Lycée Loges

Chapitre 2

Algorithmes classiques TP

Simon Dauguet
simon.dauguet@gmail.com

2 octobre 2025

Exercice 1 :
On considère un tableau contenant des entiers ou des tableaux d'entiers. Proposer une fonction `maxProfond(tab: list) -> int`, qui renvoie le maximum des valeurs contenues dans le tableau et ses sous-tableaux.

Exercice 2 :
Proposer une fonction `decompPremier(n: int) -> list`, qui renvoie la décomposition du nombre n en produit de nombres premiers. Par exemple, `decompPremier(300)` renvoie `[2,2,3,5,5]`.

Exercice 3 :
On considère `phrase` une chaîne de caractères composée que de minuscules.

a) Proposer une fonction `occLettre(phrase: str, let: str) -> int`, qui compte le nombre de lettre `let`.

b) Proposer une fonction `maxLettres(phrase: str) -> str`, qui renvoie l'une des lettres le plus utilisée dans `phrase`.

c) Proposer une fonction `freqLettres(phrase: str) -> list`, qui renvoie un tableau de longueur 26 contenant des tuples (lettres, fréquence). Par exemple, `freqLettres( "ababa")` renvoie `[('a',0.6),('b',0.4),... ('z',0)]`.
On rappel que `ord('a')` renvoie 97 (le code ascii de la lettre `a`)

Exercice 4 :
On suppose que `phrase` est une suite de mots composés que de lettres minuscules sans accents et séparées par des espaces.

a) Sans utiliser la méthode `split()`, proposer une fonction `phrase2tab(phrase: str) -> list`, qui renvoie la liste des mots composants la chaîne de caractères `phrase`. Par exemple, `phrase2tab( "la tasse est chaude")` renvoie `['la','tasse','est','chaude']`.

b) Proposer une fonction `plusLong(phrase: str) -> str`, qui renvoie l'un des mots de longueur maximale.

c) Proposer une fonction `plusLongs(phrase: str) -> list`, qui renvoie tous les mots de longueur maximale.

d) Proposer une fonction `premierAlpha(phrase: str) -> str`, qui renvoie le premier mot dans l'ordre alphabétique de la chaîne de caractères `phrase`.

Exercice 5 (Nombre d'argent) :
On considère la suite de Pell généralisée, définie par

$$\begin{cases} u_0 = 0,\ u_1 = 1 \\ \forall n \in \mathbb{N},\ u_{n+2} = 2u_{n+1} + u_n \end{cases}$$

Le nombre d'argent φ_2 est défini par $\varphi_2 = \lim_{n\to+\infty} \frac{u_{n+1}}{u_n}$.

1. Écrire une fonction `pell(n:int) -> int` qui prend en entrée un entier `n` et renvoie la valeur de u_n. *On fera attention aux cas particuliers.*
2. Adapter la fonction précédente pour qu'elle devienne une fonction `rapport(n:int) -> float` qui prend en argument un entier `n` et renvoie la valeur de $\frac{u_{n+1}}{u_n}$. On prendra garde à ne pas appeler la fonction `pell` dans `rapport`.
3. Adapter la fonction précédente pour qu'elle devienne une fonction `argent(eps:float)` qui prend en argument un flottant `eps` et renvoie la valeur de $\frac{u_{n+1}}{u_n}$ pour la première valeur de `n` telle que $\left|\frac{u_{n+2}}{u_{n+1}} - \frac{u_{n+1}}{u_n}\right| < \varepsilon$.
4. Donner une valeur approchée du nombre d'argent φ_2 avec 4 chiffres après la virgule. Conjecturer la valeur exacte de φ_2.

Exercice 6 (Listes de listes de listes) :
On ne considérera dans cet exercice, que des listes d'entiers.

1. Écrire une fonction `min1D(L:list) -> tuple` qui prend en argument une liste d'entiers non vide `L` et qui renvoie le minimum de `L` ainsi que l'indice de ce minimum.
   Si plusieurs indices sont possibles, on prendra le dernier d'entre eux.
   On renverra une erreur si `L` est vide.
2. Écrire une fonction `zero1D(L:list) -> list` qui prend en argument une liste non vide d'entiers `L` et renvoie la liste `L` après avoir supprimé toutes les occurrences du minimum de `L`. On fera attention à ne pas modifier la liste `L`.
3. Écrire une fonction `Lzero2D(L:list) -> list` qui prend en argument une liste de listes d'entiers et renvoie la liste des listes de `L` en enlevant, pour chacune des listes de `L`, toutes les occurrences de leur minimum respectifs.
4. Écrire une fonction `Czero2D(L:list) -> list` qui fait la même chose que `Lzero2D(L)` mais en agissant sur les colonnes de `L`. On supposera que toutes les listes de `L` ont la même longueur. Sinon, on renverra un message d'erreur.

Exercice 7 :
On souhaite expérimenter numériquement les décompositions d'entiers naturels à l'aide de trois cubes. Une idée possible serait de décomposer un nombre avec le plus grand cube possible et recommencer avec le reste. On propose alors l'algorithme suivant :

```
c1 ← ⌊∛n⌋
Tant que c1 > 0, faire :
    nb ← n − c1³
    c2 ← ⌊∛nb⌋
    Si c1 < c2, alors
        c1, c2 ← 0
    Tant que c2 > 0, faire :
        nb2 ← nb − c2³
        c3 ← ⌊∛nb2⌋
        Si c2 < c3, alors
            c2, nb2 ← 0
        Si nb2 est un cube non nul, alors
            Renvoyer la décomposition
        Décrémenter c2 de 1
    Décrémenter c1 de 1
Renvoyer une réponse vide
```

Remarque : en posant, s'il existent, c_1, c_2, c_3 les trois nombres de la décomposition ($n = c1^3 + c2^3 + c3^3$), nous avons $\lfloor \sqrt[3]{n} \rfloor \geqslant c_1 \geqslant c_2 \geqslant c_3 \geqslant 1$ qui fournit un critère d'arrêt.

Par exemple, avec $n = 155$,
On essaye avec $c_1 = \left\lfloor \sqrt[3]{155} \right\rfloor = 5$.
D'où $nb = 155 - c_1^3 = 30$
mais $nb - c_2^3 = 3$ n'est pas un cube.
On décrémente c_1.

On essaye avec $c_1 = 4$.
D'où $nb = 155 - c_1^3 = 91$
et puisque $nb - c_2^3 = 27$ est un cube non nul
155 est décomposable.

On a $155 = 4^3 + 4^3 + 3^3$.

a) Décomposer 36 et 153 comme somme de trois cubes. 151 est-il décomposable ainsi ?

b) Comment vérifier qu'un nombre n est un cube ? (on pourra utiliser la fonction `round()` qui donne un arrondi à l'entier et se rappeler que $\sqrt[3]{n} = n^{1/3}$)

c) Proposer une fonction `decomp3cubes(n: int) -> list`, qui renvoie une liste vide s'il est impossible de décomposer n à l'aide de trois cubes ou la liste des trois nombres à mettre au cube pour trouver n.

d) Combien de nombres inférieurs à 1000 sont-ils ainsi décomposables ? Parmi eux, combien sont premiers ?