

Having a comprehensive cheat sheet at hand when starting your programming adventure or even while coding as an experienced developer makes you more confident in your skills and saves time.

# python 3 beginner's cheat sheet

a great tool for offline learning

Mihai Cătălin Teodosiu @ pythontutorial.io

---

## Python 3 Beginner's Cheat Sheet

Mihai Cătălin Teodosiu

Visit my website at [www.PythonTutorial.io](http://www.PythonTutorial.io)

First Release: April 2019

---

*About this cheat sheet and its author*

---

Mihai Cătălin Teodosiu is a Network Engineer (CCNP), QA Specialist (ISTQB) and Python Developer who decided to share his knowledge and skills with anyone looking to learn Python programming from scratch, in an easy-to-understand, learn-by-doing fashion, without the fancy wording and endless rambling and gibberish that most authors tend to include in their books and training courses.

Mihai's beginner-friendly teaching methods turned out to be very efficient for tens of thousands of students enrolled in his Python 2.x and 3.x video courses, published on various e-learning platforms. As a matter of fact, this book is based on Mihai's most comprehensive, **bestselling online course for beginners**, called **Python 3 Complete Masterclass**.

From California to Fiji and from Norway to South Africa, Mihai helped programming rookies become proficient in Python, upgrade their skills and nail job interviews. Now, he's grateful for having the chance to help you, as well.

## Table of Contents

Python 3 - Basics .....	4
Python 3 - Strings .....	4
Python 3 - Numbers and Booleans .....	8
Python 3 - Lists .....	10
Python 3 - Sets and Frozensets .....	12
Python 3 - Tuples .....	14
Python 3 - Ranges .....	15
Python 3 - Dictionaries. Conversions between data types .....	15
Python 3 - Conditionals .....	17
Python 3 - For and While Loops .....	17
Python 3 - If / For / While Nesting .....	18
Python 3 - Break / Continue / Pass .....	19
Python 3 - Try / Except / Else / Finally .....	20
Python 3 - Functions .....	21
Python 3 - Modules .....	21
Python 3 - File Operations .....	22
Python 3 - Regular Expressions .....	24
Python 3 - Basics of OOP. Classes and Objects .....	26
Python 3 - List comprehensions .....	27
Python 3 - Lambda functions .....	27
Python 3 - map() and filter() .....	28
Python 3 - Basics of Iterators and Generators .....	28
Python 3 - itertools .....	29
Python 3 - Basics of Decorators .....	30

## Python 3 - Basics

```
#Defining a variable
my_var = 10 #type integer
my_var = "Hello" #type string
my_var = True #type boolean
```

```
#User input
input("Please enter the string you want to be printed out: ")
#raw_input() in Python 2; input() in Python 3
```

```
#Saving the input to a variable
user_says = input("Please enter the string you want to be printed
out: ")
```

```
#The input of the user is saved as a string by the input() function!
```

## Python 3 - Strings

```
#Strings - indexing
a = "Cisco Switch"
```

```
a.index("i")
```

```
#Strings - character count
a = "Cisco Switch"
```

```
a.count("i")
```

```
#Strings - finding a character
a = "Cisco Switch"
```

```
a.find("sco")
```

```
#Strings - converting the case
a = "Cisco Switch"
```

```
a.lower() #lowercase
```

```
a.upper() #uppercase
```

```
#Strings - checking whether the string starts with a character
a = "Cisco Switch"
```

```

a.startswith("C")

#Strings - checking whether the string ends with a character
a = "Cisco Switch"

a.endswith("h")

#Strings - removing a character from the beginning and the end of
a string
a = "  Cisco Switch  "

a.strip() #remove whitespaces

b = "$$$Cisco Switch$$$"

b.strip("$") #remove a certain character

#Strings - removing all occurrences of a character from a string
a = "  Cisco Switch  "

a.replace(" ", "") #replace each space character with the absence
of any character

#Strings - splitting a string by specifying a delimiter; the result
is a list
a = "Cisco,Juniper,HP,Avaya,Nortel" #the delimiter is a comma

a.split(",")

#Strings - inserting a character in between every two characters
of the string / joining the characters by using a delimiter
a = "Cisco Switch"

"_".join(a)

#Additional methods
#source:
https://www.tutorialspoint.com/python3/python\_strings.htm

capitalize()
#Capitalizes first letter of string.

lstrip()
#Removes all leading whitespace in string.

rstrip()
#Removes all trailing whitespace of string.

```

```
swapcase()  
#Inverts case for all letters in string.  
  
title()  
#Returns "titlecased" version of string, that is, all words begin  
with uppercase and the rest are lowercase.  
  
isalnum()  
#Returns true if string has at least 1 character and all characters  
are alphanumeric and false otherwise.  
  
isalpha()  
#Returns true if string has at least 1 character and all characters  
are alphabetic and false otherwise.  
  
isdigit()  
#Returns true if string contains only digits and false otherwise.  
  
islower()  
#Returns true if string has at least 1 cased character and all  
cased characters are in lowercase and false otherwise.  
  
isnumeric()  
#Returns true if a unicode string contains only numeric characters  
and false otherwise.  
  
isspace()  
#Returns true if string contains only whitespace characters and  
false otherwise.  
  
istitle()  
#Returns true if string is properly "titlecased" and false  
otherwise.  
  
isupper()  
#Returns true if string has at least one cased character and all  
cased characters are in uppercase and false otherwise.  
#source:  
https://www.tutorialspoint.com/python3/python\_strings.htm  
  
#Strings - concatenating two or more strings  
a = "Cisco"  
b = "2691"  
  
a + b
```

```
#Strings - repetition / multiplying a string
```

```
a = "Cisco"
```

```
a * 3
```

```
#Strings - checking if a character is or is not part of a string
```

```
a = "Cisco"
```

```
"o" in a
```

```
"b" not in a
```

```
#Strings - formatting v1
```

```
"Cisco model: %s, %d WAN slots, IOS %f" % ("2600XM", 2, 12.4)
```

```
"Cisco model: %s, %d WAN slots, IOS %.f" % ("2600XM", 2, 12.4)
```

```
"Cisco model: %s, %d WAN slots, IOS %.1f" % ("2600XM", 2, 12.4)
```

```
"Cisco model: %s, %d WAN slots, IOS %.2f" % ("2600XM", 2, 12.4)
```

```
#Strings - formatting v2
```

```
"Cisco model: {}, {} WAN slots, IOS {}".format("2600XM", 2, 12.4)
```

```
"Cisco model: {0}, {1} WAN slots, IOS {2}".format("2600XM", 2, 12.4)
```

```
#Strings - formatting v3 (f-strings)
```

```
model = "2950M"
```

```
wan = 4
```

```
ios = "12.2"
```

```
f"Cisco model: {model}, {wan} WAN slots, IOS {ios}"
```

```
#Strings - slicing
```

```
string1 = "O E2 10.110.8.9 [160/5] via 10.119.254.6, 0:01:00, Ethernet2"
```

```
string1[5:15] #slice starting at index 5 up to, but NOT including, index 15; so index 14 represents the last element in the slice
```

```
string1[5:] #slice starting at index 5 up to the end of the string
```

```
string1[:10] #slice starting at the beginning of the string up to, but NOT including, index 10
```

```
string1[:] #returns the entire string
```

```
string1[-1] #returns the last character in the string
```

```
string1[-2] #returns the second to last character in the string
```

```
string1[-9:-1] #extracts a certain substring using negative indexes
```

```
string1[-5:] #returns the last 5 characters in the string
```

```
string1[:-5] #returns the string minus its last 5 characters
```

```
string1[::2] #adds a third element called step; skips every second character of the string
```

```
string1[::-1] #returns string1's elements in reverse order
```



## Python 3 - Numbers and Booleans

```
#Numbers
num1 = 10
num2 = 2.5

type(num1) #checking the type of this variable; integer
type(num2) #checking the type of this variable; float

#Numbers - math operations
1 + 2 #addition

2 - 1 #subtraction

4 / 2 #division

4 * 2 #multiplication

4 ** 2 #raising to a power

5 % 2 #modulo (this means finding out the remainder after division
of one number by another)

#Numbers - float division vs. integer division (special case)
3 / 2 #float division; result is 1 in Python 2 and 1.5 in Python
3

3 // 2 #integer division; result is 1 in Python 2 and Python 3

#Numbers - order of evaluation in math operations
#Highest priority: raising to a power; Medium priority: division,
multiplication and modulo; Low priority: addition and subtraction
100 - 5 ** 2 / 5 * 2 #1st: 5 ** 2, second: / then *, third - ;
result is 90.0

#Numbers - conversion between numeric types
int(1.5) #result is 1

float(2) #result is 2.0

#Numbers - useful functions
abs(5) #the distance between the number in between parantheses and
0

abs(-5) #returns the same result as abs(5)

max(1, 2) #returns the largest number
```

```
min(1, 2) #returns the smallest number
```

```
pow(3, 2) #another way of raising to a power
```

```
#Booleans - logical operations
```

```
(1 == 1) and (2 == 2) #result is True; AND means that both operands  
should be True in order to get the expression evaluated as True
```

```
(1 == 1) or (2 == 2) #result is True; when using OR, it is enough  
if only one expression is True, in order to have True as the final  
result
```

```
not(1 == 1) #result is False; using the NOT operator means denying  
an expression, in this case denying a True expression
```

```
not(1 == 2) #result is True; using the NOT operator means denying  
an expression, in this case denying a False expression
```

```
None, 0, 0.0, 0L, 0j, empty string, empty list, empty tuple, empty  
dictionary #these values always evaluate to False
```

```
bool(None) #returns False; function that evaluates values and  
expressions
```

```
bool(0) #returns False; function that evaluates values and  
expressions
```

```
bool(2) #returns True; function that evaluates values and  
expressions
```

```
bool("router") #returns True; function that evaluates values and  
expressions
```

## Python 3 - Lists

```
#Lists
list1 = ["Cisco", "Juniper", "Avaya", 10, 10.5, -11] #creating a list

len(list) #returns the number of elements in the list

list1[0] #returns "Cisco" which is the first element in the list
(index 0)

list1[0] = "HP" #replacing the first element in the list with
another value

#Lists - methods
list2 = [-11, 2, 12]

min(list2) #returns the smallest element (value) in the list
max(list2) #returns the largest element (value) in the list

list1 = ["Cisco", "Juniper", "Avaya", 10, 10.5, -11]

list1.append(100) #appending a new element to the list

del list1[4] #removing an element from the list by index

list1.pop(0) #removing an element from the list by index

list1.remove("HP") #removing an element from the list by value

list1.insert(2, "Nortel") #inserting an element at a particular
index

list1.extend(list2) #appending a list to another list

list1.index(-11) #returns the index of element -11

list1.count(10) #returns the number of times element 10 is in the
list

list2 = [9, 99, 999, 1, 25, 500]

list2.sort() #sorts the list elements in ascending order; modifies
the list in place
```

```
list2.reverse() #sorts the list elements in descending order;  
modifies the list in place
```

```
sorted(list2) #sorts the elements of a list in ascending order and  
creates a new list at the same time
```

```
sorted(list2, reverse = True) #sorts the elements of a list in  
descending order and creates a new list at the same time
```

```
list1 + list2 #concatenating two lists
```

```
list1 * 3 #repetition of a list
```

```
#Lists - slicing (works the same as string slicing, but with list  
elements instead of string characters)
```

```
a_list[5:15] #slice starting at index 5 up to, but NOT including,  
index 15; so index 14 represents the last element in the slice
```

```
a_list[5:] #slice starting at index 5 up to the end of the list
```

```
a_list[:10] #slice starting at the beginning of the list up to,  
but NOT including, index 10
```

```
a_list[:] #returns the entire list
```

```
a_list[-1] #returns the last element in the list
```

```
a_list[-2] #returns the second to last element in the list
```

```
a_list[-9:-1] #extracts a certain sublist using negative indexes
```

```
a_list[-5:] #returns the last 5 elements in the list
```

```
a_list[:-5] #returns the list minus its last 5 elements
```

```
a_list[::2] #adds a third element called step; skips every second  
element of the list
```

```
a_list[::-1] #returns a_list's elements in reverse order
```

## Python 3 - Sets and Frozensets

```
#Sets - unordered collections of unique elements
set1 = {"1.1.1.1", "2.2.2.2", "3.3.3.3", "4.4.4.4"} #creating a set

list1 = [11, 12, 13, 14, 15, 15, 15, 11]
string1 = "aaabbcdeeffgg"

set1 = set(list1) #creating a set from a list; removing duplicate
elements; returns {11, 12, 13, 14, 15}

set2 = set(string1) #creating a set from a string; removing
duplicate characters; returns {'b', 'a', 'g', 'f', 'c', 'd', 'e'};
remeber that sets are UNORDERED collections of elements

len(set1) #returns the number of elements in the set

11 in set1 #returns True; checking if a value is an element of a
set

10 not in set 1 #returns True; checking if a value is an element
of a set

set1.add(16) #adding an element to a set

set1.remove(16) #removing an element from a set

#Frozensets - immutable sets. The elements of a frozenset remain
the same after creation.
fs1 = frozenset(list1) #defining a frozenset

fs1
frozenset({11, 12, 13, 14, 15}) #the result

type(fs1)
<class 'frozenset'> #the result

#proving that frozensets are indeed immutable
fs1.add(10)
AttributeError: 'frozenset' object has no attribute 'add'

fs1.remove(1)
AttributeError: 'frozenset' object has no attribute 'remove'

fs1.pop()
AttributeError: 'frozenset' object has no attribute 'pop'
```

```
fs1.clear()
AttributeError: 'frozenset' object has no attribute 'clear'

#Sets - methods
set1.intersection(set2) #returns the common elements of the two
sets

set1.difference(set2) #returns the elements that set1 has and set2
doesn't

set1.union(set2) #unifying two sets; the result is also a set, so
there are no duplicate elements; not to be confused with
concatenation

set1.pop() #removes a random element from the set; set elements
cannot be removed by index because sets are UNORDERED collections
of elements, so there are no indexes to use

set1.clear() #clearing a set; the result is an empty set
```

## Python 3 - Tuples

```
#Tuples - immutable lists (their contents cannot be changed by
adding, removing or replacing elements)
my_tuple = () #creating an empty tuple

my_tuple = (9,) #creating a tuple with a single element; DO NOT
forget the comma

my_tuple = (1, 2, 3, 4)

#Tuples - the same indexing & slicing rules apply as for lists
len(my_tuple) #returns the number of elements in the tuple

my_tuple[0] #returns the first element in the tuple (index 0)
my_tuple[-1] #returns the last element in the tuple (index -1)
my_tuple[0:2] #returns (1, 2)
my_tuple[:2] #returns (1, 2)
my_tuple[1:] #returns (2, 3, 4)
my_tuple[:] #returns (1, 2, 3, 4)
my_tuple[:-2] #returns (1, 2)
my_tuple[-2:] #returns (3, 4)
my_tuple[::-1] #returns (4, 3, 2, 1)
my_tuple[::2] #returns (1, 3)

#Tuples - tuple assignment / packing and unpacking
tuple1 = ("Cisco", "2600", "12.4")

(vendor, model, ios) = tuple1 #vendor will be mapped to "Cisco"
and so are the rest of the elements with their corresponding
values; both tuples should have the same number of elements

(a, b, c) = (1, 2, 3) #assigning values in a tuple to variables in
another tuple

min(tuple1) #returns "12.4"

max(tuple1) #returns "Cisco"

tuple1 + (5, 6, 7) #tuple concatenation

tuple1 * 20 #tuple multiplication

"2600" in tuple1 #returns True
784 not in tuple1 #returns True

del tuple1 #deleting a tuple
```

## Python 3 - Ranges

```
#Ranges - unlike in Python 2, where the range() function returned
a list, in Python 3 it returns an iterator; cannot be sliced
r = range(10)    #defining a range

r
range(0, 10) #the result

type(r)
<class 'range'> #the result

list(r) #converting a range to a list
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] #the result

list(r)[2:5]    #slicing a range by using the list() function first
[2, 3, 4]      #the result
```

## Python 3 - Dictionaries. Conversions between data types

```
#Dictionaries - a dictionary is an unordered set of key-value pairs
dict1 = {} #creating an empty dictionary

dict1 = {"Vendor": "Cisco", "Model": "2600", "IOS": "12.4",
"Ports": "4"}

dict1["IOS"] #returns "12.4"; extracting a value for a specified
key

dict1["IOS"] = "12.3" #modifies an existing key-value pair

dict1["RAM"] = "128" #adds a new key-value pair to the dictionary

del dict1["Ports"] #deleting a key-value pair from the dictionary

len(dict1) #returns the number of key-value pairs in the dictionary

"IOS" in dict1 #verifies if "IOS" is a key in the dictionary

"IOS2" not in dict1 #verifies if "IOS2" is not a key in the
dictionary

#Dictionaries - methods
```



```
dict1.keys() #returns a list having the keys in the dictionary as elements
```

```
dict1.values() #returns a list having the values in the dictionary as elements
```

```
dict1.items() #returns a list of tuples, each tuple containing the key and value of each dictionary pair
```

```
#Conversions between data types
```

```
str() #converting to a string
```

```
int() #converting to an integer
```

```
float() #converting to a float
```

```
list() #converting to a list
```

```
tuple() #converting to a tuple
```

```
set() #converting to a set
```

```
bin() #converting to a binary representation
```

```
hex() #converting to a hexadecimal representation
```

```
int(variable, 2) #converting from binary back to decimal
```

```
int(variable, 16) #converting from hexadecimal back to decimal
```

## Python 3 - Conditionals

#If / Elif / Else conditionals - executing code based on one or more conditions being evaluated as True or False; the "elif" and "else" clauses are optional

```
x = 5
```

```
if x > 5: #if the "x > 5" expression is evaluated as True, the
code indented under the "if" clause gets executed, otherwise the
execution jumps to the "elif" clause...
```

```
    print("x is greater than 5")
```

```
elif x == 5: #...if the "x == 5" expression is evaluated as True,
the code indented under the "elif" clause gets executed, otherwise
the execution jumps to the "else" clause
```

```
    print("x IS 5")
```

```
else: #this covers all situations not covered by the "if" and
"elif" clauses; the "else" clause, if present, is always the last
clause in the code block
```

```
    print("x is NOT greater than 5" )
```

```
#result of the above "if" block
```

```
x IS 5
```

## Python 3 - For and While Loops

#For / For Else loops - executes a block of code a number of times, depending on the sequence it iterates on; the "else" clause is optional

```
vendors = ["Cisco", "HP", "Nortel", "Avaya", "Juniper"]
```

```
for element in vendors: #iterating over a sequence and executing
the code indented under the "for" clause for each element in the
sequence
```

```
    print(element)
```

```
else: #the indented code below "else" will be executed when "for"
has finished looping over the entire list
```

```
    print("The end of the list has been reached")
```

```
#result of the above "for" block
```

```
Cisco
```

```
HP
```

```
Nortel
```

```
Avaya
```

```
Juniper
```

The end of the list has been reached

#While / While Else loops - a while loop executes as long as an user-specified condition is evaluated as True; the "else" clause is optional

```
x = 1
```

```
while x <= 10:
    print(x)
    x += 1
else:
    print("Out of the while loop. x is now greater than 10")
```

#result of the above "while" block

```
1 2 3 4 5 6 7 8 9 10
```

Out of the **while** loop. x **is** now greater than 10

## Python 3 - If / For / While Nesting

#If / For / While Nesting

```
x = "Cisco"
```

```
if "i" in x:
    if len(x) > 3: #if nesting
        print(x, len(x))
```

Cisco 5 #result of the above block

```
list1 = [4, 5, 6]
list2 = [10, 20, 30]
for i in list1:
    for j in list2: #for nesting
        print(i*j)
```

40 80 120 50 100 150 60 120 180 #result of the above block

```
x = 1
while x <= 10:
    z = 5
    x += 1
    while z <= 10: #while nesting
        print(z)
        z += 1
```

```
5 6 7 8 9 10 5 6 7 8 9 10 5 6 7 8 9 10 5 6 7 8 9 10 5 6 7 8 9 10
5 6 7 8 9 10 5 6 7 8 9 10 5 6 7 8 9 10 5 6 7 8 9 10 5 6 7 8 9 10
#result of the above block
```

```
for number in range(10):
    if 5 <= number <= 9: #mixed nesting
        print(number)
```

```
5 6 7 8 9 #result of the above block
```

## Python 3 - Break / Continue / Pass

```
#Break, Continue, Pass
```

```
list1 = [4, 5, 6]
```

```
list2 = [10, 20, 30]
```

```
for i in list1:
    for j in list2:
        if j == 20:
            break #stops the execution here, ignores the print
statement below and completely quits THIS "for" loop; however, it
doesn't quit the outer "for" loop, too!
        print(i * j)
    print("Outside the nested loop")
```

```
#result of the above block
```

```
40
```

```
Outside the nested loop
```

```
50
```

```
Outside the nested loop
```

```
60
```

```
Outside the nested loop
```

```
list1 = [4, 5, 6]
```

```
list2 = [10, 20, 30]
```

```
for i in list1:
    for j in list2:
        if j == 20:
            continue #ignores the rest of the code below for the
current iteration, then goes up to the top of the loop (inner
"for") and starts the next iteration
        print(i * j)
    print("Outside the nested loop")
```

```
#result of the above block
```

```
40
```

```
120
```

```
Outside the nested loop
```

```
50
```

```
150
```

```
Outside the nested loop
```

```
60
```

```
180
```

```
Outside the nested loop
```

```
for i in range(10):
```

```
    pass #pass is the equivalent of "do nothing"; it is actually
a placeholder for when you just want to write a piece of code that
you will treat later
```

## Python 3 - Try / Except / Else / Finally

#Try / Except / Else / Finally - handling an exception when it occurs and telling Python to keep executing the rest of the lines of code in the program

```
try:
```

```
    print(4/0) #in the "try" clause you insert the code that you
think might generate an exception at some point
```

```
except ZeroDivisionError:
```

```
    print("Division Error!") #specifying what exception types
Python should expect as a consequence of running the code inside
the "try" block and how to handle them
```

```
else:
```

```
    print("No exceptions raised by the try block!") #executed if
the code inside the "try" block raises NO exceptions
```

```
finally:
```

```
    print("I don't care if an exception was raised or not!")
#executed whether the code inside the "try" block raises an
exception or not
```

```
#result of the above block
```

```
Division Error!
```

```
I don't care if an exception was raised or not!
```

## Python 3 - Functions

#Functions - Basics

```
def my_first_function(x, y): #defining a function that takes two
    parameters
```

```
    sum = x + y
```

```
    return sum #this statement is used to exit a function and
    return something when the function is called
```

```
my_first_function(1, 2) #calling a function and passing two
    POSITIONAL arguments, the values of 1 and 2; result is 3
```

```
my_first_function(x = 1, y = 2) #calling a function and passing
    two KEYWORD arguments, the values of 1 and 2; result is 3
```

```
my_first_function(1, y = 2) #calling a function and passing mixed
    types of arguments, the values of 1 and 2; result is 3; rule:
    positional arguments always before keyword arguments!
```

```
def my_first_function(x, y, z = 3): #specifying a default parameter
    value in a function definition
```

```
def my_first_function(x, *args) #specifying a variable number of
    positional parameters in a function definition; args is a tuple
```

```
def my_first_function(x, **kwargs) #specifying a variable number
    of keyword parameters in a function definition; args is a tuple
```

```
global my_var #"importing" a variable in the global namespace to
    the local namespace of a function
```

## Python 3 - Modules

#Modules and importing - Basics

```
import sys #importing the sys module; the import statements should
    be placed before any other code in your application
```

```
from math import pi #importing only a variable (pi) from the math
    module
```

```
from math import sin #importing only a function (sin()) from the
    math module; there's no need to add the parantheses of the function
    when importing it
```

```
from math import * #importing all the names (variables and functions) from the math module
```

#Installing a non-default Python 3 module in Windows is done from the command line (e.g. the openpyxl module)

```
C:\WINDOWS\system32> pip install openpyxl
```

#Installing a non-default Python 3 module in macOS is done from the terminal (e.g. the openpyxl module)

```
mihais-MacBook-Pro:~ mihai$ pip3 install openpyxl
```

## Python 3 - File Operations

```
#Files - opening and reading a file
myfile = open("routers.txt", "r") # "r" is the file access mode for reading and it is the default mode when opening a file
```

```
myfile.mode #checking the mode in which a file has been opened
```

```
myfile.read() #method that returns the entire content of a file in the form of a string
```

```
myfile.read(5) #returning only the first 5 characters (bytes) in the file
```

```
myfile.seek(0) #moving the cursor at the beginning of the file
```

```
myfile.tell() #checking the current position of the cursor inside the file
```

```
myfile.readline() #returns the file content one line at a time, each time you use the method
```

```
myfile.readlines() #returns a list where each element is a line in the file
```

```
#Files - writing and appending to a file
newfile = open("newfile.txt", "w") #opens/creates a new file for writing; the "w" method also creates the file for writing if the file doesn't exist and overrides the file if the file already exists; remember to close the file after writing to it to save the changes!
```

```
newfile.writelines(["Cisco", "Juniper", "HP", "\n"]) #this method
takes a sequence of strings as an argument and writes those strings
to the file
```

```
newfile = open("newfile.txt", "a") #opening a file for appending
```

```
newfile = open("newfile.txt", "w+") #opens a file for both writing
and reading at the same time
```

```
newfile = open("newfile.txt", "x") #opens for exclusive creation,
failing if the file already exists
```

```
#Files - closing a file
```

```
newfile.closed #checking if a file is closed
```

```
newfile.close() #closing a file
```

```
with open("python.txt", "w") as f: #using the with-as solution,
the files gets closed automatically, without needing the close()
method
```

```
    f.write("Hello Python!\n")
```

```
#Truncating files - the file should be open for reading AND
writing, not just reading!
```

```
f = open("D:\\test.txt", "r+")
```

```
f.truncate() #this deletes all the content inside the file
```

```
#Truncating files - the file should be open for reading AND
writing, not just reading!
```

```
f = open("D:\\test.txt", "r+")
```

```
f.truncate(10) #this will keep the first 10 characters in the
file and delete the rest
```



## Python 3 - Regular Expressions

#Regular Expressions - the "re.match" and "re.search" methods

`a = re.match(pattern, string, optional flags)` #general match syntax; "a" is called a match object if the pattern is found in the string, otherwise "a" will be None

```
mystr = "You can learn any programming language, whether it is
Python2, Python3, Perl, Java, javascript or PHP."
```

`import re` #importing the regular expressions module

`a = re.match("You", mystr)` #checking if the characters "You" are indeed at the beginning of the string

`a.group()` #result is 'You'; Python returns the match it found in the string according to the pattern we provided

`a = re.match("you", mystr, re.I)` #re.I is a flag that ignores the case of the matched characters

`a = re.search(pattern, string, optional flags)` #general search syntax; searching for a pattern throughout the entire string; will return a match object if the pattern is found and None if it's not found

```
arp = "22.22.22.1          0                b4:a9:5a:ff:c8:45  VLAN#222
L"
```

`a = re.search(r"(.+?) +(\d) +(.+?)\s{2,}(\w)*", arp)` #result is '22.22.22.1'; 'r' means the pattern should be treated like a raw string; any pair of parentheses indicates the start and the end of a group; if a match is found for the pattern inside the parentheses, then the contents of that group can be extracted with the `group()` method applied to the match object; in regex syntax, a dot represents any character, except a new line character; the plus sign means that the previous expression, which in our case is just a dot, may repeat one or more times; the question mark matching as few characters as possible

`a.groups()` #returns all matches found in a given string, in the form of a tuple, where each match is an element of that tuple  
 ('22.22.22.1', '0', 'b4:a9:5a:ff:c8:45 VLAN#222', 'L')

#Regular Expressions - the "re.findall" and "re.sub" methods

```

a = re.findall(r"\d\d\.\d{2}\.[0-9][0-9]\.[0-9]{1,3}", arp)
#returns a list where each element is a pattern that was matched
inside the target string
['22.22.22.1'] #result of the above operation - a list with only
one element, the IP address matched by the regex

b = re.sub(r"\d", "7", arp) #replaces all occurrences of the
specified pattern in the target string with a string you enter as
an argument
'77.77.77.7      7      b7:a7:7a:ff:c7:77 VLAN#777      L
77.77.77.77' #result of the above operation

```

## Python 3 - Basics of OOP. Classes and Objects

```
#Classes and objects
class MyRouter(object): #creating a class which inherts from the
default "object" class
    def __init__(self, routename, model, serialno, ios): #class
constructor; initializing some variables and the method is called
whenever you create a new instance of the class
        self.routename = routename #"self" is a reference to the
current instance of the class
        self.model = model
        self.serialno = serialno
        self.ios = ios

    def print_router(self, manuf_date):
        print("The router name is: ", self.routename)
        print("The router model is: ", self.model)
        print("The serial number of: ", self.serialno)
        print("The IOS version is: ", self.ios)
        print("The model and date combined: ", self.model +
manuf_date)

router1 = MyRouter('R1', '2600', '123456', '12.4') #creating an
object by simply calling the class name and entering the arguments
required by the __init__ method in between parentheses

router1.model #accessing the object's attributes; result is '2600'

router1.print_router("20150101") #accessing a function (actually
called method) from within the class
The router name is: R1
The router model is: 2600
The serial number of: 123456
The IOS version is: 12.4
The model and date combined: 260020150101

getattr(router1, "ios") #getting the value of an attribute

setattr(router1, "ios", "12.1") #setting the value of an attribute

hasattr(router1, "ios") #checking if an object attribute exists

delattr(router1, "ios") #deleting an attribute

isinstance(router1, MyRouter) #verifying if an object is an
instance of a particular class
```

```

class MyNewRouter(MyRouter): #creating a new class (child)
    inheriting from the MyRouter parent class
    ...

issubclass(MyNewRouter, MyRouter) #returns True or False; checking
if a class is the child of another class

```

## Python 3 - List comprehensions

```

#List / Set / Dictionary comprehensions
#Instead of...
list1 = []
for i in range(10):
    j = i ** 2
    list1.append(j)

#...we can use a list comprehension
list2 = [x ** 2 for x in range(10)]

list3 = [x ** 2 for x in range(10) if x > 5] #with a conditional
statement

set1 = {x ** 2 for x in range(10)} #set comprehension

dict1 = {x: x * 2 for x in range(10)} #dictionary comprehension

```

## Python 3 - Lambda functions

```

#Lambda functions - anonymous functions
lambda arg1, arg2, ..., arg n: an expression using the arguments
#general syntax

a = lambda x, y: x * y #defining a lambda function

a(20, 10) #result is 200; calling the lambda function

#Instead of...
def myfunc(list):
    prod_list = []
    for x in range(10):
        for y in range(5):
            product = x * y

```

```

        prod_list.append(product)
    return prod_list + list

#...we can use a lambda function, a list comprehension and
concatenation on a single line of code
b = lambda list: [x * y for x in range(10) for y in range(5)] +
list

```

## Python 3 - map() and filter()

### #Map and Filter

#map() - takes a function and a sequence as arguments and applies the function to all the elements of the sequence, returning a list as the result

```

def product10(a):
    return a * 10

```

```
list1 = range(10)
```

map(product10, list1) #result is [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]; applying the product10() function to each element of list1  
#or...

```
map((lambda a: a * 10), list1) #result is [0, 10, 20, 30, 40, 50, 60, 70, 80, 90] as well
```

#filter() - takes a function and a sequence as arguments and extracts all the elements in the list for which the function returns True

```
filter(lambda a: a > 5, list1) #result is [6, 7, 8, 9]
```

## Python 3 - Basics of Iterators and Generators

#Iterators - an object which allows a programmer to traverse through all the elements of a collection

```
my_list = [1, 2, 3, 4, 5, 6, 7]
```

```
my_iter = iter(my_list) #iter() returns an iterator object
```

next(my\_iter) #in Python 2 and 3, it returns the elements of a sequence one by one; raises StopIteration when the sequence is exhausted

#Generators - special routines that can be used to control the iteration behavior of a loop; defined using the "def" keyword;

```
def my_gen(x, y): #creating a generator function
    for i in range(x):
        print("i is %d" % i)
        print("y is %d" % y)
        yield i * y    #yields the values one at a time; traversing
a sequence up to a certain point, getting the result and suspending
the execution
```

```
my_object = my_gen(10, 5) #creating a generator object
```

```
next(my_object) #manually yield the next element returned by the
my_gen() function; raises StopIteration when the sequence is
exhausted
```

```
gen_exp = (x for x in range(5)) #creating a generator expression;
similar to list comprehensions, but using parentheses instead of
square brackets
```

```
next(gen_exp) #extracting each value in the list generated by
range(5), one value at a time; raises StopIteration when the
sequence is exhausted
```

## Python 3 - itertools

#Itertools - built-in Python module for working with iterable data sets

```
import itertools
```

```
list1 = [1, 2, 3, 'a', 'b', 'c']
list2 = [101, 102, 103, 'X', 'Y']
```

```
#chain() - takes several sequences and chains them together
chain(list1, list2)
```

```
list(chain(list1, list2)) #result is [1, 2, 3, 'a', 'b', 'c', 101,
102, 103, 'X', 'Y']
```

```
#count() - returns an iterator that generates consecutive integers
until you stop it, otherwise it will go on forever
```

```
for i in count(10, 2.5):
    if i <= 50:
        print(i)
```

```

    else:
        break    #result is printing the numbers between 10 and 50
inclusively, with a step of 2.5

#cycle() - returns an iterator that simply repeats the value given
as argument infinitely; you have to find a way to break out of the
infinite loop
a = range(11, 16)

for i in cycle(a):
    print(i) #use Ctrl+C to break out of the infinite loop

#filterfalse() - returns the elements for which the function you
give as argument returns False
list(filterfalse(lambda x: x < 5, [1, 2, 3, 4, 5, 6, 7])) #in
Python 2 the result is [5, 6, 7]; in Python 3 there is no ifilter()
like in Python 2, just filter() and filterfalse()

#islice() - performs slicing; we can specify a starting point of
the slice, an end point and a step
list(islice(range(10), 2, 9, 2)) #result is [2, 4, 6, 8]

```

## Python 3 - Basics of Decorators

```

#Decorators - functions that take another function as a parameter
and extend its functionality and behavior without modifying it
def my_decorator(target_function):
    def function_wrapper():
        return "Python is the " + target_function() + " programming
language!"
    return function_wrapper

@my_decorator
def target_function():
    return "coolest"

target_function() #returns 'Python is the coolest programming
language!'

```

**Note:** The official documentation of Python 3 is available at <https://docs.python.org/3/>

---

*This cheat sheet is based on Mihai's most comprehensive, **bestselling online course for beginners**, called **Python 3 Complete Masterclass**.*

---