Contents

Part I

Introduction to Python3

Chapter 1

Getting started with Python3

1.1 Python3 Interpretor

If you are new to Python then we will suggest you should start with any Linux OS, it will give you more scope to design the program and other tools. In this tutorial I'm using Fedora Linux to teach you how to start with `Python3`. Basically Python language is developed in C language. Python is one of the easy language to learn and you can start with it easily. Let's get started with Python3 interpretor. You need to type `python3` in to your interpreter and you are here.

```
$ python3
Python 3.7.0 (default, Sep 10 2018, 16:52:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

1.1.1 Operators

As I mentioned, Python is interpreted language, you will give input to it and it will give you output instantly.

You can start with additions of numbers.

```
>>> 10 + 20
30
>>> 30 - 10
20
>>> 10 * 2
```

```
20
>>> 10 / 2
5.0
>>> 10 // 2
5
>>> 10 % 2
0
>>>
```

In above example you could see that as soon as the input is given, Python interpreter is giving output.

In Python // operator have special meaning. It will return you the absolute result, while / operator will return result in the float.

1.1.2 Relational Operators

In python you can directly compare two numbers, Python interpreter provide functionality to compare two numbers which most of the language do not provide.

For more clarification see the following examples:

```
>>> 10 == 10
True
>>> 10 > 10
False
>>> 10 < 10
False
>>> 10 <= 10
True
>>> 10 >= 10
True
>>> 10 != 10
False
```

1.1.3 Bitwise Operators

Bitwise operators are used to perform bit operations on the numbers. Those numbers are treated as string of bits written in twos complement binary.

- 0 is written as "0"

- 1 is written as "1"
- 2 is written as "10"
- 3 is "11"
- .
- .
- 1029 is "10000000101" == 2**10 + 2**2 + 2**0 == 1024 + 4 + 1

Operators

- $x \ll y$: x operator is shifted to left by y bits and it return result of x with shifted y bits. It is same as $x * 2 * *y$

```
>>> 3<<2
12
>>> 3*2**2
12
```

- $x \gg y$: x operator is shifted to right by y bits and it return result of x with shifted y bits. It is same as $x//2 * *y$.

```
>>> 8>>2
2
>>> 8//2**2
2
```

- $x\&y$: Bitwise AND, each bit of output is 1 if corresponding bit of x AND y is 1, otherwise it is 0

```
>>> 8 & 8
8
>>> 8 & 2
0
```

- $x|y$: Bitwise OR, each bit of output is 1 if corresponding bit of x AND y is 1, otherwise it is 0

```
>>> 2 | 2
2
>>> 2 | 4
6
>>>
```

- $\sim x$: Complement, It will gives complement of x, the number you get by switching each 1 for a 0 and each 0 for a 1. This is the same as -x - 1.

```
>>> ~2
-3
```

- $x \wedge y$: Does a "bitwise exclusive or". Each bit of the output is the same as the corresponding bit in x if that bit in y is 0, and it's the complement of the bit in x if that bit in y is 1.

```
>>> 1^1
0
>>> 1^2
3
```

1.1.4 Keywords

In any programming language keywords are present, those are nothing but reserve words. You can not use those words for any variable assignment or for function deceleration.

List of keywords present in the Python

class	False	None	True
and	as	assert	break
continue	def	del	elif
else	expect	finally	for
from	global	if	import
in	is	lambda	not
or	pass	raise	return
try	while	with	yield

1.2 Variables and assignment

Python is dynamically typed language, it means it will decide what to do with your input at the run time.

Variable declaration and definition in python is something different. If you want to define integer in other languages, for that you follow this syntax `<data_type> <variable_name> = <value>`

In python you can directly assign value to the variable without specifying it's data type. Once you assign value to it Python interpreter will automatically parse it. It's general syntax is: <variable1>...<variablen> = <value1>...<valuen>

```
>>> a = 10
>>> type(a)
<class 'int'>
```

In above example you assigned `a = 10` and `type()` function shows it belongs to the `class int`. Now let's assign `a = ''Hello''` and check it's type.

```
>>> a = "Hello"
>>> type(a)
<class 'str'>
```

In above example you could see that you can assign different value to same variable and interpreter accepted it.

1.2.1 Indentation

In python body of the statement is not defined by the curly brackets, you have to define body using indentations i.e. spaces.

If you are writing any statement, consider, if statement once you write if statement then you need to specify colon (:), then on the next line specify the spaces to define it's body.

```
if ch == 1:
    print("You selected 1st choice")
```

For spaces you can either type 4 spaces or you can use tab.
NOTE: Mixture of both space and tabs will throw an exception.

1.3 Conditional statements

1.3.1 If statement

If statement is conditional statement. It is used to check the condition between two variables.

Any non zero value is considered as a True value. In python it's syntax is little bit different.

True keyword is used to represent boolean value. In first statement condition is true so it will print the if block. In second statement $10 > 10$ condition is false, it will not print "True statement".

```
>>> if True:
...     print("True statement")
...
True statement
>>> if 10>10:
...     print("True statement")
...
>>>
```

1.3.2 if..else statement

Like I mentioned in the above statement if is the conditional statement, in if statement if the condition is true, then it will execute the if block or it will skip it.

Here if condition is false then, else block got executed.

```
>>> if False:
...     print("If block is printed")
... else:
...     print("Else block is printed")
...
Else block is printed
>>>
```

1.3.3 if..elif statement

In the python switch statement is not present. If you want to check any choice which is matches to the multiple statement then you need to use elif statement.

elif statement is check the condition, if the condition is true then it will execute the block or else block.

You can write as many elif statement as you want. But remember if one elif statement is executed it will not execute the remaining statement.

```
>>> no = 10
>>> if no > 100:
...     print("no is grater")
... elif no < 10:
...     print("No is < 10")
... elif no == 10:
...     print("No is equal to 10")
... elif no == 10:
...     print("No is 10")
... else:
...     print("Unknown value of no")
...
No is equal to 10
```

In above statement you can see that once the condition is true it skip the all the next elif statements.

1.4 Loops

In Python there are two main loops:

1. For loop
2. While loop

1.4.1 For loop

For loop is basic loop. It is designed in such a way that it can iterate on the multiple objects.

Consider the following example:

```
>>> for i in range(10):
...     print(i)
...
```

```
0
1
2
3
4
5
6
7
8
9
>>>
```

In above example range is the inbuilt function which will return the list of the integer numbers from 0 to 9.

For loop by default pointing to zero location and printing the value. In for loop values are auto increment.

1.4.2 While loop

While loop is used where for loop have limitations. One thing about the for loop it is basically used for iterating the objects. While loop is used to perform some mathematical operations or to perform some operations on user defined inputs etc.

```
>>> i = 0
>>> while i < 10:
...     print(i)
...     i = i + 1
...
0
1
2
3
4
5
6
7
8
9
>>>
```

1.4.3 Continue statement

Continue statement is used to transfer the program execution back to the loop.

```
>>> i = 0
>>> while i < 10:
...     i = i + 1
...     if i == 5:
...             continue
...     print(i)
...
1
2
3
4
6
7
8
9
10
>>>
```

In above example the control statement is forwarded to while loop again when condition is true. In output you can observe that it is not printed 5.

1.4.4 Break statement

Break statement is used to exit the loop. It's uses is same as continue statement.

```
>>> i = 0
>>> while i < 10:
...     i = i + 1
...     if i == 5:
...             break
...     print(i)
...
1
2
3
4
>>>
```

1.5 Strings

In Python strings are defined using either ‘ quotes or “ quotes, but all of them treated as equally. When write a string it is a object of class `str`.

```
>>> ‘hello’ == "hello"
True
>>> type(’hello’)
<class ’str’>
```

If you specify hello without quotes then python interpreter will treat it as variable. If interpreter will find any variable then it will work otherwise it will throw an error.

```
>>> hello
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name ’hello’ is not defined
```

String Methods

You can check properties of string using `dir()` method.

```
>>> dir(’hello’)
[’__add__’, ’__class__’, ’__contains__’, ’__delattr__’,
’__dir__’, ’__doc__’, ’__eq__’, ’__format__’, ’__ge__’,
’__getattribute__’, ’__getitem__’, ’__getnewargs__’,
’__gt__’, ’__hash__’, ’__init__’, ’__init_subclass__’,
’__iter__’, ’__le__’, ’__len__’, ’__lt__’, ’__mod__’,
’__mul__’, ’__ne__’, ’__new__’, ’__reduce__’,
’__reduce_ex__’, ’__repr__’, ’__rmod__’, ’__rmul__’,
’__setattr__’, ’__sizeof__’, ’__str__’, ’__subclasshook__’,
’capitalize’, ’casefold’, ’center’, ’count’, ’encode’,
’endswith’, ’expandtabs’, ’find’, ’format’, ’format_map’,
’index’, ’isalnum’, ’isalpha’, ’isdecimal’, ’isdigit’,
’isidentifier’, ’islower’, ’isnumeric’, ’isprintable’,
’isspace’, ’istitle’, ’isupper’, ’join’, ’ljust’, ’lower’,
’lstrip’, ’maketrans’, ’partition’, ’replace’, ’rfind’,
’rindex’, ’rjust’, ’rpartition’, ’rsplit’, ’rstrip’, ’split’,
’splitlines’, ’startswith’, ’strip’, ’swapcase’, ’title’,
’translate’, ’upper’, ‘zfill’]
```

You can use the string properties using . symbol.

```
>>> s = ‘Hello’
>>> s.upper()
HELLO
```

1.5.1 Operation on strings

String Concatenation

You can concat two string using + operator.

```
>>> h = "Hello "
>>> i = "World"
>>> h + i
’Hello World’
```

String Multiplication

You can not subtract or divide string in Python. But you can multiply the strings with Python. When you multiply with the number it will create that strings references.

```
>>> "Hello" * 5
‘HelloHelloHelloHelloHello’
```

If you can multiply to the string with the numbers then it is easy to print star pattern in Python.

```
>>> for i in range(1, 6):
...     print(‘*’ * i)
...
*
**
***
****
*****
```

Substring in String

To check substring is present in the string is too much easy in the Python. You can use `in` keyword to check the substring is present or not.

```
>>> s = "Incredible India!!"
>>> "India" in s
True
>>> "!!" in s
True
>>> "#" in s
False
>>>
```

1.5.2 Iteration on string

You can iterate on the string using for loop. For loop automatically stops when string ended, you do not need to provide length of the string.

```
>>> for i in 'hello':
...     print(i)
...
h
e
l
l
o
```

1.5.3 String Slicing

It will be more useful if strings are able to access using Python's array like syntax. Here in Python indexing is always starts with zero.
In python you can use -1, -2.. and so on to access the characters in the string, it will start accessing it from end of the string.

```
>>> s = "Hello"
>>> s[0]
'H'
>>> s[-1]
'o'
```

Python will give you more easy syntax to access the string. In python you can access the strings with the following syntax.

```
[start:stop:jump]
```

Start is from where you want to start accessing the string.
Stop is to where you want to stop accessing the string.
Jump is for jumping on the string.
Let's see few examples:

```
>>> s[2:]   # It will print all the string from 2.
'llo'
>>> s[2:3]  # It will print string between index 2 to 3.
'l'
>>> s[:3]   # It will print the string till 3.
'Hel'
>>> s[::2]  # It will jump on the string.
'Hlo'
>>> s[-1:]  # It will print the string starting with o char.
'o'
>>> s[:-1]  # It will print the string expect last char.
'Hell'
>>> s[::-1] # It will reverse the string
'olleH'
```

1.6 Lists

Lists are the basic data structure in the Python. Lists are nothing but the array but they do not have fixed size, and they are heterogeneous. Lists can accept different type of elements.

You can define the lists using following ways:

```
>>> l = []
>>> l1 = list()
```

Above statements will create the empty lists. Here `list` is the inbuilt method used for creating the list instance.

Again using `dir()` method you can check the properties of the lists. If you want to add the element in to the empty list then you can use append method to add element in to list.

```
>>> l.append(10)
>>> l
[10]
```

1.6.1 List methods

- append(): To append element to the list.
- clear(): To clear the list.
- copy(): To copy the list in to the another variable.
- count(): Count the element occurrences in the list.
- extend(): To extend the existing list with another.
- index(): Index of the element.
- insert(): To insert the element at specific location.
- pop(): To pop the element from the list.
- remove(): To remove the element from the list.
- reverse(): To reverse the string.
- sort(): To sort the elements in the string.

1.6.2 Operation On Lists

On lists you can perform different operations. You can add two lists. Ex:

```
>>> l = [1, 2, 3]
>>> l1 = [4, 5, 6]
>>> l + l1
[1, 2, 3, 4, 5, 6]
```

You can multiply to list by number. It will create it's references but not the actual list.

```
>>> [1] * 3
[1, 1, 1]
```

1.6.3 Iteration on Lists

You can iterate on the list with for loop and using while loop.

```
>>> l = [1, 2, 3]
>>> for i in l:
...     print(i)
...
1
2
3
>>> i = 0
>>> while i < len(l):
...     print(l[i])
...     i += 1
...
1
2
3
>>>
```

1.6.4 List Slicing

List slicing is same as string slicing. See the following few examples.

```
>>> l
[1, 2, 3]
>>> l[0]
1
>>> l[:1]
[1]
>>> l[:2]
[1, 2]
>>> l[0:2]
[1, 2]
>>> l[0::2]
[1]
>>> l[::-1]
[3, 2, 1]
>>>
```

1.7 Tuples

Tuples are same like lists. It is represent using round bracket '(' and ')'.
Tuples are immutable objects, those can not be modified. You can not update the value in the Tuple. You can not add or remove the elements in the Tuple.

```
>>> t = tuple()
>>> t
()
>>> t = (10, 20, 30)
>>> t
(10, 20, 30)
>>>
```

Tuple methods

You can check the properties of the Tuples using dir method. You will find only two properties.

- count(): It will give you the count of the elements in the tuple.
- index(): It will give you the index of the element.

Operations on Tuples

As we saw in the lists, you can add two Tuples. Ex:

```
>>> t
(10, 20, 30)
>>> t + t
(10, 20, 30, 10, 20, 30)
>>>
```

You can multiply the Tuple by number. It will create it's references, but not the actual list.

```
>>> t * 3
(10, 20, 30, 10, 20, 30, 10, 20, 30)
```

Iteration on Tuples

You can iterate on the list using for loop and while loop.

```
>>> for i in t:
...     print(i)
...
10
20
30
>>> i = 0
>>> while i < len(t):
...     print(t[i])
...     i += 1
...
10
20
30
>>>
```

Tuple Slicing

You can perform slicing operations on the lists. See following few examples:

```
>>> t[0]
10
>>> t[:2]
(10, 20)
>>> t[2:5]
(30, 40, 50)
>>> t[4:]
(50, 60, 70)
>>> t[4::-1]
(50, 40, 30, 20, 10)
>>> t[::2]
(10, 30, 50, 70)
>>>
```

1.8 Dictionaries

Dictionaries are based on the hash tables data structure in C. Dictionaries are defined using key and value. You can use curly braces to define the dictionaries.

Dictionaries are based on key and values. For each value there is one key associated with it.

```
>>> d = {}
>>> d = {'country': 'India',
...      'capital': 'Delhi',
...      'states': 29}
>>> d
{'country': 'India', 'capital': 'Delhi', 'states': 29}
```

You can access the values using associated keys, and you can replace it using simple way.

```
>>> d['states']
29
>>> d['states'] = 30
>>> d
{'country': 'India', 'capital': 'Delhi', 'states': 30}
>>> d['Languages'] = ['Hindi', 'English', 'Tamil', 'Malyalum',
 'Marathi', 'Gujarati']
```

Keys in the dictionary are anything those can be string, integer, floating point values or any module object.

```
>>> import os
>>> d = {}
>>> d[os] = 'os'
>>> d[1] = 'One'
>>> d['two'] = 2
>>> d[3.3] = 'three.three'
>>> d
{<module 'os' from '/usr/lib64/python3.6/os.py'>: 'os', 1: 'One', 'two': 2, 3.3: '
```

1.8.1 Dictionary Methods

Dictionary have following methods:

- clear() : To clear the dictionary.
- copy() : To copy the dictionary to the another variable.
- get() : Get is used for getting the value of the specific key in the dictionary or it will return `None`.
- items() : This method will return the all the items in the dictionary inside the list. This list will consist the pair of the key and value tuple.
- keys() : This method will return all the keys inside the dict.
- pop() : This method will pop specified key and return the value.
- popitem(): This method will pop any random key and value from they dictionary, it will return the tuple of key and value.
- update(): This method will update the dictionary with new values or old keys with new values.
- values() : This method will return the list of the all values.
- fromkeys() : This method will return the new dictionary with keys from iterable and values equal to values.

1.8.2 Iteration on Dictionary

You can iterate on the dictionary using the for loop. By default it will iterate on the keys.

```
>>> d = {'one' : 1, 'two': 2, 'three': 3}
>>> for i in d:
...     print(i)
...
one
two
three
>>>
```

In the dictionary you can iterate only values. For that you need to use `values()` method.

```
>>> for i in d.values():
...     print(i)
...
1
2
3
>>>
```

Using the beauty of the for loop you can iterate on the key and value at the same time. For that you can use `items()` method.

```
>>> for i, j in d.items():
...     print(i, j)
...
one 1
two 2
three 3
>>>
```

1.9 Functions

Function is a block of statements which will perform single action. Function will provide modular code, which will be more easy to read. You can reuse this code to perform some action which will save your time.

In Python you can define the function using the `def` keyword. After the def keyword you define the name of the function and later in the parenthesis you provide the arguments, after the arguments line ends with the : which indicate that from the next line body of the function will be begin.

```
>>> def add():
...     print("Addition : ", 10 + 20)
...
>>> add()
Addition : 30
```

Above code will show the simple example of the function. This add function will print the addition of the two numbers. You can call any function using `function_name(args)` syntax, for Ex. `add()` in above code.

In Python you can return the values from the function using `return` keyword. Below code shows some example of the function with with return values.

```
>>> def add():
...     return 10+20
...
>>> add()
30
>>> t = add()
>>> t
30
>>> def add():
...     return 10+20
...
>>> def sub():
...     print(20 - 10)
...
>>>
>>> u = sub()
10
>>> print(u)
None
>>>
```

In above example add function will return the value, which is addition of the two numbers. You can store that value in a variable for later use.

Let's see other side, what if function do not return any value? In that case it will return `None` keyword. You can see the example in above code with function sub. Function will print the subtraction but it will not return anything. By default the value is `None`.

1.9.1 Functions with arguments

```
>>> def add(a, b):
...     return a + b
...
>>> add()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: add() missing 2 required positional arguments: 'a' and 'b'
```

```
>>> add(109, 23)
132
>>> add(109)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: add() missing 1 required positional argument: 'b'
>>> add("abc", "def")
'abcdef'
>>> add([1], [2])
[1, 2]
>>>
```

1.10 Decorators

1.11 Exception Handling

```
try:
    int("abc")
except:
    print("Error occured")

print("Hello")


while True:
    val = input("Enter Value: ")
    try:
        int(val)
        break
    except:
        print("Invalid value inserted.")
        continue

print("Hello")
$ python3 /tmp/exception.py
Enter Value: abc
Invalid value inserted.
Enter Value: 123sad
```

```
Invalid value inserted.
Enter Value: 123
Hello

d = {1:'abc'}

while True:
    val = input("Enter Value: ")
    try:
        int(val)
        d[int(val)]
        break
    except ValueError:
        print("Invalid value inserted.")
    except KeyError:
        print("Key is not available in dict")
    except Exception as e:
        print(e)
        print("Error occured")

print("Hello")
```

1.12 Class

1.13 File Operations

1.14 Modules

1.14.1 sys module

1.14.2 os module

1.14.3 re module

1.14.4 sqlite module

1.15 Python Built In methods

Part II

Testing with Python

Part III

Django