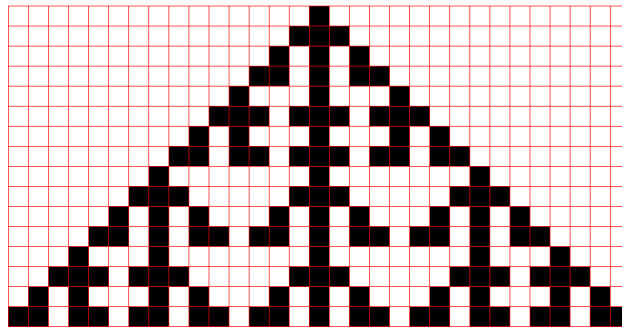


Automates

Tu vas programmer des automates cellulaires qui, à partir de règles simples, produisent des visualisations amusantes.



Activité 1 (Une suite logique).

Objectifs : programmer une suite logique amusante (mais pas nécessaire pour l'activité suivante).

Voici une suite :

```
1
11
21
1211
111221
312211
13112221
1113213211
```

Pour passer d'une ligne à la suivante, il suffit de lire à haute voix en comptant les nombres ! Par exemple la ligne 1211 est lue « un un (pour 1), un deux (pour 2), deux un (pour 1 1) », la ligne d'après est donc 111221 ! Cette dernière ligne se lit « trois uns, deux deux, un un » donc la ligne suivante sera 312211. Programme une fonction `lecture(mot)` qui calcule et renvoie la lecture de la chaîne `mot`. Par exemple `lecture("1211")` renvoie "111221".

- Essaie de programmer cette fonction sans lire les indications suivantes !
- *Indications.* Tu peux utiliser trois variables : une variable qui lit chaque caractère du mot, une variable correspondant au caractère précédent, un compteur à incrémenter si ces deux caractères sont égaux.
- *Algorithme.* Si tu n'y arrives pas tout seul, voici les grandes lignes d'un algorithme possible.

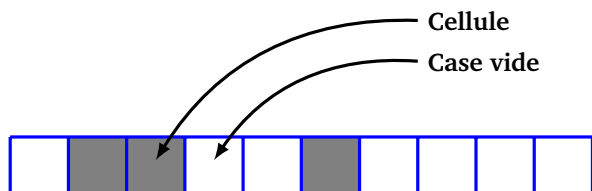
Pour chaque caractère du mot :

- si le caractère est le même que le caractère précédent, incrémenter le compteur,
- sinon, rajouter au mot à créer la valeur du compteur suivie du caractère précédent.

À la fin, il faut aussi rajouter au mot à créer la valeur du compteur suivie du caractère précédent.
Question. Trouve le premier mot qui contient 33.

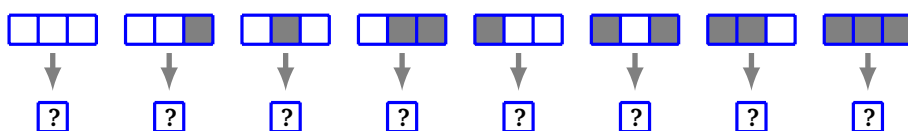
Cours 1 (Automates linéaires).

On travaille sur des lignes superposées, formées de cases. Chaque case peut contenir une cellule (la case est alors noire/contient 1) ou être vide (la case est blanche/contient 0).



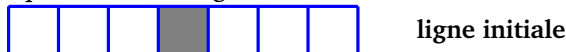
Un **automate linéaire** est une règle qui à partir du contenu de trois cases consécutives sur une ligne, détermine le contenu de la case sur la ligne du dessous.

La **règle** est donc donnée par la liste des 8 configurations possibles au départ, avec pour chacune la naissance ou pas d'une cellule en-dessous.

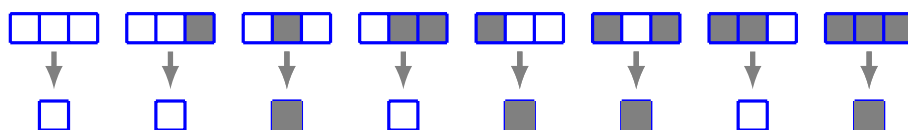


Exemple.

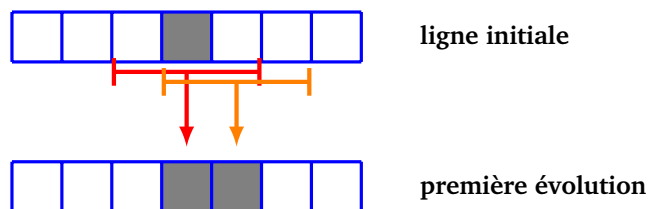
- Partons d'une seule cellule qui sera sur la ligne du haut.



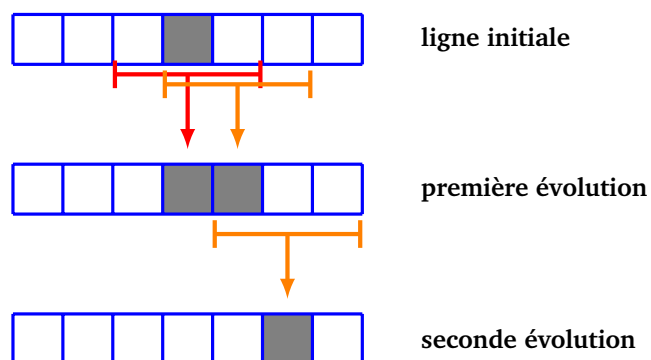
- Et choisissons la règle définie par les configurations :



- Pour décider de la naissance d'une cellule dans une case de la ligne en-dessous, on regarde les trois cases au-dessus et on applique la règle. Sur le dessin ci-dessous deux cellules sont vivantes après l'évolution (on a indiqué par des flèches seulement les règles pour lesquelles une cellule apparaît).

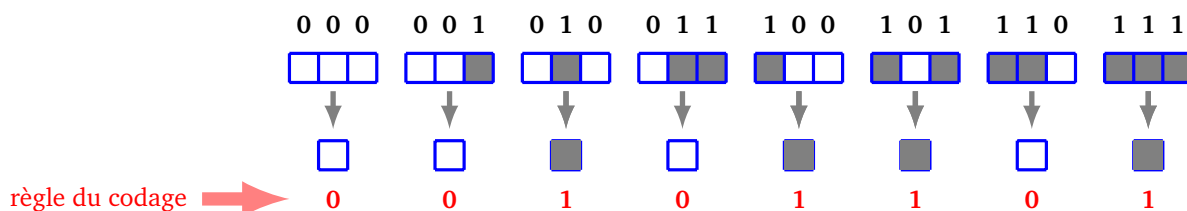


- On peut itérer le processus. Une seule cellule apparaît lors de cette seconde évolution.



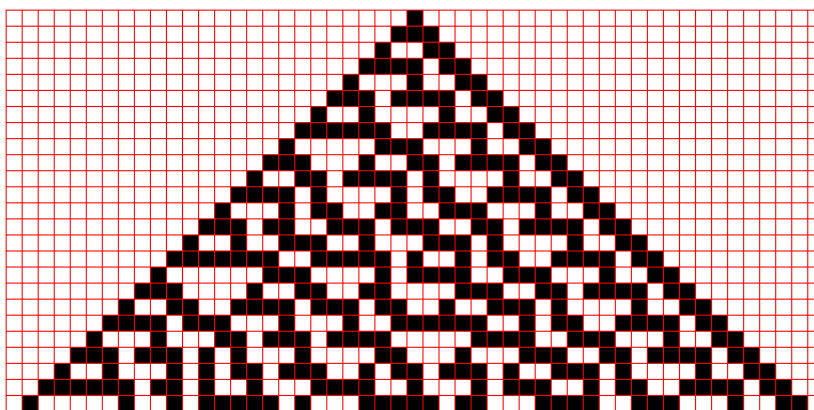
Notations.

- On note 0 pour une case vide et 1 pour une case contenant une cellule vivante.
- Une ligne est représentée par une liste de 0 et de 1. Par exemple $[0, 0, 0, 1, 0, 1, 0, 1, 0, 0]$ est une ligne de 10 cases, contenant 3 cellules.
- La règle est codée par une liste de 0 et 1 et de longueur 8, déterminée par l'image des 8 configurations possibles. Par exemple : $[0, 0, 1, 0, 1, 1, 0, 1]$ correspond à la règle :

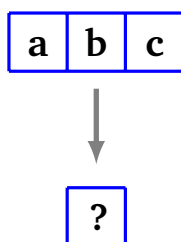


Activité 2 (Automates linéaires).

Objectifs : calculer et afficher les automates linéaires.



1. **Évolution d'une cellule.** Programme une fonction `cellule_suivante(a,b,c,regle)` qui calcule et renvoie la couleur (0 ou 1) de la case située sous les trois cases contenant a, b, c selon la règle donnée.



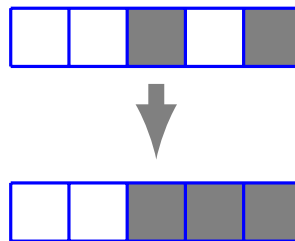
- a, b, c sont les couleurs (0 ou 1) des cases situées au-dessus (a est la case la plus à gauche).

- La loi de transformation est donnée par la liste `regle` formée d'une suite de 8 entiers 0 ou 1.
 - Exemple avec `regle = [0,0,1,0,1,1,0,1]` alors `cellule_suivante(0,0,0,regle)` renvoie 0, `cellule_suivante(0,0,1,regle)` renvoie aussi 0, `cellule_suivante(0,1,0,regle)` renvoie 1, etc.
 - Si tu ne veux pas écrire les 8 cas possibles, calcule $4a + 2b + c$!
2. **Affichage de la règle.** Déduis-en une fonction `affiche_regle(regle)` qui affiche à l'écran la règle donnée sous la forme « $a, b, c \rightarrow d$ » où d est la couleur de la nouvelle case, par exemple :

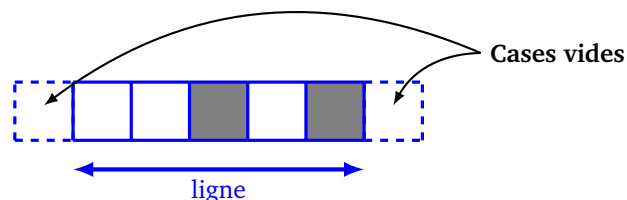
```
0 0 0 -> 0
0 0 1 -> 0
0 1 0 -> 1
...
```

3. **Évolution d'une ligne.** Programme une fonction `ligne_suivante(ligne, regle)` qui à partir d'une liste `ligne` formée de 0 et 1, calcule les cellules de la ligne suivante (renvoyée sous la forme d'une liste de 0 et de 1).

Exemple. Avec la règle `[0, 0, 1, 0, 1, 1, 0, 1]` alors la ligne suivant `[0, 0, 1, 0, 1]` est la ligne `[0, 0, 1, 1, 1]`.



Remarque. Lors du calcul des cases situées à l'une des deux extrémités de la ligne, on considère qu'au delà, il n'y a pas de cellule (c'est donc comme si à droite et à gauche de la ligne initiale il y avait un 0).

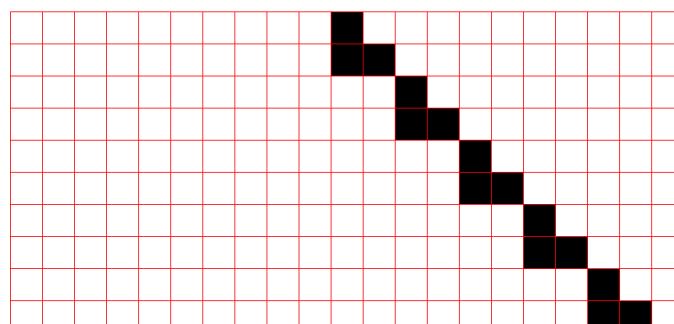


4. **Itérations.** Déduis-en une fonction `plusieurs_lignes(n, ligne, regle)` qui affiche sur le terminal les n lignes qui suivent la ligne donnée.

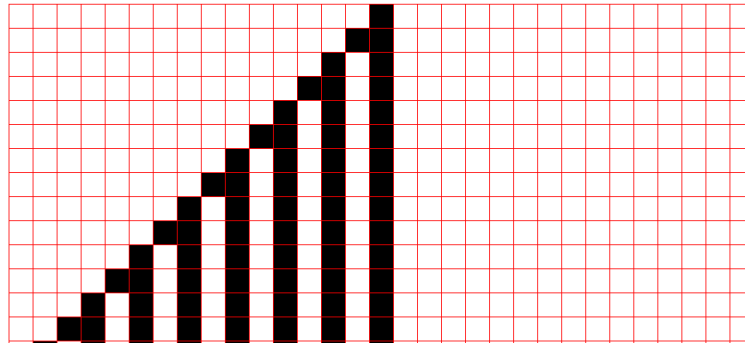
Exemple. Toujours avec la règle `[0, 0, 1, 0, 1, 1, 0, 1]`, pars d'une ligne définie par une seule cellule au milieu :

```
ligne = [0]*10 + [1] + [0]*10
```

Alors l'itération du processus correspond à l'évolution d'une cellule, qui voyage vers la droite en se dédoublant une fois sur deux.



5. **Affichage.** Programme une fonction `afficher_lignes(n, ligne, regle)` qui réalise un bel affichage graphique d'une ligne de cellules et de son évolution sur n lignes. La présence d'une cellule (marquée par 1) est affichée par une case noire, l'absence de cellule (marquée par 0) est affichée par une case blanche.



6. **Numérotation des règles.** Il y a en tout $2^8 = 256$ règles possibles, car une règle est une liste de 8 bits. On décide donc de numéroter la règle en fonction du nombre binaire représenté par la liste :

$$\underbrace{[0, 0, 1, 0, 1, 1, 0, 1]}_{\text{règle}} \longleftrightarrow \underbrace{0.0.1.0.1.1.0.1}_{\text{nombre binaire}} \longleftrightarrow \underbrace{45}_{\text{numéro}}$$

Écris une fonction `definir_regle(numero)` qui n'est autre que la conversion d'un entier en écriture binaire sur 8 bits. Par exemple `definir_regle(45)` renvoie la règle `[0, 0, 1, 0, 1, 1, 0, 1]`.

Algorithme.

- — Entrée : un entier n entre 0 et 255.
- — Sortie : le nombre n en écriture binaire sous la forme d'une liste de 8 bits.
- Démarrer avec une liste vide.
- Répéter 8 fois :
 - Ajouter $n \% 2$ au début de la liste.
 - Faire $n \leftarrow n // 2$.
- Renvoyer la liste.

7. Types d'automates.

En partant d'une seule cellule, essaie de trouver différents types de comportements :

- des automates cellulaires qui convergent vers un état stable (voire vide),
- des automates cellulaires qui convergent vers un état périodique,
- des automates cellulaires ayant des structures symétriques (par exemple, qui réalisent des triangles de Sierpinski),
- des automates cellulaires avec des structures qui semblent aléatoires.

