TP du 16 décembre 2025

1 Révisions

- Écrire une fonction `puissance(a,n)` qui prend en argument un float a et un entier naturel n et qui renvoie a^n.
- Ecrire une fonction `Conversion(n,b)` qui renvoie une liste donnant les chiffres de l'écriture de n en base b.
- Ecrire une fonction `est_croissante(L)` qui prend en argument une liste `L` d'entiers et qui renvoie un booléen indiquant si L est croissante.
- Écrire une fonction `Fibonacci(n)` qui renvoie le $n^{\text{ème}}$ nombre de Fibonacci.
- Écrire une fonction `Renverser(L)` qui prend en argument une liste `L` et qui renvoie la liste `L` lue à l'envers.
- Écrire une fonction `Palindrome(mot)` qui prend en argument une chaîne de caractères `mot` et qui renvoie un booléen indiquant si `mot` est un palindrome.

2 Diviseurs

Écrire une fonction `diviseurs(n)` qui prend en argument un entier naturel `n` et renvoie la liste de ses diviseurs positifs. Penser à la division euclidienne. Indication : les diviseurs de n sont à chercher parmi les entiers compris entre … et …

```
>>> diviseurs(42)
[1, 2, 3, 6, 7, 14, 21, 42]

>>> diviseurs(0)
[]
```

3 Même signe

Écrire une fonction `meme_signe(L)` qui prend en argument une liste `L` d'entiers et renvoie `True` si les entiers ont même signe et `False` sinon.
L'entier 0 est à la fois positif et négatif.

```
>>> L = [0,3,3]
>>> meme_signe(L):
True

>>> L = [0,-3,3]
>>> meme_signe(L):
False
```

Faites en sorte que votre fonction puisse prendre en argument la liste vide :

```
>>> L = []
>>> meme_signe(L):
True
```

4 Décomposition d'une liste d'entiers en sous-listes croissantes

- Écrire une fonction `decomposition(L)` qui prend en argument une liste d'entiers `L` et qui renvoie une liste de listes contenant les sous-listes croissantes de `L`.

```
>>> decomposition([3,9,1,9,8,7])
[[3, 9], [1, 9], [8], [7]]

>>> decomposition([])
[]
```

Indication

```
def decomposition(L):
    D = []
    sous_suite = []
    for i in range(.......):

        ......

        if .........
            D.append(......)
            sous_suite = .....
    return D
```

5 Mélange de listes

Écrire une fonction `melange(L1,L2)` qui prend en arguments deux listes `L1` et `L2` et renvoie la liste :
`L = [L1[0], L2[0], L1[1], L2[1], L1[2], L2[2], ...]`
Dans le cas où `L1` et `L2` ne sont pas de la même longueur, votre fonction ajoutera les éléments en surplus à la fin de la liste `L`.
Par exemple :

```
>>> L1 = list(range(0,8))
>>> L2 = [x for x in "pcsi"]
>>> melange(L1,L2)
[0, 'p', 1, 'c', 2, 's', 3, 'i', 4, 5, 6, 7]
```

Tester également :

```
>>> melange(L1, [])
???
>>> melange([], L2)
???
>>> melange([], [])
???
```

6 Matrice magique

Une matrice carrée de taille n est dite magique si elle contient tous les numéros de 1 à n^2 et si les sommes des nombres de chaque ligne, de chaque colonne et des deux diagonales sont toutes égales à une même constante s.

- Exprimer la constante s en fonction de n.
- Créer une fonction `est_magique(A)` d'argument une matrice A (carrée de taille n codée sous forme de liste de listes) et qui renvoie un booléen indiquant si A est magique ou pas.

  Tester cette fonction sur les matrices :

$$A = \begin{pmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{pmatrix} \quad \text{et} \quad B = \begin{pmatrix} 1 & 8 & 2 \\ 4 & 5 & 7 \\ 6 & 9 & 3 \end{pmatrix}.$$

  Construction d'une matrice magique de taille impaire (méthode de Bachet-Méziriac).

  En 1612, dans son livre Problèmes plaisants et délectables qui se font par les nombres, Claude-Gaspar Bachet de Méziriac propose une méthode de construction de matrice magique de taille impaire.

  Nous allons exposer sa méthode.

  Pour cela, on aura besoin de considérer une matrice carrée de taille n comme une matrice infinie « n-périodique en ligne et en colonne ».

  — On construit une matrice de taille n (par exemple remplie de 0).

  — On place les nombres de 1 à n^2 comme suit :

  ⋆ on met le 1 dans la case située juste en dessous de la case centrale de la matrice (cette case centrale existe car n est impair)

  ⋆ on place ensuite chaque nombre de 2 à n^2 dans la case située à la ligne suivante et colonne suivante de la case où on a mis le nombre précédent. Si cette case est déjà remplie, on avance encore d'une ligne et on recule d'une colonne (on admet ici que la nouvelle case n'est pas déjà prise).

  On admet que la matrice ainsi construite est magique.
- Construire à la main selon cette méthode une matrice magique de taille 3, puis de taille 5.
- Écrire une fonction `MatriceMagiqueTailleImpaire(n)` d'argument un entier n impair qui renvoie la matrice magique de taille n créée à l'aide de la méthode précédente.

Aller sur le site canal math, construction des carrés magiques, méthode de Bachet de Méziriac
Et sur ce site pour une tentative de preuve automath site, carré magique de bachet

7 Permutation en spirale

- Écrire une fonction `spirale(L)` d'argument une liste `L` et qui renvoie la liste obtenue en prenant d'abord le dernier terme de L puis le premier, puis l'avant-dernier, puis le deuxième etc.
  Quelques tests :

```
>>> L = [x for x in 'pcsi3BJ']
>>> L
['p', 'c', 's', 'i', '3', 'B', 'J']
>>> spirale(L)
['J', 'p', 'B', 'c', '3', 's', 'i']
```

```
>>> L = list(range(1, 7))       %% L = [k for k in range(1,7)]
[1, 2, 3, 4, 5, 6]

>>> spirale(L)
[6, 1, 5, 2, 4, 3]
```

Pour la culture, voici un code utilisant le type « deque » (signifiant *double ended queue*).

```
from collections import deque

def spirale_deque(L):
    a = deque(L)
    droite = True
    res = []
    while len(a) > 0:
        if droite:
            res.append(a.pop())
        else:
            res.append(a.popleft())
        droite = not droite
    return res
```

- Vérifier que si l'on itère la fonction `spirale` sur $L = [1,2,3,4,5,6]$, on retombe sur L à la sixième itération. Qu'en est-il pour $L = [1,2,3,4]$?
- Écrire une fonction `periode(n)` d'argument un entier n qui renvoie le nombre minimum p d'itérations de la fonction `spirale` pour retomber sur la liste $[1,2,\dots,n]$ en partant de cette même liste.

Pour la culture mathématique, on est en train de calculer l'**ordre** de la permutation « spirale », permutation qui réalise $1 \mapsto n$, $2 \mapsto 1$, $3 \mapsto n-1$, $4 \mapsto 3$ etc.

```
>>> periode(1987)
260

>>> periode(39)
39
```

- Écrire une fonction `nombres_Queneau(n)` qui renvoie une liste contenant tous les nombres $k \in [\![1,n]\!]$ tels que `periode(k)=k`.

```
>>> nombres_Queneau(99)
??
```

- Taper :

```
>>> [2*q + 1 for q in nombres_Queneau(99)]
??
```

Que remarquez-vous ?

- Aller sur le site https://oeis.org (le connaissiez-vous ?) et rentrer la liste obtenue à la question précédente.
- Charger le module `matplotlib.pyplot` pour tracer la suite $n \mapsto \dfrac{\text{periode}(n)}{n}$, disons pour $n \in [\![1,99]\!]$.
- Déterminer la valeur de n qui minimise $\dfrac{\text{periode}(n)}{n}$.