

Consignes

- L'énoncé comporte trois exercices indépendants.
- Les documents, calculatrices et ordinateurs ne sont pas autorisés.
- La présentation, la qualité de la rédaction, la clarté et la précision des raisonnements entreront pour une part importante dans l'appréciation des copies. Les résultats doivent être mis en valeur et les pages (ou les feuilles) doivent être numérotées.
- Les programmes doivent être rédigés en Python et les requêtes en SQL.
Il n'est pas nécessaire d'écrire la spécification des fonctions mais il peut être parfois judicieux de commenter son code afin d'en faciliter la compréhension.

Exercice 1 - base de données rugbyistique

On rappelle tout d'abord que pour marquer des points dans un match de rugby, on peut réaliser un *drop* ou une *pénalité* (trois points dans les deux cas) ou un *essai* (cinq points), chaque essai pouvant être suivi d'une tentative de *transformation* (deux points de plus en cas de réussite). Ainsi, un score total de 23 est par exemple possible avec quatre essais non transformés et un drop ($4 \times (5 + 0) + 1 \times 3$), mais aussi deux essais transformés et trois drops ($2 \times (5 + 2) + 3 \times 3$), trois essais dont un transformé et deux drops ($1 \times (5 + 2) + 2 \times (5 + 0) + 2 \times 3$), et d'autres combinaisons.

1. Écrire une fonction qui prend en argument un entier positif, vu comme un score, et qui renvoie le nombre de façons de réaliser ce score au rugby. L'ordre dans lequel les points sont marqués n'est pas important.
2. Considérons le dictionnaire suivant, appelé `joueurs`, qui est une variable globale : les clés sont des équipes, et les valeurs sont des listes de noms de joueurs (il n'y a aucun doublon sur l'ensemble des listes).
On a par exemple dans `joueurs["France"]` les éléments "Melvyn Jaminet", "Antoine Dupont", etc. En particulier, en tant que variable globale, ce dictionnaire n'a jamais besoin d'être mis en argument d'une fonction.
Écrire une fonction qui prend en argument une chaîne de caractères dont on garantit qu'elle est présente quelque part dans `joueurs` et qui renvoie l'équipe dans laquelle le joueur associé est. Préciser la complexité de cette fonction.
3. Écrire une fonction sans argument qui construit un dictionnaire indexé par les noms de joueurs et dont les valeurs sont les équipes associées.
4. Considérons à présent un compte-rendu de match. La gestion chaotique fait que tous les événements ont été mélangés sans séparer les équipes. Ainsi, un glorieux essai français se situe côte à côte avec une misérable pénalité anglaise. Nous avons donc des comptes-rendus sous la forme de listes dont les éléments sont le joueur ayant marqué, ce qu'il a marqué ("essai", "transformation", "drop" ou "penalite") et le temps de jeu associé, heureusement dans l'ordre chronologique. Un exemple d'élément sera alors ["Anthony Jelonch", "essai", 9].
Écrire une fonction prenant en argument un compte-rendu et déterminant s'il est cohérent : une transformation ne peut avoir lieu que directement après un essai ; le joueur doit être dans la même équipe ; enfin il ne peut y avoir que (au plus) deux équipes concernées pour un compte-rendu.

La suite de l'épreuve se fera en SQL, avec la base de données suivante :

- table `Equipe`, avec pour attributs `equipe_id` (entier) et `equipe_nom` (chaîne de caractères) ;
- table `Joueur`, avec pour attributs `joueur_id` (entier), `joueur_nom` (chaîne de caractères) et `equipe_id` (entier) ;
- table `Match`, avec pour attributs `match_id` (entier), `match_date` (chaîne de caractères), `match_equipe1` (entier) et `match_equipe2` (entier) ;
- table `Deroulement`, avec pour attributs `match_id` (entier), `joueur_id` (entier), `match_evenement` (chaîne de caractères) et `match_moment` (entier) ;
- table `Points`, avec pour attributs `match_evenement` (chaîne de caractères) et `points_points` (entier).

Explications :

- les attributs nommés `id` préfixés par le nom de la table où ils figurent sont des clés primaires (dans les autres cas ce sont des clés étrangères y faisant référence) ;
- la plupart des attributs correspond aux informations utilisées précédemment en Python, notamment `match_evenement` ;
- les attributs `match_equipe1` (entier) et `match_equipe2` sont aussi des clés étrangères et correspondent à `equipe_id` ;
- la date d'un match est donnée sous la forme AAAA-MM-JJ ;
- la table `Points` sert à faciliter le travail des dernières requêtes tout en économisant de la mémoire en ne mettant pas l'information dans chaque enregistrement de la table `Deroulement`.

5. Écrire une requête qui affiche le nom de tous les joueurs de l'équipe de France sans autre attribut.
6. Écrire une requête qui affiche le moment le plus tôt et le moment le plus tardif où des points ont été marqués sur l'ensemble des matchs.
7. Écrire une requête qui affiche le nom de tous les joueurs de l'équipe de France ayant marqué au moins un essai.
8. Écrire une requête qui affiche le nom des deux joueurs de l'équipe de France ayant marqué le plus d'essais.
9. Écrire une requête qui affiche le résultat du match France-Angleterre sous la forme d'une table contenant deux enregistrements et ayant deux attributs : `equipe_nom` et l'attribut que l'on nommera `score`. On admettra que les deux équipes ont marqué pour ne pas avoir à compliquer la requête.

Exercice 2 - sur l'algorithme des k plus proches voisins

On redonne ci-dessous une version de l'algorithme des k plus proches voisins dans un cas assez général où la fonction de distance sur \mathbb{R}^d est en argument.

```

1 def kNN(les_points, point_sup, k, dist):
2     """ entrées
3         les_points : liste des couples de la forme (point, étiquette)
4             où point est une liste de flottants de longueur d
5             et étiquette est une chaîne de caractère
6         point_sup : point (liste de flottants de longueur d)
7         k : entier
8         dist : fonction distance entre deux éléments (points) de  $\mathbb{R}^d$ 
9
10    sortie
11    rep : chaîne de caractère
12           correspondant à l'étiquette de point_sup
13
14    """
15    distances = [(dist(P[0], point_sup), P[1]) for P in les_points]
16    pass
17    etiquettes = dict()
18    for p, etiq in distances[:k]:
19        if etiq in etiquettes:
20            etiquettes[etiq] += 1
21        else:
22            etiquettes[etiq] = 1
23    rep = etiq
24    for e in etiquettes:
25        if etiquettes[e] > etiquettes[rep]:
26            rep = e
27    return rep

```

1. À la ligne 14, l'instruction `pass` a remplacé une ligne de code indispensable. Quel était l'effet de cette ligne? (on ne demande pas de la programmer).
2. Expliquer le rôle des lignes 21 à 24. Préciser ce que vaut précisément `etiq` à la ligne 21 (par rapport aux arguments de la fonction).
3. Écrire une fonction `distance` de calcul de la distance euclidienne entre deux vecteurs de \mathbb{R}^d qui puisse être compatible avec la fonction `kNN`.
4. On souhaite modifier la fonction de sorte que, plus un des k plus proches voisins est proche, plus il va contribuer pour le choix de son étiquette.

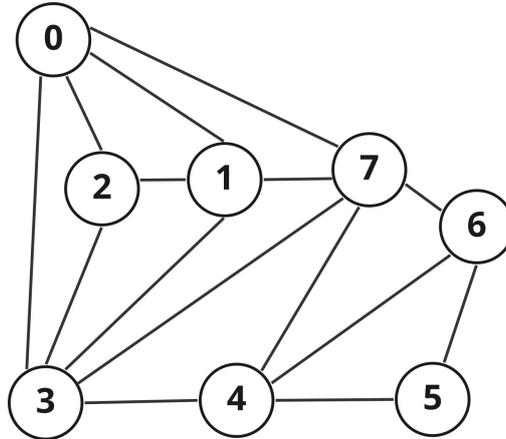
La formule suggérée est de le pondérer par l'inverse de la distance au point supplémentaire (en traitant le cas particulier d'une distance nulle : dans ce cas l'étiquette est forcément retenue).

On admettra que les points fournis sont distincts deux à deux.

Écrire une nouvelle version de la fonction `kNN` (on pourra se contenter de ne rédiger que les modifications à apporter à la fonction de l'énoncé).

Exercice 3 - coloration d'un graphe

On considère le graphe non orienté suivant :



1. Représenter le graphe de l'énoncé à l'aide d'un dictionnaire G dont les clés sont les noms (entiers de 0 à 7) des sommets et les valeurs associées sont les listes des voisins. Par exemple :

```
>>> G[4]
[3, 5, 6, 7]
```

2. Écrire une fonction voisins(gr, i, j) où gr est un dictionnaire représentant un graphe comme ci-dessus, i et j sont des numéros de sommets différents, et qui renvoie un booléen indiquant si ces deux sommets sont voisins ou non. Par exemple :

```
>>> voisins(G, 0, 1)
True
>>> voisins(G, 0, 5)
False
```

3. Écrire une fonction aretes(gr) où gr est un dictionnaire représentant un graphe comme ci-dessus et qui renvoie le nombre d'arêtes de ce graphe. Par exemple :

```
>>> aretes(G)
15
```

Le but de ce qui suit est de colorer ce graphe (c'est-à-dire d'attribuer une couleur pour chaque sommet) de sorte que deux sommets adjacents n'aient pas la même couleur et en cherchant à minimiser le nombre de couleurs. Les couleurs seront représentées par des numéros : 0, 1, etc.

4. La fonction colorerSommet(gr, i, couleurs) ci-dessous prend en arguments un dictionnaire gr représentant un graphe (comme ci-dessus), un entier i correspondant à un sommet et une liste couleurs (dont la longueur est le nombre de sommets) et qui indique le numéro de la couleur attribuée au sommet ou -1 si le sommet n'est pas coloré.

Cette fonction ne renvoie rien mais modifie la liste couleurs pour attribuer le plus petit numéro possible au sommet i. Par exemple :

```
>>> C = [0, 1, 2, -1, -1, -1, -1, -1]
>>> colorerSommet(G, 3, C)
>>> C
[0, 1, 2, 3, -1, -1, -1, -1]
>>> colorerSommet(G, 4, C)
>>> C
[0, 1, 2, 3, 0, -1, -1, -1]
```

Compléter les lignes 17, 18, 21 et 22 en remplaçant les tirets :

```
1 def colorerSommet(gr, i, couleurs):
2     """ entrée
3         gr dictionnaire représentant un graphe
4         i (int) nom d'un sommet
5         couleurs : liste d'entiers telle que couleurs[j]
6             soit la couleur attribuée au sommet j;
7             couleurs[j] = -1 si j n'est pas encore coloré
8
9         La fonction modifie couleurs en attribuant la couleur la plus
10            petite possible (plus petit numéro) au sommet i
11
12     pas de sortie
13     """
14     n = len(couleurs)
15     col = [] # liste des voisins de i déjà colorés
16     for j in gr[i]:
17         if -----
18             -----
19     # on recherche maintenant la plus petite couleur possible
20     new = 0
21     while -----:
22         -----
23     couleurs[i] = new
```

5. Écrire une fonction `colorer` (`gr`) d'argument un dictionnaire `gr` représentant un graphe et qui renvoie la liste des couleurs attribuées aux sommets. On doit donc avoir :

```
>>> colorer(G)
[0, 1, 2, 3, 0, 1, 2, 4]
```