Exercice 1. Plus court chemin dans le plan

```
import math
import random
import matplotlib.pyplot as plt
```

Question 1 –

```
def make_dist(pts):
    n = len(pts)
    dist = [[0]*n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            x1,y1 = pts[i]
            x2,y2 = pts[j]
            dist[i][j] = math.sqrt((x2-x1)**2 + (y2-y1)**2)
    return dist
```

Question 2 –

```
def prochain_point(dist, i, vis):
    n = len(dist)
    dist_min = None
    j_min = None
    for j in range(n):
        if not vis[j]:
            if dist_min is None or dist[i][j] < dist_min:
                dist_min = dist[i][j]
                j_min = j
    return j_min
```

Question 3 –

```
def relier_glouton(pts):
    n = len(pts)
    if n == 0:
        return []
    dist = make_dist(pts)
    L = [0]
    vis = [False]*n
    vis[0] = True
    while len(L) < n:
        j = prochain_point(dist, L[-1], vis)
        vis[j] = True
        L.append(j)
    return L
```

Question 4 –

La fonction `make_dist` s'exécute en temps quadratique en n.
La fonction `prochain_point` s'exécute en temps linéaire en n.
La fonction `relier_glouton` s'exécute en temps quadratique en n.

Question 5 –

```
def make_pts_alea(n):
    pts = []
    for _ in range(n):
        x = random.random()
        y = random.random()
        p = x,y
        pts.append(p)
    return pts
```

```
def plot_aleatoire(n):
    pts = make_pts_alea(n)
    L = relier_glouton(pts)
    X = []; Y = []
    for i in L:
        x,y = pts[i]
        X.append(x)
        Y.append(y)
    plt.figure()
    x0,y0 = pts[0]
    plt.plot(x0,y0,'o')
    plt.plot(X,Y,'x-')
    plt.axis('equal')
    plt.show()
```

Question 6 – On considère la liste de points

`[(0,0), (1,0), (2,0), (-1.1,0)]`.

L'algorithme glouton va relier les points dans l'ordre : $A_0 - A_1 - A_2 - A_3$ pour une longueur totale de `5.1`. En reliant les points dans l'ordre : $A_0 - A_3 - A_1 - A_2$, on obtient une longueur totale de `4.2`.

Exercice 2. Programme TV

```
import random
```

Question 1 – Montrons que les stratégies 1 et 2 ne sont pas optimales. On prend l'exemple de la liste `[(0,10), (1,5), (6,10), (4,7)]`. Les programmes sont donc les suivants :

	Début	Fin	Durée
Programme 0	0	10	10
Programme 1	1	5	4
Programme 2	6	10	4
Programme 3	4	7	3

- La stratégie 1 sélectionne en priorité les programmes dont la durée est la plus courte. On sélectionne donc d'abord le programme 3. Il n'est alors plus possible de regarder un autre programme.
- La stratégie 2 sélectionne en priorité les programmes qui commencent le plus tôt. On sélectionne donc d'abord le programme 0. Il n'est alors plus possible de regarder un autre programme.

Dans les deux cas, on ne peut regarder qu'un seul programme alors qu'il est possible d'en regarder deux (programmes 1 et 2).

Question 2 – On sélectionne les programmes en choisissant en priorité ceux qui terminent le plus tôt. Montrons que cette stratégie est optimale.

Soient $i_1, \dots, i_k$ les indices des programmes obtenus par la stratégie ci-dessus rangés par ordre chronologique. On suppose qu'avec la stratégie optimale, il est possible de regarder ℓ programmes dont les indices sont $j_1, \dots, j_\ell$ (suivant l'ordre chronologique). On sait déjà que $k \leqslant \ell$ et on souhaite montrer que $k = \ell$.

Pour tout i, on note d_i et f_i les heures de début et de fin du programme d'indice i. Montrons par récurrence sur $m \in [\![1, k]\!]$ que $f_{i_m} \leqslant f_{j_m}$:

- Soit $m = 1$. Par définition de i_1, f_{i_1} est le minimum des f_k, donc la propriété est vérifiée.
- Soit $m \in \mathbb{N}$ avec $1 \leqslant m < k$. On suppose que $f_{i_m} \leqslant f_{j_m}$ et on montre que $f_{i_{m+1}} \leqslant f_{j_{m+1}}$. Par définition, $f_{i_{m+1}}$ est le plus petit des f_i tels que $d_i \geqslant f_{i_m}$. Comme, $d_{j_{m+1}} \geqslant f_{j_m} \geqslant f_{i_m}$, on conclut : $f_{i_{m+1}} \leqslant f_{j_{m+1}}$.

En prenant $m = k$, on obtient $f_{i_k} \leqslant f_{j_k}$. De plus, il n'existe pas d'indice i tel que $d_i \geqslant f_{i_k}$, sinon l'algorithme glouton aurait trouvé un programme à regarder après i_k. Donc il n'existe pas de i tel que $d_i \geqslant f_{j_k}$, c'est à dire que j_k est le dernier indice de la solution optimale. On en déduit que $\ell = k$.

Question 3 –

```
# Hypothèse: les heures de début et de fin sont >= 0
def prog_tv_glouton(prog):
    prog2 = []
    for i in range(len(prog)):
        d,f = prog[i]
        e = (f,d,i)
        prog2.append(e)
    prog2 = sorted(prog2)
    res = []
    libre = 0  # Instant où la personne est libre
    for f,d,i in prog2:
        if libre <= d:
            libre = f
            res.append(i)
    return res
```

Exercice 3. Allocation de salles de cours

```
import matplotlib.pyplot as plt
import random
```

Question 1 –

```
def trier_cours(cours):
    cours2 = [(f,d) for d,f in cours]
    cours3 = sorted(cours2)
    cours_tries = [(d,f) for f,d in cours3]
    return cours_tries
```

Question 2 –

```
def possible(salle, d1, f1):
    """
    salle: list[int,int] ---  La liste des cours ayant lieu dans la salle
    d1: int --- Heure de début du cours
    f1: int --- Heure de fin du cours
    return: bool --- Indique si la salle peut accueillir le cours
    """
    for d2, f2 in salle:
        if d1 < f2 and d2 < f1:
            return False
    return True
```

```
def trouver_salle(salles, d1, f1):
    for i in range(len(salles)):
        if possible(salles[i], d1, f1):
            return i
    return len(salles)
```

Question 3 –

```
def allocation_glouton(cours):
    cours_tries = trier_cours(cours)
    salles = []
    for d,f in cours_tries:
        s = trouver_salle(salles, d, f)
        if s == len(salles):
            salles.append([])
        salles[s].append((d,f))
    return salles
```

Question 4 –

```
def generer_cours(n):
    cours = []
    for _ in range(n):
        if random.random() < 0.7:
            duree = 1
            deb = random.choice([8,9,10,11] + [13,14,15,16,17])
        else:
            duree = 2
            deb = random.choice([8,9,10] + [13,14,15,16])
        cours.append((deb, deb + duree))
    return cours
```

Question 5 –

```
def tracer_allocations(salles):
    plt.figure()
    hmin = 7
    hmax = 19
    plt.xticks([i for i in range(hmin,hmax+1)])
    plt.xlim((hmin, hmax))
    plt.yticks([i for i in range(len(salles))])
    for i in range(len(salles)):
        for d,f in salles[i]:
            plt.plot(d,i, 'd', color = 'black')
            plt.plot(f,i, 'd', color = 'black')
            plt.plot([d,f],[i,i], '--', color = 'black')
    plt.show()
```

Exercice 4. Rendu de monnaie – Optimalité de l'algorithme glouton

Question 1.a –
⋆ On commence montrer que n_1 (le nombre de pièces de valeur 1) est 0 ou 1. Si $n_1 \geqslant 2$, alors il suffit de remplacer deux pièces de valeur 1 par une pièce de valeur 2 pour obtenir une meilleure solution. Ceci contredirait l'hypothèse que la solution est optimale.
⋆ Si $n_2 \geqslant 3$, alors il suffit de remplacer trois pièces de valeur 2 par une pièce de 1 et une pièce de 5 pour obtenir une meilleure solution. Ceci contredirait l'hypothèse que la solution est optimale.
⋆ On procède de même pour les autres valeurs.

Question 1.b –
⋆ Si $s \geqslant 200$ alors $v = 200$ et on doit montrer que la stratégie optimale rend au moins un pièce de valeur 200.

Supposons par l'absurde qu'aucune pièce de valeur 200 n'est rendue. D'après la question précédente, si $n_{20} \leqslant 1$, alors la somme rendue est d'au plus 190. Dans notre cas, on a donc $n_{20} = 2$. Par conséquent $n_{10} = 0$, sinon, il suffirait de remplacer les deux pièces de 20 et la pièce de 10 par une pièce de 50 pour obtenir une meilleure solution.

En utilisant $n_{20} = 2$ et $n_{10} = 0$, on déduit de la question précédente que $s \leqslant 200$ donc $s = 200$. Or la seule stratégie optimale pour $s = 200$ est de rendre une pièce de 200, ce qui contredit notre hypothèse.
⋆ On procède de la même façon pour les cas $100 \leqslant s < 200$, puis $50 \leqslant s < 100$, puis $20 \leqslant s < 50$, puis $10 \leqslant s < 20$, puis $5 \leqslant s < 10$, puis $2 \leqslant s < 5$, puis $1 \leqslant s < 2$.

Question 2 – On le montre par récurrence forte sur s :
⋆ Pour $s = 0$, la solution optimale consiste à ne rendre aucune pièce. C'est bien la solution choisie par l'algorithme glouton.
⋆ Soit $s > 0$. Supposons la propriété vraie jusqu'au rang $s-1$, et montrons la au rang s. On note k le nombre de pièces rendues par l'algorithme glouton et ℓ le nombre de pièces rendues par la stratégie optimale. On sait que $\ell \leqslant k$ et on doit montrer que $\ell = k$.

Soit v comme défini dans la question précédente. L'algorithme glouton rend une pièce de valeur v puis rend la somme $s - v$ en utilisant $k - 1$ pièces. D'après la question précédente, la stratégie optimale rend une pièce de valeur v, puis $\ell - 1$ pièces dont la somme des valeurs est $s - v$. Par hypothèse de récurrence, l'algorithme glouton rend la somme $s - v$ de manière optimale ; on en déduit que $k - 1 \leqslant \ell - 1$, c'est à dire $k \leqslant \ell$ et donc $k = \ell$.