



Python:
Data structures, control flow,
OO Programming, Regular
Expressions, System Programming

AAA-Python Edition



Plan

- 1- if / else , For, While
- 2- Lists, Tuples, List comprehensions
- 3- Dictionaries
- 4- Sets
- 5- Object Oriented Programming
- 6- Regular Expression
- 7- System Programming



If / else

- These statements are used to **control which** block of code **to execute**:

```
[1] 1 a=3
     2 if a>8:
     3     print("a is greater than 8")
     4 else:
     5     print("a is not greater than 8")
```

☞ a is not greater than 8

If the “condition” is true
(the corresponding expression
Is Evaluated to True), then
the if “clause” is executed
(the if block)

The condition was false, so
The “else” clause was executed

```
[4] 1 a=3
     2 if a>8 :
     3     print("a is greater than 8")
     4 elif a==3:
     5     print("actually, a=3")
     6 else:
     7     print("a is not greater than 8")
```

☞ actually, a=3

The “elif” clause is executed,
If its condition is true



While

- This statement is used to **control how many times** a block of code has to be **executed**:

```
[21] 1 i=j=1
      2 while(i>0):
      3     print("**** execution number "+str(j)+"**** ")
      4     i=float(input("Give a float value for i: "))
      5     print("Last given i =",i)
      6     j=j+1
      7
```

```
↳ **** execution number 1***
   Give a float value for i: 7.3
   Last given i = 7.3
   **** execution number 2***
   Give a float value for i: 0
   Last given i = 0.0
```

While the condition is true, the block code will be executed.

In this loop, the block has been executed 2 times



For

- This statement is used to **execute** a block of code **a certain number of times**

```
[12] 1 for i in range(4):  
      2     print(i)
```

```
↳ 0  
   1  
   2  
   3
```

This “range” function: will generate a range of numbers from “0” to “4-1”

“i” will take its values from the numbers created by the ‘range’ function

This loop will be executed 4 times



2- Lists, Tuples, List comprehensions

List

- A list is a **value** that contains **multiple values**.

```
[25] 1 l1=[1,15,"element",0.3]
      2 l2= list(range(5))
      3 print(l1)
      4 print(l2)
```

```
➞ [1, 15, 'element', 0.3]
   [0, 1, 2, 3, 4]
```

l1 and l2 are lists

Function "list" to create a list

Tuple

A a tuple is a **list** of **immutable values**.

```
[29] 1 t1=("here",5)
      2 t2 =tuple(range(3))
      3 print(t1)
      4 print(t2)
```

t1 and t2 are tuples

```
➞ ('here', 5)
   (0, 1, 2)
```



2- Lists, Tuples, List comprehensions

Lists and tuples (suite)

Modifying the value of the
element of index 0 (first element)

```
1 l1=[2,5,9]
2 l1[0]="first"
3 print("l1=",l1)
4 print("t1[0]=",t1[0])
5 t1[0]=5
```

Access to the first element

```
l1= ['first', 5, 9]
t1[0]= here
```

TypeError

Traceback (most recent call last)

```
<ipython-input-37-820a4679b5dc> in <module>()
      3 print("l1=",l1)
      4 print("t1[0]=",t1[0])
----> 5 t1[0]=5
```

TypeError: 'tuple' object does not support item assignment

[SEARCH STACK OVERFLOW](#)

Trying to modify the value of an element of a tuple



2- Lists, Tuples, List comprehensions

Some operations with lists

```
[60] 1 l1=list(range(-5,2))
      2 print("l1=",l1)
      3
      4 l2=list(range(7,20,3))
      5 print("l2= ",l2)
      6
      7 l3=l1[2:4]
      8 print("l3= ",l3)
      9
     10 del(l3[0]);print ("l3=",l3)
     11
     12 l4=l3+[10,11]; print("l4=",l4)
     13
     14 l5=7*[2];print("l5=",l5); print("l5 has",str(len(l5))+ " elements")
     15
     16
```

a slice: values
From index 2 to index 4

Concatenating two
lists

Number of elements
of a list

```
l1= [-5, -4, -3, -2, -1, 0, 1]
l2= [7, 10, 13, 16, 19]
l3= [-3, -2]
l3= [-2]
l4= [-2, 10, 11]
l5= [2, 2, 2, 2, 2, 2, 2]
l5 has 7 elements
```




2- Lists, Tuples, List comprehensions

Some operations with lists (suite)

```
[70] 1 l1= list("ABC")
      2 for i in l1:
      3     print(i)
      4
      5 for i in range(len(l1)):
      6     print(str(i)+"- "+l1[i])
      7
      8 if 'G' not in l1:
      9     print ("G is not in l1")
     10
```

Iterate through list

"not" with "in"

Functions "min" and "max"

```
☞ A
   B
   C
0- A
1- B
2- C
G is not in l1
```

Affecting list values
To multiple variables

```
[20] 1 l1=list(range(2,10,5))
      2 l2=list(range(5,25,9))
      3 print(l1);print(l2)
      4 print("The greatest value in l1=",max(l1))
      5 print("The smallest value in l2",min(l2))
      6 x,y=l1
      7 print(x,y)
      8
      9
```

```
☞ [2, 7]
   [5, 14, 23]
   The greatest value in l1= 7
   The smallest value in l2 5
   2 7
```



2- Lists, Tuples, List comprehensions

List comprehensions

Filtering elements

```
[22] 1 l1=[x for x in range(4) if x!= 2]
      2 print (l1)
      3 l2=[x**2 for x in [1,2,3]]
      4 print(l2)
      5 l3=[[x,y,z] for x in range(3) for y in ("A") for z in ["el1","el2"]]
      6 print(l3)
      7
      8
```

Creating new elements from range

List of lists using 3 loops

```
↳ [0, 1, 3]
   [1, 4, 9]
   [[0, 'A', 'el1'], [0, 'A', 'el2'], [1, 'A', 'el1'], [1, 'A', 'el2'], [2, 'A', 'el1'], [2, 'A', 'el2']]
```



2- Lists, Tuples, List comprehensions

List methods

Finding an element in a list

- A list has some methods. We will talk about methods later.

```
[33] 1 l1=list("LETTERS")  
      2 print(l1.index("R"))
```

```
↳ 5
```

Add an element to the
end of a list

```
1 l1.append("G")  
2 print(l1)
```

```
↳ ['L', 'E', 'T', 'T', 'E', 'R', 'S', 'G']
```

```
[35] 1 l1.insert(5, "H")  
      2 print(l1)
```

Insert an element at a certain position

```
↳ ['L', 'E', 'T', 'T', 'E', 'H', 'R', 'S', 'G']
```

Remove an element
from a list

```
[36] 1 l1.remove("T")  
      2 print(l1)
```

```
↳ ['L', 'E', 'T', 'E', 'H', 'R', 'S', 'G']
```



3- Dictionaries

Dictionaries

- A **dictionary** is a list of values with corresponding **keys**

```
[41] 1 d1={"Name":"bob","Age":36}
      2 print(d1)
      3 d2={1:"First",2:"Second"}
      4 print(d2)
```

key

Value

```
☞ {'Name': 'bob', 'Age': 36}
   {1: 'First', 2: 'Second'}
```

Method that returns
the dictionary values

```
[47] 1 for i in d2.values():
      2     print(i)
```

```
☞ First
   Second
```

Method that returns
the dictionary keys

```
[48] 1 for i in d2.keys():
      2     print(i)
```

```
☞ 1
   2
```



3- Dictionaries

Dictionaries (suite)

```
1 for i in d2.items():  
2     print(i)  
3     k,l=i  
4     print(k,l)
```

```
➤ (1, 'First')  
1 First  
(2, 'Second')  
2 Second
```

The key doesn't exist
so a default value is given

A key is created with a
default value

The key already exists,
So no other key is created

Method that returns the
dictionary items: pairs of key,value

```
[56] 1 print(d1.get("Name"))  
2     print(d1.get("name","Smith"))  
3
```

```
➤ bob  
Smith
```

The key exists, its value
Is returned

```
[63] 1 d2.setdefault(3,"third")  
2     print(d2)
```

```
➤ {1: 'First', 2: 'Second', 3: 'third'}
```

```
[64] 1 d2.setdefault(3,"other element")  
2     print(d2)
```

```
➤ {1: 'First', 2: 'Second', 3: 'third'}
```



Sets

- A **set** is a list of **distinct** values

Intersection between
s1 and s2

Elements in
S1 and not in
s2

```
1 l1=list(range(2))+list(range(2))
2 s1=set(l1)
3 print("l1=",l1)
4 print("s1=",s1)
5 s1.add(3);print(s1)
6 s2=set(list(range(1,3)))
7 print(s1.isdisjoint(s2))
```

```
l1= [0, 1, 0, 1]
s1= {0, 1}
{0, 1, 3}
False
```

The duplicates
are eliminated

```
[82] 1 print(s1 & s2)
```

```
{1}
```

```
[84] 1 print(s1-s2)
      2 print(s1<=s2)
```

```
{0, 3}
False
```

Is s1 a subset of
s2



5- Object Oriented Programming

Classes

- In Python, we can define “**classes**”: a defined prototype that **encapsulates data** and the **functions** to operate on them.
- An instance of a **class** is called an “**object**”. We **already** used **objects** when we **used lists, sets and dictionaries**.

```
1 #definition of class MyTable
2
3 class MyTable:
4     l1=0
5     def __init__(self,name,length=0):
6         self.length=length
7         self.name=name
8         self.myList=[None for _ in range(length)]
9
10    # to be sure that the lenght represents the actual list length
11    def validL(self):
12        self.length=len(self.myList)
13
14    # print the type of the list
15    def myType(self):
16        print("I am a TABLE 1 ")
17
```

Name of the class

Called when creating an object of that class

To indicate that elements of range are not the elements of the list

Data attribute

A method (a function attribute)



5- Object Oriented Programming

Classes

```
17
18 # insert doesn't accept negative values or values greater than length
19 def insert (self,ind,val):
20     self.validL()
21     if ind >= self.length :
22         print("The given index: "+str(ind)+
23             " exceeds the table length: "+str(self.length))
24     elif ind < 0:
25         print("The given index: "+str(ind)+" is negative")
26     else:
27         self.myList.insert(ind,val)
28         print("The value has been inserted at the index"+str(ind) )
29
30
31 # print myList and the length attribute
32 def printme(self):
33     self.validL()
34     print(self.name+" (" +str(self.length)+"): ",end=" ")
35     for i in self.myList:
36         print (i,end=" ")
37     print(" ")
38
39 # append a new element at the end of the
40 def add (self,val=None):
41     self.validL()
42     self.myList=self.myList +[val]
43     self.length=self.length+1
44
```

A comment

Each time we use
myList, we ensure
that length==
len(myList)

Optional attribute for
function print



5- Object Oriented Programming

Classes

```
47 # MyTable2 inherit MyTable1 functions
48
49 class MyTable2(MyTable):
50     def myType(self):
51         print("I am a TABLE 2 ")
52
53 class MyTable3(MyTable):
54     def myType(self):
55         print("I am a TABLE 3 ")
56
57 print("#####")
58 a= MyTable("table1",3)
59 a.insert(22,"B")
60 a.insert(2,"A")
61 a.printme()
62 a.add(2)
63 a.printme()
64 a.add()
65 a.printme()
66 a.length=8
67 print("a.length=",a.length)
68 a.printme()
69 print("#####")
```

A subclass of class MyTable
Inherits all its methods
and data attributes:
We can use them without
redefining them.

Redefine "myType"
(already defined in MyTable)
It's overriding myType
parent class method

Object creation(
Call of __init__)

Call of a method

Access of a data attribute



5- Object Oriented Programming

Classes

Same class,
different
values

Call of the same
Method had
different results
==
Polymorphism

[By Amina Delali]

```
70 b=MyTable("table2",4)
71 b.printme()
72 print("#####")
73 c=MyTable2("table2",3)
74 c.printme()
75 d=MyTable3("table3")
76 l=[a,b,c,d]
77 for i in l:
78     i.myType()
79
80
```

c an instance
of MyTable2:
Use of `__init__`
method of class
MyTable

An other different subclass

List of 4 instances of different
classes

```
#####
The given index: 22 exceeds the table length: 3
The value has been inserted at the index2
table1 (4):  None None A None
table1 (5):  None None A None 2
table1 (6):  None None A None 2 None
a.length= 8
table1 (6):  None None A None 2 None
#####
table2 (4):  None None None None
#####
table2 (3):  None None None
I am a TABLE 1
I am a TABLE 1
I am a TABLE 2
I am a TABLE 3
```



6- Regular expressions

Regular expressions

A year from 1970 to 2999

- A regular expression is a **description** of **pattern** of **text**

Need of module re

```
[190] 1 import re
      2 #creating a regex pattern object
      3 myReg=re.compile(r"([0-2][1-9]|30|31)-(0[1-9]|1[0-2])-(19[7-9][0-9]|2[0-9]{3})")
      4 myReg2=re.compile(r"[a]+")
      5
      6 print(myReg.findall("It starts from 11-02-2018 and ends at 25-09-2029."))
      7 res=myReg.search("It starts from 11-02-2018* and ends at 25-09-2029.")
      8 print(">"+res.group()+">")
      9 print(myReg.match("It starts from 11-02-2018 and ends at 25-09-2029."))
     10 print(myReg2.findall("a string aa and aaaa"))
     11 print(myReg2.findall("my string"))
     12
```

3 digits

```
❏ [(['11', '02', '2018'), ('25', '09', '2029')]
   *11-02-2018*
   None
   ['a', 'aa', 'a', 'aaaa']
   []
```

Search for the first date

One or more a(+)

A month: composed of:
0 and a digit from 1 to 9 (0[1-9])
Or (|)
1 and a digit from 0 to 2 (1[0-2])

Search a pattern at the start



7- System programming

System Programming

- We will **focus** on system programming in **Colab**.
- Some Python functions can be **simply** done on **Colab**.
- For example the **bash commands**: they can be used as they are by **prefixing** them by **“!”** or **“%”**: !ls, !mkdir, !git, !pip, %cd ... etc

```
1 import subprocess
2 p = subprocess.run(["ls", "-l"])
3 !ls
4 from google.colab import files
5 myFile= files.upload()
6
```

Running 'ls'
using subprocess

Import a local
file

Browse...

Hello.py

Cancel upload

```
[ ] 1 import Hello as h
    2 h.sayHello()
```

After selecting the
Script file, import it

```
➦ Hello.
  Welcome to School Of AI!
```



7- System programming

System Programming

- Second way of using a **user** defined **script**:
- We have to **mount Google Drive**

```
1  
2 # mounting google drive  
3  
4 from google.colab import drive  
5 drive.mount('/content/drive')  
6
```

Mounted at /content/drive

```
[24] 1  
2 import sys  
3  
4 sys.path.append("drive/My Drive/myScripts/")
```

To avoid using the full path of the script, use `sys.path.append`

```
[26] 1 import Quit as me  
2  
3 me.quit()  
4  
5 !python3 "drive/My Drive/myScripts/Quit.py"  
6  
7
```

Using a defined function



Have a nice day!

This is the execution of `Quit.py` as a main script

Running the file as a script



7- System programming

System Programming

Using '!cat' to print the content of the file

```
1 #printing the script content using cat command
2 print("-----")
3 !cat "drive/My Drive/myScripts/Quit.py"
4
5 #printing the script content using path.join an open functions
6 import os
7
8 print("\n-----")
9 myFile=os.path.join("drive","My Drive","myScripts","Quit.py")
10 f=open(myFile,'r')
11 lines=f.readlines()
12 f.close()
13 for l in lines:
14     print(l,end="")
```

Creating the file path

Open and read the file content into a list

Print the list

Use if __name__ == "__main__" for the code
To be executed if the module is not imported
And run as a script

```
-----
def quit():
    print ("Have a nice day!")

if __name__ == "__main__":
    print("This is the execution of Quit.py as a main script\n")
-----
def quit():
    print ("Have a nice day!")

if __name__ == "__main__":
    print("This is the execution of Quit.py as a main script\n")
```



References

- Duchesnay Edouard and Löfstedt Tommy. Statistics and machine learning in python release 0.2. On-line at ftp://ftp.cea.fr/pub/unati/people/educhesnay/pystatml/M1_IMSV/StatisticsMachineLearningPythonDraft.pdf. Accessed on 23-09-2018.
- Python Software Foundation. The python standard library. On-line at <https://docs.python.org/3.6/library/index.html>. Accessed on 23-09-2018.
- Joel Grus. Data science from scratch: first principles with python. O'Reilly Media, Inc, 2015.
- Chandat Sunny. Learn Python in 24 Hours. CreateSpace Independent Publishing Platform, 2016.
- Al Sweigart. Automate the boring stuff with Python: practical programming for total beginners. No Starch Press, 2015.



Thank you!

FOR ALL YOUR TIME