IPSL python tutorial: some exercises for beginners

WARNING!

WARNING! This is the version of the tutorial that does NOT include 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 [ ]:
```

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

```
In [ ]:
```

Determine the length of the string

```
In [ ]:
```

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

```
In [ ]:
```

Extract the first character of the string

```
In [ ]:
```

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

Hint: use a positive and a negative index

```
In [ ]:
```

```
In [ ]:
```

Use indices to display the full string

```
In [ ]:
```

```
In [ ]:
```

Use indices to display every 3rd character of the string

```
In [ ]: # Use explicit index values
```

```
In [ ]: # Use implicit end of the string
```

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

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 [ ]:
```

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 [ ]:
```

```
In [ ]:
```

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

```
In [ ]:
```

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

```
In [ ]:
```

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

```
In [ ]:
```

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 [ ]:

In [ ]:

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

In [ ]:

In [ ]:

In [ ]:

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 [ ]:

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 [ ]:

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 [ ]:

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 [ ]:
```

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

```
In [ ]:
```

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

```
In [ ]:
```

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 [ ]:
```

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 [ ]:
```

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 [ ]:
```

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 [ ]:
```

```
In [ ]:
```

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

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

```
In [ ]:
```

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 [ ]:
```

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 [ ]:
```

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 [ ]:
```

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 [ ]:
```

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 [ ]:
```