Initiation aux tableaux

Bonus 02

Exercice 1

1. Écrire en Python une fonction `variance(tab)` qui permet de calculer la variance du tableau `tab`. Pour rappel, la variance d'un tableau est la moyenne des $(x-\mu)^2$, où x parcourt les éléments de `tab` et μ est la moyenne de `tab`.
2. Écrire en Python une fonction `ecart_type(tab)` qui permet de calculer l'écart-type de `tab`. Pour rappel, l'écart-type est la racine carrée de la variance.
3. La fonction `sorted` permet de trier un tableau. Écrire une fonction `mediane(tab)` qui calcule la médiane d'un tableau.

Exercice 2

On dit qu'un mot est un carré s'il peut s'écrire comme deux fois un mot, comme par exemple « bonbon » ou « mimi » : on écrit « bonbon = bon^2 ». De manière générale, pour $k \in \mathbb{N}^*$, on dit qu'un mot est une puissance k s'il peut s'écrire une répétition de k fois un même mot.
Lorsqu'un mot est une puissance, on appelle racine primitive le mot dont il est une puissance : la racine primitive de « coucou » est « cou », celle de « blablabla » est « bla ». Si un mot n'est pas une puissance, il est sa propre racine primitive (on dit que le mot lui-même est primitif).

1. Écrire une fonction `est_carre(mot)` qui renvoie `True` si `mot` est un carré.
2. Écrire une fonction `est_puissance(mot)` si `mot` est une puissance k pour un $k \geqslant 2$.
3. Écrire une fonction `racine_primitive(mot)` qui renvoie la racine primitive de `mot`.
4. Écrire une fonction `est_primitif(mot)` qui renvoie `True` si `mot` est primitif.

Exercice 3

Pour $\mathtt{n} \in \mathbb{N}^*$, on dit qu'un tableau est une **permutation** de $[\![1, \mathtt{n}]\!]$ s'il contient exactement une occurrence de tout élément de $[\![1, \mathtt{n}]\!]$. On peut aussi considérer que ce tableau décrit une bijection de $[\![1, \mathtt{n}]\!]$ dans lui-même.

1. Quelle est la longueur d'un tel tableau ?
2. Écrire une fonction `est_permutation(tab)` qui renvoie `True` si `tab` est une permutation.
3. Pour `tab` une permutation, on dit que $(i,j) \in [\![1,\mathtt{n}]\!]^2$ est une inversion de `tab` si $i < j$ et `tab[`i`]` $>$ `tab[`j`]`. Par exemple, le tableau `[1,5,3,4,2]` admet $(1,2)$ comme inversion : en effet, $1 < 2$ et `tab[1]` $= 5 > 2 =$ `tab[2]`.
   Écrire une fonction Python `nb_inversions(tab)` qui renvoie le nombre d'inversions dans `tab`.
4. Pour `tab` une permutation, en notant K son nombre d'inversions, on appelle signature de `tab` la valeur $(-1)^K$. Écrire une fonction `signature(tab)` qui renvoie la signature de `tab` (si `tab` n'est pas une permutation, on renverra 0).
5. Les permutations étant comparables à des fonctions, on doit pouvoir les composer. Écrire une fonction `composition(tab1,tab2)` qui renvoie la composition de deux permutations (dans le sens `tab1` $\circ$ `tab2`).
6. Comparer la signature de deux tableaux `tab1` et `tab2` avec la signature de `composition(tab1,tab2)`. Empiriquement, qu'observez-vous ?
   Vous le démontrerez plus tard en mathématiques.

Exercice 4 : un point sur les variables

Le langage Python met des limites aux noms des variables : tout n'est pas possible. Tester les commandes du format « <*nom de variable*> `= 2` » avec les noms suivants : `abc`, `a`, `a1`, `1234`, `1a`, `r2d2`, `for`, `A`, `_A`, `marie_pierre`, `marie-pierre`, `tarte aux quetsches`, « `a,b` », `l'animal`.

Exercice 5 : sur la copie des tableaux

1. Tester dans l'invite de commandes les deux premières successions de commandes ci-contre. Qu'observez-vous ?
2. La commande `a.copy()` renvoie une copie du tableau `a`. Adapter l'une des commandes à droite pour obtenir le résultat escompté.
3. Écrire vous-même une fonction `copie(tab)` qui renvoie une copie du tableau `tab`.
4. Maintenant, tester la 3ème succession de commandes. Que se passe-t-il ?
5. La bibliothèque `copy` contient une fonction `deepcopy`, qui renvoie ce qu'on appelle une copie profonde : la syntaxe est `copy.deepcopy(a)`. Tester et expliquer ce qu'est une copie profonde.

```
>>> a = 12
>>> b = a
>>> a = 15
>>> b
```

```
>>> a = [1,2,3]
>>> b = a
>>> a[0] = 10
>>> b
```

```
>>> a = [[1,2],[3,4],[5,6]]
>>> b = a.copy()
>>> a[0][1] = 10
>>> b
```