TD n° 1 : révisions des fondamentaux

Tout d'abord, ci-contre le lien vers un notebook de rappels sur les listes, les chaînes de caractères et les tuples. Vous y trouverez les principales commandes à connaître à leurs sujets ainsi que les différences qui les séparent (et en particulier quelques erreurs à ne pas commettre).

Exercice 1. *Parcours d'une liste*
On souhaite disposer d'une fonction `recherche` d'entête :

```
def recherche(element: float, tab: [float]) -> int:
```

qui renvoie l'indice de la première occurrence de `element` s'il se trouve dans `tab` et `-1` sinon. Par exemple :

```
>>> recherche(3.1, [2.4, 3.1, -4.6, 3.1])
1
>>> recherche(1.0, [-2.3, 3.9, 4.5, 3.0])
-1
```

1. Quelle stratégie employer ?
2. Donner un jeu de tests permettant de tester le bon fonctionnement de votre future fonction (penser aux cas limites).
3. Écrire une fonction `recherche` répondant au problème.
4. Modifier votre fonction afin qu'elle renvoie l'indice de la dernière occurrence, si elle existe, au lieu de la première.

⇝ Faire l'épreuve 2.09 (*Absence de diversité*) du challenge.

Exercice 2. *Parcours d'une chaîne de caractères*
Écrire une fonction d'entête :

```
def sans_espace(texte: str) -> str:
```

qui renvoie une nouvelle chaîne de caractères identique à `texte` mais dans laquelle tous les espaces ont été supprimés. Par exemple :

```
sans_espace("Ada Lovelace a écrit le premier programme informatique.")
'AdaLovelaceaécritlepremierprogrammeinformatique.'
```

⇝ Faire l'épreuve 2.10 (*Occurrences d'un caractère dans un texte*) du challenge.

Exercice 3. *Recherche d'un extremum*

1. Écrire une fonction `minimum` d'entête :

   ```
   def minimum(tab: [float]) -> (int, float):
   ```

   qui renvoie l'indice et la valeur du plus petit élément de `tab` (s'il est atteint plusieurs fois, n'importe quelle occurrence convient).
   Par exemple, `minimum([3.1, 2.8, -5.7, 1.0, -4.6])` renvoie `(2, -5.7)`.
2. Que se passe-t-il si la liste donnée en argument est vide ?
3. Écrire une fonction `mini_maxi` qui prend en argument une liste de nombres et qui renvoie un couple composé de la valeur du minimum et de la valeur du maximum de la liste donnée en argument. On fera en sorte qu'une erreur soit levée si la liste est vide.

⇝ Faire l'épreuve 2.05 (*Trouver le plus grand élément d'une liste*) du challenge.

Exercice 4. *Somme*

1. Écrire une fonction `moyenne` d'entête :
   ```
   def moyenne(valeurs: (int)) -> float:
   ```
   qui renvoie la moyenne des éléments contenus dans le tuple `valeurs`.
2. Quelle est la complexité de cette fonction en fonction de la longueur n de la liste `valeurs` ?

⇝ Faire les épreuves 2.02 (*Somme des premiers entiers*) et 2.04 (*Calcul d'une somme*) du challenge.

Exercice 5. *Boucles et complexités*
On considère de nouveau la fonction `moyenne` de l'exercice précédent et on s'intéresse aux deux fonctions suivantes permettant de calculer la variance d'une liste de nombres.

```
def variance_v1(valeurs: [int]) -> float:
    var = 0
    for element in valeurs:
        var += (element - moyenne(valeurs))**2
    return var / len(valeurs)
```

```
def variance_v2(valeurs: [int]) -> float:
    var = 0
    moy = moyenne(valeurs)
    for element in valeurs:
        var += (element - moy)**2
    return var / len(valeurs)
```

Donner la complexité de chacune en fonction de la longueur n de la liste passée en argument. Commenter.

Exercice 6. *Image et boucles imbriquées*
On considère une image en nuances de gris représentée par une matrice d'entiers. Chaque pixel est représenté par un entier dont la valeur peut aller de 0 pour noir à 255 pour blanc, les intermédiaire étant les différents gris.

1. Écrire une fonction d'entête
   ```
   def noir_blanc(img: [[int]], seuil: int) -> None:
   ```
   qui **modifie** l'image de sorte qu'en dessous de la valeur de `seuil` donnée, le pixel devient noir et au-dessus blanc.
2. Écrire une fonction d'entête
   ```
   def rotation(img: [[int]]) -> [[int]]:
   ```
   qui renvoie une nouvelle image issue de la première après une rotation d'un quart de tour dans le sens trigonométrique.

⇝ Faire les épreuves 3.08 (*Une paire de somme donnée*) et 3.12 (*Éloge de la différence*) du challenge.

Exercice 7. *Dichotomie*
Écrire une fonction `recherche_dicho` d'entête :

```
def recherche_dicho(x: float, tab: [float]) -> bool:
```

où `tab` est supposée triée par ordre croissant et qui

- renvoie `True` si `x` est dans `tab` et `False` sinon ;
- procède par dichotomie (pour avoir une complexité logarithmique).

⇝ Faire l'épreuve 4.04 (*Deviner un nombre*) du challenge.