# Levenstein vs. les Sup

On utilisera pour ce TP `Spyder`. On pourra télécharger les projets anonymisés des Sup ici.

## I Chargement des codes sources

Un code source est représenté par la liste de ses lignes. Une ligne de code source est une chaîne de caractère.

Par exemple, le code suivant :

```
def maximum(l) :
    m = -float("inf")
    for x in l :
        if x > m :
            m = x
    return m
```

Est représenté par la liste :

```
['def maximum(l) :', '    m = -float("inf")', '    for x in l :', '        if x > m :', '
            m = x', '    return m']
```

**Exercice 1.** *Chargement d'un code source.*

Écrire une fonction `chargement(chemin : str) -> list[str]` qui prend en entrée le chemin d'un fichier et qui renvoie la liste des lignes du fichier.

*N.B. : ici.*

**Exercice 2.** *Chargement des projets.*

Les projets anonymisés qui vous avez télécharger sont au nombre de `128`. Il sont numéroté à partir de 0 et le projet numéro `i` se nomme `proj_music_i.py` (`proj_music_0.py`, `proj_music_1.py`, ...).

Écrire une fonction `projets(nombre : int) -> list[list[str]]` qui prend en entrée le nombre de projets et qui renvoie la liste des codes sources des projets, comme obtenus par la fonction `chargement`. Le projet numéro `i` sera à l'indice `i` de la liste.

*On pourra utiliser* `str(n)` *pour convertir un entier* `n` *en sa représentation textuelle.*

## II Étude du problème

La distance de Levenstein entre deux codes sources `c1` et `c2` est définie comme le nombre minimal d'opération qu'il faut effectuer pour passer de `c1` à `c2` parmi :

— La suppression d'une ligne de `c1`.
— L'insertion d'une ligne quelconque dans `c1`.
— La substitution d'une ligne complète dans `c1`.

Chacune de ces opérations coûtera `1`, bien qu'elle consiste en la manipulation de *ligne entière*.

Le problème `Lev` est donc le calcul de la valeur de la distance de Levenstein sur deux codes sources données en entrées.

On fixe `c1` et `c2` deux codes sources, entrées du problème `Lev`. La taille des entrées consiste en le nombre de ligne de ces codes sources. Les sous problèmes auxquels nous allons nous restreindre sont les calculs de la distance de Levenstein sur les sous listes des *i* premières lignes de `c1` et des *j* première ligne de `c2`.

On les notes `L(i,j) = Lev(c1[:i],c2[:j])`.

**Exercice 3.** *Propriété de sous problèmes optimaux.*

- Exprimer `Lev(c1,c2)` en fonction de `L`.
- Donner une valeur pour `L(0,j)`.
- Donner une valeur pour `L(i,0)`.
- Proposer une relation de récurrence pour obtenir `L(i+1,j+1)` en fonction de
  - `L(i,j+1)`
  - `L(i+1,j)`
  - `L(i,j)`

  On pourra distinguer les cas `c1[i]` $=$ `c2[j]` et `c1[i]` $\neq$ `c2[j]`.

**Exercice 4.** *Chevauchement.*

Trouver deux sous problèmes dépendant pour les codes sources suivants :

```
def somme_liste(_X) :
    _X = 0
    for _X in _X  :
        _X = _X + _X
    return _X
```

```
def max_liste(_X) :
    _X = -float("inf")
    for _X in _X  :
        if _X < _X :
            _X = _X
    return _X
```

# III Distance de Levenstein

**Exercice 5.** *Distance de deux codes sources (descendante).*

Implémentez en version descendante efficace la fonction `levenstein(c1,c2)` qui prend deux codes sources sous la forme de la liste de leurs ligne et qui renvoie la distance de Levenstein de ces deux codes.

Calculez la complexité de votre fonction.

**Exercice 6.** *Distance de deux codes sources (montante).*

Implémentez en version montante efficace de la fonction `levenstein(c1,c2)` qui prend deux codes sources sous la forme de la liste de leurs ligne et qui renvoie la distance de Levenstein de ces deux codes.

Calculez la complexité de votre fonction.

**Exercice 7.** *Distance de tous les projets*

Implémentez une fonctions `diffs(projets)` qui prend en entrée la liste des projets telle que renvoyée par la fonction `projets`, et qui renvoie une matrice contenant à la ligne `i`, colonne `j`, la distance de Levenstein des codes sources numérotés `i` et `j`.

*N.B. : La matrice est symétrique à diagonale nulle.*

**Exercice 8.** *Extraction de données*

Pour les extractions si dessus, ne pas prendre en compte les 0 de la diagonale.

- Relevez la moyenne des distances des codes.
- Relevez la plus grande et la plus petite distance.
- Relevez les couples de projets qui ont distance plus petite que 50% de la moyenne.
- Relevez les couples de projets qui ont distance plus petite que 25% de la moyenne.
- Donnez une liste contenant à l'indice `i`, le numéro du code le plus proche du code `i`, ansi que la distance qui les sépare.