intro_Python

November 25, 2015

1 A Introduction to Python for dummies...

This is part of the Python lecture given by Christophe Morisset at IA-UNAM. More informations at: http://python-astro.blogspot.mx/

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 + 22
```

```
Out[1]: 24
```

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

```
Out[2]: 1
```

Python likes the use of spaces to make scripts more readable

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

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

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 [4]: a = 4
```

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

```
In [5]: a
```

```
Out[5]: 4
```

```
In [6]: a = a + 1
        a
```

```
Out[6]: 5
```

```
In [7]: a *= 4 # similar to a = a * 4
        a
```

```
Out[7]: 20
```

```
In [8]: a, b = 1, 3
        a, b
```

```
Out[8]: (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 [9]: lambda_ = 2
        file = 3
```

1.0.3 Comments

```
In [10]: a = 2 # this is a comment
```

```
In [11]: """ This is a large comment
         on multiple lines
         ending as it started
         """
```

```
Out[11]: ' 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.

```
In [12]: 2
```

```
Out[12]: 2
```

```
In [13]: 2 / 3 # Take care, this will result in an integer, may not be what you expect. This changes in
```

```
Out[13]: 0
```

```
In [14]: float(2) / 3 # This is the way exact division is performed, adding a dot to promote one of the
```

```
Out[14]: 0.6666666666666666
```

```
In [15]: from __future__ import division
         2 / 3
```

```
Out[15]: 0.6666666666666666
```

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

The function type gives the type of its argument:

```
In [16]: type(2)
```

```
Out[16]: int
```

```
In [17]: type(2.3)
```

```
Out[17]: float
```

```
In [18]: int(0.8) # truncating
```

```
Out[18]: 0
```

```
In [19]: round(0.8765566777) # nearest, result is float
```

```
Out[19]: 1.0
```

```
In [20]: int(round(0.88766)) # nearest, with the result being an integer:
```

```
Out[20]: 1
```

1.0.5 Complex numbers

```
In [21]: a = 1.5 + 0.5j

In [22]: a**2.

Out[22]: (2+1.5j)

In [23]: (1+2j)*(1-2j)

Out[23]: (5+0j)

In [24]: a.real

Out[24]: 1.5

In [25]: (a**3).imag

Out[25]: 3.25

In [26]: a.conjugate() # this is a function, it requieres ()

Out[26]: (1.5-0.5j)
```

1.0.6 Booleans

Comparison operators are ¡, ¿, ¡=, ¿=, ==, !=

```
In [27]: 5 < 7

Out[27]: True

In [28]: a = 5
         b = 7

In [29]: b < a

Out[29]: False

In [30]: c = 2

In [31]: c < a < b

Out[31]: True

In [32]: a < b and b < c

Out[32]: False

In [33]: res = a < 7
         print(res, type(res))

(True, <type 'bool'>)

In [34]: print int(res)
         print int(not res)

1
0

In [35]: not res is True

Out[35]: False

In [36]: a = True
         print a

True
```

1.0.7 Formating strings

```
In [37]: print "Hello world!"
```

```
Hello world!
```

```
In [38]: print 'Hello world!'
```

```
Hello world!
```

```
In [39]: print "Hello I'm here" # ' inside ""
```

```
Hello I'm here
```

```
In [40]: print('Hello') # this is the Python 3 style
```

```
Hello
```

```
In [41]: # 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 [42]: # The new way is using the format() method of the string object, and {} to define which value
         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/2/tutorial/inputoutput.html

1.0.8 Strings

```
In [43]: a = "this is a    string"
```

```
In [44]: len(a)
```

```
Out[44]: 19
```

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

```
In [45]: a.upper()
```

```
Out[45]: 'THIS IS A    STRING'
```

```
In [46]: a.title()
```

```
Out[46]: 'This Is A    String'
```

```
In [47]: a.split()
```

```
Out[47]: ['this', 'is', 'a', 'string']
```

```
In [48]: a.split()[1]

Out[48]: 'is'

In [49]: a = "This is a string.   With various sentences."

In [50]: a.split('.')

Out[50]: ['This is a string', '   With various sentences', '']

In [51]: a.split('.')[1].strip() # Here we define the character used to split. The default is space (an

Out[51]: 'With various sentences'

In [52]: a = 'tra'
         b = 'la'
         print ' '.join((a,b,b))
         print '-'.join((a,b,b))
         print ''.join((a,b,b))
         print ' '.join((a,b,b)).split()
         print ' & '.join((a,b,b)) + '\\\\'

tra la la
tra-la-la
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 [53]: L = ['red','green','blue'] # squared brackets are used to define lists

In [54]: type(L) # Print the type of L

Out[54]: list

In [55]: L[1]

Out[55]: 'green'

In [56]: L[0] # indexes start at 0 !!!

Out[56]: 'red'

In [57]: L[-1] # last element

Out[57]: 'blue'

In [58]: L[-3]

Out[58]: 'red'

In [59]: L = L + ['black', 'white'] # addition symbol is used to agregate values to a list. See below o

In [60]: print L

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

```
In [61]: L[1:3] # L[start:stop] : elements if index i, where start <= i < stop !! stop not included !!

Out[61]: [’green’, ’blue’]

In [62]: L[2:] # boudaries can be omited

Out[62]: [’blue’, ’black’, ’white’]

In [63]: L[-2:]

Out[63]: [’black’, ’white’]

In [64]: L[::2] # L[start:stop:step] every 2 elements

Out[64]: [’red’, ’blue’, ’white’]

In [65]: L[::-1]

Out[65]: [’white’, ’black’, ’blue’, ’green’, ’red’]
```

Lists are mutable: their content can be modified.

```
In [66]: L[2] = ’yellow’
         L

Out[66]: [’red’, ’green’, ’yellow’, ’black’, ’white’]

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

Out[67]: [’red’, ’green’, ’yellow’, ’black’, ’white’, ’pink’]

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

Out[68]: [’red’, ’green’, ’blue’, ’yellow’, ’black’, ’white’, ’pink’]

In [69]: L.extend([’magenta’, ’purple’])
         L

Out[69]: [’red’,
          ’green’,
          ’blue’,
          ’yellow’,
          ’black’,
          ’white’,
          ’pink’,
          ’magenta’,
          ’purple’]

In [70]: L.append([’magenta’, ’azul’])
         L

Out[70]: [’red’,
          ’green’,
          ’blue’,
          ’yellow’,
          ’black’,
          ’white’,
          ’pink’,
          ’magenta’,
          ’purple’,
          [’magenta’, ’azul’]]
```

```
In [71]: L.append(2)
         L
```

```
Out[71]: ['red',
          'green',
          'blue',
          'yellow',
          'black',
          'white',
          'pink',
          'magenta',
          'purple',
          ['magenta', 'azul'],
          2]
```

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

```
Out[72]: [2,
          ['magenta', 'azul'],
          'purple',
          'magenta',
          'pink',
          'white',
          'black',
          'yellow',
          'blue',
          'green',
          'red']
```

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

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

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

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

        <ipython-input-74-c16babb9288f> in <module>()
    ----> 1 L[25] # Out of range leads to error


        IndexError: list index out of range
```

```
In [75]: 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', 'red
[]
[]
['purple', 'magenta', 'pink', 'white', 'black', 'yellow', 'blue', 'green', 'red']
```

```
In [76]: print L.count('yellow')
```

```
1
```

```
In [77]: L.sort() # One can use TAB to look for the methods (functions that apply to an object)
         print L
```

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

```
In [78]: a = [1,2,3]
         b = [10,20,30]
```

```
In [79]: print(a+b) # may not be what you expected, but rather logical too
```

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

```
In [80]: print(a*b) # Does NOT multiply element by element. Numpy will do this job.
```

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

        <ipython-input-80-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 [81]: L = range(4) # Create a list. Notice the parameter is the number of elements, not the last one
         L
```

```
Out[81]: [0, 1, 2, 3]
```

```
In [82]: L = range(2, 20, 2) # every 2 integer
         L
```

```
Out[82]: [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

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

```
In [83]: L = [1, '1', 1.4]
         L
```

```
Out[83]: [1, '1', 1.4]
```

Remove the n+1-th element:

```
In [84]: L = 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 [85]: 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
```

```
In [86]: print(a[1,1]) # Does NOT work
```

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

        <ipython-input-86-d8214b6adea8> in <module>()
    ----> 1 print(a[1,1]) # Does NOT work


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

```
In [87]: b = a[1]
         print b
```

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

```
In [88]: b[1] = 999
         print b
```

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

```
In [89]: print a # Changing b changed a !!!
```

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

```
In [90]: b[1] is a[1][1]
```

```
Out[90]: True
```

```
In [91]: 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 [92]: T = (1,2,3)
         T

Out[92]: (1, 2, 3)

In [93]: T2 = 1, 2, 3
         print T2
         type(T2)

(1, 2, 3)

Out[93]: tuple

In [94]: T[1]

Out[94]: 2
```

tuples are unmutables

```
In [95]: T[1] = 3 # Does NOT work!


        ---------------------------------------------------------------------------
    TypeError                                 Traceback (most recent call last)

        <ipython-input-95-6dd68cc28786> 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 [96]: D = {'Christophe': 12, 'Antonio': 15} # defined by {key : value}

In [97]: D['Christophe'] # access to a value by the key

Out[97]: 12

In [98]: D.keys() # list of the dictionary keys

Out[98]: ['Christophe', 'Antonio']

In [99]: D['Julio'] = 16 # adding a new entry

In [100]: print D

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

1.0.10 Blocks

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

```
In [101]: for i in [1,2,3]: print(i) # compact way, not recomended.
```

```
1
2
3
```

```
In [102]: 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 [103]: # 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 wan
          for key in ATOMIC_MASS.keys():
              print key, ATOMIC_MASS[key]
```

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

```
In [104]: 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 [105]: 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 [106]: for idx, elem in enumerate(sorted(ATOMIC_MASS, key = ATOMIC_MASS.get)): # adding an index tha
              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 [107]: for i in range(10):
              if i > 5:
                  print i
```

```
6
7
8
9
```

```
In [108]: 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 [109]: 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 [110]: # more compact way to do the same thing
          B = [i**2 for i in range(4)]
          print B
```

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

```
In [111]: # The same is also used for dictionnaries
          D = {'squared_{}'.format(k) : k**2 for k in range(10)}
          print D
```

```
{’squared_3’: 9, ’squared_2’: 4, ’squared_1’: 1, ’squared_0’: 0, ’squared_7’: 49, ’squared_6’: 36, ’squar
```

1.0.12 Functions, procedures

```
In [112]: def func1(x):
              print(x**3)
          func1(5)
```

```
125
```

```
In [113]: def func2(x):
              """
              Return the cube of the parameter
              """
              return(x**3)
          a = func2(3)

          help(func2)
```

```
Help on function func2 in module __main__:

func2(x)
    Return the cube of the parameter
```

```
In [115]: #func2() shift-TAB inside the parenthesis
          func2?

In [116]: print(a)
          print(func2(4))
```

```
27
64
```

```
In [117]: 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
```

```
7.07106781187
```

```
In [118]: E = func3(3, 4, 5, 10, 100)
          print E
```

```
717.106781187
```

```
In [119]: F = func3(x=3, y=4, z=5, a=10, b=100)
          print F
```

```
717.106781187
```

```
In [120]: G = func3(3, 4, 5, a=10, 100) # ERROR!
          print G
```

```
          File "<ipython-input-120-a2bc66692446>", line 1
        G = func3(3, 4, 5, a=10, 100) # ERROR!
    SyntaxError: non-keyword arg after keyword arg
```

```
In [121]: H = func3(3, 4, 5, a=10, b=100)
          print H
```

```
717.106781187
```

```
In [122]: I = func3(z=5, x=3, y=4, a=10, b=100) # quite risky!
          print I
```

```
717.106781187
```

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

```
In [123]: J = lambda x, y, z: (x**2 + y**2 + z**2)**0.5
          J(1,2,3)

Out[123]: 3.7416573867739413

In [124]: 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 [125]: 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 [126]: a = 5
          def test_a(x):
              print a*x
          test_a(5)
          a = 10
          test_a(5)
          print(a)
```

```
25
50
10
```

```
In [127]: # 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 [128]: def test_g2():
              g2 = 5
              print g2
          test_g2()
          print g2
```

```
5


        ---------------------------------------------------------------------------
    NameError                                 Traceback (most recent call last)

        <ipython-input-128-f60224a7598e> in <module>()
          3     print g2
          4 test_g2()
    ----> 5 print g2


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

Global variable is known outside:

```
In [129]: def test_g3():
              global g3
              g3 = 5
              print g3
          test_g3()
          print g3
```

```
5
5
```

Recursivity

```
In [130]: def fact(n):
              if n <= 0:
                  return 1
              return n*fact(n-1)
          print(fact(5))
          print(fact(20))
          print(fact(100))
```

```
120
2432902008176640000
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253
```

1.0.13 Scripting

```
In [131]: %%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
```

```
In [132]: !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
```

```
In [133]: # load a file in the next cell. Usefull for small scripts.
          %load ex1.py
```

```
In [ ]: # 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
```

```
In [134]: # 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
```

```
In [135]: 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
```

```
In [136]: from ex1 import f1
          print f1(3)
```

```
9
```

```
In [137]: from ex1 import * # DO NOT DO THIS! Very hard to know where f1 is comming from (debuging, nam
          print f1(4)
```

```
16
```

```
In [138]: import ex1 as tt
          print tt.f1(10)
```

```
100
```

```
In [139]: %run ex1 # The same as doing a copy-paste of the content of the file.
          f1(8)
```

```
Out[139]: 64
```

```
In [140]: !pwd
```

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

```
In [141]: !pydoc -w ex1 # ! used to call a Unix command
```

```
wrote ex1.html
```

```
In [142]: from IPython.display import HTML
          HTML(open('ex1.html').read())
```

```
Out[142]: <IPython.core.display.HTML at 0x107777050>
```

Help with TAB or ?

```
In [143]: f1?
```

```
In [144]: 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:

```
In [145]: print sin(3.)
```

```
0.14112000806
```

```
In [146]: import math
          print math.sin(3.)
```

```
0.14112000806
```

```
In [147]: math?
```

```
In [148]: math.
```

```
      File "<ipython-input-148-186ff497df9b>", line 1
    math.
         ^
SyntaxError: invalid syntax
```

```
In [149]: # We can import all the elements of the library in the current domain name (NOT A GOOD IDEA!!
          from math import *
          sin(3.)

Out[149]: 0.1411200080598672
```

```
In [150]: # One can look at the contents of a library with dir:
          print(dir(math))

['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh
```

```
In [151]: # 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 [152]: 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 [153]: print math.pi

3.14159265359

In [154]: math.pi = 2.71

In [155]: print math.pi

2.71

In [156]: import math

In [157]: math.pi

Out[157]: 2.71

In [158]: reload(math)

Out[158]: <module 'math' from '/Users/christophemorisset/Ureka/variants/common/lib/python2.7/lib-dynloa
```

```
In [159]: math.pi

Out[159]: 3.141592653589793

In [160]: from math import pi as pa

In [161]: pa

Out[161]: 3.141592653589793

In [162]: math = 2
          math.pi


        ---------------------------------------------------------------------------
    AttributeError                            Traceback (most recent call last)

        <ipython-input-162-70a02d6227fb> in <module>()
          1 math = 2
    ----> 2 math.pi


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


In [163]: pa

Out[163]: 3.141592653589793
```