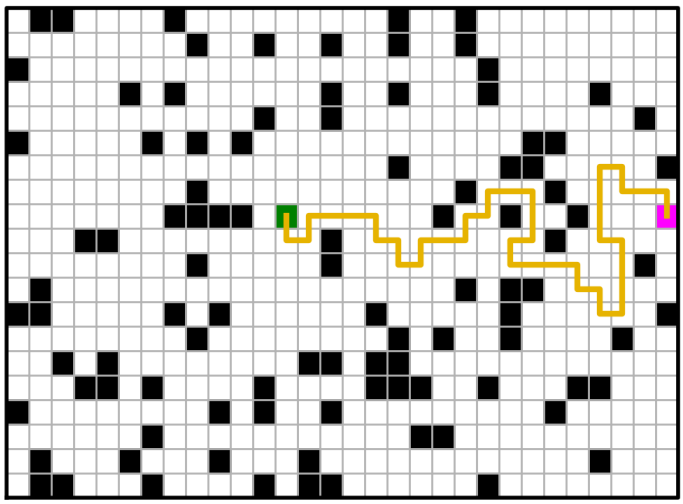
CHEMINS DANS UNE GRILLE

TP

Alors voilà : on a une grille, des obstacles, une case de départ (en rose), une case d'arrivée (en vert), et il faut aller de l'une à l'autre.



Partie I — Grilles

Exercice 1 — Que fait le programme ci-dessous ?

```
def CréerGrille(M, N) :
    return [[None for j in range(N)] for i in range(M)]
```

Exercice 2 — Dans tout le reste de cette fiche, on appellera *grille* tout objet créé par le programme précédent. La *taille* d'une grille sera le couple (M, N). Écrire un programme `Taille(G)` qui renvoie ce couple.

Exercice 3 — Écrire le programme `Copie(G)` qui fait ce qu'on imagine.

Partie II — Cases aléatoires

Exercice 4 — Écrire un programme `CasesVides(G)` qui construit et renvoie la liste des couples (i, j) tels que `G[`i`][`j`]` contient la valeur `None`.

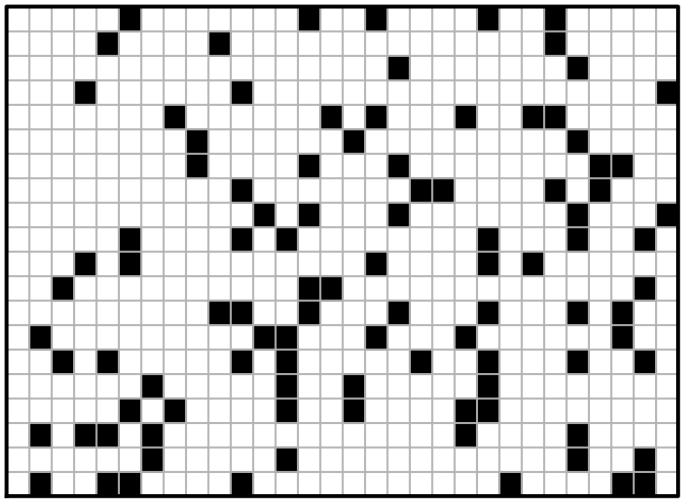
Exercice 5 — On importe les deux fonctions ci-dessous. En utilisant la documentation, préciser ce qu'elles font.

```
from random import randint as EntAléa, shuffle as Mélanger
```

Exercice 6 — Nous allons murer certaines cases d'une grille. Ce qui, pour ce qui nous concerne, va consister à attribuer la valeur `"MUR"` à certaines cases.

1) Expliquer comment, avec la fonction `Mélanger`, choisir aléatoirement n éléments distincts (c'est-à-dire situés dans des cases différentes, même s'ils peuvent avoir la même valeur) d'une liste.

2) En déduire un programme `CréerMurs(G, n)` qui mure n cases vides distinctes de la grille `G`.



Partie III — Représentation graphique

Dans cette partie, on va programmer les fonctions auxiliaires du grand programme suivant.

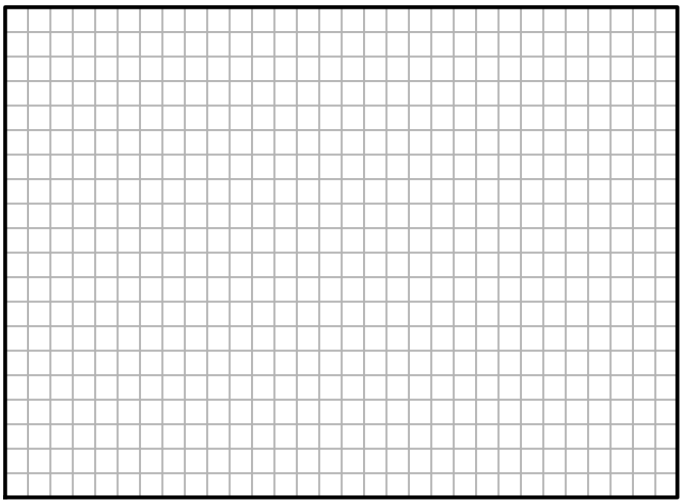
```
from matplotlib.pyplot import axis, fill, plot, show
def DessinerTout(G, CasesSpéciales, Chemins) :
    (M, N) = Taille(G)
    DessinerGrille(M, N)
    DessinerMurs(G)
    for (i, j, couleur) in CasesSpéciales :
        DessinerCase(i, j, couleur)
    for (ch, couleur) in Chemins :
        DessinerChemin()
    axis([-1, N, -1, M]) ; show()
```

Exercice 7 — La case (i, j) sera représentée par un petit carré, de centre (j, i), et de côté 1.

1) Expliquer pourquoi (j, i) et pas (i, j).

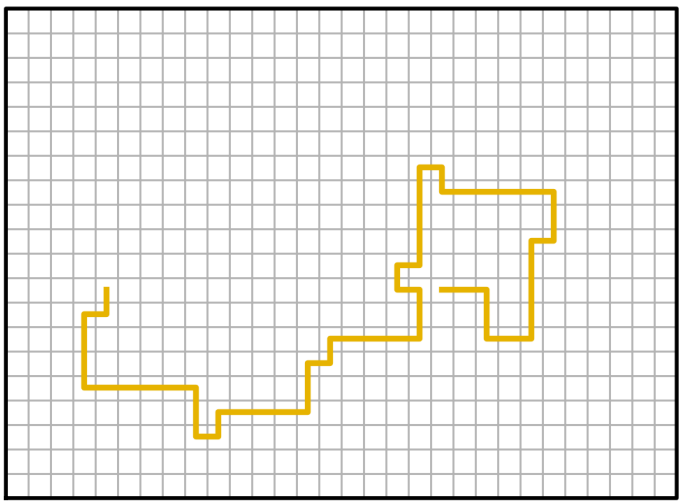
2) En déduire le programme `DessinerCase(`i`, `j`, couleur)` qui peint la case (i, j) avec la `couleur`. On utilisera l'instruction `fill`.

Exercice 8 — Écrire un programme `DessinerGrille(M, N)` qui dessine le quadrillage (en gris clair) et les bords (en noir et un peu plus épais).



Exercice 9 — Écrire le programme `DessinerMurs(G)` qui peint en noir les cases murées de la grille.

Exercice 10 — Appelons *chemin* une suite de cases (supposées vides) d'une grille, rangées dans une liste `ch`, telle que pour tout indice k, `ch[`k`]` et `ch[`$k+1$`]` soient des cases adjacentes de la grille. Écrire un programme `DessinerChemin(ch, couleur)` qui fait ce qu'on imagine.

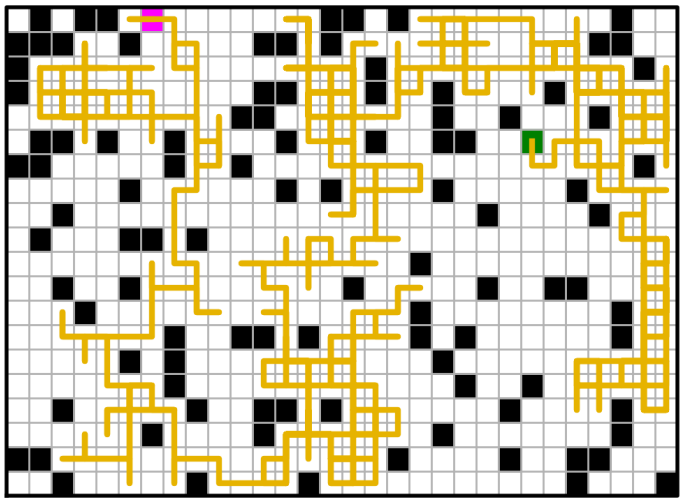


Partie IV — Chemins

Exercice 11 — Écrire un programme `VoisinesVides(G,` i`,` j`)` qui étant donnée une grille `G` et une case (i, j) construit et renvoie la liste des couples (i', j') correspondant à des cases à adjacentes à `G[`i`][`j`]`, et contenant la valeur `None`. Une case a au plus quatre voisines, éventuellement moins si elle est située sur un bord ou dans un coin.

Exercice 12 — Écrire le programme `Choisir(L)` qui renvoie un élément aléatoire d'une liste.

Exercice 13 — Déduire des deux exercices précédents un programme `MarcheAléatoire(G,` i_0`,` j_0`,` i_F`,` j_F`)` qui construit un chemin reliant les cases (i_0, j_0) et (i_F, j_F). On utilisera un alorithme naïf, qui part de la case (i_0, j_0), puis qui se déplace aléatoirement de voisines en voisines, jusqu'à trouver la case cible. Ce chemin passe sans doute de nombreuses fois par les mêmes cases.



Exercice 14

1) Écrire un programme `ChercherRépétition(L)` qui étant donnée une liste renvoie un couple (i, j), avec $i < j$, tel que `L[`i`]` est égal à `L[`j`]`.

2) En déduire un programme `Simplifier(ch)` qui prend en argument un chemin, cherche une case par laquelle il passe deux fois (disons `ch[`k_1`]` et `ch[`k_2`]`), et renvoie le chemin obtenu en supprimant la boucle correspondante (donc la portion `ch[`k_1 `:` k_2`]`). Ce programme renverra `ch` s'il ne comporte aucune boucle.

3) Écrire finalement le programme `SimplifierAuMaximum(ch)` qui construit et renvoie le chemin obtenu en supprimant toutes les boucles de `ch`.

Exercice 15

1) Écrire un programme `TrouverChemin(G,` i_0`,` j_0`,` i_F`,` j_F`)` qui construit et renvoie un chemin sans boucle reliant les cases (i_0, j_0) et (i_F, j_F).

2) Tester tout ce qui précède : en générant une grille vide, en y ajoutant des murs, en choisissant une case de départ et une case d'arrivée aléatoires, et en calculant un chemin les reliant. Et évidemment, on veut dessiner le résultat !