EXERCICES D'INFORMATIQUE

---

1. **Exercice : termes d'une suite réelle**

   On considère la suite réelle $u = (u_n)_{n\geq 0}$ définie par son premier $u_0 = 2$ et la relation de récurrence :

   $$\forall n \in \mathbb{N} \quad u_{n+1} = u_n^2 - 5$$

   a) Écrire une fonction `TermeSuite(n)` en utilisant une boucle `for` qui renvoie la valeur de u_n.

   b) Écrireune **fonction récursive** `TermeSuiteRec(n)` qui renvoie la valeur de u_n.

---

2. **Exercice : enregistrer des triplets dans une liste**

   Vous achetez pour S euros de gâteaux chez le pâtissier et vous avez dans votre portefeuille x billets de 5 euros, y pièces de 2 euros et z pièces de 1 euro.

   Écrire une fonction `Monnaie(S,x,y,z)` qui retourne la liste des triplets (a, b, c) où a désigne le nombre billets de 5 euros, b le nombre de pièces de 2 euros et c le nombre pièces de 1 euro avec lesquels vous payez la somme de S euros.
   Par exemple, `Monnaie(8,2,2,6)` renvoie la liste [(0, 1, 6), (0, 2, 4), (1, 0, 3), (1, 1, 1)].

---

3. **Exercice : boucle while**

   Écrire une fonction qui détermine si un entier n donné est une puissance de 7.

---

4. **Exercice : chaîne de caractères**

   Écrire une fonction qui, étant donné une chaîne de caractères (ne comprenant par de caractères accentués), revoie le nombre de voyelles que contient cette chaîne.

---

5. **Exercice : chaîne de caractères**

   Écrire une fonction qui, étant donné une chaîne de caractères `s` (ne comprenant par de caractères accentués), construit une chaîne de caractères dans laquelle on rajoute la chaîne `'av'` après chaque consonne suivie d'une voyelle. Par exemple, si la chaîne de caractères donnée est `'abricot'`, la fonction doit renvoyer `abravicavot'`.

---

6. **Exercice : parcours de listes**

   Écrire une fonction `MinStrictPos(t)` qui prend en argument une liste `t` de nombres et qui renvoie le plus petit nombre strictement positif de la liste s'il existe et `None` dans le cas contraire.

---

7. **Exercice : parcours de listes**

   Écrire une fonction `Distincts(t)` qui prend en argument une liste `t` de nombres et qui renvoie `True` si les nombres de la liste `t` sont deux à deux distincts et `False` sinon.
   Quelle est la complexité de votre fonction dans le pire des cas.

---

8. **Exercice : à l'endroit et à l'envers ...**

   A partir d'un entier n dont l'écriture décimale comporte au plus 4 chiffres, on effectue les manipulations suivantes : on construit les entiers n_- et n_+ possédant les mêmes chiffres, mais respectivement rangés dans l'ordre croissant et dans l'ordre décroissant. Ensuite, on calcule $n' = n_+ - n_-$. Si $n' = n$, on s'arrête, et sinon, on recommence avec n'.

   Exemple : si $n = 8709$, alors $n_- = 789$ et $n_+ = 9870$, d'où $n' = 9870 - 789 = 9081$.

   Comme $9081 \neq 8709$, on recommence ....

   On obtient ainsi une séquence commençant par : 8709, 9081, 9621, ...

   (a) Écrire une fonction `decompose(n)` qui renvoie le quadruplet (a,b,c,d) des chiffres décimaux de `n`. Par exemple, `decompose(4701)` renvoie le quadruplet `(4,7,0,1)`

(b) Écrire une fonction `recompose(a,b,c,d)` qui renvoie l'entier `n` dont les chiffres décimaux de `n` sont `a`, `b`, `c` et `d`. Par exemple, `recompose(4,7,0,1)` renvoie l'entier `4701`.

(c) Écrire une fonction `tri` qui, étant donné un quadruplet (a,b,c,d) renvoie le quadruplet (a',b',c',d') rangé par ordre croissant. Par exemple, `tri(4,7,0,1)` renvoie le quadruplet `(0,1,4,7)`

(d) Écrire une fonction `nPlus(n)` qui renvoie la valeur n_+. Par exemple, `nPlus(4701)` renvoie `7410`.

(e) Écrire une fonction `nMoins(n)` qui renvoie la valeur n_-. Par exemple, `nMoins(4701)` renvoie `147`.

(f) Écrire une fonction `suivant(n)` qui renvoie la valeur n_-n_-. Par exemple, `suivant(4701)` renvoie `7263`.

(g) Écrire une fonction `iterPlusMoins(n)` qui en partant de l'entier `n` affiche les entiers successifs obtenus en répétant la règle donné en début d'énoncé, à raison d'un entier par ligne.
Que peut-on conjecturer ?

---

9. **Exercice : création et visualisation de grilles**

On souhaite ici créer des *grilles* `G` de taille $n \times n$ (n lignes et n colonnes) dont les éléments sont des entiers naturels.

On procède très souvent en deux étapes :

• on crée un tableau `G` de taille $n \times n$ (où n est choisi par l'utilisateur) dont les éléments sont tous nuls :

```
n = 10
G = [ [0 for i in range(n)] for j in range(n)]
```

G est une liste de n listes.

L'accès à un terme de la grille `G` se fait à l'aide de l'opération d'indexage `G[i][j]` où `i` désigne la ligne et `j` la colonne.
Attention, les indices commencent à zéro !

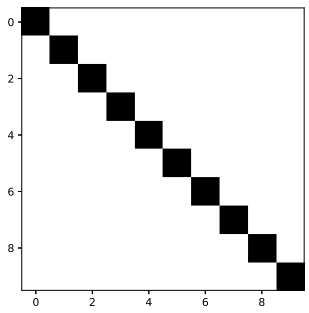
• on modifie certains termes de `G` à l'aide d'une boucle ou de boucles imbriquées.
Sur le premier exemple, on souhaite que les coefficients diagonaux `G[i][i]` soient égaux à 1 :

```
for i in range(n):
    G[i][i] = 1
```

On peut ensuite utiliser la fonction `imshow` pour visualiser la grille en associant à chaque valeur entière une couleur :

```
import matplotlib.pyplot as plt
import matplotlib as mpl

CM = mpl.colors.ListedColormap(['white', 'black'])
plt.imshow(G, cmap = CM)
plt.show()
```



L'indice de ligne i se lit sur l'axe des ordonnées, et l'indice de colonne j sur l'axe des abscisses.
La case de coordonnées $(0,0)$ se trouve donc en haut à gauche, celle qui se situe juste en dessous a pour coordonnées $(1,0)$.

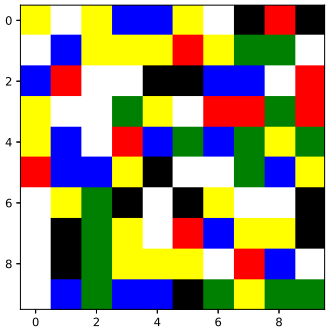
Exemple 2 : coloration aléatoire d'une grille (6 couleurs)

```
import random as rd

n = 10
G = [ [0 for i in range(n)] for j in range(n)]
```

```
for i in range(n):
    for j in range(n):
        G[i][j] = rd.randint(0,5)

CM = mpl.colors.ListedColormap(['white', 'black', 'green', 'red', 'yellow', 'blue'])
plt.imshow(G, cmap = CM)
plt.show()
```

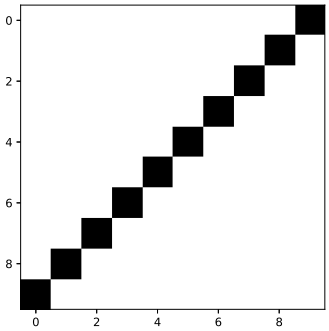


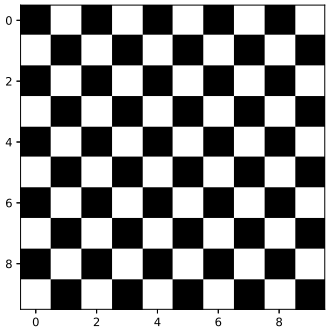
On peut ensuite créer des fonctions utiles pour analyser les éléments d'une grille `G` :

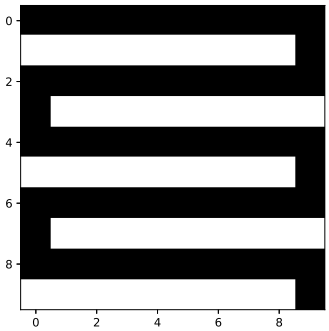
```
def NbCases(G,c):
    # compte le nombre de cases de couleur c
    N = 0
    n = len(G)
    for i in range(n):
        for j in range(n):
            if G[i][j] == c:
                N = N+1
    return N
```

`NbCases(G,0)` renvoie 23 (il y a 23 cases blanches), `NbCases(G,1)` renvoie 12 (il y a 12 cases noires) ...

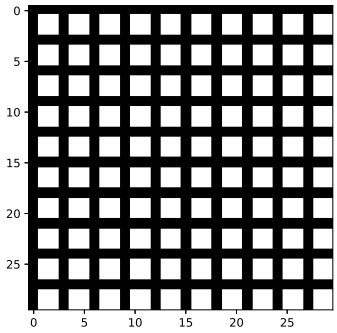
Exercices : créer les grilles suivantes



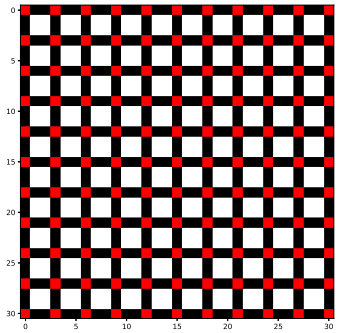




Pour les grilles suivantes, on pourra distinguer les cas suivants : i pair, le reste de i par 4 vaut 1, le reste de i par 4 vaut 3

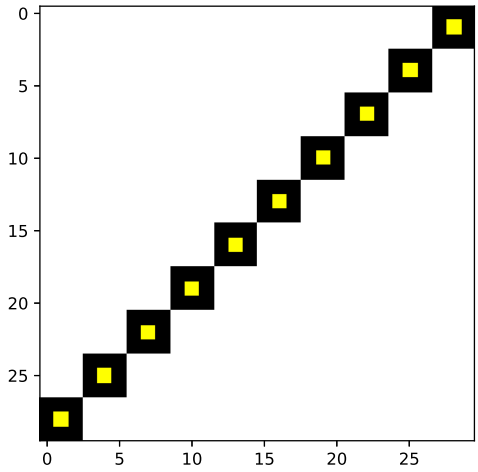


Une ligne et une colonne sur trois sont noires



même chose avec des intersections coloriées en rouge

Dernière question *(plus difficile)* : ici le nombre n de lignes et de colonnes est un multiple de 3
Les carrés sont des blocs 3×3 avec une cellule au centre coloriée en jaune.



---

10. **Exercice : automate cellulaire unidimensionnel**

On propose de programmer la règle 126 : lire le document **AutomateCellulaire.pdf** (*déposé sur cahier prepa*).

Écrire une fonction `transition(t)` qui prend en argument une liste `t` et qui renvoie la liste obtenue en appliquant la règle 126 aux cases de `t` qui ne sont pas des bords.

Par exemple, `transition([0,1,1,1,0,1,0])` renverra la liste `[0, 1, 0, 1, 1, 1, 0]`.

Représentation de l'évolution de l'automate :

L'idée est d'empiler les différentes listes obtenues :

`t = [0]*500 + [1] + [0]*500` *liste de départ : une seule case noire au milieu de la liste*

```
G = [t]
for k in range(500):
    t = transition(t)
    G.append( t )
```

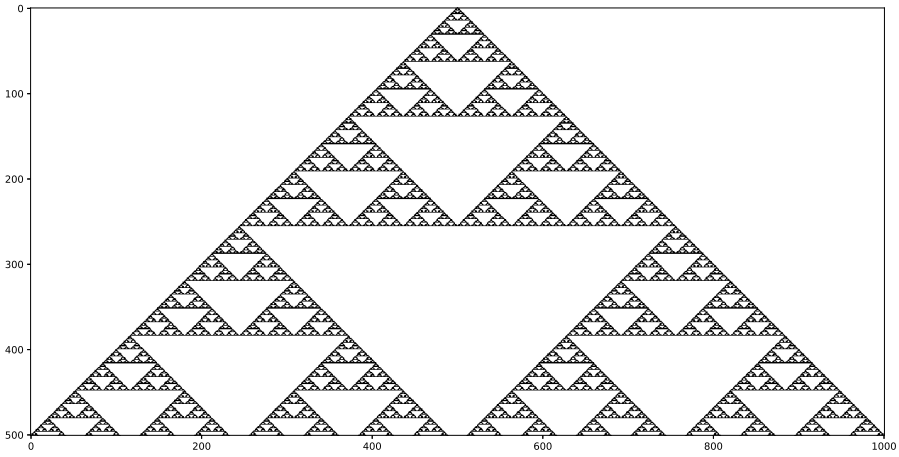
Ainsi défini, `G` est un tableau de 501 lignes et 1001 colonnes qui permet de représenter l'évolution de la liste `t` donnée au départ.

```
import matplotlib.pyplot as plt

CM = mpl.colors.ListedColormap(['white', 'black'])

plt.imshow(G, cmap = CM)
```

```
plt.show()
```



Autre configuration de départ :

`t = [ (k % 2) for k in range(101)]` *alternance de 0 et de 1*