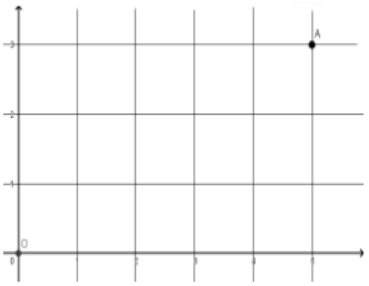
On considère un jeu de plateforme où un personnage se déplace dans un espace à deux dimensions. Pour cela, on autorise seulement deux déplacements élémentaires : de la gauche vers la droite ou du bas vers le haut.
La longueur d'un déplacement correspond au nombre de déplacements élémentaires qui le constituent. Afin de représenter ces déplacements, on se place dans un repère où les coordonnées sont des nombres entiers positifs.
Le personnage au début du jeu est situé à l'origine du repère de coordonnées (0,0) et il souhaite se rendre au point A de coordonnées (5,3).



On décide de coder les déplacements élémentaires de la manière suivante : le caractère '0' représente un déplacement élémentaire vers la droite, le caractère '1' représente un déplacement élémentaire vers le haut. Un déplacement de longueur n sera donc une chaîne de caractères composée de n caractères '0' ou '1'. Par exemple, un déplacement possible de O à A est '00011001' et sa longueur est 8

1. On considère la fonction mystere ci-dessous.

```
def mystere(dep):
'''
entrée: dep est une chaîne de caractère
contenant des 0 et des 1 modélisant
un déplacement
'''
x = 0
y = 0
for c in dep:
    if c == '0':
        x = x+1
    else:
        y = y+1
return [x, y]
```

   (a) Que renvoie mystere('01110111') ?
   (b) De manière plus générale, que renvoie la fonction mystere pour une chaîne représentant un déplacement donné ?

2. Ecrire une fonction `accessible`(dep, arrivee) de sorte qu'elle renvoie True si le déplacement dep se termine sur le point `arrivee` et False dans le cas contraire.

   Les types des paramètres sont donc : dep : une chaîne de caractères
   arrivee : une liste de deux entiers

   Par exemple : accessible('00110001', [5,3]) renvoie True alors que accessible('01010111', [5,3]) renvoie False.

   On rappelle que l'expression `str(randint`(0,1)) utilisant la fonction randint de la bibliothèque random renvoie aléatoirement un caractère égal à '0' ou '1'. On décide de trouver un déplacement

de longueur 8 qui permette d'atteindre le point `arrivee`. Pour cela, on se propose de créer de manière aléatoire un déplacement de longueur 8, de tester si le point `arrivee` est accessible pour ce déplacement et de recommencer tant que ce n'est pas le cas.

3. Recopier et compléter la fonction chemin ci-dessous qui prend en paramètre les coordonnées du point `arrivee` et qui renvoie un déplacement de 8 pas qui permette d'atteindre le point `arrivee`. Quelles sont les préconditions sur le paramètre `arrivee` ?

```
from random import randint
def chemin(arrivee):
    deplacement = '00000000'
    while ..... :
        .....
    for k in range(8):
        pas = str(randint(0,1))
        ..... = deplacement + .....
    return deplacement
```

4. Ecrire une version généralisée `chemin_v2` qui renvoie un déplacement qui permet d'atteindre un point d'arrivée quelconque.

5. Ecrire une version (on appellera la fonction `tous_les_chemins`) qui renvoie la liste de tous les chemins qui mènent vers l'arrivée.

6. Tester le programme suivant et essayer de comprendre ce qu'il fait.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection
import random

# 1. Génération des données du trajet
x, y = 0, 0
points = [(x, y)]
limite = 10
nb_pas = 10

for _ in range(nb_pas):
    direction = random.choice(['droite', 'haut'])
    if direction == 'droite' and x < limite:
        x += 1
    elif direction == 'haut' and y < limite:
        y += 1
    points.append((x, y))

# 2. Préparation des segments pour la coloration
# On transforme la liste de points en liste
# de segments [(p1, p2), (p2, p3), ...]
pts = np.array(points)
segments = np.concatenate([pts[:-1, np.newaxis, :],\
 pts[1:, np.newaxis, :]], axis=1)

# 3. Création de la figure
fig, ax = plt.subplots(figsize=(8, 8))

# Création de la collection de lignes avec un dégradé (cmap)
# 'viridis', 'plasma' ou 'jet' sont de bons choix
norm = plt.Normalize(0, len(segments))
lc = LineCollection(segments, cmap='viridis', norm=norm, linewidth=3)

# On définit les couleurs en fonction de l'index du segment
```

```
lc.set_array(np.arange(len(segments)))

# Ajout de la ligne à l'axe
ax.add_collection(lc)

# 4. Habillage du graphique
ax.set_xlim(-0.5, limite + 0.5)
ax.set_ylim(-0.5, limite + 0.5)
ax.set_xticks(range(limite + 1))
ax.set_yticks(range(limite + 1))
ax.grid(True, linestyle='--', alpha=0.5)

# Ajout d'une barre de couleur pour indiquer le temps (début -> fin)
#cbar = plt.colorbar(lc, ax=ax)
#cbar.set_label('Progression du trajet (Temps)')

# Marquer le départ et l'arrivée
plt.scatter(points[0][0], points[0][1], color='green',\
 label='Départ', zorder=5)
plt.scatter(points[-1][0], points[-1][1], color='red',\
 marker='X', s=100, label='Arrivée', zorder=5)

plt.legend()
plt.title("Trajectoire ")
plt.show()
```

7. Modifier le programme précédent afin qu'il affiche le trajet du point (0,0) à un point d'arrivée dont les coordonnées sont entrées au clavier et en **utilisant la fonction chemin_v2**

8. On peut apporter d'autres modifications .Par exemple:
   On peut faire une animation : visualiser un point mobile qui fait le trajet de (0,0) vers le point d'arrivée.
   On peut également faire une version du programme qui propose un point de départ autre que (0,0).