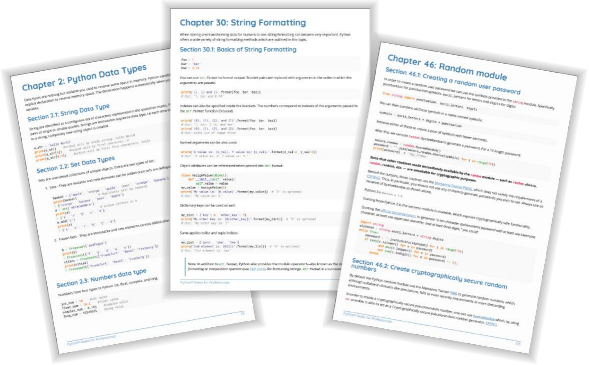
Notes for Professionals **Python®**

**Python**

**Notes for Professionals**

****

**700+ pages**

of professional hints and tricks

**GoalKicker.com** Free Programming Books

**Disclaimer**

This is an unocial free book created for educational purposes and is not aliated with ocial Python® group(s) or company(s). All trademarks and registered trademarks are

the property of their respective owners

**Contents**

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

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

Section 14.3: Filter as short-circuit check .................................................................................................................. 81 Section 14.4: Complementary function: filterfalse, ifilterfalse ................................................................................. 81 **Chapter 15: Arrays** ...................................................................................................................................................... 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 buer information through buer\_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 **Chapter 16: Dictionary** .............................................................................................................................................. 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 **Chapter 17: List** ............................................................................................................................................................. 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 **Chapter 18: List slicing (selecting parts of lists)** ....................................................................................... 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 **Chapter 19: Linked lists** .......................................................................................................................................... 110 Section 19.1: Single linked list example .................................................................................................................... 110 **Chapter 20: Linked List Node** ............................................................................................................................. 114 Section 20.1: Write a simple Linked List Node in python ....................................................................................... 114 **Chapter 21: Tuple** ...................................................................................................................................................... 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 **Chapter 22: Functions** ............................................................................................................................................. 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 **Chapter 23: Defining functions with list arguments** ............................................................................... 136 Section 23.1: Function and Call ................................................................................................................................. 136 **Chapter 24: Functional Programming in Python** ..................................................................................... 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 **Chapter 25: Partial functions** ............................................................................................................................. 138 Section 25.1: Raise the power ................................................................................................................................... 138 **Chapter 26: Decorators** ......................................................................................................................................... 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 **Chapter 27: Classes** ................................................................................................................................................. 145 Section 27.1: Introduction to classes ........................................................................................................................ 145

Section 27.2: Bound, unbound, and static methods .............................................................................................. 146 Section 27.3: Basic inheritance ................................................................................................................................ 148 Section 27.4: Monkey Patching ................................................................................................................................ 150 Section 27.5: New-style vs. old-style classes .......................................................................................................... 150 Section 27.6: Class methods: alternate initializers ................................................................................................. 151 Section 27.7: Multiple Inheritance ............................................................................................................................ 153 Section 27.8: Properties ............................................................................................................................................ 155 Section 27.9: Default values for instance variables ............................................................................................... 156 Section 27.10: Class and instance variables ........................................................................................................... 157 Section 27.11: Class composition .............................................................................................................................. 158 Section 27.12: Listing All Class Members ................................................................................................................. 159 Section 27.13: Singleton class ................................................................................................................................... 160 Section 27.14: Descriptors and Dotted Lookups .................................................................................................... 161

**Chapter 28: Metaclasses** ....................................................................................................................................... 162 Section 28.1: Basic Metaclasses ............................................................................................................................... 162 Section 28.2: Singletons using metaclasses ........................................................................................................... 163 Section 28.3: Using a metaclass .............................................................................................................................. 163 Section 28.4: Introduction to Metaclasses .............................................................................................................. 163 Section 28.5: Custom functionality with metaclasses ........................................................................................... 164 Section 28.6: The default metaclass ....................................................................................................................... 165

**Chapter 29: String Methods** ................................................................................................................................ 167 Section 29.1: Changing the capitalization of a string ............................................................................................. 167 Section 29.2: str.translate: Translating characters in a string .............................................................................. 168 Section 29.3: str.format and f-strings: Format values into a string ..................................................................... 168 Section 29.4: String module's useful constants ...................................................................................................... 169 Section 29.5: Stripping unwanted leading/trailing characters from a string ..................................................... 171 Section 29.6: Reversing a string .............................................................................................................................. 171 Section 29.7: Split a string based on a delimiter into a list of strings .................................................................. 172 Section 29.8: Replace all occurrences of one substring with another substring ................................................ 173 Section 29.9: Testing what a string is composed of .............................................................................................. 173 Section 29.10: String Contains .................................................................................................................................. 176 Section 29.11: Join a list of strings into one string .................................................................................................. 176 Section 29.12: Counting number of times a substring appears in a string ......................................................... 176 Section 29.13: Case insensitive string comparisons ............................................................................................... 177 Section 29.14: Justify strings .................................................................................................................................... 178 Section 29.15: Test the starting and ending characters of a string ..................................................................... 178 Section 29.16: Conversion between str or bytes data and unicode characters ................................................. 179

**Chapter 30: String Formatting** .......................................................................................................................... 181 Section 30.1: Basics of String Formatting ............................................................................................................... 181 Section 30.2: Alignment and padding ..................................................................................................................... 182 Section 30.3: Format literals (f-string) .................................................................................................................... 183 Section 30.4: Float formatting ................................................................................................................................. 183 Section 30.5: Named placeholders .......................................................................................................................... 184 Section 30.6: String formatting with datetime ....................................................................................................... 185 Section 30.7: Formatting Numerical Values ........................................................................................................... 185 Section 30.8: Nested formatting .............................................................................................................................. 186 Section 30.9: Format using Getitem and Getattr ................................................................................................... 186 Section 30.10: Padding and truncating strings, combined .................................................................................... 186 Section 30.11: Custom formatting for a class ......................................................................................................... 187 **Chapter 31: Conditionals** ....................................................................................................................................... 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

**Chapter 32: Loops** .................................................................................................................................................... 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 dierent portion of a list with dierent step size ............................................................ 202 Section 32.10: While Loop ......................................................................................................................................... 203

**Chapter 33: Using loops within functions** ..................................................................................................... 204 Section 33.1: Return statement inside loop in a function ....................................................................................... 204 **Chapter 34: Importing modules** ........................................................................................................................ 205 Section 34.1: Importing a module ............................................................................................................................ 205 Section 34.2: The \_\_all\_\_ 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: \_\_import\_\_() function ..................................................................................................................... 210 **Chapter 35: Dierence between Module and Package** ........................................................................ 211 Section 35.1: Modules ................................................................................................................................................ 211 Section 35.2: Packages ............................................................................................................................................. 211 **Chapter 36: Math Module** ..................................................................................................................................... 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 **Chapter 37: Complex math** .................................................................................................................................. 221 Section 37.1: Advanced complex arithmetic ........................................................................................................... 221 Section 37.2: Basic complex arithmetic ................................................................................................................... 222 **Chapter 38: Collections module** ........................................................................................................................ 223

Section 38.1: collections.Counter .............................................................................................................................. 223 Section 38.2: collections.OrderedDict ...................................................................................................................... 224 Section 38.3: collections.defaultdict ......................................................................................................................... 225 Section 38.4: collections.namedtuple ...................................................................................................................... 226 Section 38.5: collections.deque ................................................................................................................................ 227 Section 38.6: collections.ChainMap ......................................................................................................................... 228

**Chapter 39: Operator module** ............................................................................................................................ 230 Section 39.1: Itemgetter ............................................................................................................................................ 230 Section 39.2: Operators as alternative to an infix operator ................................................................................. 230 Section 39.3: Methodcaller ....................................................................................................................................... 230

**Chapter 40: JSON Module** ................................................................................................................................... 232 Section 40.1: Storing data in a file ........................................................................................................................... 232 Section 40.2: Retrieving data from a file ................................................................................................................ 232 Section 40.3: Formatting JSON output ................................................................................................................... 232 Section 40.4: `load` vs `loads`, `dump` vs `dumps` .................................................................................................. 233 Section 40.5: Calling `json.tool` from the command line to pretty-print JSON output ...................................... 234 Section 40.6: JSON encoding custom objects ....................................................................................................... 234 Section 40.7: Creating JSON from Python dict ...................................................................................................... 235 Section 40.8: Creating Python dict from JSON ...................................................................................................... 235

**Chapter 41: Sqlite3 Module** .................................................................................................................................. 236 Section 41.1: Sqlite3 - Not require separate server process .................................................................................. 236 Section 41.2: Getting the values from the database and Error handling ............................................................ 236

**Chapter 42: The os Module** ................................................................................................................................. 238 Section 42.1: makedirs - recursive directory creation ........................................................................................... 238 Section 42.2: Create a directory .............................................................................................................................. 239 Section 42.3: Get current directory .......................................................................................................................... 239 Section 42.4: Determine the name of the operating system ............................................................................... 239 Section 42.5: Remove a directory ........................................................................................................................... 239 Section 42.6: Follow a symlink (POSIX) ................................................................................................................... 239 Section 42.7: Change permissions on a file ............................................................................................................ 239

**Chapter 43: The locale Module** ......................................................................................................................... 240 Section 43.1: Currency Formatting US Dollars Using the locale Module ............................................................. 240 **Chapter 44: Itertools Module** ............................................................................................................................ 241 Section 44.1: Combinations method in Itertools Module ....................................................................................... 241 Section 44.2: itertools.dropwhile .............................................................................................................................. 241 Section 44.3: Zipping two iterators until they are both exhausted ...................................................................... 242 Section 44.4: Take a slice of a generator ............................................................................................................... 242 Section 44.5: Grouping items from an iterable object using a function .............................................................. 243 Section 44.6: itertools.takewhile ............................................................................................................................... 244 Section 44.7: itertools.permutations ........................................................................................................................ 244 Section 44.8: itertools.repeat .................................................................................................................................... 245 Section 44.9: Get an accumulated sum of numbers in an iterable ...................................................................... 245 Section 44.10: Cycle through elements in an iterator ............................................................................................ 245 Section 44.11: itertools.product ................................................................................................................................. 245 Section 44.12: itertools.count .................................................................................................................................... 246 Section 44.13: Chaining multiple iterators together ............................................................................................... 247 **Chapter 45: Asyncio Module** ............................................................................................................................... 248 Section 45.1: Coroutine and Delegation Syntax ..................................................................................................... 248 Section 45.2: Asynchronous Executors ................................................................................................................... 249

Section 45.3: Using UVLoop ..................................................................................................................................... 250 Section 45.4: Synchronization Primitive: Event ....................................................................................................... 250 Section 45.5: A Simple Websocket .......................................................................................................................... 251 Section 45.6: Common Misconception about asyncio .......................................................................................... 251

**Chapter 46: Random module** ............................................................................................................................. 253 Section 46.1: Creating a random user password ................................................................................................... 253 Section 46.2: Create cryptographically secure random numbers ....................................................................... 253 Section 46.3: Random and sequences: shue, choice and sample .................................................................... 254 Section 46.4: Creating random integers and floats: randint, randrange, random, and uniform ...................... 255 Section 46.5: Reproducible random numbers: Seed and State ............................................................................ 256 Section 46.6: Random Binary Decision ................................................................................................................... 257

**Chapter 47: Functools Module** .......................................................................................................................... 258 Section 47.1: partial ................................................................................................................................................... 258 Section 47.2: cmp\_to\_key ....................................................................................................................................... 258 Section 47.3: lru\_cache ............................................................................................................................................. 258 Section 47.4: total\_ordering ..................................................................................................................................... 259 Section 47.5: reduce .................................................................................................................................................. 260

**Chapter 48: The dis module** ............................................................................................................................... 261 Section 48.1: What is Python bytecode? ................................................................................................................. 261 Section 48.2: Constants in the dis module .............................................................................................................. 261 Section 48.3: Disassembling modules ..................................................................................................................... 261

**Chapter 49: The base64 Module** ...................................................................................................................... 263 Section 49.1: Encoding and Decoding Base64 ....................................................................................................... 264 Section 49.2: Encoding and Decoding Base32 ....................................................................................................... 265 Section 49.3: Encoding and Decoding Base16 ........................................................................................................ 265 Section 49.4: Encoding and Decoding ASCII85 ...................................................................................................... 266 Section 49.5: Encoding and Decoding Base85 ....................................................................................................... 266

**Chapter 50: Queue Module** .................................................................................................................................. 268 Section 50.1: Simple example ................................................................................................................................... 268 **Chapter 51: Deque Module** ................................................................................................................................... 269 Section 51.1: Basic deque using ................................................................................................................................ 269 Section 51.2: Available methods in deque ............................................................................................................... 269 Section 51.3: limit deque size .................................................................................................................................... 270 Section 51.4: Breadth First Search ........................................................................................................................... 270 **Chapter 52: Usage of "pip" module: PyPI Package Manager** ............................................................ 271 Section 52.1: Example use of commands ............................................................................................................... 271 Section 52.2: Handling ImportError Exception ....................................................................................................... 271 Section 52.3: Force install ......................................................................................................................................... 272 **Chapter 53: Webbrowser Module** ..................................................................................................................... 273 Section 53.1: Opening a URL with Default Browser ............................................................................................... 273 Section 53.2: Opening a URL with Dierent Browsers .......................................................................................... 274 **Chapter 54: pyautogui module** ......................................................................................................................... 275 Section 54.1: Mouse Functions .................................................................................................................................. 275 Section 54.2: Keyboard Functions ........................................................................................................................... 275 Section 54.3: ScreenShot And Image Recognition ................................................................................................. 275 **Chapter 55: Plotting with Matplotlib** .............................................................................................................. 276 Section 55.1: Plots with Common X-axis but dierent Y-axis : Using twinx() ....................................................... 276 Section 55.2: Plots with common Y-axis and dierent X-axis using twiny() ....................................................... 277 Section 55.3: A Simple Plot in Matplotlib ................................................................................................................. 279

Section 55.4: Adding more features to a simple plot : axis labels, title, axis ticks, grid, and legend ................ 280 Section 55.5: Making multiple plots in the same figure by superimposition similar to MATLAB ...................... 281 Section 55.6: Making multiple Plots in the same figure using plot superimposition with separate plot

commands ......................................................................................................................................................... 282 **Chapter 56: Comparisons** ..................................................................................................................................... 284 Section 56.1: Chain Comparisons ............................................................................................................................. 284 Section 56.2: Comparison by `is` vs `==` ................................................................................................................... 285 Section 56.3: Greater than or less than ................................................................................................................... 286 Section 56.4: Not equal to ........................................................................................................................................ 286 Section 56.5: Equal To ............................................................................................................................................... 287 Section 56.6: Comparing Objects ............................................................................................................................ 287 **Chapter 57: Sorting, Minimum and Maximum** ............................................................................................ 289 Section 57.1: Make custom classes orderable ........................................................................................................ 289 Section 57.2: Special case: dictionaries ................................................................................................................... 291 Section 57.3: Using the key argument .................................................................................................................... 292 Section 57.4: Default Argument to max, min .......................................................................................................... 292 Section 57.5: Getting a sorted sequence ................................................................................................................ 293 Section 57.6: Extracting N largest or N smallest items from an iterable ............................................................ 293 Section 57.7: Getting the minimum or maximum of several values .................................................................... 294 Section 57.8: Minimum and Maximum of a sequence ........................................................................................... 294 **Chapter 58: Variable Scope and Binding** ..................................................................................................... 295 Section 58.1: Nonlocal Variables .............................................................................................................................. 295 Section 58.2: Global Variables ................................................................................................................................. 295 Section 58.3: Local Variables ................................................................................................................................... 296 Section 58.4: The del command .............................................................................................................................. 297 Section 58.5: Functions skip class scope when looking up names ...................................................................... 298 Section 58.6: Local vs Global Scope ........................................................................................................................ 299 Section 58.7: Binding Occurrence ............................................................................................................................ 301 **Chapter 59: Basic Input and Output** ............................................................................................................... 302 Section 59.1: Using the print function ...................................................................................................................... 302 Section 59.2: Input from a File ................................................................................................................................. 302 Section 59.3: Read from stdin .................................................................................................................................. 304 Section 59.4: Using input() and raw\_input() .......................................................................................................... 304 Section 59.5: Function to prompt user for a number ............................................................................................ 304 Section 59.6: Printing a string without a newline at the end ................................................................................. 305 **Chapter 60: Files & Folders I/O** ......................................................................................................................... 307 Section 60.1: File modes ............................................................................................................................................ 307 Section 60.2: Reading a file line-by-line ................................................................................................................. 308 Section 60.3: Iterate files (recursively) .................................................................................................................... 309 Section 60.4: Getting the full contents of a file ...................................................................................................... 309 Section 60.5: Writing to a file ................................................................................................................................... 310 Section 60.6: Check whether a file or path exists .................................................................................................. 311 Section 60.7: Random File Access Using mmap .................................................................................................... 312 Section 60.8: Replacing text in a file ....................................................................................................................... 312 Section 60.9: Checking if a file is empty ................................................................................................................. 312 Section 60.10: Read a file between a range of lines .............................................................................................. 313 Section 60.11: Copy a directory tree ........................................................................................................................ 313 Section 60.12: Copying contents of one file to a dierent file .............................................................................. 313 **Chapter 61: Indexing and Slicing** ....................................................................................................................... 314 Section 61.1: Basic Slicing .......................................................................................................................................... 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: \_\_getitem\_\_, \_\_setitem\_\_ and \_\_delitem\_\_ .................................... 316 Section 61.6: Basic Indexing ...................................................................................................................................... 317

**Chapter 62: Generators** ......................................................................................................................................... 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 next() 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

**Chapter 63: Reduce** ................................................................................................................................................. 327 Section 63.1: Overview ............................................................................................................................................... 327 Section 63.2: Using reduce ....................................................................................................................................... 327 Section 63.3: Cumulative product ............................................................................................................................ 328 Section 63.4: Non short-circuit variant of any/all .................................................................................................. 328

**Chapter 64: Map Function** ................................................................................................................................... 329 Section 64.1: Basic use of map, itertools.imap and future\_builtins.map ............................................................. 329 Section 64.2: Mapping each value in an iterable ................................................................................................... 329 Section 64.3: Mapping values of dierent 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

**Chapter 65: Exponentiation** ................................................................................................................................. 335 Section 65.1: Exponentiation using builtins: \*\* and pow() ....................................................................................... 335 Section 65.2: Square root: math.sqrt() and cmath.sqrt ......................................................................................... 335 Section 65.3: Modular exponentiation: pow() with 3 arguments .......................................................................... 336 Section 65.4: Computing large integer roots ......................................................................................................... 336 Section 65.5: Exponentiation using the math module: math.pow() ..................................................................... 337 Section 65.6: Exponential function: math.exp() and cmath.exp() ......................................................................... 338 Section 65.7: Exponential function minus 1: math.expm1() .................................................................................... 338 Section 65.8: Magic methods and exponentiation: builtin, math and cmath ...................................................... 339 Section 65.9: Roots: nth-root with fractional exponents ....................................................................................... 340

**Chapter 66: Searching** ............................................................................................................................................ 341 Section 66.1: Searching for an element ................................................................................................................... 341 Section 66.2: Searching in custom classes: \_\_contains\_\_ and \_\_iter\_\_ .......................................................... 341 Section 66.3: Getting the index for strings: str.index(), str.rindex() and str.find(), str.rfind() .............................. 342 Section 66.4: Getting the index list and tuples: list.index(), tuple.index() .............................................................. 343 Section 66.5: Searching key(s) for a value in dict .................................................................................................. 343 Section 66.6: Getting the index for sorted sequences: bisect.bisect\_left() .......................................................... 344 Section 66.7: Searching nested sequences ............................................................................................................. 344

**Chapter 67: Counting** .............................................................................................................................................. 346 Section 67.1: Counting all occurence of all items in an iterable: collections.Counter ......................................... 346

Section 67.2: Getting the most common value(-s): collections.Counter.most\_common() ................................ 346 Section 67.3: Counting the occurrences of one item in a sequence: list.count() and tuple.count() .................. 346 Section 67.4: Counting the occurrences of a substring in a string: str.count() ................................................... 347 Section 67.5: Counting occurences in numpy array .............................................................................................. 347

**Chapter 68: Manipulating XML** .......................................................................................................................... 348 Section 68.1: Opening and reading using an ElementTree ................................................................................... 348 Section 68.2: Create and Build XML Documents .................................................................................................... 348 Section 68.3: Modifying an XML File ........................................................................................................................ 349 Section 68.4: Searching the XML with XPath .......................................................................................................... 349 Section 68.5: Opening and reading large XML files using iterparse (incremental parsing) ............................. 350

**Chapter 69: Parallel computation** .................................................................................................................... 351 Section 69.1: Using the multiprocessing module to parallelise tasks ................................................................... 351 Section 69.2: Using a C-extension to parallelize tasks .......................................................................................... 351 Section 69.3: Using Parent and Children scripts to execute code in parallel ...................................................... 351 Section 69.4: Using PyPar module to parallelize ................................................................................................... 352

**Chapter 70: Processes and Threads** ............................................................................................................... 353 Section 70.1: Global Interpreter Lock ....................................................................................................................... 353 Section 70.2: Running in Multiple Threads .............................................................................................................. 354 Section 70.3: Running in Multiple Processes ........................................................................................................... 355 Section 70.4: Sharing State Between Threads ....................................................................................................... 355 Section 70.5: Sharing State Between Processes .................................................................................................... 356

**Chapter 71: Multithreading** .................................................................................................................................. 357 Section 71.1: Basics of multithreading ...................................................................................................................... 357 Section 71.2: Communicating between threads ..................................................................................................... 358 Section 71.3: Creating a worker pool ....................................................................................................................... 359 Section 71.4: Advanced use of multithreads ........................................................................................................... 359 Section 71.5: Stoppable Thread with a while Loop ................................................................................................. 361

**Chapter 72: Writing extensions** ......................................................................................................................... 362 Section 72.1: Hello World with C Extension ............................................................................................................. 362 Section 72.2: C Extension Using c++ and Boost ..................................................................................................... 362 Section 72.3: Passing an open file to C Extensions ................................................................................................ 364

**Chapter 73: Unit Testing** ....................................................................................................................................... 365 Section 73.1: Test Setup and Teardown within a unittest.TestCase ..................................................................... 365 Section 73.2: Asserting on Exceptions ..................................................................................................................... 365 Section 73.3: Testing Exceptions .............................................................................................................................. 366 Section 73.4: Choosing Assertions Within Unittests ............................................................................................... 367 Section 73.5: Unit tests with pytest .......................................................................................................................... 368 Section 73.6: Mocking functions with unittest.mock.create\_autospec ................................................................ 371

**Chapter 74: Regular Expressions (Regex)** ................................................................................................... 373 Section 74.1: Matching the beginning of a string ................................................................................................... 373 Section 74.2: Searching ............................................................................................................................................ 374 Section 74.3: Precompiled patterns ......................................................................................................................... 374 Section 74.4: Flags .................................................................................................................................................... 375 Section 74.5: Replacing ............................................................................................................................................. 376 Section 74.6: Find All Non-Overlapping Matches ................................................................................................... 376 Section 74.7: Checking for allowed characters ...................................................................................................... 377 Section 74.8: Splitting a string using regular expressions ..................................................................................... 377 Section 74.9: Grouping .............................................................................................................................................. 377 Section 74.10: Escaping Special Characters ........................................................................................................... 378

Section 74.11: Match an expression only in specific locations ............................................................................... 379 Section 74.12: Iterating over matches using `re.finditer` ........................................................................................ 380 **Chapter 75: Incompatibilities moving from Python 2 to Python 3** .................................................. 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: Dierences 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 dierent types ........................................................................................................ 391 Section 75.11: .next() method on iterators renamed .............................................................................................. 392 Section 75.12: filter(), map() and zip() return iterators instead of sequences .................................................... 393 Section 75.13: Renamed modules ............................................................................................................................ 393 Section 75.14: Removed operators <> and ``, synonymous with != and repr() .................................................... 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: map() .................................................................................................................................................. 398 Section 75.20: The round() function tie-breaking and return type ...................................................................... 399 Section 75.21: File I/O ................................................................................................................................................ 400 Section 75.22: cmp 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: exec 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: hasattr function bug in Python 2 .................................................................................................... 404 **Chapter 76: Virtual environments** .................................................................................................................... 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 dierent version of python ................................................. 407 Section 76.4: Making virtual environments using Anaconda ................................................................................ 407 Section 76.5: Managing multiple virtual enviroments with virtualenvwrapper ................................................... 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 virtualenv with fish shell .......................................................................................................... 411 **Chapter 77: Copying data** .................................................................................................................................... 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 **Chapter 78: Context Managers (“with” Statement)** ............................................................................... 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

**Chapter 79: Hidden Features** ............................................................................................................................. 419 Section 79.1: Operator Overloading ........................................................................................................................ 419 **Chapter 80: Unicode and bytes** ........................................................................................................................ 420 Section 80.1: Encoding/decoding error handling .................................................................................................. 420 Section 80.2: File I/O ................................................................................................................................................. 420 Section 80.3: Basics ................................................................................................................................................... 421 **Chapter 81: The \_\_name\_\_ special variable** ............................................................................................ 423 Section 81.1: \_\_name\_\_ == '\_\_main\_\_' ................................................................................................................. 423 Section 81.2: Use in logging ...................................................................................................................................... 423 Section 81.3: function\_class\_or\_module.\_\_name\_\_ ........................................................................................... 423 **Chapter 82: Checking Path Existence and Permissions** ......................................................................... 425 Section 82.1: Perform checks using os.access ........................................................................................................ 425 **Chapter 83: Python Networking** ....................................................................................................................... 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 **Chapter 84: The Print Function** ......................................................................................................................... 429 Section 84.1: Print basics ........................................................................................................................................... 429 Section 84.2: Print parameters ................................................................................................................................ 430 **Chapter 85: os.path** ................................................................................................................................................. 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 **Chapter 86: Creating Python packages** ........................................................................................................ 434 Section 86.1: Introduction .......................................................................................................................................... 434 Section 86.2: Uploading to PyPI ............................................................................................................................... 434 Section 86.3: Making package executable ............................................................................................................. 436 **Chapter 87: Parsing Command Line arguments** ...................................................................................... 438 Section 87.1: Hello world in argparse ...................................................................................................................... 438 Section 87.2: Using command line arguments with argv ..................................................................................... 438 Section 87.3: Setting mutually exclusive arguments with argparse .................................................................... 439 Section 87.4: Basic example with docopt ............................................................................................................... 440 Section 87.5: Custom parser error message with argparse ................................................................................. 440 Section 87.6: Conceptual grouping of arguments with argparse.add\_argument\_group() ............................. 441 Section 87.7: Advanced example with docopt and docopt\_dispatch ................................................................. 442 **Chapter 88: HTML Parsing** ................................................................................................................................... 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

**Chapter 89: Subprocess Library** ....................................................................................................................... 446 Section 89.1: More flexibility with Popen ................................................................................................................. 446 Section 89.2: Calling External Commands .............................................................................................................. 447 Section 89.3: How to create the command list argument .................................................................................... 447

**Chapter 90: setup.py** .............................................................................................................................................. 448 Section 90.1: Purpose of setup.py ............................................................................................................................ 448 Section 90.2: Using source control metadata in setup.py .................................................................................... 448 Section 90.3: Adding command line scripts to your python package ................................................................ 449 Section 90.4: Adding installation options ................................................................................................................ 449

**Chapter 91: Sockets** ................................................................................................................................................. 451 Section 91.1: Raw Sockets on Linux .......................................................................................................................... 451 Section 91.2: Sending data via UDP ......................................................................................................................... 451 Section 91.3: Receiving data via UDP ...................................................................................................................... 452 Section 91.4: Sending data via TCP ......................................................................................................................... 452 Section 91.5: Multi-threaded TCP Socket Server .................................................................................................... 452

**Chapter 92: Recursion** ............................................................................................................................................ 455 Section 92.1: The What, How, and When of Recursion .......................................................................................... 455 Section 92.2: Tree exploration with recursion ........................................................................................................ 458 Section 92.3: Sum of numbers from 1 to n .............................................................................................................. 459 Section 92.4: Increasing the Maximum Recursion Depth ...................................................................................... 459 Section 92.5: Tail Recursion - Bad Practice ............................................................................................................ 460 Section 92.6: Tail Recursion Optimization Through Stack Introspection ............................................................ 460

**Chapter 93: Type Hints** .......................................................................................................................................... 462 Section 93.1: Adding types to a function ................................................................................................................. 462 Section 93.2: NamedTuple ....................................................................................................................................... 463 Section 93.3: Generic Types ..................................................................................................................................... 463 Section 93.4: Variables and Attributes .................................................................................................................... 463 Section 93.5: Class Members and Methods ............................................................................................................ 464 Section 93.6: Type hints for keyword arguments .................................................................................................. 464

**Chapter 94: pip: PyPI Package Manager** ..................................................................................................... 465 Section 94.1: Install Packages .................................................................................................................................. 465 Section 94.2: To list all packages installed using `pip` ........................................................................................... 465 Section 94.3: Upgrade Packages ............................................................................................................................. 465 Section 94.4: Uninstall Packages ............................................................................................................................. 466 Section 94.5: Updating all outdated packages on Linux ...................................................................................... 466 Section 94.6: Updating all outdated packages on Windows ................................................................................ 466 Section 94.7: Create a requirements.txt file of all packages on the system ....................................................... 466 Section 94.8: Using a certain Python version with pip .......................................................................................... 467 Section 94.9: Create a requirements.txt file of packages only in the current virtualenv .................................. 467 Section 94.10: Installing packages not yet on pip as wheels ................................................................................ 468

**Chapter 95: Exceptions** .......................................................................................................................................... 471 Section 95.1: Catching Exceptions ............................................................................................................................ 471 Section 95.2: Do not catch everything! ................................................................................................................... 471 Section 95.3: Re-raising exceptions ......................................................................................................................... 472 Section 95.4: Catching multiple exceptions ............................................................................................................ 472 Section 95.5: Exception Hierarchy ........................................................................................................................... 473 Section 95.6: Else ....................................................................................................................................................... 475 Section 95.7: Raising Exceptions .............................................................................................................................. 475 Section 95.8: Creating custom exception types ..................................................................................................... 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

**Chapter 96: Web scraping with Python** ......................................................................................................... 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

**Chapter 97: Distribution** ........................................................................................................................................ 483 Section 97.1: py2app ................................................................................................................................................. 483 Section 97.2: cx\_Freeze ............................................................................................................................................ 484

**Chapter 98: Property Objects** ............................................................................................................................ 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

**Chapter 99: Overloading** ....................................................................................................................................... 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

**Chapter 100: Debugging** ....................................................................................................................................... 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

**Chapter 101: Reading and Writing CSV** .......................................................................................................... 496 Section 101.1: Using pandas ...................................................................................................................................... 496 Section 101.2: Writing a TSV file ............................................................................................................................... 496

**Chapter 102: Dynamic code execution with `exec` and `eval`** ............................................................. 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

**Chapter 103: PyInstaller - Distributing Python Code** .............................................................................. 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

**Chapter 104: Iterables and Iterators** ............................................................................................................. 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

**Chapter 105: Data Visualization with Python** ............................................................................................. 504 Section 105.1: Seaborn ............................................................................................................................................... 504 Section 105.2: Matplotlib ........................................................................................................................................... 506 Section 105.3: Plotly ................................................................................................................................................... 507 Section 105.4: MayaVI ............................................................................................................................................... 509

**Chapter 106: The Interpreter (Command Line Console)** ....................................................................... 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

**Chapter 107: \*args and \*\*kwargs** ...................................................................................................................... 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

**Chapter 108: Garbage Collection** ...................................................................................................................... 518 Section 108.1: Reuse of primitive objects ................................................................................................................ 518 Section 108.2: Eects 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

**Chapter 109: Pickle data serialisation** ............................................................................................................ 523 Section 109.1: Using Pickle to serialize and deserialize an object ......................................................................... 523 Section 109.2: Customize Pickled Data ................................................................................................................... 523

**Chapter 110: urllib** ..................................................................................................................................................... 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

**Chapter 111: Binary Data** ....................................................................................................................................... 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

**Chapter 112: Python and Excel** ........................................................................................................................... 529 Section 112.1: Read the excel data using xlrd module ............................................................................................ 529 Section 112.2: Format Excel files with xlsxwriter ..................................................................................................... 529 Section 112.3: Put list data into a Excel's file ............................................................................................................ 530 Section 112.4: OpenPyXL ........................................................................................................................................... 531

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

Section 126.5: line\_profiler in command line .......................................................................................................... 560 **Chapter 127: User-Defined Methods** ............................................................................................................... 561 Section 127.1: Creating user-defined method objects ............................................................................................ 561 Section 127.2: Turtle example ................................................................................................................................... 562 **Chapter 128: Working around the Global Interpreter Lock (GIL)** ..................................................... 563 Section 128.1: Multiprocessing.Pool .......................................................................................................................... 563 Section 128.2: Cython nogil: ...................................................................................................................................... 564 **Chapter 129: Deployment** ..................................................................................................................................... 565 Section 129.1: Uploading a Conda Package ........................................................................................................... 565 **Chapter 130: Logging** .............................................................................................................................................. 567 Section 130.1: Introduction to Python Logging ....................................................................................................... 567 Section 130.2: Logging exceptions ........................................................................................................................... 568 **Chapter 131: Database Access** ............................................................................................................................ 571 Section 131.1: SQLite ................................................................................................................................................... 571 Section 131.2: Accessing MySQL database using MySQLdb ................................................................................. 576 Section 131.3: Connection .......................................................................................................................................... 577 Section 131.4: PostgreSQL Database access using psycopg2 .............................................................................. 578 Section 131.5: Oracle database ................................................................................................................................ 579 Section 131.6: Using sqlalchemy ............................................................................................................................... 580 **Chapter 132: Python HTTP Server** .................................................................................................................... 582 Section 132.1: Running a simple HTTP server ......................................................................................................... 582 Section 132.2: Serving files ........................................................................................................................................ 582 Section 132.3: Basic handling of GET, POST, PUT using BaseHTTPRequestHandler ......................................... 583 Section 132.4: Programmatic API of SimpleHTTPServer ....................................................................................... 584 **Chapter 133: Web Server Gateway Interface (WSGI)** ............................................................................. 586 Section 133.1: Server Object (Method) ..................................................................................................................... 586 **Chapter 134: Python Server Sent Events** ..................................................................................................... 587 Section 134.1: Flask SSE ............................................................................................................................................. 587 Section 134.2: Asyncio SSE ........................................................................................................................................ 587 **Chapter 135: Connecting Python to SQL Server** ....................................................................................... 588 Section 135.1: Connect to Server, Create Table, Query Data ................................................................................ 588

**Chapter 136: Sockets And Message Encryption/Decryption Between Client and Server** ............................................................................................................................................................................................ 589 Section 136.1: Server side Implementation .............................................................................................................. 589 Section 136.2: Client side Implementation ............................................................................................................... 591 **Chapter 137: Alternatives to switch statement from other languages** ........................................ 593 Section 137.1: Use what the language oers: the if/else construct ...................................................................... 593 Section 137.2: Use a dict of functions ...................................................................................................................... 593 Section 137.3: Use class introspection ..................................................................................................................... 594 Section 137.4: Using a context manager ................................................................................................................. 595 **Chapter 138: List Comprehensions** ................................................................................................................... 596 Section 138.1: Conditional List Comprehensions ..................................................................................................... 596 Section 138.2: List Comprehensions with Nested Loops ........................................................................................ 597 Section 138.3: Refactoring filter and map to list comprehensions ....................................................................... 598 Section 138.4: Nested List Comprehensions ........................................................................................................... 599 Section 138.5: Iterate two or more list simultaneously within list comprehension .............................................. 600 **Chapter 139: List destructuring (aka packing and unpacking)** ......................................................... 601 Section 139.1: Destructuring assignment ................................................................................................................. 601

Section 139.2: Packing function arguments ............................................................................................................ 602 Section 139.3: Unpacking function arguments ....................................................................................................... 604 **Chapter 140: Accessing Python source code and bytecode** .............................................................. 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 **Chapter 141: Mixins** ................................................................................................................................................... 607 Section 141.1: Mixin ..................................................................................................................................................... 607 Section 141.2: Overriding Methods in Mixins ............................................................................................................ 608 **Chapter 142: Attribute Access** ............................................................................................................................ 609 Section 142.1: Basic Attribute Access using the Dot Notation ............................................................................... 609 Section 142.2: Setters, Getters & Properties ............................................................................................................ 609 **Chapter 143: ArcPy** .................................................................................................................................................. 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 **Chapter 144: Abstract Base Classes (abc)** .................................................................................................. 612 Section 144.1: Setting the ABCMeta metaclass ....................................................................................................... 612 Section 144.2: Why/How to use ABCMeta and @abstractmethod ...................................................................... 612 **Chapter 145: Plugin and Extension Classes** ................................................................................................. 614 Section 145.1: Mixins ................................................................................................................................................... 614 Section 145.2: Plugins with Customized Classes ..................................................................................................... 615 **Chapter 146: Websockets** ..................................................................................................................................... 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 **Chapter 147: Immutable datatypes(int, float, str, tuple and frozensets)** .................................. 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 **Chapter 148: String representations of class instances: \_\_str\_\_ and \_\_repr\_\_ methods** ........................................................................................................................................................................ 621 Section 148.1: Motivation ........................................................................................................................................... 621 Section 148.2: Both methods implemented, eval-round-trip style \_\_repr\_\_() .................................................. 625 **Chapter 149: Polymorphism** ................................................................................................................................ 626 Section 149.1: Duck Typing ....................................................................................................................................... 626 Section 149.2: Basic Polymorphism ......................................................................................................................... 626 **Chapter 150: Non-ocial Python implementations** ............................................................................... 629 Section 150.1: IronPython .......................................................................................................................................... 629 Section 150.2: Jython ................................................................................................................................................ 629 Section 150.3: Transcrypt ......................................................................................................................................... 630 **Chapter 151: 2to3 tool** ............................................................................................................................................. 633 Section 151.1: Basic Usage ......................................................................................................................................... 633 **Chapter 152: Abstract syntax tree** ................................................................................................................... 635 Section 152.1: Analyze functions in a python script ................................................................................................ 635 **Chapter 153: Unicode** .............................................................................................................................................. 637 Section 153.1: Encoding and decoding .................................................................................................................... 637 **Chapter 154: Python Serial Communication (pyserial)** ......................................................................... 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

**Chapter 155: Neo4j and Cypher using Py2Neo** ......................................................................................... 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

**Chapter 156: Basic Curses with Python** .......................................................................................................... 642 Section 156.1: The wrapper() helper function ......................................................................................................... 642 Section 156.2: Basic Invocation Example ................................................................................................................ 642

**Chapter 157: Performance optimization** ....................................................................................................... 643 Section 157.1: Code profiling ..................................................................................................................................... 643 **Chapter 158: Templates in python** ................................................................................................................... 645 Section 158.1: Simple data output program using template ................................................................................. 645 Section 158.2: Changing delimiter ............................................................................................................................ 645 **Chapter 159: Pillow** ................................................................................................................................................... 646 Section 159.1: Read Image File ................................................................................................................................. 646 Section 159.2: Convert files to JPEG ........................................................................................................................ 646 **Chapter 160: The pass statement** .................................................................................................................... 647 Section 160.1: Ignore an exception ........................................................................................................................... 647 Section 160.2: Create a new Exception that can be caught .................................................................................. 647 **Chapter 161: py.test** ................................................................................................................................................. 648 Section 161.1: Setting up py.test ................................................................................................................................ 648 Section 161.2: Intro to Test Fixtures .......................................................................................................................... 648 Section 161.3: Failing Tests ........................................................................................................................................ 651 **Chapter 162: Heapq** ................................................................................................................................................. 653 Section 162.1: Largest and smallest items in a collection ...................................................................................... 653 Section 162.2: Smallest item in a collection ............................................................................................................ 653 **Chapter 163: tkinter** ................................................................................................................................................. 655 Section 163.1: Geometry Managers .......................................................................................................................... 655 Section 163.2: A minimal tkinter Application ........................................................................................................... 656 **Chapter 164: CLI subcommands with precise help output** .................................................................. 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 **Chapter 165: PostgreSQL** ...................................................................................................................................... 661 Section 165.1: Getting Started ................................................................................................................................... 661 **Chapter 166: Python Persistence** ...................................................................................................................... 662 Section 166.1: Python Persistence ............................................................................................................................ 662 Section 166.2: Function utility for save and load .................................................................................................... 663 **Chapter 167: Turtle Graphics** .............................................................................................................................. 664 Section 167.1: Ninja Twist (Turtle Graphics) ............................................................................................................ 664 **Chapter 168: Design Patterns** ............................................................................................................................. 665 Section 168.1: Introduction to design patterns and Singleton Pattern ................................................................. 665 Section 168.2: Strategy Pattern ................................................................................................................................ 667

Section 168.3: Proxy ................................................................................................................................................... 668 **Chapter 169: Multidimensional arrays** ........................................................................................................... 670 Section 169.1: Lists in lists .......................................................................................................................................... 670 Section 169.2: Lists in lists in lists in.. ........................................................................................................................ 670 **Chapter 170: Audio** ................................................................................................................................................... 672 Section 170.1: Working with WAV files ..................................................................................................................... 672 Section 170.2: Convert any soundfile with python and mpeg ............................................................................ 672 Section 170.3: Playing Windows' beeps ................................................................................................................... 672 Section 170.4: Audio With Pyglet .............................................................................................................................. 673 **Chapter 171: Pyglet** .................................................................................................................................................. 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 **Chapter 172: Flask** .................................................................................................................................................... 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 **Chapter 173: groupby()** .......................................................................................................................................... 681 Section 173.1: Example 4 ............................................................................................................................................ 681 Section 173.2: Example 2 ........................................................................................................................................... 681 Section 173.3: Example 3 ........................................................................................................................................... 682 **Chapter 174: pygame** ............................................................................................................................................. 684 Section 174.1: Pygame's mixer module .................................................................................................................... 684 Section 174.2: Installing pygame ............................................................................................................................. 685 **Chapter 175: hashlib** ................................................................................................................................................ 686 Section 175.1: MD5 hash of a string ......................................................................................................................... 686 Section 175.2: algorithm provided by OpenSSL ..................................................................................................... 687 **Chapter 176: getting start with GZip** .............................................................................................................. 688 Section 176.1: Read and write GNU zip files ............................................................................................................ 688 **Chapter 177: ctypes** ................................................................................................................................................. 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 **Chapter 178: Creating a Windows service using Python** ...................................................................... 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 **Chapter 179: Mutable vs Immutable (and Hashable) in Python** ....................................................... 695 Section 179.1: Mutable vs Immutable ....................................................................................................................... 695 Section 179.2: Mutable and Immutable as Arguments .......................................................................................... 697 **Chapter 180: Python speed of program** ....................................................................................................... 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

**Chapter 181: configparser** .................................................................................................................................... 703 Section 181.1: Creating configuration file programatically .................................................................................... 703 Section 181.2: Basic usage ........................................................................................................................................ 703

**Chapter 182: Commonwealth Exceptions** ..................................................................................................... 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

**Chapter 183: Optical Character Recognition** .............................................................................................. 710 Section 183.1: PyTesseract ........................................................................................................................................ 710 Section 183.2: PyOCR ................................................................................................................................................ 710

**Chapter 184: graph-tool** ....................................................................................................................................... 712 Section 184.1: PyDotPlus ............................................................................................................................................ 712 Section 184.2: PyGraphviz ......................................................................................................................................... 712

**Chapter 185: Python Virtual Environment - virtualenv** ......................................................................... 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

**Chapter 186: sys** ........................................................................................................................................................ 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

**Chapter 187: virtual environment with virtualenvwrapper** ................................................................ 717 Section 187.1: Create virtual environment with virtualenvwrapper ...................................................................... 717 **Chapter 188: Create virtual environment with virtualenvwrapper in windows** ........................ 719 Section 188.1: Virtual environment with virtualenvwrapper for windows ............................................................ 719 **Chapter 189: Python Requests Post** ................................................................................................................ 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 **Chapter 190: Python Lex-Yacc** ........................................................................................................................... 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 **Chapter 191: ChemPy - python package** ....................................................................................................... 734 Section 191.1: Parsing formulae ................................................................................................................................ 734 Section 191.2: Balancing stoichiometry of a chemical reaction ............................................................................ 734

Section 191.3: Balancing reactions ........................................................................................................................... 734 Section 191.4: Chemical equilibria ............................................................................................................................ 735 Section 191.5: Ionic strength ...................................................................................................................................... 735 Section 191.6: Chemical kinetics (system of ordinary dierential equations) ..................................................... 735

**Chapter 192: pyaudio** .............................................................................................................................................. 737 Section 192.1: Callback Mode Audio I/O .................................................................................................................. 737 Section 192.2: Blocking Mode Audio I/O ................................................................................................................. 738

**Chapter 193: shelve** .................................................................................................................................................. 740 Section 193.1: Creating a new Shelf .......................................................................................................................... 740 Section 193.2: Sample code for shelve .................................................................................................................... 741 Section 193.3: To summarize the interface (key is a string, data is an arbitrary object): .................................. 741 Section 193.4: Write-back ......................................................................................................................................... 741

**Chapter 194: IoT Programming with Python and Raspberry PI** ....................................................... 743 Section 194.1: Example - Temperature sensor ........................................................................................................ 743 **Chapter 195: kivy - Cross-platform Python Framework for NUI Development** ....................... 746 Section 195.1: First App .............................................................................................................................................. 746 **Chapter 196: Call Python from C#** .................................................................................................................... 748 Section 196.1: Python script to be called by C# application .................................................................................. 748 Section 196.2: C# code calling Python script .......................................................................................................... 748 **Chapter 197: Similarities in syntax, Dierences in meaning: Python vs. JavaScript** ............. 750 Section 197.1: `in` with lists ......................................................................................................................................... 750 **Chapter 198: Raise Custom Errors / Exceptions** ....................................................................................... 751 Section 198.1: Custom Exception .............................................................................................................................. 751 Section 198.2: Catch custom Exception ................................................................................................................... 751

**Chapter 199: Pandas Transform: Preform operations on groups and concatenate the results** ............................................................................................................................................................................. 752 Section 199.1: Simple transform ............................................................................................................................... 752 Section 199.2: Multiple results per group ................................................................................................................ 753 **Chapter 200: Security and Cryptography** ................................................................................................... 754 Section 200.1: Secure Password Hashing ............................................................................................................... 754 Section 200.2: Calculating a Message Digest ........................................................................................................ 754 Section 200.3: Available Hashing Algorithms ......................................................................................................... 754 Section 200.4: File Hashing ...................................................................................................................................... 755 Section 200.5: Generating RSA signatures using pycrypto .................................................................................. 755 Section 200.6: Asymmetric RSA encryption using pycrypto ................................................................................ 756 Section 200.7: Symmetric encryption using pycrypto .......................................................................................... 757 **Chapter 201: Secure Shell Connection in Python** ...................................................................................... 758 Section 201.1: ssh connection ................................................................................................................................... 758 **Chapter 202: Python Anti-Patterns** ................................................................................................................. 759 Section 202.1: Overzealous except clause .............................................................................................................. 759 Section 202.2: Looking before you leap with processor-intensive function ....................................................... 759 **Chapter 203: Common Pitfalls** ........................................................................................................................... 761 Section 203.1: List multiplication and common references ................................................................................... 761 Section 203.2: Mutable default argument .............................................................................................................. 764 Section 203.3: Changing the sequence you are iterating over ............................................................................ 765 Section 203.4: Integer and String identity .............................................................................................................. 768 Section 203.5: Dictionaries are unordered ............................................................................................................. 769 Section 203.6: Variable leaking in list comprehensions and for loops ................................................................ 770

Section 203.7: Chaining of or operator ................................................................................................................... 770 Section 203.8: sys.argv[0] is the name of the file being executed ...................................................................... 771 Section 203.9: Accessing int literals' attributes ...................................................................................................... 772 Section 203.10: Global Interpreter Lock (GIL) and blocking threads ................................................................... 772 Section 203.11: Multiple return .................................................................................................................................. 773 Section 203.12: Pythonic JSON keys ....................................................................................................................... 773

**Credits** ............................................................................................................................................................................ 775 **You may also like** ...................................................................................................................................................... 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

Python® Notes for Professionals 1

**Chapter 1: Getting started with Python Language**

**Python 3.x**

**Version Release Date**

[3.7] 2017-05-08

3.6 2016-12-23

3.5 2015-09-13

3.4 2014-03-17

3.3 2012-09-29

3.2 2011-02-20

3.1 2009-06-26

3.0 2008-12-03

**Python 2.x**

**Version Release Date**

2.7 2010-07-03

2.6 2008-10-02

2.5 2006-09-19

2.4 2004-11-30

2.3 2003-07-29

2.2 2001-12-21

2.1 2001-04-15

2.0 2000-10-16

**Section 1.1: Getting Started**

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library.

Two major versions of Python are currently in active use:

Python 3.x is the current version and is under active development.

Python 2.x is the legacy version and will receive only security updates until 2020. No new features will be implemented. Note that many projects still use Python 2, although migrating to Python 3 is getting easier.

You can download and install either version of Python here. See Python 3 vs. Python 2 for a comparison between them. In addition, some third-parties offer re-packaged versions of Python that add commonly used libraries and other features to ease setup for common use cases, such as math, data analysis or scientific use. See the list at the official site.

**Verify if Python is installed**

To confirm that Python was installed correctly, you can verify that by running the following command in your favorite terminal (If you are using Windows OS, you need to add path of python to the environment variable before using it in command prompt):

$ python --version

Python 3.x Version ≥ 3.0

If you have *Python 3* installed, and it is your default version (see **Troubleshooting** for more details) you should see

Python® Notes for Professionals 2

something like this:

$ python --version

Python 3.6.0

Python 2.x Version ≤ 2.7

If you have *Python 2* installed, and it is your default version (see **Troubleshooting** for more details) you should see something like this:

$ python --version

Python 2.7.13

If you have installed Python 3, but $ python --version outputs a Python 2 version, you also have Python 2 installed. This is often the case on MacOS, and many Linux distributions. Use $ python3 instead to explicitly use the Python 3 interpreter.

**Hello, World in Python using IDLE**

IDLE is a simple editor for Python, that comes bundled with Python.

**How to create Hello, World program in IDLE**

Open IDLE on your system of choice.

In older versions of Windows, it can be found at All Programs under the Windows menu. In Windows 8+, search for IDLE or find it in the apps that are present in your system.

On Unix-based (including Mac) systems you can open it from the shell by typing $ idle python\_file.py.

It will open a shell with options along the top.

In the shell, there is a prompt of three right angle brackets:

>>>

Now write the following code in the prompt:

>>> **print**("Hello, World")

| Enter |
| --- |

Hit .

>>> **print**("Hello, World")

Hello, World

**Hello World Python file**

Create a new file hello.py that contains the following line:

Python 3.x Version ≥ 3.0

**print**('Hello, World')

Python 2.x Version ≥ 2.6

You can use the Python 3 **print** function in Python 2 with the following **import** statement: **from** \_\_future\_\_ **import** print\_function

Python 2 has a number of functionalities that can be optionally imported from Python 3 using the \_\_future\_\_

Python® Notes for Professionals 3

module, as discussed here.

Python 2.x Version ≤ 2.7

If using Python 2, you may also type the line below. Note that this is not valid in Python 3 and thus not recommended because it reduces cross-version code compatibility.

**print** 'Hello, World'

In your terminal, navigate to the directory containing the file hello.py.

| Enter |
| --- |

Type python hello.py, then hit the key.

$ python hello.py

Hello, World

You should see Hello, World printed to the console.

You can also substitute hello.py with the path to your file. For example, if you have the file in your home directory and your user is "user" on Linux, you can type python /home/user/hello.py.

**Launch an interactive Python shell**

By executing (running) the python command in your terminal, you are presented with an interactive Python shell. This is also known as the Python Interpreter or a REPL (for 'Read Evaluate Print Loop').

$ python

Python 2.7.12 (default, Jun 28 2016, 08:46:01)

[GCC 6.1.1 20160602] on linux

Type "help", "copyright", "credits" **or** "license" **for** more information.

>>> **print** 'Hello, World'

Hello, World

>>>

If you want to run Python 3 from your terminal, execute the command python3.

$ python3

Python 3.6.0 (default, Jan 13 2017, 00:00:00)

[GCC 6.1.1 20160602] on linux

Type "help", "copyright", "credits" **or** "license" **for** more information.

>>> **print**('Hello, World')

Hello, World

>>>

Alternatively, start the interactive prompt and load file with python -i <file.py>.

In command line, run:

$ python -i hello.py

"Hello World"

>>>

There are multiple ways to close the Python shell:

>>> exit()

Python® Notes for Professionals 4

or

>>> quit()

| CTRL + D |
| --- |

Alternatively, will close the shell and put you back on your terminal's command line.

If you want to cancel a command you're in the middle of typing and get back to a clean command prompt, while

| CTRL + C |
| --- |

staying inside the Interpreter shell, use .

Try an interactive Python shell online.

**Other Online Shells**

Various websites provide online access to Python shells.

Online shells may be useful for the following purposes:

Run a small code snippet from a machine which lacks python installation(smartphones, tablets etc). Learn or teach basic Python.

Solve online judge problems.

Examples:

Disclaimer: documentation author(s) are not affiliated with any resources listed below.

https://www.python.org/shell/ - The online Python shell hosted by the official Python website. https://ideone.com/ - Widely used on the Net to illustrate code snippet behavior.

https://repl.it/languages/python3 - Powerful and simple online compiler, IDE and interpreter. Code, compile, and run code in Python.

https://www.tutorialspoint.com/execute\_python\_online.php - Full-featured UNIX shell, and a user-friendly project explorer.

http://rextester.com/l/python3\_online\_compiler - Simple and easy to use IDE which shows execution time **Run commands as a string**

Python can be passed arbitrary code as a string in the shell:

$ python -c 'print("Hello, World")'

Hello, World

This can be useful when concatenating the results of scripts together in the shell.

**Shells and Beyond**

*Package Management* - The PyPA recommended tool for installing Python packages is PIP. To install, on your command line execute pip install <the package name>. For instance, pip install numpy. (Note: On windows you must add pip to your PATH environment variables. To avoid this, use python -m pip install <the package name>)

*Shells* - So far, we have discussed different ways to run code using Python's native interactive shell. Shells use Python's interpretive power for experimenting with code real-time. Alternative shells include IDLE - a pre-bundled GUI, IPython - known for extending the interactive experience, etc.

*Programs* - For long-term storage you can save content to .py files and edit/execute them as scripts or programs

Python® Notes for Professionals 5

with external tools e.g. shell, IDEs (such as PyCharm), Jupyter notebooks, etc. Intermediate users may use these tools; however, the methods discussed here are sufficient for getting started.

Python tutor allows you to step through Python code so you can visualize how the program will flow, and helps you to understand where your program went wrong.

PEP8 defines guidelines for formatting Python code. Formatting code well is important so you can quickly read what the code does.

**Section 1.2: Creating variables and assigning values** To create a variable in Python, all you need to do is specify the variable name, and then assign a value to it. **<variable** name**>** = **<value>**

Python uses

=

to assign values to variables. There's no need to declare a variable in advance (or to assign a data type to it), assigning a value to a variable itself declares and initializes the variable with that value. There's no way to declare a variable without assigning it an initial value.

*# Integer*

a = 2

**print**(a)

*# Output: 2*

*# Integer*

b = 9223372036854775807

**print**(b)

*# Output: 9223372036854775807*

*# Floating point*

pi = 3.14

**print**(pi)

*# Output: 3.14*

*# String*

c = 'A'

**print**(c)

*# Output: A*

*# String*

name = 'John Doe'

**print**(name)

*# Output: John Doe*

*# Boolean*

q = True

**print**(q)

*# Output: True*

*# Empty value or null data type*

x = None

**print**(x)

*# Output: None*

Variable assignment works from left to right. So the following will give you an syntax error.

Python® Notes for Professionals 6

0 = x

=> Output: SyntaxError: can't assign to literal

You can not use python's keywords as a valid variable name. You can see the list of keyword by:

**import** keyword

**print**(keyword.kwlist)

Rules for variable naming:

1. Variables names must start with a letter or an underscore.

x = True *# valid*

\_y = True *# valid*

9x = False *# starts with numeral*

=> SyntaxError: invalid syntax

$y = False *# starts with symbol*

=> SyntaxError: invalid syntax

2. The remainder of your variable name may consist of letters, numbers and underscores. has\_0\_in\_it = "Still Valid"

3. Names are case sensitive.

x = 9

y = X\*5

=>NameError: name 'X' **is not** defined

Even though there's no need to specify a data type when declaring a variable in Python, while allocating the necessary area in memory for the variable, the Python interpreter automatically picks the most suitable built-in type for it:

a = 2

**print**(type(a))

*# Output: <type 'int'>*

b = 9223372036854775807

**print**(type(b))

*# Output: <type 'int'>*

pi = 3.14

**print**(type(pi))

*# Output: <type 'float'>*

c = 'A'

**print**(type(c))

*# Output: <type 'str'>*

name = 'John Doe'

**print**(type(name))

*# Output: <type 'str'>*

q = True

**print**(type(q))

*# Output: <type 'bool'>*

Python® Notes for Professionals 7

x = None

**print**(type(x))

*# Output: <type 'NoneType'>*

Now you know the basics of assignment, let's get this subtlety about assignment in python out of the way. When you use

=

to do an assignment operation, what's on the left of

=

is a ***name*** for the ***object*** on the right. Finally, what

=

does is assign the ***reference*** of the object on the right to the ***name*** on the left.

That is:

a\_name = an\_object *# "a\_name" is now a name for the reference to the object "an\_object"*

So, from many assignment examples above, if we pick pi = 3.14, then pi is **a** name (not **the** name, since an object can have multiple names) for the object 3.14. If you don't understand something below, come back to this point and read this again! Also, you can take a look at this for a better understanding.

You can assign multiple values to multiple variables in one line. Note that there must be the same number of arguments on the right and left sides of the

=

operator:

a, b, c = 1, 2, 3

**print**(a, b, c)

*# Output: 1 2 3*

a, b, c = 1, 2

=> Traceback (most recent call last):

=> File "name.py", line N, **in** <module>

=> a, b, c = 1, 2

=> ValueError: need more than 2 values to unpack

a, b = 1, 2, 3

=> Traceback (most recent call last):

=> File "name.py", line N, **in** <module>

=> a, b = 1, 2, 3

=> ValueError: too many values to unpack

The error in last example can be obviated by assigning remaining values to equal number of arbitrary variables. This dummy variable can have any name, but it is conventional to use the underscore (\_) for assigning unwanted values:

a, b, \_ = 1, 2, 3

**print**(a, b)

*# Output: 1, 2*

Note that the number of \_ and number of remaining values must be equal. Otherwise 'too many values to unpack

Python® Notes for Professionals 8

error' is thrown as above:

a, b, \_ = 1,2,3,4

=>Traceback (most recent call last):

=>File "name.py", line N, **in** <module>

=>a, b, \_ = 1,2,3,4

=>ValueError: too many values to unpack (expected 3)

You can also assign a single value to several variables simultaneously.

a = b = c = 1

**print**(a, b, c)

*# Output: 1 1 1*

When using such cascading assignment, it is important to note that all three variables a, b and c refer to the *same object* in memory, an int object with the value of 1. In other words, a, b and c are three different names given to the same int object. Assigning a different object to one of them afterwards doesn't change the others, just as expected:

a = b = c = 1 *# all three names a, b and c refer to same int object with value 1* **print**(a, b, c)

*# Output: 1 1 1*

b = 2 *# b now refers to another int object, one with a value of 2*

**print**(a, b, c)

*# Output: 1 2 1 # so output is as expected.*

The above is also true for mutable types (like list, dict, etc.) just as it is true for immutable types (like int, string, tuple, etc.):

x = y = [7, 8, 9] *# x and y refer to the same list object just created, [7, 8, 9]* x = [13, 8, 9] *# x now refers to a different list object just created, [13, 8, 9]* **print**(y) *# y still refers to the list it was first assigned*

*# Output: [7, 8, 9]*

So far so good. Things are a bit different when it comes to *modifying* the object (in contrast to *assigning* the name to a different object, which we did above) when the cascading assignment is used for mutable types. Take a look below, and you will see it first hand:

x = y = [7, 8, 9] *# x and y are two different names for the same list object just created, [7, 8, 9]*

x[0] = 13 *# we are updating the value of the list [7, 8, 9] through one of its names, x in this case*

**print**(y) *# printing the value of the list using its other name*

*# Output: [13, 8, 9] # hence, naturally the change is reflected*

Nested lists are also valid in python. This means that a list can contain another list as an element.

x = [1, 2, [3, 4, 5], 6, 7] *# this is nested list*

**print** x[2]

*# Output: [3, 4, 5]*

**print** x[2][1]

*# Output: 4*

Lastly, variables in Python do not have to stay the same type as which they were first defined -- you can simply use =

to assign a new value to a variable, even if that value is of a different type.

Python® Notes for Professionals 9

a = 2

**print**(a)

*# Output: 2*

a = "New value"

**print**(a)

*# Output: New value*

If this bothers you, think about the fact that what's on the left of

=

is just a name for an object. First you call the int object with value 2 a, then you change your mind and decide to give the name a to a string object, having value 'New value'. Simple, right?

**Section 1.3: Block Indentation**

Python uses indentation to define control and loop constructs. This contributes to Python's readability, however, it requires the programmer to pay close attention to the use of whitespace. Thus, editor miscalibration could result in code that behaves in unexpected ways.

Python uses the colon symbol (:) and indentation for showing where blocks of code begin and end (If you come from another language, do not confuse this with somehow being related to the ternary operator). That is, blocks in Python, such as functions, loops, **if** clauses and other constructs, have no ending identifiers. All blocks start with a colon and then contain the indented lines below it.

For example:

**def** my\_function(): *# This is a function definition. Note the colon (:)*

a = 2 *# This line belongs to the function because it's indented* **return** a *# This line also belongs to the same function*

**print**(my\_function()) *# This line is OUTSIDE the function block*

or

**if** a > b: *# If block starts here*

**print**(a) *# This is part of the if block*

**else**: *# else must be at the same level as if*

**print**(b) *# This line is part of the else block*

Blocks that contain exactly one single-line statement may be put on the same line, though this form is generally not considered good style:

**if** a > b: **print**(a)

**else**: **print**(b)

Attempting to do this with more than a single statement will *not* work:

**if** x > y: y = x

**print**(y) *# IndentationError: unexpected indent*

**if** x > y: **while** y != z: y -= 1 *# SyntaxError: invalid syntax*

An empty block causes an IndentationError. Use **pass** (a command that does nothing) when you have a block with no content:

Python® Notes for Professionals 10

**def** will\_be\_implemented\_later():

**pass**

**Spaces vs. Tabs**

In short: **always** use 4 spaces for indentation.

Using tabs exclusively is possible but PEP 8, the style guide for Python code, states that spaces are preferred. Python 3.x Version ≥ 3.0

Python 3 disallows mixing the use of tabs and spaces for indentation. In such case a compile-time error is generated: Inconsistent use of tabs **and** spaces **in** indentation and the program will not run.

Python 2.x Version ≤ 2.7

Python 2 allows mixing tabs and spaces in indentation; this is strongly discouraged. The tab character completes the previous indentation to be a multiple of **8** spaces. Since it is common that editors are configured to show tabs as multiple of **4** spaces, this can cause subtle bugs.

Citing PEP 8:

When invoking the Python 2 command line interpreter with the -t option, it issues warnings about code that illegally mixes tabs and spaces. When using -tt these warnings become errors. These options are highly recommended!

Many editors have "tabs to spaces" configuration. When configuring the editor, one should differentiate between

| Tab |
| --- |

the tab *character* ('\t') and the key.

The tab *character* should be configured to show 8 spaces, to match the language semantics - at least in cases when (accidental) mixed indentation is possible. Editors can also automatically convert the tab character to spaces.

| Tab |
| --- |

However, it might be helpful to configure the editor so that pressing the key will insert 4 spaces, instead of inserting a tab character.

Python source code written with a mix of tabs and spaces, or with non-standard number of indentation spaces can be made pep8-conformant using autopep8. (A less powerful alternative comes with most Python installations: reindent.py)

**Section 1.4: Datatypes**

**Built-in Types**

**Booleans**

bool: A boolean value of either True or False. Logical operations like **and**, **or**, **not** can be performed on booleans.

x **or** y *# if x is False then y otherwise x*

x **and** y *# if x is False then x otherwise y*

**not** x *# if x is True then False, otherwise True*

In Python 2.x and in Python 3.x, a boolean is also an int. The bool type is a subclass of the int type and True and False are its only instances:

Python® Notes for Professionals 11

issubclass(bool, int) *# True*

isinstance(True, bool) *# True*

isinstance(False, bool) *# True*

If boolean values are used in arithmetic operations, their integer values (1 and 0 for True and False) will be used to return an integer result:

True + False == 1 *# 1 + 0 == 1*

True \* True == 1 *# 1 \* 1 == 1*

**Numbers**

int: Integer number

a = 2

b = 100

c = 123456789

d = 38563846326424324

Integers in Python are of arbitrary sizes.

Note: in older versions of Python, a long type was available and this was distinct from int. The two have been unified.

float: Floating point number; precision depends on the implementation and system architecture, for CPython the float datatype corresponds to a C double.

a = 2.0

b = 100.e0

c = 123456789.e1

complex: Complex numbers

a = 2 + 1j

b = 100 + 10j

The <, <=, > and >= operators will raise a TypeError exception when any operand is a complex number.

**Strings**

Python 3.x Version ≥ 3.0

str: a **unicode string**. The type of 'hello'

bytes: a **byte string**. The type of b'hello'

Python 2.x Version ≤ 2.7

str: a **byte string**. The type of 'hello'

bytes: synonym for str

unicode: a **unicode string**. The type of u'hello'

**Sequences and collections**

Python differentiates between ordered sequences and unordered collections (such as set and dict).

Python® Notes for Professionals 12

strings (str, bytes, unicode) are sequences

reversed: A reversed order of str with reversed function

a = reversed('hello')

tuple: An ordered collection of n values of any type (n >= 0).

a = (1, 2, 3)

b = ('a', 1, 'python', (1, 2))

b[2] = 'something else' *# returns a TypeError*

Supports indexing; immutable; hashable if all its members are hashable

list: An ordered collection of n values (n >= 0)

a = [1, 2, 3]

b = ['a', 1, 'python', (1, 2), [1, 2]]

b[2] = 'something else' *# allowed*

Not hashable; mutable.

set: An unordered collection of unique values. Items must be hashable.

a = {1, 2, 'a'}

dict: An unordered collection of unique key-value pairs; keys must be hashable.

a = {1: 'one',

2: 'two'}

b = {'a': [1, 2, 3],

'b': 'a string'}

An object is hashable if it has a hash value which never changes during its lifetime (it needs a \_\_hash\_\_() method), and can be compared to other objects (it needs an \_\_eq\_\_() method). Hashable objects which compare equality must have the same hash value.

**Built-in constants**

In conjunction with the built-in datatypes there are a small number of built-in constants in the built-in namespace:

True: The true value of the built-in type bool

False: The false value of the built-in type bool

None: A singleton object used to signal that a value is absent.

Ellipsis or ...: used in core Python3+ anywhere and limited usage in Python2.7+ as part of array notation. numpy and related packages use this as a 'include everything' reference in arrays.

NotImplemented: a singleton used to indicate to Python that a special method doesn't support the specific arguments, and Python will try alternatives if available.

a = None *# No value will be assigned. Any valid datatype can be assigned later*

Python® Notes for Professionals 13

Python 3.x Version ≥ 3.0

None doesn't have any natural ordering. Using ordering comparison operators (<, <=, >=, >) isn't supported anymore and will raise a TypeError.

Python 2.x Version ≤ 2.7

None is always less than any number (None < -32 evaluates to True).

**Testing the type of variables**

In python, we can check the datatype of an object using the built-in function type.

a = '123'

**print**(type(a))

*# Out: <class 'str'>*

b = 123

**print**(type(b))

*# Out: <class 'int'>*

In conditional statements it is possible to test the datatype with isinstance. However, it is usually not encouraged to rely on the type of the variable.

i = 7

**if** isinstance(i, int):

i += 1

**elif** isinstance(i, str):

i = int(i)

i += 1

For information on the differences between type() and isinstance() read: Differences between isinstance and type in Python

To test if something is of NoneType:

x = None

**if** x **is** None:

**print**('Not a surprise, I just defined x as None.')

**Converting between datatypes**

You can perform explicit datatype conversion.

For example, '123' is of str type and it can be converted to integer using int function.

a = '123'

b = int(a)

Converting from a float string such as '123.456' can be done using float function.

a = '123.456'

b = float(a)

c = int(a) *# ValueError: invalid literal for int() with base 10: '123.456'* d = int(b) *# 123*

You can also convert sequence or collection types

a = 'hello'

Python® Notes for Professionals 14

list(a) *# ['h', 'e', 'l', 'l', 'o']*

set(a) *# {'o', 'e', 'l', 'h'}*

tuple(a) *# ('h', 'e', 'l', 'l', 'o')*

**Explicit string type at definition of literals**

With one letter labels just in front of the quotes you can tell what type of string you want to define.

b'foo bar': results bytes in Python 3, str in Python 2

u'foo bar': results str in Python 3, unicode in Python 2

'foo bar': results str

r'foo bar': results so called raw string, where escaping special characters is not necessary, everything is taken verbatim as you typed

normal = 'foo**\n**bar' *# foo*

*# bar*

escaped = 'foo**\\**nbar' *# foo\nbar*

raw = r'foo**\n**bar' *# foo\nbar*

**Mutable and Immutable Data Types**

An object is called *mutable* if it can be changed. For example, when you pass a list to some function, the list can be changed:

**def** f(m):

m.append(3) *# adds a number to the list. This is a mutation.*

x = [1, 2]

f(x)

x == [1, 2] *# False now, since an item was added to the list*

An object is called *immutable* if it cannot be changed in any way. For example, integers are immutable, since there's no way to change them:

**def** bar():

x = (1, 2)

g(x)

x == (1, 2) *# Will always be True, since no function can change the object (1, 2)*

Note that **variables** themselves are mutable, so we can reassign the *variable* x, but this does not change the object that x had previously pointed to. It only made x point to a new object.

Data types whose instances are mutable are called *mutable data types*, and similarly for immutable objects and datatypes.

Examples of immutable Data Types:

int, long, float, complex

str

bytes

tuple

frozenset

Examples of mutable Data Types:

bytearray

list

set

Python® Notes for Professionals 15

dict

**Section 1.5: Collection Types**

There are a number of collection types in Python. While types such as int and str hold a single value, collection types hold multiple values.

**Lists**

The list type is probably the most commonly used collection type in Python. Despite its name, a list is more like an array in other languages, mostly JavaScript. In Python, a list is merely an ordered collection of valid Python values. A list can be created by enclosing values, separated by commas, in square brackets:

int\_list = [1, 2, 3]

string\_list = ['abc', 'defghi']

A list can be empty:

empty\_list = []

The elements of a list are not restricted to a single data type, which makes sense given that Python is a dynamic language:

mixed\_list = [1, 'abc', True, 2.34, None]

A list can contain another list as its element:

nested\_list = [['a', 'b', 'c'], [1, 2, 3]]

The elements of a list can be accessed via an *index*, or numeric representation of their position. Lists in Python are *zero-indexed* meaning that the first element in the list is at index 0, the second element is at index 1 and so on:

names = ['Alice', 'Bob', 'Craig', 'Diana', 'Eric']

**print**(names[0]) *# Alice*

**print**(names[2]) *# Craig*

Indices can also be negative which means counting from the end of the list (-1 being the index of the last element). So, using the list from the above example:

**print**(names[-1]) *# Eric*

**print**(names[-4]) *# Bob*

Lists are mutable, so you can change the values in a list:

names[0] = 'Ann'

**print**(names)

*# Outputs ['Ann', 'Bob', 'Craig', 'Diana', 'Eric']*

Besides, it is possible to add and/or remove elements from a list:

Append object to end of list with L.append(object), returns None.

names = ['Alice', 'Bob', 'Craig', 'Diana', 'Eric']

names.append("Sia")

**print**(names)

Python® Notes for Professionals 16

*# Outputs ['Alice', 'Bob', 'Craig', 'Diana', 'Eric', 'Sia']*

Add a new element to list at a specific index. L.insert(index, object)

names.insert(1, "Nikki")

**print**(names)

*# Outputs ['Alice', 'Nikki', 'Bob', 'Craig', 'Diana', 'Eric', 'Sia']*

Remove the first occurrence of a value with L.remove(value), returns None

names.remove("Bob")

**print**(names) *# Outputs ['Alice', 'Nikki', 'Craig', 'Diana', 'Eric', 'Sia']*

Get the index in the list of the first item whose value is x. It will show an error if there is no such item.

name.index("Alice")

0

Count length of list

len(names)

6

count occurrence of any item in list

a = [1, 1, 1, 2, 3, 4]

a.count(1)

3

Reverse the list

a.reverse()

[4, 3, 2, 1, 1, 1]

*# or*

a[::-1]

[4, 3, 2, 1, 1, 1]

Remove and return item at index (defaults to the last item) with L.pop([index]), returns the item names.pop() *# Outputs 'Sia'*

You can iterate over the list elements like below:

**for** element **in** my\_list:

**print** (element)

**Tuples**

A tuple is similar to a list except that it is fixed-length and immutable. So the values in the tuple cannot be changed nor the values be added to or removed from the tuple. Tuples are commonly used for small collections of values that will not need to change, such as an IP address and port. Tuples are represented with parentheses instead of square brackets:

ip\_address = ('10.20.30.40', 8080)

Python® Notes for Professionals 17

The same indexing rules for lists also apply to tuples. Tuples can also be nested and the values can be any valid Python valid.

A tuple with only one member must be defined (note the comma) this way:

one\_member\_tuple = ('Only member',)

or

one\_member\_tuple = 'Only member', *# No brackets*

or just using tuple syntax

one\_member\_tuple = tuple(['Only member'])

**Dictionaries**

A dictionary in Python is a collection of key-value pairs. The dictionary is surrounded by curly braces. Each pair is separated by a comma and the key and value are separated by a colon. Here is an example:

state\_capitals = {

'Arkansas': 'Little Rock',

'Colorado': 'Denver',

'California': 'Sacramento',

'Georgia': 'Atlanta'

}

To get a value, refer to it by its key:

ca\_capital = state\_capitals['California']

You can also get all of the keys in a dictionary and then iterate over them:

**for** k **in** state\_capitals.keys():

**print**('{} is the capital of {}'.format(state\_capitals[k], k))

Dictionaries strongly resemble JSON syntax. The native json module in the Python standard library can be used to convert between JSON and dictionaries.

**set**

A set is a collection of elements with no repeats and without insertion order but sorted order. They are used in situations where it is only important that some things are grouped together, and not what order they were included. For large groups of data, it is much faster to check whether or not an element is in a set than it is to do the same for a list.

Defining a set is very similar to defining a dictionary:

first\_names = {'Adam', 'Beth', 'Charlie'}

Or you can build a set using an existing list:

my\_list = [1,2,3]

my\_set = set(my\_list)

Python® Notes for Professionals 18

Check membership of the set using **in**:

**if** name **in** first\_names:

**print**(name)

You can iterate over a set exactly like a list, but remember: the values will be in a arbitrary, implementation-defined order.

**defaultdict**

A defaultdict is a dictionary with a default value for keys, so that keys for which no value has been explicitly defined can be accessed without errors. defaultdict is especially useful when the values in the dictionary are collections (lists, dicts, etc) in the sense that it does not need to be initialized every time when a new key is used.

A defaultdict will never raise a KeyError. Any key that does not exist gets the default value returned. For example, consider the following dictionary

>>> state\_capitals = {

'Arkansas': 'Little Rock',

'Colorado': 'Denver',

'California': 'Sacramento',

'Georgia': 'Atlanta'

}

If we try to access a non-existent key, python returns us an error as follows

>>> state\_capitals['Alabama']

Traceback (most recent call last):

File "<ipython-input-61-236329695e6f>", line 1, **in** <module>

state\_capitals['Alabama']

KeyError: 'Alabama'

Let us try with a defaultdict. It can be found in the collections module.

>>> **from** collections **import** defaultdict

>>> state\_capitals = defaultdict(**lambda**: 'Boston')

What we did here is to set a default value (**Boston**) in case the give key does not exist. Now populate the dict as before:

>>> state\_capitals['Arkansas'] = 'Little Rock'

>>> state\_capitals['California'] = 'Sacramento'

>>> state\_capitals['Colorado'] = 'Denver'

>>> state\_capitals['Georgia'] = 'Atlanta'

If we try to access the dict with a non-existent key, python will return us the default value i.e. Boston

>>> state\_capitals['Alabama']

'Boston'

and returns the created values for existing key just like a normal dictionary

>>> state\_capitals['Arkansas']

Python® Notes for Professionals 19

'Little Rock'

**Section 1.6: IDLE - Python GUI**

IDLE is Python’s Integrated Development and Learning Environment and is an alternative to the command line. As the name may imply, IDLE is very useful for developing new code or learning python. On Windows this comes with the Python interpreter, but in other operating systems you may need to install it through your package manager.

The main purposes of IDLE are:

Multi-window text editor with syntax highlighting, autocompletion, and smart indent

Python shell with syntax highlighting

Integrated debugger with stepping, persistent breakpoints, and call stack visibility

Automatic indentation (useful for beginners learning about Python's indentation)

Saving the Python program as .py files and run them and edit them later at any them using IDLE.

In IDLE, hit F5 or run Python Shell to launch an interpreter. Using IDLE can be a better learning experience for new users because code is interpreted as the user writes.

Note that there are lots of alternatives, see for example this discussion or this list.

**Troubleshooting**

**Windows**

If you're on Windows, the default command is python. If you receive a "'python' is not recognized" error, the most likely cause is that Python's location is not in your system's PATH environment variable. This can be accessed by right-clicking on 'My Computer' and selecting 'Properties' or by navigating to 'System' through 'Control Panel'. Click on 'Advanced system settings' and then 'Environment Variables...'. Edit the PATH variable to include the directory of your Python installation, as well as the Script folder (usually

C:\Python27;C:\Python27\Scripts). This requires administrative privileges and may require a restart.

When using multiple versions of Python on the same machine, a possible solution is to rename one of the python.exe files. For example, naming one version python27.exe would cause python27 to become the Python command for that version.

You can also use the Python Launcher for Windows, which is available through the installer and comes by default. It allows you to select the version of Python to run by using py -[x.y] instead of python[x.y]. You can use the latest version of Python 2 by running scripts with py -2 and the latest version of Python 3 by running scripts with py -3.

**Debian/Ubuntu/MacOS**

This section assumes that the location of the python executable has been added to the PATH environment variable.

If you're on Debian/Ubuntu/MacOS, open the terminal and type python for Python 2.x or python3 for Python 3.x.

Type which python to see which Python interpreter will be used.

Python® Notes for Professionals 20

**Arch Linux**

The default Python on Arch Linux (and descendants) is Python 3, so use python or python3 for Python 3.x and python2 for Python 2.x.

**Other systems**

Python 3 is sometimes bound to python instead of python3. To use Python 2 on these systems where it is installed, you can use python2.

**Section 1.7: User Input**

**Interactive input**

To get input from the user, use the input function (**note**: in Python 2.x, the function is called raw\_input instead, although Python 2.x has its own version of input that is completely different):

Python 2.x Version ≥ 2.3

name = raw\_input("What is your name? ")

*# Out: What is your name? \_*

**Security Remark** Do not use input() in Python2 - the entered text will be evaluated as if it were a Python expression (equivalent to eval(input()) in Python3), which might easily become a vulnerability. See this article for further information on the risks of using this function.

Python 3.x Version ≥ 3.0

name = input("What is your name? ")

*# Out: What is your name? \_*

The remainder of this example will be using Python 3 syntax.

The function takes a string argument, which displays it as a prompt and returns a string. The above code provides a prompt, waiting for the user to input.

name = input("What is your name? ")

*# Out: What is your name?*

If the user types "Bob" and hits enter, the variable name will be assigned to the string "Bob":

name = input("What is your name? ")

*# Out: What is your name? Bob*

**print**(name)

*# Out: Bob*

Note that the input is always of type str, which is important if you want the user to enter numbers. Therefore, you need to convert the str before trying to use it as a number:

x = input("Write a number:")

*# Out: Write a number: 10*

x / 2

Python® Notes for Professionals 21

*# Out: TypeError: unsupported operand type(s) for /: 'str' and 'int'*

float(x) / 2

*# Out: 5.0*

NB: It's recommended to use **try**/**except** blocks to catch exceptions when dealing with user inputs. For instance, if your code wants to cast a raw\_input into an int, and what the user writes is uncastable, it raises a ValueError.

**Section 1.8: Built in Modules and Functions**

A module is a file containing Python definitions and statements. Function is a piece of code which execute some logic.

>>> pow(2,3) *#8*

To check the built in function in python we can use dir(). If called without an argument, return the names in the current scope. Else, return an alphabetized list of names comprising (some of) the attribute of the given object, and of attributes reachable from it.

>>> dir(\_\_builtins\_\_)

[

'ArithmeticError',

'AssertionError',

'AttributeError',

'BaseException',

'BufferError',

'BytesWarning',

'DeprecationWarning',

'EOFError',

'Ellipsis',

'EnvironmentError',

'Exception',

'False',

'FloatingPointError',

'FutureWarning',

'GeneratorExit',

'IOError',

'ImportError',

'ImportWarning',

'IndentationError',

'IndexError',

'KeyError',

'KeyboardInterrupt',

'LookupError',

'MemoryError',

'NameError',

'None',

'NotImplemented',

'NotImplementedError',

'OSError',

'OverflowError',

'PendingDeprecationWarning',

'ReferenceError',

'RuntimeError',

'RuntimeWarning',

'StandardError',

'StopIteration',

'SyntaxError',

'SyntaxWarning',

'SystemError',

Python® Notes for Professionals 22

'SystemExit',

'TabError',

'True',

'TypeError',

'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError',

'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',

'ValueError',

'Warning',

'ZeroDivisionError', '\_\_debug\_\_',

'\_\_doc\_\_',

'\_\_import\_\_',

'\_\_name\_\_',

'\_\_package\_\_',

'abs',

'all',

'any',

'apply',

'basestring',

'bin',

'bool',

'buffer',

'bytearray',

'bytes',

'callable',

'chr',

'classmethod',

'cmp',

'coerce',

'compile',

'complex',

'copyright',

'credits',

'delattr',

'dict',

'dir',

'divmod',

'enumerate',

'eval',

'execfile',

'exit',

'file',

'filter',

'float',

'format',

'frozenset',

'getattr',

'globals',

'hasattr',

'hash',

'help',

'hex',

'id',

'input',

'int',

'intern',

'isinstance',

Python® Notes for Professionals 23

'issubclass',

'iter',

'len',

'license',

'list',

'locals',

'long',

'map',

'max',

'memoryview',

'min',

'next',

'object',

'oct',

'open',

'ord',

'pow',

'print',

'property',

'quit',

'range',

'raw\_input',

'reduce',

'reload',

'repr',

'reversed',

'round',

'set',

'setattr',

'slice',

'sorted',

'staticmethod',

'str',

'sum',

'super',

'tuple',

'type',

'unichr',

'unicode',

'vars',

'xrange',

'zip'

]

To know the functionality of any function, we can use built in function help .

>>> help(max)

Help on built-**in** function max **in** module \_\_builtin\_\_:

max(...)

max(iterable[, key=func]) -> value

max(a, b, c, ...[, key=func]) -> value

With a single iterable argument, **return** its largest item.

With two **or** more arguments, **return** the largest argument.

Built in modules contains extra functionalities.For example to get square root of a number we need to include math module.

>>> **import** math

>>> math.sqrt(16) *# 4.0*

Python® Notes for Professionals 24

To know all the functions in a module we can assign the functions list to a variable, and then print the variable.

>>> **import** math

>>> dir(math)

['\_\_doc\_\_', '\_\_name\_\_', '\_\_package\_\_', 'acos', 'acosh',

'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',

'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1',

'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma',

'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10',

'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt',

'tan', 'tanh', 'trunc']

it seems \_\_doc\_\_ is useful to provide some documentation in, say, functions

>>> math.\_\_doc\_\_

'This module is always available. It provides access to the**\n**mathematical

functions defined by the C standard.'

In addition to functions, documentation can also be provided in modules. So, if you have a file named helloWorld.py like this:

"""This is the module docstring."""

**def** sayHello():

"""This is the function docstring."""

**return** 'Hello World'

You can access its docstrings like this:

>>> **import** helloWorld

>>> helloWorld.\_\_doc\_\_

'This is the module docstring.'

>>> helloWorld.sayHello.\_\_doc\_\_

'This is the function docstring.'

For any user defined type, its attributes, its class's attributes, and recursively the attributes of its class's base classes can be retrieved using dir()

>>> **class** MyClassObject(object):

... **pass**

...

>>> dir(MyClassObject)

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_doc\_\_', '\_\_format\_\_', '\_\_getattribute\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_module\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_']

Any data type can be simply converted to string using a builtin function called str. This function is called by default when a data type is passed to **print**

>>> str(123) *# "123"*

**Section 1.9: Creating a module**

A module is an importable file containing definitions and statements.

A module can be created by creating a .py file.

Python® Notes for Professionals 25

*# hello.py*

**def** say\_hello():

**print**("Hello!")

Functions in a module can be used by importing the module.

For modules that you have made, they will need to be in the same directory as the file that you are importing them into. (However, you can also put them into the Python lib directory with the pre-included modules, but should be avoided if possible.)

$ python

>>> **import** hello

>>> hello.say\_hello()

=> "Hello!"

Modules can be imported by other modules.

*# greet.py*

**import** hello

hello.say\_hello()

Specific functions of a module can be imported.

*# greet.py*

**from** hello **import** say\_hello

say\_hello()

Modules can be aliased.

*# greet.py*

**import** hello **as** ai

ai.say\_hello()

A module can be stand-alone runnable script.

*# run\_hello.py*

**if** \_\_name\_\_ == '\_\_main\_\_':

**from** hello **import** say\_hello

say\_hello()

Run it!

$ python run\_hello.py

=> "Hello!"

If the module is inside a directory and needs to be detected by python, the directory should contain a file named \_\_init\_\_.py.

**Section 1.10: Installation of Python 2.7.x and 3.x**

**Note**: Following instructions are written for Python 2.7 (unless specified): instructions for Python 3.x are similar.

**WINDOWS**

Python® Notes for Professionals 26

First, download the latest version of Python 2.7 from the official Website (https://www.python.org/downloads/). Version is provided as an MSI package. To install it manually, just double-click the file.

By default, Python installs to a directory:

C:\Python27\

Warning: installation does not automatically modify the PATH environment variable.

Assuming that your Python installation is in C:\Python27, add this to your PATH:

C:\Python27\;C:\Python27\Scripts\

Now to check if Python installation is valid write in cmd:

python --version

**Python 2.x and 3.x Side-By-Side**

To install and use both Python 2.x and 3.x side-by-side on a Windows machine:

1. Install Python 2.x using the MSI installer.

Ensure Python is installed for all users.

Optional: add Python to PATH to make Python 2.x callable from the command-line using python. 2. Install Python 3.x using its respective installer.

Again, ensure Python is installed for all users.

Optional: add Python to PATH to make Python 3.x callable from the command-line using python. This may override Python 2.x PATH settings, so double-check your PATH and ensure it's configured to your preferences.

Make sure to install the py launcher for all users.

Python 3 will install the Python launcher which can be used to launch Python 2.x and Python 3.x interchangeably from the command-line:

P:\>py -3

Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)] on win32 Type "help", "copyright", "credits" **or** "license" **for** more information.

>>>

C:\>py -2

Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 Intel)] on win32 Type "help", "copyright", "credits" **or** "license" **for** more information.

>>>

To use the corresponding version of pip for a specific Python version, use:

C:\>py -3 -m pip -V

pip 9.0.1 **from** C:\Python36\lib\site-packages (python 3.6)

C:\>py -2 -m pip -V

pip 9.0.1 **from** C:\Python27\lib\site-packages (python 2.7)

Python® Notes for Professionals 27

**LINUX**

The latest versions of CentOS, Fedora, Redhat Enterprise (RHEL) and Ubuntu come with Python 2.7. To install Python 2.7 on linux manually, just do the following in terminal:

**wget** --no-check-certificate https:**//**www.python.org**/**ftp**/**python**/**2.7.X**/**Python-2.7.X.tgz **tar** -xzf Python-2.7.X.tgz

**cd** Python-2.7.X

.**/**configure

**make**

**sudo make install**

Also add the path of new python in PATH environment variable. If new python is in /root/python-2.7.X then run export PATH = $PATH:/root/python-2.7.X

Now to check if Python installation is valid write in terminal:

python --version

*Ubuntu (From Source)*

If you need Python 3.6 you can install it from source as shown below (Ubuntu 16.10 and 17.04 have 3.6 version in the universal repository). Below steps have to be followed for Ubuntu 16.04 and lower versions:

**sudo** apt **install** build-essential checkinstall

**sudo** apt **install** libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm dev libc6-dev libbz2-dev

**wget** https:**//**www.python.org**/**ftp**/**python**/**3.6.1**/**Python-3.6.1.tar.xz

**tar** xvf Python-3.6.1.tar.xz

**cd** Python-3.6.1**/**

.**/**configure --enable-optimizations

**sudo make** altinstall

**macOS**

As we speak, macOS comes installed with Python 2.7.10, but this version is outdated and slightly modified from the regular Python.

The version of Python that ships with OS X is great for learning but it’s not good for development. The version shipped with OS X may be out of date from the official current Python release, which is considered the stable production version. (source)

Install Homebrew:

/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)" Install Python 2.7:

brew install python

For Python 3.x, use the command brew install python3 instead.

Python® Notes for Professionals 28

**Section 1.11: String function - str() and repr()**

There are two functions that can be used to obtain a readable representation of an object.

repr(x) calls x.\_\_repr\_\_(): a representation of x. eval will usually convert the result of this function back to the original object.

str(x) calls x.\_\_str\_\_(): a human-readable string that describes the object. This may elide some technical detail. **repr()**

For many types, this function makes an attempt to return a string that would yield an object with the same value when passed to eval(). Otherwise, the representation is a string enclosed in angle brackets that contains the name of the type of the object along with additional information. This often includes the name and address of the object.

**str()**

For strings, this returns the string itself. The difference between this and repr(object) is that str(object) does not always attempt to return a string that is acceptable to eval(). Rather, its goal is to return a printable or 'human readable' string. If no argument is given, this returns the empty string, ''.

Example 1:

s = """w'o"w"""

repr(s) *# Output: '\'w\\\'o"w\''*

str(s) *# Output: 'w\'o"w'*

eval(str(s)) == s *# Gives a SyntaxError*

eval(repr(s)) == s *# Output: True*

Example 2:

**import** datetime

today = datetime.datetime.now()

str(today) *# Output: '2016-09-15 06:58:46.915000'*

repr(today) *# Output: 'datetime.datetime(2016, 9, 15, 6, 58, 46, 915000)'*

When writing a class, you can override these methods to do whatever you want:

**class** Represent(object):

**def** \_\_init\_\_(self, x, y):

self.x, self.y = x, y

**def** \_\_repr\_\_(self):

**return** "Represent(x={},y=**\"**{}**\"**)".format(self.x, self.y)

**def** \_\_str\_\_(self):

**return** "Representing x as {} and y as {}".format(self.x, self.y)

Using the above class we can see the results:

r = Represent(1, "Hopper")

**print**(r) *# prints \_\_str\_\_*

**print**(r.\_\_repr\_\_) *# prints \_\_repr\_\_: '<bound method Represent.\_\_repr\_\_ of*

*Represent(x=1,y="Hopper")>'*

rep = r.\_\_repr\_\_() *# sets the execution of \_\_repr\_\_ to a new variable*

**print**(rep) *# prints 'Represent(x=1,y="Hopper")'*

Python® Notes for Professionals 29

r2 = eval(rep) *# evaluates rep*

**print**(r2) *# prints \_\_str\_\_ from new object*

**print**(r2 == r) *# prints 'False' because they are different objects*

**Section 1.12: Installing external modules using pip**

pip is your friend when you need to install any package from the plethora of choices available at the python package index (PyPI). pip is already installed if you're using Python 2 >= 2.7.9 or Python 3 >= 3.4 downloaded from python.org. For computers running Linux or another \*nix with a native package manager, pip must often be manually installed.

On instances with both Python 2 and Python 3 installed, pip often refers to Python 2 and pip3 to Python 3. Using pip will only install packages for Python 2 and pip3 will only install packages for Python 3.

**Finding / installing a package**

Searching for a package is as simple as typing

$ pip search <query>

*# Searches for packages whose name or summary contains <query>*

Installing a package is as simple as typing *(in a terminal / command-prompt, not in the Python interpreter)*

$ pip install [package\_name] *# latest version of the package*

$ pip install [package\_name]==x.x.x *# specific version of the package*

$ pip install '[package\_name]>=x.x.x' *# minimum version of the package*

where x.x.x is the version number of the package you want to install.

When your server is behind proxy, you can install package by using below command:

$ pip --proxy http://<server address>:<port> install

**Upgrading installed packages**

When new versions of installed packages appear they are not automatically installed to your system. To get an overview of which of your installed packages have become outdated, run:

$ pip list --outdated

To upgrade a specific package use

$ pip install [package\_name] --upgrade

Updating all outdated packages is not a standard functionality of pip.

**Upgrading pip**

You can upgrade your existing pip installation by using the following commands

On Linux or macOS X:

$ pip install -U pip

Python® Notes for Professionals 30

You may need to use sudo with pip on some Linux Systems

On Windows:

py -m pip install -U pip

or

python -m pip install -U pip

For more information regarding pip do read here.

**Section 1.13: Help Utility**

Python has several functions built into the interpreter. If you want to get information of keywords, built-in functions, modules or topics open a Python console and enter:

>>> help()

You will receive information by entering keywords directly:

>>> help(help)

**or** within the utility:

help> help

which will show an explanation:

Help on \_Helper **in** module \_sitebuiltins object:

**class** \_Helper(builtins.object)

| Define the builtin 'help'.

|

| This **is** a wrapper around pydoc.help that provides a helpful message | when 'help' **is** typed at the Python interactive prompt.

|

| Calling help() at the Python prompt starts an interactive help session. | Calling help(thing) prints help **for** the python object 'thing'.

|

| Methods defined here:

|

| \_\_call\_\_(self, \*args, \*\*kwds)

|

| \_\_repr\_\_(self)

|

| ---------------------------------------------------------------------- | Data descriptors defined here:

|

| \_\_dict\_\_

| dictionary **for** instance variables (**if** defined)

|

| \_\_weakref\_\_

| list of weak references to the object (**if** defined)

Python® Notes for Professionals 31

You can also request subclasses of modules:

help(pymysql.connections)

You can use help to access the docstrings of the different modules you have imported, e.g., try the following: >>> help(math)

and you'll get an error

>>> **import** math

>>> help(math)

And now you will get a list of the available methods in the module, but only AFTER you have imported it. Close the helper with quit

Python® Notes for Professionals 32

**Chapter 2: Python Data Types**

Data types are nothing but variable you used to reserve some space in memory. Python variables do not need an explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.

**Section 2.1: String Data Type**

String are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Strings are immutable sequence data type, i.e each time one makes any changes to a string, completely new string object is created.

a\_str = 'Hello World'

**print**(a\_str) *#output will be whole string. Hello World*

**print**(a\_str[0]) *#output will be first character. H*

**print**(a\_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 set :

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

basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}

**print**(basket) *# duplicates will be removed*

> {'orange', 'banana', 'pear', 'apple'}

a = set('abracadabra')

**print**(a) *# unique letters in a*

> {'a', 'r', 'b', 'c', 'd'}

a.add('z')

**print**(a)

> {'a', 'c', 'r', 'b', 'z', 'd'}

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

b = frozenset('asdfagsa')

**print**(b)

> frozenset({'f', 'g', 'd', 'a', 's'})

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 = 3.14j *#complex value*

long\_num = 1234567L *#long value*

Python® Notes for Professionals 33

**Section 2.4: List Data Type**

A list contains items separated by commas and enclosed within square brackets [].lists are almost similar to arrays in C. One difference is that all the items belonging to a list can be of different data type.

list = [123,'abcd',10.2,'d'] *#can be a array of any data type or single data type.* list1 = ['hello','world']

**print**(list) *#will ouput whole list. [123,'abcd',10.2,'d']*

**print**(list[0:2]) *#will output first two element of list. [123,'abcd']*

**print**(list1 \* 2) *#will gave list1 two times. ['hello','world','hello','world']* **print**(list + list1) *#will gave concatenation of both the lists.*

*[123,'abcd',10.2,'d','hello','world']*

**Section 2.5: Dictionary Data Type**

Dictionary consists of key-value pairs.It is enclosed by curly braces {} and values can be assigned and accessed using square brackets[].

dic={'name':'red','age':10}

**print**(dic) *#will output all the key-value pairs. {'name':'red','age':10}*

**print**(dic['name']) *#will output only value with 'name' key. 'red'*

**print**(dic.values()) *#will output list of values in dic. ['red',10]*

**print**(dic.keys()) *#will output list of keys. ['name','age']*

**Section 2.6: Tuple Data Type**

Lists are enclosed in brackets [ ] and their elements and size can be changed, while tuples are enclosed in parentheses ( ) and cannot be updated. Tuples are immutable.

tuple = (123,'hello')

tuple1 = ('world')

**print**(tuple) *#will output whole tuple. (123,'hello')*

**print**(tuple[0]) *#will output first value. (123)*

**print**(tuple + tuple1) *#will output (123,'hello','world')*

tuple[1]='update' *#this will give you error.*

Python® Notes for Professionals 34

**Chapter 3: Indentation**

**Section 3.1: Simple example**

For Python, Guido van Rossum based the grouping of statements on indentation. The reasons for this are explained in the first section of the "Design and History Python FAQ". Colons, :, are used to declare an indented code block, such as the following example:

**class** ExampleClass:

*#Every function belonging to a class must be indented equally*

**def** \_\_init\_\_(self):

name = "example"

**def** someFunction(self, a):

*#Notice everything belonging to a function must be indented*

**if** a > 5:

**return** True

**else**:

**return** False

*#If a function is not indented to the same level it will not be considers as part of the parent class* **def** separateFunction(b):

**for** i **in** b:

*#Loops are also indented and nested conditions start a new indentation*

**if** i == 1:

**return** True

**return** False

separateFunction([2,3,5,6,1])

**Spaces or Tabs?**

The recommended indentation is 4 spaces but tabs or spaces can be used so long as they are consistent. ***Do not mix tabs and spaces in Python*** as this will cause an error in Python 3 and can causes errors in Python 2.

**Section 3.2: How Indentation is Parsed**

Whitespace is handled by the lexical analyzer before being parsed.

The lexical analyzer uses a stack to store indentation levels. At the beginning, the stack contains just the value 0, which is the leftmost position. Whenever a nested block begins, the new indentation level is pushed on the stack, and an "INDENT" token is inserted into the token stream which is passed to the parser. There can never be more than one "INDENT" token in a row (IndentationError).

When a line is encountered with a smaller indentation level, values are popped from the stack until a value is on top which is equal to the new indentation level (if none is found, a syntax error occurs). For each value popped, a "DEDENT" token is generated. Obviously, there can be multiple "DEDENT" tokens in a row.

The lexical analyzer skips empty lines (those containing only whitespace and possibly comments), and will never generate either "INDENT" or "DEDENT" tokens for them.

At the end of the source code, "DEDENT" tokens are generated for each indentation level left on the stack, until just the 0 is left.

For example:

Python® Notes for Professionals 35

**if** foo:

**if** bar:

x = 42

**else**:

**print** foo

is analyzed as:

**<if> <foo> <:>** [0]

**<INDENT> <if> <bar> <:>** [0, 4]

**<INDENT> <x>** <=**>** <42**>** [0, 4, 8]

**<DEDENT> <DEDENT> <else> <:>** [0]

**<INDENT> <print> <foo>** [0, 2]

**<DEDENT>**

The parser than handles the "INDENT" and "DEDENT" tokens as block delimiters.

**Section 3.3: Indentation Errors**

The spacing should be even and uniform throughout. Improper indentation can cause an IndentationError or cause the program to do something unexpected. The following example raises an IndentationError:

a = 7

**if** a > 5:

**print** "foo"

**else**:

**print** "bar"

**print** "done"

Or if the line following a colon is not indented, an IndentationError will also be raised:

**if** True:

**print** "true"

If you add indentation where it doesn't belong, an IndentationError will be raised:

**if** True:

a = 6

b = 5

If you forget to un-indent functionality could be lost. In this example None is returned instead of the expected False:

**def** isEven(a):

**if** a%2 ==0:

**return** True

*#this next line should be even with the if*

**return** False

**print** isEven(7)

Python® Notes for Professionals 36

**Chapter 4: Comments and Documentation Section 4.1: Single line, inline and multiline comments** Comments are used to explain code when the basic code itself isn't clear.

Python ignores comments, and so will not execute code in there, or raise syntax errors for plain english sentences. Single-line comments begin with the hash character (*#*) and are terminated by the end of line. Single line comment:

*# This is a single line comment in Python*

Inline comment:

**print**("Hello World") *# This line prints "Hello World"*

Comments spanning multiple lines have """ or ''' on either end. This is the same as a multiline string, but they can be used as comments:

"""

This type of comment spans multiple lines.

These are mostly used for documentation of functions, classes and modules. """

**Section 4.2: Programmatically accessing docstrings**

Docstrings are - unlike regular comments - stored as an attribute of the function they document, meaning that you can access them programmatically.

**An example function**

**def** func():

"""This is a function that does nothing at all"""

**return**

The docstring can be accessed using the \_\_doc\_\_ attribute:

**print**(func.\_\_doc\_\_)

This is a function that does nothing at all

help(func)

Help on function func in module \_\_main\_\_:

func()

This is a function that does nothing at all

**Another example function**

Python® Notes for Professionals 37

function.\_\_doc\_\_ is just the actual docstring as a string, while the help function provides general information about a function, including the docstring. Here's a more helpful example:

**def** greet(name, greeting="Hello"):

"""Print a greeting to the user `name`

Optional parameter `greeting` can change what they're greeted with."""

**print**("{} {}".format(greeting, name))

help(greet)

Help on function greet in module \_\_main\_\_:

greet(name, greeting='Hello')

Print a greeting to the user name

Optional parameter greeting can change what they're greeted with.

**Advantages of docstrings over regular comments**

Just putting no docstring or a regular comment in a function makes it a lot less helpful.

**def** greet(name, greeting="Hello"):

*# Print a greeting to the user `name`*

*# Optional parameter `greeting` can change what they're greeted with.*

**print**("{} {}".format(greeting, name))

**print**(greet.\_\_doc\_\_)

None

help(greet)

Help on function greet in module **main**:

greet(name, greeting='Hello')

**Section 4.3: Write documentation using docstrings**

A docstring is a multi-line comment used to document modules, classes, functions and methods. It has to be the first statement of the component it describes.

**def** hello(name):

"""Greet someone.

Print a greeting ("Hello") for the person with the given name.

"""

**print**("Hello "+name)

**class** Greeter:

"""An object used to greet people.

Python® Notes for Professionals 38

It contains multiple greeting functions for several languages

and times of the day.

"""

The value of the docstring can be accessed within the program and is - for example - used by the help command.

**Syntax conventions**

**PEP 257**

PEP 257 defines a syntax standard for docstring comments. It basically allows two types: One-line Docstrings:

According to PEP 257, they should be used with short and simple functions. Everything is placed in one line, e.g:

**def** hello():

"""Say hello to your friends."""

**print**("Hello my friends!")

The docstring shall end with a period, the verb should be in the imperative form.

Multi-line Docstrings:

Multi-line docstring should be used for longer, more complex functions, modules or classes.

**def** hello(name, language="en"):

"""Say hello to a person.

Arguments:

name: the name of the person

language: the language in which the person should be greeted

"""

**print**(greeting[language]+" "+name)

They start with a short summary (equivalent to the content of a one-line docstring) which can be on the same line as the quotation marks or on the next line, give additional detail and list parameters and return values.

Note PEP 257 defines what information should be given within a docstring, it doesn't define in which format it should be given. This was the reason for other parties and documentation parsing tools to specify their own standards for documentation, some of which are listed below and in this question.

**Sphinx**

Sphinx is a tool to generate HTML based documentation for Python projects based on docstrings. Its markup language used is reStructuredText. They define their own standards for documentation, pythonhosted.org hosts a very good description of them. The Sphinx format is for example used by the pyCharm IDE.

A function would be documented like this using the Sphinx/reStructuredText format:

**def** hello(name, language="en"):

"""Say hello to a person.

:param name: the name of the person

:type name: str

:param language: the language in which the person should be greeted

:type language: str

Python® Notes for Professionals 39

:return: a number

:rtype: int

"""

**print**(greeting[language]+" "+name)

**return** 4

**Google Python Style Guide**

Google has published Google Python Style Guide which defines coding conventions for Python, including documentation comments. In comparison to the Sphinx/reST many people say that documentation according to Google's guidelines is better human-readable.

The pythonhosted.org page mentioned above also provides some examples for good documentation according to the Google Style Guide.

Using the Napoleon plugin, Sphinx can also parse documentation in the Google Style Guide-compliant format. A function would be documented like this using the Google Style Guide format:

**def** hello(name, language="en"):

"""Say hello to a person.

Args:

name: the name of the person as string

language: the language code string

Returns:

A number.

"""

**print**(greeting[language]+" "+name)

**return** 4

Python® Notes for Professionals 40

**Chapter 5: Date and Time**

**Section 5.1: Parsing a string into a timezone aware datetime object**

Python 3.2+ has support for %z format when parsing a string into a datetime object.

UTC offset in the form +HHMM or -HHMM (empty string if the object is naive).

Python 3.x Version ≥ 3.2

**import** datetime

dt = datetime.datetime.strptime("2016-04-15T08:27:18-0500", "%Y-%m-%dT%H:%M:%S%z")

For other versions of Python, you can use an external library such as dateutil, which makes parsing a string with timezone into a datetime object is quick.

**import** dateutil.parser

dt = dateutil.parser.parse("2016-04-15T08:27:18-0500")

The dt variable is now a datetime object with the following value:

datetime.datetime(2016, 4, 15, 8, 27, 18, tzinfo=tzoffset(None, -18000))

**Section 5.2: Constructing timezone-aware datetimes**

By default all datetime objects are naive. To make them timezone-aware, you must attach a tzinfo object, which provides the UTC offset and timezone abbreviation as a function of date and time.

**Fixed Offset Time Zones**

For time zones that are a fixed offset from UTC, in Python 3.2+, the datetime module provides the timezone class, a concrete implementation of tzinfo, which takes a timedelta and an (optional) name parameter:

Python 3.x Version ≥ 3.2

**from** datetime **import** datetime, timedelta, timezone

JST = timezone(timedelta(hours=+9))

dt = datetime(2015, 1, 1, 12, 0, 0, tzinfo=JST)

**print**(dt)

*# 2015-01-01 12:00:00+09:00*

**print**(dt.tzname())

*# UTC+09:00*

dt = datetime(2015, 1, 1, 12, 0, 0, tzinfo=timezone(timedelta(hours=9), 'JST')) **print**(dt.tzname)

*# 'JST'*

For Python versions before 3.2, it is necessary to use a third party library, such as dateutil. dateutil provides an equivalent class, tzoffset, which (as of version 2.5.3) takes arguments of the form dateutil.tz.tzoffset(tzname, offset), where offset is specified in seconds:

Python 3.x Version < 3.2

Python® Notes for Professionals 41

Python 2.x Version < 2.7

**from** datetime **import** datetime, timedelta

**from** dateutil **import** tz

JST = tz.tzoffset('JST', 9 \* 3600) *# 3600 seconds per hour*

dt = datetime(2015, 1, 1, 12, 0, tzinfo=JST)

**print**(dt)

*# 2015-01-01 12:00:00+09:00*

**print**(dt.tzname)

*# 'JST'*

**Zones with daylight savings time**

For zones with daylight savings time, python standard libraries do not provide a standard class, so it is necessary to use a third party library. pytz and dateutil are popular libraries providing time zone classes.

In addition to static time zones, dateutil provides time zone classes that use daylight savings time (see the documentation for the tz module). You can use the tz.gettz() method to get a time zone object, which can then be passed directly to the datetime constructor:

**from** datetime **import** datetime

**from** dateutil **import** tz

local = tz.gettz() *# Local time*

PT = tz.gettz('US/Pacific') *# Pacific time*

dt\_l = datetime(2015, 1, 1, 12, tzinfo=local) *# I am in EST*

dt\_pst = datetime(2015, 1, 1, 12, tzinfo=PT)

dt\_pdt = datetime(2015, 7, 1, 12, tzinfo=PT) *# DST is handled automatically*

**print**(dt\_l)

*# 2015-01-01 12:00:00-05:00*

**print**(dt\_pst)

*# 2015-01-01 12:00:00-08:00*

**print**(dt\_pdt)

*# 2015-07-01 12:00:00-07:00*

**CAUTION**: As of version 2.5.3, dateutil does not handle ambiguous datetimes correctly, and will always default to the *later* date. There is no way to construct an object with a dateutil timezone representing, for example 2015-11-01 1:30 EDT-4, since this is *during* a daylight savings time transition.

All edge cases are handled properly when using pytz, but pytz time zones should *not* be directly attached to time zones through the constructor. Instead, a pytz time zone should be attached using the time zone's localize method:

**from** datetime **import** datetime, timedelta

**import** pytz

PT = pytz.timezone('US/Pacific')

dt\_pst = PT.localize(datetime(2015, 1, 1, 12))

dt\_pdt = PT.localize(datetime(2015, 11, 1, 0, 30))

**print**(dt\_pst)

*# 2015-01-01 12:00:00-08:00*

**print**(dt\_pdt)

*# 2015-11-01 00:30:00-07:00*

Be aware that if you perform datetime arithmetic on a pytz-aware time zone, you must either perform the calculations in UTC (if you want absolute elapsed time), or you must call normalize() on the result:

Python® Notes for Professionals 42

dt\_new = dt\_pdt + timedelta(hours=3) *# This should be 2:30 AM PST* **print**(dt\_new)

*# 2015-11-01 03:30:00-07:00*

dt\_corrected = PT.normalize(dt\_new)

**print**(dt\_corrected)

*# 2015-11-01 02:30:00-08:00*

**Section 5.3: Computing time dierences** the timedelta module comes in handy to compute differences between times:

**from** datetime **import** datetime, timedelta

now = datetime.now()

then = datetime(2016, 5, 23) *# datetime.datetime(2016, 05, 23, 0, 0, 0)* Specifying time is optional when creating a new datetime object

delta = now-then

delta is of type timedelta

**print**(delta.days)

*# 60*

**print**(delta.seconds)

*# 40826*

To get n day's after and n day's before date we could use :

**n day's after date:**

**def** get\_n\_days\_after\_date(date\_format="%d %B %Y", add\_days=120):

date\_n\_days\_after = datetime.datetime.now() + timedelta(days=add\_days) **return** date\_n\_days\_after.strftime(date\_format)

**n day's before date:**

**def** get\_n\_days\_before\_date(self, date\_format="%d %B %Y", days\_before=120):

date\_n\_days\_ago = datetime.datetime.now() - timedelta(days=days\_before) **return** date\_n\_days\_ago.strftime(date\_format)

**Section 5.4: Basic datetime objects usage** The datetime module contains three primary types of objects - date, time, and datetime.

**import** datetime

*# Date object*

today = datetime.date.today()

new\_year = datetime.date(2017, 01, 01) *#datetime.date(2017, 1, 1)*

*# Time object*

noon = datetime.time(12, 0, 0) *#datetime.time(12, 0)*

*# Current datetime*

now = datetime.datetime.now()

Python® Notes for Professionals 43

*# Datetime object*

millenium\_turn = datetime.datetime(2000, 1, 1, 0, 0, 0) *#datetime.datetime(2000, 1, 1, 0, 0)*

Arithmetic operations for these objects are only supported within same datatype and performing simple arithmetic with instances of different types will result in a TypeError.

*# subtraction of noon from today*

noon-today

Traceback (most recent call last):

File "<stdin>", line 1, **in** <module>

TypeError: unsupported operand type(s) **for** -: 'datetime.time' **and** 'datetime.date' However, it **is** straightforward to convert between types.

*# Do this instead*

**print**('Time since the millenium at midnight: ',

datetime.datetime(today.year, today.month, today.day) - millenium\_turn)

*# Or this*

**print**('Time since the millenium at noon: ',

datetime.datetime.combine(today, noon) - millenium\_turn)

**Section 5.5: Switching between time zones**

To switch between time zones, you need datetime objects that are timezone-aware.

**from** datetime **import** datetime

**from** dateutil **import** tz

utc = tz.tzutc()

local = tz.tzlocal()

utc\_now = datetime.utcnow()

utc\_now *# Not timezone-aware.*

utc\_now = utc\_now.replace(tzinfo=utc)

utc\_now *# Timezone-aware.*

local\_now = utc\_now.astimezone(local)

local\_now *# Converted to local time.*

**Section 5.6: Simple date arithmetic**

Dates don't exist in isolation. It is common that you will need to find the amount of time between dates or determine what the date will be tomorrow. This can be accomplished using timedelta objects

**import** datetime

today = datetime.date.today()

**print**('Today:', today)

yesterday = today - datetime.timedelta(days=1)

**print**('Yesterday:', yesterday)

tomorrow = today + datetime.timedelta(days=1)

**print**('Tomorrow:', tomorrow)

**print**('Time between tomorrow and yesterday:', tomorrow - yesterday)

Python® Notes for Professionals 44

This will produce results similar to:

Today: 2016-04-15

Yesterday: 2016-04-14

Tomorrow: 2016-04-16

Difference between tomorrow **and** yesterday: 2 days, 0:00:00

**Section 5.7: Converting timestamp to datetime** The datetime module can convert a POSIX timestamp to a ITC datetime object.

The Epoch is January 1st, 1970 midnight.

**import** time

**from** datetime **import** datetime

seconds\_since\_epoch=time.time() *#1469182681.709*

utc\_date=datetime.utcfromtimestamp(seconds\_since\_epoch) *#datetime.datetime(2016, 7, 22, 10, 18, 1, 709000)*

**Section 5.8: Subtracting months from a date accurately** Using the calendar module

**import** calendar

**from** datetime **import** date

**def** monthdelta(date, delta):

m, y = (date.month+delta) % 12, date.year + ((date.month)+delta-1) // 12 **if not** m: m = 12

d = min(date.day, calendar.monthrange(y, m)[1])

**return** date.replace(day=d,month=m, year=y)

next\_month = monthdelta(date.today(), 1) *#datetime.date(2016, 10, 23)*

Using the dateutils module

**import** datetime

**import** dateutil.relativedelta

d = datetime.datetime.strptime("2013-03-31", "%Y-%m-%d")

d2 = d - dateutil.relativedelta.relativedelta(months=1) *#datetime.datetime(2013, 2, 28, 0, 0)*

**Section 5.9: Parsing an arbitrary ISO 8601 timestamp with minimal libraries**

Python has only limited support for parsing ISO 8601 timestamps. For strptime you need to know exactly what format it is in. As a complication the stringification of a datetime is an ISO 8601 timestamp, with space as a separator and 6 digit fraction:

str(datetime.datetime(2016, 7, 22, 9, 25, 59, 555555))

*# '2016-07-22 09:25:59.555555'*

but if the fraction is 0, no fractional part is output

str(datetime.datetime(2016, 7, 22, 9, 25, 59, 0))

Python® Notes for Professionals 45

*# '2016-07-22 09:25:59'*

But these 2 forms need a *different* format for strptime. Furthermore, strptime' does not support at all parsing minute timezones that have a:**in** it, thus2016-07-22 09:25:59+0300can be parsed, but the standard format2016-07-22 09:25:59+03:00` cannot.

There is a single-file library called iso8601 which properly parses ISO 8601 timestamps and only them. It supports fractions and timezones, and the T separator all with a single function:

**import** iso8601

iso8601.parse\_date('2016-07-22 09:25:59')

*# datetime.datetime(2016, 7, 22, 9, 25, 59, tzinfo=<iso8601.Utc>)*

iso8601.parse\_date('2016-07-22 09:25:59+03:00')

*# datetime.datetime(2016, 7, 22, 9, 25, 59, tzinfo=<FixedOffset '+03:00' ...>)* iso8601.parse\_date('2016-07-22 09:25:59Z')

*# datetime.datetime(2016, 7, 22, 9, 25, 59, tzinfo=<iso8601.Utc>)*

iso8601.parse\_date('2016-07-22T09:25:59.000111+03:00')

*# datetime.datetime(2016, 7, 22, 9, 25, 59, 111, tzinfo=<FixedOffset '+03:00' ...>)*

If no timezone is set, iso8601.parse\_date defaults to UTC. The default zone can be changed with default\_zone keyword argument. Notably, if this is None instead of the default, then those timestamps that do not have an explicit timezone are returned as naive datetimes instead:

iso8601.parse\_date('2016-07-22T09:25:59', default\_timezone=None)

*# datetime.datetime(2016, 7, 22, 9, 25, 59)*

iso8601.parse\_date('2016-07-22T09:25:59Z', default\_timezone=None)

*# datetime.datetime(2016, 7, 22, 9, 25, 59, tzinfo=<iso8601.Utc>)*

**Section 5.10: Get an ISO 8601 timestamp**

**Without timezone, with microseconds**

**from** datetime **import** datetime

datetime.now().isoformat()

*# Out: '2016-07-31T23:08:20.886783'*

**With timezone, with microseconds**

**from** datetime **import** datetime

**from** dateutil.tz **import** tzlocal

datetime.now(tzlocal()).isoformat()

*# Out: '2016-07-31T23:09:43.535074-07:00'*

**With timezone, without microseconds**

**from** datetime **import** datetime

**from** dateutil.tz **import** tzlocal

datetime.now(tzlocal()).replace(microsecond=0).isoformat()

*# Out: '2016-07-31T23:10:30-07:00'*

See ISO 8601 for more information about the ISO 8601 format.

**Section 5.11: Parsing a string with a short time zone name into a timezone aware datetime object**

Using the dateutil library as in the previous example on parsing timezone-aware timestamps, it is also possible to

Python® Notes for Professionals 46

parse timestamps with a specified "short" time zone name.

For dates formatted with short time zone names or abbreviations, which are generally ambiguous (e.g. CST, which could be Central Standard Time, China Standard Time, Cuba Standard Time, etc - more can be found here) or not necessarily available in a standard database, it is necessary to specify a mapping between time zone abbreviation and tzinfo object.

**from** dateutil **import** tz

**from** dateutil.parser **import** parse

ET = tz.gettz('US/Eastern')

CT = tz.gettz('US/Central')

MT = tz.gettz('US/Mountain')

PT = tz.gettz('US/Pacific')

us\_tzinfos = {'CST': CT, 'CDT': CT,

'EST': ET, 'EDT': ET,

'MST': MT, 'MDT': MT,

'PST': PT, 'PDT': PT}

dt\_est = parse('2014-01-02 04:00:00 EST', tzinfos=us\_tzinfos)

dt\_pst = parse('2016-03-11 16:00:00 PST', tzinfos=us\_tzinfos)

After running this:

dt\_est

*# datetime.datetime(2014, 1, 2, 4, 0, tzinfo=tzfile('/usr/share/zoneinfo/US/Eastern'))* dt\_pst

*# datetime.datetime(2016, 3, 11, 16, 0, tzinfo=tzfile('/usr/share/zoneinfo/US/Pacific'))* It is worth noting that if using a pytz time zone with this method, it will *not* be properly localized:

**from** dateutil.parser **import** parse

**import** pytz

EST = pytz.timezone('America/New\_York')

dt = parse('2014-02-03 09:17:00 EST', tzinfos={'EST': EST})

This simply attaches the pytz time zone to the datetime:

dt.tzinfo *# Will be in Local Mean Time!*

*# <DstTzInfo 'America/New\_York' LMT-1 day, 19:04:00 STD>*

If using this method, you should probably re-localize the naive portion of the datetime after parsing:

dt\_fixed = dt.tzinfo.localize(dt.replace(tzinfo=None))

dt\_fixed.tzinfo *# Now it's EST.*

*# <DstTzInfo 'America/New\_York' EST-1 day, 19:00:00 STD>)*

**Section 5.12: Fuzzy datetime parsing (extracting datetime out of a text)**

It is possible to extract a date out of a text using the dateutil parser in a "fuzzy" mode, where components of the string not recognized as being part of a date are ignored.

**from** dateutil.parser **import** parse

Python® Notes for Professionals 47

dt = parse("Today is January 1, 2047 at 8:21:00AM", fuzzy=True)

**print**(dt)

dt is now a *datetime object* and you would see datetime.datetime(2047, 1, 1, 8, 21) printed. **Section 5.13: Iterate over dates**

Sometimes you want to iterate over a range of dates from a start date to some end date. You can do it using datetime library and timedelta object:

**import** datetime

*# The size of each step in days*

day\_delta = datetime.timedelta(days=1)

start\_date = datetime.date.today()

end\_date = start\_date + 7\*day\_delta

**for** i **in** range((end\_date - start\_date).days):

**print**(start\_date + i\*day\_delta)

Which produces:

2016-07-21

2016-07-22

2016-07-23

2016-07-24

2016-07-25

2016-07-26

2016-07-27

Python® Notes for Professionals 48

**Chapter 6: Date Formatting Section 6.1: Time between two date-times**

**from** datetime **import** datetime

a = datetime(2016,10,06,0,0,0)

b = datetime(2016,10,01,23,59,59)

a-b

*# datetime.timedelta(4, 1)*

(a-b).days

*# 4*

(a-b).total\_seconds()

*# 518399.0*

**Section 6.2: Outputting datetime object to string** Uses C standard format codes.

**from** datetime **import** datetime

datetime\_for\_string = datetime(2016,10,1,0,0)

datetime\_string\_format = '%b %d %Y, %H:%M:%S'

datetime.strftime(datetime\_for\_string,datetime\_string\_format) *# Oct 01 2016, 00:00:00*

**Section 6.3: Parsing string to datetime object** Uses C standard format codes.

**from** datetime **import** datetime

datetime\_string = 'Oct 1 2016, 00:00:00'

datetime\_string\_format = '%b %d %Y, %H:%M:%S'

datetime.strptime(datetime\_string, datetime\_string\_format)

*# datetime.datetime(2016, 10, 1, 0, 0)*

Python® Notes for Professionals 49

**Chapter 7: Enum**

**Section 7.1: Creating an enum (Python 2.4 through 3.3)**

Enums have been backported from Python 3.4 to Python 2.4 through Python 3.3. You can get this the enum34 backport from PyPI.

pip install enum34

Creation of an enum is identical to how it works in Python 3.4+

**from** enum **import** Enum

**class** Color(Enum):

red = 1

green = 2

blue = 3

**print**(Color.red) *# Color.red*

**print**(Color(1)) *# Color.red*

**print**(Color['red']) *# Color.red*

**Section 7.2: Iteration**

Enums are iterable:

**class** Color(Enum):

red = 1

green = 2

blue = 3

[c **for** c **in** Color] *# [<Color.red: 1>, <Color.green: 2>, <Color.blue: 3>]*

Python® Notes for Professionals 50

**Chapter 8: Set**

**Section 8.1: Operations on sets**

**with other sets**

*# Intersection*

{1, 2, 3, 4, 5}.intersection({3, 4, 5, 6}) *# {3, 4, 5}* {1, 2, 3, 4, 5} & {3, 4, 5, 6} *# {3, 4, 5}*

*# Union*

{1, 2, 3, 4, 5}.union({3, 4, 5, 6}) *# {1, 2, 3, 4, 5, 6}* {1, 2, 3, 4, 5} | {3, 4, 5, 6} *# {1, 2, 3, 4, 5, 6}*

*# Difference*

{1, 2, 3, 4}.difference({2, 3, 5}) *# {1, 4}*

{1, 2, 3, 4} - {2, 3, 5} *# {1, 4}*

*# Symmetric difference with*

{1, 2, 3, 4}.symmetric\_difference({2, 3, 5}) *# {1, 4, 5}* {1, 2, 3, 4} ^ {2, 3, 5} *# {1, 4, 5}*

*# Superset check*

{1, 2}.issuperset({1, 2, 3}) *# False*

{1, 2} >= {1, 2, 3} *# False*

*# Subset check*

{1, 2}.issubset({1, 2, 3}) *# True*

{1, 2} <= {1, 2, 3} *# True*

*# Disjoint check*

{1, 2}.isdisjoint({3, 4}) *# True*

{1, 2}.isdisjoint({1, 4}) *# False*

**with single elements**

*# Existence check*

2 **in** {1,2,3} *# True*

4 **in** {1,2,3} *# False*

4 **not in** {1,2,3} *# True*

*# Add and Remove*

s = {1,2,3}

s.add(4) *# s == {1,2,3,4}*

s.discard(3) *# s == {1,2,4}*

s.discard(5) *# s == {1,2,4}*

s.remove(2) *# s == {1,4}*

s.remove(2) *# KeyError!*

Set operations return new sets, but have the corresponding in-place versions:

**method in-place operation in-place method** union s |= t update

intersection s &= t intersection\_update difference s -= t difference\_update symmetric\_difference s ^= t symmetric\_difference\_update

Python® Notes for Professionals 51

For example:

s = {1, 2}

s.update({3, 4}) *# s == {1, 2, 3, 4}*

**Section 8.2: Get the unique elements of a list**

Let's say you've got a list of restaurants -- maybe you read it from a file. You care about the *unique* restaurants in the list. The best way to get the unique elements from a list is to turn it into a set:

restaurants = ["McDonald's", "Burger King", "McDonald's", "Chicken Chicken"] unique\_restaurants = set(restaurants)

**print**(unique\_restaurants)

*# prints {'Chicken Chicken', "McDonald's", 'Burger King'}*

Note that the set is not in the same order as the original list; that is because sets are *unordered*, just like dicts.

This can easily be transformed back into a List with Python's built in list function, giving another list that is the same list as the original but without duplicates:

list(unique\_restaurants)

*# ['Chicken Chicken', "McDonald's", 'Burger King']*

It's also common to see this as one line:

*# Removes all duplicates and returns another list*

list(set(restaurants))

Now any operations that could be performed on the original list can be done again.

**Section 8.3: Set of Sets**

{{1,2}, {3,4}}

leads to:

TypeError: unhashable type: 'set'

Instead, use frozenset:

{frozenset({1, 2}), frozenset({3, 4})}

**Section 8.4: Set Operations using Methods and Builtins** We define two sets a and b

>>> a = {1, 2, 2, 3, 4}

>>> b = {3, 3, 4, 4, 5}

NOTE: {1} creates a set of one element, but {} creates an empty dict. The correct way to create an empty set is set().

Python® Notes for Professionals 52

**Intersection**

a.intersection(b) returns a new set with elements present in both a and b

>>> a.intersection(b)

{3, 4}

**Union**

a.union(b) returns a new set with elements present in either a and b

>>> a.union(b)

{1, 2, 3, 4, 5}

**Difference**

a.difference(b) returns a new set with elements present in a but not in b

>>> a.difference(b)

{1, 2}

>>> b.difference(a)

{5}

**Symmetric Difference**

a.symmetric\_difference(b) returns a new set with elements present in either a or b but not in both

>>> a.symmetric\_difference(b)

{1, 2, 5}

>>> b.symmetric\_difference(a)

{1, 2, 5}

**NOTE**: a.symmetric\_difference(b) == b.symmetric\_difference(a)

**Subset and superset**

c.issubset(a) tests whether each element of c is in a.

a.issuperset(c) tests whether each element of c is in a.

>>> c = {1, 2}

>>> c.issubset(a)

True

>>> a.issuperset(c)

True

The latter operations have equivalent operators as shown below:

**Method Operator**

a.intersection(b) a & b

a.union(b) a|b

a.difference(b) a - b

a.symmetric\_difference(b) a ^ b

a.issubset(b) a <= b

a.issuperset(b) a >= b

**Disjoint sets**

Sets a and d are disjoint if no element in a is also in d and vice versa.

Python® Notes for Professionals 53

>>> d = {5, 6}

>>> a.isdisjoint(b) *# {2, 3, 4} are in both sets*

False

>>> a.isdisjoint(d)

True

*# This is an equivalent check, but less efficient*

>>> len(a & d) == 0

True

*# This is even less efficient*

>>> a & d == set()

True

**Testing membership**

The builtin **in** keyword searches for occurances

>>> 1 **in** a

True

>>> 6 **in** a

False

**Length**

The builtin len() function returns the number of elements in the set

>>> len(a)

4

>>> len(b)

3

**Section 8.5: Sets versus multisets**

Sets are unordered collections of distinct elements. But sometimes we want to work with unordered collections of elements that are not necessarily distinct and keep track of the elements' multiplicities.

Consider this example:

>>> setA = {'a','b','b','c'}

>>> setA

set(['a', 'c', 'b'])

By saving the strings 'a', 'b', 'b', 'c' into a set data structure we've lost the information on the fact that 'b' occurs twice. Of course saving the elements to a list would retain this information

>>> listA = ['a','b','b','c']

>>> listA

['a', 'b', 'b', 'c']

but a list data structure introduces an extra unneeded ordering that will slow down our computations.

For implementing multisets Python provides the Counter class from the collections module (starting from version 2.7):

Python 2.x Version ≥ 2.7

>>> **from** collections **import** Counter

>>> counterA = Counter(['a','b','b','c'])

Python® Notes for Professionals 54

>>> counterA

Counter({'b': 2, 'a': 1, 'c': 1})

Counter is a dictionary where where elements are stored as dictionary keys and their counts are stored as dictionary values. And as all dictionaries, it is an unordered collection.

Python® Notes for Professionals 55