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

AAA-Python Edition

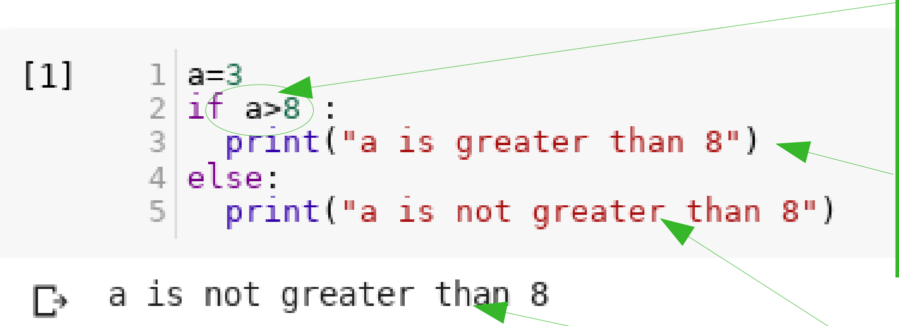
Plan

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

1- 1- if / else , For, While

If / else

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

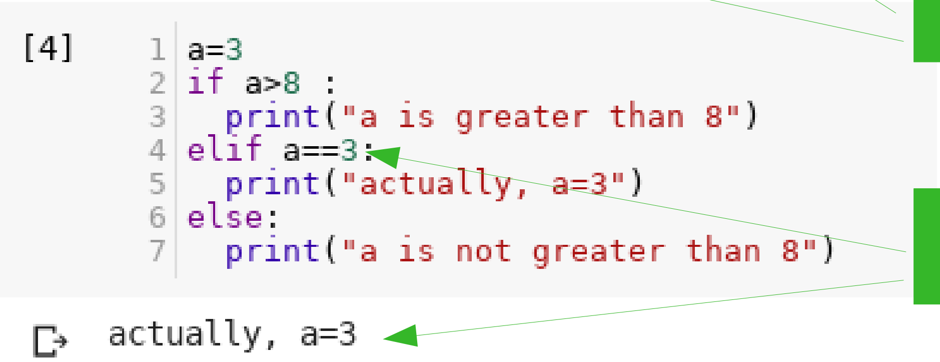


```
[1]
a=3
if  a>8 :
  print("a is greater than 8")
else:
  print("a is not greater than 8")
```

```
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



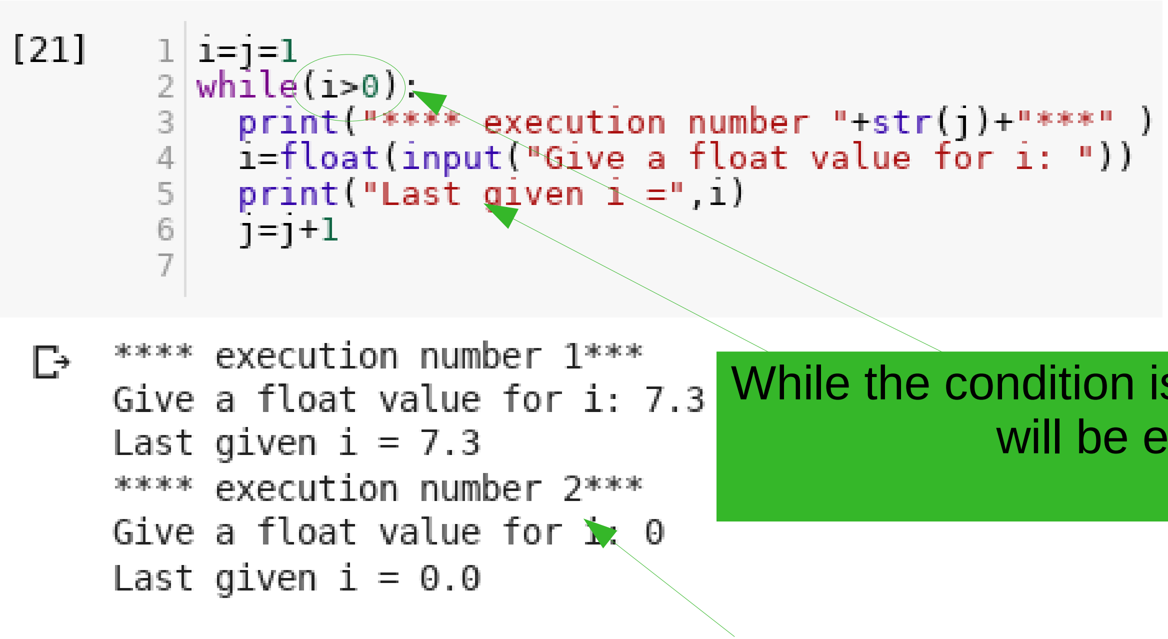
```
[4]
a=3
if  a>8 :
  print("a is greater than 8")
elif  a==3:
  print("actually, a=3")
else:
  print("a is not greater than 8")
```

```
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]
i=j=1
while(i>0):
  print("**** execution number "+str(j)+"***" )
  i=float(input("Give a float value for i: "))
  print("Last given i =",i)
  j=j+1

```

```
**** 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
```

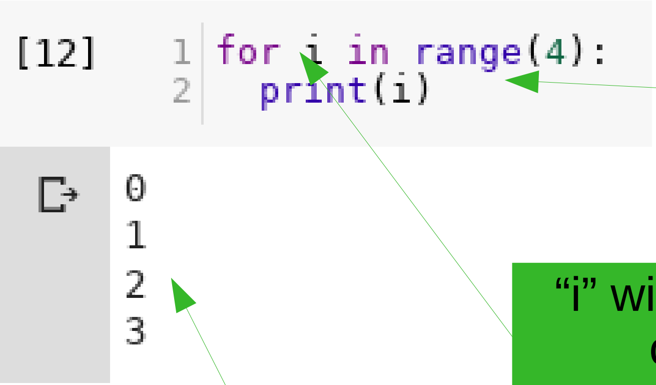
While the condition is true, the block code will be executed.

In this loop, the block has been executed 2 times

1- 1- if / else , For, While

For

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



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

```
0
1
2
3
```

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

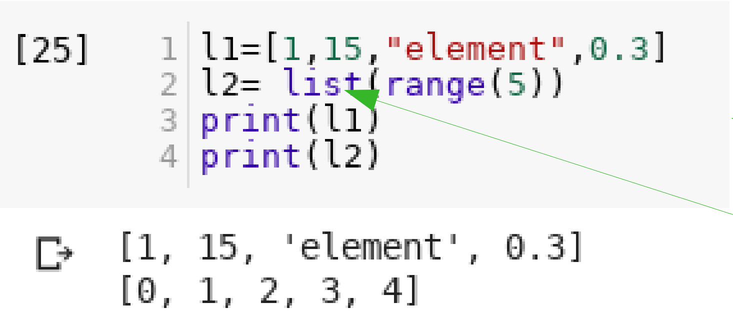
“i” will take its values from the numbers created by the ‘range’ function

This loop will be executed 4 times

2- Lists, Tuples, List comprehensions

List

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



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

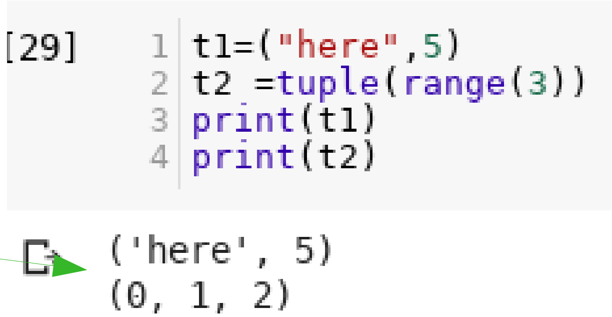
```
[1, 15, 'element', 0.3]
[0, 1, 2, 3, 4]
```

l1 and l2 are lists

Function “list” to create a list

Tuple

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



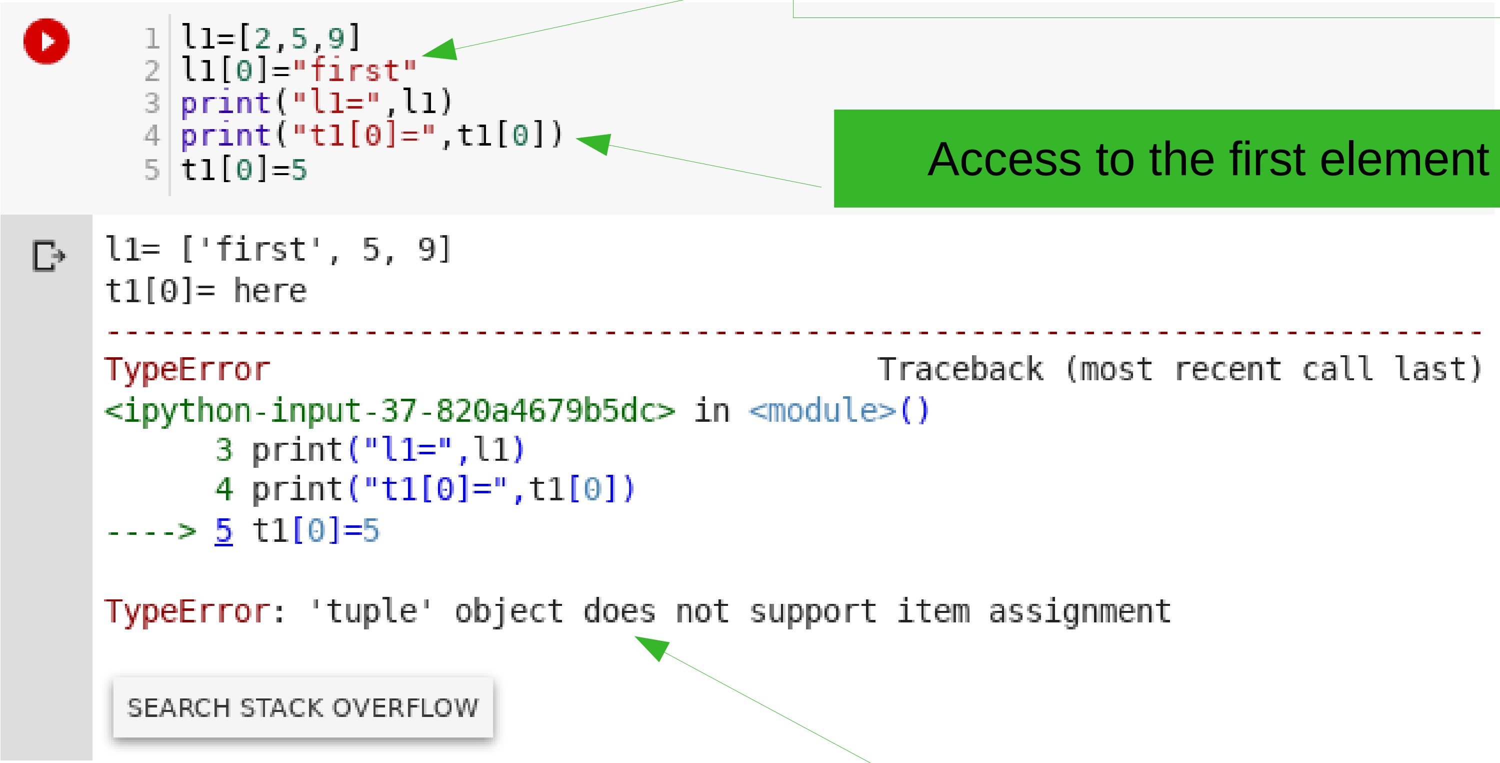
```
[29]  t1=("here",5)
      t2 =tuple(range(3))
      print(t1)
      print(t2)
```

```
('here', 5)
(0, 1, 2)
```

t1 and t2 are tuples

2- Lists, Tuples,
List comprehensions

Lists and tuples (suite)



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

```
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
```

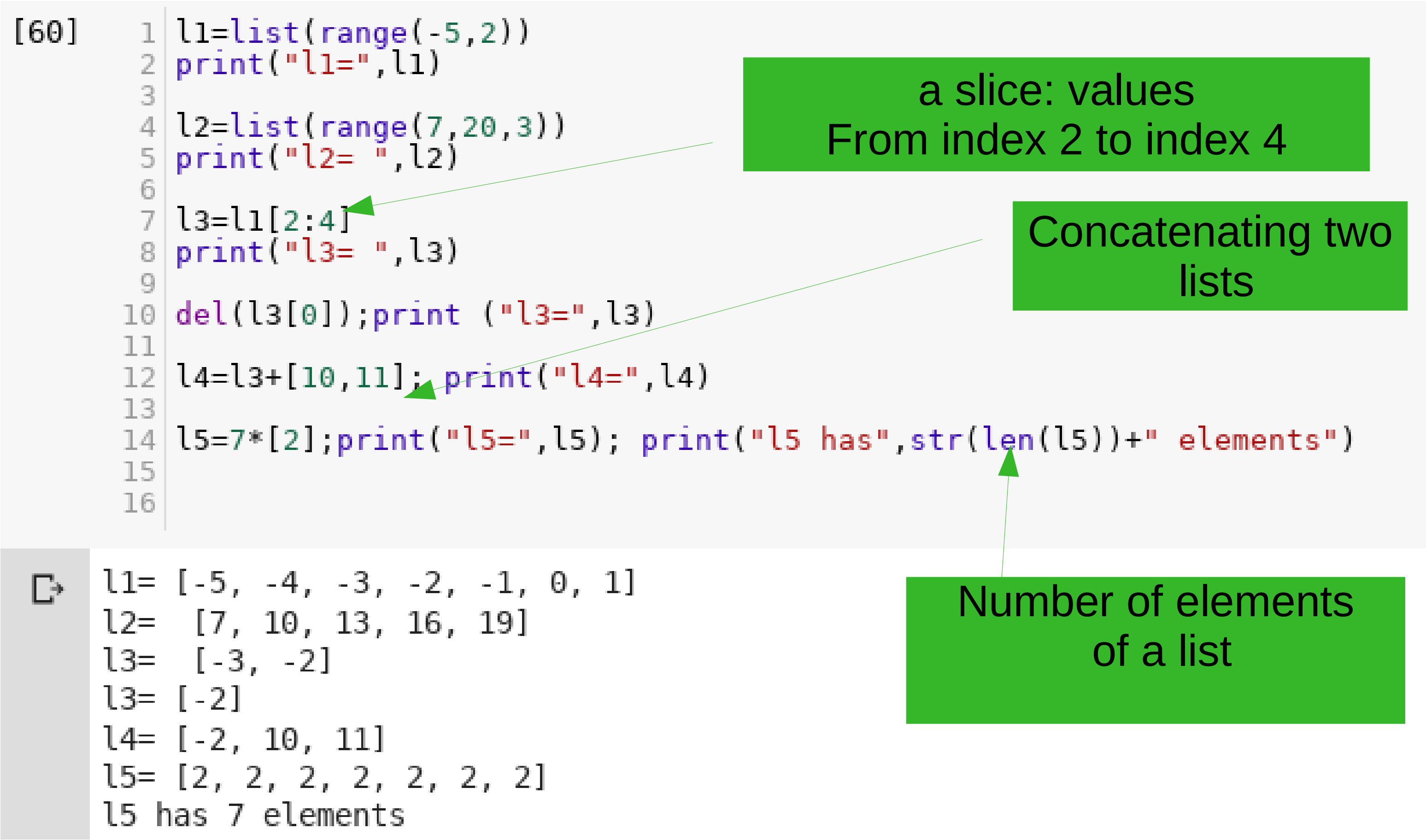
Modifying the value of the element of index 0 (first element)

Access to the first element

Trying to modify the value of an element of a tuple

[By]

Some operations with lists



```
[60]
l1=list(range(-5,2))
print("l1=",l1)

l2=list(range(7,20,3))
print("l2= ",l2)

l3=l1[2:4]
print("l3= ",l3)

del(l3[0]);print ("l3=",l3)

l4=l3+[10,11]; print("l4=",l4)

l5=7*[2];print("l5=",l5); print("l5 has",str(len(l5))+" elements")
```

```
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
```

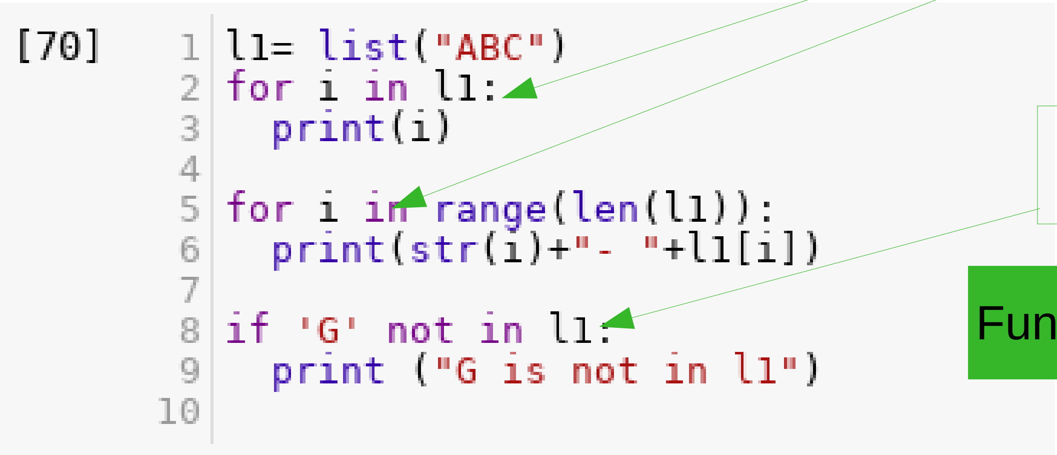
a slice: values
From index 2 to index 4

Concatenating two lists

Number of elements of a list

2- Lists, Tuples,
List comprehensions

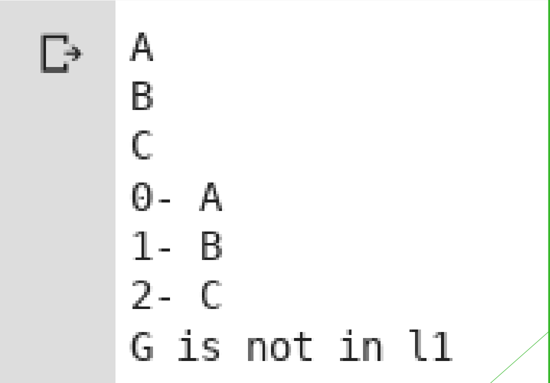
Some operations with lists (suite)



```
[70]
l1= list("ABC")
for i in l1:
  print(i)

for i in range(len(l1)):
  print(str(i)+"- "+l1[i])

if 'G' not in l1:
  print ("G is not in l1")
```

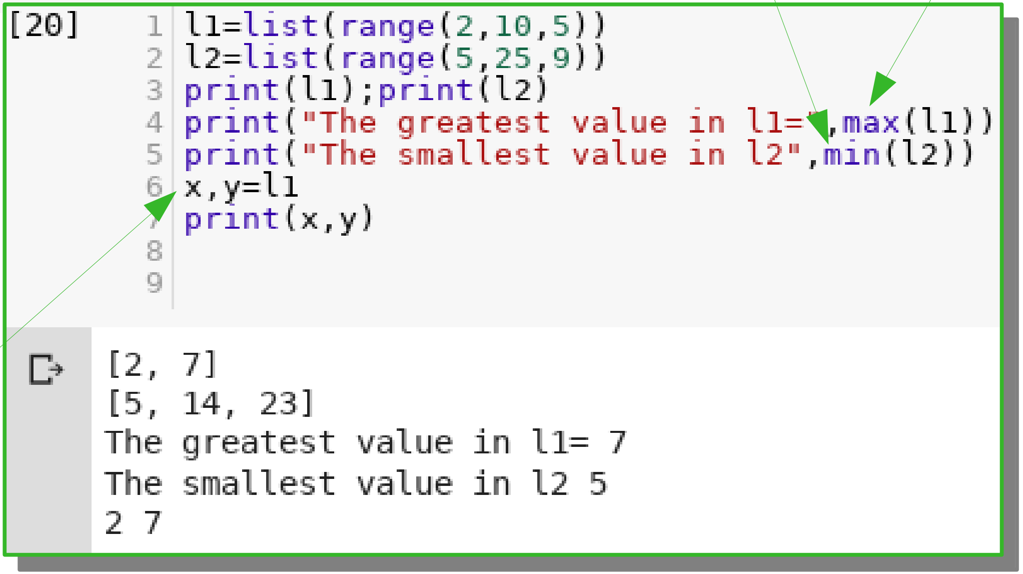


```
A
B
C
0- A
1- B
2- C
G is not in l1
```

Iterate through list

“not” with “in”

Functions “min” and “max”



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

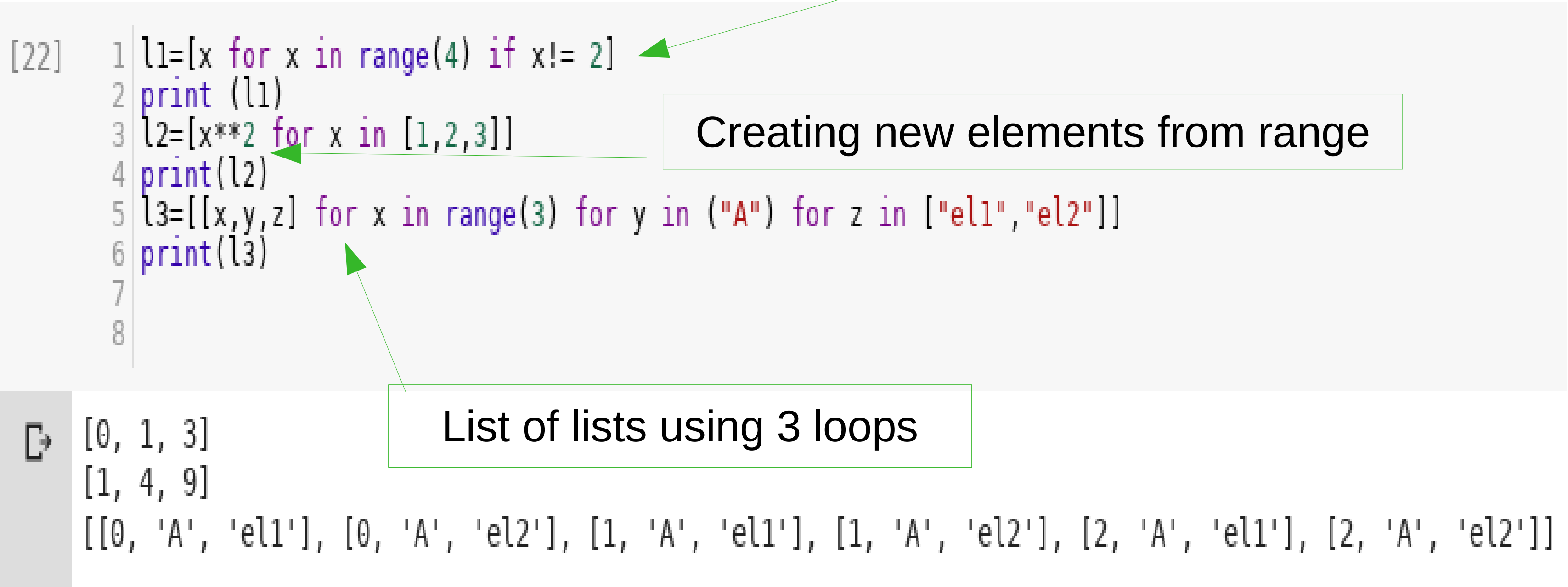
```
[2, 7]
[5, 14, 23]
The greatest value in l1= 7
The smallest value in l2 5
2 7
```

Affecting list values
To multiple variables

[By]

2- Lists, Tuples,
List comprehensions

List comprehensions



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

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

Filtering elements

Creating new elements from range

List of lists using 3 loops

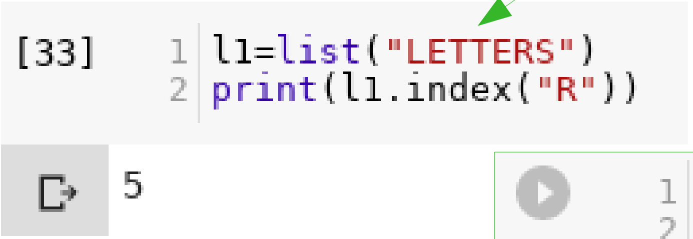
[By]

2- Lists, Tuples, List comprehensions

List methods

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

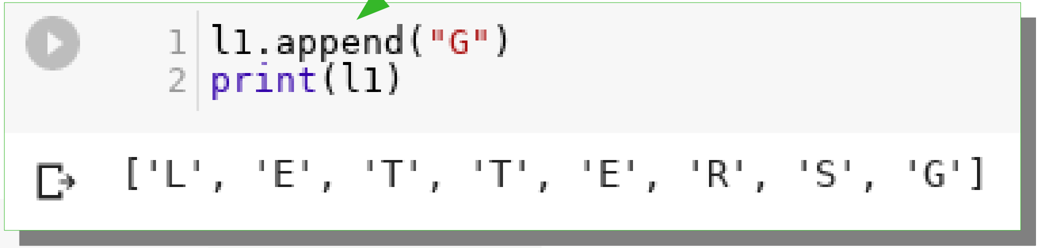
Finding an element in a list



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

5
```

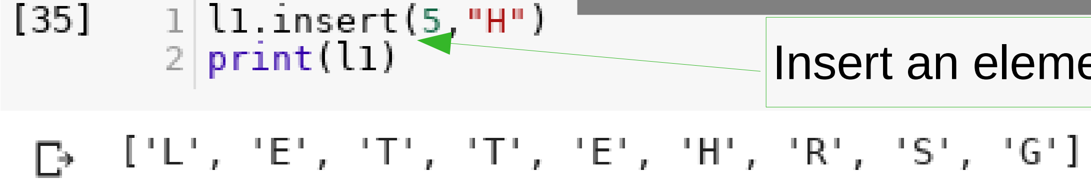
Add an element to the end of a list



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

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

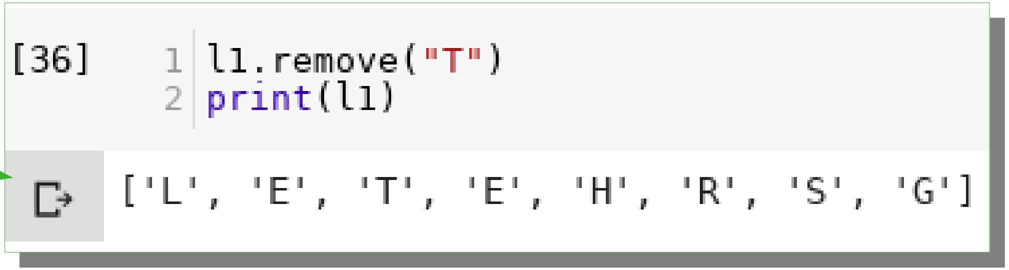
Insert an element at a certain position



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

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

Remove an element from a list



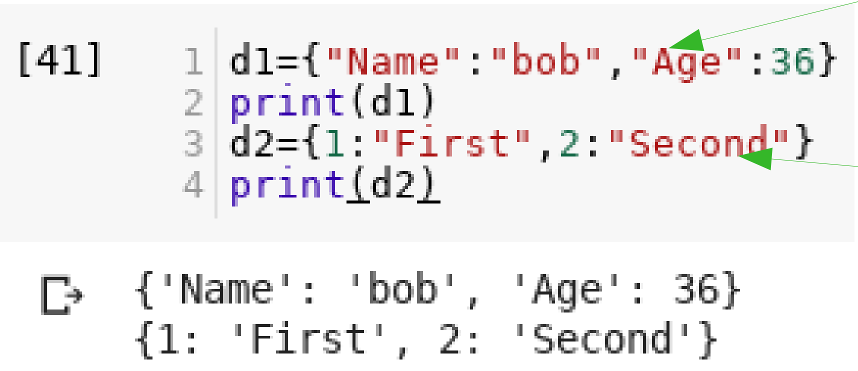
```
[36]  l1.remove("T")
      print(l1)

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

Dictionaries

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

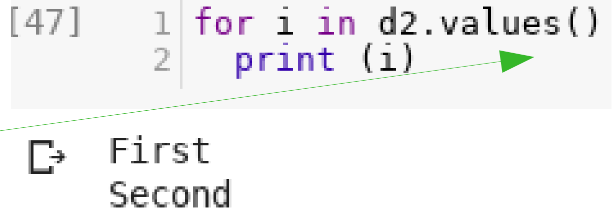
key



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

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

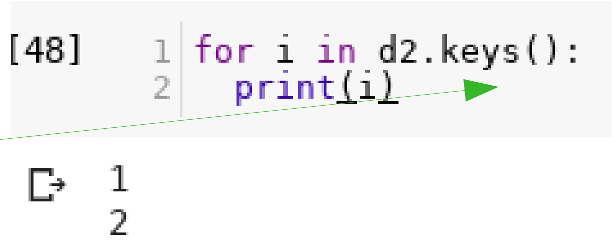
Value



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

```
First
Second
```

Method that returns the dictionary values



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

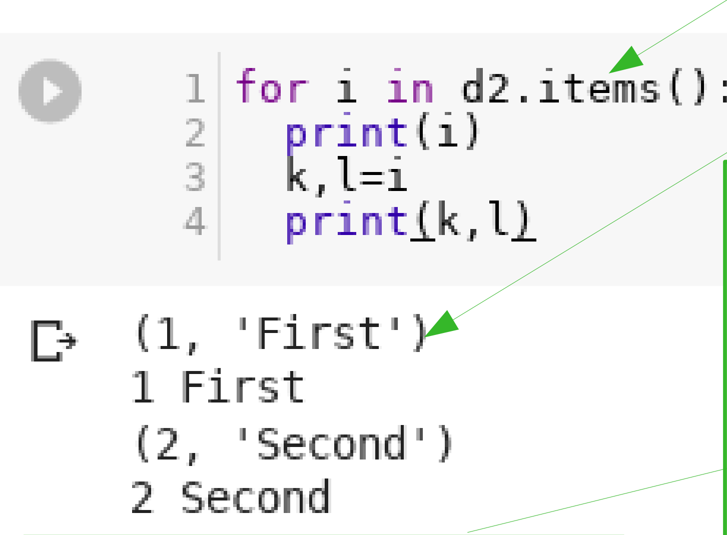
```
1
2
```

Method that returns the dictionary keys

[By]

3- Dictionaries

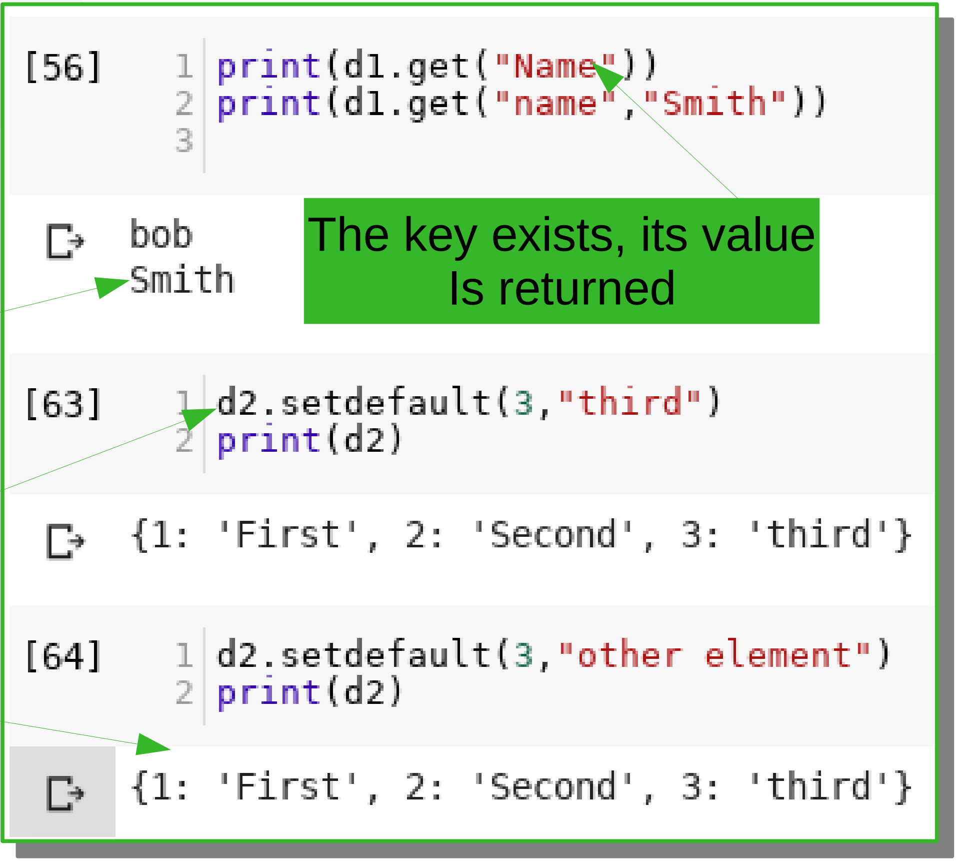
Dictionaries (suite)



```
for i in d2.items():
  print(i)
  k,l=i
  print(k,l)
```

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

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



[56]

```
print(d1.get("Name"))
print(d1.get("name","Smith"))
```

```
bob
Smith
```

The key exists, its value Is returned

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

[63]

```
d2.setdefault(3,"third")
print(d2)
```

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

A key is created with a default value

[64]

```
d2.setdefault(3,"other element")
print(d2)
```

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

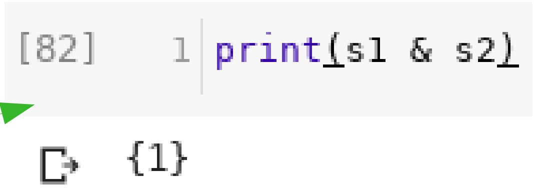
The key already exists, So no other key is created

[By]

Sets

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

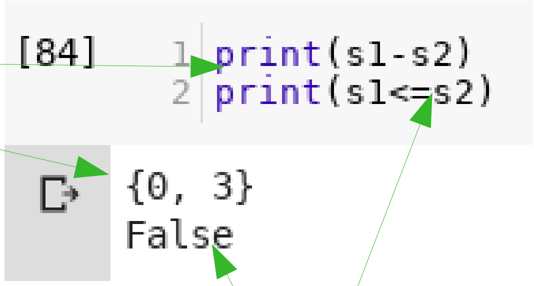
Intersection between s1 and s2



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

```
{1}
```

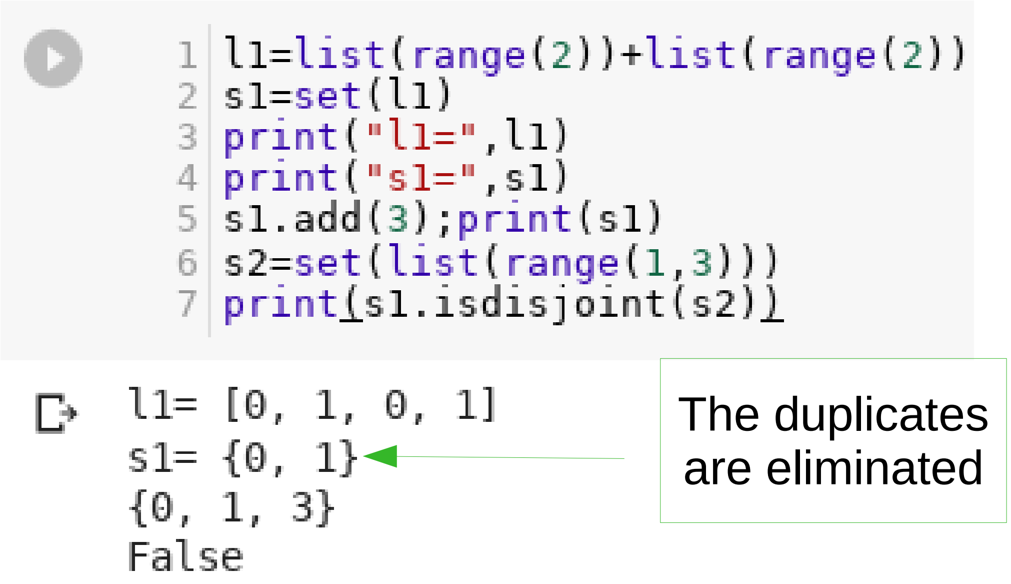
Elements in S1 and not in s2



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

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

Is s1 a subset of s2



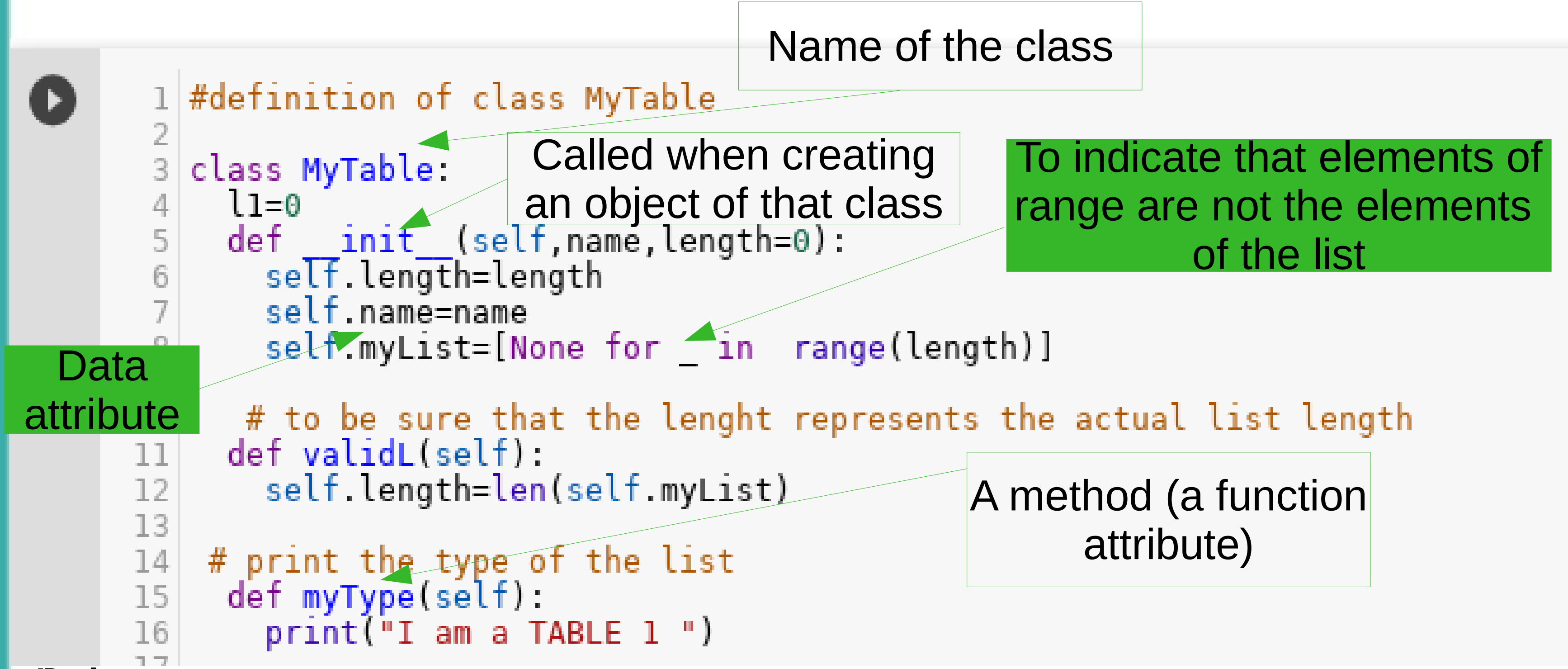
```
l1=list(range(2))+list(range(2))
s1=set(l1)
print("l1=",l1)
print("s1=",s1)
s1.add(3);print(s1)
s2=set(list(range(1,3)))
print(s1.isdisjoint(s2))
```

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

The duplicates are eliminated

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.**



```
#definition of class MyTable

class MyTable:
  l1=0
  def __init__(self,name,length=0):
    self.length=length
    self.name=name
    self.myList=[None for _ in  range(length)]

   # to be sure that the lenght represents the actual list length
  def validL(self):
    self.length=len(self.myList)

 # print the type of the list
  def myType(self):
    print("I am a TABLE 1 ")
```

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

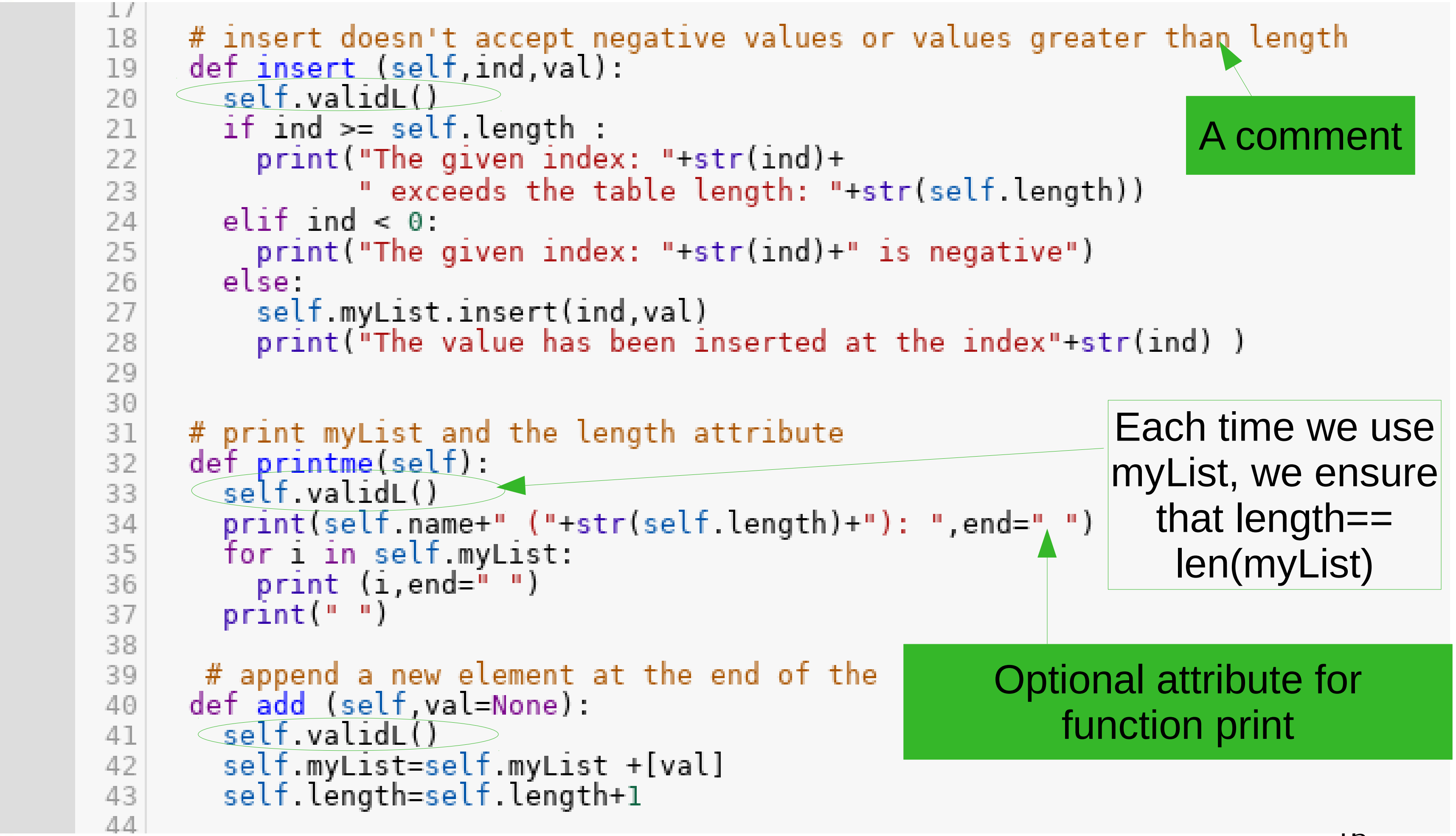
Data attribute

A method (a function attribute)

[By]

5- Object Oriented Programming

Classes



```
  # insert doesn't accept negative values or values greater than length
  def insert (self,ind,val):
    self.validL()
    if ind >= self.length :
      print("The given index: "+str(ind)+
            " exceeds the table length: "+str(self.length))
    elif ind < 0:
      print("The given index: "+str(ind)+" is negative")
    else:
      self.myList.insert(ind,val)
      print("The value has been inserted at the index"+str(ind) )


  # print myList and the length attribute
  def printme(self):
    self.validL()
    print(self.name+" ("+str(self.length)+"): ",end=" ")
    for i in self.myList:
      print (i,end=" ")
    print(" ")

   # append a new element at the end of the
  def add (self,val=None):
    self.validL()
    self.myList=self.myList +[val]
    self.length=self.length+1
```

A comment

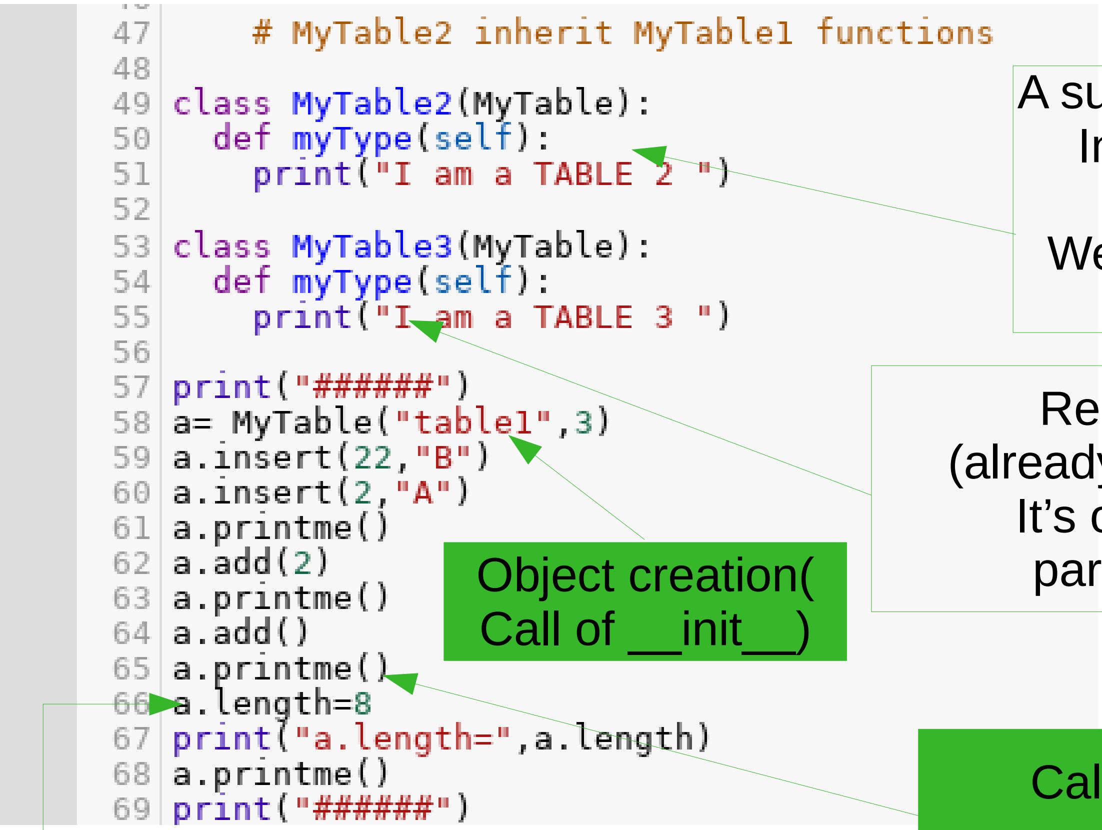
Each time we use myList, we ensure that length== len(myList)

Optional attribute for function print

[By]

5- Object Oriented Programming

Classes



```
    # MyTable2 inherit MyTable1 functions

class MyTable2(MyTable):
  def myType(self):
    print("I am a TABLE 2 ")

class MyTable3(MyTable):
  def myType(self):
    print("I am a TABLE 3 ")

print("######")
a= MyTable("table1",3)
a.insert(22,"B")
a.insert(2,"A")
a.printme()
a.add(2)
a.printme()
a.add()
a.printme()
a.length=8
print("a.length=",a.length)
a.printme()
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__)

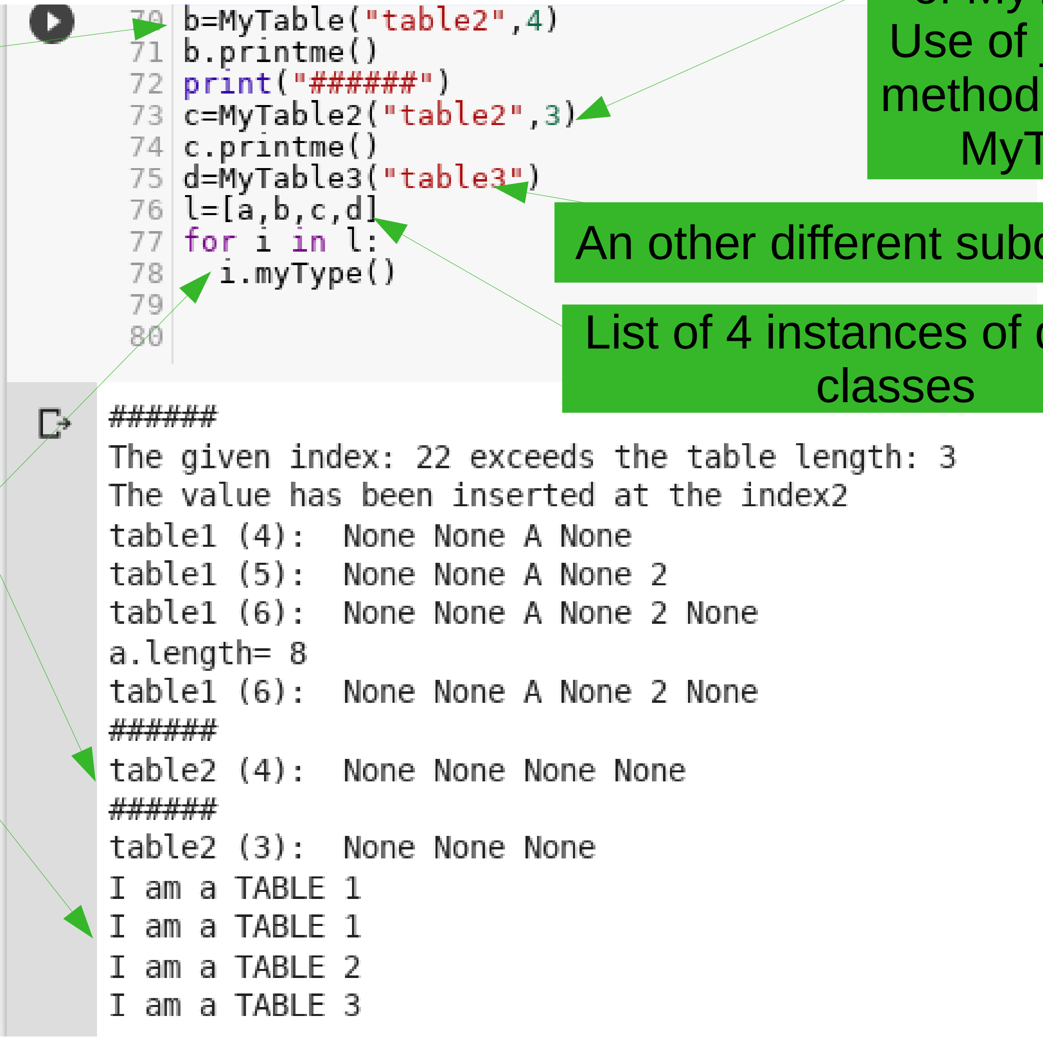
Call of a method

Access of a data attribute

[By]

5- Object Oriented Programming

Classes



```
b=MyTable("table2",4)
b.printme()
print("######")
c=MyTable2("table2",3)
c.printme()
d=MyTable3("table3")
l=[a,b,c,d]
for i in l:
    i.myType()
```

```
######
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
```

Same class, different values

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

An other different subclass

List of 4 instances of different classes

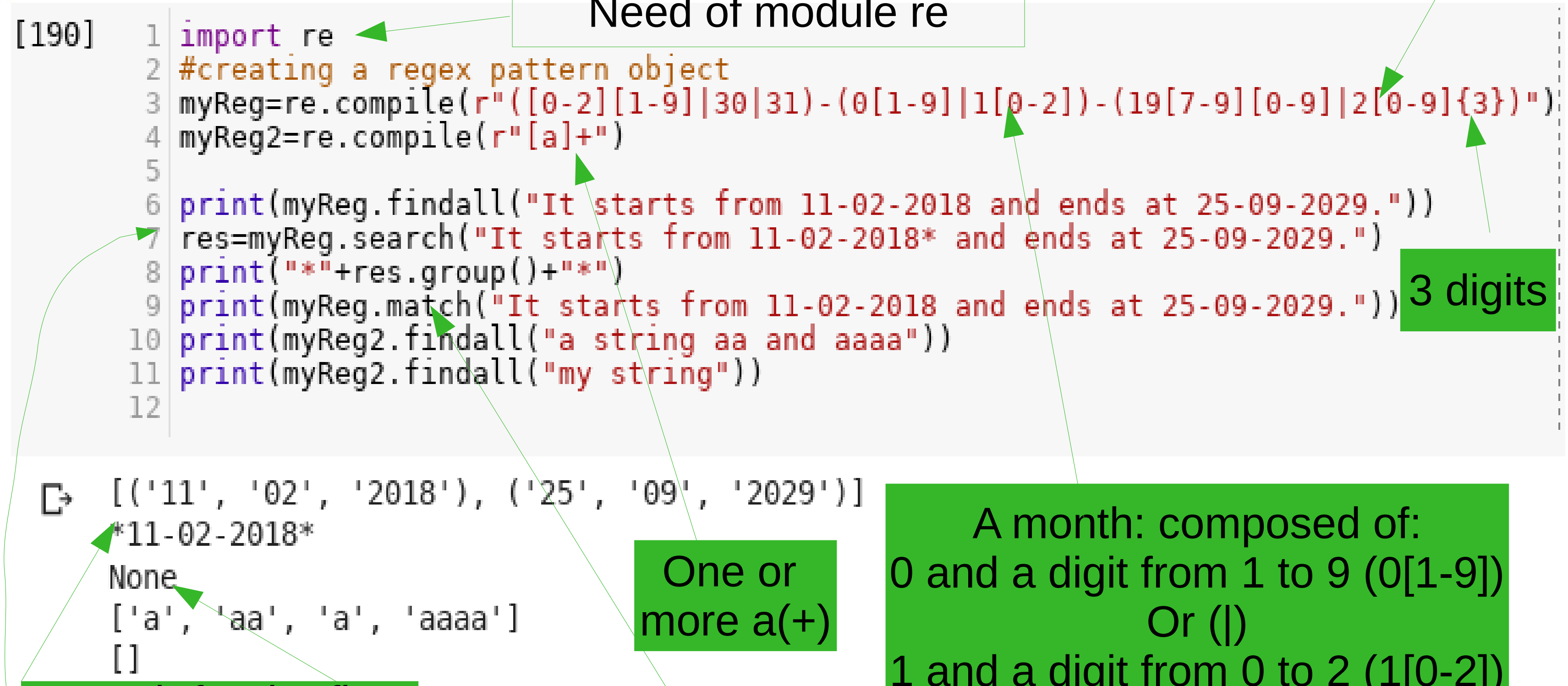
Call of the same Method had different results == Polymorphism

[By]

6- Regular expressions

Regular expressions

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



```
[190]
import re
#creating a regex pattern object
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})")
myReg2=re.compile(r"[a]+")

print(myReg.findall("It starts from 11-02-2018 and ends at 25-09-2029."))
res=myReg.search("It starts from 11-02-2018* and ends at 25-09-2029.")
print("*"+res.group()+"*")
print(myReg.match("It starts from 11-02-2018 and ends at 25-09-2029."))
print(myReg2.findall("a string aa and aaaa"))
print(myReg2.findall("my string"))
```

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

A year from 1970 to 2999

Need of module re

3 digits

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 for the first date

Search a pattern at the start

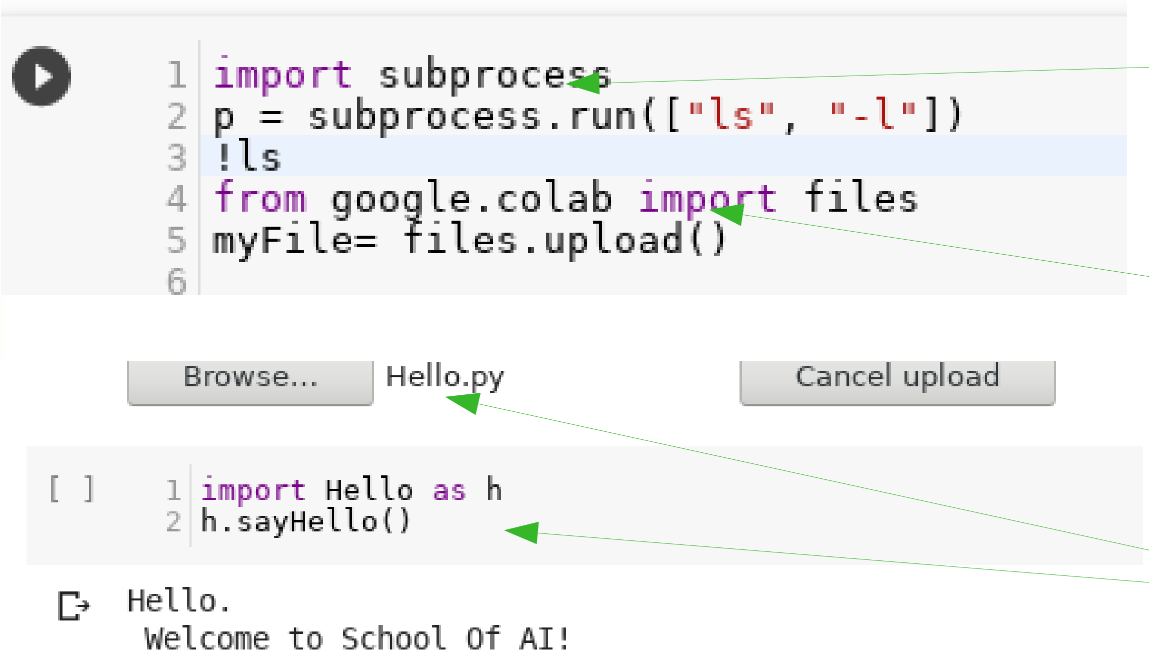
[By]

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



```
import subprocess
p = subprocess.run(["ls", "-l"])
!ls
from google.colab import files
myFile= files.upload()
```

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

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

Running ‘ls’ using subprocess

Import a local file

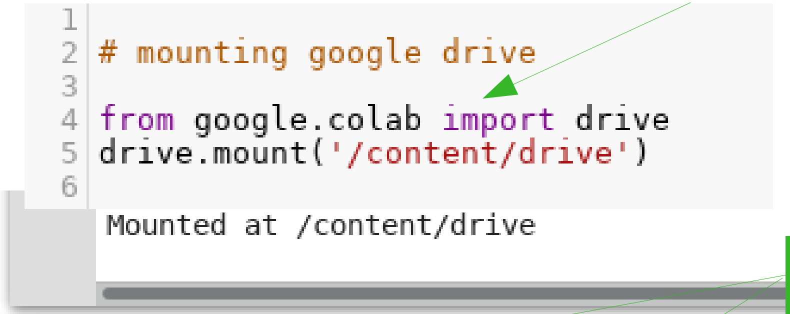
After selecting the Script file, import it

[By]

7- System programming

System Programming

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

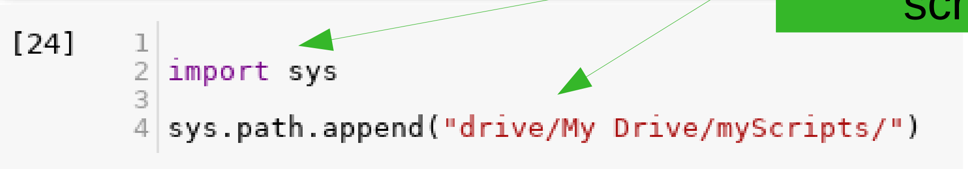


```
# mounting google drive

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

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

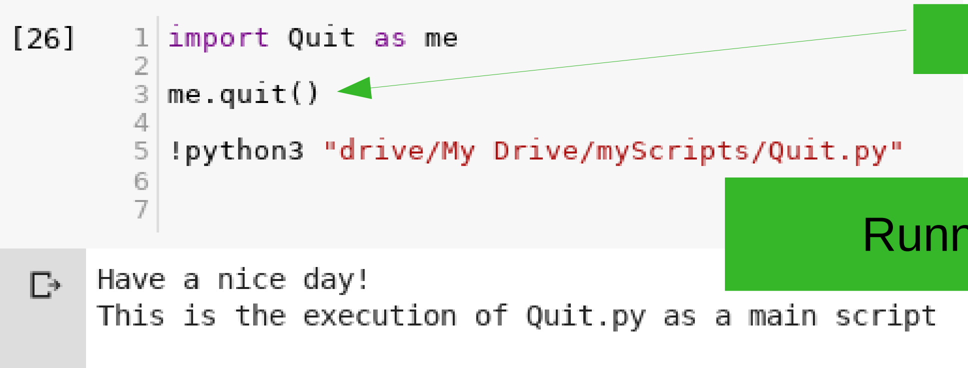


[24]

```
import sys

sys.path.append("drive/My Drive/myScripts/")
```

Using a defined function



[26]

```
import Quit as me

me.quit()

!python3 "drive/My Drive/myScripts/Quit.py"
```

```
Have a nice day!
This is the execution of Quit.py as a main script
```

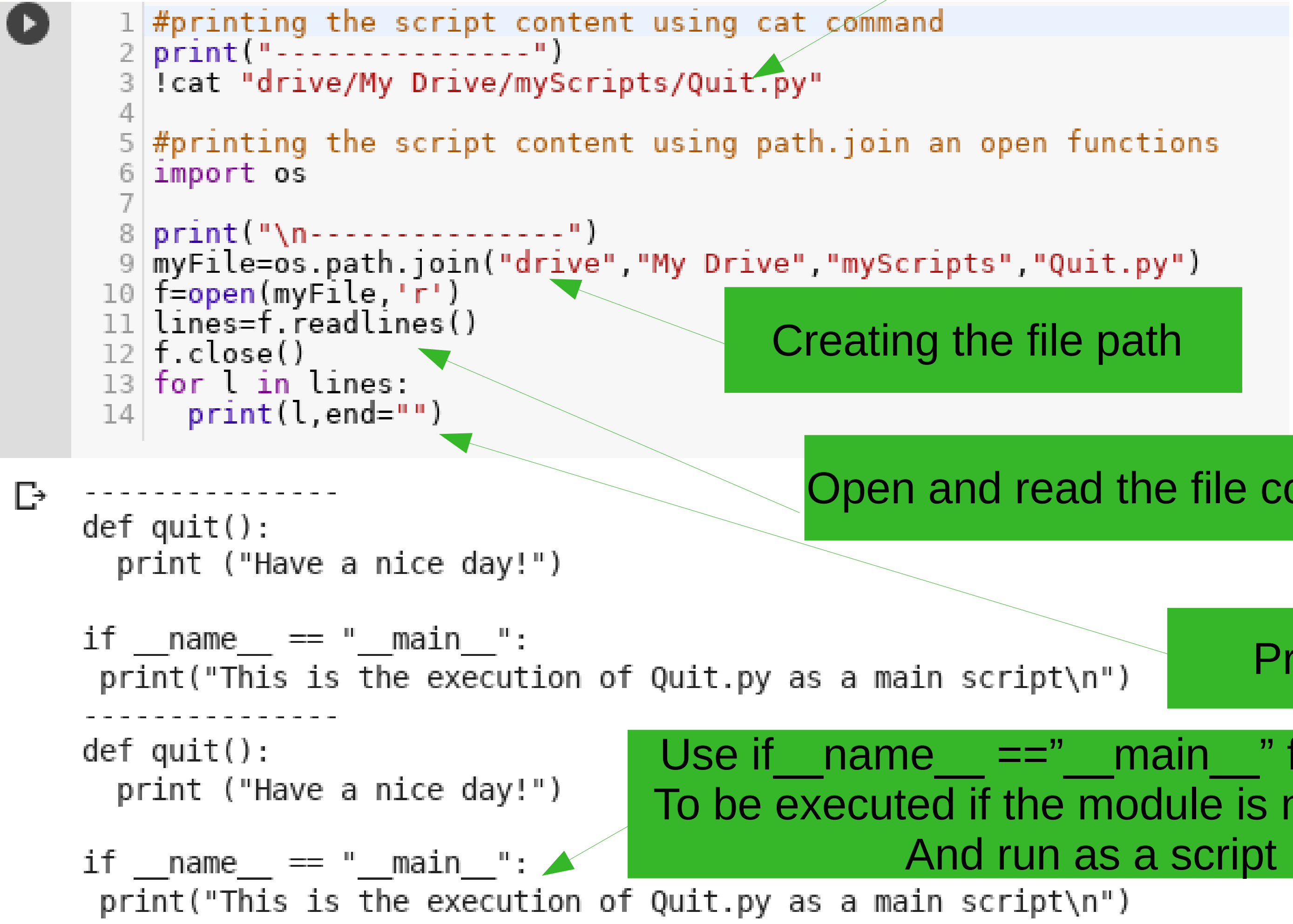
Running the file as a script

[By]

7- System programming

System Programming

Using ‘!cat’ to print the content of the file



```
#printing the script content using cat command
print("---------------")
!cat "drive/My Drive/myScripts/Quit.py"

#printing the script content using path.join an open functions
import os

print("\n---------------")
myFile=os.path.join("drive","My Drive","myScripts","Quit.py")
f=open(myFile,'r')
lines=f.readlines()
f.close()
for l in lines:
  print(l,end="")
```

```
---------------
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")
```

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

[By]

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