TD : Programmation (2), listes et fonctions

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).

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 $n \geq 0$.

Pour les trois exercices suivants, il est préférable de lire le paragraphe sur `print` du polycopié de programmation.

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

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

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

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

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
```

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)`.

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.

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 ?

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.

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

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.

Exercice 12. *Un tri.*

1. Écrire une fonction `echange(L,i,j)` échangeant les éléments aux indices i et j d'une liste `L`.
2. Écrire une fonction `retablit_croissance(L)` prenant en entrée une liste `L` supposée non vide, et modifiant la liste pour qu'elle soit triée à la fin de l'exécution. On suppose que tous les éléments de la liste sont dans l'ordre croissant, excepté peut-être le dernier élément. On utilisera une boucle `while`.
3. À l'aide de la fonction précédente, écrire une fonction `copie_triee(L)` prenant en entrée une liste `L` (qu'on pourra supposer non vide) et renvoyant une liste triée constituée des mêmes éléments. On utilisera une boucle `for`.

Exercice 13. *Un autre tri.*

1. Écrire une fonction `parcours(L)` parcourant la liste de gauche à droite. Si on trouve deux éléments de la liste d'indices i et $i+1$ tels que `L[i+1]` $<$ `L[i]`, on les échange.
2. Écrire une fonction `tri(L)` effectuant autant de parcours que la taille de la liste.
3. Vérifiez que cette fonction tri la liste dans l'ordre croissant.

4 Boucles imbriquées

Un moyen de travailler en Python avec des tableaux à deux dimensions est d'utiliser des listes de listes : la première liste représente la première ligne du tableau, de même la deuxième liste la deuxième ligne, etc... Si `T` est une telle liste de listes représentant un tableau, l'élément `T[i][j]` donne l'élément sur la i-ème ligne et la j-ème colonne, en indexant à partir de zéro et du coin en haut à gauche. Voici par exemple un tableau et sa représentation en Python.

```
T=[[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11]]
```

0	1	2	3
4	5	6	7
8	9	10	11

Dans un tel tableau, le nombre de lignes est donné par `len(T)` et le nombre de colonne par le nombre d'éléments d'une ligne quelconque, `len(T[0])` par exemple.

Exercice 14. Écrire une fonction `somme(T)` faisant la somme des éléments d'un tableau représenté comme liste de listes.

Exercice 15. Écrire une fonction `genere(n,m)` renvoyant un tableau constitué des entiers de 0 à $nm-1$, ayant n lignes et m colonnes. La liste `T` précédente est celle qu'on obtient avec `genere(3, 4)`. Attention : pour initialiser la liste de listes, on utilisera `[[0]*m for i in range(n)]` et non `[[0]*m]*n`, qui ne crée qu'une seule ligne en mémoire (les lignes du tableau seraient physiquement toutes les mêmes).

Exercice 16. Écrire une fonction `pos_du_min(T)` prenant en entrée un tableau de nombres et renvoyant le couple `(i,j)` tel que `T[i][j]` soit l'élément minimal du tableau.