Révisions ITC MPSI

October 27, 2025

Exercice 1

Écrire une fonction `est_croissante` qui prend en trée une liste d'entiers `L` et renvoie `True` si `L` est croissante et `False` sinon.

Exercice 2

Sans utiliser la syntaxe `L[::-1]` et à l'aide d'une boucle `for`, écrire une fonction `inverser` qui prend en entrée une liste d'entiers `L` et renvoie une liste contenant les mêmes éléments dans l'ordre inverse.

Exercice 3

On considère une liste de n entiers compris entre 1 et 99 (inclus). Par exemple la liste de $n = 20$ entiers :`[16,2,85,27,9,45,98,73,12,26,46,25,26,49,18,99,10,86,7,42]` qui a été obtenue par la commande : `liste_20 = [randint(1,99) for i in range(20)]`

Écrire une fonction `somme_deux_consecutifs_100(liste)` qui teste s'il existe deux éléments consécutifs de la liste dont la somme vaut 100. La fonction renvoie `Vrai` ou `Faux` (mais elle peut aussi afficher les nombres et leur position pour vérification). Pour l'exemple donné la fonction renvoie `False`.

Exercice 4

1. Écrire une fonction `supprimer_element(liste,element)` renvoyant une liste qui contient tous les éléments sauf ceux égaux à l'élément spécifié.

   Par exemple `supprimer_element([8,7,4,6,5,4],4)` renvoie la liste `[8,7,6,5]` (tous les éléments égaux à 4 ont été supprimés).

2. Écrire une fonction `melange` qui prend en entrée deux listes `L1` et `L2` et renvoie une nouvelle liste :

   `[L1[0], L2[0], L1[1], L2[1], L1[2], L2[2], ...]`

   Dans le cas où les deux listes ne sont pas de même longueur, le programme ajoutera les éléments en surplus à la fin de la liste `L`. Par exemple :

   `melange([], [])` renvoie `[]`,
   `melange([1, 2, 3], [4, 5, 6])` renvoie `[1, 4, 2, 5, 3, 6]`,
   `melange([], [1, 2, 3])` renvoie `[1, 2, 3]`,
   `melange([1, 2, 3], [])` renvoie `[1, 2, 3]`,
   `melange([1, 2, 3], [4, 5, 6, 7, 8, 9])` renvoie `[1, 4, 2, 5, 3, 6, 7, 8, 9]`,
   `melange([1, 2, 3, 4, 5, 6], [7, 8, 9])` renvoie `[1, 7, 2, 8, 3, 9, 4, 5, 6]`,

Exercice 5

On considère une liste de n entiers compris entre 1 et 99 (inclus).
Par exemple la liste de $n = 20$ entiers :`[16,2,85,27,9,45,98,73,12,26,46,25,26,49,18,99,10,86,7,42]` qui a été obtenue par la commande : `liste_20=[randint(1,99) for i in range(20)]`.
On cherche différentes manières de trouver des nombres de la liste dont la somme fait exactement 100.

1. Écrire une fonction `somme_deux_100(liste)` qui teste s'il existe deux éléments de la liste, situés à des positions différentes, dont la somme vaut 100. Pour l'exemple donné la fonction renvoie `True` et peut afficher les entiers 2 et 98 (aux rangs 1 et 6 de la liste).

2. Écrire une fonction `somme_suite_100(liste)` qui teste s'il existe des éléments consécutifs de la liste dont la somme vaut 100. Pour l'exemple donné la fonction renvoie `True` et peut afficher les entiers à suivre 25, 26, 49 (aux rangs 11, 12 et 13).

Exercice 6

Un carré magique est un tableau carré de taille $n \times n$ qui contient tous les entiers de 1 à n^2 et qui vérifie : la somme de chaque ligne, la somme de chaque colonne, la somme de la diagonale principale et la somme de l'anti-diagonale ont toutes la même valeur.

Pour un carré magique de taille $n \times n$, la valeur de la somme est :

$$S_n = \frac{n(n^2+1)}{2}.$$

Voici un exemple de carré magique de taille 3×3 et un de taille 4×4.

```
carre_3x3 = [[4,9,2], [3,5,7], [8,1,6]]
carre_4x4 = [[1,14,15,4], [7,9,6,12], [10,8,11,5], [16,3,2,13]]
```

1. Définir une fonction `est_carre_magique(carre)` qui teste si un tableau donné est (ou pas) un carré magique.

2. Générer de façon aléatoire des carrés contenant les entiers de 1 à n^2 grâce à une fonction `carre_aleatoire(n)`. Vérifier expérimentalement qu'il est rare d'obtenir ainsi un carré magique !

   Aide: Pour une liste `maliste`, la commande `shuffle(maliste)` (issue du module `random`) mélange aléatoirement la liste.

Exercice 7

Le but de cet exercice est d'approcher le nombre π à l'aide d'un processus aléatoire. On considère un repère orthonormé dans le plan, ainsi qu'un disque $\mathscr{D}$ de rayon 2 centré sur l'origine du repère et $\mathscr{C}$ le carré plein $[-2,2] \times [-2,2]$.

1. Sur feuille, dessiner $\mathscr{C}$ et $\mathscr{D}$ et calculer leurs aires.

2. On choisit un point aléatoire dans $\mathscr{C}$. Sur feuille, déterminer la probabilité p_0 que ce point appartienne à $\mathscr{D}$. On attend une valeur exacte.

   Afin de calculer une approximation de π, on approche p_0 de la manière suivante : on choisit aléatoirement n points dans $\mathscr{C}$ et on approche p_0 par m/n où m est le nombre de points appartenant à $\mathscr{D}$. Pour information, cette méthode pour estimer la valeur de π est très lente. Il existe des méthodes beaucoup plus rapides.

3. À l'aide de la fonction `uniform` du module `random`, écrire une fonction `point_aleatoire` qui renvoie un couple `(x, y)` représentant l'abscisse et l'ordonnée d'un point choisi aléatoirement et uniformément dans $\mathscr{C}$.

4. Écrire une fonction `est_dans_D` qui prend en entrée un point `p`, et renvoie `True` si `p` appartient à $\mathscr{D}$ ou `False` sinon. Le point `p` sera un tuple `(x, y)` représentant l'abscisse et l'ordonnée du point. Votre fonction ne doit prendre qu'un argument `p` qui est un couple, pas deux arguments de type `float`.

5. Implémenter une fonction `approx_pi` qui prend en entrée un entier `n` et renvoie une approximation de π en utilisant la méthode décrite ci-dessus avec `n` points aléatoires.

6. Donner une fonction `diff_approx_pi` qui prend en entrée un entier `n` et renvoie la valeur absolue de la différence entre `approx_pi(n)` et la constante `pi` du module `math`. Vérifiez que `diff_approx_pi(n)` s'approche de 0 quand `n` est grand.

Exercice 8

Dans cet exercice, il ne faut pas confondre les notions d'"entier" et de "chiffre". Un entier est un élément de $\mathbb{N}$, un chiffre est un élément de $\{0,1,2,3,4,5,6,7,8,9\}$. Soit $n \in \mathbb{N}$ un entier naturel. On s'intéresse à l'opération consistant à multiplier les chiffres de n entre eux. Par exemple, lorsqu'on multiplie les chiffres de 15638 , on obtient $1 \times 5 \times 6 \times 3 \times 8 = 720$ et lorsqu'on multiplie les chiffres de 720 on obtient $7 \times 2 \times 0 = 0$.

1. À l'aide d'une boucle `while`, écrire une fonction de signature `prod_chiffres(n: int)-> int` qui renvoie le produit des chiffres de n. Par exemple :

`n`	0	1	6	473	678	720	15638
`prod_chiffres(n)`	0	1	6	84	336	0	720

   La persistance multiplicative d'un entier est le nombre d'itérations nécessaires pour obtenir un résultat avec un seul chiffre. Par exemple, la persistance multiplicative de 15638 est 2.

2. (a) Quelle est la persistance multiplicative de 678 ?

(b) À l'aide d'une boucle `while`, écrire une fonction de signature `pers_mult(n: int) -> int` qui renvoie la persistance multiplicative de `n`. Par exemple :

`n`	0	1	6	473	678	720	15638
`pers_mult(n)`	0	0	0	3	4	1	2

Pour tout $k \in \mathbb{N}$, on note N_k le plus petit entier naturel dont la persistance multiplicative vaut k. Par exemple, $N_0 = 0$ et $N_1 = 10$.

3. (a) Sur feuille, déterminer N_2 et N_3.

   (b) À l'aide d'une boucle `while`, écrire une fonction de signature `get_Nk(k:int)->int` qui renvoie N_k. Tester votre fonction pour $k \in [\![0; 9]\!]$.