Python for undergraduate

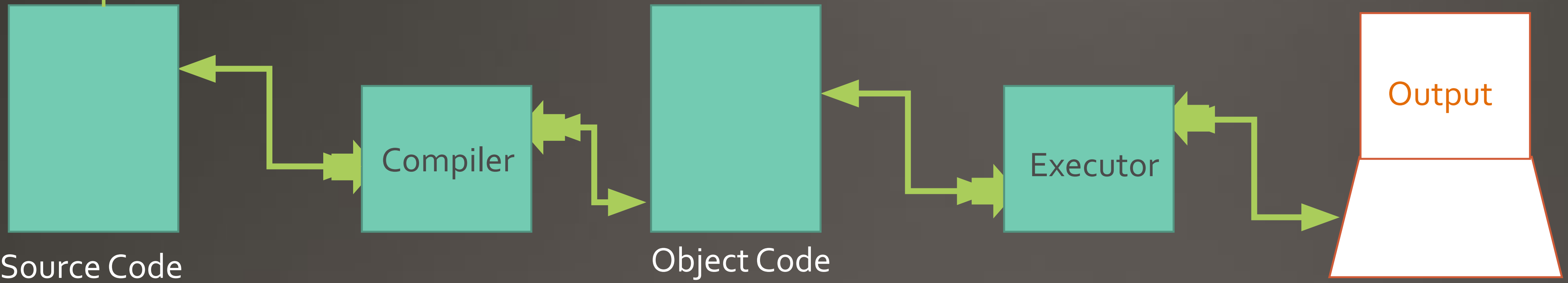
Introduction To Python

- Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.
- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.).
- Python can be treated in a procedural way, an object-oriented way or a functional way.
- The most recent major version of Python is Python 3, Python 2, although not being updated with anything other than security updates, is still quite popular.

How does python work?

Compiler vs Interpreter

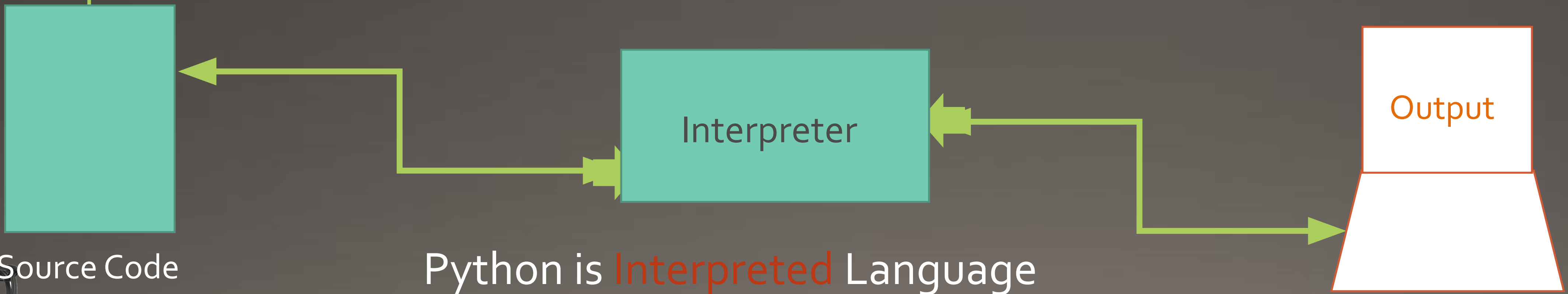
Compiler



Source Code

Object Code

Interpreter



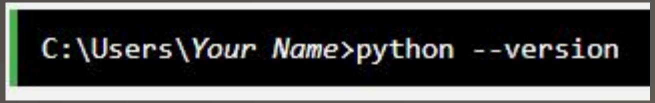
Source Code

Python is Interpreted Language

Introduction To Python

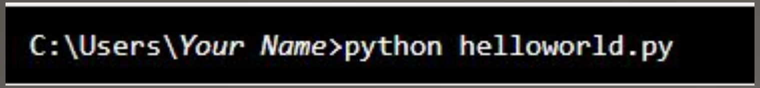
- Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.

- To get the version on the python installed in your machine run this command:



```
C:\Users\Your Name>python --version
```

- The way to run a python file is like this on the command line:



```
C:\Users\Your Name>python helloworld.py
```

TOPICS

First part: (the easy one)

- running .py file
- print()
- input()
- python from CLI:
    - python (or python3)
    - exit()

Then:

- keywords (try to guess some of them)
- variable type is not fixed (types of variables)
- multi assignment:
    - Assigning multiple values to multiple variables (x, y = 4, 5)
    - One Value to Multiple Variables (x=y=5)
    - Unpack a Collection: (fruits = [“Apple”, ‘Mango’, ‘Orange’]; x, y, z = fruits)
- Data types (dir(), help())

BEFORE WE START

- You should be able to run python files:
  - using editor and python command from CLI (command line interface), or
  - google colaboratory: Google Drive -> new -> more -> google colaboratory
- Each part of the session ends with some exercises mostly on: HackerRanck
  - You should have your hackerrank account by now!
- We will star practical, then more official:
  - recap and **more**

FIRST PART

- Exercise: Hackerrank
- https://www.hackerrank.com/challenges/py-hello-world/
  - Very simple, be careful
  - selecting the proper language
  - diff bet:
    - “Run Code” and
    - “Submit Code”
  - explore:
    - Discussions
    - Editorial
    - Tutorial

tekomoro

THEN PART (second)

- keywords
- variable type is not fixed
- multi assignment
- Data types

variables type is not fixed: try:

- *name = "ITI"*
- *print ('Hello', name)*
- *name = 5*
- *print (name)*

THEN PART (second)

- keywords
- variable type is not fixed
- multi assignment
- Data types

Multi assignment: try:

```
a, b, c = 5, 3.2, 'tekomoro'
print(a)
print(b)
print(c)

x = y = "ITI"
print (x, y)

fruits = ["Apple", 'Mango', 'Orange']
x, y, z = fruits
print(x, y, z)
```

THEN PART (second)

- keywords
- variable type is not fixed
- multi assignment
- Data types

Data Types in Python:

- Mutable Vs immutable: (search for meaning of: Mutable)
  - list, set and dict: [mutable or immutable]
  - string, tuple and frozen set ?
- Ordered: (sequence or indexed)
  - list, tuple, range, string
- Iterable: (can loop on items): string, dict, list, set, tuple (not to be confused with Ordered)

ref: https://thomas-cokelaer.info/tutorials/python/data_structures.html

THEN PART (second)

- keywords
- variable type is not fixed
- multi assignment
- Data types

Data Types in Python:

dir(object | class) and help(object | class):

- dir() and help() will display information about the object or class. Such as: operations to perform on the object

try:

```
s = {1, 4, 5, 5} # set
print(type(s))
print(dir(s))
lst = [1, 4, 5, 5] # list
print(type(lst))
print(dir(type(lst)))
print(help(lst))
```

THEN PART (second)

- keywords
- variable type is not fixed
- multi assignment
- Data types

Data Types in Python:

dir() and help(): (use grep!)

- modify the above program to display help() for one type only:
  - ‘set.py’
  - ‘list.py’
- Ordered: Should have: ‘index’: Use:
  - python list.py | grep “index”
- Iterable: Should have: ‘__iter__’: Use:
  - python list.py | grep “__iter__”

set.py:

help(set)

list.py:

help(list)

string.py:

help(str)

THEN PART (second)

- keywords
- variable type is not fixed
- multi assignment
- Data types

Data Types in Python (string are IMMUTABLE!):

- In many programming languages, strings are mutable (e.g.: C). This means that you can change one character by replacing one byte (or 2 for unicode)
- This is not the case for python. When trying to replace a character, a new memory is created for it!
- Note: We rarely need to modify part of a string!

THEN PART (second)

- keywords
- variable type is not fixed
- multi assignment
- Data types

Data Types in Python (string are IMMUTABLE!):

```
try:
s = "Hello"
print(id(s), s)

#s[0] = 'a' # error!
s = s + " World"
print(id(s), s)

lst = [1, 2, 3]
print(id(lst), lst)
lst [0] = 2
print(id(lst), lst)
lst.append(5)
print(id(lst), lst)
```

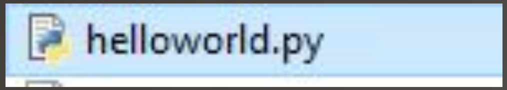
THEN PART (second)

Exercise: Hackerrank (30 min)

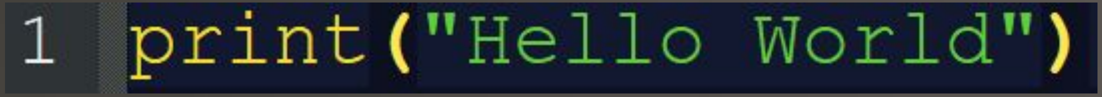
- What's Your Name?: https://www.hackerrank.com/challenges/whats-your-name/problem
- String Split and Join: https://www.hackerrank.com/challenges/python-string-split-and-join/problem
- Extra: select the ‘string’ box. You should get 15 exercise

Hello World Program

- Open an empty file and save it with the extension .py
  - Where "helloworld.py" is the name of your python file.

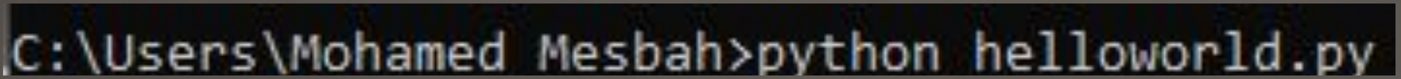


helloworld.py



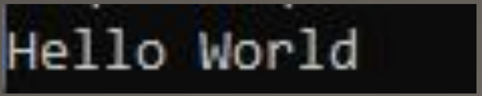
```python
print("Hello World")
```

- Save your file. Open your command line, navigate to the directory where you saved your file, and run:



```
C:\Users\Mohamed Mesbah>python helloworld.py
```

- The output should read:

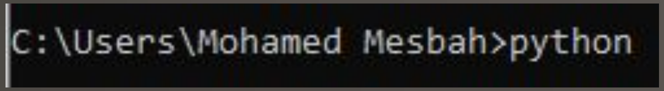


```
Hello World
```

The Python Command Line

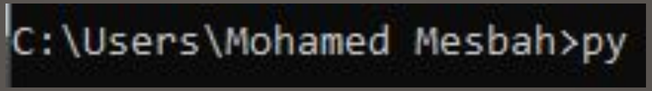
- To test a short amount of code in python sometimes it is quickest and easiest not to write the code in a file. This is made possible because Python can be run as a command line itself.

- Type the following on the Windows, Mac or Linux command line:



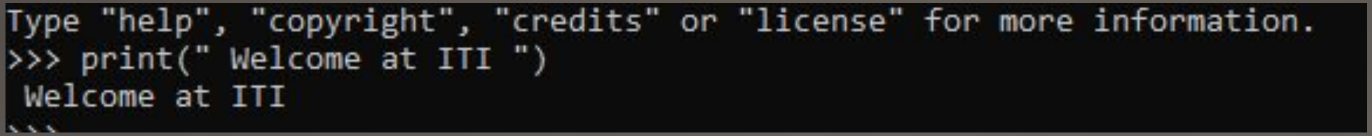
```
C:\Users\Mohamed Mesbah>python
```

OR



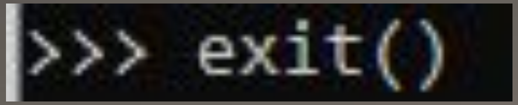
```
C:\Users\Mohamed Mesbah>py
```

- Now you are in python command line, you can write any python code and press enter to run. Example:



```
Type "help", "copyright", "credits" or "license" for more information.
>>> print(" Welcome at ITI ")
 Welcome at ITI
>>>
```

- Whenever you want to exit, run this command



```
>>> exit()
```

Python Identifiers Rules

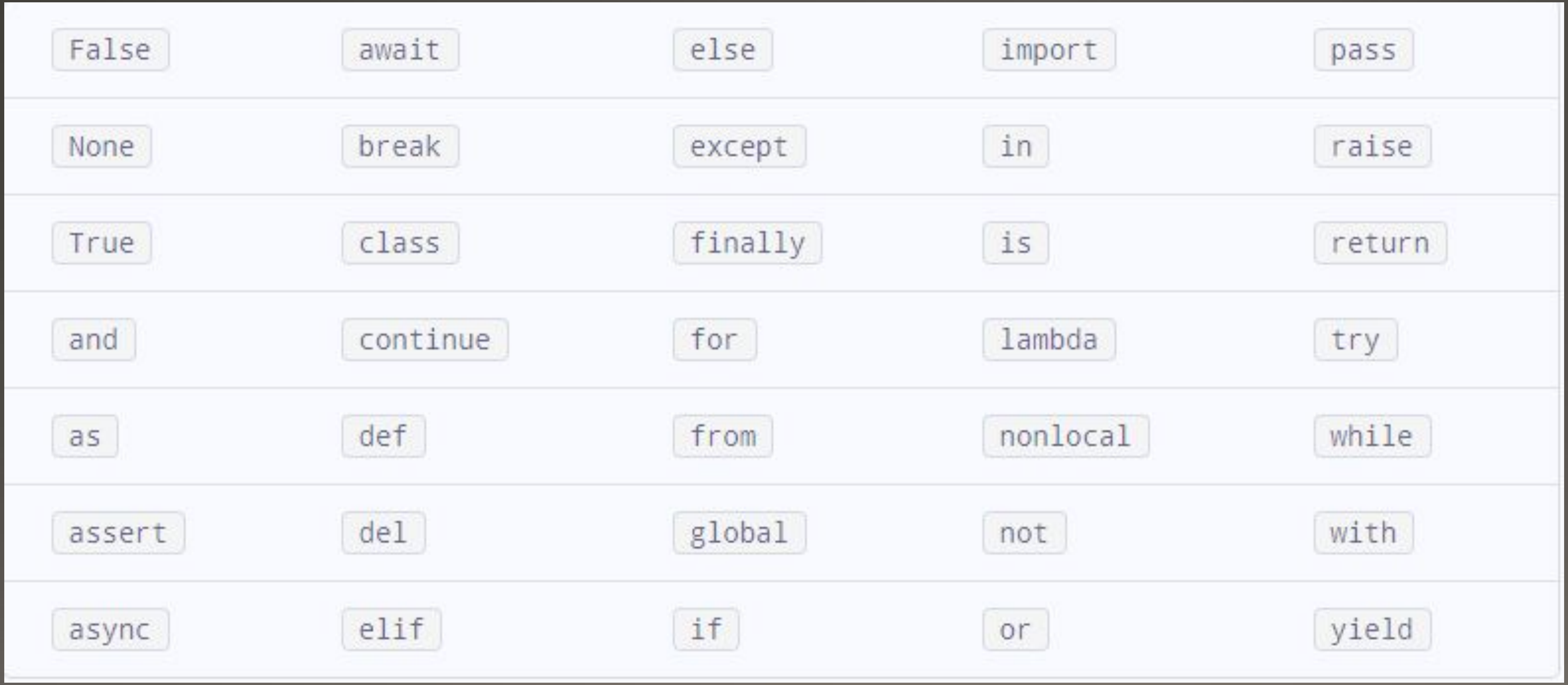
- An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.
- Starts only with : lowercase (a to z) or uppercase (A to Z) or an underscore (_)

  Name like : MyClass , My_Class , Var_1
- Keywords cannot be used as identifiers.
- We cannot use special symbols like !, @, #, $, %,? etc. in our identifier.
- An identifier can be of any length.

Python is a case-sensitive language.

Python Keywords

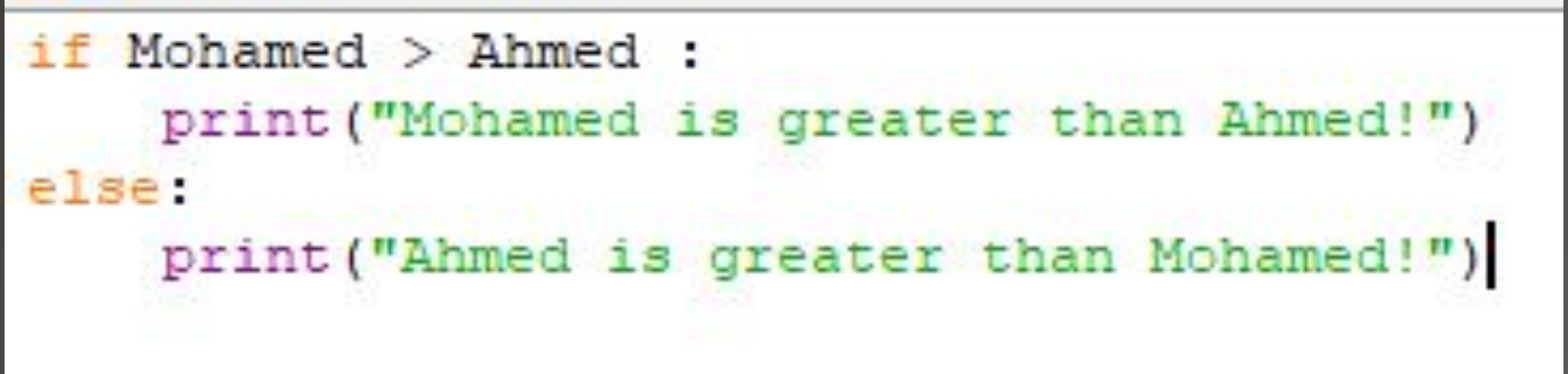
Keywords are the reserved words in Python.



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

Python Indentation

Indentation refers to the spaces at the beginning of a code line.



```
if Mohamed > Ahmed :
    print("Mohamed is greater than Ahmed!")
else:
    print("Ahmed is greater than Mohamed!")
```

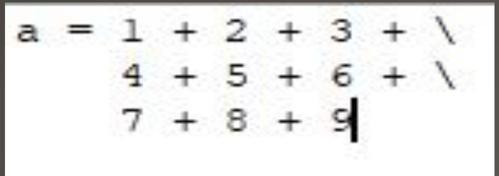
Python Statement

- Instructions that a Python interpreter can execute are called statements.

For example: a = 1 is an assignment statement. if statement, for statement, while statement, etc.

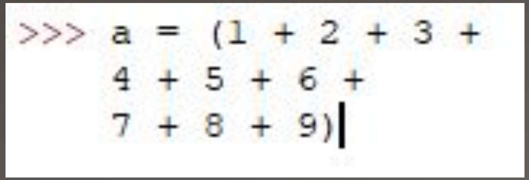
- Multi-line statement

we can make a statement extend over multiple lines with the line continuation character (\).

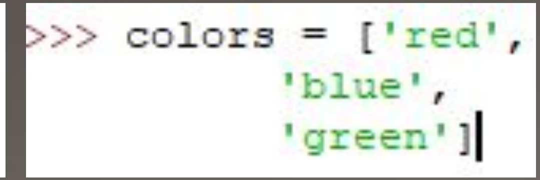


```
a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8 + 9
```

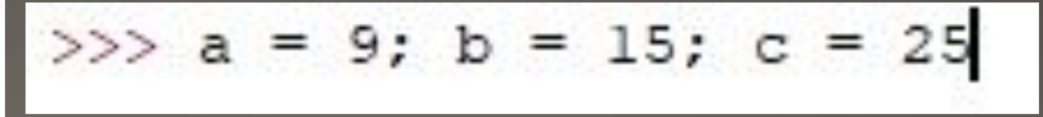
line continuation is implied inside parentheses ( ), brackets [ ], and braces { }



```
>>> a = (1 + 2 + 3 +
    4 + 5 + 6 +
    7 + 8 + 9)
```



```
>>> colors = ['red',
              'blue',
              'green']
```

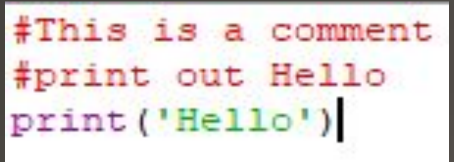


```
>>> a = 9; b = 15; c = 25
```

Python Comments

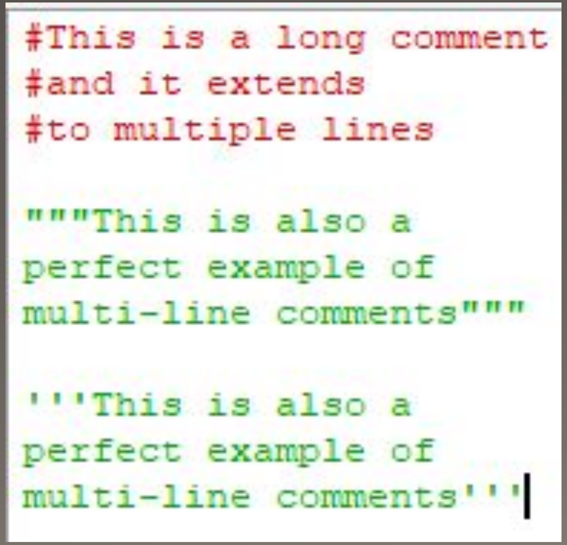
- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

Comments starts with a #, and Python will ignore them:



```
#This is a comment
#print out Hello
print('Hello')
```

Multi-line comments

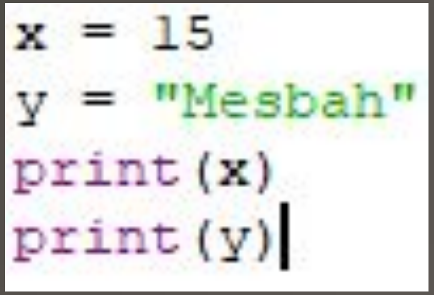


```
#This is a long comment
#and it extends
#to multiple lines

"""This is also a
perfect example of
multi-line comments"""

'''This is also a
perfect example of
multi-line comments'''
```

Python Variables

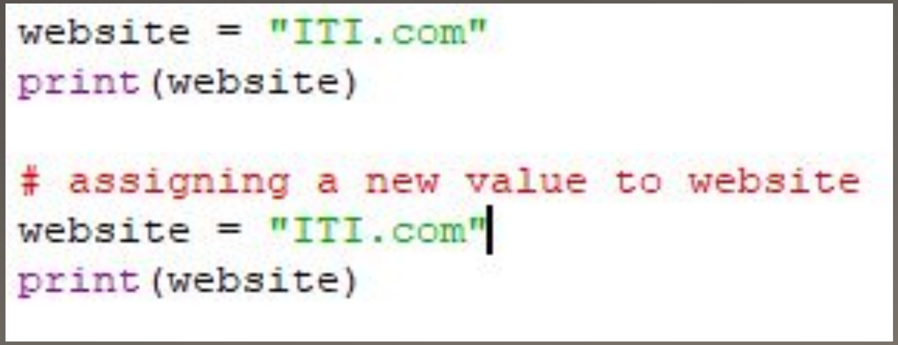
A variable is a named location used to store data in the memory.

Unlike other programming languages, Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.



```
x = 15
y = "Mesbah"
print(x)
print(y)
```

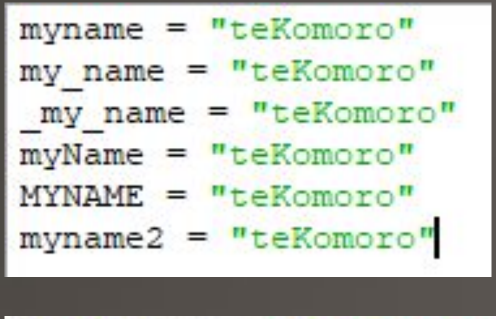
Variables do not need to be declared with any particular type and can even change type after they have been set.



```
website = "ITI.com"
print(website)

# assigning a new value to website
website = "ITI.com"
print(website)
```

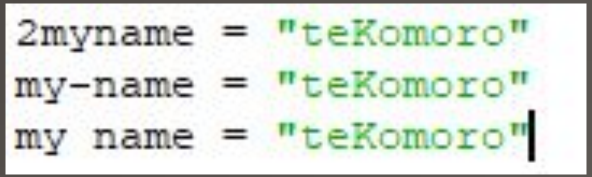
Python Variable Names

Legal variable names:



```
myname = "teKomoro"
my_name = "teKomoro"
_my_name = "teKomoro"
myName = "teKomoro"
MYNAME = "teKomoro"
myname2 = "teKomoro"
```

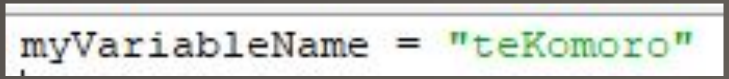
Illegal variable names:



```
2myname = "teKomoro"
my-name = "teKomoro"
my name = "teKomoro"
```

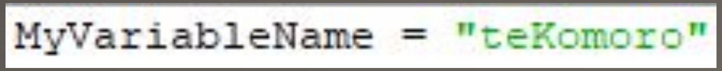
Multi Words Variable Names

- Camel Case



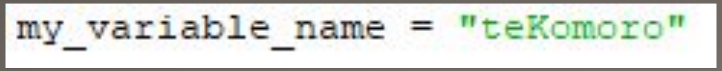
```
myVariableName = "teKomoro"
```

- Pascal Case



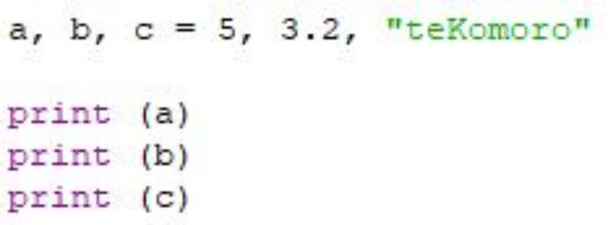
```
MyVariableName = "teKomoro"
```

- Snake Case



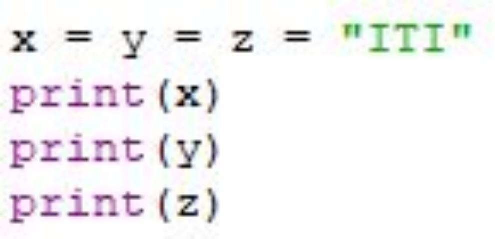
```
my_variable_name = "teKomoro"
```

Assigning multiple values to multiple variables



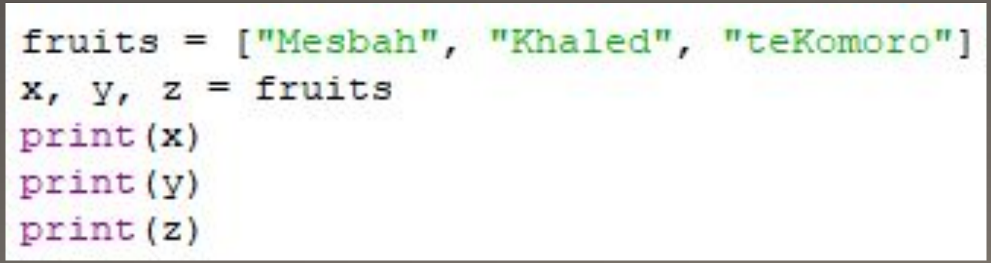
```
a, b, c = 5, 3.2, "teKomoro"

print (a)
print (b)
print (c)
```

One Value to Multiple Variables



```
x = y = z = "ITI"
print(x)
print(y)
print(z)
```

Unpack a Collection

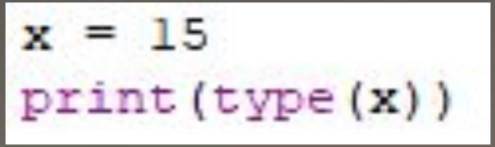


```
fruits = ["Mesbah", "Khaled", "teKomoro"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

Python Data Types

Data Types			
Text Type:	str		
Numeric Types:	int	float	complex
Sequence Types:	list	tuple	range
Mapping Type:	dict		
Set Types:	set	frozenset	
Boolean Type:	bool		
Binary Types:	bytes	bytearray	memoryview

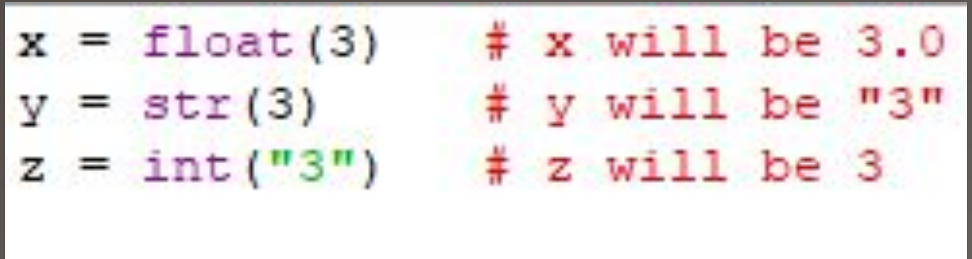
Print the data type of the variable x:



```
x = 15
print(type(x))
```

Python Casting

- int() - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

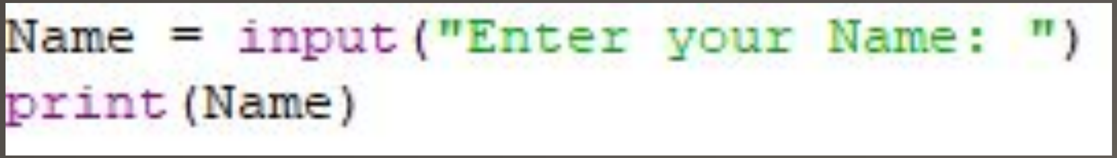


```
x = float(3)     # x will be 3.0
y = str(3)       # y will be "3"
z = int("3")     # z will be 3
```

Input internal function

- Syntax : input(Optinal_Statement)
- Description : This function first takes the input from the user and then evaluates the expression, which means Python automatically identifies whether user entered a string or a number or list.

Example :



```
Name = input("Enter your Name: ")
print(Name)
```

Python String

```
str.format(*args,**kwargs)
```

EX:

```
intro = “My Name is {0}”
intro.format(‘Mesbah’) # My Name is Mesbah
intro = “My Name is {1}, I work at {0}”
intro.format(‘ITI’,‘Ali’) # My Name is Mesbah, I work at ITI
intro = “My Name is {name}, I work at {place}”
intro.format(name=‘Mesbah’, place=‘ITI’) # My Name is Mesbah, I work at ITI
```

Python Operators

Arithmetic operators	
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponent
//	Floor Division

Identity Operators
is
is not

Comparison Operators	
==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Return True if a not equals b

Logical operators	
and	Logical AND
Or	Logical OR
not	Logical NOT

Bitwise operators	
&	AND
|	OR
^	XOR
<	Less than
~	NOT
<<	Zero fill left shift
>>	Signed right shift

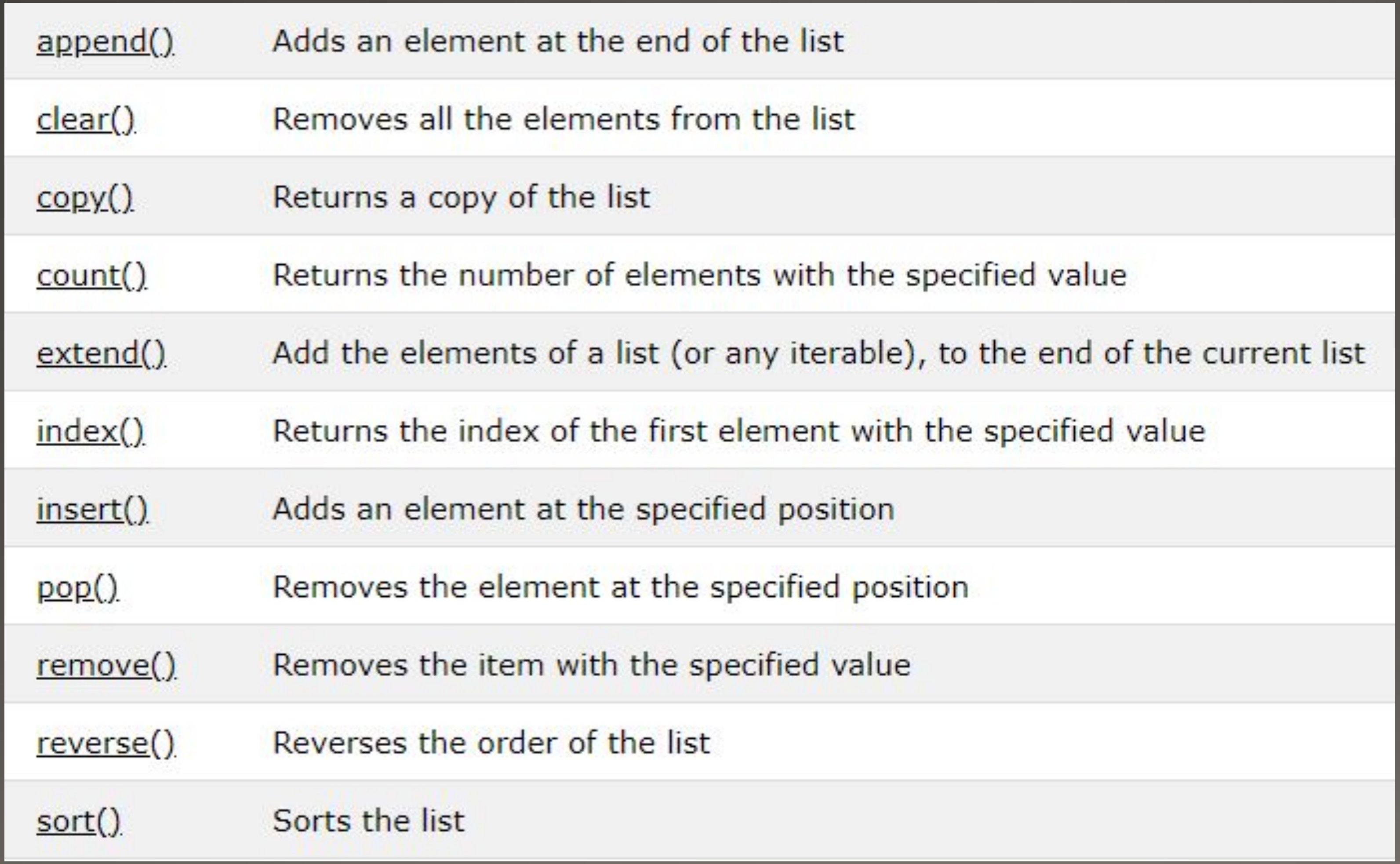
Membership Operators
in
not in

Python Lists

Lists are used to store multiple items in a single variable.

```
newList = []
newList = [5, "Hello", True]
newList[0]  #5
newList[1]  #" Hello"
newList[2]  #True
newList[-1] #True
newList[3]  #Index Error
```

Methods



append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Methods

```
myList = ["Mohamed", "C", "Python", "C++", "Khaled", "Mostafa"]

    myList.append("Mabmoud")
    myList.pop(4)
    myList.remove(“Python »)
    myList.insert(1, ‘PHP’)
    myList.sort()

myList = ["Mohamed", "C", "Python", "C++", "Khaled", "Mostafa"]
Thislist=["kiwi", "mango"]

myList.extend(Thislist)
```

FIRST PART

- Indexing and Slicing
- List comprehension
- Errors and Exceptions (Try Except)

Indexing and Slicing

- ref: https://www.geeksforgeeks.org/python-list-slicing/
  - Go to the site and do all exercise
- Slicing syntax: lst[Initial : End : IndexJump]
- Indexing: Positive Indexes:
- # Initialize list
  - *Lst = [50, 70, 30, 20, 90, 10, 50]*
  - *# Display list*
  - *print(Lst[::])*
- Indexing: Negative Indexes:
  - *# Initialize list*
  - *Lst = [50, 70, 30, 20, 90, 10, 50]*
  - *# Display list*
  - *print(Lst[-7::1])*
- Reverse a list: *print(Lst[::-1])* Discuss… Why?

FIRST PART

- Indexing and Slicing
- List comprehension
- Errors and Exceptions (Try Except)

Indexing and Slicing

- What are the other types we can apply slicing?
  - How to check if you can apply slicing on a datatypes?
    help(type) | grep index
  - tuples?
  - range?
  - string?

Indexing and Slicing

- negative indexing question:
- Is negative indexing a circular index? I.e.: 3, 2, 1, 0, -1, 2 (Ans: NO)
- Positive indexing is linear from left to right
- Negative indexing is linear from right to left

FIRST PART

- Indexing and Slicing
- List comprehension
- Errors and Exceptions (Try Except)

Errors and Exceptions (Try Except)

- Ref:
  - Go to the site and try the first exercise at least
- We will not go deeply in exceptions!
- Syntax:

```
try:
    st = "Hello"
    st[0] = 'a'
except Exception as e:
    print (e)
```

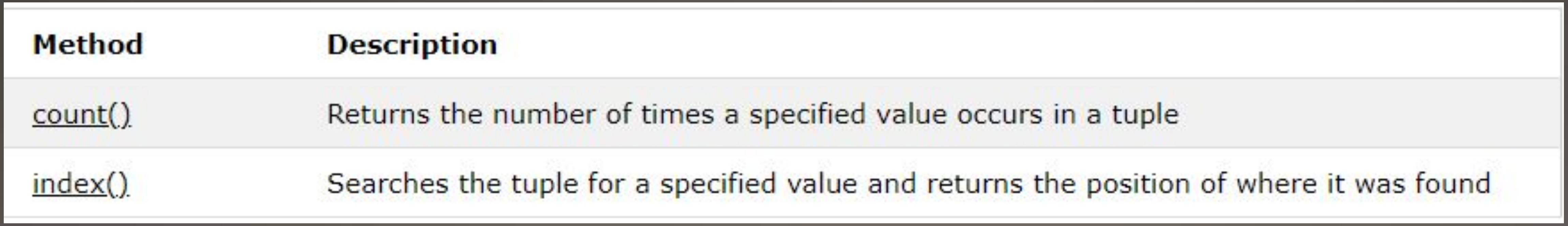
FIRST PART

- Indexing and Slicing
- List comprehension
- Errors and Exceptions (Try Except)

- Practice 1: Nice problems
- Practice 2:

Python Tuples

```
myTuple=("apple", "banana", "cherry")
myTuple[1]
#banana
myTuple[1]=4
TypeError: 'tuple' object does not support item assignment
```



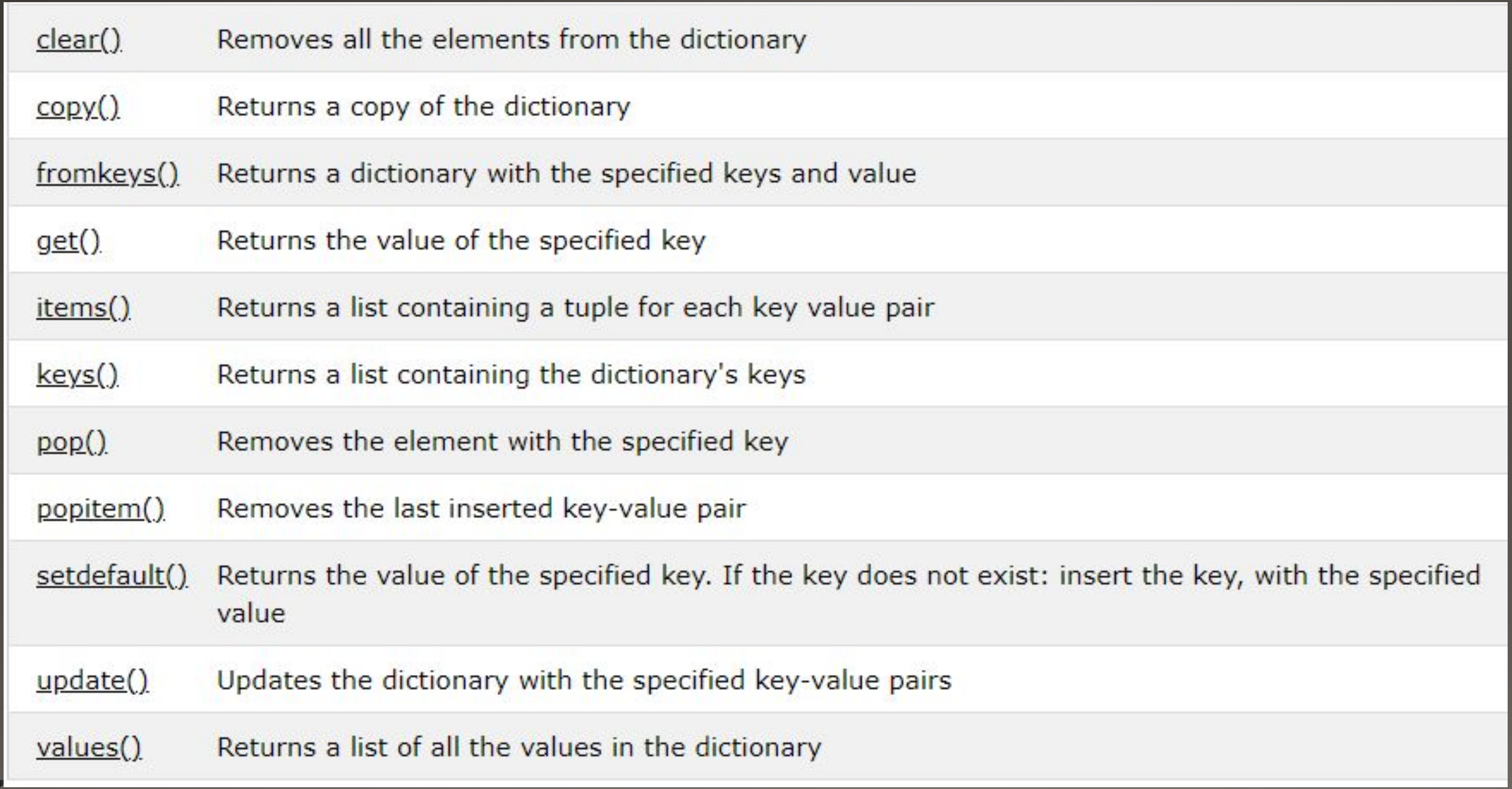
Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Python Dictionaries

Dictionaries are used to store data values in key: value pairs.

```
thisdict={}
thisdict={"Name":“Mona“,“track“:“Self_Driving"}
thisdict[“Name”]
#Mona
thisdict[“Name”]=“Mohamed”
#{Name:”Mohamed”,track:”Self_Driving”}
```

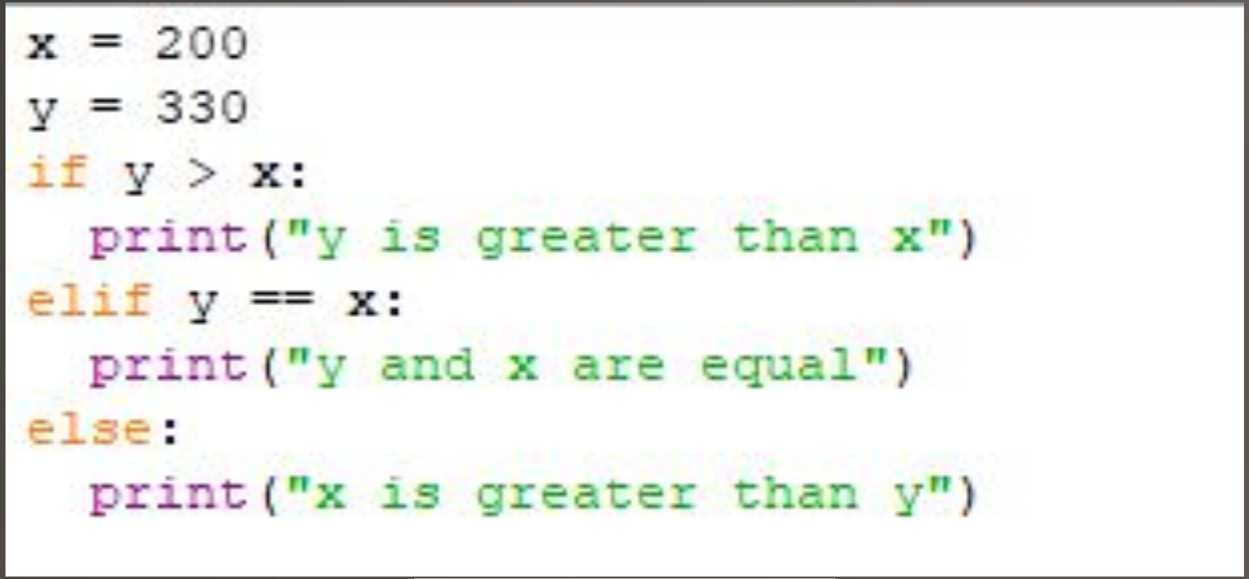
Methods



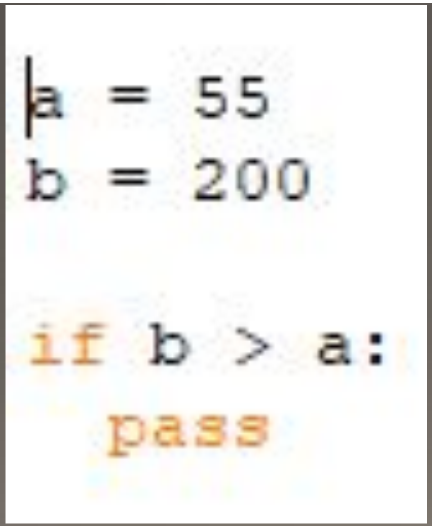
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Control Flow

If statement

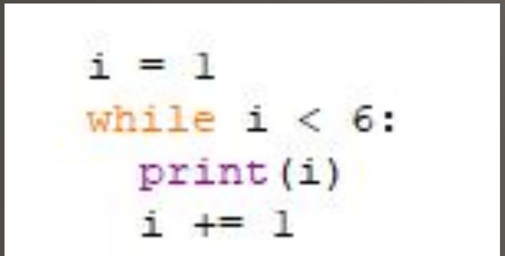


```python
x = 200
y = 330
if y > x:
  print("y is greater than x")
elif y == x:
  print("y and x are equal")
else:
  print("x is greater than y")
```



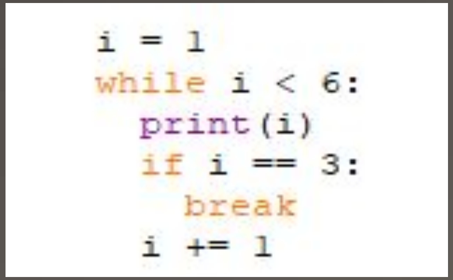
```python
a = 55
b = 200

if b > a:
  pass
```

The while Loop



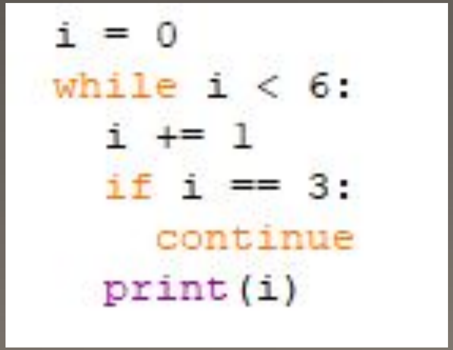
```
i = 1
while i < 6:
  print(i)
  i += 1
```

- The break Statement



```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

- The continue Statement



```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

THEN PART (second)

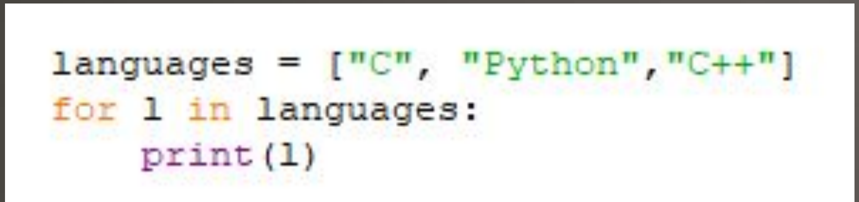
- indentation
- loops()
- functions

loops

- We loop over: Iterables (natural way)
  - *lst = [4, 8, 7, 4.4]*
  - *for item in lst:*
  - * print(item)*
- While loops:
  - *st = ''*
  - *while st != "q":*
  - * st = input()*
  - * print (st)*

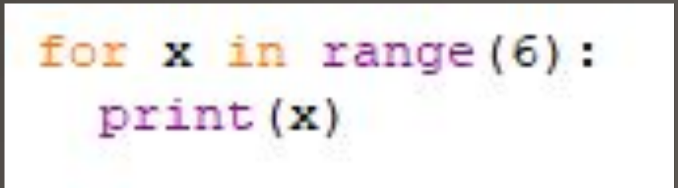
- infinite loop:
  - *while (1):*
  - * print(" - ", end='')*
- loops specific number (range):
  - *lst = [3, 6, 6 ,8]*
  - *for i in range(0, len(lst)):*
  - * print(lst[i])*

For Loops

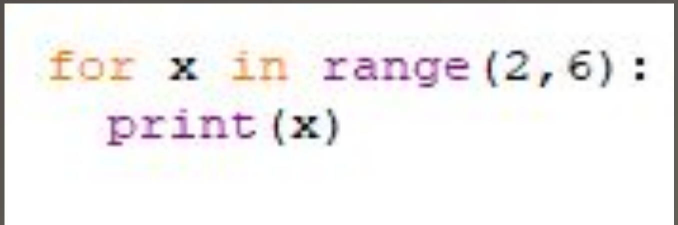


```
languages = ["C", "Python","C++"]
for l in languages:
    print(l)
```

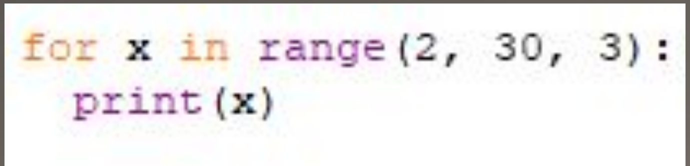
- The range() Function



```
for x in range(6):
  print(x)
```

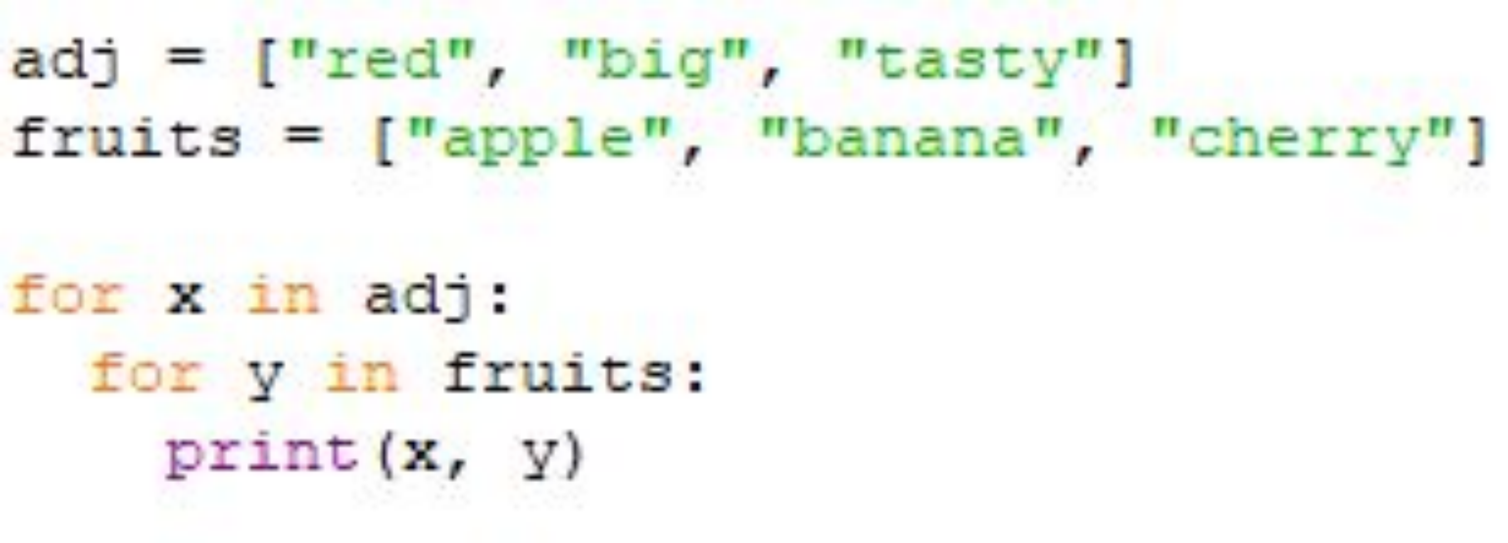


```
for x in range(2,6):
  print(x)
```



```
for x in range(2, 30, 3):
  print(x)
```

Nested Loops



```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

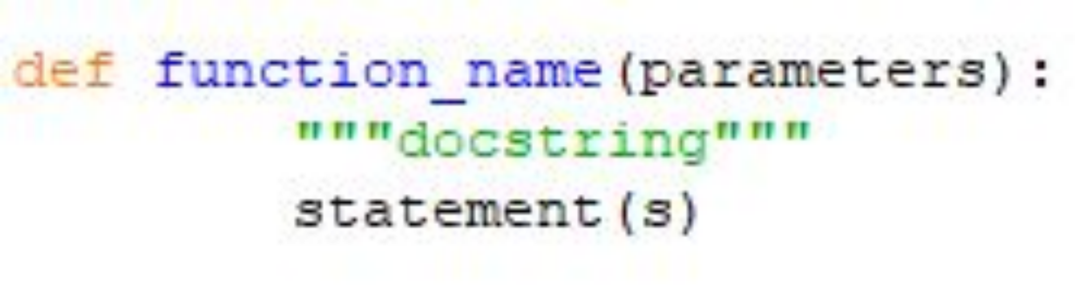
THEN PART (second)

- Exercise: Hackerrank: function and loop
  - Write a function: https://www.hackerrank.com/challenges/write-a-function/problem
  - Loops: https://www.hackerrank.com/challenges/python-loops/problem

Python Functions

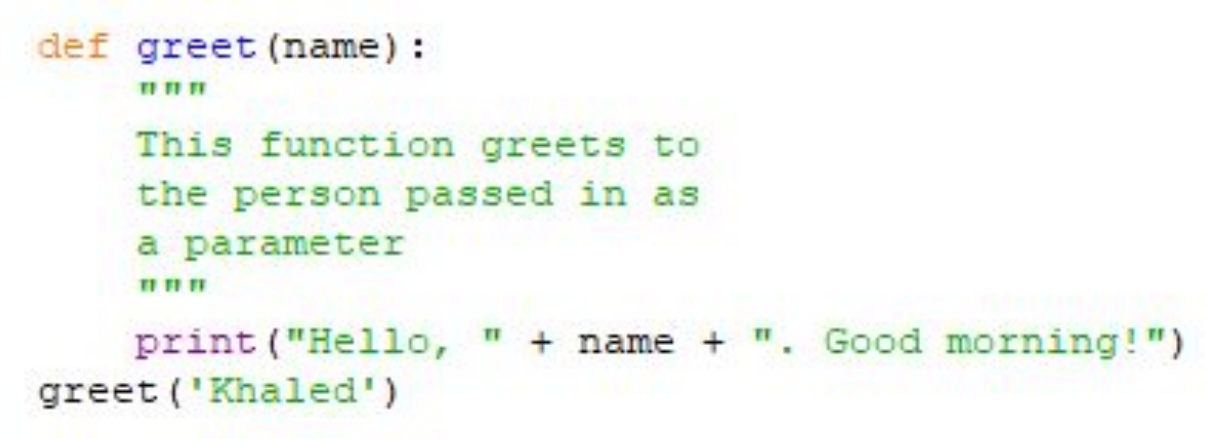
function is a group of related statements that performs a specific task.

Syntax of Function

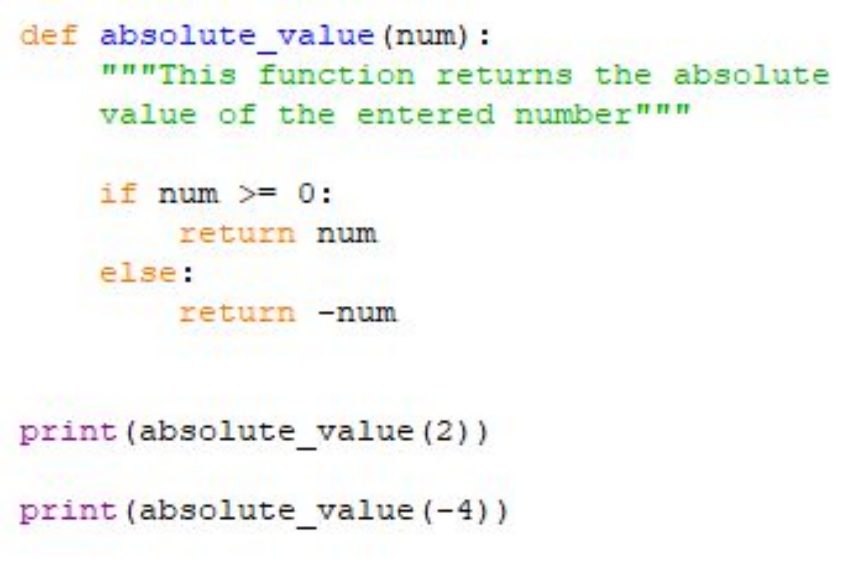


```
def function_name(parameters):
        """docstring"""
        statement(s)
```

Ex:



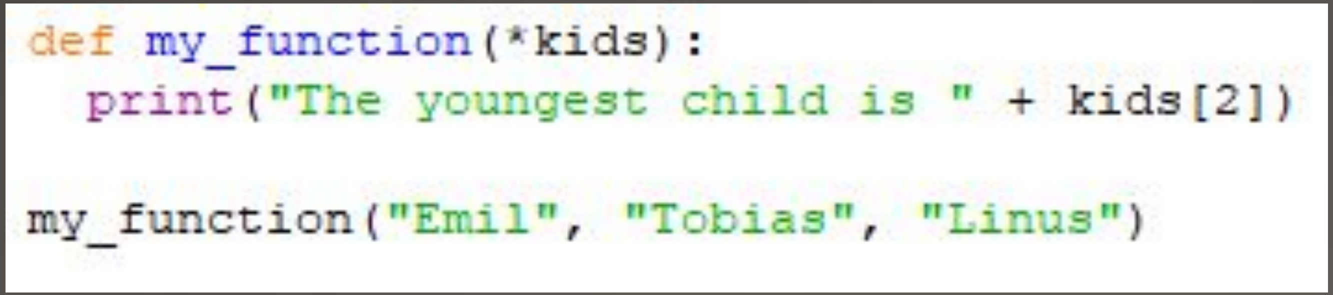
```
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
greet('Khaled')
```



```
def absolute_value(num):
    """This function returns the absolute
    value of the entered number"""

    if num >= 0:
        return num
    else:
        return -num


print(absolute_value(2))

print(absolute_value(-4))
```

Arbitrary Arguments, *args

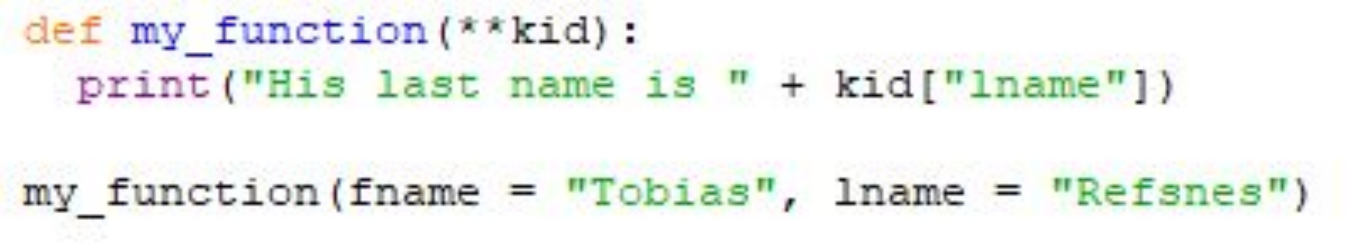
- Add a * before the parameter name in the function definition, If you do not know how many arguments that will be passed into your function.
- If the number of arguments is unknown, add a * before the parameter name:



```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

Arbitrary Keyword Arguments, **kwargs

- If the number of keyword arguments is unknown, add a double ** before the parameter name:

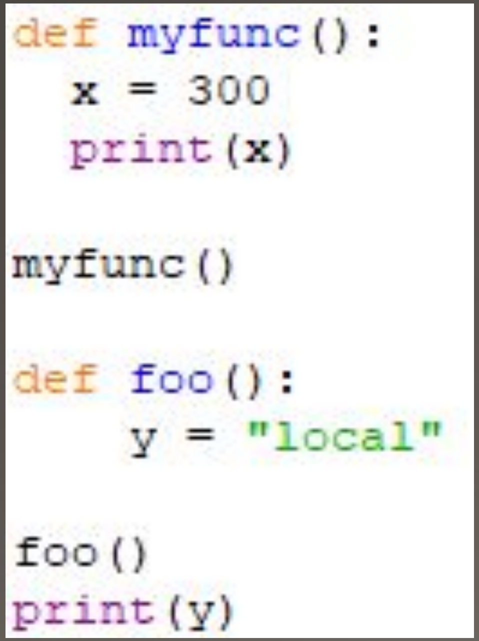


```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

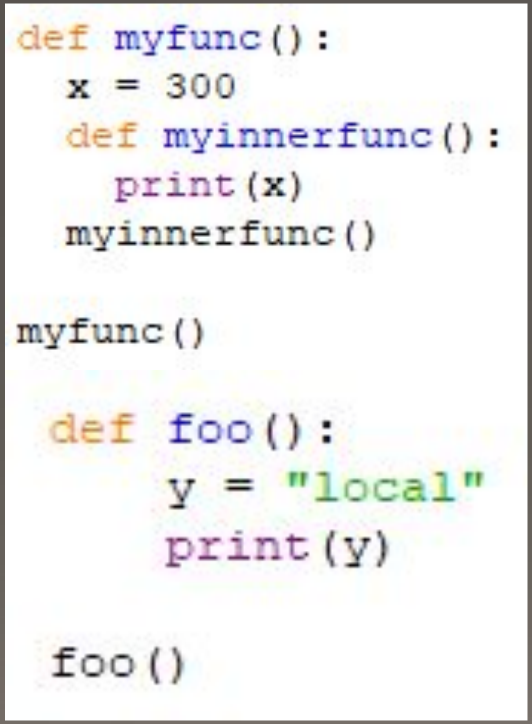
Python Scope

Local Scope

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.



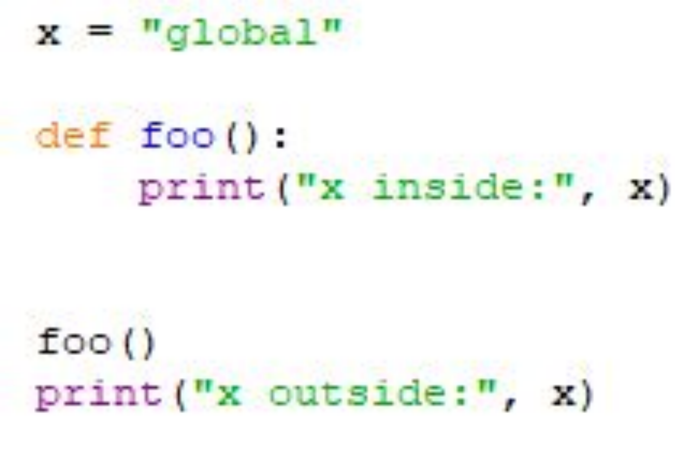
```
def myfunc():
  x = 300
  print(x)

myfunc()

def foo():
    y = "local"

foo()
print(y)
```



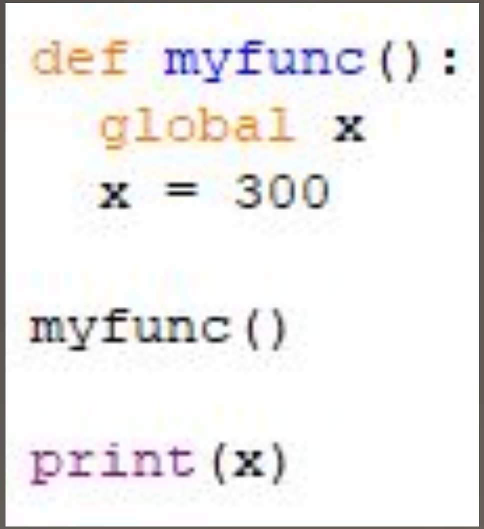
```
def myfunc():
  x = 300
  def myinnerfunc():
    print(x)
  myinnerfunc()

myfunc()

 def foo():
     y = "local"
     print(y)

 foo()
```

Global Scope

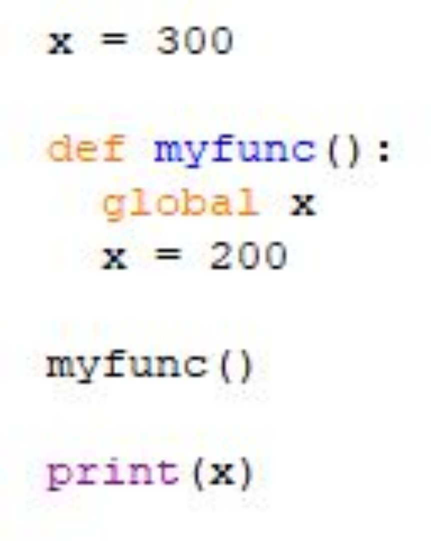
A variable created outside of a function is global and can be used by anyone:



```
x = "global"

def foo():
    print("x inside:", x)


foo()
print("x outside:", x)
```



```
def myfunc():
  global x
  x = 300

myfunc()

print(x)
```



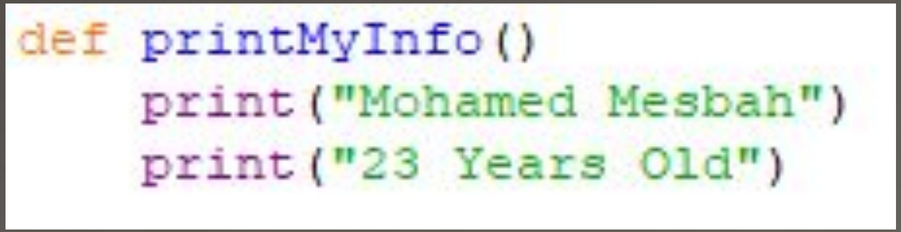
```
x = 300

def myfunc():
  global x
  x = 200

myfunc()

print(x)
```

Python Modules

A module is a code library. Simple it is a file containing a set of functions you want to include in your application. To create a module just save the code you want in a file with the file extension .py

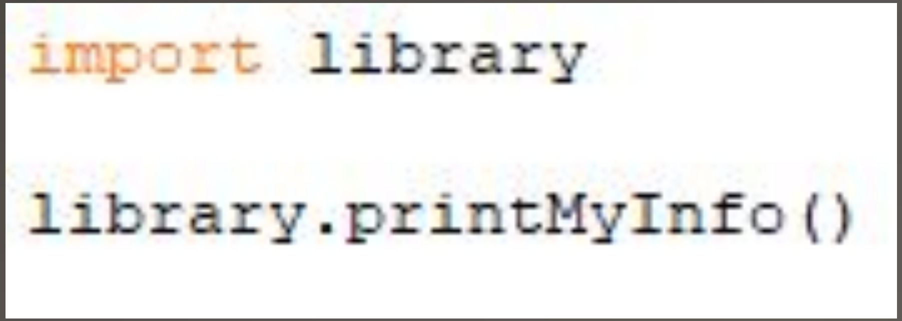
Example:

define the following function a file and save it as library.py



```
def printMyInfo()
    print("Mohamed Mesbah")
    print("23 Years Old")
```

- To import a module write the import keyword then the name of the module.
- To use a defined function in a module, use the following syntax: module.functionName()

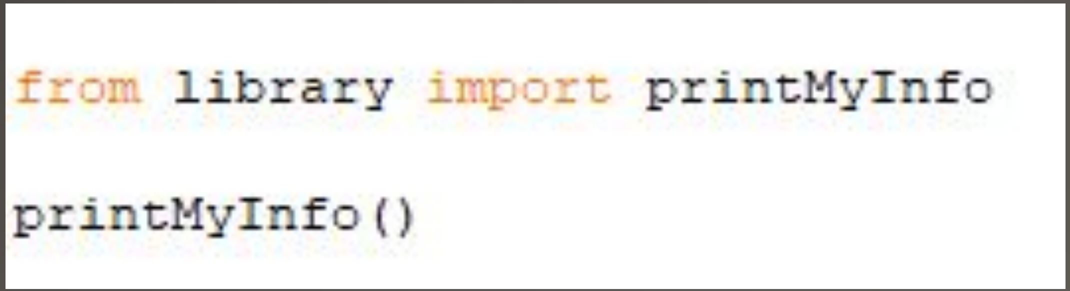


```
import library

library.printMyInfo()
```

- You can choose to import only parts from a module, by using the from keyword.

Syntax:

from Module_Name import Function_Name

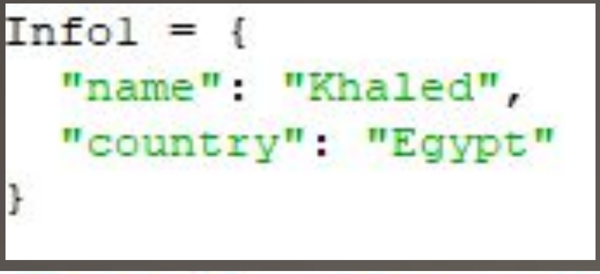


```
from library import printMyInfo

printMyInfo()
```

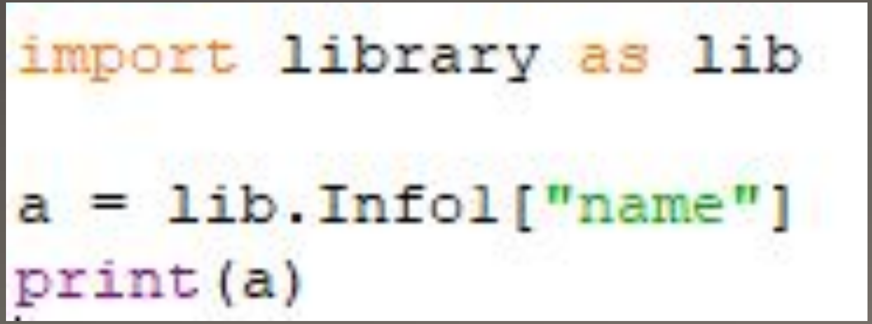
Re-naming a Module

You can create an alias when you import a module, by using the as keyword:

Save this code in the file library.py



```
Infol = {
  "name": "Khaled",
  "country": "Egypt"
}
```



```
import library as lib

a = lib.Infol["name"]
print(a)
```

File Input & Output

To open a file use the open() function, it takes two arguments; the path the of the file to be opened and the open mode.

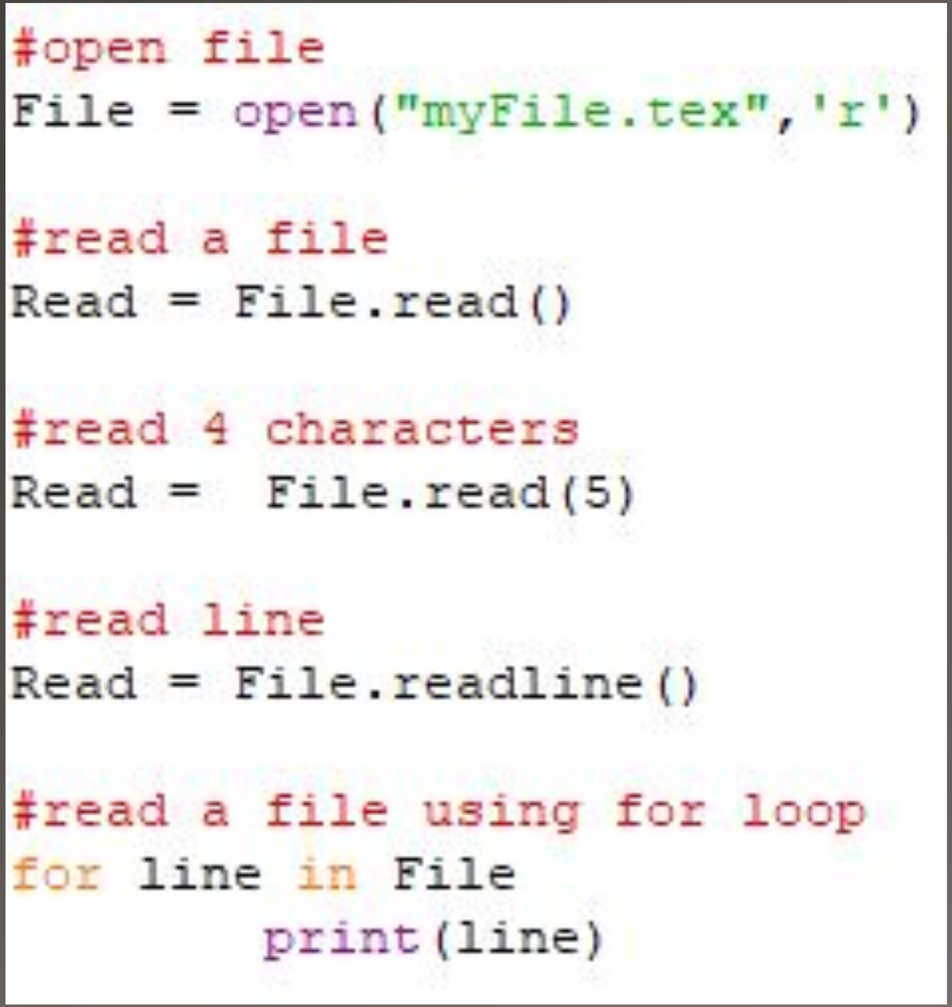
open ("File Path" , Mode)

mode

Different methods for opening a file

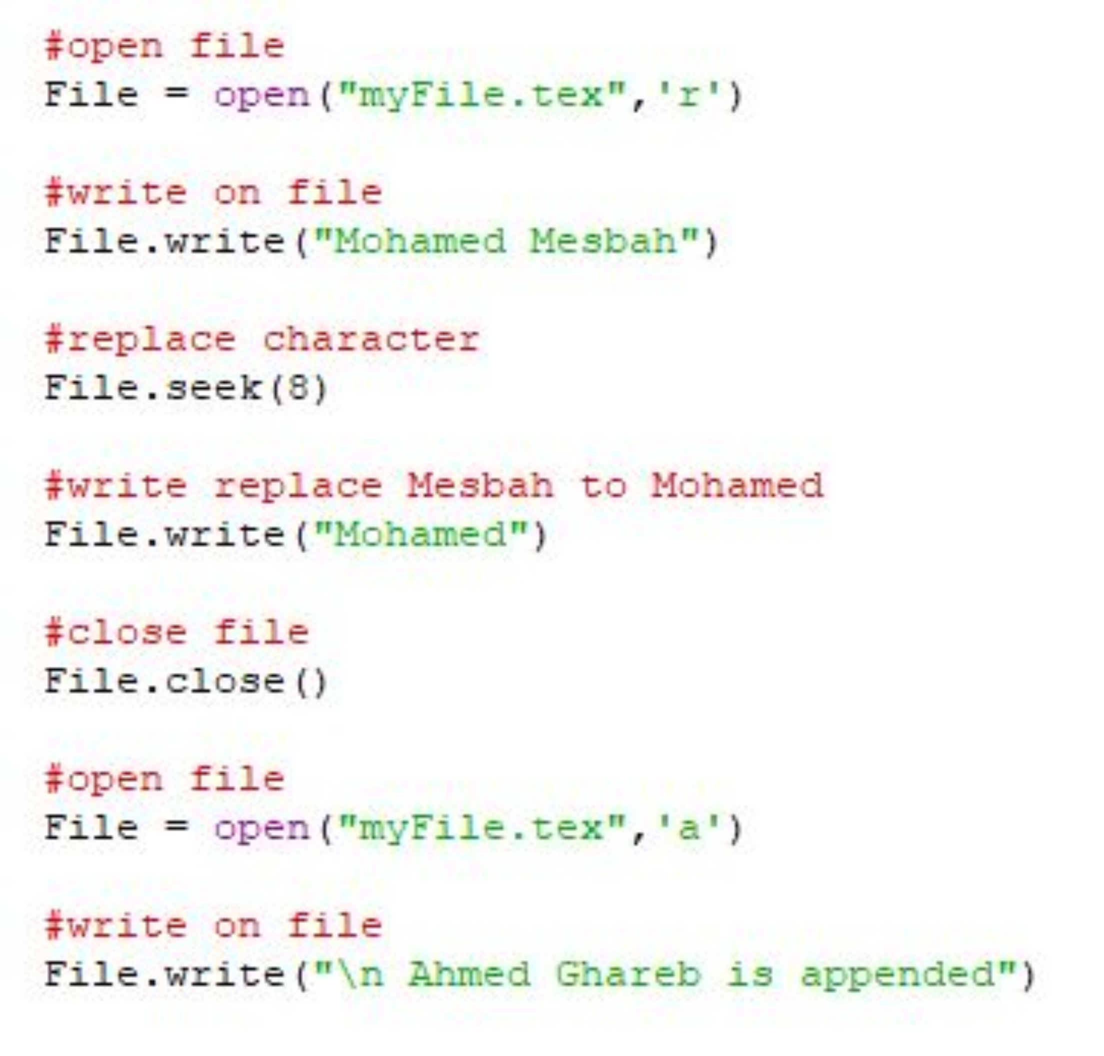
Mode Options	
r	Read Only Mode
w	Write Only Mode
a	Append Mode
r+	Reading and writing
w+	It deletes all the content of the file.
a+	Open for reading and appending (writing at end of file).

Read Files



```
#open file
File = open("myFile.tex",'r')

#read a file
Read = File.read()

#read 4 characters
Read =  File.read(5)

#read line
Read = File.readline()

#read a file using for loop
for line in File
        print(line)
```

Write on Files



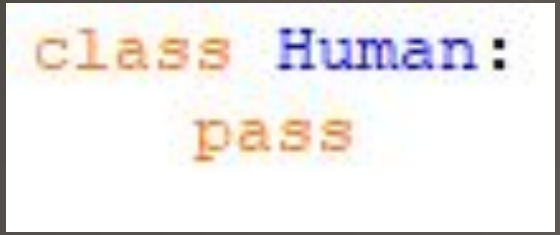
```
#open file
File = open("myFile.tex",'r')

#write on file
File.write("Mohamed Mesbah")

#replace character
File.seek(8)

#write replace Mesbah to Mohamed
File.write("Mohamed")

#close file
File.close()

#open file
File = open("myFile.tex",'a')

#write on file
File.write("\n Ahmed Ghareb is appended")
```

Classes and Objects

One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

Class

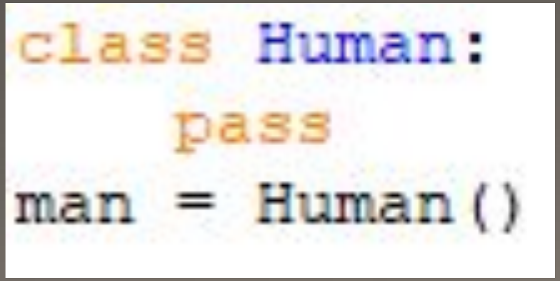
A class is a template definition of an object's properties and methods.



```
class Human:
    pass
```

Object

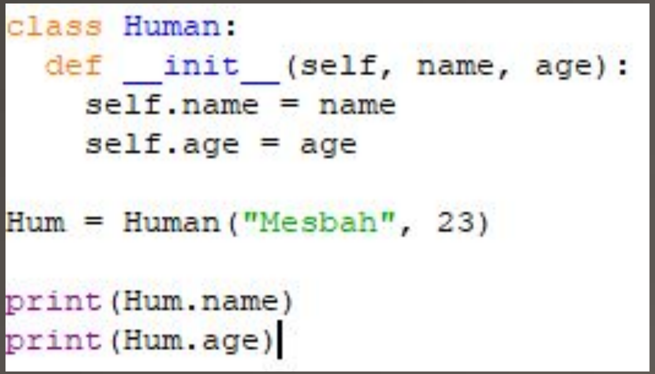
An object (instance) is an instantiation of a class.



```
class Human:
    pass
man = Human()
```

__init__() Function

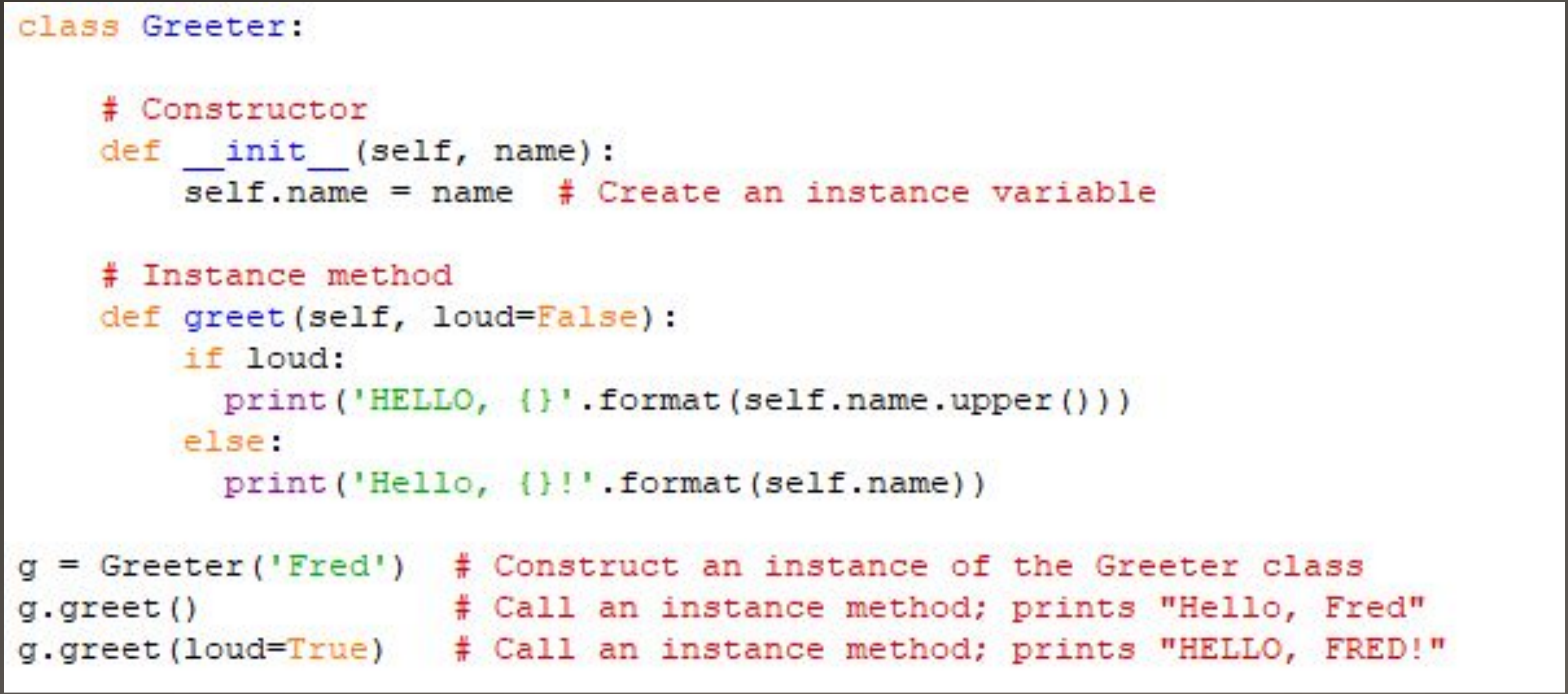
Most classes (and object) have an initialization function. This is **automatically** run when we create an instance of the class. To define the initialization routine, we name it __init__ () .



```
class Human:
  def __init__(self, name, age):
    self.name = name
    self.age = age

Hum = Human("Mesbah", 23)

print(Hum.name)
print(Hum.age)
```

self Parameter

- The word “self” refers to the fact that we are using a variable or function within the specific instance of the class or object.
- The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.



```
class Greeter:

    # Constructor
    def __init__(self, name):
        self.name = name  # Create an instance variable

    # Instance method
    def greet(self, loud=False):
        if loud:
          print('HELLO, {}'.format(self.name.upper()))
        else:
          print('Hello, {}!'.format(self.name))

g = Greeter('Fred')  # Construct an instance of the Greeter class
g.greet()            # Call an instance method; prints "Hello, Fred"
g.greet(loud=True)   # Call an instance method; prints "HELLO, FRED!"
```

Degrees() and Radians() in Python

Degrees() and Radians() are methods specified in math module in Python 3 and Python 2. Often one is in need to handle mathematical computation of conversion of radians to degrees and vice-versa.

Radians()

This function accepts the “degrees” as input and converts it into its radians equivalent.

Syntax : ***radians(deg)***
Parameters :
deg : *The degrees value that one needs to convert into radians*
Returns : *This function returns the floating point radians equivalent of argument.*
Computational Equivalent : *1 Radians = 180/pi Degrees.*

Degrees()

This function accepts the “radians” as input and converts it into its degrees equivalent.

Syntax : **degrees**(rad)
Parameters :
rad : The radians value that one needs to convert into degrees.
Returns : This function returns the floating point degrees equivalent of argument.
Computational Equivalent : 1 Degrees = pi/180 Radians.

courses to be finished

https://app.pluralsight.com/library/courses/advanced-python/table-of-contents

https://app.pluralsight.com/library/courses/core-python-classes-object-orientation/table-of-contents

https://www.udacity.com/course/introduction-to-python--ud1110

Thank You