# Correction

NSI - 2021 Métropole Jour 1 (21-NSIJ1ME2)

## Exercice 4 - Labyrinthe

### Partie A : Représentation d'un labyrinthe

1. `lab2[1][0] = 2`

2.
```
def est_valide(i, j, n, m):
    """ Renvoie True si les coordonnées (i, j) correspondent à des coordonnées
        valides du labyrinthe de taille (n, m), et False sinon."""
    return -1 < i < n and -1 < j < m
```

3.
```
def depart(lab):
    n = len(lab)
    m = len(lab[0])
    for y in range(n):
        for x in range(m):
            if lab[y][x] == 2:
                return (y, x)
    return None
```

4.
```
def nb_cases_vides(lab):
    """ Renvoie le nombre de cases vides du
    labyrinthe (avec l'arrivée et le départ)."""
    total = 0
    for y in range(len(lab)):
        for x in range(len(lab[0])):
            if lab[y][x] in (0, 2, 3):
                total += 1
    return total
```

# Correction

NSI - 2021 Métropole Jour 1 (21-NSIJ1ME2)

## Partie B : Recherche d'une solution dans un labyrinthe

1. L'appel `voisines(1, 2, [[1, 1, 4], [0, 0, 0], [1, 1, 0]])`
renvoie `[(1, 1), (2, 2)]`

2. a.
```
# entrée: (1, 0), sortie (1, 5)
chemin = [(1, 0)]
chemin.append((1, 1))
chemin.append((2, 1))
chemin.pop() # Supprime la dernière case de la liste (impasse)
chemin.append((1, 2))
chemin.append((1, 3))
chemin.append((2, 3))
chemin.append((3, 3))
chemin.append((3, 4))
chemin.pop()
chemin.pop()
chemin.pop()
chemin.append((1, 4))
chemin.append((1, 5))
```

2. b.
```
def solution(lab):
    chemin = [depart(lab)] # ajout de la case départ
    case = chemin[0]
    i = case[0]
    j = case[1]
    while lab[i][j] != 3: # Tant qu'on est pas sur l'arrivée
        lab[i][j] = 4
        cases_voisines = voisines(i, j, lab)
        if not cases_voisines :   # S'il n'y a pas de voisines
            chemin.pop()          # enlève la dernière case
        else:                     # Sinon ajout de la première case voisine
            chemin.append(cases_voisines[0])
            i, j = cases_voisines[0]
    return chemin
```