Cours d'informatique TSI2

1 Dictionnaires et algorithme pour l'intelligence artificielle

1.1 Les dictionnaires. Définition et premiers exemples

Un dictionnaire est une collection de paires `clé : valeur`. Contrairement aux listes, il n'y a aucune notion d'ordre dans un dictionnaire.

Un élément du dictionnaire `dico` associé à la clé `cle` est obtenu par la commande :

```
e=dico[cle]
```

La clé associée à un élément d'un dictionnaire peut être un entier ou une chaîne de caractères mais ne peut pas être une liste. Cette contrainte permet d'utiliser des tables de **hachage** par accélérer l'accès aux valeurs d'un dictionnaire (voir plus loin...)

Exemple 1. On définit un dictionnaire dont les clés sont `'a'`, `'b'` et `'ab'`.

```
>>> dico1 = {'a':1,'b':2,'ab':4}
>>> dico1['a']
1
>>> dico1['ab']
4
>>> len(dico1)
3
```

On crée un dictionnaire vide `D` par la commande `D=dict()` ou tout simplement par la commande `D={}`. Pour ajouter un élément à un dictionnaire, on donne la clé et la valeur par une simple affectation.

Exemple 2.

```
>>> dico2=dict()
>>> dico2['un']='one'
>>> dico2['deux']='two'
>>> print(dico2)
 {'un':'one','deux':'two'}
```

On supprime un élément avec l'instruction **del** :

```
del(D[clé])
```

Comme pour les listes ou les chaînes de caractère, on teste l'appartenance d'une clé à un dictionnaire grâce à l'opérateur **in**

> **Exemple 3.** On reprend l'exemple précédent :
>
> ```
> >>> 'un' in dico2
> True
> >>> 'two' in dico2
> False
> ```

Quelques méthodes pour travailler avec des dictionnaires :

- `clear()` supprime tous les éléments.
- `values()` renvoie une séquence des valeurs.
- `keys()` renvoie une séquence des clés.
- `items()` renvoie une séquence de tuples sour la forme `(clé, valeur)`
- `pop(clé)` supprime l'élément de la clé en argument et renvoie sa valeur.

> **Exemple 4.**
>
> ```
> >>> D={'a':1,'b':2}
> >>> D.values()
> dict_values([1, 2])
> >>> D.items()
> dict_items([('a', 1), ('b', 2)])
> >>> D.clear()
> >>> D
> {}
> ```

Les séquences obtenues par ces méthodes peuvent être converties en liste avec l'instruction `list`.

Exercice 1. Écrire une fonction qui prend en argument un dictionnaire et qui renvoie une liste qui contient les clés du dictionnaire et une liste qui contient ses valeurs.

Proposer une fonction avec les méthodes `values()` et `keys()` et une fonction sans les utiliser.

Exercice 2. Soit `d` un dictionnaire contenant un stock d'articles. La clé est le nom de l'article, la valeur est une liste contenant le prix et la quantité de l'article.

Écrire une fonction qui calcule la valeur du stock.

Exercice 3. Écrire une fonction qui prend en argument un dictionnaire dont toutes les valeurs sont distinctes et qui renvoie le dictionnaire inversé (les clés de l'un sont les valeurs de l'autre).

Exercice 4. Écrire une fonction `frequence(ch)` qui prend en paramètre une chaîne de caractères `ch` et qui renvoie un dictionnaire dont les clés sont les caractères de la chaîne `ch` et les valeurs sont les fréquences de la clé dans la chaîne `ch`.

Exercice 5. Écrire une fonction `mots(texte)` qui recherche tous les mots d'un texte et qui construit un dictionnaire. Les clés de ce dictionnaire sont les mots du texte. À chaque mot, le dictionnaire associe le nombre de fois qu'il apparaît.

On pourra utiliser la méthode `split()` qui découpe une chaîne en liste. On peut préciser un séparateur, par défaut le séparateur est l'espace.

Exercice 6. On considère Ω un ensemble dont les éléments sont stockés dans la liste `omega` et A une partie de Ω.
On dit qu'un dictionnaire représente une partie de Ω si l'ensemble de ses clés correspond à Ω et ses valeurs sont des booléens.

1. Écrire une fonction `dico(A,omega)` prenant en argument la liste des éléments de A et qui renvoie un dictionnaire dont les clés sont les éléments de Ω et les valeurs `True` ou `False` suivant que la clé appartient ou non à A.
2. Écrire une fonction `liste(D)` prenant en argument un dictionnaire `D` représentant une partie de Ω et qui renvoie la liste des éléments de Ω et la liste des éléments de la partie de Ω correspondant à `D`.
3. Écrire des fonctions `union(D,E)` et `intersection(D,E)` dont les arguments sont des dictionnaires représentant des parties A et B de Ω et qui renvoient respectivement les dictionnaires correspondant à $A \cup B$ et $A \cap B$.

1.2 Implémentation d'un dictionnaire. Principe du hachage

Pour implémenter un dictionnaire, on crée une liste de taille fixe N puis, à chaque clé, on associe un indice i compris entre 0 et $N-1$. On stocke le tuple `(clé, valeur)` à l'indice i.

La fonction qui, à chaque clé, fait correspondre un indice s'appelle **fonction de hachage**.

Une fonction de hachage n'est pas injective en général. Autrement dit, un même indice peut très bien correspondre à plusieurs clés. On dit alors qu'il y a collision.

Prenons l'exemple où les clés sont des chaînes de caractères. La fonction qui renvoie la longueur est une fonction de hachage. Dans ce cas, des clés de même longueur provoquent des collisions.

Les collisions sont toujours possibles mais doivent être le plus possible évitées. Pour cela, on doit choisir un « N » grand (en tout cas nettement plus grand que la longueur du dictionnaire) et une « bonne » fonction de hachage.

Lorsqu'il y a collision, on peut par exemple choisir la première case du tableau disponible après l'indice obtenu par la fonction de hachage.

Exercice 7. On considère la fonction de hachage suivante. Pour chaque clé de longeur n, on note $a_0, \ldots, a_{n-1}$ le code ASCII de ses lettres de droite à gauche et on considère le polynôme $P(X) = \sum_{k=0}^{n-1} a_k X^k$.

Écrire la fonction de hachage `hachage(cle,N)` qui calcule $P(256)$ modulo N.

On rappelle que :
- pour calculer le reste de la division euclidienne de a par b, on utilise la commande `a %b` (pour le quotient : `a//b`).
- la fonction `ord()` convertit tout caractère alphanumérique en son code ASCII qui est un nombre compris entre 0 et 255.

Exercice 8. Implémenter le dictionnaire dont les clés sont les mois de l'année en anglais et les valeurs leur traduction en français en prenant la fonction de hachage de l'exercice précédent (choisir N convenablement).
Tester en écrivant une fonction prenant en argument une clé et qui renvoie la valeur correspondante.

Exercice 9. Implémenter un dictionnaire, sans utiliser de table de hachage à l'aide d'une liste dont les éléments sont les tuples `(clé,valeur)`.

Quels sont les avantages et inconvénients de cette implémentation ?

1.3 Algorithme des k plus proches voisins

On dispose d'objets classés en n catégories. À chaque objet est associé un vecteur de $\mathbb{R}^d$.

Le problème est de classifier un nouvel objet connaissant uniquement le vecteur qui lui est associé.

Ces données peuvent être stockées sous forme de dictionnaire. Les clés peuvent être des numéros et les valeurs une liste dont le premier élément est la catégorie de l'objet (sous forme de chaîne de caractères le plus souvent) et le deuxième le vecteur qui lui est associé.

On peut également stocker ces données uniquement à l'aide de listes.

L'idée est de considérer les k plus proches voisins du nouvel objet (dont on ne connaît pas la catégorie) où k est un entier qu'il faudra choisir judicieusement (s'il n'y a que deux catégories, on choisit k impair !).

On peut alors penser que cet objet est de la catégorie majoritaire parmi ces plus proches voisins.

Il faut donc calculer la distance entre l'objet étudié et n'importe qu'elle autre objet. En général, on utilise pour cela la distance euclidienne de $\mathbb{R}^d$:

$$d((x_1,\ldots,x_d),(y_1,\ldots,y_d)) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}.$$

On classe les distances obtenue par ordre décroissant et on s'intéresse aux k premières.

Exemple 5. On dispose d'objets rouges (R) et bleus (B) :

Objet	1	2	3	4	5	6	7	8	9	10	11	12
couleur	R	B	B	R	R	R	B	B	B	B	R	R
x	0	3	4	2	4	5	7	5	6	7	2	1
y	0	7	1	7	4	0	5	6	2	6	3	2

On stocke ces données dans une variable globale de type liste `L`, chaque élément de la liste est lui-même une liste contenant dans l'ordre : x, y et la chaîne de caractère `'red'` ou `'blue'` suivant la couleur de l'objet.

Ainsi `L[0]` contient la liste `[0,0,'red']`.

Exercice 10. Écrire une fonction sans paramètre permettant d'afficher un graphique représentant les objets de l'exemple.

Pour afficher un point de coordonnées (x, y) avec une croix rouge, on peut écrire :

```
import matplotlib.pyplot as plt
plt.plot(x,y,'x',color='red')
```

Exercice 11. Écrire une fonction distance prenant en paramètre deux vecteurs de $\mathbb{R}^d$ et qui renvoie la distance euclidienne entre ces deux vecteurs.

Exercice 12. Écrire une fonction prenant en paramètre un couple de coordonnées (x, y) et qui renvoie une liste `D` contenant la distance de (x, y) à chaque objet de la liste `L` et la couleur de l'objet.

Plus précisément `D[i][0]` est la distance entre l'objet `i` et le point (x, y), `D[i][1]` est la couleur de l'objet `i`.

Exercice 13. Écrire une fonction qui trie, par ordre croissant des distances, la liste `D` obtenue dans l'exercice précédent.

Exercice 14. Écrire une fonction qui renvoie la liste des couleurs des k plus proches voisins d'un point (x, y). Les paramètres de la fonction sont : les coordonnées (x, y) et un entier k.

Exercice 15. On considère une liste `C` de chaînes de caractères. Écrire une fonction `comptage` de paramètre `C` qui renvoie un dictionnaire dont les clés sont les éléments de `C` et les valeurs le nombre de fois que la clé apparaît dans la liste `C`.

Exercice 16. Écrire une fonction qui prend en argument un point de coordonnées (x, y) et un entier k et qui affiche ce point de la couleur majoritaire de ses k plus proches voisins.

Tester la fonction sur tous les points dont les coordonnées sont compris entre 0 et 10.

> **Exemple 6.** Les iris de Fisher sont des données collectées par Edgar Enderson, et proposées en 1933 par le statisticien Ronald Aylmer Fisher comme données de référence pour l'analyse discriminante et la classification.
>
> Il s'agit de reconnaître le type d'iris (setosa, virginica, et versicolor) à partir seulement de la longueur et de la largeur de ses sépales et pétales. Le fichier contient 50 fleurs de chaque type.
>
> Voici les cinq premières lignes :
>
> ```
> 5.1,3.5,1.4,0.2, Iris-setosa
> 4.9,3.0,1.4,0.2, Iris-setosa
> 4.7,3.2,1.3,0.2, Iris-setosa
> 4.6,3.1,1.5,0.2, Iris-setosa
> 5.0,3.6,1.4,0.2, Iris-setosa
> ```

Chaque ligne indique, dans l'ordre, la longueur et la largeurs des sépales puis la longueur et la largeur des pétales.

Exercice 17. Récupérer le fichier de données des iris de Fisher et donner une représentation graphique de ces données en ne prenant en compte que les sépales.

Exercice 18. Écrire une fonction qui donne le type d'un iris dont on connaît les caractéristiques (longueur, largeur des sépales et pétales) grâce à l'algorithme des k plus proches voisins.

1.4 Algorithme des k-moyennes

Le partitionnement en k-moyennes est une méthode de partitionnement d'objets en k ≪ clusters ≫ (groupe en français) de façon à faire des regroupements d'objets ≪ qui se ressemblent ≫ ou qui potentiellement appartiennent à la même catégorie.

On suppose que chaque objet est caractérisé par un vecteur de $\mathbb{R}^d$ comme dans l'algorithme des k plus proches voisins. On a un certain nombre n d'objets que l'on veut répartir en k clusters. Dans l'exemple qui suit $d = 2$, $n = 12$ et on prendra $k = 3$.

Étant donné une répartition des objets en clusters $S_1, S_2, \dots S_k$, on considère les points moyens $\mu_1, \mu_2, \dots, \mu_k$ de ces clusters et on s'intéresse à la quantité, appelée parfois ≪ fonction de coût ≫ :

$$C = \sum_{i=1}^{k} \sum_{x \in S_i} d(x, \mu_i)^2$$

où $d(x, y)$ désigne la distance entre les vecteurs x et y de $\mathbb{R}^d$ (voir définition dans le paragraphe précédent).

L'objectif est de trouver le partionnement qui rende cette quantité C minimale. Malheureusement, lorsque le nombre de données est trop grand, il n'est pas possible de chercher tous les partionnements pour trouver le meilleur. Le temps de calcul serait trop long !

Exercice 19. Donner l'ordre de grandeur du nombre de partionnements de n objets en 3 clusters.

On obtient en général un partionnement ≪ acceptable ≫ en procédant de la façon suivante.

On commence par choisir un ensemble E_0 de k points $\mu_1, \mu_2, \ldots, \mu_k$ pris, en général, au hasard (ou arbitrairement) dans le jeu de données et on fait un premier partionnement en mettant dans le cluster i, les données les plus proches de μ_i pour tout i compris entre 1 et k. On calcule les points moyens de chaque cluster, ce qui donne un ensemble de k nouveaux points donnant lieu à un nouveau partionnement du jeu de données en k clusters avec une fonction coût inférieur. À chaque itération du processus, la fonction coût diminue. Donc, après un certain d'itérations, on peut espérer avoir un bon partionnement du jeu de données en k clusters.

Comme mentionné plus haut, cette algorithme ne fournit pas obligatoirement le meilleur partionnement en k clusters mais il donne souvent de bons résultats. Le résultat peut d'ailleurs dépendre du choix initial des points $\mu_1, \ldots, \mu_k$.

Exemple 7. On dispose de 12 objets que l'on veut répartir en trois catégories.

Objet	0	1	2	3	4	5	6	7	8	9	10	11
x	0	1	3	3	4	10	0	10	8	6	1	9
y	0	1	4	6	2	6	2	7	4	8	0	6

On dit qu'une liste `D` d'entiers définit une partition de la liste `L` en k lorsqu'elle est de même longueur que `L` et contient des entiers compris entre 0 à $k-1$.

Par exemple, la liste `D=[0,1,0,0,1,2,2,2,1,1,0,2]` définit une partition de `L` en trois clusters. Le cluster 0 contient les objets de coordonnées $(0,0)$, $(3,4)$, $(3,6)$, $(1,0)$.

Dans toute la suite, les données sont stockées dans une liste globale `L` :

```
L=[[0, 0], [1, 1], [3, 4], [3, 6], [4, 2], [10, 6],[0, 2], [10, 7], [8, 4],
[6, 8], [1, 0], [9, 6]]
```

Exercice 20. Écrire une fonction `afficheclusters` qui prend en argument une partition `D` de `L` et qui affiche les objets de `L` d'un même cluster de la même couleur.

Exercice 21. Écrire une fonction `pointsmoyens` de paramètre une partition `D` de `L`, et qui renvoie pour chaque cluster les coordonnées du point moyen.

On utilisera, dans cette fonction, un dictionnaire dont les clés sont les clusters. Les valeurs sont des listes contenant le nombre de points du cluster, la somme des abscisses et la somme des ordonnées des points du cluster.

À chaque cluster d'une partition, on associe un point. Si la partition est composée de k clusters, on dispose d'une liste `C` de k points (représentés par une liste de deux réels) et on associe le point `C[0]` au cluster 0, le point `C[1]` au cluster 1, etc.

Les points `C[0]`, ..., `C[k-1]` doivent être distincts.

Dans l'exercice ci-dessous, on pourra prendre la liste des points $(0, 0)$, $(0, 1)$, $(0, 2)$.

Exercice 22. Écrire une fonction `affichage` qui prend en argument les listes `C` et `D` et qui relie chaque point d'un même cluster au point qui lui est associé.

On utilisera la fonction `plot` du module `matplotlib.pyplot`. Par exemple, pour relier les points de coordonnées (x_0, y_0), (x_1, y_1), on écrit :

```
import matplotlib.pyplot as plt
plt.plot([x0,x1],[y0,y1])
```

Exercice 23. Écrire une fonction `repartition` qui prend en argument une liste `C` de points et qui cherche pour chaque objet de `L` le numéro du (ou d'un) point de `C` le plus proche. La fonction renvoie la liste `D` de ces numéros.

Par exemple si `D[2]` vaut 0 cela signifie que le point numéro 0 de `C` est le point de `C` le plus proche d'objet dont les coordonnées sont $(3, 4)$.

Tester la fonction en prenant les objets 0, 1 et 3 pour former la liste `C`.

On remarque que la liste `D` réalise une partition de `L`. Naturellement, on va considérer la liste `C` comme la liste des points associés à chaque cluster défini par `D`.

Exercice 24. Écrire une fonction `cout` de paramètres la liste des points `C` et la partition `D` est qui renvoie la valeur de la fonction coût.

Exercice 25. Écrire une fonction `kmoyennes` qui prend en paramètre une liste I de k entiers compris entre 0 et $n - 1$ (où n est le nombre d'objets de `L`) et un entier $N > 0$, et qui réalise N itérations de l'algorithme des k moyennes en prenant comme points de départ les points de `L` dont les indices sont indiqués dans la liste donnée en paramètre.

Pour créer une nouvelle figure, on utilise la commande `plt.figure()`. Pour afficher plusieurs graphiques sur la même figure, on utilise la commande `plt.subplot(n,p,k)` où n, p, k sont des entiers.

Exercice 26. Afficher l'évolution de la fonction coût pour mettre en évidence qu'elle diminue à chaque itération de l'algorithme des k moyennes et qu'elle converge.

Exercice 27. Modifier la fonction `kmoyennes` pour que l'algorithme s'arrête lorsque la variation de la fonction coût est inférieure à un réel ε donné en paramètre.

1.5 Jeux d'accessibilité à deux joueurs

On rappelle qu'un graphe orienté G est un couple (V, E) où V est l'ensemble des sommets (que l'on supposera fini) et E l'ensemble des arêtes, c'est-à-dire une partie de $V \times V$.

Ainsi $(x, y) \in E$ signifie qu'il existe une arête reliant les sommets x et y et qu'elle est orientée de x vers y.

La matrice d'adjacence d'un graphe $G = (V, E)$ à n sommets est la matrice M de $\mathcal{M}_n(\mathbb{R})$ telle que, pour tout i et j, le coefficient $m_{i,j}$ vaut 1 s'il existe une arête du sommet numéro i vers le sommet numéro j et 0 sinon.

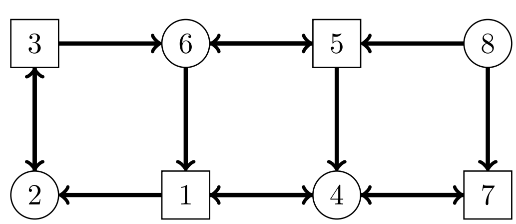
Exercice 28. On numérote les sommets d'un graphe à n sommets de 0 à $n-1$. Écrire une fonction `adjacence` de paramètre un graphe `G` et qui renvoie sa matrice d'adjacence.

On utilisera les matrices du module `numpy`. Pour créer une matrice nulle taille $n \times n$:

```
M=np.zeros((n,n))
```

Le graphe est dit **biparti** s'il existe une partition de l'ensemble des sommets V en deux sous-ensembles V_0 et V_1 (cela signifie que $V = V_0 \cup V_1$ et $V_0 \cap V_1 = \emptyset$) tels que chaque arête a un sommet dans V_0 et l'autre dans V_1.

Exemple 8. On considère le graphe biparti suivant :



On a $V_0 = \{2, 4, 6, 8\}$ et $V_1 = \{1, 3, 5, 7\}$.

Pour stocker en mémoire ce graphe, on utilise des listes :

```
V=[1,2,3,4,5,6,7,8]
E=[[1,2],[2,3],[3,2],...
G=[V,E]
```

Pour trouver le numéro d'un sommet dont on connaît le nom, on utilisera la commande :

```
i=V.index(x)
```

Exercice 29. Écrire une fonction `biparti` prenant en argument un ensemble d'arêtes E et des ensembles de sommets V_0 et V_1 qui vérifie si chaque arête a un sommet dans V_0 et l'autre dans V_1. La fonction renvoie `True` dans ce cas et `False` sinon.

Exercice 30. Écrire une fonction `voisins` prenant en argument un graphe G et un sommet v et qui renvoie la liste des sommets w tels que (v, w) est une arête du graphe.

On s'intéresse maintenant au jeu à deux joueurs J_0 et J_1 suivant. On se donne un graphe biparti $G = (V_0 \cup V_1, E)$. Le triplet (G, V_0, V_1) est appelé graphe de jeu ou arène.

On décide que le joueur J_0 contrôle les sommets V_0, le joueur J_1 contrôle les sommets V_1.

Un sommet v_0 est choisi pour commencer la partie. C'est au joueur qui contrôle ce sommet de jouer, il choisit une arête dont l'origine est v_0 si c'est possible (sinon la partie s'arrête). L'extrémité de l'arête choisie est un sommet v_1 appartenant à l'autre joueur (puisque le graphe est biparti) qui choisit à son tour (si c'est possible) une arête d'origine v_1 définissant un sommet v_2, etc.

Lorsqu'un joueur ne peut pas jouer, la partie s'arrête et il a perdu.

Exercice 31. Dans le graphe de l'exemple précédent, justifier qu'il ne peut pas y avoir de perdant. Qu'en est-il si on enlève l'arête $(7, 4)$?

Exercice 32. Écrire une fonction qui donne la liste des culs-de-sac c'est-à-dire la liste des sommets qui peuvent bloquer l'un des joueurs.

On suppose que l'arène (G, V_0, V_1) est sans cul-de-sac (donc aucun joueur ne peut être bloqué) et on se donne un ensemble $F \subset V$.

Le joueur J_0 est déclaré gagnant si un sommet appartenant à F est atteint. Pour gagner, le joueur J_1 doit d'empêcher l'accès à un sommet appartenant à F.

Les éléments de F sont appelés **sommets finaux**.

> **Exemple 9.** On reprend l'exemple précédent dans lequel J_0 contrôle les sommets pairs V_0 et J_1 les sommets impairs V_1. On fixe $F = \{1\}$.
>
> Si le point de départ est le sommet 3, le joueur J_1 peut obtenir un match nul. Si le point de départ est le sommet 5, il est sûr de perdre !

Le problème est de connaître les positions de départ ayant une stratégie gagnante pour J_0, c'est-à-dire lui permettant de gagner à coup sûr.

Pour cela on construit une suite croissante $E_0, E_1, \ldots$ d'ensembles de sommets par récurrence de la façon suivante. On pose $E_0 = F$ et, pour obtenir E_n :

1. si n est impair, on ajoute à E_{n-1} l'ensemble des sommets x contrôlés par J_0 pour lesquels il existe une arête (x, y) avec $y \in E_{n-1}$.
2. si n est pair, on ajoute à E_{n-1} l'ensemble des sommets x contrôlés par J_1 tels que toute arête d'origine x a une extrémité dans E_{n-1}.

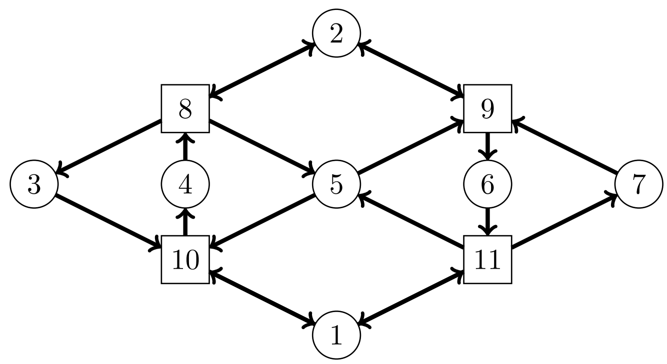
Comme le nombre de sommets est fini, la suite $(E_n)_{n\in\mathbb{N}}$ est stationnaire. Il existe donc un plus petit entier k tel que $E_k = E_{k+1} = E_{k+2} = \cdots$ Sur l'ensemble E_k, le joueur J_0 dispose d'une stratégie gagnante. Sur les autres sommets, le joueur J_1 peut obtenir un match nul.

Exercice 33. Écrire une fonction `cible` dont les paramètres sont un graphe G, un ensemble de sommets W inclus dans V et un sommet x appartenant à V. Cette fonction renvoie des compteurs c_1 et c_2. Le compteur c_1 est le nombre d'arêtes (x, y) telles que $y \in W$ et c_2 est le nombre d'arêtes (x, y) telles que $y \notin W$.

Exercice 34. Écrire une fonction `strategie` qui détermine l'ensemble E_k. La fonction prend en argument l'ensemble `F`.

On peut considérer deux ensembles de sommets finaux F_0 et F_1 (bien sûr ces ensembles sont disjoints). Pour J_0, il s'agit que l'un des points de F_0 soit atteint et d'empécher que les points de F_1 le soit et pour J_1 c'est le contraire.

Exemple 10. Pour le graphe biparti suivant, on pourra prendre $F_0 = \{4\}$ et $F_1 = \{6\}$.



Exercice 35. Écrire la fonction `strategie2` qui donne la stratégie gagnante pour chaque joueur.

On remarque qu'il peut exister des sommets qui n'appartiennent à aucune stratégie gagnante. En partant de tels sommets, si les deux joueurs jouent convenablement, aucun ne peut gagner : il y a match nul.

1.6 Problèmes du sac à dos, notion d'heuristique

1.6.1 Algorithme glouton et algorithme exhaustif

Considérons le problème de sac à dos suivant. On veut remplir un sac avec des objets donnés de manière à atteindre (si possible) un poids total fixé, appelé cible.

Exercice 36. On dispose d'un sac pouvant contenir 200kg et des poids (en kg) :

p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
2	13	17	23	29	31	38	51	68	78

Quels poids doit-on choisir pour remplir le sac ?

On se rend compte sur cet exemple que résoudre un problème de sac à dos peut être difficile même avec un petit nombre d'objets.

Il existe néanmoins un cas particulier où la solution est évidente : le cas où chaque poids de la liste est strictement supérieur à la somme de tous les précédents. On dit alors que la liste des poids est **supercroissante**.

Dans ce cas, on prend les poids les uns après les autres en partant du plus grand. On met dans le sac, uniquement les poids qui peuvent y rentrer !

Exercice 37. On dispose d'un sac pouvant contenir 400kg et des poids (en kg) :

p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
4	5	11	21	44	89	177	352	704	1408

1. Vérifier que la suite est supercroissante. Quels poids doit-on choisir pour remplir le sac ?
2. Écrire une fonction qui teste si une suite est supercroissante.
3. Écrire une fonction `glouton( cible ,L)` où `cible` est la contenance du sac et L une liste de poids supercroissante, et qui renvoie la liste des poids utilisés.

Si la suite des poids n'est pas supercroissante, on peut tenter une recherche exhaustive. C'est ce que fait la fonction `recherche` :

```
def recherche(L, cible ):
    SacsPleins=[]
    ListeX=[len(L)*[0]]
    while ListeX !=[]:
        X=ListeX.pop(0)
        if somme(X,L)==cible:
            SacsPleins.append(contenusac(X,L))
        else:
            j=indmin(X)
            for l in range(j+1,len(L)):
                if somme(X,L)+L[l]<=cible:
                    Y=X.copy()
                    Y[l]=1
                    ListeX.append(Y)
    return SacsPleins
```

Dans cette fonction, on suppose que la liste `L` est une liste triée par ordre décroissant, ce qui peut être faire au préalable par la commande suivante :

```
L.sort ( reverse=True)
```

Le paramètre `cible` est un entier positif.

La liste `X` est une liste de même taille que `L` contenant des 0 ou des 1. Cette liste représente le contenu d'un sac à dos. Si l'objet numéro i est présent dans le sac alors `X[i]=1` sinon `X[i]=0`.

Si le sac à dos est vide, la liste `X` ne contient donc que des 0.

La fonction indmin prend en paramètre une liste contenant uniquement des 0 et des 1. Elle renvoie -1 si la liste donnée en paramètre est vide, elle renvoie l'indice du dernier 1 de la liste sinon.

Exercice 38. Écrire la fonction indmin.

Exercice 39. Écrire les fonctions somme et contenusac. puis justifier que le programme recherche donne la liste des sacs à dos dont le poids est égal à la cible (s'il en existe un). Expliquer pourquoi la boucle **while** a une fin.

Exercice 40. Donner une estimation de la complexité. Expliquer pourquoi la fonction ne donnera probablement rien avec une cible de 10000 sur la liste :

```
L=[239, 764, 287, 295, 1408, 428, 779, 320, 935,316, 1537, 395, 57, 1610, 7690, 6117,
25, 312, 9528, 110, 8431, 21, 7385, 345, 812, 2131, 4303, 355, 1517, 478]
```

Le problème est que la complexité n'est pas polynomiale. Lorsque la liste dépasse quelques dizaines le temps de calcul est trop long !

Exercice 41. Modifier la fonction recherche pour qu'elle renvoie, dès que la cible est atteinte, le contenu du premier sac à dos correspondant.

1.6.2 Approximation par un algorithme glouton

On s'intéresse maintenant à un problème de sac à dos légèrement différent. On veut remplir, cette fois, un sac avec des objets numérotés de 1 à n. Chaque objet a non seulement un poids mais aussi une valeur. Le poids de l'objet i est noté w_i et sa valeur v_i.

L'autre différence avec le problème précédent est que l'on dispose d'autant d'objets i que l'on veut.

Le sac a une contenance maximale que l'on note $w_{\max} > 0$. L'objectif est de le remplir de sorte que la valeur totale des objet qu'il contient soit maximale.

> **Exemple 11.** On pourra tester les fonctions dans le cas suivant. On a un sac à dos pouvant contenir un poids maximal de 100 et on a 7 objets différents dont les poids sont indiqués dans la liste W et les valeurs dans la liste V ci-dessous :
>
> ```
> W=[2,3,5,7,11,13,20]
> V=[1,10,12,25,70,100,150]
> ```

Comme dans le problème précédent, la recherche exhaustive n'aboutit pas dès que le nombre d'objets dépasse quelques dizaines ou même lorsque le poids maximal est « nettement » plus grand que le poids des objets. On utilise donc un algorithme approché (une heuristique), comme pour l'algorithme des k-moyennes.

On classe les objets par ordre décroissant de leur valeur par unité de poids. On prend le premier de la liste qui peut rentrer dans le sac et on en met autant de copies de cet objet que possible. Ensuite, on met dans le sac l'objet suivant dans la liste qui peut rentrer dans le sac et on met autant que possible de copies de cet objet dans le sac. On continue ainsi jursqu'à ce que l'on ne puisse plus mettre d'objet du tout.

Exercice 42. Complèter les trois lignes de la fonction ci-dessous triant les objets par ordre décroissant de leur valeur par unité de poids.
La liste `Q` contient la liste des valeurs par unité de poids de chaque objet.

```
def tri(V,W):
    n=len(W)
    Q= ________________________________

    for i in range(1,n):
        j=i
        while j>=1 and ______________ :
            Q[j], Q[j-1]=Q[j-1],Q[j]
            V[j], V[j-1]=V[j-1],V[j]
            W[j],W[j-1]=W[j-1],W[j]
              ____________
    return V,W
```

Exercice 43. Écrire une fonction `glouton(V,W,wmax)` utilisant la fonction `tri`, et qui renvoie la liste `L` contenant les numéros d'objets choisis ainsi que la valeur du sac.

On rappelle que la commande `randint(n)` renvoie un nombre compris aléatoire compris entre 0 et $n-1$:

```
import numpy.random as rd
i=rd.randint(10)
```

Exercice 44. Écrire une fonction `aleatoire` qui remplit un sac en choisissant des objets au hasard jusqu'à ce qu'un objet choisi n'ait pas la place de rentrer. La fonction renvoie alors la liste des numéros choisis ainsi que la valeur du sac.

Exercice 45. Utiliser la fonction aleatoire pour montrer que le sac obtenu par l'algorithme glouton ne donne pas le meilleur sac possible dans l'exemple donné.

Il semble que la fonction aleatoire répétée un grand nombre de fois donne un meilleur résultat que la fonction glouton.

Ce n'est pas le cas pour un poids maximal plus grand (par exemple 1000).

Exercice 46. En modifiant la fonction recherche du paragraphe précédent, écrire une fonction qui détermine tous les sacs possibles et qui en donne un de valeur maximale.

Dans l'exemple donné avec un poids maximal de 100, on obtient 201756 sacs possibles ! On obtient donc rapidement le meilleur sac. Ce n'est pas le cas lorsque le poids maximal est 1000.

2 Algorithmique numérique

2.1 Méthode d'Euler

2.1.1 La méthode expliquée sur un exemple simple

Pour tracer la courbe $y = f(x)$, on utilise la fonction `plot` du module `matplotlib.pyplot` et la fonction `linspace` du module `numpy`.

```
import numpy as np;
import matplotlib.pyplot as plt
```

La courbe est tracée point par point. Les abscisses et les ordonnées sont stockés dans deux listes que l'on donne en argument de la fonction `plot`.

On suppose que la fonction f, les réels a, b l'entiers N sont définies. Voici deux exemples :

```
N=100;
X=np.linspace(a,b,N);
Y=[f(x) for x in X]
plt.plot(X,Y)
plt.show()
```

On obtient la courbe de f sur l'intervalle $[a, b]$ à l'aide de N points. On prend au moins 100 points (sauf pour représenter une fonction affine auquel cas deux points suffisent).

Exercice 47. Tracer la courbe de la fonction $x \longmapsto \mathrm{e}^x$ sur l'intervalle $[-2, 2]$. Écrire une fonction qui trace la tangente à la courbe de la fonction exponentielle en un point d'abscisse a donné en paramètre.

Une équation différentielle du premier ordre $y' = f(y, t)$ peut se résoudre numériquement à l'aide de la méthode d'Euler.

Cette méthode consiste à calculer de proche en proche les valeurs $y(t)$ à partir de la condition initiale $y(t_0) = y_0$ en remplaçant $y'(t)$ le taux d'accroissement :

$$\frac{y(t+h) - y(t)}{h}$$

où h est un réel « proche » de 0 fixé. Donc, connaissant $y(t)$, on calcule $y(t + h)$ par la formule :

$$y(t + h) \approx y(t) + hf(y(t), t).$$

On obtient des valeurs approchées de $y(t_0 + kh)$ pour tout $k \in \mathbb{N}$.

Exemple 12. La fonction exponentielle est solution de l'équation $y' = y$ avec pour condition initiale $y(0) = 1$. Donc si h est « proche » de 0 :

$$y(t+h) \approx y(t) + hy'(t) = (1+h)y(t)$$

En prenant $t = nh$, on a :

$$y((n+1)h) \approx (1+h)y(nh)$$

On en déduit que $\exp(nh) \approx (1+h)^n$. Cette approximation est « correcte » à condition que h soit « petit » et n ne soit pas trop « grand ».

Exercice 48. En utilisant l'exemple ci-dessus tracer des courbes approchées de la fonction exponentielle sur l'intervalle $[0, 2]$ en prenant $h = 1/10$ puis $h = 1/100$. Comparer avec la « vraie » courbe de la fonction exponentielle.

2.1.2 Le pendule

On considère une masse ponctuelle G au bout d'un fil sans masse, inextensible de longueur ℓ pouvant osciller dans un plan vertical sous l'effet de la pesanteur.
On repère la position du point G par l'angle θ qu'il fait avec la verticale.
Si l'on néglige les frottements, l'angle θ vérifie l'équation différentielle du second ordre :

$$\ddot{\theta} + \frac{g}{\ell}\sin\theta = 0$$

On résout numériquement cette équation en adaptant la méthode d'Euler utilisée pour les équations du premier ordre mais en faisant cette fois deux approximations :

$$\dot{\theta}(t+h) \approx \dot{\theta}(t) + \ddot{\theta}(t)h$$
$$\theta(t+h) \approx \theta(t) + \dot{\theta}(t)h$$

Pour les applications numériques, on prendra $g = 9,81\text{m.s}^{-2}$, $\ell = 0,20\text{m}$ et $\dot{\theta}(0) = 0$.

Exercice 49. Écrire une fonction de paramètres θ_0 et $t_{\max}$ qui affiche la courbe de θ en fonction du temps. Afficher sur un même graphique les courbes pour θ_0 variant de 0 à π de $\pi/10$ en $\pi/10$ sur l'intervalle $[0, t_{\max}]$ avec $t_{\max} = 10\,\text{s}$.

Lorsque $\theta \approx 0$, on peut faire l'approximation $\sin\theta$ par θ. On obtient une équation différentielle que l'on sait résoudre à l'aide des fonctions usuelles.

Exercice 50. Résoudre cette nouvelle équation et comparer graphiquement avec les courbes obtenues sans cette approximation pour $\theta_0 = \frac{\pi}{10}$ et $\theta_0 = \frac{\pi}{2}$.

2.1.3 Chute d'un corps dans un fluide

On lance, dans le plan vertical (xOz), un point matériel M, initialement au point O, avec une vitesse initiale $\overrightarrow{v_0}$ dans un fluide visqueux exerçant une force de frottement quadratique en vitesse :

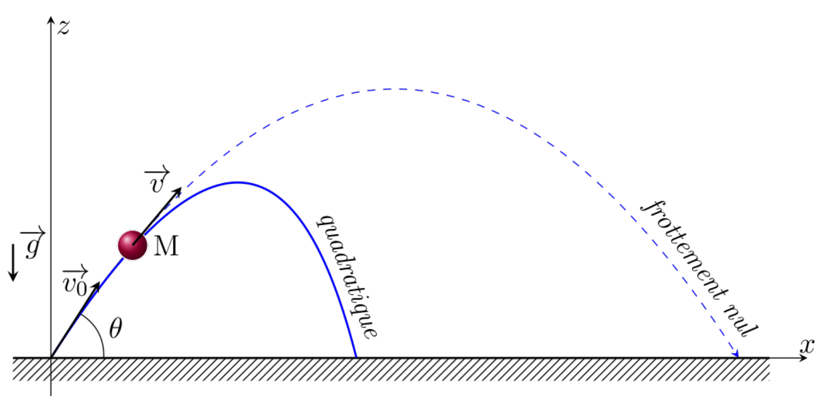
$$\overrightarrow{F}_t = -\beta v \overrightarrow{v}$$

Si l'on tient compte uniquement du poids et de la force de frottement, l'équation du mouvement donne :

$$m\frac{\mathrm{d}^2\overrightarrow{OM}}{\mathrm{d}t^2} = m\overrightarrow{g} - \beta v \overrightarrow{v}.$$

On est donc amené à résoudre le système différentiel :

$$\begin{cases} \ddot{x} & = -\frac{\beta}{m}\dot{x}\sqrt{\dot{x}^2+\dot{z}^2} \\ \ddot{z} & = -g - \frac{\beta}{m}\dot{z}\sqrt{\dot{x}^2+\dot{z}^2} \end{cases}$$



Exercice 51. Écrire la fonction `f` qui, à tout couple de réels $(\dot{x}, \dot{z})$ associe le couple $(\ddot{x}, \ddot{z})$.

On note θ l'angle entre l'axe (Ox) et le vecteur $\overrightarrow{v_0}$.

Exercice 52. Écrire une fonction `vitesseinitiale` de paramètres v_0 et θ, et qui renvoie le couple $(\dot{x}, \dot{z})$ à l'instant $t = 0$.

On fixe $\Delta t = 10^{-4}$ et, pour toute fonction dérivable, on fait l'approximation :

$$f'(t + \Delta t) \approx f(t) + \Delta t f'(t)$$

On prend $g = 9,81$, $v_0 = 10$, $\theta = 60°$, $m = 1$.

Exercice 53. Écrire une fonction `trajectoire` de paramètre β et qui affiche la trajectoire du point M jusqu'à ce que celui-ci revienne au sol.
Afficher sur le même graphique les trajectoires pour $\beta \in \left\{0, \frac{1}{10}, \frac{1}{4}, \frac{1}{2}, 1, 2\right\}$.

2.2 Résolution d'un système linéaire par la méthode du pivot de Gauss

On se limite à la résolution de systèmes de Cramer, c'est-à-dire de systèmes linéaires ayant autant d'équations que d'inconnues et admettant une et une seule solution.
On considère donc le système suivant :

$$(S)\begin{cases} a_{0,0}x_0 + a_{0,1}x_1 + \ldots + a_{0,n-1}x_{n-1} & = b_0 \\ a_{1,0}x_0 + a_{1,1}x_1 + \ldots + a_{1,n-1}x_{n-1} & = b_1 \\ \vdots & = \vdots \\ a_{n-1,0}x_0 + a_{n-1,1}x_1 + \ldots + a_{n-1,n-1}x_{n-1} & = b_{n-1} \end{cases}$$

Matriciellement, ce système s'écrit sous la forme $AX = B$ où

$$A = \begin{pmatrix} a_{0,0} & \cdots & \cdots & a_{0,n-1} \\ \vdots & & & \vdots \\ a_{n-1,0} & \cdots & \cdots & a_{n-1,n-1} \end{pmatrix} \quad X = \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix} \quad B = \begin{pmatrix} b_0 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

La matrice A est la matrice des coefficients du système, le vecteur B est la matrice colonne des seconds membres.
Enfin, on note M la matrice augmentée du système. C'est la matrice regroupant les coefficients et les seconds membres :

$$M = \left(\begin{array}{cccc|c} a_{0,0} & \cdots & \cdots & a_{0,n-1} & b_0 \\ \vdots & & & \vdots & \vdots \\ a_{n-1,0} & \cdots & \cdots & a_{n-1,n-1} & b_{n-1} \end{array}\right)$$

Exemple 13. On pourra tester les différentes fonctions pour résoudre le système :

$$\begin{cases} x+2y+3z=1 \\ 2x+4y+8z=5 \\ 3x+2y+2z=15 \end{cases}$$

2.2.1 Opérations élémentaires sur les lignes

Nous allons structurer notre programme en créant les fonctions suivantes afin de découper le travail : recherche d'un pivot, échange de lignes, transvections.

On suppose que la matrice augmentée est une variable globale `M`

On pourra la définir comme une liste de listes ou comme une matrice du module `numpy`. Dans le premier cas, on obtient le coefficient de la ligne i, colonne j par l'instruction `M[i][j]` ; dans le deuxième cas par la commande `M[i,j]`.

Exercice 54. Créer une fonction `echange(i0,i1)` qui prend en entrée deux indices de lignes i_0 et i_1, et qui échange les lignes L_{i_0} et L_{i_1} de la matrice M. Donner la complexité de cette fonction.

Exercice 55. Créer une fonction `transvection(i0,i1,mu)` qui prend en entrée deux indices de lignes i_0, et i_1, une constante `mu` et qui remplace la ligne i_1 par la ligne i_1 à laquelle on retire μ fois la ligne i_0. Donner la complexité de cette fonction.

Exercice 56. Créer une fonction `recherchepivot(j)` qui prend en entrée un indice de colonne j, qui cherche un indice de ligne $i \in [\![j, n-1]\!]$ tel que $|a_{i,j}|$ soit maximal et qui échange les lignes i et j. Donner la complexité de cette fonction.

2.2.2 Résolution du système

Grâce aux opérations élémentaires sur les lignes, on peut échelonner la matrice M par la méthode gauss. On obtient ainsi un système linéaire triangulaire supérieur équivalent au système initial.

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} & b'_0 \\ 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & a_{n-2,n-1} & b'_{n-2} \\ 0 & \cdots & 0 & a_{n,n} & b'_{n-1} \end{pmatrix}$$

Exercice 57. Compléter la fonction Gauss ci-dessous qui échelonne la matrice M par opérations élémentaires sur les lignes.

```
def gauss():
    for j in range(0,n):
        recherchepivot(j)
        for i in ______________________________
            ______________________________
```

Si les coefficients de la diagonale sont tous non nuls, le système admet une unique solution. On l'obtient en divisant chaque ligne par son pivot puis en faisant de bas en haut des opérations sur les lignes pour obtenir une matrice de la forme :

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & b''_0 \\ 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & b''_{n-2} \\ 0 & \cdots & 0 & 1 & b''_{n-1} \end{pmatrix}$$

Les solutions sont les coefficients de la dernière colonne.

Exercice 58. Créer une fonction `dilatation(i,mu)` qui prend en entrée un indice de ligne i et qui remplace la ligne i par la ligne i multipliée par μ.

Exercice 59. Compléter la fonction résolution qui renvoie les solutions du système.

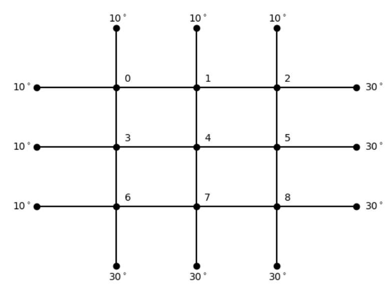
```
def resolution():
    gauss()
    for i in range(n):
            dilatation______________________________

    for j in range(n-1,-1,-1):
        for i in ______________________________
            transvection______________________________
```

Exercice 60.

1. Écrire une fonction prenant en paramètre une matrice carrée, et qui renvoie cette matrice augmentée de la matrice identité de même taille.
2. Écrire une fonction qui détermine l'inverse d'une matrice carrée inversible.

2.2.3 Une application

On considère la grille de la figure ci-dessous composée de tiges métalliques. Soit $T_0, \dots T_8$ les températures aux neufs nœds intérieurs du quadrillage de la figure. La température en un nœd est à peu près égale à la moyenne des températures aux quatre nœds voisins (au-dessus,à gauche à droite et en dessous). La température aux extrémités est fixée telle qu'indiquée sur la figure.



Exercice 61. Donner une estimation des températures $T_0, \dots, T_8$ en résolvant un système linéaire.

Exercice 62. Généraliser à une grille $n \times n$. Représenter les nœuds de la grille avec une couleur qui dépend de sa température.

2.3 Interpolation de Lagrange

2.3.1 La méthode sur un exemple simple

On se donne $(n+1)$ points $M_0 = (x_0, y_0), M_1 = (x_1, y_1), \dots M_n = (x_n, y_n)$ d'abscisses disctintes (autrement dit $x_i \neq x_j$ pour tout $i \neq j$).

On cherche un polynôme P de degré n tel que $P(x_i) = y_i$ pour tout $i \in [\![0, n]\!]$.

Pour cela, on introduit les polynômes d'interpolation de Lagrange :

$$\forall i \in [\![0, n]\!], \quad L_i(X) = \prod_{k \neq i} \frac{X - x_k}{x_i - x_k}.$$

Exemple 14. On considère les points $M_0 = (1, 2)$, $M_1 = (2, 4)$, $M_2 = (4, 5)$, $M_3 = (5, -1)$. Les polynômes d'interpolation de Lagrange sont :

$$\begin{aligned}
L_0 &= -\frac{1}{12}(X-2)(X-4)(X-5)\\
L_1 &= \frac{1}{6}(X-1)(X-4)(X-5)\\
L_2 &= \frac{1}{6}(X-1)(X-2)(X-5)\\
L_3 &= \frac{1}{12}(X-1)(X-2)(X-4)
\end{aligned}$$

Dans la suite, on utilisera cette liste de points que l'on stocke dans une liste de listes :

```
L=[[1,2],[2,4],[4,5],[5,-1]]
```

Exercice 63. Écrire une fonction `lagrange` qui prend en argument une liste de points L, un entier i et un réel x, et qui renvoie $L_i(x)$.

On remarque que $L_i(x_i) = 1$ et que $L_i(x_j) = 0$ si $i \neq j$. Il est donc facile de trouver un polynôme P de degré au plus n tel que $P(x_i) = y_i$:

$$P(X) = \sum_{i=1}^{n} y_i L_i(X).$$

Exercice 64. Écrire une fonction `interpolation` de paramètre une liste de points L, un réel x et qui renvoie le polynôme d'interpolation évalué en x.

Exercice 65. Tracer les points et la courbe d'interpolation sur le même graphique.

2.3.2 Interpolation d'une fonction

On cherche à approcher une fonction f définie sur un intervalle $I = [a, b]$.

Pour cela, on choisit n et des réels distincts $x_0, \ldots, x_n$ appartenant à l'intervalle $[a, b]$. En notant L_i les polynômes de Lagrange associés aux abscisses $x_0, \ldots, x_n$, le polynôme d'interpolation s'écrit :

$$P(X) = \sum_{i=1}^{n} f(x_i) L_i(X)$$

Exercice 66. Tracer la courbe d'interpolation de la fonction sinus sur l'intervalle $[0, \pi]$ avec $x_k = \frac{k\pi}{n}$ pour $k \in [\![0, n]\!]$ à l'aide d'une fonction prenant n en paramètre.

2.3.3 Interpolation par morceaux

On considère une fonction f définie sur un intervalle $I = [a, b]$ que l'on découpe en n intervalles $I_k = [a_k, a_{k+1}]$ avec $a_0 = a < a_1 < \cdots < a_n = b$.

La méthode consiste à interpoler f sur chaque intervalle I_k et non pas globalement sur I.

Exercice 67. On veut faire l'interpolation par morceaux de la fonction sinus sur l'intervalle $[0, \pi]$. Pour cela on pose $I_k = \left[\frac{k\pi}{n}, \frac{(k+1)\pi}{n}\right]$ pour k compris entre 0 et $n - 1$. L'entier n sera donné en paramètre des fonctions d'interpolation.

1. Tracer la courbe d'interpolation par morceaux en prenant sur I_k les extremités de l'intervalle.
2. Tracer la courbe d'interpolation par morceaux en prenant sur I_k pour unique point son centre.
3. Tracer la courbe d'interpolation par morceaux en prenant sur I_k les extrémites et le centre.

3 Les bases de données

3.1 Installation avec Xampp (Windows, Linux, OS X)

1. Télécharger Xampp sur https://www.apachefriends.org/
2. Installer Xampp (sous windows : sur la racine du disque dur de préférence). Il faut installer au moins Apache et MySQL.
3. Vérifier que tout est ok en lançant l'exécutable xampp−control.exe :

```
15:40:11  [Apache] XAMPP Apache is already running on port 443
15:40:12  [mysql] XAMPP MySQL is already running on port 3306
15:40:12  [main] Enabling autostart for module "Apache"
15:40:12  [main] Enabling autostart for module "MySQL"
15:40:12  [main] Starting Check−Timer
15:40:12  [main] Control Panel Ready
15:40:12  [Apache] Autostart aborted: Apache is already running
15:40:12  [mysql] Autostart aborted: MySQL is already running
```

4. Sur un navigateur, taper dans la barre d'adresse : 127.0.0.1, en haut de la page, dans le menu, choisir phpMyAdmin
5. Créer trois nouvelles bases de données : cours1, cours2 et tsi2024.
6. Choisir l'une après l'autre les bases de données créées et importer les fichiers cours1.sql, cours2.sql, tsi2024.sql

   Les fichiers sont à télécharger sur https://cahier−de−prepa.fr/tsi2−coubertin.

3.2 Vocabulaire

Une base de données est une collection de données structurées et organisées à l'aide de **tables** que l'on représente à l'aide de tableaux à 2 dimensions.

Une table est définie par une liste d'attributs. Un **attribut** est un nom décrivant une information. Cela peut être par exemple s'il s'agit d'information sur des personnes, le nom, le prénom, l'âge, la taille, etc.

Pour chaque attribut, on définit un **domaine**, c'est-à-dire l'ensemble de ses valeurs possibles. S'il s'agit du nom ou du prénom, le domaine sera l'ensemble des chaînes de caractères, s'il s'agit de l'âge, le domaine sera l'ensemble des entiers, s'il s'agit de la taille, le domaine pourra être l'ensemble des flottants.

Enfin, une table contient des **enregistrements**. Un enregistrement contient les valeurs des attributs qui se rapportent au même objet

Chaque enregistrement doit être identifié par un ou plusieurs attributs (le plus souvent un seul). Ce (ou ces) attribut(s) définisse(nt) la **clé primaire** de la table.

Très souvent, la table contient un attribut particulier appelé *idendifiant* qui caractérise l'objet. Cet attribut peut tout simplement être un numéro (donc un entier) qui augmente de 1 à chaque nouvel enregistrement. Cet attribut définit alors la clé primaire.

Lorsque l'on représente une table, les enregistrements sont les lignes, les atributs sont les colonnes.

Chaque table d'une base de données a un nom qui commence, selon l'usage, par une majuscule.

Exemple 15. Base de données cours1.sql
La table ci-dessous, que l'on appelle Pays, a 4 attributs : nom_pays, nom_continent, superficie (en km^2), population (en millions d'habitants). Elle contient 4 enregistrements. L'attribut pays peut être la clé primaire puisqu'il n'y a pas deux pays ayant le même nom.

nom_pays	nom_continent	superficie	population
Allemagne	Europe	357588	83,2
France	Europe	551695	67,75
Italie	Europe	301230	59,11
Japon	Asie	377973	125,7

Table Pays

Le domaine des attributs nom_pays et nom_continent sont des chaînes de caractère, le domaine de l'attribut superficie est le type entier, le domaine de l'attribut population est le type flottant.

Le **schéma d'une table** est constitué de l'ensemble de ses attributs et du domaine de chacun de ses attributs.

3.3 Les requêtes simples

Pour extraire des informations d'une table de données, on utilise un langage de **requêtes**, le langage SQL (Structured Query Language).

Toutes les requêtes que nous verrons commencent par le mot clé SELECT et contienne la clause FROM.

■ Pour afficher certains attributs d'une table, on utilise l'instruction SELECT...FROM.

```
SELECT nom_pays , population FROM Pays;
```

```
+-----------+------------+
| nom_pays  | population |
+-----------+------------+
| Allemagne |       83.2 |
| France    |      67.75 |
| Italie    |      59.11 |
| Japon     |      125.7 |
+-----------+------------+
```

■ Pour ne pas afficher deux fois la même ligne, on écrit SELECT DISTINCT

```
SELECT DISTINCT nom_continent FROM Pays;
```

■ Pour afficher toute la table, on peut écrire tous les attributs ou bien :

```
SELECT * FROM Pays;
```

```
+---------------+
| nom_continent |
+---------------+
| Europe        |
| Asie          |
+---------------+
```

■ On peut sélectionner certains enregistrements avec la clause WHERE. Par exemple, si l'on veut uniquement la population des pays d'Europe, on écrira :

```
SELECT nom_pays,population FROM Pays WHERE nom_continent='Europe';
```

```
+-----------+------------+
| nom_pays  | population |
+-----------+------------+
| Allemagne |       83.2 |
| France    |      67.75 |
| Italie    |      59.11 |
+-----------+------------+
```

Seuls les enregistrements vérifiant la condition précisée après la clause WHERE sont affichés.

Pour former des critères simples, on utilise les symboles d'égalité et d'inégalité : =, <, <= (pour $\leq$), >, >= (pour $\geq$) et <> (pour différent).

On peut utiliser les opérateurs logiques OR, AND, NOT pour écrire un critère de sélection plus complexe.

■ On peut faire des opérations sur les attributs dont le domaine est l'ensemble des entiers ou l'ensemble des flottants. Ainsi, par exemple, la requête suivante affiche la densité de population des pays de la table Pays.

```
SELECT nom_pays AS pays,population/superficie*1E6 AS densité FROM Pays;
```

```
+-----------+--------------------+
| pays      | densité            |
+-----------+--------------------+
| Allemagne | 232.66999157757584 |
| France    | 122.80336055247918 |
| Italie    | 196.22879729891298 |
| Japon     |  332.5634289968918 |
+-----------+--------------------+
```

Le mot clé AS permet de nommer ou de renommer un attribut (ou une table).

■ On peut classer les résultats avec la clause ORDER BY (ordre croissant) ou ORDER BY ... DESC (ordre décroissant). La requête suivante affiche les pays par ordre décroissant de la superficie.

```
SELECT nom_pays FROM Pays ORDER BY superficie DESC;
```

```
+-----------+------------+
| nom_pays  | superficie |
+-----------+------------+
| France    |     551695 |
| Japon     |     377973 |
| Allemagne |     357588 |
| Italie    |     301230 |
+-----------+------------+
```

■ On peut ajouter la clause LIMIT pour limiter le nombre de résultats. Pour afficher les deux plus grands pays :

```
SELECT nom_pays FROM Pays ORDER BY superficie DESC LIMIT 2;
```

```
+----------+
| nom_pays |
+----------+
| France   |
| Japon    |
+----------+
```

■ La commande OFFSET permet de faire un décalage des résultats. Si l'on veut afficher les deuxième et troisième plus grands pays, on fait un décalage de 1.

```
SELECT nom_pays FROM Pays ORDER BY superficie DESC LIMIT 2 OFFSET 1;
```

```
+-----------+
| nom_pays  |
+-----------+
| Japon     |
| Allemagne |
+-----------+
```

La commande OFFSET doit être utilisée avec la commande LIMIT.

Exercice 68.

1. Afficher tous les pays d'Europe dont la population et inférieure à 60 millions d'habitants.
2. Afficher le pays le plus peuplé.
3. Afficher les deux pays les moins peuplés.
4. Afficher le plus petit pays d'Europe.
5. Afficher tous les pays d'Europe par ordre de densité décroissante.
6. Afficher les pays en indiquant leur nom, leur population en milliers d'habitants et leur superficie en milliers de km^2.

3.4 Opérateurs ensemblistes

Les tables et les résultats de requêtes peuvent être considérées comme des ensembles. Il est donc naturel de pouvoir faire des intersections, des réunions et des différences. Eh bien, c'est possible en SQL ! Pour cela on utilise les mots clés UNION, INTERSECT et EXCEPT.

Il faut simplement que les tables aient le même schéma (mêmes attributs, mêmes domaines). Lorsque les attributs sont différents, on peut toujours renommer avec le mot clé AS.

Exemple 16. Base de données cours1.sql

titre	realisateur	annee
Forrest Gump	Robert Zemeckis	1994
La liste de Schindler	Steven Spielberg	1993
12 hommes en colère	Sidney Lumet	1957
Vol au-dessus d'un nid de coucou	Milos Forman	1975
Amadeus	Milos Forman	1984
Duel	Steven Spielberg	1971

Table Film

nom	annee_de_naissance	pays_de_naissance
Robert Zemeckis	1952	États-Unis
Steven Spielberg	1946	États-Unis
Sidney Lumet	1930	États-Unis
Milos Forman	1953	Tchécoslovaquie
Ridley Scott	1937	Royaume-Uni

Table Realisateur

On veut afficher tous les réalisateurs présents dans les deux tables. On veut donc réunir les résultats des requêtes : SELECT realisateur FROM Film et SELECT nom FROM Realisateur

Les schémas ne sont pas tout à fait identiques. Dans chaque table il y a bien un seul attribut avec le même domaine mais les noms sont différents. On renomme avec le mot clé AS pour régler ce problème :

```
SELECT realisateur FROM Film
UNION
SELECT nom AS realisateur FROM Realisateur;
```

```
+------------------+
| realisateur      |
+------------------+
| Sidney Lumet     |
| Milos Forman     |
| Steven Spielberg |
| Robert Zemeckis  |
| Ridley Scott     |
+------------------+
```

Exercice 69.

1. Afficher la liste des réalisateurs qui apparaissent dans les tables Film et Realisateur.
2. Afficher la liste des réalisateurs qui apparaissent dans la table Realisateur mais pas dans la table Film.
3. Afficher la liste des années présentes dans les tables Realisateur et Film classée par ordre croissant.

3.5 Produit cartésien

■ La clause FROM définit un espace de recherche. Lorsqu'une seule table est précisée, cet espace correspond à l'ensemble des lignes de la table. Lorsqu'il y a deux tables, l'espace est constitué de toutes les combinaisons possibles des lignes des deux tables.

Autrement dit, si on note L_1 et L_2 l'ensemble des lignes des deux tables, l'espace de recherche est le produit cartésien $L_1 \times L_2$.

Exemple 17. La requête SELECT * FROM Film, Realisateur donne une table de 30 lignes.

```
SELECT titre, annee, annee_de_naissance AS annee2
FROM Film, Realisateur LIMIT 10;
```

```
+----------------------+-------+--------+
| titre                | annee | annee2 |
+----------------------+-------+--------+
| 12 hommes en colère  |  1957 |   1953 |
| 12 hommes en colère  |  1957 |   1937 |
| 12 hommes en colère  |  1957 |   1952 |
| 12 hommes en colère  |  1957 |   1930 |
| 12 hommes en colère  |  1957 |   1946 |
| Amadeus              |  1984 |   1953 |
| Amadeus              |  1984 |   1937 |
| Amadeus              |  1984 |   1952 |
| Amadeus              |  1984 |   1930 |
| Amadeus              |  1984 |   1946 |
+----------------------+-------+--------+
```

■ Dans une table produit, beaucoup de lignes ne nous intéressent pas. Dans l'exemple précédent, seules ont un intérêt, les lignes où l'attribut nom de la table Realisateur et l'attribut realisateur de la table Film coïncident. On peut obtenir cela, avec la requête suivante :

```
SELECT titre, annee, annee_de_naissance AS annee2
FROM Film, Realisateur
WHERE nom=realisateur;
```

Pour plus de clarté, on précise souvent la table d'où provient l'attribut :

```
SELECT titre, annee, annee_de_naissance AS annee2
FROM Film, Realisateur
WHERE Realisateur.nom=Film.realisateur;
```

On peut utiliser des alias pour simplifier :

```
SELECT titre, annee, annee_de_naissance AS annee2
FROM Film AS F, Realisateur AS R
WHERE R.nom=F.realisateur;
```

```
+-----------------------------------+-------+--------+
| titre                             | annee | annee2 |
+-----------------------------------+-------+--------+
| 12 hommes en colère               |  1957 |   1930 |
| Amadeus                           |  1984 |   1953 |
| Duel                              |  1971 |   1946 |
| Forrest Gump                      |  1994 |   1952 |
| La liste de Schindler             |  1993 |   1946 |
| Vol au-dessus d'un nid de coucou  |  1975 |   1953 |
+-----------------------------------+-------+--------+
```

Exercice 70. Déterminer le tableau produit par la requête :

```
SELECT annee,anne_de_naissance
FROM Film, Realisateur
WHERE annee>1990 AND pays_de_naissance='Etats-Unis';
```

3.6 Clés primaires et clés étrangères

■ Une clé primaire n'est pas nécessairement définie par un seul attribut même si c'est le cas le plus fréquent.

Considérons une table contenant une liste d'étudiants ayant 4 attributs : nom, prenom, 'date de naissance' et adresse. On utilise alors pour clé primaire le couple (nom,prénom) plutôt que l'attribut nom tout seul.

■ Une clé étrangère indique les attributs de la table qui font référence à la clé primaire d'autres tables.

Comme la clé primaire, les clés étrangères sont précisées lors de la création de la table.

Exemple 18. Base de données cours2.sql
On considère les tables suivantes (la clé primaire de chaque table est l'attribut écrit gras).

id	nom_pays	nom_continent
1	Royaume-Uni	Europe
2	Tchécoslovaquie	Europe
3	États-Unis	Amérique

Table Pays2

L'attribut id_pays est une clé étrangère qui fait référence à la clé primaire de la table Pays2

id	nom	annee_de_naissance	id_pays
1	Robert Zemeckis	1952	3
2	Steven Spielberg	1946	3
3	Sidney Lumet	1930	3
4	Milos Forman	1953	2
5	Ridley Scott	1937	1

Table Realisateur2

L'attribut id_rea est une clé étrangère qui fait référence à la clé primaire de la table Realisateur2

titre	id_rea	annee
Forrest Gump	1	1994
La liste de Schindler	2	1993
12 hommes en colère	3	1957
Vol au-dessus d'un nid de coucou	4	1975
Amadeus	4	1984
Duel	2	1971

Table Film2

3.7 Les jointures

■ La jointure permet d'exprimer des requêtes portant sur des données réparties dans plusieurs tables.

Prenons l'exemple des tables Film2 et Realisateur2. La table Film2 contient un numéro d'identification du réalisateur mais pas son nom. Il faut utiliser la table Realisateur2 pour l'obtenir.

On joint les deux tables avec l'instruction Film2 JOIN Realisateur2 et on précise que l'attribut id_rea correspond à l'attribut id avec la clause ON.

La requête s'écrit :

```
SELECT titre , nom FROM Film2 JOIN Realisateur2 ON id=id_rea;
```

Pour plus de clarté (ou pour lever toute ambiguité), on utilise des alias :

```
SELECT f.titre , r.nom
```

```
FROM Film2 AS f JOIN Realisateur2 AS r
ON r.id=f.id_rea ;
```

```
+------------------------------------+-------------------+
| titre                              | nom               |
+------------------------------------+-------------------+
| 12 hommes en colère                | Sidney Lumet      |
| Amadeus                            | Milos Forman      |
| Duel                               | Steven Spielberg  |
| Forrest Gump                       | Robert Zemeckis   |
| La liste de Schindler              | Steven Spielberg  |
| Vol au-dessus d'un nid de coucou   | Milos Forman      |
+------------------------------------+-------------------+
```

On remarque ici que la jointure est construite en faisant coïncider la clé primaire d'une table avec une clé étrangère de l'autre. C'est très souvent le cas.

■ Les tables formant la jointure peuvent être identiques. On parle dans ce cas d'**autojointure**.

Exemple 19. On veut déterminer la liste des paires de films de la table Film2 réalisées par la même personne.
L'utilisation d'alias est indispensable pour pouvoir écrire la conjonction d'égalité après la clause ON :

```
SELECT f1.titre, f2.titre
FROM Film2 AS f1 JOIN Film2 AS f2
ON f1.id_rea=f2.id_rea ;
```

Comme on s'interesse aux paires de films différents, on rajoute :

```
WHERE f1.Film2 < f2.Film2
```

```
+---------+------------------------------------+
| titre   | titre                              |
+---------+------------------------------------+
| Amadeus | Vol au-dessus d'un nid de coucou   |
| Duel    | La liste de Schindler              |
+---------+------------------------------------+
```

Exercice 71. Afficher la table où sont indiqués le nom, l'année de naissance et le pays de naissance de chaque réalisateur présent dans la table Realisateur2 .

Exercice 72. Afficher les paires distinctes de réalisateurs qui sont nés dans le même pays.

3.8 Agrégation

Jusqu'à maintenant le résultat d'une requête était toujours constitué à partir de lignes individuelles. Les fonctions d'agrégation de SQL permettent de faire des groupes et d'appliquer une fonction à ce groupe.

■ Les fonctions d'agrégation à connaître sont MIN, MAX, SUM (pour calculer une somme), AVG (pour calculer une moyenne) et COUNT (pour compter des lignes).

> **Exemple 20.** La requête suivante donne le nombre de films présents dans la table Film.
>
> ```
> SELECT COUNT(*) AS 'nombre de films' FROM Film;
> ```
>
> ```
> +-----------------+
> | nombre de films |
> +-----------------+
> |               6 |
> +-----------------+
> ```
>
> La requête suivante calcule la population totale des pays d'Europe présents dans la table Pays.
>
> ```
> SELECT SUM(population) FROM Pays WHERE nom_continent='Europe';
> ```
>
> ```
> +--------------------+
> | SUM(population)    |
> +--------------------+
> | 210.05999755859375 |
> +--------------------+
> ```

■ La clause GROUP BY partitionne la table en groupes. Par exemple, pour la table Film, la commande GROUP BY nom_continent groupe les lignes de la table par continent. Deux groupes sont formés. Les fonctions d'agrégation vont s'appliquer à chaque groupe.

Exemple 21. La requête suivante répartit les pays par continent. Il y a deux groupes donc le résultat a deux lignes. Les fonctions d'agrégation calcule, pour chaque groupe, le nombre de pays et la superficie totale.

```
SELECT nom_continent,count(*) , sum(superficie)
FROM Pays GROUP BY nom_continent;
```

```
+---------------+----------+------------------+
| nom_continent | count(*) | sum(superficie)  |
+---------------+----------+------------------+
| Asie          |        1 |           377973 |
| Europe        |        3 |          1210513 |
+---------------+----------+------------------+
```

■ La clause HAVING permet d'imposer des conditions sur les groupes. Cette clause s'écrit donc toujours avec une fonction d'agrégation.

Exemple 22. On veut uniquement les continents dont au moins deux pays sont présents dans la table Pays.

```
SELECT nom_continent,count(*) AS N,sum(superficie) AS 'Superficie totale'
FROM Pays GROUP BY nom_continent HAVING N>=2;
```

```
+---------------+---+-------------------+
| nom_continent | N | Superficie totale |
+---------------+---+-------------------+
| Europe        | 3 |           1210513 |
+---------------+---+-------------------+
```

■ La clause WHERE qui peut aussi être utilisée avec la clause GROUP BY (mais doit être placée avant !), ne peut exprimer des conditions que sur des lignes prises individuellement.

Exemple 23. On ne garde que les pays de la table dont la superficie est inférieure à 500 000 km^2.

```
SELECT nom_continent,count(*) AS N,sum(superficie) AS 'Superficie totale'
FROM Pays WHERE superficie<500000
GROUP BY nom_continent
```

```
+---------------+---+-------------------+
```

```
| nom_continent | N | Superficie totale |
+---------------+---+-------------------+
| Asie          | 1 |            377973 |
| Europe        | 2 |            658818 |
+---------------+---+-------------------+
```

3.9 Applications

3.9.1 Les pays

```
MariaDB [tsi]> DESCRIBE  Pays;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | NO   | PRI | NULL    |       |
| pays       | varchar(64) | NO   |     | NULL    |       |
| population | int(11)     | NO   |     | NULL    |       |
| superficie | int(11)     | NO   |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
```

Exercice 73.

1. Combien y-a-t'il d'habitants au Liechtenstein ?
2. Donner les 10 pays les plus peuplés classés par ordre décroissant.
3. Donner les 10 plus petits pays classés par ordre croissant.
4. Afficher les pays de plus de 100 millions d'habitants et indiquer leur densité de population. Classer par ordre décroissant de la population.
5. Donner les 10 pays ayant la densité de population la plus élevée, classés par décroissant. Indiquer la densité pour chacun de ces pays.

```
MariaDB [tsi]> DESCRIBE Capitales;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| id        | int(11)     | NO   | PRI | NULL    |       |
| pays      | varchar(64) | NO   |     | NULL    |       |
| capitale  | varchar(64) | NO   |     | NULL    |       |
| continent | varchar(64) | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

Exercice 74.

1. Quelle est la capitale du Suriname ?
2. Afficher tous les pays d'Europe avec leur capitale.
3. Afficher la liste des continents.
4. Afficher le nombre de pays de chaque continent.
5. Afficher pour tous les pays d'Asie, le nom, la capitale et la population. Classer par ordre alphabétique des pays.
6. Afficher le pays le plus peuplé de chaque continent.
7. Afficher la population et la superficie de chaque continent. Classer par ordre croissant de la population totale.

3.9.2 Base de données des « meilleurs » films

```
MariaDB [tsi]> DESCRIBE Films;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| id           | int(11)      | NO   | PRI | NULL    |       |
| titre        | varchar(255) | NO   |     | NULL    |       |
| annee        | int(11)      | NO   |     | NULL    |       |
| duree        | int(11)      | NO   |     | NULL    |       |
| note         | int(11)      | NO   |     | NULL    |       |
| realisateur1 | varchar(63)  | YES  |     | NULL    |       |
| realisateur2 | varchar(63)  | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
```

Exercice 75.

1. Afficher tous les films de Martin Scorsese ainsi que leur année de sortie.
2. Combien de films n'ont qu'un seul réalisateur ?
3. Donner la liste des réalisateurs (`realisateur1` ou `realisateur2`).
4. Afficher tous les films ayant une note supérieure ou égale à 90.
5. Afficher tous les réalisateurs dont au moins un film a obtenu une note supérieure ou égale à 90.
6. Afficher tous les films sorties entre 1980 et 2000.
7. Afficher tous les films de plus de 3h sorties avant l'an 2000.

8. Afficher les films ayant deux réalisateurs. Préciser les réalisateurs et la durée du film.
9. Afficher tous les réalisateurs dont au moins deux films sont dans la base de données.

```
MariaDB [tsi]> DESCRIBE Genres;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| id    | int(11)      | NO   | PRI | NULL    |       |
| titre | varchar(255) | NO   |     | NULL    |       |
| genre | varchar(32)  | NO   |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
```

Exercice 76.

1. Donner la liste des genres cinématographiques.
2. Afficher la liste des westerns.
3. Dans quels genres est classé le film `Chinatown` ?
4. Quelle est la durée du film `Taxi Driver` ?
5. Donner le nombre de films de chaque genre cinématographique.
6. Donner la liste des réalisateurs de thrillers.
7. Quelle est la plus longue comédie ?
8. Donner la liste des comédies de moins de 20 ans.

3.9.3 Les transports en île de France

La table Routes donne la liste des lignes de métro, trains et tramways en île de France. Chaque enregistrement de la table correspond à une ligne où sont indiqués, dans cet ordre, l'identifiant de la ligne, le réseau auquel appartient la ligne, le nom court et long de la ligne, et le type de la ligne codé par un entier de 0 à 2 dont la signification se trouve dans la table Transport.

```
MariaDB [tsi]> SELECT * FROM Transport;
+----+---------+
| id | name    |
+----+---------+
|  0 | Tramway |
|  1 | Métro   |
|  2 | Train   |
+----+---------+

MariaDB [tsi]> DESCRIBE Routes;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| route_id         | varchar(19) | NO   | PRI | NULL    |       |
| agency_id        | int(11)     | NO   |     | NULL    |       |
| route_short_name | varchar(9)  | NO   |     | NULL    |       |
| route_long_name  | varchar(84) | NO   |     | NULL    |       |
| route_type       | int(11)     | NO   |     | NULL    |       |
+------------------+-------------+------+-----+---------+-------+
```

Exercice 77.

1. À quoi correcpond l'enregistrement de la table Routes dont l'attribut route_id est 100110011:11 ?
2. Afficher la liste de toutes les lignes de trains.
3. Afficher la liste de toutes les lignes appartenant au réseau RER (le nom du réseau est donné par l'attribut _name de la table Agency).
4. Afficher le nombre de lignes de chaque type (donner deux colonnes : le numéro du type et le nombre de lignes).
5. Afficher la liste de toutes les lignes avec le nom et le type de ligne dans une deuxième colonne. Classer la liste par type de ligne.

Les trajets sont définis dans la table Trips dont l'attribut trip_id est la clé primaire.

```
MariaDB [tsi]> describe Trips;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| route_id      | varchar(30) | NO   |     | NULL    |       |
| service_id    | int(11)     | NO   |     | NULL    |       |
| trip_id       | varchar(18) | NO   | PRI | NULL    |       |
| trip_headsign | varchar(60) | NO   |     | NULL    |       |
| block_id      | varchar(30) | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
```

Exercice 78.

1. Combien y-a-t'il d'enregistrements dans la table Trips ?
2. Afficher les valeurs distinctes de l'attribut trip_headsign (indiquant le terminus) de la table Trips.
3. Quels sont les terminus de la ligne 800:P ?

L'attribut service_id de la table Trips est une clé étrangère qui associe cette table à la table Calendar.

La table Calendar2 permet de savoir quels jours le train circule.

```
MariaDB [tsi]> describe Calendar;
+------------+---------+------+-----+---------+-------+
| Field      | Type    | Null | Key | Default | Extra |
+------------+---------+------+-----+---------+-------+
| service_id | int(11) | NO   | PRI | NULL    |       |
| monday     | int(11) | NO   |     | NULL    |       |
| tuesday    | int(11) | NO   |     | NULL    |       |
| wednesday  | int(11) | NO   |     | NULL    |       |
| thursday   | int(11) | NO   |     | NULL    |       |
| friday     | int(11) | NO   |     | NULL    |       |
| saturday   | int(11) | NO   |     | NULL    |       |
| sunday     | int(11) | NO   |     | NULL    |       |
| start_date | date    | NO   |     | NULL    |       |
| end_date   | date    | NO   |     | NULL    |       |
+------------+---------+------+-----+---------+-------+
```

Exercice 79.

1. Quels jours a circulé un train dont l'attribut service_id est 11913 ?
2. Quel est le nom de la ligne où circule le train dont l'attribut trip_id est 128064984−1_132 ? Quel est son terminus ? Quels jours a fonctionné ce service ?

La table Trips permet de retrouver les étapes du parcours d'un train (ou d'un métro, ou d'un tramway) grâce à l'attribut trip_id qui est une clé étrangère de la table StopTimes.

La clé primaire de la table StopTimes est la paire (trip_id , departure_time).

```
MariaDB [tsi]> describe StopTimes;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| trip_id        | varchar(18) | NO   | PRI | NULL    |       |
| departure_time | varchar(8)  | NO   | PRI | NULL    |       |
| stop_id        | varchar(28) | NO   |     | NULL    |       |
| stop_sequence  | int(11)     | NO   |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
```

Exercice 80.

On s'intéresse au trajet dont l'attribut trip_id est 128064984−1_132.

1. Afficher les enregistrements de la table StopTimes correspondant à ce trajet classé dans l'ordre du parcours (l'attribut stop_sequence indique cet ordre).
2. À quelle heure est parti ce train ? À quelle heure est-il arrivé ?
3. Afficher le nombre d'arrêts.

On obtient le nom des arrêts grâce à la table Stops. Cette table indique aussi les coordonnées GPS et la zone de tarification de 1 à 5 (100 pour les zones hors île de France).

```
MariaDB [tsi]> describe Stops;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| stop_id   | varchar(28)  | NO   | PRI | NULL    |       |
| stop_name | varchar(69)  | NO   |     | NULL    |       |
```

```
| stop_lat   | decimal(9,6) | NO   |     | NULL    |       |
| stop_lon   | decimal(8,6) | NO   |     | NULL    |       |
| zone_id    | int(11)      | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

Exercice 81.

1. Comment sont associées les tables Stops et StopTimes ?
2. Quel est le point d'arrêt le plus à l'est ? Quelle est sa zone de tarification ?
3. Afficher le nombre de point d'arrêts dans chaque zone de tarification.
4. Afficher les noms de points d'arrêts dont la longitude est comprise entre 2,35 et 2,36 et dont la latitude est comprise entre 48,85 et 48,87.
5. Indiquer les étapes du parcours du train dont l'attribut trip_id est 128064984−1_132.

Exercice 82.

1. Sur quelle ligne se trouve le point d'arrêt StopPoint:59212 ? Quel est son nom ?
2. Déterminer tous les points d'arrêt dont le nom est 'Opéra'.
3. Déterminer toutes lignes passant par Opéra.

Exercice 83.

1. Décrire le trajet dont l'attribut trid_id de la table Trips est 127463895−1_311460.
2. En utilisant une auto-jointure, écrire une requête qui donne les paires de stations consécutives du trajet.