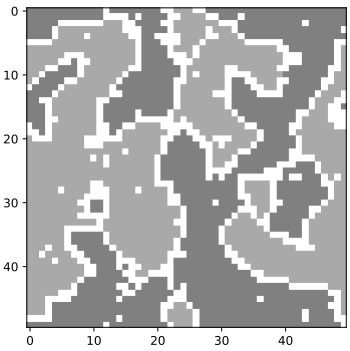
TP 15 d'informatique - MPSI - 10/03/2026



Exercice 1. Ségrégation entre MPSI

Dans cet exercice on étudie le modèle introduit en 1969 par Thomas Schelling pour modéliser la ségrégation urbaine. On s'intéresse à la cantine du lycée Poincaré où mangent `m` élèves, chaque élève est soit en MPSI 1 soit en MPSI 2. La cantine est représentée par une grille composée de `n` lignes et `n` colonnes dont chaque case peut être vide ou bien occupée par un élève.

Un élève est heureux lorsqu'il n'a pas de voisin, ou bien lorsque la proportion de ses voisins de la même classe que lui est strictement supérieure à 2/3 (sans prendre en compte les cases adjacentes inoccupées). Les voisins d'un élève sont ceux qui occupent l'une des 8 cases adjacentes, sachant que les bords droit/gauche et haut/bas sont reliés. Par exemple, la case de coordonnées `(0, 1)` est voisine des cases `(0, 2)`, `(1, 2)`, `(1, 1)`, `(1, 0)`, `(0, 0)`, `(n − 1, 0)`, `(n − 1, 1)` et `(n − 1, 2)`.

Initialement, chaque élève a 50% de chances d'être en MPSI 1 et 50% de chances d'être en MPSI 2, et se place au hasard dans la cantine. À chaque étape, l'un des élèves malheureux se déplace sur une case inoccupée (l'élève et la case sont choisis aléatoirement – la nouvelle case n'est pas nécessairement adjacente à la case initiale). Le processus s'arrête lorsque tout le monde est heureux ou bien lorsque le nombre maximum d'étapes noté `N` est atteint.

1. Écrire une fonction qui prend en entrée `n` (la dimension de la cantine), `m` (le nombre d'élèves), `N` (le nombre maximal d'étapes) et affiche la situation finale. Pour l'affichage, on pourra s'inspirer du programme donné ci-dessous. Testez avec :

   `(n, m, N) = (10, 50, 1000)` puis `(n, m, N) = (50, 2000, 20000)`.

```
# 'C' est une liste de listes.
# C[i][j] = 0, 1 ou 2 (case blanche, bleue ou rouge)
import matplotlib.pyplot as plt
import matplotlib.colors as col
plt.figure() # Crée une nouvelle figure
plt.imshow(C, interpolation = 'Nearest',
           cmap = col.ListedColormap(['white', 'blue', 'red']))
plt.show()
```

2. Faire en sorte de voir l'évolution de la population au fur et à mesure en s'inspirant du programme :

```
# Sous Spyder: lancez d'abord la commande "%matplotlib qt5" dans la console
import random
def alea():
    fig = plt.figure() # Crée une nouvelle figure
    while True:
        C = [[random.randint(0,2) for _ in range(20)] for _ in range(20)]
        plt.pause(0.01) # Pause de 0.01 seconde
        if not plt.fignum_exists(fig.number): # Si la fenêtre a été fermée
            return
        plt.clf() # Efface la figure
        plt.imshow(C, interpolation = 'Nearest',
                   cmap = col.ListedColormap(['white', 'blue', 'red']))
    plt.show()
alea()
```

Exercice 2.

Reprendre le sujet X/ENS 2022 (voir le lien sur la page du cours).

Exercices à rendre au plus tard le 15/03/2026 à 20h

Exercice 3. Automates cellulaires

Un automate cellulaire est un ensemble de cellules qui évoluent au cours du temps. À chaque instant, une cellule est blanche ou noire et sa couleur à l'instant $t+1$ dépend des couleurs qui lui sont adjacentes à l'instant t.

Automates cellulaires en dimension 1 - Règle 129. Soit `n` $\in \mathbb{N}$. On appelle ***configuration*** une ligne composée de `n` cellules. En Python, une configuration sera représentée par une liste de taille `n` contenant des `0` (pour les cellules blanches) et des `1` (pour les cellules noires). Par exemple avec `n = 13`, la liste `[0,1,0,0,0,1,1,0,1,0,1,1,1]` correspond à :

Soit `T` $\in \mathbb{N}$. Le système évolue du temps `t = 0` au temps `t = T`. À chaque instant, chaque cellule est mise à jour en fonction de la couleur de ses deux voisines et de sa propre couleur. Plus précisément, dans cette partie, on s'intéresse à "l'automate élémentaire 129" dont les règles d'évolution sont :

→ Une cellule c est blanche à l'instant `t+1` si on se trouve dans l'un des deux cas suivants :

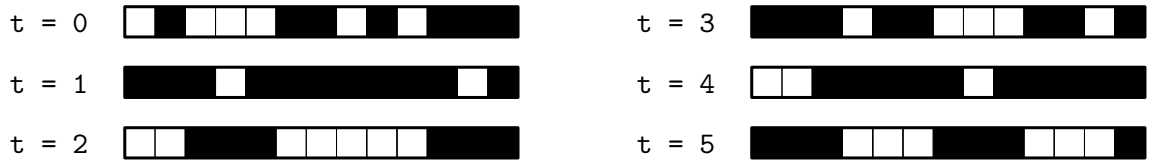
- c et ses deux voisines sont blanches à l'instant `t`.
- c et ses deux voisines sont noires à l'instant `t`.

→ Une cellule c sera noire à l'instant `t+1` dans tous les autres cas.

Afin que toutes les cellules aient deux voisines, on considère que :

$$\begin{cases} \text{La voisine gauche de la cellule d'indice } \texttt{0} \text{ est la cellule d'indice } \texttt{n-1}. \\ \text{La voisine droite de la cellule d'indice } \texttt{n-1} \text{ est la cellule d'indice } \texttt{0}. \end{cases}$$

Voici un exemple d'évolution :



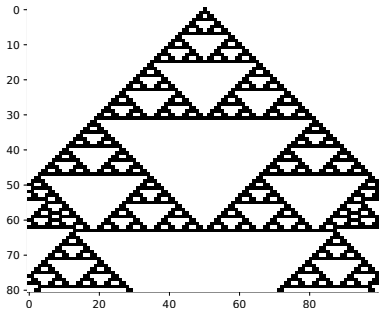
1. Écrire une fonction « `etape(C: list[int]) -> list[int]` » qui prend en entrée une configuration à un instant `t`, et renvoie la configuration à l'instant `t+1`.

Le ***diagramme espace-temps*** d'un automate cellulaire est une liste « `D: list[list[int]]` » contenant `T+1` sous-listes de taille `n`. Pour tout `t` $\in [\![\texttt{0}, \texttt{T}]\!]$ et tout `i` $\in [\![\texttt{0}, \texttt{n} - 1]\!]$, la case `D[t][i]` a la même couleur que la cellule numéro `i` au temps `t`. En Python, pour afficher un digramme espace-temps, on s'inspirera du programme :

```
import matplotlib.pyplot as plt
import matplotlib.colors as col
D = [[0,1,0,1], [1,1,1,0], [0,0,1,0]]
plt.figure() # Crée une nouvelle figure
plt.imshow(D, interpolation='Nearest',
           cmap = col.ListedColormap(['white','black']))
plt.show()
```

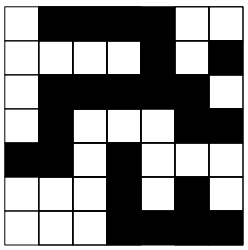
2. Écrire une fonction « `diag(C0: list[int], T: int) -> NoneType` » qui prend en entrée la configuration initiale `C0` ainsi que `T`, et affiche le diagramme espace-temps associé. Par exemple :

```
n = 100; T = 80
C0 = [0]*n
C0[n//2] = 1
diag(C0, T)
```



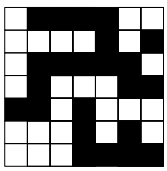
Automates cellulaires en dimension 2 – Règle de Fredkin. Soit $\texttt{n} \in \mathbb{N}$. Un automate cellulaire en dimension 2 contient $\texttt{n}^2$ cellules réparties sur `n` lignes et `n` colonnes. En Python, une configuration sera représentée par une liste « `C: list[list[int]]` » contenant des `0` et des `1`. Par exemple :

```
C = [[0,1,1,1,1,0,0],
     [0,0,0,0,1,0,1],
     [0,1,1,1,1,1,0],
     [0,1,0,0,0,1,1],
     [1,1,0,1,0,0,0],
     [0,0,0,1,0,1,0],
     [0,0,0,1,1,1,1]]
```

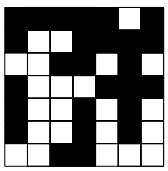


Pour que chaque cellule ait huit voisines, on considère que les bords droit/gauche et haut/bas sont reliés. Par exemple, la case de coordonnées `(0, 1)` est voisine des cases `(0, 2)`, `(1, 2)`, `(1, 1)`, `(1, 0)`, `(0, 0)`, `(n − 1, 0)`, `(n − 1, 1)` et `(n − 1, 2)`.

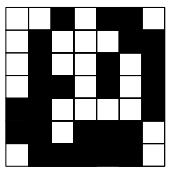
Dans cette partie, on s'intéresse à ***l'automate de Fredkin*** dans lequel l'état d'une cellule à l'instant `t + 1` est la somme modulo 2 des états des huit cellules adjacentes à l'instant `t`, par exemple :



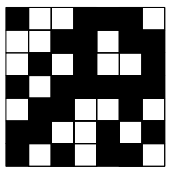
`t = 0`



`t = 1`

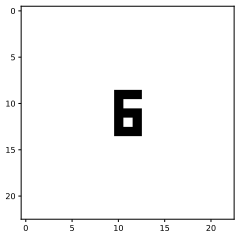


`t = 2`

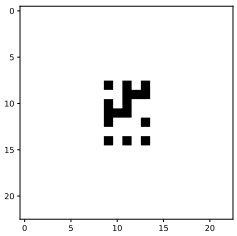


`t = 3`

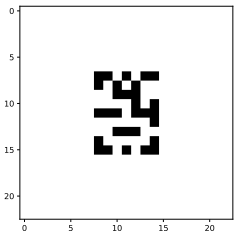
Autre exemple :



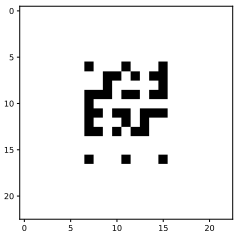
`t = 0`



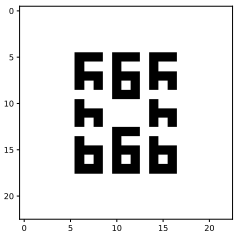
`t = 1`



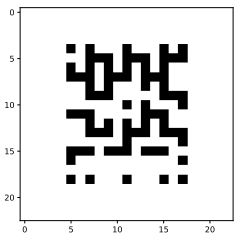
`t = 2`



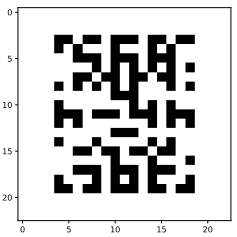
`t = 3`



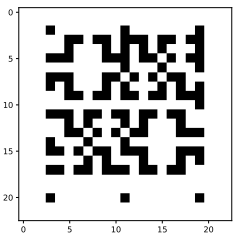
`t = 4`



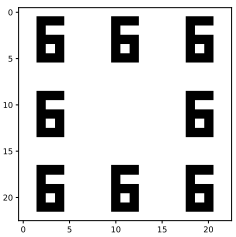
`t = 5`



`t = 6`



`t = 7`



`t = 8`

3. Écrire une fonction

```
Fredkin(C0: list[list[int]], T: int) -> list[list[int]]
```

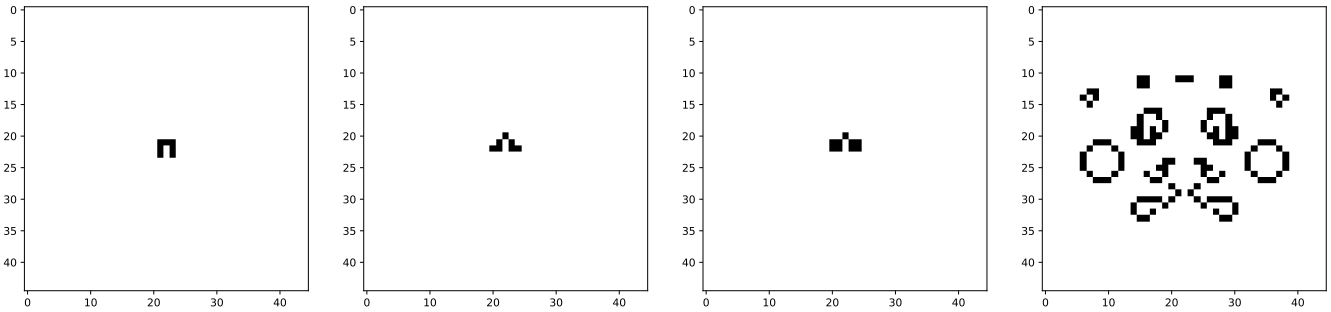
qui prend en entrée la configuration `C0` à l'instant `0` et renvoie la configuration à l'instant `T` en suivant la règle de Fredkin. Votre fonction doit renvoyer une liste et ne doit rien afficher (en particulier, elle ne doit pas faire appel au module `matplotlib`).

Le jeu de la vie (partie facultative). Le ***jeu de la vie*** de John Conway est le plus connu des automates cellulaires. Dans ce contexte, on dit qu'une cellule blanche est ***morte*** et qu'une cellule noire est ***vivante***. Une cellule c est vivante à l'instant `t+1` si et seulement si l'une des deux conditions suivantes est vérifiée :

→ c est morte à l'instant `t` et exactement trois voisines de c sont vivantes à l'instant `t`.

→ c est vivante à l'instant `t` et exactement deux ou trois voisines de c sont vivantes à l'instant `t`.

Par exemple (la configuration pour `t = 110` s'appelle "le clown") :



`t = 0` `t = 1` `t = 2` `t = 110`

4. Écrire une fonction

```
JDV(C0: list[list[int]], T: int) -> NoneType
```

qui prend en entrée la configuration `C0` à l'instant `0` et affiche à l'aide de `matplotlib` le jeu de la vie pour `t` $\in [\![0 ; T]\!]$. On fera en sorte de voir l'évolution au fur et à mesure en s'inspirant du programme :

```
# Sous Spyder: lancez d'abord la commande "%matplotlib qt5" dans la console
import random
def alea():
    fig = plt.figure() # Crée une nouvelle figure
    while True:
        C = [[random.randint(0,1) for _ in range(20)] for _ in range(20)]
        plt.pause(0.01) # Pause de 0.01 seconde
        if not plt.fignum_exists(fig.number): # Si la fenêtre a été fermée
            return
        plt.clf() # Efface la figure
        plt.imshow(C, interpolation = 'Nearest',
                   cmap = col.ListedColormap(['white', 'black']))
    plt.show()
alea()
```