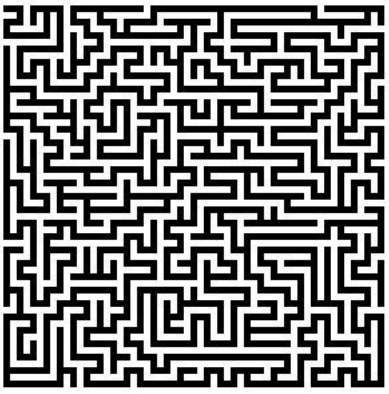
TP D'INFORMATIQUE 3
Labyrinthes parfaits

Dans ce TP, on considère des labyrinthes sur les points de $[\![0, n-1]\!]^2$, chaque point pouvant être relié à n'importe lequel de ses plus proches voisins. Un tel labyrinthe est dit **parfait** s'il y a un et un seul chemin entre chaque paire de points.



Un labyrinthe parfait

1 Dessin d'un labyrinthe parfait

L'objectif de cette section est d'afficher à l'écran un labyrinthe parfait aléatoire de dimensions $n \times n$, en s'appuyant sur la structure de pile.

1. Reprendre en début de fichier les fonctions implémentant la structure de pile
2. Lors du tracé, nous allons tenir à jour dans une matrice de booléens de taille $n \times n$ les points qui ont déjà été atteints dans le labyrinthe. Écrire une fonction `est_atteint` prenant en argument une telle matrice M et deux entiers x et y, et renvoyant `True` ou `False` selon que le point (x, y) a déjà été atteint ou non. On renverra `True` pour les points extérieurs à $[\![0, n-1]\!]^2$.
3. Écrire une fonction `choix` prenant en argument une telle matrice M et deux entiers x et y et renvoyant la liste des points à distance 1 de (x, y) et qui n'ont pas été déjà atteints.
4. Écrire une fonction `tirage` prenant en argument une liste L et renvoyant de façon uniformément aléatoire un élément de L. On pourra utiliser l'instruction `from random import *` et la fonction `randint`.
5. Comparer la création d'une matrice de booléens de taille 3*3 avec `M = [[False]*3]*3` et `N = [[False]*3 for i in range(3)]`.
   Quelle différence observe-t-on lorsqu'on modifie une case de M et de N ? Quelle écriture est préférable ?
6. Écrire une fonction `labyrinthe` prenant en argument la dimension n, et dessinant un labyrinthe parfait. On pourra suivre les étapes suivantes :
   - Utiliser l'instruction `from matplotlib.pyplot import *`
   - Utiliser les instructions `axis('off')` et `axis('equal')` pour ne pas afficher les axes et prendre des échelles égales.
   - Créer une matrice booléenne enregistrant les points déjà atteints.
   - Créer une pile P dans laquelle on stockera les points à partir desquels on est susceptible d'ouvrir un nouveau chemin.
   - Empiler le point (0,0) dans P et marquer ce point comme atteint.

- Tant que P n'est pas vide, on effectue l'une des deux branches suivantes :
  — Si le sommet de P n'a pas de voisins non atteints, c'est un cul-de-sac : on se contente de le dépiler.
  — Si le sommet de P a au moins un voisin non atteints, on en tire un au hasard, on le marque comme atteint, on l'empile et on trace le chemin correspondant (on pourra utiliser l'instruction `plot([x,x2],[y,y2],'-r')`, où (x, y) est le point initial et $(x2, y2)$ le voisin sélectionné).

2 Enregistrement d'un labyrinthe parfait

Afin de pouvoir effectuer le parcours d'un labyrinthe, il va falloir le représenter en mémoire, plutôt que simplement l'afficher. On représentera un labyrinthe comme une matrice $n \times n$, chaque case de coordonnées (x, y) contenant une liste de 4 booléens indiquant les chemins existant à partir de ce point. L'indice 0 correspondra à la droite, l'indice 1 au haut, l'indice 2 à la gauche et l'indice 3 au bas.
Compléter la fonction précédente pour avoir comme valeur de retour une telle matrice représentant le labyrinthe généré.

3 Parcours d'un labyrinthe parfait

1. Écrire une fonction `parcours` prenant en argument un labyrinthe, un point de départ et un point d'arrivée, et renvoyant une liste de directions représentant le chemin menant de ce point de départ à ce point d'arrivée.
2. Compléter la fonction précédente pour qu'elle dessine le bon chemin en bleu par dessus le dessin d'un labyrinthe.
   On pourra utiliser des instructions sur le modèle de `t, = plot([x,x2],[y,y2],'-b')` pour tracer un trait bleu de (x, y) à $(x2, y2)$ en le stockant dans la variable t. L'instruction `t.remove()` permet alors d'effacer ce trait.