U.E. Logiciels scientifiques - Licence de Mathématiques, 2ième année

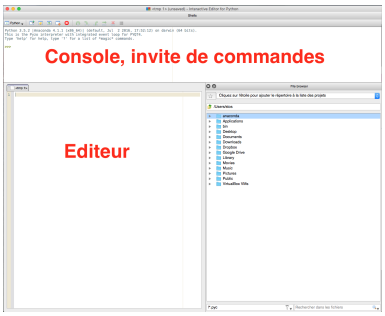
TP 1 : Introduction au langage Python

Prise en main de l'environnement de travail :

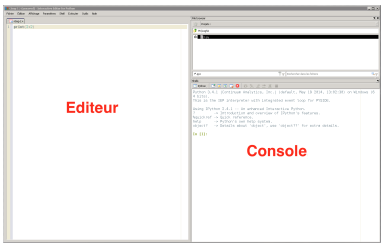
1. En utilisant vos identifiants ENT, logguez-vous sur le serveur **Windows (cours et TP)**. Certaines salles (p. ex. 2001 et 2002) sont équipées de postes PC : vous pouvez alors travailler directement sur le poste (et non sur le serveur).
2. Ouvrez l'explorateur de fichiers et trouvez le disque `M`.

   C'est un disque en réseau, le seul qui vous permettra de retrouver vos fichiers sur n'importe quel poste de travail. Par ailleurs, les fichiers enregistrées sur un autre disque (C, D,...) risquent de disparaître !
3. Sur votre disque `M`, créez un dossier `LogSci`. Dans la suite, vous allez stocker tout vos fichiers dans ce dossier en les nommant, par exemple, `tp1exo2.py` etc. D'une manière générale, sauvegardez chaque exercice dans un autre fichier. Ces programmes vont vous servir comme une base d'exemples pour les contrôles.
4. Dans le menu `Démarrer` → `Tous les programmes` trouvez (et lancez) l'application Pyzo 

. Il s'agit d'environnement de travail permettant d'écrire et d'exécuter de programmes (appelés aussi scripts) en Python.
5. Au premier lancement, Pyzo démarre un guide d'auto-présentation. La lecture de ce guide est optionnelle, l'essentiel se trouvant ci-dessous. Par défaut la fenêtre de Pyzo contient quelques sections. Les parties les plus importantes sont l'éditeur de texte et la console interactive :

   On peut modifier l'arrangement de la fenêtre à l'aide de glisser-déposer des sections. Il est commode de déplacer la console dans la partie verticale à droite.

6. Micro-exercice : Dans la console, calculez `2+2`.
7. A la différence de calculettes dont vous avez l'habitude, en Python la fonction puissance est notée `**` (deux étoiles).

   Micro-exercice : En utilisant cette fonction, calculez (dans la console) 5 puissance 4.

8. Pour faire des calculs plus complexes et pour les sauvegarder, on crée des *scripts* (programmes) dans l'éditeur. Sur le disque ce sont des fichiers d'extension `.py`.

   Micro-exercice : Allez dans la fenêtre de l'éditeur (à gauche) et tapez : `print(2+2)`

   Sauvegardez ce script dans le dossier `LogSci` sur le disque `M` (on pourra le nommer `tp1exo0.py`).

   Lancez votre script en tapant `Ctrl-E`. Observez le résultat dans la console.

   Prenez l'habitude de sauvegarder tous vos programmes dans le bon dossier ! **Sauvegardez souvent votre travail** (`Ctrl-S`) !

   Remarque 1 : Parfois, un programme peut tourner en rond (jamais finir son travail). Dans ce cas, on peut l'arrêter en cliquant dans le console puis en tapant `Ctrl-C`. Si cela ne fonctionne pas, on peut fermer la console Python à l'aide du bouton "Stop" rouge  .

   Dans ce cas, il faudra relancer une nouvelle console à l'aide du menu `Shell -> Create shell`.

   Remarque 2 : Pour ouvrir vos fichiers `*.py` dans une fenêtre d'explorateur de fichiers Windows, un double-click ne fonctionnera pas (l'extension n'est pas reconnue par Windows). On peut toujours ouvrir ses scriptes à partir de l'application Pyzo.

9. Dans un script, un commentaire se met après le signe dièse `#`. Par exemple :

   ```
   1+1 # c'est trop facile!
   ```

   Cela est très utile pour expliquer ce qui se passe dans vos programmes et noter les réponses aux questions posées dans la feuille de TP.

Remarque : à partir de maintenant, on mettra le code de chaque exercice dans un script (un fichier crée dans l'éditeur), que l'on nommera `tp1exoXX.py`, où `XX` est le numéro de l'exercice.

Exercice 1. *Variables et opérations*

On peut penser qu'une *variable* est une boîte qui contient une valeur. Pour affecter à une variable `x` une valeur, on utilise le symbol "`=`", par exemple :

```
x = 5
```

Attention ! Il faut comprendre que ceci ne représente pas une égalité mathématique, ni d'ailleurs une comparaison !

C'est une action effectuée par l'ordinateur (une modification dans la mémoire vive de la machine).

On va lire : *x prends la valeur 5.*

Effectuons maintenant quelques opérations basiques (à taper dans la console) :

```
x = 5
y = 2 * x
print("la somme x + y = ", x + y)
```

Dans la dernière ligne, on remarquera une chaine de caractères entre guillemets.

Python distingue majuscules et minuscules. Tapez :

```
a = 1
A = 2
print(a,A)
```

A présent, on se propose d'augmenter la valeur de `x` de 1. On tape (dans la console) :

```
x = x + 1
print(x)
```

Encore une fois, notez bien la différence entre l'égalité mathématique et l'affectation ci-dessus. Alors que l'équation algébrique $x = x + 1$ n'a pas beaucoup d'intérêt, l'action effectuée ci-dessus est bien courante en programmation :

tout d'abord, la machine calcule la valeur de l'expression à droite, c'est-à-dire de `x + 1`, puis elle affecte cette valeur à la variable qui se trouve à gauche, `x` (en oubliant son ancienne valeur 5).

Par ailleurs, Python propose une élégante syntaxe pour cette opération : au lieu d'écrire `x = x + 1` on peut taper `x += 1` (essayez !). Cela a l'avantage d'éviter une confusion avec une équation algébrique. On devine facilement d'autres opérateurs : `-=` `*=` `/=`

Exercice 2. *Affichage*

Tapez les exemples de la section "*Affichage*" de l'aide mémoire. Observez les résultats. Afficher 1023.98765 arrondi à 2 chiffres après la virgule.

Ensuite, écrivez un code qui affecte une valeur à une variable `x` (p. ex. `x = 10.23`) et qui calcule cette valeur élevée au carré.

On affichera ce calcul sous forme d'une equation $x * x = y$ où les valeurs seront arrondis à 2 chiffres après la virgule (cf. `print` et `round`).

Exercice 3. *Structure modulaire et des fonctions mathématiques*

En Python toutes les fonctions et outils variés sont regroupés dans les *modules* (appelés aussi *bibliothèques* ou *packages*). Par exemple, les fonctions mathématiques usuelles se trouvent dans le module `numpy`.

Pour en profiter, il faut d'abord le charger à l'aide de la commande

```
import numpy as np
```

On mettra cette commande une fois au début du programme (fichier). Ensuite, on peut utiliser une fonction mathématique en mettant le prefix `np`, par exemple : `np.sin(2)**2 + np.cos(2)**2`. Aussi, la constante π est définie dans `numpy` : `np.pi`.

1. Étudiez les exemples dans la section "*Opérations, fonctions*" de l'aide mémoire (effectuez quelques calculs avec des différents valeurs valeurs des variables `a`, `b` ou `x`).
2. Calculez $\sin(\pi^2)$ et le logarithme népérien (de base e) de 2021 (cf. l'aide-mémoire).
   Calculez les logarithmes de base 10 des nombres suivantes : 90, 99, 101, 120, 999, 500, 1001 (et d'autres si vous voulez !) Quelle est la relation entre le résultat obtenu et le nombre de chiffres de l'entier initial ?
3. À partir de la dernière observation, proposez une formule mathématique permettant de déterminer le nombre de chiffres de la valeur entière affectée à la variable `x`. On pourra utiliser la fonction `int` (cf. l'aide mémoire).
   Vérifiez sur quelques exemples faciles. Puis, à l'aide de votre formule, déterminez le nombre de chiffres de $2^{64} - 1$ (réponse : 20).
   Pour des raisons techniques, la fonction `np.log10` n'accepte pas d'argument plus grand que 2^{64}. Étant donnée cette contrainte, calculez le nombre de chiffres de 123^{234} (réponse : 490).

Exercice 4. *Test* `if`

1. Étudiez les exemples concernant le test `if` dans l'aide mémoire. Tapez dans la console :
   ```
   0 == 1
   2 > 1
   ```
   Il faut bien distinguer l'affectation "=" de la comparaison "==" !
   Tapez par exemple :
   ```
   a = 10
   a == 20
   print(a)
   ```
2. Créez un script avec le code ci-dessous. Faites attention à respecter le décalage (de 4 espaces) pour des commandes `print` (sauf la dernière ligne).

```
a = 10
b = 12
if a >= b:
    print ("a est plus grand")
    print ("j'ai verifie!")
else:
    print ("b est plus grand")
    print ("b est vraiment plus grand")
print("ceci s'affiche toujours")
```

Lancez ce script plusieurs fois avec différentes valeurs de variables `a` et `b`.

Observez le rôle du décalage dans ce code.

On dit que le décalage marque un *bloc* de commandes. C'est une propriété importante de la syntaxe Python qui contribue beaucoup à la lisibilité du code. Attention, ce décalage doit être toujours de la même taille : 4 espaces, de préférence.

3. On peut utiliser plusieurs conditions en les séparant par une opération logique `and` ("et") ou `or` ("ou"). Par exemple :

```
a = 10
if a >= 9 and a <= 11:
    print("a est dans l'intervalle [9, 11]")
```

Mettez ce code dans un script, testez-le avec différentes valeurs de la variable `a`.

4. Pour écrire des conditions encore plus complexes, on peut utiliser des parenthèses.
Écrire une commande `if` (comme ci-dessus) qui affiche un message si `a` appartient à $[-1, 0[\cup[3, 4]$.
Tester.

Exercice 5. *Boucle* `for`

La boucle `for` permet d'effectuer les mêmes commandes plusieurs fois avec la valeur d'un index variant dans une liste. Cette liste est le plus souvent générée à l'aide de la commande `range` qui fait varier l'index entre deux bornes. Par exemple :

— `range(3, 7)` renvoie `3, 4, 5, 6`, la borne supérieure 7 étant par définition exclue.
— `range(4)` est un raccourci pour `range(0, 4)` : cela renvoie donc `0, 1, 2, 3`.

1. Créez un script avec l'exemple d'une boucle `for` dans l'aide mémoire (encore attention au décalage !), exécutez et observez les résultats.
2. Créez un script qui affiche les nombres pairs entre 10 et 40 inclus. On pourra utiliser l'opérateur `%` (le reste de la division euclidienne, cf. l'aide mémoire).

Exercice 6. *Boucle* `while`

Exécutez l'exemple de la boucle `while` donné dans l'aide mémoire. Puis déterminez le plus petit entier n tel que l'expression $\frac{n*(n+1)}{2}$ dépasse 1 million.

Exercice 7. *Procédures*

1. Créez un nouveau fichier dans l'éditeur Pyzo. Recopiez dans ce fichier la procédure `carre_plus(n)` de l'aide mémoire (attention au décalage !).
Plus bas dans le même fichier, faites appel à cette procédure (comme à une fonction mathématique usuelle) avec différents valeurs de l'argument `n` (et affichez le résultat avec `print`.
Calculez `carre_plus(345) + carre_plus(567)` (réponse : 440518).
2. Écrire une procédure `racines(a,b,c)` qui, étant donnés 3 réels `a`, `b` et `c`, détermine et **retourne** le nombre des racines réelles de l'équation du second degré $ax^2 + bx + c = 0$ (on ne demande pas de la résoudre).

Testez avec les équations suivantes :
$x^2 + x - 2 = 0, \quad 4x^2 - 12x + 9 = 0, \quad 2x^2 + x + 1 = 0.$

Remarque : D'une manière générale, dans une procédure on n'utilisera jamais `print` pour afficher le résultat, mais `return` pour renvoyer le résultat pour la suite du programme.

Ainsi dans cet exercice, on va afficher le résultat renvoyé par la procédure `racines` à l'aide de `print` à l'extérieur de la procédure (sans décalage dans le fichier), par exemple : `print(racines(1, 2, 3))`.

Exercice 8. *Procédures, boucles et tests*[1]

Une procédure peut renvoyer, à l'aide de la commande `return`, une valeur qu'elle a calculée. Cette valeur peut être éventuellement utilisée ensuite pour d'autres calculs.

Si on écrit tous les entiers divisibles par 3 ou par 5 strictement inférieurs à 10, on obtient 3, 5, 6 et 9. Leur somme est 23.

Écrire une procédure `div35(n)` qui renvoie la somme des entiers divisibles par 3 ou par 5 strictement inférieurs à `n`. Tester : div35(1000) = 233168.

Que vaut `div35(2000) + div35(3000)` ?

Exercice 9. *La suite de Syracuse*

Si un nombre est pair, on le divise par 2. S'il est impair, on le multiplie par 3, on lui ajoute 1 et on divise le résultat par 2. Ainsi, en commençant avec n=10, on obtient la suite suivante :

$10 \to 5 \to 8 \to 4 \to 2 \to 1.$

Une fois tombée sur 1, la suite tourne en rond : $1 \to 2 \to 1 \to ...$

Selon la *conjecture de Syracuse*, quelle que soit la valeur initiale n choisie, la suite finira toujours par atteindre 1. Plusieurs projets visent à vérifier cette conjecture. On se propose ici d'étudier quelques questions basiques.

1. Écrire une procédure `syracuse(n)` qui, étant donné un entier `n`, retourne la longueur de la suite de Syracuse commençant par `n` (la suite se termine par la première occurrence de 1).
   Tester : `syracuse(10)` = 6, `syracuse(100)` = 19.
2. Calculer (et afficher) les longueurs de la suite de Syracuse pour la valeur initiale `n` variant de 10 à 25.
3. Trouver la longueur maximale de la suite de Syracuse pour la valeur initiale `n` variant de 10 à 10 000.
   Indication : Dans le code de la question précédente, on pourra introduire une variable `lmax`, initialisée à 0. A chaque tour de boucle, si on trouve une suite de longueur supérieure à `lmax`, on modifie cette dernière variable. Ainsi, à la fin, `lmax` sera la longueur maximale recherchée.
4. Trouver la valeur de `n` (entre 10 et 10000) qui produit la suite de la longueur maximale.

1. Cette exercice à été inspiré par le Project Euler, `http://projecteuler.net`