper function .....	642
Section 156.2: Basic Invocation Example .....	642
<b>Chapter 157: Performance optimization</b> .....	643
Section 157.1: Code profiling .....	643
<b>Chapter 158: Templates in python</b> .....	645
Section 158.1: Simple data output program using template .....	645
Section 158.2: Changing delimiter .....	645
<b>Chapter 159: Pillow</b> .....	646
Section 159.1: Read Image File .....	646
Section 159.2: Convert files to JPEG .....	646
<b>Chapter 160: The pass statement</b> .....	647
Section 160.1: Ignore an exception .....	647
Section 160.2: Create a new Exception that can be caught .....	647
<b>Chapter 161: py.test</b> .....	648
Section 161.1: Setting up py.test .....	648
Section 161.2: Intro to Test Fixtures .....	648
Section 161.3: Failing Tests .....	651
<b>Chapter 162: Heapq</b> .....	653
Section 162.1: Largest and smallest items in a collection .....	653
Section 162.2: Smallest item in a collection .....	653
<b>Chapter 163: tkinter</b> .....	655
Section 163.1: Geometry Managers .....	655
Section 163.2: A minimal tkinter Application .....	656
<b>Chapter 164: CLI subcommands with precise help output</b> .....	658
Section 164.1: Native way (no libraries) .....	658
Section 164.2: argparse (default help formatter) .....	658
Section 164.3: argparse (custom help formatter) .....	659
<b>Chapter 165: PostgreSQL</b> .....	661
Section 165.1: Getting Started .....	661
<b>Chapter 166: Python Persistence</b> .....	662
Section 166.1: Python Persistence .....	662
Section 166.2: Function utility for save and load .....	663
<b>Chapter 167: Turtle Graphics</b> .....	664
Section 167.1: Ninja Twist (Turtle Graphics) .....	664
<b>Chapter 168: Design Patterns</b> .....	665
Section 168.1: Introduction to design patterns and Singleton Pattern .....	665
Section 168.2: Strategy Pattern .....	667

Section 168.3: Proxy .....	668
<b>Chapter 169: Multidimensional arrays</b> .....	670
Section 169.1: Lists in lists .....	670
Section 169.2: Lists in lists in lists in.. .....	670
<b>Chapter 170: Audio</b> .....	672
Section 170.1: Working with WAV files .....	672
Section 170.2: Convert any soundfile with python and ffmpeg .....	672
Section 170.3: Playing Windows' beeps .....	672
Section 170.4: Audio With Pyglet .....	673
<b>Chapter 171: Pyglet</b> .....	674
Section 171.1: Installation of Pyglet .....	674
Section 171.2: Hello World in Pyglet .....	674
Section 171.3: Playing Sound in Pyglet .....	674
Section 171.4: Using Pyglet for OpenGL .....	674
Section 171.5: Drawing Points Using Pyglet and OpenGL .....	674
<b>Chapter 172: Flask</b> .....	676
Section 172.1: Files and Templates .....	676
Section 172.2: The basics .....	676
Section 172.3: Routing URLs .....	677
Section 172.4: HTTP Methods .....	678
Section 172.5: Jinja Templating .....	678
Section 172.6: The Request Object .....	679
<b>Chapter 173: groupby()</b> .....	681
Section 173.1: Example 4 .....	681
Section 173.2: Example 2 .....	681
Section 173.3: Example 3 .....	682
<b>Chapter 174: pygame</b> .....	684
Section 174.1: Pygame's mixer module .....	684
Section 174.2: Installing pygame .....	685
<b>Chapter 175: hashlib</b> .....	686
Section 175.1: MD5 hash of a string .....	686
Section 175.2: algorithm provided by OpenSSL .....	687
<b>Chapter 176: getting start with GZip</b> .....	688
Section 176.1: Read and write GNU zip files .....	688
<b>Chapter 177: ctypes</b> .....	689
Section 177.1: ctypes arrays .....	689
Section 177.2: Wrapping functions for ctypes .....	689
Section 177.3: Basic usage .....	690
Section 177.4: Common pitfalls .....	690
Section 177.5: Basic ctypes object .....	691
Section 177.6: Complex usage .....	692
<b>Chapter 178: Creating a Windows service using Python</b> .....	693
Section 178.1: A Python script that can be run as a service .....	693
Section 178.2: Running a Flask web application as a service .....	694
<b>Chapter 179: Mutable vs Immutable (and Hashable) in Python</b> .....	695
Section 179.1: Mutable vs Immutable .....	695
Section 179.2: Mutable and Immutable as Arguments .....	697
<b>Chapter 180: Python speed of program</b> .....	699

Section 180.1: Deque operations .....	699
Section 180.2: Algorithmic Notations.. .....	699
Section 180.3: Notation .....	700
Section 180.4: List operations .....	701
Section 180.5: Set operations .....	701
<b>Chapter 181: configparser</b> .....	703
Section 181.1: Creating configuration file programmatically .....	703
Section 181.2: Basic usage .....	703
<b>Chapter 182: Commonwealth Exceptions</b> .....	704
Section 182.1: Other Errors .....	704
Section 182.2: NameError: name '???' is not defined .....	705
Section 182.3: TypeErrors .....	706
Section 182.4: Syntax Error on good code .....	707
Section 182.5: IndentationErrors (or indentation SyntaxErrors) .....	708
<b>Chapter 183: Optical Character Recognition</b> .....	710
Section 183.1: PyTesseract .....	710
Section 183.2: PyOCR .....	710
<b>Chapter 184: graph-tool</b> .....	712
Section 184.1: PyDotPlus .....	712
Section 184.2: PyGraphviz .....	712
<b>Chapter 185: Python Virtual Environment - virtualenv</b> .....	714
Section 185.1: Installation .....	714
Section 185.2: Usage .....	714
Section 185.3: Install a package in your Virtualenv .....	714
Section 185.4: Other useful virtualenv commands .....	715
<b>Chapter 186: sys</b> .....	716
Section 186.1: Command line arguments .....	716
Section 186.2: Script name .....	716
Section 186.3: Standard error stream .....	716
Section 186.4: Ending the process prematurely and returning an exit code .....	716
<b>Chapter 187: virtual environment with virtualenvwrapper</b> .....	717
Section 187.1: Create virtual environment with virtualenvwrapper .....	717
<b>Chapter 188: Create virtual environment with virtualenvwrapper in windows</b> .....	719
Section 188.1: Virtual environment with virtualenvwrapper for windows .....	719
<b>Chapter 189: Python Requests Post</b> .....	720
Section 189.1: Simple Post .....	720
Section 189.2: Form Encoded Data .....	721
Section 189.3: File Upload .....	721
Section 189.4: Responses .....	722
Section 189.5: Authentication .....	722
Section 189.6: Proxies .....	723
<b>Chapter 190: Python Lex-Yacc</b> .....	725
Section 190.1: Getting Started with PLY .....	725
Section 190.2: The "Hello, World!" of PLY - A Simple Calculator .....	725
Section 190.3: Part 1: Tokenizing Input with Lex .....	727
Section 190.4: Part 2: Parsing Tokenized Input with Yacc .....	730
<b>Chapter 191: ChemPy - python package</b> .....	734
Section 191.1: Parsing formulae .....	734
Section 191.2: Balancing stoichiometry of a chemical reaction .....	734

<a href="#">Section 191.3: Balancing reactions</a>	734
<a href="#">Section 191.4: Chemical equilibria</a>	735
<a href="#">Section 191.5: Ionic strength</a>	735
<a href="#">Section 191.6: Chemical kinetics (system of ordinary differential equations)</a>	735
<b>Chapter 192: pyaudio</b>	737
<a href="#">Section 192.1: Callback Mode Audio I/O</a>	737
<a href="#">Section 192.2: Blocking Mode Audio I/O</a>	738
<b>Chapter 193: shelve</b>	740
<a href="#">Section 193.1: Creating a new Shelf</a>	740
<a href="#">Section 193.2: Sample code for shelve</a>	741
<a href="#">Section 193.3: To summarize the interface (key is a string, data is an arbitrary object):</a>	741
<a href="#">Section 193.4: Write-back</a>	741
<b>Chapter 194: IoT Programming with Python and Raspberry PI</b>	743
<a href="#">Section 194.1: Example - Temperature sensor</a>	743
<b>Chapter 195: kivy - Cross-platform Python Framework for NUI Development</b>	746
<a href="#">Section 195.1: First App</a>	746
<b>Chapter 196: Call Python from C#</b>	748
<a href="#">Section 196.1: Python script to be called by C# application</a>	748
<a href="#">Section 196.2: C# code calling Python script</a>	748
<b>Chapter 197: Similarities in syntax, Differences in meaning: Python vs. JavaScript</b>	750
<a href="#">Section 197.1: `in` with lists</a>	750
<b>Chapter 198: Raise Custom Errors / Exceptions</b>	751
<a href="#">Section 198.1: Custom Exception</a>	751
<a href="#">Section 198.2: Catch custom Exception</a>	751
<b>Chapter 199: Pandas Transform: Preform operations on groups and concatenate the results</b>	752
<a href="#">Section 199.1: Simple transform</a>	752
<a href="#">Section 199.2: Multiple results per group</a>	753
<b>Chapter 200: Security and Cryptography</b>	754
<a href="#">Section 200.1: Secure Password Hashing</a>	754
<a href="#">Section 200.2: Calculating a Message Digest</a>	754
<a href="#">Section 200.3: Available Hashing Algorithms</a>	754
<a href="#">Section 200.4: File Hashing</a>	755
<a href="#">Section 200.5: Generating RSA signatures using pycrypto</a>	755
<a href="#">Section 200.6: Asymmetric RSA encryption using pycrypto</a>	756
<a href="#">Section 200.7: Symmetric encryption using pycrypto</a>	757
<b>Chapter 201: Secure Shell Connection in Python</b>	758
<a href="#">Section 201.1: ssh connection</a>	758
<b>Chapter 202: Python Anti-Patterns</b>	759
<a href="#">Section 202.1: Overzealous except clause</a>	759
<a href="#">Section 202.2: Looking before you leap with processor-intensive function</a>	759
<b>Chapter 203: Common Pitfalls</b>	761
<a href="#">Section 203.1: List multiplication and common references</a>	761
<a href="#">Section 203.2: Mutable default argument</a>	764
<a href="#">Section 203.3: Changing the sequence you are iterating over</a>	765
<a href="#">Section 203.4: Integer and String identity</a>	768
<a href="#">Section 203.5: Dictionaries are unordered</a>	769
<a href="#">Section 203.6: Variable leaking in list comprehensions and for loops</a>	770

<a href="#">Section 203.7: Chaining of or operator</a>	770
<a href="#">Section 203.8: sys.argv[0] is the name of the file being executed</a>	771
<a href="#">Section 203.9: Accessing int literals' attributes</a>	772
<a href="#">Section 203.10: Global Interpreter Lock (GIL) and blocking threads</a>	772
<a href="#">Section 203.11: Multiple return</a>	773
<a href="#">Section 203.12: Pythonic JSON keys</a>	773
<a href="#">Credits</a>	775
<a href="#">You may also like</a>	789

# About

Please feel free to share this PDF with anyone for free,  
latest version of this book can be downloaded from:  
<http://GoalKicker.com/PythonBook>

This *Python® Notes for Professionals* book is compiled from [Stack Overflow Documentation](#), the content is written by the beautiful people at Stack Overflow. Text content is released under Creative Commons BY-SA, see credits at the end of this book whom contributed to the various chapters. Images may be copyright of their respective owners unless otherwise specified

This is an unofficial free book created for educational purposes and is not affiliated with official Python® group(s) or company(s) nor Stack Overflow. All trademarks and registered trademarks are the property of their respective company owners

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to [web@petercv.com](mailto:web@petercv.com)