IPSL python tutorial: some exercises for beginners

WARNING!

WARNING! This is the FULL version of the tutorial (including the solutions)

WARNING!

Jean-Yves Peterschmitt - LSCE

October 2014

Documents

These exercises are based on the *python_intro_ipsl_oct2013.pdf* tutorial that you can download from the following pages

- http://www.lsce.ipsl.fr/Phocea/Cours/index.php?uid=jean-yves.peterschmitt
- http://www.lmd.polytechnique.fr/~dkhvoros/training.html

You should also download the following useful pdf files:

- Python 2.7 Quick Reference

  http://rgruet.free.fr/PQR27/PQR2.7_printing_a4.pdf
- Official Python Tutorial (*tutorial.pdf*)

  Official Python Library Reference (*library.pdf*)

  Both pdf files are available in the following archive, on the Python web site

  http://docs.python.org/2.7/archives/python-2.7.5-docs-pdf-a4.zip

Notes

- This document is an *ipython notebook*. It can be opened and (re)played in ipython (start '**ipython notebook**' and open the notebook from the browser interface), or the commands can just be typed in a regular python or ipython interpreter.
- In a python interpreter (in interactive mode), the value of a variable can be printed by just typing the name of the variable (and the *Enter* key), or with the *print* command. The behavior is subtly different in the ipython notebook, so we sometimes use *print* below, when it gives more useful output
- The most useful ipython notebook shorcuts that you need to know in this tutorial are
  - *Shift-Enter*: run cell
  - *Ctrl-Enter*: run cell in-place

  You can display the other available shortcuts by typing: *Ctrl-m h*

Playing with strings (and objects, indices, loops)

Create a string named **ipsl** with the following content:

Institut Pierre Simon Laplace

In [47]:
```
ipsl = 'Institut Pierre Simon Laplace'
```

Display the type of the string object with **type()**

In [48]:
```
type(ipsl)
```

Out[48]:
```
str
```

Determine the length of the string

In [49]:
```
len(ipsl)
```

Out[49]:
```
29
```

Try to access the 40th character of the string and look at the error that is generated

In [50]:
```
ipsl[40]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-50-175bfd4e069e> in <module>()
----> 1 ipsl[40]

IndexError: string index out of range
```

Extract the first character of the string

In [51]:
```
ipsl[0]
```

Out[51]:
```
'I'
```

Use 2 different ways to extract the last character of the string

Hint: use a positive and a negative index

In [52]:
```
ipsl[len(ipsl)-1]
```

Out[52]:
```
'e'
```

In [53]:
```
ipsl[-1]
```

Out[53]:
```
'e'
```

Use indices to display the full string

In [54]:
```
ipsl[0:29]
```

Out[54]:
```
'Institut Pierre Simon Laplace'
```

In [55]:
```
ipsl[0:len(ipsl)]
```

Out[55]:
```
'Institut Pierre Simon Laplace'
```

Use indices to display every 3rd character of the string

In [56]:
```
ipsl[0:29:3] # Use explicit index values
```

Out[56]:
```
'ItuPr m pc'
```

In [57]:
```
ipsl[0::3] # Use implicit end of the string
```

Out[57]:
```
'ItuPr m pc'
```

In [58]:
```
ipsl[::3] # Use implicit beginning and end of the string
```

Out[58]:
```
'ItuPr m pc'
```

Use **help()** on the **find** method of the string

Note: help on *help* (in a regular python interpreter): *space*: next screen, *b*: back one screen, *q*:quit, */*: search

In [59]:
```
help(ipsl.find)
```

```
Help on built-in function find:

find(...)
    S.find(sub [,start [,end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.
```

Use 2 different ways to extract the last word of the **ipsl** string and store it in a new **lap_str** string

*Hint: first use find and indices, then use the **split** method of the string*

In [60]:
```
ipsl.find('Laplace')
```

Out[60]:
```
22
```

In [61]:
```
lap_str = ipsl[22:29]
print lap_str
lap_str = ipsl[ipsl.find('Laplace'):]
print lap_str
```

```
Laplace
Laplace
```

In [62]:
```
help(ipsl.split)
```

```
Help on built-in function split:

split(...)
    S.split([sep [,maxsplit]]) -> list of strings

    Return a list of the words in the string S, using sep as the
    delimiter string.  If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator and empty strings are removed
    from the result.
```

In [63]:
```
ipsl.split()
```

Out[63]:
```
['Institut', 'Pierre', 'Simon', 'Laplace']
```

In [64]:
```
lap_str = ipsl.split()[-1]
print lap_str
```

```
Laplace
```

Use **help()** to determine how the python built-in **range** function works

In [65]:
```
help(range)
```

```
Help on built-in function range in module __builtin__:

range(...)
    range([start,] stop[, step]) -> list of integers

    Return a list containing an arithmetic progression of integers.
    range(i, j) returns [i, i+1, i+2, ..., j-1]; start (!) defaults to 0.
    When step is given, it specifies the increment (or decrement).
    For example, range(4) returns [0, 1, 2, 3].  The end point is omitted!
    These are exactly the valid indices for a list of 4 elements.
```

Use **range** to generate a list of integers going from 0 to 8

In [66]:
```
range(9)
```

Out[66]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

Use **range** to generate a list of as many integers as there are letters in the last word of the **ipsl** string

In [67]:
```
range(len(lap_str))
```

Out[67]:
```
[0, 1, 2, 3, 4, 5, 6]
```

Use 2 different ways to revert the caracters of the last word of **ipsl**

*Hint: first use a **for** loop, then use just a slice operation with the apropriate indices*

In [68]:
```
l_revert = ''
for i in range(len(lap_str)):
    l_revert += lap_str[-1 - i] # Same as l_revert = l_revert + lap_str[-1 - i]
print l_revert
```

```
ecalpaL
```

In [69]:
```
lap_str[::-1]
```

Out[69]:
```
'ecalpaL'
```

Use **dir()** on the **ipsl** string object and find a way to convert it to uppercase characters

In [70]:
```
print dir(ipsl)
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__',
'__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '_formatter_field_name_split',
'_formatter_parser', 'capitalize', 'center', 'count', 'decode', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'index', 'isalnum', 'isalpha',
'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust',
'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',
'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

In [71]:
```
help(ipsl.upper)
```

```
Help on built-in function upper:

upper(...)
    S.upper() -> string

    Return a copy of the string S converted to uppercase.
```

In [72]:
```
ipsl.upper()
```

Out[72]:
```
'INSTITUT PIERRE SIMON LAPLACE'
```

Using lists to experiment with python subtleties

Use the **split** method of the **ipsl** string to create an **ipsl_words** list variable (4 strings with the individual words of IPSL), and display **ipsl_words**

In [73]:
```
ipsl_words = ipsl.split()
ipsl_words
```

Out[73]:
```
['Institut', 'Pierre', 'Simon', 'Laplace']
```

Create 2 *copies* of ipsl_words with **ipsl_pnt = ipsl_words** (copy the *reference*) and **ipsl_cp = ipsl_words[:]** (copy the *values*) and display all the lists by typing:

ipsl_words, ipsl_pnt, ipsl_cp

In [74]:
```
ipsl_pnt = ipsl_words
ipsl_cp = ipsl_words[:]
ipsl_words, ipsl_pnt, ipsl_cp
```

Out[74]:
```
(['Institut', 'Pierre', 'Simon', 'Laplace'],
 ['Institut', 'Pierre', 'Simon', 'Laplace'],
 ['Institut', 'Pierre', 'Simon', 'Laplace'])
```

Assign a new value *'Bob'* to the 2nd string of **ipsl_pnt**, and the value *'Bill'* to the 3rd string of **ipsl_cp**, and display the 3 lists again

In [75]:
```
ipsl_pnt[1] = 'Bob'
ipsl_cp[2] = 'Bill'
ipsl_words, ipsl_pnt, ipsl_cp
```

Out[75]:
```
(['Institut', 'Bob', 'Simon', 'Laplace'],
 ['Institut', 'Bob', 'Simon', 'Laplace'],
 ['Institut', 'Pierre', 'Bill', 'Laplace'])
```

Congratulations, you have just learned the subtle difference between having 2 variables that *point to the same object* in memory (**ipsl_words** and **ipsl_pnt** point to the same list), and using the *copy* of a variable (**ipsl_cp**)!

Just to be sure, replace the 4th value of **ipsl_words** with the string **'LAPLACE'** (all uppercase characters), and display again the 3 lists

In [76]:
```
ipsl_words[3] = 'LAPLACE'
ipsl_words, ipsl_pnt, ipsl_cp
```

Out[76]:
```
(['Institut', 'Bob', 'Simon', 'LAPLACE'],
 ['Institut', 'Bob', 'Simon', 'LAPLACE'],
 ['Institut', 'Pierre', 'Bill', 'Laplace'])
```

Import the **copy** module and have a quick look at the built-in documention of the module with **help()**

In [77]:
```
import copy
# The output of help(copy) below is a bit too long for printing...
# When you have opened this notebook, uncomment the line below
# and execute the cell
# help(copy)
```

Display only the help of the **copy** function of the **copy** module (e.g. **copy.copy()**)

In [78]:
```
help(copy.copy)
```

```
Help on function copy in module copy:

copy(x)
    Shallow copy operation on arbitrary Python objects.

    See the module's __doc__ string for more info.
```

Notes:

- It's usually enough to copy lists with a *slicing* operation (**my_list[:]** or **my_list[start:end]**). There no need to use the **copy** module when there is an easier way to make a copy (many objects provide a built-in method for copying them)!
- If you ever need more information about the difference between *shallow* and *deep* copy (**copy.deepcopy**), you can check the following section of **library.pdf**: *8.17 copy — Shallow and deep copy operations*
- There are lots of cases when it's a good thing to avoid uselessly copying objects (e.g. BIG data arrays)!
- You should not worry too much about the *reference/copy* choice (what happens by default is usually what you want), and you just need to be aware that this can sometimes cause side-effects

Copy **ipsl_cp** to **ipsl_cp_2** and display the 2 lists

In [79]:
```
ipsl_cp_2 = ipsl_cp[:]
ipsl_cp, ipsl_cp_2
```

Out[79]:
```
(['Institut', 'Pierre', 'Bill', 'Laplace'],
 ['Institut', 'Pierre', 'Bill', 'Laplace'])
```

Use the built-in **sorted()** function of python on the **ipsl_cp** list, and the **sort()** method of the **ipsl_cp_2** list, then display the 2 lists again

In [80]:
```
print 'This is what we get with sorted:', sorted(ipsl_cp)
print 'This is what we get with sort():', ipsl_cp_2.sort()
ipsl_cp, ipsl_cp_2
```

```
This is what we get with sorted: ['Bill', 'Institut', 'Laplace', 'Pierre']
This is what we get with sort(): None
```

Out[80]:
```
(['Institut', 'Pierre', 'Bill', 'Laplace'],
 ['Bill', 'Institut', 'Laplace', 'Pierre'])
```

Warning! What happened is that the 1st way of sorting created a *sorted copy* of **ipsl_cp** (without altering **ipsl_cp**) and the 2nd way of sorting *directly sorted the original* **ipsl_cp_2** list, without returning a result (this is called an *in place operation*). *In-place* operations can have side-effects if they change an object, but you don't know about it :-) Luckily, the documentation mentions this sort of behavior!

Display (and read!) the help of the **sort** method of the **ipls_cp** list

In [81]:
```
help(ipsl_cp.sort)
```

```
Help on built-in function sort:

sort(...)
    L.sort(cmp=None, key=None, reverse=False) -- stable sort *IN PLACE*;
    cmp(x, y) -> -1, 0, 1
```

More experiments with loops

Use 2 different kinds of loops to print the words of **ipsl_words**

Hint: you can either loop on a list of indices, or directly on the elements of the list

In [82]:
```
for i in range(len(ipsl_words)):
    print ipsl_words[i]
```

```
Institut
Bob
Simon
LAPLACE
```

In [83]:
```
for w in ipsl_words:
    print w
```

```
Institut
Bob
Simon
LAPLACE
```

Use **enumerate** to loop on both the *indices* AND the *values* of **ipsl_words**

*Hint: look for **Looping Techniques** in **tutorial.pdf***

In [84]:
```
for i, w in enumerate(ipsl_words):
    print i, w
```

```
0 Institut
1 Bob
2 Simon
3 LAPLACE
```

Use the following formatted print in the *enumerate* loop to get a nicer output, where:

- **i** is the variable that loops on the indices
- **w** is the variable that loops on the words

print 'The word at index %03i is [%15s]' % (i, w)

*Note: more information about formats is available in the **String Formatting Operations** section of **library.pdf***

In [85]:
```
for i, w in enumerate(ipsl_words):
    print 'The word at index %03i is [%15s]' % (i, w)
```

```
The word at index 000 is [       Institut]
The word at index 001 is [            Bob]
The word at index 002 is [          Simon]
The word at index 003 is [        LAPLACE]
```

Use ONE line to store each word of **ipsl_words** in individual **I**, **P**, **S** and **L** variables. Print the **I** and **L** variables

*Hint: look for **unpack** in **PQR2.7_printing_a4.pdf***

In [86]:
```
I, P, S, L = ipsl_words
print I, L
```

```
Institut LAPLACE
```

Use an **if** test and a **break** command in one of the previous loops to exit the loop when you have reached the word defined in the **S** string

WARNING! Remember that you have to use **'=='** (and not just a single **'='** sign) to test the equality of variables!

In [87]:
```
for w in ipsl_words:
    if w == S:
        break
    print w
```

```
Institut
Bob
```

WARNING! Always think and be careful before using BIG lists/loops/objects.

Open another terminal (or the *Task Manager* if you are using Windows), and start monitoring your processes by using **top** (then type *u*, then your login, to display only your processes).

Then make a loop on **range(50000000)** and print the index every 10000000 loops. Python will first create a BIG temporary list of 50000000 integers, then loop over it. Carefully monitor the memory usage of your process in the **top** terminal window

*Hint: look for **modulo** in **PQR2.7_printing_a4.pdf** and use it in order to print the index only every 10000000 loops*

In [88]:
```
for i in range(50000000):
    if i % 10000000 == 0:
        print i
```

```
0
10000000
20000000
30000000
40000000
```

Make the same loop over **xrange(50000000)** and keep monitoring the memory usage of your process. It is faster and it does not use any extra memory because the indices are generated on the fly

In [89]:
```
for i in xrange(50000000):
    if i % 10000000 == 0:
        print i
```

```
0
10000000
20000000
30000000
40000000
```