TP 4 : Boucles imbriquées

I Calculs de sommes / produits

Exercice 1 *Calculer une somme*

Voici une fonction qui calcule : $\sum_{k=0}^{n} \sqrt{k}$

```
import math #Bibliothèque math

def somme(n: int)-> float:
    s = 0 #La somme au départ vaut toujours 0
    for k in range(0,n+1):
        s = s + math.sqrt(k)
    return s
```

Modifier cette fonction afin qu'elle calcule $\sum_{k=0}^{n} e^k + k^2$

```
import math #Bibliothèque math

def somme2(n: int)-> float:
```

Exercice 2 *Calculer une somme*

Écrire une fonction qui calcule : $\sum_{i=1}^{n} 2^i$.

```
def somme(n: int)-> int:
```

Exercice 3 *Calculer un produit*

Écrire une fonction qui calcule :

$$\prod_{k=1}^{n} (2+k)$$

```
def produit(n: int)-> int:
```

Exercice 4 *Calculer une somme double*
Écrire une fonction qui calcule :

$$\sum_{k=1}^{n}\sum_{i=1}^{n} e^{-ik}$$

```
def somme_somme(n: int)-> float:
```

Exercice 5 *Calculer une somme de produits*
Écrire une fonction qui calcule :

$$\sum_{k=1}^{n}\prod_{l=0}^{n-1} \sin(k+l)$$

```
def somme_prod(n: int)-> float:
```

Combien de parcours de boucle fait-on quand on appelle chacune des fonctions précédentes ?

II Parcours d'un tableau à deux dimensions

Les tableaux à deux dimensions sont représentés dans python comme une liste de listes : chaque ligne est stockée comme une liste, et on construit une liste avec toutes ces listes. Par exemple, le tableau

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

est représenté par

```
T = [[0,1,2,3,4] , [5,6,7,8,9] , [10,11,12,13,14]]
```

Compléter les résultats des commandes :

1. `T[0]`
2. `T[1]`
3. `T[1][1]`
4. `T[0][1]`

5. `T[1][0]`
6. `T[1:]`
7. `len(T)`
8. `len(T[0])`

La valeur de la case (i, j) est accessible par la commande `T[i][j]`

Exercice 6 *Parcourir un tableau à deux dimensions*
Étant donné un tableau (à deux dimensions) comprenant des entiers, écrire un algorithme qui compte le nombre d'entiers pairs.

```
def nb_pairs(L: list) -> int:
```

Exercice 7
Que renvoient les commandes suivantes ?

1. `[k for k in range(4)]`
2. `[k*2 for k in range(5)]`
3. `[k+5 for k in range(4)]`
4. `[[1,2] for k in range(4)]`

Exercice 8 *Créer un tableau à deux dimensions*
Écrire une fonction qui renvoie un tableau à deux dimensions de taille $m \times n$ avec des zéros comme entrées.

```
def grille_vide(m:int, n:int) -> list:
```

Exercice 9 *Remplir un tableau à deux dimensions*
Écrire une fonction qui renvoie un "échiquier" de taille $m \times n$, c'est-à-dire un tableau à deux dimensions de taille $m \times n$ avec un zéro dans la case (i, j) si $i + j$ est pair et 1 sinon. On pourra utiliser la fonction `grille_vide` de l'exercice précédent.

```
def echiquier(m:int, n:int) -> list :
```

III Recherche d'un facteur dans un texte

Exercice 10 *Rechercher un mot dans un texte*
Écrire un algorithme qui envoie `True` si `mot` est contenu dans la chaîne de caractères `texte` et `False` sinon.
Voici quelques idées :

- On boucle sur (presque tous) les caractères du `texte`
- Si le caractère du `texte` est égal à la première lettre du `mot`, alors :
  - On initialise un compteur à 1 (nombre de lettres du `mot` trouvées)
  - On boucle sur les lettres du `mot` : à chaque bonne lettre on incrémente le compteur
  - Si le compteur a la même valeur que la longueur du `mot`, on renvoie `True`
- Si on est arrivé là, on renvoie `False`

```
def recherche(texte:str, mot:str)->bool:
```

→ **Testez votre algorithme !**
Que doit renvoyer `recherche("pitapipapa","pipa")` ?
Et `recherche("pitapipapa","tapa")` ?

IV Recherche de deux valeurs les plus proches

Exercice 11
Écrire un algorithme qui envoie la position des deux valeurs les plus proches dans une liste (dont l'écart en valeur absolue est minimal).
Voici quelques idées :

- On initialise les positions de départ à 0 et 1
- On initialise l'écart dans une variable adaptée
- On parcourt la liste avec un indice i :
  - Pour chaque valeur `L[i]`, on calcule l'écart entre `L[i]` et `L[j]` où $j > i$.
  - Si l'écart obtenu est inférieur à l'écart enregistré, on le remplace et on sauvegarde les valeurs i et j.
- On renvoie le couple (i, j).

```
def rech_val_proches(L: list) -> tuple:
```

→ **Testez votre algorithme !**
Que doit renvoyer `rech_val_proches([32,16,8,4,2,1])` ?
Et `rech_val_proches([5,3,10,8,25,18,11,14])` ?

V Tri à bulles

```
def une_remontee(tab: list) -> list :
    n = len(tab)
    for j in range(0,n-1):
        if tab[j] > tab[j+1]:
            tab[j] , tab[j+1] = tab[j+1] , tab[j]
    return tab
```

→ **Que fait la fonction ci-dessus ?**

Exercice 12

Tri de liste Écrire un algorithme qui trie une liste de nombres en faisant remonter les plus grands à la fin.
Le principe du tri à bulles est le suivant :

- À chaque étape i, on fait remonter le plus grand élément de `L[:n-i]` en position $n-i-1$:
  - On compare le premier élément de la liste avec son suivant et s'il est plus grand alors on échange les deux
  - On répète cette opération jusqu'à la position $n-i$
- On renvoie la liste (facultatif)

```
def tri_bulle(L: list) -> list :
```