# T.P. 2 : Premiers pas avec PYTHON et pyzo.

## 1) Pour lancer Pyzo

A partir du bureau Debian ouvrir un Terminal. Dans le Terminal, taper `pyzo &` pour lancer Pyzo. Rappel du T.P. 1 : à quoi sert le `&` ?

Pyzo est l'interface qui va nous permettre d'écrire des programmes en python et de visualiser agréablement leur résultat. Lorsque vous l'installerez chez vous, vous devrez aussi installer une distribution Python.

Attention : `pyzo` est lancé à partir du Terminal, si vous fermez le terminal, vous fermez pyzo.

## 2) Présentation des zones de travail de Pyzo.

a) Le `shell` est au dessus (si jamais il est de côté, on peut le bouger en cliquant sur le haut de la fenêtre de shell). Il permet de rentrer des instructions en ligne de commande : il a un *prompt* `>>>`, qui, un peu comme dans le terminal, attend votre commande. Essayez de faire un calcul dans le shell.

b) En dessous la fenêtre `fichier` (si elle n'existe pas Ctrl-N ou menu déroulant `Fichier` puis `Ouvrir`). Elle permet de manipuler un fichier texte, qui contiendra le code qu'on exécutera dans un second temps. Essayer de taper : `print("Hello world")` dans la zone de fichier, puis d'exécuter ce programme. (Menu Executer ou Run puis apprendre le raccourcis clavier).

c) Enregistrer votre premier fichier avec `Ctrl-S` : en indiquant le chemin vers votre clef usb perso. Penser au `Ctrl-S` régulièrement, c'est un bon ami.

## 3) Expérimentation sur les entiers, les flottants, et un peu de chaînes de caractères

a) **Les entiers longs :** comment trouver le nombre de chiffres d'un entier long écrit à l'écran, par exemple $7**(1000)$ ? Rappelons que la double astérisque `**` sert à faire une .... ? On pourrait le faire par une opération mathématique [1], mais ici, une simple manipulation informatique suffit.

On stocke le résultat dans une variable `a`, qu'on transforme en chaîne de caractères via la commande `b=str(a)` puis on demande la longueur de `b` via la commande `len(b)`. Essayez !

b) **La division des flottants et la division euclidienne :**

i) **Une précaution préliminaire :**

Toujours vérifier que votre shell est en Python 3 : première ligne du shell

Sinon changer de shell (vu en cours).

ii) Taper `a=4/2` sans appuyer sur `return` (entrée). Saurez-vous prévoir quel est le type de `a` ? Vérifiez !

iii) Que donne `4//3` et `4//2` ? Expliquer.

iv) Tester et expliquer ce que fait le programme suivant :

```
a=input('Entrer un entier') # Affiche le message, attend une réponse de l'utilisateur,
# et  stocke la réponse dans a
print(type(a))
a=int(a)
if a%2==1 :
    print('Oui')
else :
    print('Non')
```

1. Typiquement avec un logarithme... je vous laisse y réfléchir...

c) **Pour ne pas confondre les noms de variables avec les chaînes de caractères**
Tapez d'abord dans le shell `>>> abc="trois lettres"`.
Essayez de prévoir la réponse du shell Python aux entrées suivantes et vérifiez votre réponse !

i) `>>> print(abc)`
ii) `>>> print('abc')`
iii) `>>> print(abc*3)`
iv) `>>> '1'+'2'+'3'`
v) `>>> 1+2+3`
vi) `>>> abc+'d'`
vii) `>>>abc+d`

## 4) Jouons encore un peu avec la commande `input`

Nous l'utiliserons peu par la suite, elle sera remplacée par l'usage de *fonctions* mais pour aujourd'hui elle nous permettra de faire notre premier programme. Cette commande `input` permet à la fois d'afficher un message et de recevoir une réponse de l'utilisateur : pour cela, on faut stocker le résultat de `input` dans une variable.

Par exemple, si un programme commence par :

```
nombre=input('Entrez un nombre et moi je vais lui ajouter 2')
# la valeur rentrée par l'utilisateur sera stockée dans la variable appelée ici nombre
```

comment compléter ce programme pour qu'effectivement l'ordinateur nous affiche le nombre plus deux ?

*Attention, il y a un petit piège, en cas d'erreur.. lisez le message d'erreur...*

## 5) Introduction aux structures conditionnelles : `if ... elif .. else`

**Un point sur la syntaxe :**

```
if condition1: # le :  a valeur de then
    instruction
    instruction  # toutes les instructions indentées sous la condition 1
#sont exécutées si condition1 est vraie.
elif condition2: # le elif est  pour else if.
     instruction
else  :
     instruction
```

a) Expliquer le différence de comportement entre les deux programmes suivants :

```
a=17
if a==17 :
    print("ok")
    a=0
if a<17:
    print("trop petit")
else :
    print("trop grand")
```

```
a=17
if a==17 :
    print("ok")
    a=0
elif a<17:
    print("trop petit")
else :
    print("trop grand")
```

b) Ecrire un programme qui demande d'entrer un nombre puis qui affiche *gagné* si ce nombre est égal à 17, *pas assez* si le nombre est strictement inférieur et *trop* s'il est supérieur strictement.

c) Quelques « simplifications » :
- Compte tenu de vos grandes connaissances en logique après le cours de maths sur le ET et le OU, trouver une version plus efficace du test suivant :
  `if (a<b) or ((a>=b) and (c==d)) :`
- Un élève a écrit le code suivant :

```
if b>a:
    a=a+1
else :
    a=a
```
Que lui suggéreriez-vous ?

## 6) Exercice sur les années bissextiles

Les années bissextiles sont les années multiples de 4 sauf les années divisibles par 100 (les années séculaires) qui ne sont pas divisibles par 400. Ainsi 2000 était bissextile, mais 1900 ne l'était pas et 2100 ne le sera pas non plus.

Faire deux programmes qui demandent de rentrer une année et répondent si elle est bissextile ou non :

• d'abord avec des `if ... elif.. else`

• puis seulement avec des `or and` en fabriquant alors un booléen qui répond `True ou False` comme il se doit.

## 7) Un exemple où on introduit les boucles `while` :

### 7.1. Un while pour être poli

Que fait le programme suivant ?

```
reponse=""
while reponse!="bonjour":
    reponse=input("Dis moi bonjour ")
print('bonjour à toi aussi')
```

### 7.2. Un jeu de Nim

Le jeu de Nim est un jeu qui se joue avec des allumettes et deux joueurs qui jouent à tour de rôle.

Il y a ici [2] qu'un seul tas de $N$ allumettes et chaque joueur doit prendre à chaque tour $1, 2$ ou 3 allumette(s) et celui qui prend la dernière allumette *perd*.

(i) Justifier que si un joueur a devant lui un tas avec 4,3, ou 2 allumettes avant de jouer, il gagne.

(ii) Définir une stratégie qui montre que si un joueur a, à un certain tour, avant de jouer, un nombre $N$ d'allumettes qui n'est pas de la forme $4k + 1$, il est sûr de gagner avec cette stratégie.

(iii) Mettre en oeuvre cette stratégie en programmant, à l'aide d'une boucle `while`, et des `input` pour le jeu de l'humain, une partie où au départ il y a 17 allumettes, l'humain commence et l'ordinateur gagne à coup sûr.

2. car il y a des variantes

# TP 2 : solutions

**3) Expérimentations sur entiers, flottants, chaînes de caractères**

**a)** Comprendre la différence entre les variables :

`a=7*7` où `a` est une variable dont la valeur est l'entier `49`, et `b=str(a)` où `b` est une variable dont la valeur est la chaîne de caractères `"49"`. En sens inverse avec `int(b)` on refabrique un entier.

**Remarque :** pour une chaine de caractères `L` contenant un flottant, la commande `int` ne sera pas efficace.

**3b)** (ii) En Python 3, `4/2` renvoie le flottant `2.0`. Le `/` renverra toujours un flottant.

(iii) `4//2` renvoie l'entier quotient de la division euclidienne. Ainsi `4//2` renvoie `2` et `4//3` renvoie `1`.

(iv) Bien retenir aussi le `%` qui renvoie le *reste* de la division euclidienne *très utile en informatique*. Ici `a%2==1` signifie que `a` est impair alors que `a%2==0` signifie que `a` est pair.

Pour le programme donné :

```
a=input('Entrer un entier')
a=int(a)
if a%2==1 :
    print('Oui')
else :
    print('Non')
```

La première ligne affiche 'Entrer un entier' puis attend une réponse, la réponse est stockée *à l'intérieur* d'une chaine de caractères, appelée `a`. La seconde ligne transforme la chaîne de caractères en entier. Puis est affiché 'Oui' si `a` est impair et 'Non' sinon.

**Remarque :** dans le code précédent on peut remplacer les 4 lignes du `if` par : `print(a%2==1)` ce qui est plus concis bien sûr mais surtout, à terme, on préférera stocker les résultats dans une variable, par exemple ici un booléen `R=(a%2==1)`.

(vi) Petite excursion en Python 2 :

En Python 2, l'opérateur `/` ne donne pas le même résultat suivant qu'il s'applique à des `int` ou des `float`.

Ainsi pour des entiers il renvoie le quotient de la division euclidienne : ainsi `5/2` renvoie `2`

Si on fait `float(5/2)`, l'ordinateur calcule d'abord `5/2` qui donne `2`, puis le transforme en float ce qui donne `2.0`.

Ce n'est bien sûr pas le résultat de la division des floats, qu'on obtiendrait avec `float(5)/float(2)` ou `5./2.`

**4) Le programme qui fait +2**

```
a=input("entrez un nombre et je vais lui ajouter 2")
a=int(a)
print("En ajoutant 2, on obtient",a+2)
```

**5) a)** Il faut penser chaque structure `if, elif, else` comme un seul bloc où (au plus) *une seule* des conditions sera vérifiée et donc (au plus) *une* série d'instructions sera exécutée. En revanche avec deux `if` à la suite, même si les conditions exprimées semblent s'exclure, il est possible que les instructions exécutées par le premier `if` rendent le second `if` possible. C'est le cas dans le premier exemple suivant :

```
a=17
if a==17 :
    print("ok")
    a=0
if a<17:
    print("trop petit")
else :
    print("trop grand")
```

Ici comme à la ligne 2, la condition `a==17` est vérifiée, les instructions des lignes 3 et 4 s'exécutent, `a` prend la valeur `0` et donc ce qui dépend de la condition `if a<17` va aussi être exécuté. L'affichage sera donc :

```
ok
trop petit
```

En revanche avec le `elif` pas de pb.

```
a=17
if a==17 :
    print("ok")
    a=0
elif a<17:
    print("trop petit")
else :
    print("trop grand")
```

L'affichage sera seulement

```
ok
```

b) Programme très simple, juste pour travailler encore le `input` en plus des `if`

```
a=int(input("entrez un nombre"))
if a==17:
    print("gagné")
elif a>17:
    print("trop")
else :
    print("pas assez")
```

c) Quelques « simplifications » :

- Pour simplifier le test :
  `if (a<b) or ((a>=b) and (c==d)) :`
  on peut remarquer que par distributivité du `or` ce test est est équivalent au (pas plus simple ) :
  `if ((a<b) or (a>=b) ) and ((a<b) or (c==d)) :`
  Mais `(a<b) or (a>=b)` étant toujours vrai, on peut l'enlever du `and` et donc la condition se simplifie en : `if (a<b) or (c==d)`

d) Un élève a écrit le code suivant :

```
if b>a:
    a=a+1
else :
    a=a
```

Que lui suggéreriez-vous ? Réponse : enlever le `else : a=a` qui ne fait rien d'intéressant !

**6) Bissextile**

On présente deux méthodes « extrêmes », il est possible de combiner bien sûr `if else` et `and, or`

- (M1) Avec les `if` : *noter l'intérêt de l'ordre dans lequel les conditions apparaissent*

```
annee=int(input('Entrez une année'))
if annee%400==0:
    print("bissextile")
elif annee%100==0:
    print("non bissextile")
    #N.B. dans ce cas annee est divisible par 100
    # mais PAS par 400 grâce au elif.
elif annee%4==0:
    print("bissextile")
    #dans ce cas annee n'est divisible
    # ni par 400 ni par 100 (autrement dit par 100)
    # mais est divisible par 4
 else :
    print("non bissextile")
```

Si on préfère les conditions négatives, on va dans l'ordre inverse :

```
annee=int(input("Entrez l'année de votre choix"))
if annee%4 !=0:
    print("cette année n'est pas bissextile")
elif annee%100 !=0:
    print("cette année est bissextile")
elif annee%400 !=0:
    print("cette année n'est pas  bissextile")
else:
    print("cette année est bissextile")
```

- (M2) Avec les `and, or` :

```
annee=int(input("Entrez l'année de votre choix"))
test=(annee%4==0) and ((annee%100!=0) or (annee%400==0))
print(test)
```

**7.1.** Le programme répète l'affichage de "Dis moi bonjour " tant que l'utilisateur ne tape pas exactement `bonjour`

**7.2. Jeu de Nim :** (i) Si le juste à $N \in \{4, 3, 2\}$ allumettes avant de jouer, il en prend $N - 1$ (ce qui est permis puisque $N - 1$ vaudrait $3, 2$ ou $1$) et l'autre joueur n'a aura qu'une devant lui et perd.

(ii) Au départ notre joueur a devant lui un nombre $N_0 = N$ d'allumettes tel que $N \not\equiv 1\ [4]$.

Notre joueur va prendre un nombre $a$ d'allumettes tel que $N - a \equiv 1\ [4]$.

Précisément si on note $r = N\%4$ le reste de la division euclidienne de $N$ par quatre, notre joueur prend :
• si $r = 0$, $a = 3$ allumettes car alors $N - a \equiv -3 \equiv 1\ [4]$
• si $r = 3$, $a = 2$ allumettes car alors $N - a \equiv 1\ [4]$
• si $r = 2$, $a = 1$ allumettes car alors $N - a \equiv 1\ [4]$.

A ce stade le joueur adverse a $N'$ allumettes devant lui où $N' \equiv 1\ [4]$.

Il va jouer et quel que soit sa façon de jouer, il nous laissera un nombre $N_1$ d'allumettes tel que $N_1 \not\equiv 1\ [4]$.

```
N=17
while N>1:
    print("Il y a ",N,"allumettes")
    coup_humain=int(input("Humain,combien en prends-tu ?"))
    N=N-coup_humain
    r=N%4
    if r==0:
        a=3
    elif r==2:
        a=1
    elif r==3:
        a=2
    print("Moi, j'en prends",a)
    N=N-a
print("Il reste seulement une allumette")
print("Tu es obligé de prend la dernière, j'ai gagné")
```