M1 Physique Appliquée — Université Paris-Sud XI
Année 2015-2016

Programmation et données numériques

TD1

Un memento sur l'utilisation de Python est donné à l'adresse :
`http://lptms.u-psud.fr/wiki-cours/index.php/Memento_Python`

Préliminaires

Installation

On suppose une utilisation de `Windows`. La distribution de Python et de ses librairies scientifiques installée est celle d'Anaconda [1] Une fois logué sur votre compte, vous devez ouvrir le `Launcher` d'`Anaconda` (icône verte) dans le dossier `Progam files`. Créer alors un nouvel environnement local avec votre nom d'utilisateur et choisir la version `Python 2.7`. Une fois connecté sous votre nom d'utilisateur, cliquez sur `Install` pour l'onglet `ipython-notebook` puis, l'installation terminée, cliquer sur `Launch` pour lancer l'application via un navigateur (typiquement `Firefox` ou `Chrome`).

Le notebook de IPython

Le notebook `IPython` suite le même principe que celui de `Mathematica` ou de `Maple` pour ceux qui connaissent. Il est constitué d'une succession de cellules dans lesquelles on peut sous écrire du code (cellule type `Code`), soit du texte (cellule type `Markdown`). Les autres types de cellule ne seront pas utiles.

☞ Écrire des lignes de code Python : Sélectionner la première cellule et taper :

```
print "Hello world!"
```

Si vous appuyez sur `Enter`, il s'agit uniquement d'un retour à la ligne. Exécuter la cellule en faisant `Shift-Enter`. Le résultat s'affiche en dessous. Pour utiliser Python comme calculatrice de base, essayer par exemple

```
2**3 - 1
```

Python affiche le résultat de l'expression, en l'absence de la commande `print`. Noter la différence avec le résultat de l'instruction

```
"Hello world!"
```

☞ Écrire des lignes de textes : Re-sélectionner la dernière cellule et choisir le type `Markdown`. Exécuter la cellule et regarder le résultat. Dans une nouvelle cellule type `Markdown`, vous pouvez écrire la ligne de code suivante et observer le résultat.

```
Définition de l'exponentielle :
$$e^{x} = \sum_{n=0}^{\infty}\frac{x^n}{n!}$$
```

1. Elle est libre et vous pouvez l'installer sur poste PC personnel si besoin.

☞ Numérotation et exécution des cellules : vous avez pu remarquer que la première cellule, une fois exécutée obtient un numéro d'input In [1]: qui va garder en mémoire l'ordre d'exécution des cellules. Si vous n'utilisez pas la fonction **print** pour afficher mais que vous faites par exemple

```
10**2
```

le résultat s'affiche avec un numéro d'output Out[1]: correspondant. Re-exécuter la cellule et voir le résultat. Attention, python intègre dans le temps les différentes définitions et importations que vous allez faire au fur et à mesure des cellules. Ce qui compte est le numéro des cellules et non l'ordre vertical. Prenons l'exemple suivant qui nous permet de voir le role de la commande

```
from __future__ import division
```

qui permet de faire en sorte que le symbole division ne donne pas comme résultat la division entière avec deux entiers mais la division comme pour des réels. Taper

```
print 12/5
```

dans une cellule. Créer puis exécuter une nouvelle cellule avec

```
from __future__ import division
print 12/5
```

Revenir à la précédente cellule et la re-exécuter. Observer les numéros de cellule. Si l'on souhaite réinitialiser le noyau ou `Kernel`, qui est le programme ou interpréteur qui exécute vos lignes de code, vous pouvez cliquer sur ↻ ou `Kernel > Restart`. À ce moment là, les lignes exécutées précédemment ne sont pas prises en compte et la numérotation recommencera au début. Faire l'essai d'une réinitialisation.

☞ Interrompre une exécution : lorsque vous exécutez une cellule qui prend un temps anormalement long et que vous voulez arrêter l'exécution, vous pouvez le faire via le bouton ■ ou l'onglet `Kernel > Interrupt`. Faire un essai d'interruption de cellule en lançant la commande :

```
i=0
while i<10**9:
    i += 1
```

puis en l'arrêtant après quelques secondes. Observer le numéro de la cellule avant l'arrêt et le message après l'arrêt. Faire afficher la valeur de i après l'arrêt.

☞ Manipuler les cellules : L'onglet `Edit` vous permet de déplacer, fusionner, supprimer, etc. . .les cellules. L'onglet Cell vous permets de les exécuter, en particulier Cell > Run All vous permet d'exécuter l'ensemble de la page dans l'ordre vertical.

☞ Aide : pour aller plus loin et retrouver les syntaxes, un onglet `Help` vous permet d'accéder à la documentation en ligne.

☞ Sauvegarde et export : à tout moment, vous pouvez sauvegarder le notebook sous un fichier d'extension `.ipynb` mais vous pouvez le figer, après l'avoir exécuter, au format `html` en utilisant `File > Download as`. Vous allez être sans arrêt en train de modifier votre fichier en l'éditant et en l'exécutant. Deux mécanismes utiles s'offrent à vous pour faire des sauvegardes temporaires : `File > Make a Copy` ouvre un nouvel onglet dans votre navigateur et y copie la version courante de votre notebook. `File > Save and Checkpoint` permet de sauvegarder la version courante à une heure donnée puis de repartir de celle-là en utilisant `File > Revert and Checkpoint` mais ce qui aura pour conséquence d'écraser toute la suite. Enfin, vous pouvez toujours sauvegarder une version intermédiaire sous le fichier `.ipynb`.

Prise en main de Python : petits exercices

Rappel sur l'utilisation des bibliothèques et modules

On rappelle les méthodes d'import d'une fonction depuis une bibliothèque ou un module :

☞ Import de l'espace de nom de la bibliothèque :

```
import bibliotheque
print bibliotheque.fonction(argument)
print bibliotheque.constante
```

ou bien en utilisant un nom raccourci

```
import bibliotheque as bib
print bib.fonction(argument)
```

ou en important uniquement la fonction voulue

```
from bibliotheque import fonction
print fonction(argument)
```

ou encore, on importe toutes les fonctions de la bibliothèque

```
from bibliotheque import *
print fonction(argument)
```

Un exemple simple pour récupérer le nombre π défini comme `pi` dans la bibliothèque `math`

```
import math
print math.pi
```

Une fois les bibliothèques importées, il n'y a plus besoin de les importer de nouveau. On peut faire afficher dans l'output la documentation d'une bibliothèque importée par la fonction `help()` :

```
help(bibliotheque)
```

☞ Faire afficher l'aide de la bibliothèque `math` pour voir son contenu.

☞ Utiliser la bibliothèque `cmath` adaptée aux nombres complexes pour calculer $e^{i\pi}$.

Boucles et définition de fonctions

☞ Créer une fonction `Factorielle` qui renvoie $n!$, d'abord de façon non-récursive avec une boucle `for`, puis de façon récursive, c'est-à-dire en faisant en sorte que la fonction s'appelle elle-même. *On pensera à utiliser un test conditionnel dans le second cas.*
Vérifier votre résultat avec celui de la fonction `factorial` de la bibliothèque `math`.

☞ Écrire une fonction `Exp(x,precision)` qui calcule l'exponentielle sous forme d'une série $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$ avec comme critère d'arrêt de la série que l'on ait $|a_{n+1}/\sum_{i=0}^{n} a_i| <$ precision où les a_n sont les éléments de la série. Mettre une valeur par défaut pour l'argument précision. *Utiliser la boucle* `while`.

☞ Importer la fonction `exp` de la bibliothèque `math` et comparer les erreurs relatives commises sur les valeurs de `Exp(3.0,p)` pour les précisions $p = 10^{-2}, 10^{-5}, 10^{-10}$.

Graphiques

☞ Tracer le graphe 2D de la fonction exponentielle (prise dans la librairie `math`) avec **pylab** pour $x \in [-3, 3]$ avec 101 points et en donnant des titres pour les axes et le graphiques.
Il faudra pour cela regarder le fonctionnement des listes et utiliser le memento.

Un générateur de nombres aléatoires

Un manière de générer des nombres aléatoires, très utiles pour les simulations en physique, est d'utiliser des suites congruentes de la forme

$$u_{n+1} = (a \times u_n + c)\%m$$

où a, c et m sont des paramètres, u_0 est appelée la graine et % signifie *modulo* m, c'est-à-dire le reste de la division par m qui correspond au même symbole en Python. Faire par exemple :

```
print 12345%123
print 5.42%1.2
```

- ☞ Écrire une fonction `aleatoire()` qui renvoie les nombres générés par la suite. On pourra faire une première version dans laquelle la fonction prend comme argument u_n pour renvoyer u_{n+1}. Essayer les paramètres suivants pour générer les 10 premiers nombres de la suite :

```
a, c, m = 25, 16, 256
graine = 10, 50 puis 125
a, c, m = 8121, 28411, 134456 # quickran choice
a, c, m = 16807,    0, 2**31-1 # standard minimum
```

  Quelle sont les valeurs minimale et maximale possibles pour les nombres générés ?
- ☞ Écrire une seconde version qui n'a pas d'argument mais qui génère les nombres à chaque appel et qui sera plus pratique pour créer des listes aléatoires. Il faudra dans ce cas utiliser le mot-clé `global`

```
u = ????
def aleatoire ():
        global u
         ...
        return ???
```

- ☞ Adapter le générateur précédent pour créer une fonction `Uniform(`$x_{\texttt{min}}, x_{\texttt{max}}$`)` qui génère des nombres réels sur l'intervalle $[x_{\min}, x_{\max}]$ et distribués uniformément.

Manipulation des listes

- ☞ Utiliser le mini-générateur de nombres aléatoires pour générer une liste de $N = 1000$ réels aléatoires compris entre -1 et 1. *Si vous n'avez pas réussi à créer le générateur, utiliser la fonction* `uniform()` *de la bibliothèque* `random`.
- ☞ Écrire en une ligne une expression qui renvoie la moyenne $\langle x \rangle$ des éléments à l'aide de fonctions Python adaptées aux listes.
- ☞ Écrire en une ligne une expression qui renvoie l'écart-type σ des éléments.
- ☞ Vérifier que les fonctions `mean` et `std` de la bibliothèque `numpy` redonnent les mêmes résultats.