TD1
Exercices d'échauffements !

Les documents du TD sont accessibles à l'adresse suivante :

`https://github.com/Elieoriol/1920_CPES2_Python/TD1`

Rappel des principales fonctions ou méthodes pour lire et écrire dans un fichier.

- `os.chdir(rep)` pour spécifier le le répertoire `rep` où l'interpréteur Python va lire/écrire les fichiers,
- `f = open(fich,'w')` pour définir une variable `f` qui permet de créer et d'écrire dans le fichier `fich`,
- `f.write(chaine)` pour écrire la chaîne de caractères `chaine` dans le fichier référencé par `f`,
- `f.close()` pour fermer le fichier référencé par `f`,
- `f = open(fich,'r')` pour définir une variable `f` qui permet d'ouvrir le fichier `fich` en mode lecture,
- `b = f.read()` pour lire tout le fichier référencé par `f` et stocker son contenu dans la variable `b` de type `str`,
- `b = f.readline()` pour lire une ligne dans le fichier référencé par `f` et stocker cette ligne dans la variable `b` de type `str`,
- `b = f.readlines()` pour lire toutes les lignes du fichier référencé par `f` et les stocker dans la variable `b` de type liste de chaines de caractères,
- `f = open(fich,'a')` pour ouvrir le fichier `fich` en mode ajout et y accéder par l'intermédiaire de la variable `f`.

Exercice 1

La suite de Fibonacci est une suite mathématique définie comme suit :

$$\text{Fib}(0) = 0 \tag{1}$$
$$\text{Fib}(1) = 1 \tag{2}$$
$$\forall\, n \geq 2,\ \text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2) \tag{3}$$

1. Écrivez une fonction itérative qui construit la liste des 20 premiers termes de la suite de Fibonacci et l'écrit dans le fichier texte nommé `fib20.txt` (1 terme par ligne).

2. On a voulu écrire à la main ces 20 premiers termes dans le fichier `fib20_erreur.txt` mais une erreur s'est glissée dans nos calculs. Par comparaison à `fib20.txt`, écrivez un programme qui trouve l'index de la ligne à partir de laquelle la suite est fausse dans `fib20_erreur.txt`. Pouvez-vous donner également la valeur de l'erreur qui a été commise ?

3. Refaites la première question en écrivant dans le fichier `fib20_line.txt` les termes sur une seule ligne, séparés chacun par un espace, c'est-à-dire sous la forme : `"0 1 1 2 3..."`.

4. Le fichier `fib20_erreur_line.txt` suit le même modèle que `fib20_line.txt` mais contient une nouvelle erreur. Écrivez un programme qui trouve l'index de la ligne à laquelle l'erreur se trouve, ainsi que la valeur de cette erreur.

Exercice 2

1. Écrivez le programme prenant en argument une liste L et qui demande à l'utilisateur de saisir un élément x et supprime toutes les occurrences de cet élément dans L. Testez-le pour l'élément $x = 0$ et la liste $L = [0, 1, 2, 0, 0, 4]$. Évaluez la complexité de l'algorithme proposé dans le pire des cas.

2. Si cette complexité est plus grande que O(n), pouvez-vous trouver un algorithme en O(n) ? Demandez un indice à votre chargé de TD si vous ne voyez pas...

Exercice 3

Écrivez un programme faisant saisir une chaîne de caractères à l'utilisateur, inversant l'ordre des mots de cette chaîne et l'affichant. On supposera qu'il y a exactement un espace entre chaque mot.

Si l'utilisateur saisit :

```
"Ceci est un exercice de Python"
```

Le programme doit afficher :

```
"Python de exercice un est Ceci"
```

Exercice 4

Rappel Lorsque les arguments d'une fonction sont de type de base (`int`, `float`, `str`, `bool`), la passage des paramètres se fait **par valeur**. Par contre, si un argument est de type `list`, le passage des arguments se fait **par adresse** : toute modification de la valeur de l'argument modifiera également la valeur de la variable passée en paramètre.

1. Écrivez une fonction, appelée `filtrage()`, qui prend une liste Lr de réels en argument, et enlève de Lr tous les éléments négatifs.

2. Appelez cette fonction sur la liste $[-2.3, 5, -9, 0, 12.5, -6]$. Vous devrez afficher le résultat sous la forme suivante :

```
Liste avant filtrage : [-2.3,5,-9,0,12.5,-6]
Liste après filtrage : [5,0,12.5]
```

3. Après l'appel de la fonction `filtrage()`, la liste initiale n'existe plus, elle est remplacée par la liste filtrée. Modifiez la fonction `filtrage()` pour conserver en mémoire à la fois la liste initiale et la liste filtrée.

Exercice 5

On considère une séquence de n entiers relatifs $S = (a_1, \ldots, a_n)$. On appelle coupe de S toute sous-séquence non vide constituée d'entiers consécutifs. Ainsi, pour tout i, j avec $1 \leq i \leq j \leq n$, $(a_i, \ldots, a_j)$ est une coupe qu'on notera $S[i, j]$. La valeur d'une coupe $S[i, j]$ est définie par la somme des éléments qui la compose :

$$v_{i,j} = \sum_{k=i}^{j} a_k$$

Il s'agit de déterminer un algorithme efficace pour déterminer la coupe de valeur minimale.

Par exemple dans la séquence $S = (2, -6, 4, 5, -10, -3, 2)$, $S[5, 6]$ est la coupe de valeur minimale. Sa valeur est :

$$v_{5,6} = \sum_{i=5}^{6} a_i = -13$$

1. **Algorithme naïf**

   (a) Écrivez une fonction `somme()` prenant en paramètres une coupe et renvoyant sa valeur.

   (b) Écrivez une fonction `coupeMin1()` prenant en paramètre une séquence `S`, et qui, à l'aide de la fonction `somme()`, renvoie la valeur de la coupe minimale.

2. **Algorithme plus rapide**

   (a) Écrivez une fonction `coupei()`, qui prend en paramètre une séquence `S` et un indice `i` et qui renvoie la valeur minimale d'une coupe dont le premier élément est a_i. Cette fonction ne doit parcourir la séquence à partir de a_i qu'une seule fois. (Attention : cette fonction ne doit pas appeler la fonction `somme()`).

   (b) Écrivez une fonction `coupeMin2()` prenant en paramètres une séquence `S`, et qui, à l'aide de la fonction `coupei()`, renvoie la valeur de la coupe minimale dans `S`.

3. **Algorithme très rapide.**
   Étant donnée la séquence $S_i = (a_1, \ldots, a_i)$, on note v_i la valeur de la coupe minimale dans S_i et m_i la valeur minimale d'une coupe dans S_i se terminant par a_i. On a :

   $$m_{i+1} = \min(m_i + a_{i+1}, a_{i+1}) \quad \texttt{et} \quad v_{i+1} = \min(v_i, m_{i+1})$$

   Proposez une fonction `coupeMin3()` prenant en paramètre une séquence `S` et renvoyant la coupe de valeur minimale dans `S`. Cette fonction démarre avec $v = a_1$ et $m = v$, puis à chaque itération i, utilise la relation précédente pour calculer la coupe minimale.

4. Vous aurez besoin pour cette question d'importer les librairies `random` et `time`. Générez aléatoirement des séquences de taille 100 jusqu'à 1600 par pas de 500. À l'aide de la fonction `time.time()`, calculez les temps de calculs des trois fonctions. Votre affichage devra adopter le format suivant :

	Algorithme naif	Algorithme optimisé	Algorithme linéaire
100	tps en s	tps en s	tps en s
600	tps en s	tps en s	tps en s
1100	tps en s	tps en s	tps en s
1600	tps en s	tps en s	tps en s

   Que constatez-vous ?