NSI : TP, suite de l'étude de la percolation d'un milieu de dimensions et densité données.

On va à présent simuler des expériences et essayer de déterminer le seuil de percolation en fonction des dimensions du milieu et de sa densité. Pour cela, on va créer un nouveau module simulation_experiences, qui va avoir recours au module percolation.

On importe le module nommé percolation qui est notre programme percolation.py.

```
# on importe le module percolation avec le prefixe perco
import percolation as perco
```

On peut alors faire appel aux fonctions du module percolation à l'aide du préfixe percolation :

```
g=perco.grille(10,10,0.6)
perco.afficher(g) # affichera notre grille déterminée aléatoirement
amas=perco.amas(g) # détermine les amas de notre grille
perco.percolation(g,amas) # retourne s'il y a ou non percolation par un bolléen
```

A. Mise en place de la structure de données

1. Simuler une expérience.

On va s'intéresser à présent aux résultats de nos simulations de percolation.

Pour cela, on va utiliser un dictionnaire pour stocker les données de notre expérience sous la forme :

{'dimensions': (10, 10), 'densité': 0.5, 'fréquence': 0.43, 'percolation': False}

Compléter la fonction suivante :

```
def experience(largeur,hauteur,densite):
    '''
    donne le résultat de l'expérience sous la forme d'un dictionnaire
    : largeur: int
    : hauteur : int
    : densité : float entre 0 et 1
    Exemple : experience(10,10,0.5) doit retourner un dictionnaire du type
     {'dimensions': (10, 10), 'densité': 0.5, 'fréquence': 0.43, 'percolation': False}
    '''
```

```
    g=perco.grille(largeur,hauteur,densite)
    amas=perco.amas(g)
    retour=dict()
    retour["dimensions"]=
    retour["densité"]=
    retour["fréquence"]=
    retour["percolation"]=
    return retour
```

2. Simuler une série d'expériences.

On va à présent simuler un grand nombre d'expériences de même taille et de même densité, pour étudier la fréquence de percolation en fonction de ces paramètres. On va également faire appel à un dictionnaire pour avoir un résultat du type, pour 10 expériences :

{'taille': (10, 10), 'densité': 0.5, 0: True, 1: False, 2: True, 3: False, 4: False, 5: False, 6: True, 7: False, 8: False, 9: False}

Compléter la fonction :

```
def serie_experiences(largeur,hauteur,densite,nombre):
    '''
    simule le nombre d'expériences désiré et renvoie le
    resultat de la percolation pour chaque experience.
    Le resultat est stocké sous la forme d'un dictionnaire.
    largeur : int
    hauteur : int
    densite : float entre 0 et 1
    nombre :int
    retour : dict avec pour clés le numero de l'expérience et valeurs le résultat de la percolation
    exemple : serie_experiences(10,10,0.5,10) renvoie le résultat de 10 expériences
    sur des milieux de dimmension 10,10 et de densité 0.5 sous la forme d'un dictionnaire
{'taille': (10, 10), 'densité': 0.5, 0: True, 1: False, 2: True, 3: False, 4: False, 5: False, 6: True, 7: False, 8: False, 9: False}
    '''
    retour={}
    retour["taille"]=
    retour["densité"]=
    for i in range(nombre):
        resultat_exp=experience(largeur,hauteur,densite)
        retour[i]=
    return retour
```

3.Exploitation d'une série d'expériences

On va déterminer à présent la fréquence de percolations sur notre échantillon d'expériences.

Compléter la fonction suivante

```
def frequence(serie):
    '''
    retourne le nombre de valeurs True présenter dans le dictionnaire
    serie : dict
    >>>d={'taille': (10, 10), 'densité': 0.5, 0: False, 1: True, 2: False,
3: True, 4: False, 5: False, 6: False, 7: False, 8: False, 9: False}
    >>>print(frequence(d))
    0.2
    '''
```

4. On va maintenant pouvoir rechercher notre seuil de percolation pour un milieu de dimemsions et densité données.

Pour rechercher ce seuil, on peut utiliser une méthode par balayage : on va parcourir les densités comprises entre 0 et 1 avec un pas donné (les dimensions sont fixées). On va déterminer pour chaque densité, la fréquence de percolation. On va recherche à partir de quelle densité la percolation semble basculer de 0 à 1 en fréqence. les commandes : d=recherche_seuil_percolation(10,10,100,0.1) print(d) retourneront un résultat du type : {'taille': (10, 10), "nombre d'expériences": 100, 'densité= 0': 1.0, 'densité= 0.1': 1.0, 'densité= 0.2': 0.99, 'densité= 0.30000000000000004': 0.94, 'densité= 0.4': 0.58, 'densité= 0.5': 0.23, 'densité= 0.6': 0.03, 'densité= 0.7': 0.0, 'densité= 0.7999999999999999': 0.0, 'densité= 0.8999999999999999': 0.0, 'densité= 0.9999999999999999': 0.0}

Compléter la fonction

```
def recherche_seuil_percolation(largeur,hauteur,nombre_exp,pas):
    '''
    d=recherche_seuil_percolation(10,10,100,0.1) retourne par un
dictionnaire du type
    {'taille': (10, 10),
```

```
    "nombre d'expériences": 100,
    'densité= 0': 1.0,
    'densité= 0.1': 1.0,
    'densité= 0.2': 0.99,
    'densité= 0.30000000000000004': 0.94,
    'densité= 0.4': 0.58,
    'densité= 0.5': 0.23,
    'densité= 0.6': 0.03,
    'densité= 0.7': 0.0,
    'densité= 0.7999999999999999': 0.0,
    'densité= 0.8999999999999999': 0.0,
    'densité= 0.9999999999999999': 0.0}
    '''
    retour={}
    retour["taille"]=
    retour["nombre d'expériences"]=
    d=0
    while d<1:
        densite=d
        ma_serie_d_experiences=
        retour["densité= "+str(densite)]=
        d+=
    return retour
```

B. Algorithmes de recherche d'une valeur spécifique dans un ensemble de valeurs.

Un algorithme est une succesion finies d'opérations permettant d'obtenir un résultat qui est souvent la réponse à un problème donné.
Il est important de s'assurer qu'un algorithme répond bien à la résolution du problème posé et donc de vérifier sa véracité.

La véracité d'un algorithme n'est pas assurée par sa véracité sur un certain nombre de test.

En effet, lors des tests, on peut avoir oublié par exemples certains cas particuliers.

Il existe une méthode mathématique pour montrer qu'un algorithme permet bien de répondre au problème étudie : la notion d'invariant (d'algorithme ou de boucle).

Tout invariant d'algorithme doit vérifier 3 conditions pour assurer la véracité de notre algorithme :
1 : Il est vrai à l'initialisation de notre algorithme.

2 : Il reste vrai lors de l'éxécution de notre algorithme.

3 : La terminaison de l'algorithme ou le test d'arrêt de notre algorithme assure que notre invariant est la solution au problème que doit résoudre notre algorithme.

Deux exemples :

Revenons sur la fonction qui permettait d'obtenir les amas de notre tp percolation :

```
def amas(grille):
    mes_amas=()
    for i in range(len(grille[0])):
        if grille[0][i]==0 and cellule_pas_dans_mes_amas(mes_amas,i,0):
            amas=((i,0),)
            frontiere=bord_cellules_amas(amas,grille,mes_amas)
            while len(frontiere)!=0:
                amas+=frontiere
                frontiere=bord_cellules_amas(frontiere,grille,amas)
            mes_amas+=(amas,)


    return mes_amas
```

Cette fonction contient deux algorithmes :

- un algorithme de recherche de tous les amas issus du haut de la grille,
- un algorithme de recherche d'un amas à partir d'une case de la grille.

1) Identifier bien ces 2 algorithmes dans notre fonction.

2) Préciser l'invariant pour chaque algorithme et le test de terminaison.

3) Déterminer la véracité de ces deux algorithmes.