TD : Programmation (2), listes et fonctions. Corrigé

1 Boucles for

Exercice 1. *Fibonacci*. La suite de Fibonacci définie par $F_0 = F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n$ pour tout $n \geq 2$ n'est pas une suite récurrente.

1. Écrire une fonction `liste_termes_fibo(n)` prenant en entrée un entier n qu'on pourra supposer supérieur à 2, et renvoyant la liste des n premiers termes de la suite.
2. Écrire une fonction `terme(n)` calculant F_n, sans utiliser de liste (et donc sans utiliser la fonction précédente).

Corrigé.

```
def liste_termes_fibo(n):
    F=[1, 1]
    for i in range(n-2):
        F.append(F[-1]+F[-2])
    return F

def terme(n):
    a,b=1,1
    for i in range(n):
        c=b
        b=a+b
        a=c
    return a
```

Exercice 2. On peut très bien passer une fonction en paramètre d'une autre fonction. Écrire une fonction `terme(n,f,u0)` prenant en entrée un entier $n \geq 0$, une fonction f, et un premier terme u_0, et calculant le n-ème terme (u_n) de la suite récurrente définie par la relation $u_{n+1} = f(u_n)$ pour tout $n \geq 0$.

Corrigé.

```
def terme(n,f,u0):
    u=u0
    for i in range(n):
        u=f(u)
    return u
```

Exercice 3. Écrire une fonction `carre(n)` affichant à l'écran un carré de côté `n` constitué d'étoiles :

```
>>> carre(4)
****
****
****
****
```

Corrigé.

```
def carre(n):
    for i in range(n):
        print("*"*n)
```

Exercice 4. Écrire une fonction `triangle(n)` affichant à l'écran un triangle de hauteur `n` constitué d'étoiles :

```
>>> triangle(4)
*
**
***
****
```

Corrigé.

```
def triangle(n):
    for i in range(1,n+1):
        print("*"*i)
```

Exercice 5. Écrire une fonction `losange(n)` affichant à l'écran un carré de côté $2n-1$ constitué de zéros, dans lequel est inscrit un losange rempli de uns.

```
>>> losange(4)
0001000
0011100
0111110
1111111
0111110
0011100
0001000
```

Corrigé.

```
def losange(n):
    for i in range(1,2*n):
        a=abs(i-n)
        b=2*n-1-2*a
        print("0"*a+"1"*b+"0"*a)
```

2 Boucles while

Exercice 6. *1.* On rappelle que l'algorithme d'Euclide de calcul du PGCD consiste à effectuer des divisions euclidiennes successives, le PGCD étant le dernier reste non nul. Il est basé sur la propriété $\text{PGCD}(a,b) = \text{PGCD}(b,r)$ où r est le reste dans la division euclidienne de a par b. Écrire une fonction `PGCD(a,b)` calculant le PGCD de deux entiers que l'on pourra supposer strictement positifs.

2. En se rappelant que $\text{PGCD}(a,b) \times \text{PPCM}(a,b) = ab$, écrire une fonction `PPCM(a,b)`.

Corrigé.

```
def PGCD(a,b):
    while b>0:
        a,b=b,a%b
    return a

def PPCM(a,b):
    return a*b//PGCD(a,b)
```

Exercice 7. On rappelle que `randint` importée du module `random`, fournit des entiers aléatoires. `randint(0,1)` s'évalue en un entier qui est soit 0 soit 1. Écrire une fonction `temps_1()` sans argument, effectuant des tirages aléatoires, et renvoyant le nombre d'essais nécessaires avant l'obtention d'un 1.

Corrigé.

```
from random import randint

def temps_1():
    c=0
    a=0
    while a!=1:
        a=randint(0,1)
        c+=1
    return c
```

3 Parcours de listes

Exercice 8. On considère dans cet exercice des polynômes de la forme $P = \sum_{k=0}^{n-1} p_k X^k$. On représente un tel polynôme comme la liste `[`$p_0, \dots, p_{n-1}$`]` de ses coefficients. Écrire une fonction `evalue(P,x)` prenant en entrée une liste représentant un polynôme et x un réel, et renvoyant $P(x)$. Quel est le lien avec la représentation des entiers naturels dans une base ?

Corrigé.

```
def evalue(P,x):
    s=0
    for i in range(len(P)):
        s+=P[i]*x**i
    return s
```

Évaluer un nombre écrit en binaire revient à évaluer en 2 un polynôme à coefficients dans $\{0,1\}$.

Exercice 9. Écrire une fonction `tous_pairs(L)` prenant en entrée une liste `L` constituée d'entiers, et renvoyant un booléen indiquant si oui ou non tous les entiers de `L` sont pairs.

Corrigé. Ici, on parcourt directement les termes de la liste (plutôt que leurs indices). Les deux sont possibles !

```
def tous_pairs(L):
    for x in L:
        if x%2==1:
            return False
    return True
```

Exercice 10. Écrire une fonction `est_croissante(L)` renvoyant un booléen indiquant si les éléments de `L` sont dans l'ordre croissant.

Corrigé.

```
def est_croissante(L):
    for i in range(len(L)-1):
        if L[i]>L[i+1]:
            return False
    return True
```

Exercice 11. Écrire une fonction `est_present(L,x)` renvoyant un booléen indiquant si `x` est présent dans `L`. *Rappel :* une instruction de la forme `return ...` interrompt immédiatement la fonction.

Corrigé.

```
def est_present(L,x):
    for y in L:
        if y==x:
            return True
    return False
```

Remarque : `x in L` effectue précisément ce test.

Exercice 12. *Un tri.*

```
def echange(L,i,j):
    L[i], L[j]=L[j], L[i]

def retablit_croissance(L):
    i=len(L)-1
    while i>0 and L[i]>L[i-1]:
        echange(L,i,i-1)
        i-=1

def copie_triee(L):
```

```
    T=[L[0]]
    for i in range(1,len(L)):
        T.append(L[i])
        retablit_croissance(T)
    return T
```

Exercice 13. *Un autre tri.*

```
def parcours(L):
    for i in range(len(L)-1):
        if L[i]>L[i+1]:
            echange(L,i,i+1)

def tri(L):
    for i in range(n):
        parcours(L)
```

4 Boucles imbriquées

Exercice 14.

```
def somme(T):
    s=0
    for i in range(len(T)):
        for j in range(len(T[0])):
            s+=T[i][j]
    return s
```

Exercice 15.

```
def genere(n,m):
    T=[[0]*m for i in range(n)]
    c=0
    for i in range(n):
        for j in range(m):
            T[i][j]=c
            c+=1
    return T
```

Exercice 16.

```
def pos_du_min(T):
    im, jm=0, 0
    for i in range(len(T)):
        for j in range(len(T[0])):
            if T[i][j]<T[im][jm]:
                im, jm=i, j
    return im, jm
```