

## Exercice 2

*Thèmes abordés : programmation Python, tuples et listes*

L'objectif de cet exercice est de mettre en place une modélisation d'un jeu de labyrinthe en langage Python.

On décide de représenter un labyrinthe par un tableau carré de taille  $n$ , dans lequel les cases seront des 0 si l'on peut s'y déplacer et des 1 s'il s'agit d'un mur. Voici un exemple de représentation d'un labyrinthe :



```
laby=[[0,1,1,1,1,1,1,1,1,1],  
      [0,0,0,0,0,0,0,1,0,1],  
      [1,0,1,1,1,1,0,1,0,1],  
      [1,0,1,0,0,0,0,0,0,1],  
      [1,0,1,1,1,1,1,0,1,1],  
      [1,0,1,0,0,0,1,0,1,1],  
      [1,0,1,0,1,0,1,0,1,1],  
      [1,0,1,1,1,0,1,0,1,1],  
      [1,0,0,0,0,0,1,0,0,1],  
      [1,1,1,1,1,1,1,1,0,0]]
```

L'entrée du labyrinthe se situe à la première case du tableau (celle en haut à gauche) et la sortie du labyrinthe se trouve à la dernière case (celle en bas à droite).

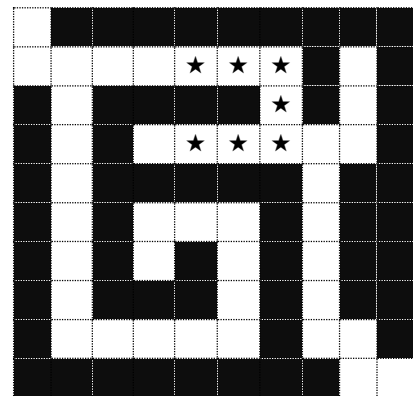
1. **Proposer**, en langage Python, une fonction `mur`, prenant en paramètre un tableau représentant un labyrinthe et deux entiers `i` et `j` compris entre 0 et `n-1` et qui renvoie un booléen indiquant la présence ou non d'un mur. Par exemple :

```
>>mur(laby, 2, 3)
True
>>mur(laby, 1, 8)
False
```

Un parcours dans le labyrinthe va être représenté par une liste de **cases**. Il s'agit de couples  $(i, j)$  où  $i$  et  $j$  correspondent respectivement aux numéros de ligne et de colonne des cases successivement visitées au long du parcours. Ainsi, la liste suivante

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

correspond au parcours repéré par des étoiles `*` ci-contre :



La liste `[(0,0), (1,0), (1,1), (5,1), (6,1)]` ne peut correspondre au parcours d'un labyrinthe car toutes les cases parcourues successivement ne sont pas adjacentes.

2. On considère la fonction `voisine` ci-dessous, écrite en langage Python, qui prend en paramètres deux cases données sous forme de couple.

```
def voisine(case1, case2) :
    l1, c1 = case1
    l2, c2 = case2
    # on vous rappelle que **2 signifie puissance 2
    d = (l1-l2)**2 + (c1-c2)**2
    return (d == 1)
```

2.a. Après avoir remarqué que les quantités  $l1-l2$  et  $c1-c2$  sont des entiers, **expliquer** pourquoi la fonction `voisine` indique si deux cases données sous forme de tuples  $(l, c)$  sont adjacentes.

2.b. **En déduire** une fonction `adjacentes` qui reçoit une liste de cases et renvoie un booléen indiquant si la liste des cases forme une chaîne de cases adjacentes.

Un parcours sera qualifié de **compatible avec le labyrinthe** lorsqu'il s'agit d'une succession de cases adjacentes accessibles (non murées). On donne la fonction `teste(cases, laby)` qui indique si le chemin `cases` est un chemin possible compatible avec le labyrinthe `laby` :

```
def teste(cases, laby) :
    if not adjacentes(cases) :
        return False
    possible = True
    i = 0
    while i < len(cases) and possible:
        if mur(laby, cases[i][0], cases[i][1]) :
            possible = False
        i = i + 1
    return possible
```

3. **Justifier** que la boucle de la fonction précédente se termine.
4. **En déduire** une fonction `echappe(cases, laby)` qui indique par un booléen si le chemin `cases` permet d'aller de l'entrée à la sortie du labyrinthe `laby`.