intro_Python

August 22, 2017

1 A Introduction to Python for dummies...

This is part of the Python lecture given by Veronica Gomez Llanos and Christophe Morisset at IA-UNAM.

1.0.1 Using Python as a calculator

Using of "print" command is not necesary to obtain a result. Just type some operations and the result is obtain with ENTER.

```
In [1]: 2 + 25
```

```
Out[1]: 27
```

```
In [2]: (2+3)*(3+4)/(5*5)
```

```
Out[2]: 1.4
```

```
In [3]: (2+3) * (3+4.) / (5*5)
```

```
Out[3]: 1.4
```

```
In [4]: # If you are using python 2.X, the default behaviour is not this one.
        # Do the following to be sure you are using the python 3.N division:
        from __future__ import division
```

Python likes the use of spaces to make scripts more readable

The art of writing good python code is described in the following document: http://legacy.python.org/dev/peps/pep-0008/

1.0.2 Assignments

Like any other langage, you can assign a value to a variable. This is done with = symbol:

```
In [5]: a = 4
```

A lot of operations can be performed on the variables. The most basics are for example:

```
In [6]: a
```

```
Out[6]: 4

In [7]: a = a + 1
        a

Out[7]: 5

In [8]: a *= 4 # similar to a = a * 4
        a

Out[8]: 20

In [9]: a, b = 1, 3
        a, b

Out[9]: (1, 3)
```

Some variable name are not available, they are reserved to python itself: and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield

```
In [10]: lambda_ = 2
         file = 3
```

1.0.3 Comments

```
In [11]: a = 2 # this is a comment

In [12]: """ This is a large comment
         on multiple lines
         ending as it started
         """

Out[12]: ' This is a large comment\non multiple lines\nending as it started\n'
```

1.0.4 Types

The types used in Python are: integers, long integers, floats (double prec.), complexes, strings, booleans.

Double precision: machine dependent, generally between 10ˆ-308 and 10ˆ308, with 16 significant digits.

The function type gives the type of its argument:

```
In [13]: type(2)

Out[13]: int

In [14]: type(2.3)

Out[14]: float

In [15]: int(0.8) # truncating

Out[15]: 0

In [16]: round(0.8765566777) # nearest, result is integer (was float with python 2.N)

Out[16]: 1
```

1.0.5 Complex numbers

```
In [17]: a = 1.5 + 0.5j

In [18]: a**2.

Out[18]: (2+1.5j)

In [19]: (1+2j)*(1-2j)

Out[19]: (5+0j)

In [20]: a.real

Out[20]: 1.5

In [21]: (a**3).imag

Out[21]: 3.25

In [22]: a.conjugate() # this is a function, it requieres ()

Out[22]: (1.5-0.5j)
```

1.0.6 Booleans

Comparison operators are <, >, <=, >=, ==, !=

```
In [23]: 5 < 3

Out[23]: False

In [24]: a = 5
         b = 7

In [25]: b < a

Out[25]: False

In [26]: c = 2

In [27]: c < a < b

Out[27]: True

In [28]: a < b and b < c

Out[28]: False

In [29]: res = a < 7
         print(res, type(res))
```

```
True <class 'bool'>
```

```
In [30]: print(int(res))
         print(int(not res))
```

```
1
0
```

```
In [31]: not res is True
```

```
Out[31]: False
```

```
In [32]: a = True
         print(a)
```

```
True
```

1.0.7 Formating strings

```
In [33]: print("Hello world!")
```

```
Hello world!
```

```
In [34]: print('Hello world!')
```

```
Hello world!
```

```
In [35]: print("Hello I'm here") # ' inside ""
```

```
Hello I'm here
```

```
In [36]: # This is the old fashion way of formating outputs (C-style)
         a = 7.5
         b = 'tralala'
         c = 8.9e-33
         print('a = %f, b = %s, c = %e' % (a, b, c))
```

```
a = 7.500000, b = tralala, c = 8.900000e-33
```

```
In [37]: # The new way is using the format() method of the string object, and {} to define which
         print('a = {} & b = {} & c = {} \\\\'.format(a,b,c))
         print('a = {0}, b = {1}, c = {2}'.format(a**2,b,c))
         print('a = {:f}, b = {:20s}, c = {:15.3e}'.format(a,b,c))
```

```
a = 7.5 & b = tralala & c = 8.9e-33 \\
a = 56.25, b = tralala, c = 8.9e-33
a = 7.500000, b = tralala             , c =       8.900e-33
```

Much more on this here: https://docs.python.org/3/tutorial/inputoutput.html

1.0.8 Strings

```
In [38]: a = "this is a    string"

In [39]: len(a)

Out[39]: 19
```

A lot of commands can operate on strings. Strings, like ANYTHING in python, are objects. Methods are run on objects by dots:

```
In [40]: a.upper()

Out[40]: 'THIS IS A    STRING'

In [41]: a.title()

Out[41]: 'This Is A    String'

In [42]: a.split()

Out[42]: ['this', 'is', 'a', 'string']

In [43]: a.split()[1]

Out[43]: 'is'

In [44]: a = "This is a string.   With various sentences."

In [45]: a.split('.')

Out[45]: ['This is a string', '   With various sentences', '']

In [46]: a.split('.')[1].strip() # Here we define the character used to split. The default is sp

Out[46]: 'With various sentences'

In [47]: a = 'tra'
         b = 'la'
         print(' '.join((a,b,b)))
         print('-'.join((a,b,b)))
         print(''.join((a,b,b)))
         print(a+b+b)
         print(' '.join((a,b,b)).split())
         print(' & '.join((a,b,b)) + '\\\\')

tra la la
tra-la-la
tralala
tralala
['tra', 'la', 'la']
tra & la & la\\
```

1.0.9 Containers: Tuples, Lists and Dictionaries

list: a collection of objects. May be of different types. It has an order.

```
In [48]: L = ['red','green','blue'] # squared brackets are used to define lists

In [49]: type(L) # Print the type of L

Out[49]: list

In [50]: L[1]

Out[50]: 'green'

In [51]: L[0] # indexes start at 0 !!!

Out[51]: 'red'

In [52]: L[-1] # last element

Out[52]: 'blue'

In [53]: L[-3]

Out[53]: 'red'

In [54]: L = L + ['black', 'white'] # addition symbol is used to agregate values to a list. See

In [55]: print(L)

['red', 'green', 'blue', 'black', 'white']


In [56]: L[1:3] # L[start:stop] : elements if index i, where start <= i < stop !! stop not inclu

Out[56]: ['green', 'blue']

In [57]: L[2:] # boudaries can be omited

Out[57]: ['blue', 'black', 'white']

In [58]: L[-2:]

Out[58]: ['black', 'white']

In [59]: L[::2] # L[start:stop:step] every 2 elements

Out[59]: ['red', 'blue', 'white']

In [60]: L[::-1]

Out[60]: ['white', 'black', 'blue', 'green', 'red']
```

Lists are mutable: their content can be modified.

```
In [61]: L[2] = 'yellow'
         L

Out[61]: ['red', 'green', 'yellow', 'black', 'white']

In [62]: L.append('pink') # append a value at the end
         L

Out[62]: ['red', 'green', 'yellow', 'black', 'white', 'pink']

In [63]: L.insert(2, 'blue')     #L.insert(index, object) -- insert object before index
         L

Out[63]: ['red', 'green', 'blue', 'yellow', 'black', 'white', 'pink']

In [64]: L.extend(['magenta', 'purple'])
         L

Out[64]: ['red',
          'green',
          'blue',
          'yellow',
          'black',
          'white',
          'pink',
          'magenta',
          'purple']

In [65]: L.append(['magenta', 'azul'])
         L

Out[65]: ['red',
          'green',
          'blue',
          'yellow',
          'black',
          'white',
          'pink',
          'magenta',
          'purple',
          ['magenta', 'azul']]

In [66]: L.append(2)
         L

Out[66]: ['red',
          'green',
          'blue',
          'yellow',
          'black',
          'white',
          'pink',
          'magenta',
          'purple',
          ['magenta', 'azul'],
          2]
```

```
In [67]: L = L[::-1] # reverse order
         L
```

```
Out[67]: [2,
          ['magenta', 'azul'],
          'purple',
          'magenta',
          'pink',
          'white',
          'black',
          'yellow',
          'blue',
          'green',
          'red']
```

```
In [68]: L2 = L[:-3] # cutting the last 3 elements
         print(L)
         print(L2)
```

```
[2, ['magenta', 'azul'], 'purple', 'magenta', 'pink', 'white', 'black', 'yellow', 'blue', 'green
[2, ['magenta', 'azul'], 'purple', 'magenta', 'pink', 'white', 'black', 'yellow']
```

```
In [69]: L[25] # Out of range leads to error
```

```
        ---------------------------------------------------------------------------

        IndexError                                Traceback (most recent call last)

        <ipython-input-69-c16babb9288f> in <module>()
    ----> 1 L[25] # Out of range leads to error


        IndexError: list index out of range
```

```
In [70]: print(L)
         print(L[20:25]) # But NO ERROR when slicing.
         print(L[20:])
         print(L[2:20])
```

```
[2, ['magenta', 'azul'], 'purple', 'magenta', 'pink', 'white', 'black', 'yellow', 'blue', 'green
[]
[]
['purple', 'magenta', 'pink', 'white', 'black', 'yellow', 'blue', 'green', 'red']
```

```
In [71]: print(L.count('yellow'))
```

```
1
```

```
In [72]: L2 = L[2:20]
         L2.sort() # One can use TAB to look for the methods (functions that apply to an object)
         print(L2)
```

```
['black', 'blue', 'green', 'magenta', 'pink', 'purple', 'red', 'white', 'yellow']
```

```
In [73]: a = [1,2,3]
         b = [10,20,30]
```

```
In [74]: print(a+b) # may not be what you expected, but rather logical too
```

```
[1, 2, 3, 10, 20, 30]
```

```
In [75]: print(a*b) # Does NOT multiply element by element. Numpy will do this job.
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-75-ddfd21d938fe> in <module>()
    ----> 1 print(a*b) # Does NOT multiply element by element. Numpy will do this job.


        TypeError: can't multiply sequence by non-int of type 'list'
```

```
In [76]: L = range(4) # Create an interator. Notice the parameter is the number of elements, not
         print(L) # In python 2, that was a lit
         print(list(L))
```

```
range(0, 4)
[0, 1, 2, 3]
```

```
In [77]: L = range(2, 20, 2) # every 2 integer
         print(L)
```

```
range(2, 20, 2)
```

The types os the elements of a list are not always the same:

```
In [78]: L = [1, '1', 1.4]
         L

Out[78]: [1, '1', 1.4]
```

Remove the n+1-th element:

```
In [79]: L = list(range(0,20,2))
         print(L)
         del(L[5])
         print(L)

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
[0, 2, 4, 6, 8, 12, 14, 16, 18]
```

Slicing: extracting sub-list of a list

```
In [80]: a = [[1, 2, 3], [10, 20, 30], [100, 200, 300]] # Not a 2D table, but rather a table of
         print(a)
         print(a[0])
         print(a[1][1])

[[1, 2, 3], [10, 20, 30], [100, 200, 300]]
[1, 2, 3]
20
```

```
In [81]: print(a[1,1]) # Does NOT work

        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-81-d8214b6adea8> in <module>()
    ----> 1 print(a[1,1]) # Does NOT work


        TypeError: list indices must be integers or slices, not tuple
```

```
In [82]: b = a[1]
         print(b)
```

```
[10, 20, 30]
```

```
In [83]: b[1] = 999 # Changing the value of a single element
         print(b)
```

```
[10, 999, 30]
```

```
In [84]: print(a) # Changing b changed a !!!
```

```
[[1, 2, 3], [10, 999, 30], [100, 200, 300]]
```

```
In [85]: b[1] is a[1][1]

Out[85]: True
```

```
In [86]: c = a[1][::] # copy instead of slicing
         print(c)
         c[0] = 77777
         print(c)
         print(a)
```

```
[10, 999, 30]
[77777, 999, 30]
[[1, 2, 3], [10, 999, 30], [100, 200, 300]]
```

tuples: like lists, but inmutables

```
In [87]: T = (1,2,3)
         T

Out[87]: (1, 2, 3)
```

```
In [88]: T2 = 1, 2, 3
         print(T2)
         type(T2)
```

```
(1, 2, 3)
```

```
Out[88]: tuple
```

```
In [89]: T[1]

Out[89]: 2
```

tuples are unmutables

```
In [90]: T[1] = 3 # Does NOT work!
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-90-6dd68cc28786> in <module>()
    ----> 1 T[1] = 3 # Does NOT work!


        TypeError: 'tuple' object does not support item assignment
```

Dictionnaries A dictionary is basically an efficient table that maps keys to values. It is an unordered container

```
In [91]: D = {'Christophe': 12, 'Antonio': 15} # defined by {key : value}

In [92]: D['Christophe'] # access to a value by the key

Out[92]: 12

In [93]: D.keys() # list of the dictionary keys

Out[93]: dict_keys(['Christophe', 'Antonio'])

In [94]: D['Yilen'] = 16 # adding a new entry

In [95]: print(D)

{'Christophe': 12, 'Antonio': 15, 'Yilen': 16}


In [96]: print(D[0]) # use the keys to acces the elements. No order in dictionnary.
```

```
        ---------------------------------------------------------------------------

        KeyError                                  Traceback (most recent call last)

        <ipython-input-96-035806c13f20> in <module>()
    ----> 1 print(D[0]) # use the keys to acces the elements. No order in dictionnary.


        KeyError: 0
```

1.0.10 Blocks

Blocks are defined by indentation. Looks nice and no needs for end :-)

```
In [97]: for i in [1,2,3]: print(i) # compact way, not recomended.
```

```
1
2
3
```

```
In [98]: for cosa in [1,'ff',2]:
             print(cosa)
             print('end')
         print('final end') # end of the identation means end of the block
```

```
1
end
ff
end
2
end
final end
```

```
In [99]: # defining a dictionary:
         ATOMIC_MASS = {}
         ATOMIC_MASS['H'] = 1
         ATOMIC_MASS['He'] = 4
         ATOMIC_MASS['C'] = 12
         ATOMIC_MASS['N'] = 14
         ATOMIC_MASS['O'] = 16
         ATOMIC_MASS['Ne'] = 20
         ATOMIC_MASS['Ar'] = 40
         ATOMIC_MASS['S'] = 32
         ATOMIC_MASS['Si'] = 28
         ATOMIC_MASS['Fe'] = 55.8
         # Print the keys and values from the dictionary. As it is not ordered , they come as th
         for key in ATOMIC_MASS.keys():
             print(key, ATOMIC_MASS[key])
```

```
H 1
He 4
C 12
N 14
O 16
Ne 20
Ar 40
S 32
```

```
Si 28
Fe 55.8
```

```
In [100]: for key in sorted(ATOMIC_MASS): # sorting using the keys
              print('Element: {0:3s}  Atomic Mass: {1}'.format(key, ATOMIC_MASS[key]))
```

```
Element: Ar   Atomic Mass: 40
Element: C    Atomic Mass: 12
Element: Fe   Atomic Mass: 55.8
Element: H    Atomic Mass: 1
Element: He   Atomic Mass: 4
Element: N    Atomic Mass: 14
Element: Ne   Atomic Mass: 20
Element: O    Atomic Mass: 16
Element: S    Atomic Mass: 32
Element: Si   Atomic Mass: 28
```

a key parameter can be used to specify a function to be called on each list element prior to making comparisons. More in sorted function here: https://wiki.python.org/moin/HowTo/Sorting or here: http://www.pythoncentral.io/how-to-sort-a-list-tuple-or-object-with-sorted-in-python/

```
In [101]: for elem in sorted(ATOMIC_MASS, key = ATOMIC_MASS.get): # sorting using the values
              print('Element: {0:3s}  Atomic Mass: {1}'.format(elem, ATOMIC_MASS[elem]))
```

```
Element: H    Atomic Mass: 1
Element: He   Atomic Mass: 4
Element: C    Atomic Mass: 12
Element: N    Atomic Mass: 14
Element: O    Atomic Mass: 16
Element: Ne   Atomic Mass: 20
Element: Si   Atomic Mass: 28
Element: S    Atomic Mass: 32
Element: Ar   Atomic Mass: 40
Element: Fe   Atomic Mass: 55.8
```

```
In [102]: for idx, elem in enumerate(sorted(ATOMIC_MASS, key = ATOMIC_MASS.get)): # adding an in
              print('{0:2} Element: {1:2s}  Atomic Mass: {2:4.1f}'.format(idx+1, elem, ATOMIC_MA
```

```
 1 Element: H   Atomic Mass:  1.0
 2 Element: He  Atomic Mass:  4.0
 3 Element: C   Atomic Mass: 12.0
 4 Element: N   Atomic Mass: 14.0
 5 Element: O   Atomic Mass: 16.0
 6 Element: Ne  Atomic Mass: 20.0
 7 Element: Si  Atomic Mass: 28.0
 8 Element: S   Atomic Mass: 32.0
 9 Element: Ar  Atomic Mass: 40.0
10 Element: Fe  Atomic Mass: 55.8
```

```
In [103]: for i in range(10):
              if i > 5:
                  print(i)
```

```
6
7
8
9
```

```
In [104]: for i in range(10):
              if i > 5:
                  print(i)
              else:
                  print('i lower than five')
          print('END')
```

```
i lower than five
i lower than five
i lower than five
i lower than five
i lower than five
i lower than five
6
7
8
9
END
```

Other commands are: if...elif...else AND while...

1.0.11 List and dictionnary comprehension

```
In [105]: A = [] # defining an empty list
          for i in range(4):
              A.append(i**2) # filling the list with values
          print(A)
```

```
[0, 1, 4, 9]
```

```
In [106]: # more compact way to do the same thing
          B = [i**2 for i in range(4)]
          print(B)
```

```
[0, 1, 4, 9]
```

```
In [107]: # The same is also used for dictionnaries
          D = {'squared_{}'.format(k) : k**2 for k in range(10)}
          print(D)
```

```
{'squared_0': 0, 'squared_1': 1, 'squared_2': 4, 'squared_3': 9, 'squared_4': 16, 'squared_5': 2
```

1.0.12 Functions, procedures

```
In [108]: def func1(x):
              print(x**3)
          func1(5)
```

```
125
```

```
In [109]: def func2(x,
                    y):
              """
              Return the cube and the 4th power of the two parameters
              """
              return(x**3, y**4)
          a = func2(3, 5)

          help(func2)
```

```
Help on function func2 in module __main__:

func2(x, y)
    Return the cube and the 4th power of the two parameters
```

```
In [110]: #func2() shift-TAB inside the parenthesis
          func2?

In [111]: print(a)
          print(func2(4, 6))
```

```
(27, 625)
(64, 1296)
```

```
In [112]: def func3(x, y, z, a=0, b=1):
              """
              This function has 5 arguments, 2 of them have default values (then not mandatory)
```

```
            """
            return a + b * (x**2 + y**2 + z**2)**0.5
        D = func3(3, 4, 5)
        print(D)
```

```
7.0710678118654755
```

```
In [113]: E = func3(3, 4, 5, 10, 100)
          print(E)
```

```
717.1067811865476
```

```
In [114]: F = func3(x=3, y=4, z=5, a=10, b=100)
          print(F)
```

```
717.1067811865476
```

```
In [115]: G = func3(3, 4, 5, a=10, 100) # ERROR!
          print(G)
```

```
          File "<ipython-input-115-7963f4c1b801>", line 1
        G = func3(3, 4, 5, a=10, 100) # ERROR!
                                ^
    SyntaxError: positional argument follows keyword argument
```

```
In [116]: H = func3(3, 4, 5, a=10, b=100)
          print(H)
```

```
717.1067811865476
```

```
In [117]: I = func3(z=5, x=3, y=4, a=10, b=100) # quite risky!
          print(I)
```

```
717.1067811865476
```

Lambda function is used to creat simple (single line) functions:

```
In [118]: J = lambda x, y, z: (x**2 + y**2 + z**2)**0.5
          J(1,2,3)
```

```
Out[118]: 3.7416573867739413
```

```
In [119]: def J(x,y,z):
              return (x**2 + y**2 + z**2)**0.5
          J(1,2,3)

Out[119]: 3.7416573867739413
```

```
In [120]: print((lambda x,y,z: x+y+z)(0,1,2))

3
```

Changing the value of variable inside a routine Parameters to functions are references to objects, which are passed by value. When you pass a variable to a function, python passes the reference to the object to which the variable refers (the value). Not the variable itself. If the value is immutable, the function does not modify the caller's variable. If the value is mutable, the function may modify the caller's variable in-place, if a mutation of the variable is done (not if a new mutable value is assigned):

```
In [121]: def try_to_modify(x, y, z):
              x = 23
              y.append(22)
              z = [29] # new reference
              print('    IN THE ROUTINE')
              print(x)
              print(y)
              print(z)

          # The values of a, b and c are set
          a = 77
          b = [79]
          c = [78]

          print('    INIT')
          print(a)
          print(b)
          print(c)

          try_to_modify(a, b, c)

          print('    AFTER THE ROUTINE')
          print(a)
          print(b)
          print(c)

   INIT
77
[79]
[78]
```

```
   IN THE ROUTINE
23
[79, 22]
[29]
   AFTER THE ROUTINE
77
[79, 22]
[78]
```

Variables from outside (from a level above) are known:

```
In [122]: a = 5
          def test_a(x):
              print(a*x)
          test_a(5)
          a = 10
          test_a(5)
          print(a)
```

```
25
50
10
```

```
In [123]: # This works even if a2 is not known when defining the function:
          def test_a2(x):
              print(a2*x)
          a2 = 10
          test_a2(5)
```

```
50
```

Variables from inside are unknown outside:

```
In [124]: def test_g2():
              g2 = 5
              print(g2)
          test_g2()
          print(g2)
```

```
5
```

```
        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)
        <ipython-input-124-edf2e6190ece> in <module>()
          3     print(g2)
          4 test_g2()
    ----> 5 print(g2)


        NameError: name 'g2' is not defined
```

Global variable is known outside:

```
In [125]: def test_g3():
              global g3
              g3 = 5
              print(g3)
          test_g3()
          print(g3)
```

```
5
5
```

Recursivity

```
In [126]: def fact(n):
              if n <= 0:
                  return 1
              return n*fact(n-1)
          print(fact(5))
          print(fact(8))
          print(fact(50))
```

```
120
40320
30414093201713378043612608166064768844377641568960512000000000000
```

1.0.13 Scripting

```
In [127]: %%writefile ex1.py
          #!/usr/bin/env python
          # -*- coding: utf-8 -*-
          def f1(x):
              """
              This is an example of a function, that returns x^2
              - parameter: x
              """
              return x**2
```

```
Overwriting ex1.py
```

```
In [128]: #We have created a script named ex1.py, to see inside the script we use:
          !cat ex1.py
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
def f1(x):
    """
    This is an example of a function, that returns x^2
    - parameter: x
    """
    return x**2
```

The first two lines of the ex1.py script are used to: tell the computer that this is python code, and to use special characters, respectevely.

There are different ways to use the functions defined in a script

```
In [129]: import ex1 #This imports the file named ex1.py from the current directory
          #or from one of the directories in the search path
          print(ex1.f1(4))#This uses the function f1(x) defined in the ex1.py file for the value
```

```
16
```

```
In [130]: from ex1 import f1 #This is another way to call the function f1(x) from ex1.py
          print(f1(4))
```

```
16
```

```
In [131]: from ex1 import * # DO NOT DO THIS! Very hard to know where f1 is comming from (debugi
          print(f1(4))
```

```
16
```

```
In [132]: import ex1 as tt
          print(tt.f1(10))
```

```
100
```

```
In [133]: %run ex1 # This is the same as doing a copy-paste of the content of the file
          f1(9)
```

```
Out[133]: 81
```

1.0.14 Importing libraries

Not all the power of python is available when we call (i)python. Some additional libraries (included in the python package, or as additional packages, like numpy) can be imported to increase the capacities of python. This is the case of the math library:

```
In [134]: print(sin(3.))
```

```
        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-134-3774d5a0e3c9> in <module>()
    ----> 1 print(sin(3.))


        NameError: name 'sin' is not defined
```

```
In [135]: import math
          print(math.sin(3.))
```

```
0.1411200080598672
```

```
In [136]: # This displays the help of the math library
          math?

In [137]: # This imports all the elements of the library in the current domain name (IS NOT A GO
          from math import *
          sin(3.)

Out[137]: 0.1411200080598672

In [138]: # One can look at the contents of a library with dir:
          print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'a
```

```
In [139]: # The help command is used to have information on a given function:
          help(math.sin)
```

```
Help on built-in function sin in module math:

sin(...)
    sin(x)

    Return the sine of x (measured in radians).
```

```
In [140]: help(log)

Help on built-in function log in module math:

log(...)
    log(x[, base])

    Return the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.
```

```
In [141]: print(math.pi)

3.141592653589793
```

```
In [142]: math.pi = 2.71

In [143]: print(math.pi)

2.71
```

```
In [144]: import math

In [145]: math.pi

Out[145]: 2.71

In [149]: # In python 3 you need to import reload, in python 2 is included
          from importlib import reload

In [150]: reload(math)

Out[150]: <module 'math' from '/home/vero/anaconda2/envs/ipy3/lib/python3.6/lib-dynload/math.cpy

In [151]: # In python 2 the value is reset, in python 3 this is not the case!!!!
          math.pi

Out[151]: 2.71

In [152]: from math import pi as pa

In [153]: pa

Out[153]: 2.71

In [154]: math = 2
          math.pi
```

```
    ---------------------------------------------------------------------------

    AttributeError                            Traceback (most recent call last)

    <ipython-input-154-70a02d6227fb> in <module>()
      1 math = 2
----> 2 math.pi


    AttributeError: 'int' object has no attribute 'pi'
```

```
In [155]: pa
```

```
Out[155]: 2.71
```

```
In [ ]:
```