TD1
Exercices d'échauffements !

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

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

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

- `os.chdir(rep)` pour spécifier 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.

Correction

1.
```
def fib(file, n):
    f = open(file, 'w')
    fib_tab = [0]
    f.write(str(0))
    if n >= 1:
        fib_tab.append(1)
        f.write("\n" + str(1))
    if n > 1:
        for k in range(2, n):
            fib_tab.append(fib_tab[-2] + fib_tab[-1])
            f.write("\n" + str(fib_tab[-1]))
    f.close()

fib('fib20.txt', 20)
```

   - La fonction `open` prend 2 arguments. Le premier est le chemin d'un fichier, qui doit être une chaîne de caractères (type `str`). Si j'écris `open(fib20.txt, 'w')` au lieu de `open('fib20.txt', 'w')`, le programme ne reconnaît pas un objet de type `str`, cherche alors une variable et renvoie l'erreur :

     `NameError: name 'fib20' is not defined`

     Le second argument de `open` est le mode d'ouverture : 'w' pour l'écriture (write), qui écrase ce qui existait auparavant dans le fichier, 'r' pour la lecture (read), 'a' pour l'ajout (add), c'est-à-dire l'écriture à la fin d'un fichier existant, au lieu d'écraser comme 'w'.
   - La fonction `f.write()` prend en argument une chaîne de caractères. On n'oubliera pas de convertir les autres types (`int`, `float`, `bool`...) en `str` sous peine de renvoyer l'erreur :

     `TypeError: write() argument must be str, not int`

     Exemples : `f.write(str(0)` ou `f.write("0")`
   - Lorsqu'on écrit dans un fichier, le retour à la ligne se fait par la chaîne de caractères "\n".
     Exemples : `f.write("\n hello")` ou `f.write("1\n")`
   - On peut accéder aux derniers éléments d'une liste en appelant les indices -1, -2, -3... Pour une liste L, L[-1] renvoie son dernier élément, L[-2] son avant-dernier élément, etc.

2.
```
def fibError(file, error_file):
    f1, f2 = open(file, 'r'), open(error_file, 'r')
    l1, l2 = f1.readlines(), f2.readlines()
    for k in range(len(l1)):
        if l1[k] != l2[k]:
            break
```

```
    f1.close()
    f2.close()
    return k, int(l2[k])-int(l1[k])

idx, error = fibError('fib20.txt', 'fib20_erreur.txt')
print("Index :", idx)
print("Erreur :", error)
```

- Lorsqu'on veut lire dans un fichier un fichier, le second argument de `open` est 'r' et non 'w' ou 'a'. L'appel d'une fonction de lecture comme `f1.readlines()` avec un mauvais mode renvoie l'erreur :

  `io.UnsupportedOperation: not readable`

- Pour lire dans le fichier, on aurait pu utiliser la fonction `f1.read()`, mais qui est moins pratique car renvoie tout le contenu du fichier sous forme d'une unique chaîne de caractères incluant les retours à la ligne sous la forme "\n". On aurait pu aussi lire les lignes une par une en itérant l'usage de `f1.readline()`, mais `f1.readlines()` a l'avantage de directement retourner une liste contenant toutes les lignes du fichier.
- A la fin de la fonction, pour calculer l'erreur commise, on fait la différence de `l1[k]` et `l2[k]`. Issus des listes renvoyées par `f1.readlines()` et `f2.readlines()`, ces deux éléments sont des chaînes de caractères. On n'oubliera donc pas de les convertir en `int` avant de faire leur différence, avec `int(l1[k])` et `int(l2[k])` sinon on a l'erreur :

  `TypeError: unsupported operand type(s) for -: 'str' and 'str'`

3.
```
def fibLine(file, n):
    f = open(file, 'w')
    fib_tab = [0]
    f.write(str(0))
    if n >= 1:
        fib_tab.append(1)
        f.write(' ' + str(1))
    if n > 1:
        for k in range(2, n):
            fib_tab.append(fib_tab[-2] + fib_tab[-1])
            f.write(' ' + str(fib_tab[-1]))
    f.close()

fibLine('fib20_line.txt', 20)
```

4.
```
def fibLineError(file, error_file):
    f1, f2 = open(file, 'r'), open(error_file, 'r')
    l1, l2 = f1.readline().split(' '), f2.readline().split(' ')
    for k in range(len(l1)):
        if l1[k] != l2[k]:
            break
    f1.close()
    f2.close()
    return k, int(l2[k])-int(l1[k])

idx, error = fibLineError('fib20_line.txt', 'fib20_erreur_line.txt')
print("Index :", idx)
print("Erreur :", error)
```

- Ici, tous les nombres de Fibonacci sont sur une même ligne. L'appel à `f1.readline()` renvoie cette ligne sous forme d'une chaîne de caractères, et pour obtenir la liste des nombres contenus dans la ligne, on utilise la méthode `split(' ')`. Cela sépare la ligne en tous ses éléments compris entre deux occurences de l'argument passé à `split()`, ici l'espace `' '`. **Exemples** : `"0 1 2".split(" ") -> ["0", "1", "2"]` ou `"bonjour".split("nj") -> ["bo", "our"]`

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...

Correction

1.
```
def occurencesDelete(L):
    x = int(input('Quelle valeur voulez-vous supprimer ? '))
    i = 0
    while i < len(L):
        if L[i] == x:
            L.pop(i)
        else:
            i += 1

L = [0,1,2,0,0,4]
occurencesDelete(L)

# Complexite en O(n^2) car n iterations et l'appel a pop est en O(n)
```

- L'utilisateur peut rentrer une chaîne de caractères dans le programme lorsque celui-ci croise une ligne avec la commande `input()`. Ici, on convertit le résultat retourné par `input()` en `int` car on veut que `x` soit de type `int` et non `str`.

```
def occurencesDelete2(L):
    x = int(input('Quelle valeur voulez-vous supprimer ? '))
    while x in L:
        L.remove(x)

L = [0,1,2,0,0,4]
occurencesDelete2(L)

# Solution alternative, mais aussi complexite en O(n^2) car au pire des cas, le
    "x in L" parcourt la longueur de la liste, et le "L.remove(x)" aussi
```

- La méthode `L.remove(x)` ne supprime que la première occurence de `x` dans `L`. On doit la répéter tant que la condition `x in L` est vérifiée. On est obligé de vérifier que `x in L` est vérifiée avant d'appeler `L.remove(x)` car si elle n'est pas vérifiée, c'est-à-dire que `x` n'est pas dans `L`, alors on a l'erreur :

```
ValueError: list.remove(x): x not in list
```

2.
```python
def occurencesDeleteON(L):
    x = int(input('Quelle valeur voulez-vous supprimer ? '))
    Lp = []
    for c in L:
        if c != x:
            Lp.append(c)
    return Lp

L = [0,1,2,0,0,4]
L = occurencesDeleteON(L)

# Complexite en O(n) car on parcourt une fois la liste
```

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"`

Correction

```python
def strReverse():
    s = input("Veuillez entrer une chaine de caracteres : ")
    split_s = s.split(' ')
    split_s.reverse()
    return ' '.join(split_s)

strReverse()
```

- La méthode `' '.join(split_s)` permet de réunir des chaînes de caractères contenus dans une liste, ici `split_s`, en les joignant en une unique chaîne de caractère par l'objet sur lequel `join()` est appelé, ici `' '`.
  Exemples : `" ".join(["0", "1", "2"]) -> "0 1 2"` ou `"blabla".join(["he", "ho"]) -> "heblablaho"`

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.

Correction

1.
```python
def filtrage(Lr):
    c = 0
    while c < len(Lr):
        if Lr[c] < 0:
            Lr.pop(c)
        else:
            c += 1

Lr = [-2.3,5,-9,0,12.5,-6]
print("Liste avant filtrage : ", Lr)
filtrage(Lr)
print("Liste apres filtrage : ", Lr)
```

2.
```python
def consFiltrage(Lr):
    f_Lr = []
    for x in Lr:
        if x >= 0:
            f_Lr.append(x)
    return f_Lr

Lr = [-2.3,5,-9,0,12.5,-6]
print("Liste avant filtrage : ", Lr)
cons_Lr = consFiltrage(Lr)
print("Liste apres filtrage : ", cons_Lr)
```

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 `coupe.i()`, 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 :

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

   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 ?

Correction

1.
```
def somme(c):
    return sum(c)

def coupeMin1(S):
    m = S[0]
    for i in range(len(S)):
        for j in range(i, len(S)):
            s = somme(S[i:j+1])
            if s < m:
                m = s
    return m

S = [2,-6,4,5,-10,-3,2]
coupeMin1(S)
```

2.
```
def coupei(S):
    m = S[0]
    s = m
    for k in S:
        s += k
        if s < m:
            m = s
    return m
```

```
def coupeMin2(S):
    m = S[0]
    for i in range(len(S)):
        mi = coupei(S[i:])
        if mi < m:
            m = mi
    return m

S = [2,-6,4,5,-10,-3,2]
coupeMin2(S)
```

3.
```
def coupeMin3(S):
    v = S[0]
    m = v
    for i in range(1, len(S)):
        m = min(m+S[i], S[i])
        v = min(v, m)
    return v

S = [2,-6,4,5,-10,-3,2]
coupeMin3(S)
```

4.
```
import time
import random
sizes = [100 + k*500 for k in range(4)]
print("\tAlgorithme naif\t\tAlgorithme optimise\tAlgorithme lineaire")
for s in sizes:
    seq = [random.randint(-10, 10) for k in range(s)]
    t0 = time.time()
    coupeMin1(seq)
    t1 = time.time()
    coupeMin2(seq)
    t2 = time.time()
    coupeMin3(seq)
    t3 = time.time()
    print(str(s)+"\t"+str(t1-t0)+"\t"+str(t2-t1)+"\t"+str(t3-t2))
```

On obtient par exemple, mais le résultat dépend des machines et de leur processeur :

	Algorithme naif	Algorithme optimisé	Algorithme linéaire
100	0.004987001419067383	0.0009968280792236328	0.0
600	0.6792821884155273	0.019952058792114258	0.0
1100	4.676499605178833	0.0857400894165039	0.0
1600	16.57237982749939	0.1495978832244873	0.0009982585906982422