Programmation en temps limité

2nd semestre 2023/2024

1 Évaluation

Un sujet de programmation de 4h vous sera proposé en fin de semestre. L'épreuve aura lieu le 6 mai à 13h30. Des exercices d'entraînement vous seront proposés tout au long du semestre.

L'épreuve sera simultanée pour tous et sans connexion internet. Le sujet sera constitué d'une série de questions qui pourront largement se traiter de manière indépendante. Il vous sera demandé d'implémenter des algorithmes et manipulations de structures de données simples, pour produire les résultats demandés par le sujet. Vos programmes devront donc être exécutables et fonctionnels ! Vous rendrez également votre code en fin d'épreuve en complément de vos réponses.

Le langage imposé est python. Vous constaterez au travers des exercices d'entraînement proposés que tout est réalisable avec des notions de base.

Durant le semestre, vous pourrez poser vos questions indifféremment à Mathieu Chapelle (mathieu.chapelle@univ-orleans.fr) ou Martin Delacourt (martin.delacourt@univ-orleans.fr).

2 Pour s'entraîner

2.1 À propos de nombres :

- pour un entier n et un entier k, retourner la chaîne de caractères représentant n en base k
- pour un entier k et une chaîne de caractères s, tester si la chaîne s ne contient que des chiffres entre 0 et $k-1$
- pour un entier k et une chaîne de caractères s, retourner (si la chaîne est valide) la valeur de l'entier représenté par s en base k
- pour un entier n, tester si n est premier
- pour deux entiers m et n, tester si n est une puissance de m

2.2 À propos de listes :

- pour une liste donnée en paramètre, retourner la somme des nombres contenus dans la liste
- pour une liste l et un entier k donnés en paramètre, retourner une liste contenant tous les éléments de l multipliés par k
- pour une liste de chaînes de caractères, retourner une liste contenant en position i la longueur de la chaîne en position i
- pour une liste l, retourner une liste contenant chaque élément de l une seule fois (en enlevant les répétitions)
- pour une liste l, retourner l'élément le plus présent dans la liste et son nombre d'occurrences (en cas d'égalité, retourner le premier à apparaître dans l)
- pour une liste l, dire si la liste contient un doublon
- pour deux paramètres k et n, générer une liste de taille k contenant des entiers tirés aléatoirement avec `random.randint` entre 1 et n, puis retourner 1 si cette liste contient au moins un doublon, 0 sinon

- pour $k = 23$ et $n = 365$, faire 1000 fois l'expérience précédente et calculer la moyenne des résultats (0 ou 1). Que constate-t-on en faisant varier k ?

2.3 À propos de dictionnaires :

- pour un dictionnaire d, retourner une liste contenant les couples `(clé,valeur)` présents dans d
- pour un dictionnaire d contenant des couples `(joueur, score)` et un seuil k, retourner la liste des joueurs dont le score est inférieur à k
- pour un dictionnaire d contenant des couples `(joueur, liste de scores)`, modifier d pour qu'il contienne la somme des scores au lieu de la liste, la fonction ne retourne rien
- pour une liste d'étudiants, retourner un dictionnaire d contenant les couples `(étudiant, note)` où les étudiants sont tous ceux de la liste et les notes sont générées aléatoirement avec `random.randint`

2.4 À propos de graphes :

On suppose qu'un graphe est donné par une liste de sommets et une liste d'arêtes représentées par des couples de sommets. Si le graphe est pondéré, les arêtes sont des triplets contenant aussi le poids.

- pour un graphe G, retourner `True` si au moins un sommet n'est relié à aucun autre, `False` sinon
- pour un graphe G, retourner le degré maximal (nombre de voisins d'un sommet)
- pour un graphe G et un sommet x, retourner la composante connexe de x (la liste des sommets y tels qu'un chemin entre x et y existe)
- pour un graphe G, tester la connectivité de G (il existe un chemin de n'importe quel sommet vers n'importe quel autre)
- pour un graphe pondéré G et un sommet x, retourner la liste des couples `(sommet,distance)` pour les sommets dans la composante connexe de x (algorithme de Dijkstra)

2.5 À propos d'automates finis :

- proposer une structure de données représentant un automate fini
- pour un automate fini A et un mot u sur son alphabet, tester si u est dans le langage de A
- pour un automate fini A, tester si le langage de A est vide
- implémenter l'algorithme de minimisation d'un automate fini