

# Python<sup>®</sup>

## Notes for Professionals

### Chapter 2: Python Data Types

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

#### Section 2.1: String Data Type

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

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

#### Section 2.2: Set Data Types

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

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

```
basket = {"apple", "orange", "apple", "pear", "orange", "banana"}
print(basket) # duplicates will be removed
s = {"apple", "banana", "pear", "apple"}
s = set("abracadabra") # unique letters in a
print(s) # {'a', 'b', 'r', 'c', 'd'}
s.add("z")
print(s) # {'a', 'b', 'r', 'c', 'd', 'z'}
```

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

```
s = frozenset(["a", "b", "c", "d"])
print(s)
cities = frozenset(["Frankfurt", "Basel", "Freiburg"])
print(cities)
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+1j #complex value
long_num = 1234567L #long value
```

### Chapter 29: Basic Input and Output

#### Section 29.1: Using the print function

Python 3.x version > 3.0

In Python 3, print functionality is in the form of a function:

```
print("This string will be displayed in the output")
# This string will be displayed in the output
print("You can print \n escape characters too.")
# You can print \n escape characters too.
```

Python 2.x version > 2.3

In Python 2, print was originally a statement, as shown below:

```
print "This string will be displayed in the output"
# This string will be displayed in the output
print "You can print \n escape characters too."
# You can print \n escape characters too.
```

Note: using `from __future__ import print_function` in Python 2 will allow users to use the `print()` function the same as Python 3 code. This is only available in Python 2.6 and above.

#### Section 29.2: Input from a File

Input can also be read from files. Files can be opened using the built-in function `open`. Using a `with` (`contextlib` module) syntax (called a Context Manager) makes using `open` and getting a handle for the file super easy:

```
with open('sample.txt', 'r') as f:
    # write code here using f.read()
```

This ensures that when code execution leaves the block the file is automatically closed.

Files can be opened in different modes. In the above example the file is opened as read-only. To open an existing file for reading only use `r`. If you want to read the file as bytes use `rb`. To append data to an existing file use `a`. `w` to create a file or overwrite any existing files of the same name. You can use `+` to open a file for both reading and writing. The first argument of `open()` is the filename, the second is the mode. If mode is left blank, it will default to `r`.

```
# let's create an example file
with open('shoppinglist.txt', 'w') as f:
    f.write('strawberries\napple\n')
    f.write('strawberries\napple\n')
```

```
with open('shoppinglist.txt', 'r') as f:
    # this method makes a list where each line
    # of the file is an element in the list
    lines = f.readlines()
print(lines)
# ['strawberries\napple\n', 'strawberries\napple\n']
```

```
with open('shoppinglist.txt', 'r') as f:
    # Python 3 Notes for Professionals
```

### Chapter 40: String Formatting

#### Section 40.1: Basics of String Formatting

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

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

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

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

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

Named arguments can be also used:

```
print('x value is: {x_val}, y value is: {y_val}'.format(x_val=10, y_val=20))
# Out: 'x value is: 10, y value is: 20'
```

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

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

Dictionary keys can be used as well:

```
my_dict = {'key': 6, 'other_key': 7}
print('My other key is: {0[other_key]}'.format(my_dict)) # '0' is optional
# Out: 'My other key is: 7'
```

Same applies to list and tuple indices:

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

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

700+ pages  
of professional hints and tricks

# Contents

<b>About</b>	1
<b>Chapter 1: Getting started with Python Language</b>	2
<a href="#">Section 1.1: Getting Started</a>	2
<a href="#">Section 1.2: Creating variables and assigning values</a>	6
<a href="#">Section 1.3: Block Indentation</a>	10
<a href="#">Section 1.4: Datatypes</a>	11
<a href="#">Section 1.5: Collection Types</a>	15
<a href="#">Section 1.6: IDLE - Python GUI</a>	19
<a href="#">Section 1.7: User Input</a>	21
<a href="#">Section 1.8: Built in Modules and Functions</a>	21
<a href="#">Section 1.9: Creating a module</a>	25
<a href="#">Section 1.10: Installation of Python 2.7.x and 3.x</a>	26
<a href="#">Section 1.11: String function - str() and repr()</a>	28
<a href="#">Section 1.12: Installing external modules using pip</a>	29
<a href="#">Section 1.13: Help Utility</a>	31
<b>Chapter 2: Python Data Types</b>	33
<a href="#">Section 2.1: String Data Type</a>	33
<a href="#">Section 2.2: Set Data Types</a>	33
<a href="#">Section 2.3: Numbers data type</a>	33
<a href="#">Section 2.4: List Data Type</a>	34
<a href="#">Section 2.5: Dictionary Data Type</a>	34
<a href="#">Section 2.6: Tuple Data Type</a>	34
<b>Chapter 3: Indentation</b>	35
<a href="#">Section 3.1: Simple example</a>	35
<a href="#">Section 3.2: How Indentation is Parsed</a>	35
<a href="#">Section 3.3: Indentation Errors</a>	36
<b>Chapter 4: Comments and Documentation</b>	37
<a href="#">Section 4.1: Single line, inline and multiline comments</a>	37
<a href="#">Section 4.2: Programmatically accessing docstrings</a>	37
<a href="#">Section 4.3: Write documentation using docstrings</a>	38
<b>Chapter 5: Date and Time</b>	41
<a href="#">Section 5.1: Parsing a string into a timezone aware datetime object</a>	41
<a href="#">Section 5.2: Constructing timezone-aware datetimes</a>	41
<a href="#">Section 5.3: Computing time differences</a>	43
<a href="#">Section 5.4: Basic datetime objects usage</a>	43
<a href="#">Section 5.5: Switching between time zones</a>	44
<a href="#">Section 5.6: Simple date arithmetic</a>	44
<a href="#">Section 5.7: Converting timestamp to datetime</a>	45
<a href="#">Section 5.8: Subtracting months from a date accurately</a>	45
<a href="#">Section 5.9: Parsing an arbitrary ISO 8601 timestamp with minimal libraries</a>	45
<a href="#">Section 5.10: Get an ISO 8601 timestamp</a>	46
<a href="#">Section 5.11: Parsing a string with a short time zone name into a timezone aware datetime object</a>	46
<a href="#">Section 5.12: Fuzzy datetime parsing (extracting datetime out of a text)</a>	47
<a href="#">Section 5.13: Iterate over dates</a>	48
<b>Chapter 6: Date Formatting</b>	49
<a href="#">Section 6.1: Time between two date-times</a>	49
<a href="#">Section 6.2: Outputting datetime object to string</a>	49

<a href="#">Section 6.3: Parsing string to datetime object</a>	49
<b>Chapter 7: Enum</b>	50
<a href="#">Section 7.1: Creating an enum (Python 2.4 through 3.3)</a>	50
<a href="#">Section 7.2: Iteration</a>	50
<b>Chapter 8: Set</b>	51
<a href="#">Section 8.1: Operations on sets</a>	51
<a href="#">Section 8.2: Get the unique elements of a list</a>	52
<a href="#">Section 8.3: Set of Sets</a>	52
<a href="#">Section 8.4: Set Operations using Methods and Builtins</a>	52
<a href="#">Section 8.5: Sets versus multisets</a>	54
<b>Chapter 9: Simple Mathematical Operators</b>	56
<a href="#">Section 9.1: Division</a>	56
<a href="#">Section 9.2: Addition</a>	57
<a href="#">Section 9.3: Exponentiation</a>	58
<a href="#">Section 9.4: Trigonometric Functions</a>	59
<a href="#">Section 9.5: Inplace Operations</a>	60
<a href="#">Section 9.6: Subtraction</a>	60
<a href="#">Section 9.7: Multiplication</a>	60
<a href="#">Section 9.8: Logarithms</a>	61
<a href="#">Section 9.9: Modulus</a>	61
<b>Chapter 10: Bitwise Operators</b>	63
<a href="#">Section 10.1: Bitwise NOT</a>	63
<a href="#">Section 10.2: Bitwise XOR (Exclusive OR)</a>	64
<a href="#">Section 10.3: Bitwise AND</a>	65
<a href="#">Section 10.4: Bitwise OR</a>	65
<a href="#">Section 10.5: Bitwise Left Shift</a>	65
<a href="#">Section 10.6: Bitwise Right Shift</a>	66
<a href="#">Section 10.7: Inplace Operations</a>	66
<b>Chapter 11: Boolean Operators</b>	67
<a href="#">Section 11.1: `and` and `or` are not guaranteed to return a boolean</a>	67
<a href="#">Section 11.2: A simple example</a>	67
<a href="#">Section 11.3: Short-circuit evaluation</a>	67
<a href="#">Section 11.4: and</a>	68
<a href="#">Section 11.5: or</a>	68
<a href="#">Section 11.6: not</a>	69
<b>Chapter 12: Operator Precedence</b>	70
<a href="#">Section 12.1: Simple Operator Precedence Examples in python</a>	70
<b>Chapter 13: Variable Scope and Binding</b>	71
<a href="#">Section 13.1: Nonlocal Variables</a>	71
<a href="#">Section 13.2: Global Variables</a>	71
<a href="#">Section 13.3: Local Variables</a>	72
<a href="#">Section 13.4: The del command</a>	73
<a href="#">Section 13.5: Functions skip class scope when looking up names</a>	74
<a href="#">Section 13.6: Local vs Global Scope</a>	75
<a href="#">Section 13.7: Binding Occurrence</a>	77
<b>Chapter 14: Conditionals</b>	78
<a href="#">Section 14.1: Conditional Expression (or "The Ternary Operator")</a>	78
<a href="#">Section 14.2: if, elif, and else</a>	78
<a href="#">Section 14.3: Truth Values</a>	78

<a href="#">Section 14.4: Boolean Logic Expressions</a>	79
<a href="#">Section 14.5: Using the cmp function to get the comparison result of two objects</a>	81
<a href="#">Section 14.6: Else statement</a>	81
<a href="#">Section 14.7: Testing if an object is None and assigning it</a>	82
<a href="#">Section 14.8: If statement</a>	82
<b>Chapter 15: Comparisons</b>	83
<a href="#">Section 15.1: Chain Comparisons</a>	83
<a href="#">Section 15.2: Comparison by `is` vs `==`</a>	84
<a href="#">Section 15.3: Greater than or less than</a>	85
<a href="#">Section 15.4: Not equal to</a>	85
<a href="#">Section 15.5: Equal To</a>	86
<a href="#">Section 15.6: Comparing Objects</a>	86
<b>Chapter 16: Loops</b>	88
<a href="#">Section 16.1: Break and Continue in Loops</a>	88
<a href="#">Section 16.2: For loops</a>	90
<a href="#">Section 16.3: Iterating over lists</a>	90
<a href="#">Section 16.4: Loops with an "else" clause</a>	91
<a href="#">Section 16.5: The Pass Statement</a>	93
<a href="#">Section 16.6: Iterating over dictionaries</a>	94
<a href="#">Section 16.7: The "half loop" do-while</a>	95
<a href="#">Section 16.8: Looping and Unpacking</a>	95
<a href="#">Section 16.9: Iterating different portion of a list with different step size</a>	96
<a href="#">Section 16.10: While Loop</a>	97
<b>Chapter 17: Arrays</b>	99
<a href="#">Section 17.1: Access individual elements through indexes</a>	99
<a href="#">Section 17.2: Basic Introduction to Arrays</a>	99
<a href="#">Section 17.3: Append any value to the array using append() method</a>	100
<a href="#">Section 17.4: Insert value in an array using insert() method</a>	100
<a href="#">Section 17.5: Extend python array using extend() method</a>	100
<a href="#">Section 17.6: Add items from list into array using fromlist() method</a>	101
<a href="#">Section 17.7: Remove any array element using remove() method</a>	101
<a href="#">Section 17.8: Remove last array element using pop() method</a>	101
<a href="#">Section 17.9: Fetch any element through its index using index() method</a>	101
<a href="#">Section 17.10: Reverse a python array using reverse() method</a>	101
<a href="#">Section 17.11: Get array buffer information through buffer_info() method</a>	102
<a href="#">Section 17.12: Check for number of occurrences of an element using count() method</a>	102
<a href="#">Section 17.13: Convert array to string using tostring() method</a>	102
<a href="#">Section 17.14: Convert array to a python list with same elements using tolist() method</a>	102
<a href="#">Section 17.15: Append a string to char array using fromstring() method</a>	102
<b>Chapter 18: Multidimensional arrays</b>	103
<a href="#">Section 18.1: Lists in lists</a>	103
<a href="#">Section 18.2: Lists in lists in lists in..</a>	103
<b>Chapter 19: Dictionary</b>	105
<a href="#">Section 19.1: Introduction to Dictionary</a>	105
<a href="#">Section 19.2: Avoiding KeyError Exceptions</a>	106
<a href="#">Section 19.3: Iterating Over a Dictionary</a>	106
<a href="#">Section 19.4: Dictionary with default values</a>	107
<a href="#">Section 19.5: Merging dictionaries</a>	108
<a href="#">Section 19.6: Accessing keys and values</a>	108
<a href="#">Section 19.7: Accessing values of a dictionary</a>	109

Section 19.8: Creating a dictionary .....	109
Section 19.9: Creating an ordered dictionary .....	110
Section 19.10: Unpacking dictionaries using the ** operator .....	110
Section 19.11: The trailing comma .....	111
Section 19.12: The dict() constructor .....	111
Section 19.13: Dictionaries Example .....	111
Section 19.14: All combinations of dictionary values .....	112
<b>Chapter 20: List</b> .....	<b>113</b>
Section 20.1: List methods and supported operators .....	113
Section 20.2: Accessing list values .....	118
Section 20.3: Checking if list is empty .....	119
Section 20.4: Iterating over a list .....	119
Section 20.5: Checking whether an item is in a list .....	120
Section 20.6: Any and All .....	120
Section 20.7: Reversing list elements .....	121
Section 20.8: Concatenate and Merge lists .....	121
Section 20.9: Length of a list .....	122
Section 20.10: Remove duplicate values in list .....	122
Section 20.11: Comparison of lists .....	123
Section 20.12: Accessing values in nested list .....	123
Section 20.13: Initializing a List to a Fixed Number of Elements .....	124
<b>Chapter 21: List comprehensions</b> .....	<b>126</b>
Section 21.1: List Comprehensions .....	126
Section 21.2: Conditional List Comprehensions .....	128
Section 21.3: Avoid repetitive and expensive operations using conditional clause .....	130
Section 21.4: Dictionary Comprehensions .....	131
Section 21.5: List Comprehensions with Nested Loops .....	132
Section 21.6: Generator Expressions .....	134
Section 21.7: Set Comprehensions .....	136
Section 21.8: Refactoring filter and map to list comprehensions .....	136
Section 21.9: Comprehensions involving tuples .....	137
Section 21.10: Counting Occurrences Using Comprehension .....	138
Section 21.11: Changing Types in a List .....	138
Section 21.12: Nested List Comprehensions .....	138
Section 21.13: Iterate two or more list simultaneously within list comprehension .....	139
<b>Chapter 22: List slicing (selecting parts of lists)</b> .....	<b>140</b>
Section 22.1: Using the third "step" argument .....	140
Section 22.2: Selecting a sublist from a list .....	140
Section 22.3: Reversing a list with slicing .....	140
Section 22.4: Shifting a list using slicing .....	140
<b>Chapter 23: groupby()</b> .....	<b>142</b>
Section 23.1: Example 4 .....	142
Section 23.2: Example 2 .....	142
Section 23.3: Example 3 .....	143
<b>Chapter 24: Linked lists</b> .....	<b>145</b>
Section 24.1: Single linked list example .....	145
<b>Chapter 25: Linked List Node</b> .....	<b>149</b>
Section 25.1: Write a simple Linked List Node in python .....	149
<b>Chapter 26: Filter</b> .....	<b>150</b>

[Click here to download full PDF material](#)