# TP 1 : Python et Pyzo

## I L'environnement Pyzo

Lorsqu'on ouvre Pyzo, on se retrouve face à :

1. Une fenêtre de commandes (**shell**) dans laquelle on peut directement exécuter des instructions.
2. Une fenêtre d'édition : dans laquelle on peut écrire différentes lignes de code, et regrouper plusieurs commandes. L'intérêt est surtout de pouvoir **sauvegarder** ces lignes de codes (dans des fichiers en `.py` ou `.pyw`).
3. Une barre des menus : qu'il vaut mieux avoir exploré un peu avant de se lancer pour en saisir la portée.

L'idéal étant de taper dans la fenêtre d'édition les lignes de codes, de sauvegarder **très régulièrement** son code, et de l'exécuter (avec le raccourci `Ctrl+E` ou manuellement). On peut aussi se contenter d'exécuter un bout du code (en le sectionnant puis en faisant `Alt+Entrée` ou manuellement)=.

Pour ne pas être trop perdu dans un code (même dans son propre code), il est encouragé de mettre divers **commentaires** dans son code : ce sont des lignes qui ne seront pas exécutées. Il suffit d'insérer un double dièse `##` au début de la ligne.

## II Les types

### II.1 L'affectation

Pour donner une valeur à une variable, on utilise le signe `=`. Et on peut faire plusieurs affectations en parallèles ou simultanément.

```
>>> a=3
>>> a
3
>>> a=a+1
>>> a
4
>>> a,b=2,a   # affectations parallèles
>>> a,b
(2, 4)
>>> a=b=3   # affectations simultanées
>>> a,b
(3, 3)
```

### II.2 Les types

Les variables affectées peuvent avoir différents types :

1. les entiers `'int'` : les entiers relatifs ;
2. les flottants `'float'` : les nombres à virgule ;
3. les booléens `'bool'` : les valeurs de vérité, qui sont `True` et `False` pour Python) ;
4. les $t$-uplets `'tuple'` : des paires, triplets, etc. d'objets, qui peuvent être de types différents ;

5. les listes `'list'` : un peu comme les $t$-uplets, mais beaucoup plus souple pour les modifier.

La fonction `type` permet de savoir le type d'un objet. Pour être utilisée avec une variable, celle-ci doit avoir été affectée.

Si, au lieu du type, on souhaite connaître la valeur d'une variable, il suffit d'écrire le nom de la variable en question. Pour l'afficher au milieu d'un code, on peut utiliser la commande `print` qui affiche ce que l'on souhaite.

**Exercice 1**

Déterminer les types des objets suivants :

```
>>> 12
>>> -6
>>> -6/2
>>> 3/5
>>> 3//5
>>> 3==3.0
>>> 3-3.0>=0
```

Des objets de type différents peuvent-ils être égaux ?

# III Les opérations élémentaires

## III.1 Quelques calculs avec des entiers ou des flottants

**Exercice 2**

Après avoir affecté différentes valeurs (de type `'int'` ou `'float'`)à `a` et `b`, effectuer les lignes de code suivantes :

```
>>> a*b
>>> a+b
>>> a**b
>>> a++b
>>> a/b
>>> a//b
>>> a%b
>>> a<b
>>> a<=b
>>> a!=b
>>> a==b
```

En déduire ce que font les opérations dans le code précédent.

## III.2 Calculs avec des booléens

**Exercice 3**

Sur le même mode que l'exercice précédent, dire ce que font les opérateurs suivants sur les booléens : `==` `and`, `or`, `not`.

En testant les commandes `+`, `*` et `**` avec des booléens, ou des booléens et des entiers, déduire comment sont interprétés les booléens avec les opérations numériques.

# IV Les listes

Par soucis de confort, on essaiera systématiquement de travailler avec des listes plutôt qu'avec des $t$-uplets. On retiendra surtout que :

1. les objets d'une liste (comme d'un $t$-uple) sont numérotés à partir de 0 ;
2. la liste vide (qui contient 0 éléments) est `[ ]` ;
3. pour créer une liste, on mettre entre les crochets `[` et `]` les différents objets que l'on souhaite, séparés par des virgules ;
4. la commande `len` donne la longueur d'une liste ;
5. on peut accéder au $i$-ème objet de la liste $L$ par la commande `L[i]` ; ou au $i$-ème objet en partant de la fin avec `L[-i]`.

```
>>> L=[2,3,5,7,11]
>>> len(L)
5
>>> L[3]
7
>>> L[-2]
7
>>> L[0]
2
```

# V Les instructions conditionnelles

On peut être amené à vouloir exécuter une commande (par exemple une affectation) seulement sous certaines conditions.

Dans ce cas on utilise la commande `if` suivi d'une condition : si la condition est remplie, toutes les lignes de commandes qui suivent sont exécutées ; sinon, les lignes **indentées** ne sont pas exécutées.

```
>>> a=0
>>> if a==0:
...     a=a+2
>>> a
2
```

On peut alors imbriquer d'autres conditions les unes dans les autres, avec la commande `elif`, qui correspond en français à un "mais si" ou "à moins que ...alors ...". Ou plus simplement donner une instruction si la condition n'est pas remplie, avec la commande `else`, qui correspond en français à un "sinon".

**Exercice 4**

Dire ce que fait le code suivant :

```
>>> if a==0:
...     a=a+3
... elif a==1:
...     a=a+2
... elif a==2:
...     a=a+1
... else :
...     a=3
```

# VI Les boucles

Lorsque l'on veut exécuter une même commande plusieurs fois (avec éventuellement des changements dans le variables), on peut utiliser les instructions de boucles `for` et `while`.

Les boucles `for` peuvent être utilisées pour parcours des entiers entre deux bornes, ou des éléments dans une liste.

Les boucles `while` sont à manipuler avec davantage de précautions : elles s'exécutent tant qu'une condition est satisfaite, mais il faut bien s'assurer que la boucle s'arrête au bout d'un moment.

Les trois boucles ci-dessous affichent la même chose :

```
>>> for i in range(2,5):
...     print(i)


>>> L=[2,3,4]
>>> for e in L:
...     print(e)


>>> i=2
>>> while i<=4 :
...     print(i)
...     i=i+1


>>> i=1
>>> while i<=3:
...     i=i+1
...     print(i)
```

**Exercice 5**

Écrire, de la même manière que les quatre modèles précédents, un code affichant les nombres impairs entre 5 et 11.

# VII La bibliothèque `math`

Certaines fonctions utiles ne sont pas implémentées de base dans Python : il faut les charger à l'aide de bibliothèque.

La bibliothèque `math` permet d'utiliser les fonctions usuelles (exponentielle, racine carrée, fonctions trigonométrique, etc.) ou des constantes (comme $\pi$).

Pour charger une bibliothèque, on utilise la commande `import` (par exemple `import math`), et on appelle les fonctions de la bibliothèque en apposant son nom (par exemple `math.exp` pour la fonction exponentielle). Pour aller plus vite, on peut importer avec quelques raccourci, par exemple :

```
>>> import math as ma
>>> ma.exp(1)
2.718281828459045
>>> ma.cos(ma.pi)
-1.0
```

On peut aussi importer uniquement une fonction ou un objet, et éventuellement lui donner au passage un nom au choix pour l'appeler.

```
>>> from math import pi
>>> pi
3.141592653589793
>>> from math import exp as exponentielle
>>> exponentielle(1)
2.718281828459045
```

**Exercice 6**

Depuis la bibliothèque `math`, importer uniquement les fonctions `sqrt` et `floor`.

En calculant les images de quelques entiers ou flottants, dire de quelles fonctions il s'agit.

# VIII Les fonctions

Lorsque l'on souhaite effectuer plusieurs fois les mêmes lignes de code pour des variables différentes, il est pratique d'utiliser des **fonctions**.

Pour cela, il faut bien définir ce que la fonction prend comme variable (l'**entrée** ou les **arguments**) et ce qu'elle retourne (les **images** ou la **sortie**). L'instruction `return` sert à définir la sortie.

Par exemple, la fonction suivante rend, pour deux nombres $a, b$, le périmètre et l'aire d'un rectangles de côtés $a$ et $b$ :

```
>>> def rectangle(a,b):
...     p=2*(a+b)
...     A=a*b
...     return [p,A]
>>> rectangle(12,3)
[30, 36]
```

**Exercice 7**

Écrire une fonction `maximum` qui prend en arguments deux nombres et rend le plus grand des deux.

# IX Mise en application

**Exercice 8**

Écrire une fonction qui, étant donnée une température et une unité (degrés Celsius ou Farenheit) rende la température dans l'autre unité.

Adapter ce programme pour que, avec les mêmes arguments, la fonction rende la température en Kelvin.

On rappelle que, si $C$ est une température en degré Celsius, $F$ la même température en degrés Farenheit, et $K$ la même en Kelvin, on a :

$$K = C + 273.15 \text{ et } F = \frac{9}{5}C + 32.$$

**Exercice 9**

Écrire une fonction `racines` qui prend en argument trois nombres $a, b, c$ (avec $a \neq 0$), et qui rend les racines réelles du polynôme $aX^2 + bX + c$ sous la forme d'une liste $[n, R]$ où :

— $n$ est le nombre de racines du polynôme ;

— $R$ est la liste des racines du polynôme, données sous forme de flottants.

**Exercice 10**

Dire ce que fait la fonction suivante :

```
def myst (n):
        nbc =0
        while n >0:
                nbc =nbc + 1
                n=n //10
        return(nbc)
```

**Exercice 11**

Construire une fonction `moyenne` qui prend en argument une liste de nombres et rend leur moyenne.

En utilisant cette fonction, construire une fonction `plusoumoins` qui prend une liste et rend sous forme d'une liste le nombre d'éléments de la listes inférieurs à la moyenne, et le nombre de ceux supérieurs à la moyenne.

**Exercice 12**

Créer la liste de tous les carrés impairs plus petits que 10000. Combien d'élément contient-elle ?

**Exercice 13**

Écrire une fonction qui prend en argument une liste d'entiers compris entre 0 et 9 et rend l'entier dont les chiffres sont ceux de la liste.

Inversement, écrire une fonction qui prend en argument un entier positif et rend les liste de ses chiffres.

**Exercice 14**

Écrire une fonction `systeme` qui prend en argument deux listes $[a_1, b_1, c_1]$ et $[a_2, b_2, c_2]$ et rend les solutions du système :

$$\begin{cases} a_1x & + & b_1y & = & c_1 \\ a_2x & + & b_2y & = & c_2 \end{cases}.$$

La sortie de la fonction fera apparaître le nombre de solutions (sous la forme 0, 1 ou $\infty$) et, à part dans le premier cas, donnera une solution du système.

On pourra commencer par traiter le cas où le système admet une unique solution. Et même se contenter de traiter ce cas.

**Exercice 15**

Décrire ce que fait la fonction suivante (qui prend en argument des entiers) :

```
def f(n):
    if n>100:
        return n-10
    else :
        return f(f(n+11))
```

**Exercice 16**

Une planche de chocolat contient 32 carreaux de chocolat. On veut la partager en 5 tas de chocolat, chacun comportant un nombre différent de carreau. Et on veut que le ppcm des nombres de carreaux de chaque tas soit le plus petit possible.

Quel est ce ppcm ?

On pourra écrire des fonctions pour le pgcd et le ppcm de deux entiers, puis les généraliser à davantage, pour comparer les ppcm de chacun des partages.

**Exercice 17**

Écrire une fonction qui prend en argument un entier naturel $n$ et rend le $n$-ème terme dans la suite de Fibonacci.

On rappelle que la suite de Fibonacci est définie par : $u_0 = 0$, $u_1 = 1$ et $u_{n+2} = u_{n+1} + u_n$.

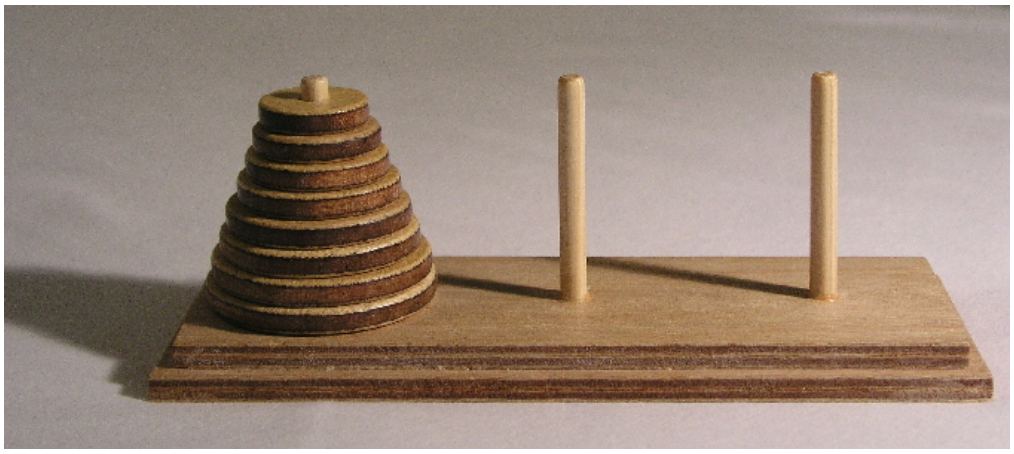
On essaiera de faire une fonction suffisamment rapide pour calculer en un temps raisonnable le 1000-ème terme de cette suite.

Trouver le rang du premier terme de la suite plus grand que 1000000.

Pour vérifier, on donne les premiers termes de la suite : 0, 1, 1, 2, 3, 5, 8, 13, 21,...

**Exercice 18**

On considère le problème des tours de Hanoï, qui consiste à déplacer une tour constituées de $n$ disques de plus en plus petits d'un emplacement à un autre.



Les emplacements sont trois tiges, et tous les disques sont initialement sur la première tige. On ne peut déplacer les disques qu'un seul à la fois, en prenant à chaque fois le disque du sommet de chaque pile, pour le mettre au sommet d'une autre pile. On n'a pas le droit de placer un disque sur un autre plus petit.

Établir une stratégie pour déplacer la tour de la tige 1 à la tige 3. Puis faire un programme qui donne une à une les instructions à suivre pour déplacer ainsi la tour.

**Exercice 19** Dans une très longue rue, les maisons sont numérotées par tous les nombres de 1 à 10000. Un groupe d'ami décide de s'y installer.

Léa, la plus jeune, choisit sa maison en premier : elle choisit la maison n° 6, car elle trouve élégant que : la somme des nombres de 1 à 5 fait 15, ce qui fait autant que la somme des nombres de 7 à 8.

Éric, le frère jumeau de Léa (comme quoi Léa n'était pas la plus jeune de beaucoup) décide de s'installer dans la maison n° 35 : il y a en effet le même phénomène, puisque la somme des nombres de 1 à 34 fait 595, soit autant que la somme des nombres de 36 à 49.

Et tous les amis trouvent finalement des nombres présentant les mêmes symétries : si leur numéro est $N$, alors la sommes des entiers de 1 à $N-1$ fait autant que la somme des entiers de $N+1$ jusqu'à un autre entier plus grand.

Mais à quelques centaines de maisons près il n'auraient pas pu se loger ainsi. Combien y a-t-il d'amis dans ce groupe ?