Exercice 1

Une base de données contient deux tables :

- Une table `mathematiciens` ayant pour attributs :
  - `nom` de type `VARCHAR()` (chaine de caractère), contenant son nom,
  - `id` de type `INT` servant d'identifiant, c'est la clé primaire,
  - `nationalite` de type `VARCHAR()`, contenant sa nationalité,
  - `siecle` de type `INT` contenant le siècle où il vécut.

Par exemple :

nom	id	nationalite	siecle
'Cauchy'	137	'france'	19

- Une table `theoremes` ayant pour attributs :
  - `nom` de type `VARCHAR()`, contenant le nom d'un théorème, c'est la clé primaire,
  - `id_auteur` de type `INT` contenant l'identifiant du mathématicien l'ayant établi,

Par exemple :

nom	id_auteur
'Théorème des résidus'	137

(Le théorème des résidus est un théorème démontré par le mathématicien français du XIX^e siècle Augustin Cauchy.)

On demande les requêtes SQL permettant d'obtenir les informations suivantes :

1. Le nom des théorèmes figurant dans la table `theoremes`.
2. Les noms des mathématiciens français du 20^e siècle, classés par ordre alphabétique.
3. Le nombre de mathématiciens pour chacune des nationalités.
4. La liste des théorèmes démontrés par Lagrange.

Exercice 2

Python dispose d'un module permettant d'implémenter les piles et leurs primitives. C'est le sous-module `deque` du module `collections`. On supposera avoir importé le module par la commande :

```
In[1]: from collections import deque
```

on pourra alors créer une pile à capacité illimité `pile` par la commande suivante :

```
In[2]: pile = deque()
```

La méthode `append` permet d'ajouter un élément au sommet de la pile ; il peut être de type quelconque :

```
In[3]: pile.append([1,2,3])
```

La méthode `pop` supprime et renvoie l'élément au sommet de la pile :

```
In[4]: pile.pop()
Out[4]: [1,2,3]
```

Utiliser les piles et leurs primitives données par le module `deque` décrit ci-dessus ci-dessus pour écrire une fonction `polonaise_inversee()` qui évalue à l'aide d'une pile une expression écrite en **notation polonaise inversée** qui sera passée en argument sous forme d'une liste.

Dans la notation polonaise inversée les 4 opérations arithmétiques ($+$, $-$, $\times$, $\div$) sont écrites à la suite des 2 opérandes auxquelles elles s'appliquent. Ainsi, par exemple :

$$2 \quad 3 \quad + \quad 4 \quad \times \qquad \text{s'évalue en } (2+3)\times 4 = 20$$

et sera passée en argument sous la forme de la liste `[2, 3, '+', 4, '*']`.

On se bornera à des nombres de type entier, et on rappelle que l'instruction `isinstance(x,int)` retourne `True` pour une donnée `x` de type entier, et retourne `False` sinon.

Exercice 3

On considère la fonction `mystere` suivante ; elle prend en paramètre deux entiers positifs `a` et `b` :

```
def mystere(a,b):
    c = 1
    while b != 0:
        if b % 2 == 1:
            c *= a
        a = a*a
        b = b//2
    return c
```

a) Justifier que le programme se termine ; pour celà exhiber un **variant de boucle**, c'est à dire une expression ne prenant que des valeurs entières et positives et dont la valeur diminue strictement à chaque passage dans la boucle. (On ne demande pas de justification).

b) Parmi les expressions suivantes laquelle est un **invariant de boucle** ? (On ne demande pas de justification).

1) `c + a ** b`
2) `c * a ** b`
3) `a + b * c`
4) `a - b * c`
5) `a * c`
6) `b * c`

c) Que représente pour `a` et `b` la valeur renvoyée par l'appel de `mystere(a,b)` ? Justifier la réponse donnée à l'aide de l'invariant de boucle trouvé à la question b) ; on notera pour cela a, b et c les valeurs initiales des variables `a`, `b` et `c` et a', b', c' les valeurs qu'elles prennent à l'issue de la boucle.

d) Ecrire le code d'une version récursive de la fonction `mystere`.

e) Quelle est sa complexité ?

Exercice 4
L'algorithme de tri rapide.

L'algorithme de tri rapide opère un tri (dans l'ordre croissant) sur une liste numérique `L` par récursivité en appliquant le principe suivant :

- Si la liste `L` contient au plus 1 élément, l'algorithme la renvoie tel quel ; c'est le cas terminal. Sinon :
- Choisir un élément `e` au hasard dans `L` par exemple `e = L[len(L)//2]` et le retirer de `L`.
- Constituer deux listes `L1` et `L2` : `L1` contiendra les éléments restants dans `L` qui sont inférieurs ou égaux à `e`, et `L2` contiendra les éléments restants dans `L` qui sont supérieurs ou égaux à `e`.
- Après appel récursifs du même procédé sur `L1` et `L2`, changer `L` en la concaténation de `L1`, de `[e]`, et de `L2`. On rappelle que l'opérateur `+` appliqué à deux listes retourne leur concaténation.

A la fin de l'algorithme la liste `L` sera triée. Ecrire une fonction <u>récursive</u> prenant en paramètre une liste numérique et qui la trie par l'algorithme de tri rapide.

Exercice 5
Calcul de déterminant.
Pour implémenter une matrice en `python` on utilisera une liste de listes numériques (les lignes), toutes ayant même longueur. Ainsi, par exemple :

`M = [[1,2], [3,4]]` implémente la matrice carrée $M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

et l'élément $M_{1,2}$ ligne 1 colonne 2 de la matrice M s'obtient grâce à `M[0][1]` (deuxième élément du premier élément (ligne) `M[0]` de la liste `M`).

1. Que fait la fonction suivante lorsque on lui passe en paramètre une "liste-matrice carrée" `M` et deux entiers `a` et `b` ? Quelle est sa complexité ?

```
def f(M,a,b):
    n = len(M)
    R = [ ]
    for i in range(n):
        if i != a:
            L = [ ]
            for j in range(n):
                if j != b:
                    L.append(M[i][j])
            R.append(L)
    return R
```

(2) Ecrire une fonction <u>récursive</u> prenant en paramètre une "liste-matrice" carrée et qui retourne son déterminant. On appliquera un développement sur la première ligne et on utilisera comme condition terminale le fait que le déterminant de la matrice carrée $(\alpha) \in \mathcal{M}_1(\mathbb{K})$ est α.

Exercice 6
On se propose d'écrire une fonction `EquaDiffLin2` prenant en paramètre :
six nombres `a`, `b`, `x0`, `xN`, `y0` et `z0`, et une fonction `f`,
et qui produit un graphique du tracé du graphe de la fonction $y : [x_0, x_N] \longrightarrow \mathbb{R}$ solution numérique approchée au problème de Cauchy d'ordre 2 :

$$(*) \quad \begin{cases} y'' + ay' + by = f(x) \\ y(x_0) = y_0 \\ y'(x_0) = z_0 \end{cases}$$

Pour le tracé <u>on prendra 100 points par unité de mesure</u>.

I. Résolution par la méthode d'Euler.

1. Après avoir posé $z = y'$, donner une système de deux équations différentielles du 1er ordre avec des conditions initiales pour y et z qui soient équivalents au problème de Cauchy $(*)$.
2. Donner le code de la fonction `EquaDiffLin2` permettant de résoudre ce système différentiel d'ordre 1 au dessus de l'intervalle $[x_0, x_N]$ par la méthode d'Euler et qui produit le tracé demandé.