intro_Python

January 10, 2021

1 A Introduction to Python for dummies...

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

[1]:
```
# The following is to know when this notebook has been run and with which␣
↪python version.
import time, sys
print(time.ctime())
print(sys.version.split('|')[0])
```

```
Wed Oct  7 12:30:39 2020
3.7.6 (default, Jan  8 2020, 13:42:34)
[Clang 4.0.1 (tags/RELEASE_401/final)]
```

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.

[2]:
```
2236342 + 22783654
```

[2]:
```
25019996
```

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

[3]:
```
1.4
```

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

[4]:
```
1.4
```

[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:

```
[6]: a = 4
```

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

```
[7]: a
```

```
[7]: 4
```

```
[8]: a = a + 1
     a
```

```
[8]: 5
```

```
[9]: a *= 4 # similar to a = a * 4
     a
```

```
[9]: 20
```

```
[10]: a, b = 1, 3
      a, b
```

```
[10]: (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

```
[11]: lambda_ = 2
      file = 3
```

1.0.3 Comments

```
[12]: a = 2 # this is a comment
```

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

```
[13]: ' 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 10^-308 and 10^308, with 16 significant digits.

The function type gives the type of its argument:

```
[14]: type(2)
```

```
[14]: int
```

```
[15]: type(2.3)
```

```
[15]: float
```

```
[16]: int(0.8) # truncating
```

```
[16]: 0
```

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

```
[17]: 1
```

1.0.5 Complex numbers

```
[18]: a = 1.5 + 0.5j
```

```
[19]: a**2.
```

```
[19]: (2+1.5j)
```

```
[20]: (1+2j)*(1-2j)
```

```
[20]: (5+0j)
```

```
[21]: a.real
```

```
[21]: 1.5
```

```
[22]: (a**3).imag
```

```
[22]: 3.25
```

```
[23]: a.conjugate() # this is a function, it requieres ()
```

```
[23]: (1.5-0.5j)
```

1.0.6 Booleans

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

```
[24]: 5 < 3
```

```
[24]: False
```

```
[25]: 5 == 5
```

```
[25]: True
```

```
[26]: 5 != 7
```

```
[26]: True
```

```
[27]: a = 5
      b = 7
```

```
[28]: b < a
```

```
[28]: False
```

```
[29]: c = 2
```

```
[30]: c < a < b
```

```
[30]: True
```

```
[31]: a < b and b < c
```

```
[31]: False
```

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

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

```
[33]: print(int(res))
      print(int(not res))
```

```
1
0
```

```
[34]: not res is True
```

```
[34]: False
```

```
[35]: a = True
      print(a)
```

```
True
```

1.0.7 Formating strings

```
[36]: print("Hello world!")
```

```
Hello world!
```

```
[37]: print('Hello world!')
```

```
Hello world!
```

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

```
Hello I'm here
```

```
[39]: # 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
```

```
[40]: # The new way is using the format() method of the string object, and {} to␣
       ↪define which value to print and using which format.
      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

```
[41]: a = "this is a    string"
```

```
[42]: len(a)
```

```
[42]: 19
```

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

```
[43]: a.upper()
```

```
[43]: 'THIS IS A    STRING'
```

```
[44]: a.title()
```

```
[44]: 'This Is A    String'
```

```
[45]: a.split()
```

```
[45]: ['this', 'is', 'a', 'string']
```

```
[46]: a.split()[1]
```

```
[46]: 'is'
```

```
[47]: a = "This is a string.   With various sentences."
```

```
[48]: a.split('.')
```

```
[48]: ['This is a string', '   With various sentences', '']
```

```
[49]: a.split('.')[1].strip() # Here we define the character used to split. The␣
       ↪default is space (any comminaison of spaces)
```

```
[49]: 'With various sentences'
```

```
[50]: 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\\
```

```
[51]: a = 5.6
      b = 7.854
      print(' & '.join(('{:6.3e}'.format(a),
                        '{:6.3f}'.format(b),
                        '{:6.3f}'.format(b))) + ' \\\\')
```

```
5.600e+00 &  7.854 &  7.854 \\
```

1.0.9 Containers: Tuples, Lists and Dictionaries

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

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

```
[53]: type(L) # Print the type of L
```

```
[53]: list
```

```
[54]: L[1]
```

```
[54]: 'green'
```

```
[55]: L[0] # indexes start at 0 !!!
```

```
[55]: 'red'
```

```
[56]: L[-1] # last element
```

```
[56]: 'blue'
```

```
[57]: L[-3]
```

```
[57]: 'red'
```

```
[58]: L = L + ['black', 'white'] # addition symbol is used to agregate values to a␣
       ↪list. See below other way.
```

```
[59]: print(L)
```

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

```
[60]: L[1:3] # L[start:stop] : elements if index i, where start <= i < stop !! stop␣
       ↪not included !!
```

```
[60]: ['green', 'blue']
```

```
[61]: L[2:] # boudaries can be omited
```

```
[61]: ['blue', 'black', 'white']
```

```
[62]: L[-2:]
```

```
[62]: ['black', 'white']
```

```
[63]: L[::2] # L[start:stop:step] every 2 elements
```

```
[63]: ['red', 'blue', 'white']
```

```
[64]: L[::-1]
```

```
[64]: ['white', 'black', 'blue', 'green', 'red']
```

Lists are mutable: their content can be modified.

```
[65]: L[2]
```

```
[65]: 'blue'
```

```
[66]: L[2] = 'yellow'
      L
```

```
[66]: ['red', 'green', 'yellow', 'black', 'white']
```

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

```
[67]: ['red', 'green', 'yellow', 'black', 'white', 'pink']
```

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

```
[68]: ['red', 'green', 'blue', 'yellow', 'black', 'white', 'pink']
```

```
[69]: L.extend(['magenta', 'purple'])
      L
```

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

```
[70]: L.append(3)
      L
```

```
[70]: ['red',
       'green',
       'blue',
       'yellow',
       'black',
       'white',
       'pink',
       'magenta',
       'purple',
       3]
```

[71]:
```
L.append(['magenta', 'azul'])
L
```

[71]:
```
['red',
 'green',
 'blue',
 'yellow',
 'black',
 'white',
 'pink',
 'magenta',
 'purple',
 3,
 ['magenta', 'azul']]
```

[72]:
```
L = L[::-1] # reverse order
L
```

[72]:
```
[['magenta', 'azul'],
 3,
 'purple',
 'magenta',
 'pink',
 'white',
 'black',
 'yellow',
 'blue',
 'green',
 'red']
```

[73]:
```
L2 = L[:-3] # cutting the last 3 elements
print(L)
print(L2)
```

```
[['magenta', 'azul'], 3, 'purple', 'magenta', 'pink', 'white', 'black',
'yellow', 'blue', 'green', 'red']
[['magenta', 'azul'], 3, 'purple', 'magenta', 'pink', 'white', 'black',
'yellow']
```

[74]:
```
L[25] # Out of range leads to error
```

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

```
<ipython-input-74-895231b0620d> in <module>
----> 1 L[25] # Out of range leads to error

IndexError: list index out of range
```

[80]:
```
print(L)
print(L[20:25]) # But NO ERROR when slicing.
print(L[20:])
print(L[2:20])
```

```
[['magenta', 'azul'], 3, 'purple', 'magenta', 'pink', 'white', 'black',
'yellow', 'blue', 'green', 'red']
[]
[]
['purple', 'magenta', 'pink', 'white', 'black', 'yellow', 'blue', 'green',
'red']
```

[81]:
```
print(L.count('yellow'))
```

```
1
```

[82]:
```
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']
```

[83]:
```
a = [1,2,3]
b = [10,20,30]
```

[84]:
```
print(a + b) # may not be what you expected, but rather logical too
```

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

[85]:
```
print(a * b) # Does NOT multiply element by element. Numpy will do this job.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-85-3b1cf8d44990> 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'
```

[86]:
```
L = range(4) # Create an iterator. Notice the parameter is the number of␣
 ↪elements, not the last one. The end point is omited.
print(L) # In python 2, that was a list
print(list(L))
```

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

[87]:
```
L = range(2, 20, 2) # every 2 integer
print(list(L))
```

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

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

[88]:
```
L = [1, '1', 1.4]
L
```

[88]:
```
[1, '1', 1.4]
```

Remove the n+1-th element:

[89]:
```
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

[90]:
```
a = [[1, 2, 3], [10, 20, 30], [100, 200, 300]] # Not a 2D table, but rather a␣
 ↪table of tables.
print(a)
print(a[0])
print(a[1][1])
```

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

[91]:
```
print(a[1,1]) # Does NOT work
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-91-c5f7c3046063> in <module>
----> 1 print(a[1,1]) # Does NOT work
```

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

[92]:
```
b = a[1]
print(b)
```

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

[93]:
```
b[1] = 999 # Changing the value of a single element
print(b)
```

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

[94]:
```
print(a) # Changing b changed a !!!
```

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

[95]:
```
b[1] is a[1][1]
```

[95]:
```
True
```

[96]:
```
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

[97]:
```
T = (1,2,3)
T
```

[97]:
```
(1, 2, 3)
```

[98]:
```
T2 = 1, 2, 3
print(T2)
type(T2)
```

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

[98]:
```
tuple
```

[99]:
```
T[1]
```

```
[99]: 2
```

tuples are unmutables

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

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-100-a6ff89efe06f> in <module>
----> 1 T[1] = 3 # Does NOT work!

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

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

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

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

```
[102]: 12
```

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

```
[103]: dict_keys(['Christophe', 'Antonio'])
```

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

```
[105]: print(D)
```

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

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

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-106-4f29f8d63da6> 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 :-)

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

```
1
2
3
```

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

```
1
-----
ff
-----
2.0
-----
final end
```

```
[109]: # 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 they want.
       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
```

[110]:
```
for key in sorted(ATOMIC_MASS): # sorting using the keys
    print('Element: {0:3s}  Atomic Mass: {1:4.1f}'.format(key,␣
 ↪ATOMIC_MASS[key]))
```

```
Element: Ar   Atomic Mass: 40.0
Element: C    Atomic Mass: 12.0
Element: Fe   Atomic Mass: 55.8
Element: H    Atomic Mass:  1.0
Element: He   Atomic Mass:  4.0
Element: N    Atomic Mass: 14.0
Element: Ne   Atomic Mass: 20.0
Element: O    Atomic Mass: 16.0
Element: S    Atomic Mass: 32.0
Element: Si   Atomic Mass: 28.0
```

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/

[111]:
```
for k in ATOMIC_MASS:
    print(k)
```

```
H
He
C
N
O
Ne
Ar
S
Si
Fe
```

[112]:
```
ATOMIC_MASS.get('He')
```

[112]: 4

[113]:
```
for elem in sorted(ATOMIC_MASS, key = lambda k: ATOMIC_MASS[k]): # 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
```

[114]:
```
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
```

[115]:
```
for i, cosa in enumerate(['tralala', 5.5]):
    print(i, cosa)
```

```
0 tralala
1 5.5
```

[116]:
```
for idx, elem in enumerate(sorted(ATOMIC_MASS, key = ATOMIC_MASS.get)): #␣
↪adding an index that run from 0.
    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
```

[117]:
```
for i in range(10):
    if i > 5:
        print(i)
    # print('---')
print('final end')
```

```
6
```

```
7
8
9
final end
```

[118]:
```
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

[119]:
```
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]
```

[120]:
```
# more compact way to do the same thing
B = [i**2 for i in range(4)]
print(B)
```

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

[121]:
```
# 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': 25, 'squared_6': 36, 'squared_7': 49, 'squared_8': 64,
'squared_9': 81}
```

1.0.12 Functions, procedures

[122]:
```
def func1(x):
    print(x**3)
func1(5)
```

```
125
```

[123]:
```
def func2(x,
          y):
    """
    Return the cube and the 4th power of the two parameters
    """
    return(x**3, y**4)

help(func2)
```

```
Help on function func2 in module __main__:

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

[124]:
```
help(func1)
```

```
Help on function func1 in module __main__:

func1(x)
```

[125]:
```
#func2() shift-TAB inside the parenthesis
func2?
```

```
Signature: func2(x, y)
Docstring: Return the cube and the 4th power of the two parameters
File:      ~/Google Drive/Pro/Python-MySQL/Notebooks/Notebooks/
 ↪<ipython-input-123-627ae8d972bb>
Type:      function
```

[126]:
```
func2??
```

```
Signature: func2(x, y)
Source:
def func2(x,
          y):
    """
    Return the cube and the 4th power of the two parameters
    """
    return(x**3, y**4)
File:      ~/Google Drive/Pro/Python-MySQL/Notebooks/Notebooks/
 ↪<ipython-input-123-627ae8d972bb>
Type:      function
```

```
[127]: a = func2(3, 5)
```

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

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

```
[129]: 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
```

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

```
717.1067811865476
```

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

```
717.1067811865476
```

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

```
  File "<ipython-input-132-b28c04152297>", line 1
    G = func3(3, 4, 5, a=10, 100) # ERROR!
                            ^
SyntaxError: positional argument follows keyword argument
```

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

```
717.1067811865476
```

[134]:
```
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:

[135]:
```
J = lambda x, y, z: (x**2 + y**2 + z**2)**0.5
J(1,2,3)
```

[135]:
```
3.7416573867739413
```

[136]:
```
def J(x,y,z):
    return (x**2 + y**2 + z**2)**0.5
J(1,2,3)
```

[136]:
```
3.7416573867739413
```

[137]:
```
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):

[138]:
```
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:

[139]:
```
a = 5
def test_a(x):
    print(a * x)
test_a(5)
a = 10
test_a(5)
print(a)
```

```
25
50
10
```

[140]:
```
# 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:

[141]:
```
def test_g2():
    g2 = 5
    print('INSIDE:', g2)
```

```
test_g2()
print('OUTSIDE:', g2)
```

```
INSIDE: 5
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-141-8bc6850c733b> in <module>
      3     print('INSIDE:', g2)
      4 test_g2()
----> 5 print('OUTSIDE:', g2)

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

Global variable is known outside:

[142]:
```
def test_g3():
    global g3
    g3 = 5
    print('INSIDE:',g3)
test_g3()
print('OUTSIDE:', g3)
```

```
INSIDE: 5
OUTSIDE: 5
```

[143]:
```
global g4
def test_g3():
    g4 = 5
    print('INSIDE:',g4)
test_g3()
print('OUTSIDE:', g4)
```

```
INSIDE: 5
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-143-104373f91e38> in <module>
      4     print('INSIDE:',g4)
      5 test_g3()
----> 6 print('OUTSIDE:', g4)

NameError: name 'g4' is not defined
```

[144]:
```
def test_g3():
    g3 = 5
    print('INSIDE:',g3)
    return g3

g5 = test_g3()
print('OUTSIDE:', g5)
```

```
INSIDE: 5
OUTSIDE: 5
```

[145]:
```
def test_6():
    print('ok')
    return 'tradhbfwduyv', 10, 6.555e54
g10 = test_6()
print(g10)
```

```
ok
('tradhbfwduyv', 10, 6.555e+54)
```

Recursivity

[146]:
```
def fact(n):
    if n <= 0:
        return 1
    return n * fact(n-1) # using recursivity
print(fact(5))
print(fact(20))
print(fact(100))
```

```
120
2432902008176640000
93326215443944152681699238856266700490715968264381621468592963895217599993229915
608941463976156518286253697920827223758251185210916864000000000000000000000000
```

1.0.13 Scripting

[147]:
```
%%writefile ex1.py
# This write the current cell to a file
def f1(x):
    """
    This is an example of a function, returning x**2
    - parameter: x
    """
    return x**2
```

```
Overwriting ex1.py
```

[148]:
```
!cat ex1.py
```

```
# This write the current cell to a file
def f1(x):
    """
    This is an example of a function, returning x**2
    - parameter: x
    """
    return x**2
```

[149]:
```python
# %load ex1.py
# This write the current cell to a file
def f1(x):
    """
    This is an example of a function, returning x**2
    - parameter: x
    """
    return x**2
```

[150]:
```python
# This write the current cell to a file
def f1(x):
    """
    This is an example of a function, returning x**2
    - parameter: x
    """
    return x**2
```

[151]:
```python
f1(3)
```

[151]:
```
9
```

[152]:
```python
import ex1 #this imports a file named ex1.py from the current directory or
# from one of the directories in the search path
print(ex1.f1(4))
```

```
16
```

[153]:
```python
from ex1 import f1
print(f1(3))
```

```
9
```

[154]:
```python
from ex1 import * # DO NOT DO THIS! Very hard to know where f1 is comming from␣
 ↪(debuging, names conflicts)
print(f1(4))
```

```
16
```

[155]:
```python
import ex1 as tt
print(tt.f1(10))
```

```
100
```

[156]:
```
%run ex1 # The same as doing a copy-paste of the content of the file.
f1(8)
```

[156]:
```
64
```

[157]:
```
!pwd
```

```
/Users/christophemorisset/Google Drive/Pro/Python-MySQL/Notebooks/Notebooks
```

[158]:
```
!pydoc -w ex1 # ! used to call a Unix command
```

```
wrote ex1.html
```

[159]:
```
from IPython.display import HTML
HTML(open('ex1.html').read())
```

[159]:
```
<IPython.core.display.HTML object>
```

Help with TAB or ?

[160]:
```
f1?
```

```
Signature: f1(x)
Docstring:
This is an example of a function, returning x**2
- parameter: x
File:      ~/Google Drive/Pro/Python-MySQL/Notebooks/Notebooks/ex1.py
Type:      function
```

[161]:
```
help(f1)
```

```
Help on function f1 in module __main__:

f1(x)
    This is an example of a function, returning x**2
    - parameter: x
```

1.0.14 Importing libraries

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

[162]:
```
print(sin(3.))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-162-23e9b31fd5de> in <module>
----> 1 print(sin(3.))

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

[163]:
```
import math
print(math.sin(3.))
```

```
0.1411200080598672
```

[164]:
```
math?
```

```
Type:        module
String form: <module 'math' from '/Users/christophemorisset/anaconda3/lib/
 ↪python3.7/lib-dynload/math.cpython-37m-darwin.so'>
File:        ~/anaconda3/lib/python3.7/lib-dynload/math.cpython-37m-darwin.so
Docstring:
This module provides access to the mathematical functions
defined by the C standard.
```

[165]:
```
math.
```

```
  File "<ipython-input-165-6994845579ab>", line 1
    math.
         ^
SyntaxError: invalid syntax
```

[166]:
```
# We can import all the elements of the library in the current domain name (NOT␣
 ↪A GOOD IDEA!!!):
from math import *
sin(3.)
```

[166]:
```
0.1411200080598672
```

[167]:
```
# One can look at the contents of a library with dir:
print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10',
'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

[168]:
```
# 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(x, /)
    Return the sine of x (measured in radians).
```

[169]:
```
help(log)
```

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

log(…)
    log(x, [base=math.e])
    Return the logarithm of x to the given base.

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

[170]:
```
print(math.pi)
```

```
3.141592653589793
```

[171]:
```
math.pi = 2.71
```

[172]:
```
print(math.pi)
```

```
2.71
```

[173]:
```
import math
```

[174]:
```
math.pi
```

[174]:
```
2.71
```

[175]:
```
# In python 3 you need to import reload
from importlib import reload
```

[176]:
```
reload(math)
```

[176]:
```
<module 'math' from '/Users/christophemorisset/anaconda3/lib/python3.7/lib-
dynload/math.cpython-37m-darwin.so'>
```

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

```
[177]: 2.71
```

```
[178]: from math import pi as pa
```

```
[179]: pa
```

```
[179]: 2.71
```

```
[180]: math = 2
       math.pi
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-180-9de984e8800f> in <module>
      1 math = 2
----> 2 math.pi

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

```
[181]: pa
```

```
[181]: 2.71
```

```
[182]: math.sin = math.cos
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-182-6025a2d1ea5a> in <module>
----> 1 math.sin = math.cos

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

```
[183]: math.sin(3)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-183-6bafe0030c44> in <module>
----> 1 math.sin(3)

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

```
[184]: sin=3
```

```
[185]: sin(3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-185-2b5fc2868c51> in <module>
----> 1 sin(3)

TypeError: 'int' object is not callable
```