Introduction to Python: basic elements I

This is an introduction to the Python language. The idea is to learn by trying, with problem of increasing diffculty (and interest)

First of all, we need to move to the directory where we put the data of the Bk.zip file provided with the course

In [3]:
```
import os
# Insert here your directory. This works for Linux and OSX
os.chdir("/home/gf/src/Python/Python-in-the-lab/Bk")
# for Windows you should put "\" instead of "/", i.e.
# os.chdir("c:\\yourdirectory\your_sub_dir\Bk")
# Now we check the directory we are in
os.path.abspath(".")
```

Out[3]:
```
'/home/gf/src/Python/Python-in-the-lab/Bk'
```

In [4]:
```
# Be carefull: python is case sensitive!
import OS
# This is your first error. Many others will come!
# Very often you understand from the error what happened
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-4-10af4188258c> in <module>
      1 # Be carefull: python is case sensitive!
----> 2 import OS
      3 # This is your first error. Many others will come!
      4 # Very often you understand from the error what happened

ModuleNotFoundError: No module named 'OS'
```

This directory contains a lot of (old) data, and we would like to plot one of files. We need first to manipulate the filenames, so to learn about the strings. Then we need to load the file, and plot. It takes a while, by the way... Let's start with this file.

In [5]:
```
filename = "F64ac_0.02_T.dat" # No previous definition of the filename varia
ble
```

In [6]:
```
# Let's check the type of this variable
type(filename)
```

Out[6]:
```
str
```

What can I do with a string? Many things, of course. Some operations are intrinsically linked to type of variable you are using. These are called Methods. Try to add a "." after filename, like this, and press "tab"

```
filename.TAB
```

In [5]:
```
filename.count
```

Out[5]:
```
<function str.count>
```

In [6]:
```
# Search how many time a string occurs
filename.count("0.02")
```

Out[6]: 1

In [7]:
```
filename.count("0")
```

Out[7]: 2

In [7]:
```
# Find the position of a string
filename.find("0.02")
```

Out[7]: 6

oh, is it right? The first "0" is at the 7th position! Python starts from zero. Remind it!

In [8]:
```
print(filename[6:]) # 6: means "from element 6 on"
filename[6:-4] # means from 6th element to the 4th element to the end
```

```
0.02_T.dat
```

Out[8]: '0.02_T'

In [8]:
```
# Other methods: explore yourself
print(filename.upper())
print(filename.islower()) # We will check this variable soon
print(filename.lower())
```

```
F64AC_0.02_T.DAT
False
f64ac_0.02_t.dat
```

A string is numerable

In [12]:
```
# If an object is numerable...
# ... you can get a single element using the square brakets and the position
print(filename[3])
print("-"*40)
# ... you get all the elements using a loop
for element in filename:
    print(element)
```

```
a
----------------------------------------
F
6
4
a
c

_
0
.
0
2

_
T
.
d
a
t
```

A string is immutable

In [13]:
```
# i.e. you cannot change one of its elements
filename[6] = "2"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-015add3d6490> in <module>
      1 # i.e. you cannot change one of its elements
----> 2 filename[6] = "2"

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

How can we extract information encoded in a string?

This filename, for instance, is made as follows: Material_Frequency_TypeOfData.dat

How do we extract the frequency?

In [12]:
```
# Use the split method with an arbitrary element
filename.split("_")
```

Out[12]:
```
['F64ac', '0.02', 'T.dat']
```

In [15]:
```
# What is ths "[]"? Let's investigate
l = filename.split("_")
type(l)
```

Out[15]:
```
list
```

I guess a list has some methods, does it?

In [16]:
```
print(l[1:]) # Numerable
print(l[:-1]) # What is this???
l.append("something")
print(l)
l.append(43)
print(l)
l.pop()
print(l)
```

```
['0.02', 'T.dat']
['F64ac', '0.02']
['F64ac', '0.02', 'T.dat', 'something']
['F64ac', '0.02', 'T.dat', 'something', 43]
['F64ac', '0.02', 'T.dat', 'something']
```

Something to say?

Something you learn by using the method and you understand. Some other time is better to ask for kelp.

In [17]:
```
l.pop?
```

In [25]:
```
out = filename[:-4].split("_")
out.append("nice!")
out.append(3)
out
```

Out[25]: ['F64ac', '0.02', 'T', 'nice!', 3]

In [26]:
```
out.append(out)
out
# the result depends on how many times you run it!
```

Out[26]: ['F64ac', '0.02', 'T', 'nice!', 3, [...]]

In [27]:
```
out2 = out.pop()
out2
```

Out[27]: ['F64ac', '0.02', 'T', 'nice!', 3]

Back to the filename: there is also a smarter way to split the name from the extension

In [30]:
```
os.path.splitext(filename)
```

Out[30]: ('F64ac_0.02_T', '.dat')

In [31]:
```
# oh, now "()" ??? What is it?
q = os.path.splitext(filename)
print(q, type(q))
```

('F64ac_0.02_T', '.dat') <class 'tuple'>

In [66]:
```
# So it splits the basename and the extension into a tuple. So can I...
basename, extension = os.path.splitext(filename)
```

In [67]:
```
# Nice, no need of parenthesis
basename # Yes!
```

Out[67]: 'F64ac_0.02_T'

In [68]:
```
extension
```

Out[68]: '.dat'

In [69]:
```
# So finally I can split the base name and take the second element for the f
requency
# Even better I can immediately get the frequency as
material, freq, measure = basename.split("_")
print(freq)
```

0.02

In [70]:
```
# Or even...
material, freq, measure = os.path.splitext(filename)[0].split("_")
freq # This is still a string!
```

Out[70]: '0.02'

Something about tuples

A tuple is immutable, but numerable

In [71]:
```
q = ('a', 'b')
type(q)
```

Out[71]:
```
tuple
```

In [72]:
```
q[0]
```

Out[72]:
```
'a'
```

In [73]:
```
q[0] = 'c'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-73-28edba4ec4fe> in <module>
----> 1 q[0] = 'c'

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

Formatting methods for strings

How to we construct a string?

In [74]:
```
# Formatting methods can be found here:
# https://docs.python.org/3.7/whatsnew/2.6.html#pep-3101-advanced-string-for
matting
# There are two versions
n = 326
print("The sqrt of {x:n} is {y:.5f}".format(x=n, y=n**.5)) # From 2.7 and 3.
5
# or
print("The sqrt of {0:n} is {1:.5f}".format(n, n**.5)) # From 2.7 and 3.5
#
print("The sqrt of %i is %.5f" % (n, n**.5))
```

```
The sqrt of 326 is 18.05547
The sqrt of 326 is 18.05547
The sqrt of 326 is 18.05547
```

Problem n. 1

Construct the string: "F64ac_0.02_T_2.dat" starting from filename

In [75]:
```
# Solution n. 1
# Not very pythonic
filename2 = filename[:-4] + "_2" + filename[-4:]
filename2
```

Out[75]:
```
'F64ac_0.02_T_2.dat'
```

In [76]:
```
# Solution n. 2
# Not bad, but not very general
filename2 = filename.replace("T", "T_2")
filename2
```

Out[76]: 'F64ac_0.02_T_2.dat'

In [77]:
```
# Solution n. 3
"%s_2%s" % os.path.splitext(filename)
# Hmmm, a little better
```

Out[77]: 'F64ac_0.02_T_2.dat'

In [78]:
```
# Solution n. 3b
basename, ext = os.path.splitext(filename)
"%s_%s%s" % (basename, 2, ext) # the 3 elements have to stay in a tuple
```

Out[78]: 'F64ac_0.02_T_2.dat'

In [79]:
```
# Even better
print("{0}_{1}{2}".format(basename, 2, extension)) # In Python 3.X
print("{}_{}{}".format(basename, 2, extension)) # This works as well
```

```
F64ac_0.02_T_2.dat
F64ac_0.02_T_2.dat
```

In general we can have thousand of files, so we need to write the number "2" with some leading zeros, so to preserve the order of the filenames: "F64ac_0.02_T_0002.dat"

In [80]:
```
print("{0}_{1}{2}".format(basename, "2".rjust(4, "0"), extension))
# Even simpler
print("{0}_{1:0>4}{2}".format(basename, "2", extension))
# But also this works
print("{0}_{1:0>4}{2}".format(basename, 2, extension))
```

```
F64ac_0.02_T_0002.dat
F64ac_0.02_T_0002.dat
F64ac_0.02_T_0002.dat
```

Now we want the filenames from 0000 to 1000 with step 100, preserving the order.

Explore the use of range

In [81]:
```
for i in range(0,1100,100):
    print("{0}_{1:0>4}{2}".format(basename, i, ext))
```

```
F64ac_0.02_T_0000.dat
F64ac_0.02_T_0100.dat
F64ac_0.02_T_0200.dat
F64ac_0.02_T_0300.dat
F64ac_0.02_T_0400.dat
F64ac_0.02_T_0500.dat
F64ac_0.02_T_0600.dat
F64ac_0.02_T_0700.dat
F64ac_0.02_T_0800.dat
F64ac_0.02_T_0900.dat
F64ac_0.02_T_1000.dat
```

Problem 2

Let's do it a little more general, between n_min and n_max, with n_step

How to change the code above? (You can do it)

In [84]:
```
n_min, n_max, n_step = 0, 1000, 1000
n_car = len(str(n_max))
for i in range(n_min, n_max+n_step, n_step):
     print("{0}_{1:0>n_car}{2}".format(basename, i, extension)) # This does
not work!
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-84-d938317cfb27> in <module>
      2 n_car = len(str(n_max))
      3 for i in range(n_min, n_max+n_step, n_step):
----> 4      print("{0}_{1:0>n_car}{2}".format(basename, i, extension)) # Thi
s does not work!

ValueError: Invalid format specifier
```

In [85]:
```
n_min, n_max, n_step = 0, 270, 30
n_car = len(str(n_max))
for i in range(n_min, n_max+n_step, n_step):
    print("{0}_{1}{2}".format(basename, str(i).rjust(n_car, "0"), extension)
)
```

```
F64ac_0.02_T_000.dat
F64ac_0.02_T_030.dat
F64ac_0.02_T_060.dat
F64ac_0.02_T_090.dat
F64ac_0.02_T_120.dat
F64ac_0.02_T_150.dat
F64ac_0.02_T_180.dat
F64ac_0.02_T_210.dat
F64ac_0.02_T_240.dat
F64ac_0.02_T_270.dat
```

Ok, I know we have not yet opened the file...

In [ ]:
```
```