TP n°1

L'IDE Spyder et premiers pas en Python

1 Lancement et configuration de Spyder

Un environnement de développement intégré est une interface qui permet de développer, compiler et exécuter un programme dans un langage donné. Pour programmer en Python, nous allons utiliser l'environnement de développement intégré (**IDE** pour Integrated Development Environment) Spyder. Plus tard, nous rencontrerons aussi l'IDE nommé IDLE, mais nous ne l'évoquerons pas dans ce document.

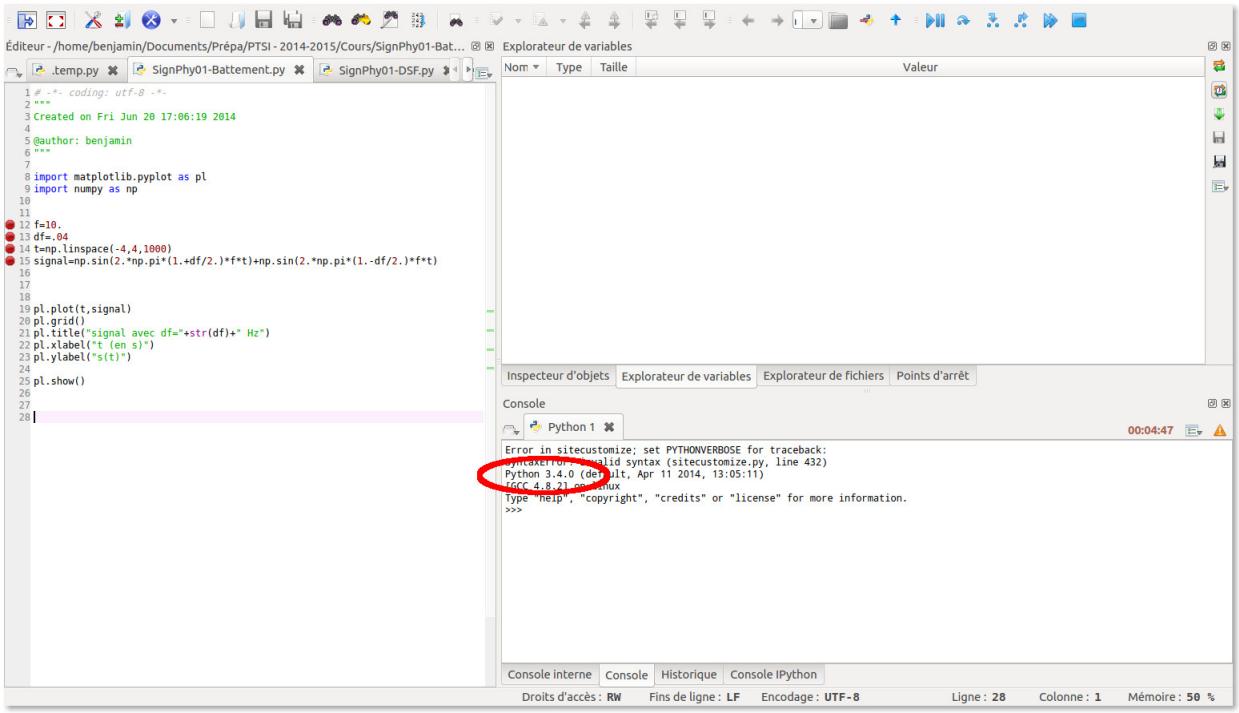
☞ Lancer Spyder et configurer-le pour que trois fenêtres apparaissent :

- un **éditeur** (Editor) ;
- une **console** (Console) ;
- un **explorateur de variables** (Variable Explorer) ;

et commenter avec l'enseignant.
Mots clés : exécution, débugger, exécution pas-à-pas,...

On peut aussi faire apparaître un **explorateur d'objets** (Object Inspector), qui est une sorte d'aide en ligne.

Capture écran de l'environnement Spyder



2 Vérification de la version de Python

Conformément à l'esprit du concours, nous travaillerons avec python 3. Pour éviter de travailler avec la mauvaise version, il faut vérifier lors du démarrage qu'on travaille effectivement avec python 3.x.

Lors du démarrage de spyder, quelques lignes apparaissent dans la console. On peut y lire la version de python.

- S'il est écrit `Python 3.4.0 (default, Apr 11 2014, 13:05:11)`, on peut coder, sans rien modifier.
- S'il est écrit `Python 2.7.6 (default, Mar 22 2014, 22:59:56)`, il faut changer de version. Pour cela, il faut se rendre dans :

  Outils > *Préférences* > *Console* > *Options avancées*

  puis changer le répertoire dans *Exécutable python* :

sous Mac et Linux	*/usr/bin/python3*
sous windows	

  Vérifier alors, dans *Substitution de PYTHONSTARTUP*, que le fichier de démarrage n'est pas appelé depuis le répertoire */usr/bin/python2.7*. Si oui, il faut revenir sur *Script PYTHONSTARTUP par défaut*.

3 Console versus éditeur

Pour éxécuter des lignes de commandes, on peut les écrire dans la **console**. Voici un exemple.

```
»> 4+5
9
»> sin(pi)
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
NameError: name 'sin' is not defined
»> import numpy
»> numpy.sin(numpy.pi)
1.2246467991473532e-16
```

On a commencé par entrer `4+5`, après le prompt `>>>` et on a éxécuté. Le résultat s'est alors affiché dans la console.
Ensuite, on a voulu calculer $\sin(\pi)$ et on a saisi `sin(pi)`. Après exécution, on a obtenu une erreur nous indiquant que la fonction `sin` n'était pas connue.
On a alors chargé la bibliothèque `numpy` qui contient sin (et π), en saisissant `import numpy`. Notons que la bibliothèque `numpy` contient beaucoup d'autres fonctions.
Enfin on a calculé $\sin(\pi)$ en saisissant `numpy.sin(numpy.pi)`.

☞ Commenter la valeur `1.2246467991473532e-16` qui est affichée.

Réponse :

On observe dans cet exemple, que chaque ligne de code est exécutée, dès que sa saisie est achevée. Dans la console, on n'a donc pas la possibilité d'écrire plusieurs lignes de commandes avant de les exécuter, ce qui est parfois gênant. En effet un programme peut comporter plusieurs dizaines de lignes de code et il n'est pas pratique de devoir les exécuter au fur et à mesure de l'écriture.

On dispose d'un autre moyen pour programmer en Python. On peut taper plusieurs lignes de codes dans l'**éditeur** et demander l'exécution seulement quand on le souhaite, après avoir sauvegardé le programme dans un fichier. Ceci présente plusieurs avantages :

- conservation des différents programmes en mémoire (en vue d'une éventuelle nouvelle utilisation) ;
- liberté d'exécuter le code au moment que l'on veut ;
- possibilité d'éxécuter le programme ligne à ligne (exécution pas à pas) pour débugger.

4 Échange du contenu de deux variables

Voyons dans un premier temps une variable comme une case mémoire possédant un nom et dans laquelle est contenue une valeur (numérique par exemple).

☞ Saisir le programme suivant dans l'éditeur, le commenter avec l'enseignant, l'éxécuter et observer ce qui s'est passé dans l'explorateur de variables et dans la console.
Mots clés : variable, nom, valeur, affectation.

Programme 1

```
1. a=2
2. b=7
```

Objectif : On se propose de compléter le programme 1 pour que le contenu des variables a et b soient échangé sous l'hypothèse que l'on ne connaît pas les valeurs stockées dans a et b, i.e. sans supposer que l'on sait que a contient 2 et b contient 7.

☞ Saisir le programme suivant dans l'éditeur, puis l'exécuter **pas-à-pas** et observer comment sont modifiées les valeurs des variables au fur et à mesure de l'exécution.
Mot clé : Pdb (Python Debugger).

Programme 2

```
1. a=2
2. b=7
3. a=b
4. b=a
```

☞ Il s'agit là d'une solution qui est en fait assez naïve et qui ne répond pas au problème posé. Pourquoi ?

Réponse :

☞ Modifier le programme 2 en ajoutant une variable auxiliaire, nommée aux, afin que le nouveau programme réponde bien à l'objectif fixé.

☞ Compléter le programme 1 en un programme répondant à l'objectif imposé, mais n'utilisant que les seules variables a et b (et donc n'utilisant pas de variable auxiliaire).

5 Calculs de sommes

Le programme suivant calcule et affiche la somme des entiers de 1 à 4 (i.e. 1+2+3+4). En modifiant la valeur affectée dans la variable n, il permet en fait de calculer la valeur de la somme

$$S_N := \sum_{k=1}^{N} k = 1+2+3+\ldots+(N-1)+N$$

où N est un entier naturel non nul. Notons qu'on a la relation de réccurence suivante :

$$\forall\, N \in \mathbb{N}_{\geq 2} \qquad S_N = S_{N-1} + N.$$

C'est elle qui est à la base de la construction du programme ci-dessous.

Programme 3

```
1. n=5
2. k=1
3. s=0
4. while (k<=n):
5.     s=s+k
6.     k=k+1
7. print(s)
```

☞ Saisir le programme précédent dans l'éditeur, le commenter, puis l'exécuter **pas-à-pas** et observer comment sont modifiées les valeurs des variables au fur et à mesure de l'exécution.
Mots clés : indice, booléen, boucle while, incrémentation, affichage...

☞ Construire un programme qui calcule la somme S_N ($N \in \mathbb{N}^*$), sans utiliser de boucle.
Indication : on dispose d'une « formule » pour la somme S_N ($N \in \mathbb{N}^*$).

☞ Pour tout $N \in \mathbb{N}^*$, on pose

$$T_N := \sum_{k=1}^{N} k^2 = 1^2 + 2^2 + 3^2 + \ldots + (N-1)^2 + N^2.$$

1. Donner une relation de récurrence vérifiée par les sommes T_N ($N \in \mathbb{N}^*$).

   Réponse :

2. S'inspirer du programme 3 pour construire un programme calculant la somme T_N ($N \in \mathbb{N}^*$) et utilisant une boucle while.

☞ Construire un programme qui calcule la somme T_N ($N \in \mathbb{N}^*$), sans utiliser de boucle.
Indication : on dispose d'une « formule » pour la somme T_N ($N \in \mathbb{N}^*$).

☞ Pour tout $N \in \mathbb{N}^*$, on note U_N la somme des N premiers entiers impairs (le premier entier impair étant 1).

1. Soit $N \in \mathbb{N}^*$. Écrire la somme U_N à l'aide du symbole $\sum$.

   Réponse :

2. Donner une relation de récurrence vérifiée par les sommes U_N ($N \in \mathbb{N}^*$).

   Réponse :

3. S'inspirer du programme 3 pour construire un programme calculant la somme U_N ($N \in \mathbb{N}^*$) et utilisant une boucle while.

☞ Construire un programme qui calcule la somme U_N ($N \in \mathbb{N}^*$), sans utiliser de boucle.
Indication : on dispose d'une « formule » pour la somme U_N ($N \in \mathbb{N}^*$).

6 Factorielle d'un entier

☞ Pour tout entier $N \in \mathbb{N}^*$, on appelle factorielle de N et on note $N!$ le nombre entier défini par :

$$N! := \prod_{k=1}^{N} k = 1 \times 2 \times 3 \times \ldots \times (N-1) \times N.$$

1. Donner une relation de récurrence vérifiée les nombres $N!$ ($N \in \mathbb{N}^*$).

   Réponse :

2. S'inspirer du programme 3 pour construire un programme calculant $N!$ ($N \in \mathbb{N}^*$) et utilisant une boucle while.