TP de Python numéro 2 - Annexe

*Semaine du **5 octobre**.*

I. Quelques fonctions à coder

Rappel : Si `L` est une liste :

- `len(L)` renvoie la longueur de la liste `L`
- Les éléments de `L` sont numérotés à partir de 0, donc de 0 à `len(L) -1`.
- `L[i]` renvoie le `i`-ième élément de `L` si `i` est un entier de $[\![0, \texttt{len(L)} - 1]\!]$, et renvoie une erreur *"list index out of range"* sinon.
- On peut boucler sur les éléments de `L` avec : `for e in L:`.
- La commande `L.append(a)` rajoute l'élément `a` à la fin de la liste `L`.

Exercice 1

Tester et comprendre le code suivant.

```
L=[1,2,4]
print("taille : ", len(L))
for x in L:
        print(x)
print("fin première boucle")
for i in range(len(L)):
        print("Le ",i,"ième élément de L est : ", L[i])
print("fin seconde boucle")
```

Exercice 2

Coder une fonction d'entête `def Moyenne(L):` prenant en entrée une liste de nombres `L` et renvoyant en sortie la moyenne des éléments de `L`.

Exercice 3

Coder une fonction d'entête `def GardePositifs(L):` prenant en entrée une liste de nombres `L` et renvoyant en sortie une nouvelle liste `G` constituée des éléments positifs de `L`.

On pourra, dans ce code, initialiser une variable de type liste `G`, vide, en écrivant :
`G=[]`
et rajouter les éléments de `L` à `G` seulement si ceux-ci sont positifs.

Les boucles while

En plus des boucles `for`, on peut faire exécuter à répétition un bloc de code à l'aide d'une boucle `while`. Voici un exemple:

```
def fonction1(x):
        n=0
        while n<x:
                n=n+1
        return(n)
```

Cette fonction :

- Prends en entrée un argument noté `x`. Cette fonction n'a de sens que si `x` est un nombre (flottant ou entier par exemple).
- Initialise une variable locale `n` à 0.

- Répète, tant que `n<x`, l'opération `n=n+1` (remarquez l'indentation). On dit que la variable `n` est incrémentée de 1 à chaque tour de boucle.
- Une fois la boucle `while` terminée (rupture d'indentation), la fonction renvoie `n`

Ainsi, la fonction `fonction1` prends en entrée un nombre, et renvoie le plus petit entier positif supérieur à ce nombre.

La syntaxe générale d'une boucle `while` est la suivante :

```
while CONDITION:
        ligne à exécuter 1
        ligne à exécuter 2
        etc...
```

où `CONDITION` doit renvoyer un booléen. Les lignes à lire à chaque tour de boucle sont, encore une fois, indentées.

Attention aux boucles while

Il est très facile de faire une boucle while ne s'arrêtant jamais. Au cas ou cela vous arrive, vous devez interrompre l'exécution de python (cherchez "interrupt the current running code" sur votre IDE). Pour que cela n'arrive pas, il faut avant tout que votre condition `CONDITION` puisse voir sa valeur de vérité changer au fur et à mesure que la boucle se répète.

Par exemple, la boucle suivante affichera "Bonjour" à tout jamais (comprenez, jusqu'à faire planter l'ordinateur) si on ne l'arrête pas.

```
n=0
while n<2:
        print("Bonjour")
        n=n-1
```

En effet, `n` est initialisé à 0 et, à chaque tour de boucle, se voit diminué de 1, donc `n` ne dépassera jamais 2 : la condition `n<2` sera toujours vraie.

Exercice 4

Que renvoie la fonction `mystere` suivante, prenant en entrée un nombre `x` ?

```
def mystere(x):
        if x>=0:
                n=0
                while n+1<=x:
                        n+=1 #n+=1 a le même effet que n=n+1
                return(n)
        else:
                while x<n:
                        n+=-1 #Plus généralement, n+=a a le même effet que n=n+a
                return(n)
```

Remarque. Les boucles `while` sont très utiles quand on ne sait pas à l'avance le nombre de tour de boucle dont nous aurons besoin.

Exercice 5

1. Coder une fonction d'entête `def puissancemax(x):` prenant en entrée un entier strictement positif `x` et renvoyant le plus grand entier `n` tel que $2^{\texttt{n}} \leq \texttt{x}$.
2. Considérons la suite $u = (u_n)_{n\in\mathbb{N}}$ définie par la relation de récurrence :

$$\begin{cases} u_0 = 1 \\ \forall n \in \mathbb{N}, u_{n+1} = 3u_n - 1 \end{cases}.$$

   Coder une fonction d'entête `def rangmax(x):` prenant en entrée un nombre `x` et renvoyant le plus petit rang `n` pour lequel $u_{\texttt{n}} > \texttt{x}$. On admet que la suite u tend vers $+\infty$, ce qui assure que cet entier existe quelque soit l'entrée `x`.

La suite de Syracuse ci-dessous est au coeur d'un problème toujours irrésolu. On ne sait pas démontrer que, quelque soit la valeur de son premier terme, elle finit toujours par retomber sur les valeurs $1, 4, 2, 1, 4, 2, \ldots$ (si la suite retombe sur 1, alors elle prendra ces valeurs 1 puis 4 puis 2 indéfiniment : elle sera *périodique à partir d'un certain rang*).

Exercice 6

Soit $A \in \mathbb{N}$. On appelle suite de Syracuse de premier terme A l'unique suite u telle que $u_0 = A$ et vérifiant la relation de récurrence :

$$\forall n \in \mathbb{N}, u_{n+1} = \begin{cases} \frac{u_n}{2} \text{ si } n \text{ pair} \\ 3u_n + 1 \text{ sinon} \end{cases} .$$

1. Coder une fonction d'entête `def Syr(A,n):` prenant en entrée deux entiers naturel `A` et `n` et renvoyant en sortie la valeur du `n` ième terme de la suite de Syracuse de premier terme A.
2. Coder une fonction d'entête `def testSyr(A):` prenant en entrée un entier `A` et renvoyant en sortie le premier rang n pour lequel la suite de Syracuse de premier terme `A` vaut 1. *Si ça devait ne jamais arriver, le code attendu tournerait indéfiniment.* On n'utilisera pas la fonction de la question précédente, pour des raisons d'efficacité algorithmique.

Affectations multiples

En `Python`, on peut affecter simultanément plusieurs variables. Par exemple :

```
a,b=1,2
```

a le même effet que :

```
a=1
b=2
```

Cela est de plus simultané. Par exemple,

```
a,b=b,a
```

inverse le contenu des variables `a` et `b` (si elles sont définies au préalable).

Exercice 7

Quelle est la différence entre la ligne précédente, et les deux lignes suivante?

```
a=b
b=a
```

Exercice 8

On rappelle l'algorithme d'Euclide permettant de déterminer le pgcd de deux entiers a et b positifs non tous nuls.

- Entrée : deux entiers a et b.
- Sortie : le pgcd de a et b.
- Affectations : $u := a$, $v := b$.
- tant que $v \neq 0$ faire :
- Affectation : $u := v$
- Affectation : v : reste de la division euclidienne de u par v.
- fin du "tant que"
- renvoyer u

Appliquer l'algorithme d'Euclide pour calculer pgcd$(98, 70)$. Puis, coder une fonction d'entête `def pgcd(a,b):` prenant en entrée deux entiers positifs `a` et `b` et renvoyant en sortie leur pgcd.