MPSI -ITC - TD 12 - 30 novembre 2025
Analyse expérimentale de la complexité

Ce sujet de tp est accompagné d'un fichier mpsi_itc_td_12_code_fourni.py.

Exercice 1 (Méthodologie)

Rien ne ressemble plus à une courbe de la forme $y = k_1 \times x^{d_1}$ qu'une autre courbe de la forme $y = k_2 \times x^{d_2}$, et beaucoup de courbes ressemblent à des courbes polynomiales sans être polynomiales.
Si on a des données qu'on suspecte suivre une courbe polynomiale de la forme $y = k \times x^n$, quel graphique tracer pour le vérifier visuellement facilement ? Quelle méthode graphique peut-on alors utiliser pour récupérer le degré d ?

1. Dans le code fourni, il y a deux tableaux `x_exo1` et `y_exo1`. En utilisant une méthode graphique, dire si ces coordonnées décrivent la courbe d'un polynôme, et si oui, quelle est son plus grand degré.
2. Même question mais cette fois-ci en s'aidant de la fonction `regression_lineaire` fournie, qui prend en argument une liste d'abscisses et un liste d'ordonnées, et qui donne la pente de la **droite** ressemblant le plus à la courbe décrite par les points en entrée.

Quand on vérifie expérimentalement la complexité temporelle d'une fonction, un des obstacles qu'on rencontre est le bruit de mesure. L'une des source de bruit de mesure est qu'un programme ne s'exécute jamais seul sur un ordinateur. Ainsi, quand on mesure le temps mis par un programme pour s'exécuter, on est en général également en train d'inclure dans la mesure le temps que le programme a passé à être interrompu par l'exécution d'autres programmes. Pour pallier à ce bruit de mesure, plusieurs solutions :

— Pour une même taille d'entrée, prendre plusieurs mesures et en faire la moyenne.
— Choisir des entrées sur lesquelles le programme ne s'exécute pas trop vite, pour diminuer la proportion de bruit vis-à-vis du temps qu'on veut vraiment mesurer.

Exercice 2 (Complexité du tri sélection)

Le but de cet exercice est de vérifier expérimentalement les résultats de complexité asymptotique vus en cours vendredi dernier. Pour cela on va s'intéresser au temps d'exécution des fonctions `indice_minimum` et `tri_selection` en fonction de la taille de la liste qu'elles reçoivent en entrée.

1. En s'aidant de la fonction `taille_dixieme_seconde` fournie, choisissez sur quelles tailles de listes en entrée vous voulez réaliser vos mesures pour chacune des deux fonctions étudiées. Les buts étant :
   — D'avoir au moins 30 points de mesure
   — De pouvoir collecter toutes les mesures dans un temps qui vous semble relativement rapide (oui, c'est subjectif).
   — D'avoir des mesures pour lesquelles les fonctions étudiées ne s'exécutent pas trop vite, pour éviter d'avoir des mesures trop bruitées.
2. En vous aidant de la fonction `temps_exec_moyen` fournie, tracer des courbes du temps d'exécution des fonctions étudiées, pour différentes valeurs de `nb_echantillons`. À partir de quelle valeur de `nb_echantillons` vos mesures vous semblent ne plus être trop bruitées ?
3. En suivant la méthodologie de l'exercice 1, et les choix des questions précédentes, déterminer expérimentalement la complexité des fonctions `indice_minimum` et `tri_selection`.

Exercice 3 (Bonus : mots mélés, partie 2)

1. Écrire une fonction `recherche_motif` qui prend en entrée une liste de caractères `motif` de taille k et une liste de caractères `texte` et qui renvoie le premier indice i tel que `texte[i:i+k]` soit égal à `motif`. Si un tel i n'existe pas, renvoyer `None`.
2. Quelle est la complexité de votre fonction ?
3. Vérifier expérimentalement cette complexité.

Exercice 4 (Bonus : mots mélés, partie 3)

1. Écrire une fonction `recherche` qui prend en arguments une grille de mots mélés `grille`, une liste de mots à trouver `mots` et qui renvoie la liste des paires (*indice dans la grille*, *direction*) auxquels on peut trouver les mots cherchés. Une erreur sera levée si l'un des mots n'est pas dans la grille.
2. Quelle est la complexité de votre fonction ?
3. Vérifier expérimentalement cette complexité.

Mémento Matplotlib

`import matplotlib.pyplot as plt`	Importe le module, préfixe `plt`.
`plt.plot(X, Y, 'ro', label='')`	`X` et `Y` sont des tableaux de même taille contenant la liste des coordonnées x et y. `'ro'` → red, symbole ∘ `'b+'` → blue, symbole +
`plt.legend()`	affiche les différents `label` définis via `plt.plot`
`plt.arrow(X,Y,Vx,Vy,head\_width=0.2, head\_length=0.5,color="red")`	Trace le vecteur de coordonnées (v_x, v_y) à partir du point de coordonnées (x, y) avec les mêmes conventions que `plt.plot`.
`plt.axis('equal')` ou si bug, `plt.gca().set\_aspect('equal')`	Permet d'avoir la même représentation de l'unité en x et en y
`plt.xlim(xmin, xmax)` `plt.ylim(ymin, ymax)`	Fixe les limites souhaitées en x et en y
`plt.text(x,y,"texte", fontsize=12)`	Ecrit `texte` au point de coordonnées (`x`, `y`)
`plt.xlabel("x (unité)")` `plt.ylabel("y (unité)")`	Pour écrire les grandeurs sur les axes
`plt.grid(linestyle='-.')`	Trace la grille en arrière plan
`plt.savefig('image.png')`	Sauvegarde le graphique dans un fichier png
`plt.show()`	Pour afficher la figure. A mettre à la fin du programme.
`plt.figure()`	Pour avoir plusieurs fenêtres de graphique, à mettre au début de chaque bloc correspondant à une fenêtre

Mémento Numpy

Les fonctions fournies par le module sont **vectorielles** !

`import numpy as np`	Importe le module, préfixe `np`.
`np.linspace(start, stop, n)`	renvoie un tableau de `n` valeurs régulièrement réparties sur l'intervalle [`start`,`stop`]
`np.arange(start, stop, step)`	renvoie un tableau de valeurs comprises en `start` inclu et `stop` **exclu** par pas de `step`
`np.zeros(N)`	renvoie un tableau de `N` zéros
`np.empty(N)`	renvoie un tableau de `N` cases non initialisées

Soit `x` un tableau.

Tracé de la fonction `f` Méthode 1 – fonction `f` vectorielle	`plt.plot(x, f(x))`
Tracé de la fonction `f` Méthode 2 – fonction `f` **non** vectorielle	`y = [f(u) for u in x]` `plt.plot(x, y)`
Tracé de la fonction `f` Méthode 3 – fonction `f` **non** vectorielle	`y = list( map(f, x) )` `plt.plot(x, y)`
Tracé de la fonction `f` Méthode 4 – fonction `f` **non** vectorielle	`g = np.vectorize(f)` `plt.plot(x, g(x))`