Python Tutorial & Cheat Sheet

Getting Started

What you need:

- Linux based machine with Python 2.7 or higher (comes default on most linux systems)

  (Python on windows also works though not ideal)
- Editor on Linux such as VI or EMACS to edit your files
- Python interpreter: operates similar to a unix shell: reads and executes commands interactively
- Setting your path variable to include the location of the Python interpreter: (usually /usr/local/bin/python

- Run python scripts by using the following syntax: $python filename.py

Python Basics

1. Primitives

Numbers

Python has integers and floats. Integers are whole numbers, like 314, 500. Floats, meanwhile, are fractional numbers like 3.14, 2.867, 76.88887
You can use the type method to check the value of an object.

```
>>> type(3)
<type 'int'>
>>> type(3.14)
<type 'float'>
>>> pi = 3.14
>>> type(pi)
<type 'float'>
```

Strings

Strings are. used quite often in Python. Strings, are just that, a string of characters. A character is anything you can type on the keyboard in one keystroke, like a letter, a number, or a backslash. Python recognizes single and double quotes as the same thing, the beginning and ends of the strings.

```
>>> “string list”
‘string list’
>>> ‘string list’
```

you can also join strings with use of variables as well.

```
>>> a = “first”
>>> b = “last”
>>> a + b
‘firstlast’
```

Methods:

There are also different string methods for you to choose from as well – like **upper** , **lower** , **replace** ,

and **count.**

Upper:

```
>>> w=‘hello!‘
>>> w.upper()
  ‘HELLO!‘
```

Lower:

```
 >>> w=‘HELLO!‘
>>> w.upper()
‘hello!‘
```

Replace:

```
>>>r='rule‘
>>>r.replace('r','m')
'mule‘
```

Count:

```
>>>numbers=['1','2','1','2','2']
>>>numbers.count('2')
3
```

Booleans

Boolean values are simply True or False .

Value is equal to another value with two equal signs:

```
>>> 10 == 10
True

>>> 10 == 11
False

>>> "jack" == "jack“
True

>>> "jack" == "jake“
False
```

To check for inequality use:

```
>>> 10 != 10
False

>>> 10 != 11
True

>>> "jack" != "jack"
False

>>> "jack" != "jake"
True
```

2. Collections

Lists:

Lists are containers for holding values.

```
>>> fruits = ['apple','lemon','orange','grape']
>>> fruits
['apple', 'lemon', 'orange', 'grape']
```

To access the elements in the list you can use the placement in the list as an indicator. This means numbering the items aligned with their placement in the list. The list starts with 0

```
‘o>>> fruits[2]
range’
```

Len, Append and Pop

Use len to find number of elements in the list. Use append to add a new object to the end of the list and pop to remove objects

from the end.

```
>>> fruits.append('blueberry')
>>> fruits
['apple', 'lemon', 'orange', 'grape', 'blueberry']
>>> fruits.append('tomato')
>>> fruits
['apple', 'lemon', 'orange', 'grape', 'blueberry', 'tomato']
>>> fruits.pop()
'tomato'
>>> fruits
['apple', 'lemon', 'orange', 'grape', 'blueberry']
```

Dictionaries

A dictionary optimizes element lookups. It uses keys and values, instead of numbers as placeholders. Each key must have a value. You can used a word to look up a value.

```
>>> words={'apple':'red','lemon':'yellow'}
>>> words
{'lemon': 'yellow', 'apple': 'red'}
>>> words['apple']
'red'
>>> words['lemon']
'yellow‘
```

Output all the keys as a list with keys() and all the values with values()

```
>>> words.keys()
['lemon', 'apple']
>>> words.values()
['yellow', 'red']
```

3. Control Statements

IF Statements

The IF statement is used to check if a condition is true. Essentially, if the condition is true, the Python interpreter runs a block of statements called the if-block. If the statement is false, the interpreter skips the if block and processes another block of statements called the else-block. The else clause is optional.

```
>>> num = 20
>>> if num == 20:
... print 'the number is 20'
... else:
... print 'the number is not 20'
...
the number is 20
>>> num = 21
>>> if num == 20:
... print 'the number is 20'
... else:
... print 'the number is not 20'
the number is not 20
```

Loops

There are 2 kinds of loops used in Python. The For loop and the While loop. For loops are traditionally used when you have a piece of code which you want to repeat n number of times. They are also commonly used to loop or iterate over lists. While loops, like the For Loop, are used for repeating sections of code - but unlike a for loop, the while loop will not run n times, but until a defined condition is met.

For Loop:

```
>>> colors =('red','blue','green')
>>> colors
('red', 'blue', 'green')
>>> for favorite in colors:

... print "I love " + favorite


I love red


I love blue


I love green
```

While loop:

```
>>> num = 1
>>> num

1


>>> while num <=5:


print num


num += 1
12345
```

4. Functions

Functions are blocks of reusable code that perform a single task. You use *def* to define (or create) a new function then you call a function by adding parameters to the function name.

```
>>> def multiply(num1, num2):
... return num1 * num2
...
>>> multiply(2,2)
4
```

You can also set default values for parameters.

```
>>> def multiply(num1, num2=10):
... return num1 * num2
...
>>> multiply(2)
20
```

5. File Handling

File Input

Open a file to read from it:

```
fin = open("foo.txt")
for line in fin:
# manipulate line
fin.close()
```

File Output

Open a file using 'w' to write to a file:

```
fout = open("bar.txt", "w")
fout.write("hello world")
fout.close()
```

Python Cheat Sheet

Common Built in Functions

Function	Returns
abs(x)	Absolute value of x
dict()	Empty dictionary, eg: d = dict()
float(x)	int or string x as float
id(obj)	memory addr of obj
int (x)	float or string x as int
len(s)	Number of items in sequence s
list()	Empty list, eg: m = list()
max(s)	Maximum value of items in s
min(s)	Minimum value of items in s
open(f)	Open filename f for input
ord(c)	ASCII code of c
pow(x,y)	x ** y
range(x)	A list of x ints 0 to x -- 1
round(x,n)	float x rounded to n places
str(obj)	str representation of obj
sum(s)	Sum of numeric sequence s
tuple(items)	tuple of items
type(obj)	Data type of obj

Common Syntax Structures

Assignment Statement

```
var = exp
```

Console Input/Output

```
 var = input( [prompt] )
 var = raw_input( [prompt] )
 print exp[,] …
```

Selection

```
 if (boolean_exp):
    stmt ...]
 [elif (boolean_exp):
    stmt ...]…
 [else:
    stmt ...]
```

Repetition

```
 while (boolean_exp):
  stmt ...]
```

Traversal

```
 for var in traversable_object:
  stmt ...]
```

Function Definition

```
  def function_name( parmameters ):
  stmt ...]
```

Function Call

```
  function_name( arguments )
```

Class Definition

```
 class Class_name [ (super_class) ]:
  [ class variables ]
  def method_name( self, parameters ):
   stmt
```

Object Instantiation

```
 obj_ref = Class_name( arguments )
```

Method Invocation

```
 obj_ref.method_name( arguments )
```

Exception Handling

```
try:
 stmt ...]
except [exception_type] [, var]:
 stmt ...]
```

Common File Methods

F.method()	**Result/Returns**
read([n])	Return str of next n chars from F, or up to EOF if n not given
readline([n])	Return str up to next newline, or at most n chars if specified
readlines()	Return list of all lines in F, where each item is a line
write(s)	Write str s to F
writelines(L)	Write all str in seq L to F
close()	Closes the file

Module Import

import *module_name*

from *module_name* import *name* , ….

from *module_name* import *