CrashCoursesPython

November 10, 2023

#

Python Crash Course

Abdulrahman ALLOUCHE / 10 November 2023

0.0.1 1. Hello World in Python

In Python this program only requires a single line of code!

[1]:
```
print("Hello World")
```

```
Hello World
```

0.0.2 2. Reserved Keywords

Every programming language has a list of reserved key words. These words cannot be used for any other purpose than what the interpreter has pre-defined. View the list of reserved words input the following code in an empty cell

`help("keywords")`

and Press **Shift + Enter**

[55]:
```
help("keywords")
```

```
Here is a list of the Python keywords.  Enter any keyword to get more help.

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

[56]:
```
#help("for")
help(abs)
```

```
Help on built-in function abs in module builtins:

abs(x, /)
    Return the absolute value of the argument.
```

[57]:
```
print(abs.__doc__)
print(print.__doc__)
```

```
Return the absolute value of the argument.
print(value, …, sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

0.0.3 3. Variables et operators

There are 4 main types of variables - int (integer) - float (decimal number) - string (character string) - bool (boolean)

[58]:
```
x = 3 # type int
y = 2.5 # type float
prenom = 'Pierre' # type string
z = True # type Bool
```

[15]:
```
# Arithmetic operations
print('x + y =', x + y)
print('x - y =', x - y)
print('x / y =', x / y)
print('x // y =', x // y) # integer division (very useful for Numpy arrays)
print('x * y =', x * y)
print('x ^ y =', x ** y) # x to the power of y
#print('x ^ y =', y^x)
print('10%3 =', 10%3) # #Remainder of 10 divided by 3
```

```
x + y = 5.5
x - y = 0.5
x / y = 1.2
x // y = 1.0
x * y = 7.5
```

```
x ^ y = 15.588457268119896
10%3 = 1
```

[19]:
```python
print(3.8//1.2)
print(3.8/1.2)
```

```
3.0
3.1666666666666665
```

[20]:
```python
# Comparison operations
print ('equality:', x == y)
print ('inequality:', x != y)
print ('less than or equal:', x <= y)
print ('greater than or equal:', x>= y)
```

```
equality: False
inequality: True
less than or equal: False
greater than or equal: True
```

[22]:
```python
# Logical operations
print ('ET:', False and True)
print ('OR:', False or True)
print ('Exclusive OR:', False ^ True)
print ('Exclusive OR:', True ^ True)
print ('OR:', True or True)
```

```
ET: False
OR: True
Exclusive OR: True
Exclusive OR: False
OR: True
```

[23]:
```python
import math
#square root. This requires importing the math library
sq = math.sqrt(4)
print(sq)
```

```
2.0
```

[24]:
```python
from math import *
#square root. This requires importing the math library
sq = sqrt(4)
print(sq)
```

```
2.0
```

0.0.4 4. Working with Strings

Get the first and last letter of the string

[25]:
```
mystring = "computer"
print(mystring[0])
print(mystring[7])
print(mystring[-1])
```

```
c
r
r
```

Get a part of the string

[26]:
```
mystring = "computer"
#Get all the letters in the string between
print(mystring[3:5])
#Get all the letters in the string starting at position 3
print(mystring[3:])
#Get all the letters in the string ending before position 3
print(mystring[:3])
```

```
pu
puter
com
```

concatenante strings

[27]:
```
# concatenante strings
greeting = "Hello "
name = "John"
print(greeting + name) #  You just need to use the "+" operator to concatenante␣
↪2 strings
```

```
Hello John
```

Check if a character or group of characters exist in a string

[28]:
```
# Check if a character or group of characters exist in a string
sample = "A43B23C83"
#check for a single character. Will return True if character is in the string
print("B" in sample)
#check for a group of characters
print("23C" in sample)
#will return false if not in the string
print("Z" in sample)
```

```
True
True
False
```

Make all letters in a string upper case

[29]:
```
mystring = "to be or not to be"
ustr = mystring.upper()
print(ustr)
```

```
TO BE OR NOT TO BE
```

[30]:
```
mystring = "to be or not to be"
mystring.upper()
print(mystring)
```

```
to be or not to be
```

Make all characters in a string lower case

[31]:
```
mystring = "THAT IS THE QUESTION"
print(mystring.lower())
```

```
that is the question
```

Make first character of a string upper case

[33]:
```
#Make first character of a string upper case
mystring = "whether 'tis nobler in the mind to suffer"
print(mystring.capitalize())
```

```
Whether 'tis nobler in the mind to suffer
```

Use Title case. First character of every word capital.

[34]:
```
#Use Title case. First character of every word capital.
mystring = "to kill a mockingbird"
print(mystring.title())
```

```
To Kill A Mockingbird
```

Check if a string is a number

[35]:
```
mystring = "1212312"
print(mystring.isnumeric())
```

```
True
```

[36]:
```
mystring = "1212A312"
print(mystring.isnumeric())
```

```
False
```

Get the length of a string

[37]:
```
mystring = "To be, or not to be, that is the question"
print(len(mystring))
```

```
41
```

Fill the string with zeros until it is 8 characters long

[39]:
```
id_1 = "12121"
id_2 = "434"
#pads the left side of a string to equal the specified length.
print(id_1.zfill(8))
print(id_2.zfill(8))
```

```
00012121
00000434
```

The format() method formats the specified value(s) and insert them inside the string's placeholder.

[40]:
```
x=90.1111
y=100.87122
txt = "X= {:.2f} Y={:.3f}".format(x, y)
print(txt)
```

```
X= 90.11 Y=100.871
```

[41]:
```
txt = "For only {price:.2f} dollars!".format(price=49)
print(txt)
```

```
For only 49.00 dollars!
```

[42]:
```
#named indexes:
txt1 = "My name is {fname}, I'm {age}".format(fname = "John", age = 36)
#numbered indexes:
txt2 = "My name is {0}, I'm {1}".format("John",36)
#empty placeholders:
txt3 = "My name is {}, I'm {}".format("John",36)
print(txt1)
print(txt2)
print(txt3)
```

```
My name is John, I'm 36
My name is John, I'm 36
My name is John, I'm 36
```

[43]:
```
#named indexes:
txt1 = "  {age}, {fname}".format(fname = "John", age = 36)
print(txt1)
```

```
  36, John
```

0.0.5 5. Getting User Keyboard input

[44]:
```
ival = input("Give me an integer : ")
print("ival : " + ival)
print("type ival : " , type(ival))
# conversion from string to float or to int
print("int(float(ival)) +2 : " + str(int(float(ival))+2))
#Press Shift + Enter on the Notebook Cell
#Enter a value
#Press Enter
#Output will include your value
```

```
Give me an integer : 555.6
ival : 555.6
type ival :  <class 'str'>
int(float(ival)) +2 : 557
```

0.0.6 6. Decision making in python (if Statements)

[45]:
```
b = 5
# not equal to
if b != 4:
    print(b)
```

```
5
```

Handling Multiple conditions using if/elif statements

[46]:
```
# Handling Multiple conditions using if/elif statements
color = "blue"
#handle multiple conditions with if/elif
if color == "red":
    print("color is red")
elif color == "green":
    print("color is green")
elif color == "blue":
    print("color is blue")
```

```
color is blue
```

Adding a default condition to you if statements

[47]:
```
# Adding a default condition to you if statements
color = "yellow"
#handle multiple conditions with if/elif/else
if color == "red":
    print("color is red")
elif color == "green":
```

```
    print("color is green")
elif color == "blue":
    print("color is blue")
else:
    print("color is not red, blue or green")
```

```
color is not red, blue or green
```

0.0.7 7. loops in Python

Loops allow you to execute a block of code multiple times. There are 2 types of loops in Python: - For Loops - While Loops

Writing a For Loop based on a range: The range function generates a sequence of numbers starting with 0.

[48]:
```
#creates a loop starting with 0 and ending in 4
for x in range(5):
    print(x)
```

```
0
1
2
3
4
```

Same loop with while

[49]:
```
x = 0
while x < 5:
    print(x)
    x += 1 # incrémente x de 1 (équivalent de x = x+1)
```

```
0
1
2
3
4
```

Loop using range from items 0 to item 10 count by 2. (5 items)

[50]:
```
# for loop using range from items 0 to item 10 count by 2. (5 items)
for x in range(0,10,2):
    print(x)
```

```
0
2
4
6
8
```

0.0.8 8. Lists in Python

A list is a collection of objects. It contains the following features:

- It is ordered
- It is changeable
- It allows duplicate members

Lists are written with square brackets

[59]:
```
#Lists are written with square brackets
shoppinglist = ["milk","bread","apples","eggs","rice"]
print(shoppinglist)
```

```
['milk', 'bread', 'apples', 'eggs', 'rice']
```

Access an item or items in a list

[60]:
```
#Create a list
shoppinglist = ["milk","bread","apples","eggs","rice"]
#print the first item in the list
print(shoppinglist[0])
#print the last item in the list
print(shoppinglist[-1])
#print items starting at position 2 and ending before position 4
print(shoppinglist[2:4])
```

```
milk
rice
['apples', 'eggs']
```

Loop through a Python List

[61]:
```
shoppinglist = ["milk","bread","apples","eggs","rice"]
for x in shoppinglist:
    print(x)
for x in shoppinglist:
    print(x, end=" ")
```

```
milk
bread
apples
eggs
rice
milk bread apples eggs rice
```

Check if an item exists in a list

[62]:
```
shoppinglist = ["milk","bread","apples","eggs","rice"]
if "milk" in shoppinglist:
```

```
    print("milk is in your shopping list")
```

```
milk is in your shopping list
```

Get the length of a Python List

[63]:
```
shoppinglist = ["milk","bread","apples","eggs","rice"]
print(len(shoppinglist))
```

```
5
```

Sort a Python List

[64]:
```
shoppinglist = ["milk","bread","apples","eggs","rice"]
shoppinglist.sort()
print(shoppinglist)
```

```
['apples', 'bread', 'eggs', 'milk', 'rice']
```

[65]:
```
shoppinglist.sort(reverse=True)
print(shoppinglist)
```

```
['rice', 'milk', 'eggs', 'bread', 'apples']
```

Reverse sort items in a Python List

[66]:
```
shoppinglist = ["milk","bread","apples","eggs","rice"]
shoppinglist.reverse()
print(shoppinglist)
```

```
['rice', 'eggs', 'apples', 'bread', 'milk']
```

Add a single item to a Python List

[67]:
```
shoppinglist = ["milk","bread","apples","eggs","rice"]
#add cookies to the list using append
shoppinglist.append("cookies")
print(shoppinglist)
```

```
['milk', 'bread', 'apples', 'eggs', 'rice', 'cookies']
```

Add multiple items to a Python List

[68]:
```
#Use extend to add a collection to a list
proteinlist = ["steak","chicken","fish"]
shoppinglist.extend(proteinlist)
print("using extend")
print(shoppinglist)
```

```
using extend
['milk', 'bread', 'apples', 'eggs', 'rice', 'cookies', 'steak', 'chicken',
'fish']
```

[69]:
```
shoppinglist = ["milk","bread","apples","eggs","rice"]
shoppinglist.append("cookies")
proteinlist = ["steak","chicken","fish"]
print(shoppinglist+proteinlist) # new list
print(shoppinglist)
```

```
['milk', 'bread', 'apples', 'eggs', 'rice', 'cookies', 'steak', 'chicken',
'fish']
['milk', 'bread', 'apples', 'eggs', 'rice', 'cookies']
```

Add an item to a Python List at a specific Location

[70]:
```
#Use extend to add a collection to a list
shoppinglist = ["steak","chicken","fish"]
shoppinglist.insert(1,"detergent")
print("using insert")
print(shoppinglist)
```

```
using insert
['steak', 'detergent', 'chicken', 'fish']
```

Remove an item from a Python List by index using pop()

[71]:
```
shoppinglist = ["milk","bread","apples","eggs","rice"]
#Use pop to remove an item by index
print("use pop")
shoppinglist.pop(1)
print(shoppinglist)
```

```
use pop
['milk', 'apples', 'eggs', 'rice']
```

Remove an item from a Python List by index using del keyword

[73]:
```
shoppinglist = ["milk","bread","apples","eggs","rice"]
print("use del")
del shoppinglist[0]
print(shoppinglist)
```

```
use del
['bread', 'apples', 'eggs', 'rice']
```

Remove an item from a Python List by value

[74]:
```
shoppinglist = ["milk","bread","apples","eggs","bread", "rice"]
print("use remove")
shoppinglist.remove("bread")
print(shoppinglist)
```

```
use remove
['milk', 'apples', 'eggs', 'bread', 'rice']
```

Remove all items from a Python List (empty a list)

[75]:
```
print("use clear")
shoppinglist = ["milk","bread","apples","eggs","rice"]
shoppinglist.clear()
print(shoppinglist)
```

```
use clear
[]
```

0.0.9 9. Tuples in Python

A tuple is a collection of objects. It contains the following features: - It is ordered - It is NOT changeable - It allows duplicate members - Smaller and faster than a list

Create a Tuple

[76]:
```
#Create a tuple. Use parenthesis
workdays = ("Monday","Tuesday","Wednesday","Thursday","Friday")
print(workdays)
```

```
('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday')
```

Access items in a Tuple

[77]:
```
#Items in a tuple are accessed by index. You can also use slice notation.

#Get items in a Tuple
workdays = ("Monday","Tuesday","Wednesday","Thursday","Friday")
#Get the first item
print(workdays[0])
#Get the last item
print(workdays[-1])
#Get the items starting at position 1 and ending before position 3
print(workdays[1:3])
#Get the first 3 items print(workdays[:3])
#Get all the items starting at position 2
print(workdays[2:])
```

```
Monday
Friday
('Tuesday', 'Wednesday')
('Wednesday', 'Thursday', 'Friday')
```

Get the count of items in an Tuple

[78]:
```
#Get number of items in a tuple
workdays = ("Monday","Tuesday","Wednesday","Thursday","Friday")
print(len(workdays))
```

```
5
```

Check if an item exists in a Tuple

[79]:
```
workdays = ("Monday","Tuesday","Wednesday","Thursday","Friday")
"Monday" in workdays
```

[79]:
```
True
```

Loop through items in a Tuple

[80]:
```
workdays = ("Monday","Tuesday","Wednesday","Thursday","Friday")
for x in workdays:
    print(x)
```

```
Monday
Tuesday
Wednesday
Thursday
Friday
```

Add items to a Tuple

Tuples are not changeable. To add an item to a Tuple you must create a new Tuple.

[81]:
```
#To add items to a tuple you must create a new Tuple
workdays = ("Monday","Tuesday","Wednesday","Thursday","Friday")
weekend = ("Saturday","Sunday")
daysofweek = workdays + weekend
print(daysofweek)
```

```
('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday')
```

0.0.10 10. Features of a Dictionary

A dictionary is a collection of objects. It contains the following features:

- It is unordered
- It is changeable
- It is indexed
- Uses key value pairs

Create a Dictionary

[82]:
```
#Create a Dictionary
car =  {
        "Make":"Honda",
        "Model":"Civic",
        "Color":"Black"
        }
print(car)
```

Values

Keys

```
{'Make': 'Honda', 'Model': 'Civic', 'Color': 'Black'}
```

Accessing items in a Dictionary

[83]:
```
car =  {"Make":"Honda",
        "Model":"Civic",
        "Color":"Black"
        }
#Use Key to access item
print(car["Make"])
#Use get method
print(car.get("Make"))
```

```
Honda
Honda
```

Change a value in a Dictionary

[84]:
```
car =  {"Make":"Honda",
        "Model":"Civic",
        "Color":"Black"
        }
#Change a value in a dictionary
car["Color"] = "Red"
print(car)
```

```
{'Make': 'Honda', 'Model': 'Civic', 'Color': 'Red'}
```

Loop through a Dictionary and access the keys in the Dictionary

[85]:
```
car =  {"Make":"Honda",
        "Model":"Civic",
        "Color":"Black"
        }
#print keys
for x in car:
    print(x)
```

```
Make
Model
Color
```

Loop through a Dictionary and access the values in the Dictionary

[86]:
```
#print values
car =  {"Make":"Honda",
        "Model":"Civic",
        "Color":"Black"
        }
for x in car:
    print(car[x])
```

```
Honda
Civic
Black
```

Loop through a Dictionary and access the values and keys in the Dictionary

[87]:
```
#print both keys and values
car =  {"Make":"Honda",
        "Model":"Civic",
        "Color":"Black"
        }
for key,value in car.items():
    print (key, value)
```

```
Make Honda
Model Civic
Color Black
```

Check if a value exists in a Dictionary

[88]:
```
car =  {"Make":"Honda",
        "Model":"Civic",
        "Color":"Black"
        }
print("Make" in car)
```

```
True
```

Get the length of a Dictionary

[89]:
```
car =  {"Make":"Honda",
        "Model":"Civic",
        "Color":"Black"
       }
print(len(car))
```

```
3
```

Remove an item from a Dictionary using a key

[90]:
```
car =  {"Make":"Honda",
        "Model":"Civic",
        "Color":"Black",
        "Year":2019
       }
#removes item at specified key
car.pop("Year")
print(car)
```

```
{'Make': 'Honda', 'Model': 'Civic', 'Color': 'Black'}
```

Clear all items from a Dictionary.

[91]:
```
car =  {"Make":"Honda",
        "Model":"Civic",
        "Color":"Black",
        "Year":2019
       }
car.clear()
print(car)
```

```
{}
```

0.0.11 11. Sets in Python

A set is a collection of objects. It contains the following features:

- It is unordered
- The items in the set are unchangeable but the set is changeable
- It is unindexed
- Doesn't allow duplicates

duplicate(?): a set cannot contain more than one occurrence of the same element. In other words, each element in a set must be unique.

[92]:
```
#Create a Set
workdays = {"Monday","Tuesday","Wednesday","Thursday","Friday"}
print(workdays)
```

```
{'Monday', 'Thursday', 'Wednesday', 'Tuesday', 'Friday'}
```

[93]:
```
# Use a for loop to loop through a set.
workdays = {"Monday","Tuesday","Wednesday","Thursday","Friday"}
for x in workdays:
    print(x)
```

```
Monday
Thursday
Wednesday
Tuesday
Friday
```

[94]:
```
# Use the add method to add an item to a set.
workdays = {"Monday","Tuesday","Wednesday","Thursday","Friday"}
workdays.add("Saturday")
print(workdays)
```

```
{'Monday', 'Thursday', 'Wednesday', 'Tuesday', 'Saturday', 'Friday'}
```

[95]:
```
# To remove an item from a Set use the remove() method
workdays = {"Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"}
workdays.remove("Saturday")
print(workdays)
```

```
{'Monday', 'Thursday', 'Wednesday', 'Tuesday', 'Friday'}
```

[96]:
```
#Combine 2 sets. Keep only unique values
workdays = {"Monday","Tuesday","Wednesday","Thursday"}
workdays2 = {"Wednesday","Thursday","Friday"}
allworkdays = workdays|workdays2
print(allworkdays)
```

```
{'Monday', 'Thursday', 'Wednesday', 'Friday', 'Tuesday'}
```

[97]:
```
#Combine 2 sets. Keep only common values
workdays = {"Monday","Tuesday","Wednesday","Thursday"}
workdays2 = {"Wednesday","Thursday","Friday"}
allworkdays = workdays&workdays2
print(allworkdays)
```

```
{'Thursday', 'Wednesday'}
```

[98]:
```
#Remove items in the first list that are present in the second list
workdays = {"Monday","Tuesday","Wednesday","Thursday"}
workdays2 = {"Wednesday","Thursday","Friday"}
allworkdays = workdays - workdays2
print(allworkdays)
```

```
{'Monday', 'Tuesday'}
```

0.0.12 11. Files in Python

Opening Files in Python

[100]:
```
f = open("test.txt")     # open file in current directory
f = open("/home/allouche/Enseignements/LMD/M2-ML/2023-2024/Tutorials/test.txt")␣
↪ # specifying full path
```

We can specify the mode while opening a file. In mode, we specify whether we want to read ***r***, write ***w*** or append ***a*** to the file. We can also specify if we want to open the file in text mode or binary mode.

[101]:
```
f = open("test.txt")       # equivalent to 'r' or 'rt'
f = open("test1.txt",'w')   # write in text mode
f = open("img.jpg",'r+b') # read and write in binary mode
```

[103]:
```
! ls test*.*
```

```
test1.txt  test.txt
```

The default encoding is platform dependent. In windows, it is cp1252 but utf-8 in Linux.

[104]:
```
f = open("test.txt", mode='r', encoding='utf-8')
```

Closing Files in Python

[105]:
```
f = open("test.txt", encoding = 'utf-8')
# perform file operations
f.close()
```

A safer way is to use a try…finally block.

[106]:
```
try:
    f = open("test.txt", encoding = 'utf-8')
   # perform file operations
finally:
    f.close()
```

The best way to close a file is by using the with statement.

[107]:
```
with open("test.txt", encoding = 'utf-8') as f:
   # perform file operations
    a=f.readlines()
    print(a)
```

```
['A 1\n', 'B 2\n', 'C 3\n']
```

Writing to Files in Python

In order to write into a file in Python, we need to open it in write ***w***, append ***a*** or exclusive creation ***x*** mode. Writing a string or sequence of bytes (for binary files) is done using the ***write()*** method.

[108]:
```
with open("test1.txt",'w',encoding = 'utf-8') as f:
    f.write("my first file\n")
    f.write("This file\n\n")
    f.write("contains three lines\n")
```

[109]:
```
! cat test1.txt
```

```
my first file
This file

contains three lines
```

Reading Files in Python

To read a file in Python, we must open the file in reading r mode.

We can use the ***read(size)*** method to read in the size number of data. If the size parameter **is not specified**, it reads and returns **up to the end** of the file.

[110]:
```
f = open("test1.txt",'r',encoding = 'utf-8')
s = f.read(4)     # read the first 4 data
print(s)
s=f.read(4)     # read the next 4 data
print(s)
```

```
s=f.read()     # read in the rest till end of file
print(s)
s=f.read()  # further reading returns empty sting
print("Laste read : ", s)
f.close()
```

```
my f
irst
 file
This file

contains three lines

Laste read :
```

we can use the readline() method to read individual lines of a file.

the readlines() method returns a list of remaining lines of the entire file.

[111]:
```
f = open("test1.txt",'r',encoding = 'utf-8')
sall =f.readlines()
print(sall)
print("\nLine by line")
print("=============")
for s in sall:
    print(s)

print("\nWith spliting in words")
print("======================")
for s in sall:
    sl=s.split() # you can split
    #print(sl)
    for x in sl:
        print(x)
f.close()
```

```
['my first file\n', 'This file\n', '\n', 'contains three lines\n']

Line by line
=============
my first file

This file



contains three lines
```

```
With spliting in words
======================
my
first
file
This
file
contains
three
lines
```

0.0.13 12. Dealing with dates and time

The DateTime module is useful for handling dates in Python.

Get Current Date

[112]:
```python
from datetime import date
#Get Current Date
today = date.today()
print(today)
```

```
2023-11-10
```

Get Current Date and Time

[113]:
```python
from datetime import datetime
#Get Current Date and Time
now = datetime.now()
print(now)
```

```
2023-11-10 17:23:07.096490
```

[114]:
```python
from datetime import datetime
now = datetime.now() # current date and time
year = now.strftime("%Y")
print("year:", year)

month = now.strftime("%m")
print("month:", month)

day = now.strftime("%d")
print("day:", day)

time = now.strftime("%H:%M:%S")
print("time:", time)
```

```
year: 2023
month: 11
day: 10
time: 17:23:19
```

Create and set a Date

[115]:
```python
from datetime import date
#Get Current Date
mydate = date(2019,2,8)
print(mydate)
```

```
2019-02-08
```

Create and set a Date and Time

[116]:
```python
from datetime import datetime
#Get Current Date and Time
now = datetime(2019,2,8,18,30,0)
print(now)
print(datetime.now())
```

```
2019-02-08 18:30:00
2023-11-10 17:23:32.553005
```

0.0.14 13. Creating and using functions

A function is a block of code which only runs when it is called.

A function can take in data and return data.

Write and call a simple Function in Python

[118]:
```python
#definition of Hello world Function
def HelloWorld():
    print("Hello World. It it is my first function in python")

#call the HelloWord function
HelloWorld()
```

```
Hello World. It it is my first function in python
```

Write a function that takes a parameter that has a default value

[119]:
```python
#Hello Function with a parameter
def Hello(name = "BOB"):
    print("Hello " + name)

#Call and return default parameter
Hello()
```

```
#Call and return parameter
Hello("John")
```

```
Hello BOB
Hello John
```

Return a value in an Function

[120]:
```
#Return a value in a Function
def add(value1, value2):
    return value1 + value2

answer = add(4,5)
print(answer)
```

```
9
```

Return 2 values in an Function

[147]:
```
#Return a value in a Function
def addSub(value1, value2):
    ''' Function to make sum
    '''
    return value1 + value2, value1 - value2

vsum, vsub = addSub(4,5)
print("Sum=",vsum, "Sub=",vsub)
```

```
Sum= 9 Sub= -1
```

[148]:
```
print(addSub.__doc__)
```

```
 Function to make sum
```

Lambda Functions

Anonymous functions in Python are called Lambda functions. Typical functions are in Python are defined using the def keyword. Anonymous functions are defined using the lambda keyword.

[122]:
```
addFive = lambda x:x+5
addFive(10)
```

[122]: 15

Lambda functions are typically used with the built in functions map() and filter(). The map() function can be used to apply the lambda function to every item on an iterable. (list, tuple, etc..).

[126]:
```
myvalues = [1,2,3,4,5,6,7,8,9,10]
#create a new list by adding 5 to all items in the list
newvalues = list(map(lambda x:x+5,myvalues)) # with map
#newvalues = (map(lambda x:x+5,myvalues)) # with map
```

```
print(newvalues)
```

```
[6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

[127]:
```
myvalues = [1,2,3,4,5,6,7,8,9,10]
#create a new list with only even numbers
newvalues = list(filter(lambda x:(x%2 == 0),myvalues)) # with filter
print(newvalues)
```

```
[2, 4, 6, 8, 10]
```

[128]:
```
myvalues = [1,2,3,4,5,6,7,8,9,10]
#create a new list with only even numbers
newvalues = list(map(lambda x:(x%2 == 0),myvalues)) # with filter
print(newvalues)
```

```
[False, True, False, True, False, True, False, True, False, True]
```

0.0.15 14. List Comprehensions

List comprehensions are an alternative to using for loops and lambda functions. They are an elegant and concise way to define and create lists based on existing list or string.

List with loop

[129]:
```
newnums =[]
nums = [0,1,2,3,4,5,6,7,8,9,10]
for x in nums:
    newnums.append(x + 5)
print(newnums)
```

```
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

List Comprehension

[130]:
```
nums = [0,1,2,3,4,5,6,7,8,9,10]
newnums = [x + 5 for x in nums]
print(newnums)
```

```
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

List comprehension with a filter

[131]:
```
nums = [0,1,2,3,4,5,6,7,8,9,10]
#add 5 only to even numbers
newnums = [x + 5 for x in nums if x%2==0]
print(newnums)
```

```
[5, 7, 9, 11, 13, 15]
```

0.0.16 15. Classes in Python

Python is an object oriented language. Learning to create objects is key to mastering Python. Classes define objects.

Create a basic class in Python

[132]:
```
#Class with a Property
class Car():
    model = "BMW"
```

Create an Object from a Class

[133]:
```
#Instantiate a Object from a class
mycar = Car()
print(mycar.model)
```

```
BMW
```

Assign values to Class Properties in Python

[134]:
```
class Car():
    def __init__(self,make,model,year,color):
        self.make = make
        self.model = model
        self.year = year
        self.color = color

mycar = Car("BMW","X5",2018,"Black")
```

Create a Class with a Method in Python

[145]:
```
class Car():
    '''
    My Classe Car
    '''
    def __init__(self,make,model,year,color):
        '''
        init function (constructor)
        '''
        self.make = make
        self.model = model
        self.year = year
        self.color = color

    def outputvalues(self):
        '''
        my outputvalues
        '''
        #returns all parameter values as a tuple
```

```
        vals = (self.make, self.model, self.year, self.color)
        return vals

mycar = Car("BMW","X5",2018,"Black")
print(mycar.outputvalues())
```

```
('BMW', 'X5', 2018, 'Black')
```

[146]:
```
print(Car.__doc__)
print(Car.outputvalues.__doc__)
print(Car.__init__.__doc__)
```

```
    My Classe Car


        my outputvalues


        init function (constructor)
```

Class with private attribute

[149]:
```
class Car():

    def __init__(self,make,model,year,color):
        '''
        init function (constructor)
        '''
        self._make = make
        self._model = model
        self._year = year
        self._color = color

    def outputvalues(self):
        #returns all parameter values as a tuple
        vals = (self.make, self.model, self.year, self.color)
        return vals

    @property
    def make(self):
        return self._make
    @property
    def model(self):
        return self._model
    @property
    def year(self):
        return self._year
```

```
    @property
    def color(self):
        return self._color

mycar = Car("BMW","X5",2018,"Black")
print(mycar.outputvalues())
mycar._make="HHHH" # _make is not realy private!
print(mycar._make)
```

```
('BMW', 'X5', 2018, 'Black')
HHHH
```

[152]:
```
mycar._make =10
```

[153]:
```
print(mycar.outputvalues())
```

```
(10, 'X5', 2018, 'Black')
```

[ ]:
```

```