

intro_Python

August 17, 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 necessary to obtain a result. Just type some operations and the result is obtain with ENTER.

```
In [2]: 2 + 22
```

```
Out[2]: 24
```

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

```
Out[3]: 1
```

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

```
Out[4]: 1.4
```

```
In [5]: # 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\nnon multiple lines\nnending 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 10^{-308} and 10^{308} , 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 Formatting 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 formatting 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 space
```

```
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 omitted
```

```
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 [71]: 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 [72]: print(L.count('yellow'))
```

```
1
```

```
In [73]: 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 [74]: a = [1,2,3]
         b = [10,20,30]
```

```
In [75]: print(a+b) # may not be what you expected, but rather logical too
```

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

```
In [76]: print(a*b) # Does NOT multiply element by element. Numpy will do this job.
```

```
-----
TypeError
```

```
Traceback (most recent call last)
```

```
<ipython-input-76-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 [78]: L = range(4) # Create an iterator. Notice the parameter is the number of elements, not
         print(L) # In python 2, that was a list
         print(list(L))
```

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

```
In [79]: L = range(2, 20, 2) # every 2 integer
         print(L)
```

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

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

```
In [80]: L = [1, '1', 1.4]
         L
```

```
Out[80]: [1, '1', 1.4]
```

Remove the n+1-th element:

```
In [81]: 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 [82]: 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 [83]: print(a[1,1]) # Does NOT work
```

TypeError

Traceback (most recent call last)

<ipython-input-83-d8214b6adea8> in <module>()

----> 1 print(a[1,1]) # Does NOT work

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

```
In [84]: b = a[1]
         print(b)
```

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

```
In [85]: b[1] = 999 # Changing the value of a single element  
        print(b)
```

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

```
In [86]: print(a) # Changing b changed a !!!
```

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

```
In [87]: b[1] is a[1][1]
```

```
Out[87]: True
```

```
In [88]: 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 immutable

```
In [89]: T = (1,2,3)  
        T
```

```
Out[89]: (1, 2, 3)
```

```
In [90]: T2 = 1, 2, 3  
        print(T2)  
        type(T2)
```

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

```
Out[90]: tuple
```

```
In [91]: T[1]
```

```
Out[91]: 2
```

tuples are unmutables

```
In [92]: T[1] = 3 # Does NOT work!
```

```
-----  
  
TypeError                                Traceback (most recent call last)  
  
  <ipython-input-92-6dd68cc28786> in <module>()  
----> 1 T[1] = 3 # Does NOT work!  
  
TypeError: 'tuple' object does not support item assignment
```

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

```
In [93]: D = {'Christophe': 12, 'Antonio': 15} # defined by {key : value}
```

```
In [94]: D['Christophe'] # access to a value by the key
```

```
Out[94]: 12
```

```
In [95]: D.keys() # list of the dictionary keys
```

```
Out[95]: dict_keys(['Christophe', 'Antonio'])
```

```
In [96]: D['Yilen'] = 16 # adding a new entry
```

```
In [97]: print(D)
```

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

```
In [98]: print(D[0]) # use the keys to acces the elements. No order in dictionnary.
```

```
-----  
  
KeyError                                Traceback (most recent call last)  
  
  <ipython-input-98-035806c13f20> 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 [99]: for i in [1,2,3]: print(i) # compact way, not recommended.
```

```
1
2
3
```

```
In [100]: for cosa in [1,'ff',2]:
           print(cosa)
           print('end')
       print('final end') # end of the indentation means end of the block
```

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

```
In [101]: # 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 t
          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 [102]: 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 [103]: 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 [104]: for idx, elem in enumerate(sorted(ATOMIC_MASS, key = ATOMIC_MASS.get)): # adding an index
          print('{0:2} Element: {1:2s} Atomic Mass: {2:4.1f}'.format(idx+1, elem, ATOMIC_MASS[elem]))
```

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

```
6
7
8
9
```

```
In [106]: 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 dictionary comprehension

```
In [107]: 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 [108]: # more compact way to do the same thing
          B = [i**2 for i in range(4)]
          print(B)
```

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

```
In [109]: # The same is also used for dictionaries
```

```
    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': 25, 'squared_6': 36, 'squared_7': 49, 'squared_8': 64, 'squared_9': 81}
```

1.0.12 Functions, procedures

```
In [110]: def func1(x):  
           print(x**3)  
           func1(5)
```

```
125
```

```
In [111]: 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 [112]: #func2() shift-TAB inside the parenthesis  
          func2?
```

```
In [113]: print(a)  
          print(func2(4, 6))
```

```
(27, 625)
```

```
(64, 1296)
```

```
In [114]: 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 [115]: E = func3(3, 4, 5, 10, 100)
          print(E)

```

717.1067811865476

```

In [116]: F = func3(x=3, y=4, z=5, a=10, b=100)
          print(F)

```

717.1067811865476

```

In [117]: G = func3(3, 4, 5, a=10, 100) # ERROR!
          print(G)

```

```

File "<ipython-input-117-7963f4c1b801>", line 1
G = func3(3, 4, 5, a=10, 100) # ERROR!
              ^

```

SyntaxError: positional argument follows keyword argument

```

In [118]: H = func3(3, 4, 5, a=10, b=100)
          print(H)

```

717.1067811865476

```

In [119]: 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 [120]: J = lambda x, y, z: (x**2 + y**2 + z**2)**0.5
          J(1,2,3)

```

Out[120]: 3.7416573867739413

```
In [121]: def J(x,y,z):
           return (x**2 + y**2 + z**2)**0.5
           J(1,2,3)
```

```
Out[121]: 3.7416573867739413
```

```
In [122]: 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 [123]: 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 [124]: a = 5
          def test_a(x):
              print(a*x)
          test_a(5)
          a = 10
          test_a(5)
          print(a)

25
50
10
```

```
In [125]: # 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 [126]: def test_g2():
          g2 = 5
          print(g2)
          test_g2()
          print(g2)

5
```

NameError

Traceback (most recent call last)

```
<ipython-input-126-edf2e6190ece> in <module>()
      3     print(g2)
      4 test_g2()
----> 5 print(g2)
```

NameError: name 'g2' is not defined

Global variable is known outside:

```
In [127]: def test_g3():
          global g3
          g3 = 5
          print(g3)
          test_g3()
          print(g3)
```

```
5
5
```