PYTHON : BILAN DE COMPÉTENCES ET APPROFONDISSEMENT SUR LES LISTES

1 BILAN DE COMPÉTENCES

- À ce stade de l'année, vous devez savoir programmer sans hésitations :
  - **compétence 0** : manier la syntaxe sur les listes (en particulier le slicing).
    Se référer au Vademecum pour la liste des commandes à connaître.
  - **compétence 1** : construction d'une liste par récurrence.
  - **compétence 2** : parcours d'une liste pour y trouver une information.
  - **compétence 3** : extraction de sous listes.
  - **compétence 4** : somme, produit, moyenne... des éléments d'une liste.
- Dans la partie approfondissement nous développerons ces compétences :
  - **compétence 1bis** : création d'une liste par compréhension.
  - **compétence 2bis** : parcours de deux listes en parallèle.

2 EXERCICES À SAVOIR REFAIRE

EXERCICE 1 (compétence 1 : liste des termes d'une suite) On définit la suite (u_n) par

$$u_0 \in \mathbb{R}_+ \quad \text{et} \quad \forall n \in \mathbb{N},\ u_{n+1} = 3u_n + 1$$

Programmer une fonction **premiersTermes**(`n,u0=1`) qui renvoie la liste des termes de la suite de u_0 à u_n.

Les termes successifs de la suite seront calculés par récurrence.

EXERCICE 2 (Recherche)

1) (**compétence 2**) Programmer une fonction **maximum**(`liste`) qui recherche la valeur du maximum d'une liste.
*On n'utilisera pas la fonction **max** de Python.*

2) (**compétence 2**) Programmer une fonction **minimum**(`liste`) qui recherche la valeur du minimum d'une liste.
*On n'utilisera pas la fonction **min** de Python.*

3) (**compétence 2**) Programmer une fonction **maxIndice**(`liste`) qui recherche le premier indice correspondant au maximum d'une liste.
On veillera à ne parcourir la liste qu'une seule fois.

4) (**compétence 2**) Programmer une fonction **superieur**(`liste, M`) qui recherche le premier élément de la liste strictement supérieur à `M`. Si aucun des éléments de la liste n'est supérieur à `M`, alors la fonction renvoie `M`.

5) (**compétence 2**) Programmer une fonction **nbreSup**(`liste, M`) qui recherche le nombre d'éléments de la liste strictement supérieurs à `M`.

6) (**compétence 2**) Programmer une fonction **proportionSup**(`liste, M`) qui renvoie la proportion d'éléments de la liste qui sont strictement supérieurs à `M`.

7) (**compétence 1-2-3**) Programmer une fonction **listeSup**(`liste, M`) qui renvoie la sous liste des éléments strictement supérieurs à `M`.

8) (**compétence 1-2-3**) Programmer une fonction **listeTriSup**(`liste, M`) effectue la même opération que la fonction **listeSup**, mais sur une liste supposée triée par ordre croissant.

EXERCICE 3 (compétence 4 : moyennes)

Dans la suite du TD nous utiliserons la fonction **round** pour arrondir toutes les moyennes 0,1 près (pas les autres résultats). La fonction **round** est une commande *native* de Python[1]. Vous pouvez consulter l'aide interactive pour connaître sa syntaxe : **help(round)**.

[1] Pas besoin de charger une bibliothèque

1) Définir une fonction **moyenne**(`liste`) qui renvoie la moyenne arithmétique des éléments d'une liste.

On rappelle que la moyenne arithmétique de $a_1, a_2, \cdots, a_n$ est définie par $a = \dfrac{1}{n}\sum_{k=1}^{n} a_k$

Pour `notes=[8,10,5,7,9,11,17,15]`, la fonction doit renvoyer la valeur 10.2.

2) Définir une fonction **moyenneGeom**(`liste`) qui renvoie la moyenne géométrique des éléments d'une liste.

La moyenne géométrique de $a_1, a_2, \cdots, a_n$ est définie par $g = \left(\prod_{k=1}^{n} a_k\right)^{\frac{1}{n}}$

Pour `notes=[8,10,5,7,9,11,17,15]`, la fonction doit renvoyer la valeur 9.6.

3) Définir une fonction **moyenneHarm**(`liste`) qui renvoie la moyenne harmonique des éléments d'une liste (on les suppose tous positifs non nuls).

La moyenne harmonique de $a_1, a_2, \cdots, a_n$ est définie par h tel que $\dfrac{1}{h} = \dfrac{1}{n}\sum_{k=1}^{n} \dfrac{1}{a_k}$

Pour `notes=[8,10,5,7,9,11,17,15]`, la fonction doit renvoyer la valeur 8.9.

3 Exercices d'approfondissement

Exercice 4 (compétence 1bis : listes en compréhension)

1) Entrer les commandes et essayer de comprendre ce qu'elles font (ne pas hésiter à essayer avec d'autres commandes de son cru).

```
carres=[(i+1)**2 for i in range(10)]
notes=[8,10,5,7,9,11,17,15]
double=[i*2 for i in notes]
table=[(i+1)*(j+1) for i in range(5) for j in range(3)]
```

2) En une seule ligne, définir la liste de tous les nombres de 1 à 100.

3) En une seule ligne, définir une liste de 20 nombres aléatoires compris entre 1 et 6.

On pourra utiliser la fonction **randrange** du module `random`.

4) Programmer une fonction **echelle** qui prend en argument une liste de notes. Et qui renvoie la liste de ces mêmes notes toutes multipliées par un même coefficient de telle sorte que la note maximale soit ramenée à 20.

Les notes seront arrondies à 0,1 près.

Par exemple pour `notes=[8,10,5,7,9,11,17,15]`, la fonction doit renvoyer

`[9.4, 11.8, 5.9, 8.2, 10.6, 12.9, 20.0, 17.6]`.

5) Tester les commandes suivantes :

```
dessus=[i for i in notes if i>=10]
dessous=[i for i in notes if i<10]
```

6) En utilisant la méthode présentée à la question précédente, programmer une fonction **octa**(`n`) qui renvoie une liste de tous les nombres x compris entre 1 et n tels que x^2+2x soit divisible par 8.

En déduire une conjecture sur les valeurs de ces nombres. Vous pourrez essayer de la démontrer chez vous.

7) Programmer la fonction **listeSup** de la question 7) de l'exercice 2 en utilisant la méthode par compréhension.

Exercice 5 (compétence 2bis : parcours simultanés de listes)

Définir une fonction **moyennePonderee** qui prend en argument deux listes de même longueur. La première liste contient les notes et la seconde les coefficients correspondants. La fonction renvoie leur moyenne pondérée.

Avec les coefficients `coeffs=[6,2,6,1,1,6,1,2]` et la liste de notes précédente, la fonction **moyennePonderee**(`notes,coeffs`) renvoie 9.1.