

Python®

Notes for Professionals

Chapter 2: Python Data Types

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

Section 2.1: String Data Type

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

```
#!/usr/bin/python
a_str = 'Hello World'
print(a_str)  # output will be Hello World
print(a_str[0])  # 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)  # output will be {'apple', 'orange', 'banana', 'pear'}
s = set('abracadabra')  # unique letters in a
print(s)
s.add('x')
print(s)
s.remove('a')
print(s)
```

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

```
f = frozenset('aefagea')
print(f)
f = frozenset({'f', 'g', 'd', 'k', 't'})
cities = frozenset(['Frankfurt', 'Basel', 'Freiburg'])
print(cities)
f = 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
```

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 in escape characters too.")
# You can print 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 in escape characters too."
# You can print 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` statement (a `with` statement is called a `Context Manager`) makes using `open` and getting a handle for the file super easy:

```
with open('somefile.txt', 'r') as fileobj:
    # write code here using fileobj
```

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 that file as bytes use `rb`. To append data to an existing file use `a`. To create a file or overwrite any existing file of the same name. You can use `x` 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 fileobj:
    fileobj.write('tomatopotatogarlic')

with open('shoppinglist.txt', 'r') as fileobj:
    # this method makes a list where each line
    # of the file is an element in the list
    lines = fileobj.readlines()

print(lines)
# ['tomatopotatogarlic']
```

```
with open('shoppinglist.txt', 'r') as fileobj:
```

Python® Notes for Professionals

Chapter 40: String Formatting

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

Section 40.1: Basics of String Formatting

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

```
print('{}, {} and {}'.format(foo, bar, baz))
# Out: '1, bar and 2.14'
```

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

```
print('{0}, {1}, {2}, and {3}'.format(foo, bar, baz))
# Out: '1, bar, 2.14, and baz'
print('{0}, {1}, {2}, and {3}'.format(foo, bar, baz))
# Out: 'Index out of range error'
```

Named arguments can be also used:

```
print('X value is: {x.val}, Y value is: {y.val}, Z value is: {z.val}'
# Out: 'X value is: 2, Y value is: 2.14, Z value is: 2.14'
```

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

```
class Assignment(object):
    def __init__(self, value):
        self.value = value
my_value = Assignment(5)
print('My value is: {0.value}'.format(my_value))
# 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))
# 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))
# 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 231](#))-for formatting strings. `str.format` is a successor of `%`.

Python® Notes for Professionals

700+ pages
of professional hints and tricks

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	15
Section 1.6: IDLE - Python GUI	19
Section 1.7: User Input	21
Section 1.8: Built in Modules and Functions	21
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()	28
Section 1.12: Installing external modules using pip	29
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 differences	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