python

https://sites.google.com/site/saiedanis87/

```
print("Hello, world!")
```

FORMATION PYTHON : LES FICHIERS

Ce chapitre a pour but d'initier à la manipulation des fichiers en Python.

said_anis
@hotmail
.com

Introduction

- Les variables sont simples à utiliser, mais elles ne contiennent que des informations **temporaires**.
- La durée de vie d'une variable n'est en effet jamais très longue. Or, vous aurez certainement besoin dans vos programmes de stocker des informations définitivement.
- Par exemple, il est impossible de stocker les résultats d’un programme dans des variables… puisque celles-ci seront supprimées à la fermeture du Shell Python !
- Par exemple, un petit jeu peut enregistrer les scores des joueurs.

Introduction

- Pour stocker ces informations longtemps, il faut les écrire sur le disque dur : créer des fichiers.
- Python permet d'enregistrer des données dans des fichiers.
- Donc, il est indispensable de savoir manipuler des fichiers avec PYTHON.
- On va s’intéresser seulement à la manipulation des fichiers texte contenant du texte (chaînes de caractères).
- Nous allons voir dans ce chapitre les fichiers : comment les ouvrir, les lire et écrire de dans.

Plan du cours

1. Introduction : Un fichier ? Types de fichiers ?
2. Fichier Texte
    1. Écrire des données dans un fichier
    2. Lecture des données depuis un fichier
3. Fichier binaire
    1. Écrire des données dans un binaire
    2. Lecture des données depuis un binaire

Introduction

Un fichier en informatique ! C’est quoi ?

Un fichier informatique est une collection d'informations numériques réunies sous un même nom, enregistrées sur un support de stockage permanent, appelé mémoire de masse, tel qu'un disque dur, un CD-ROM, ou une mémoire flash , et manipulées comme une unité.

Introduction

- En vue de faciliter leur organisation, les fichiers sont disposés dans des ***systèmes de fichiers*** qui permettent de placer les **fichiers** dans des emplacements appelés **répertoires** ou **dossiers**.
- Un fichier comporte un ***nom de fichier*** qui sert à désigner le contenu et y accéder.
- Ce nom comporte souvent un suffixe, **l´*extension***, qui renseigne sur la nature des informations contenues dans le fichier et donc des logiciels utilisables pour le manipuler.
- La nature du contenu peut être des *textes*, des *images*, de *l'audio* ou de la *vidéo*.

Introduction

Quelques catégories de fichiers :

Catégories de fichiers	Extensions
Exécutables (programmes exécutables)	.exe, .bat, ...
Compressés (fichier codé pour démeunier sa taille)	.zip, .rar, .gz, ...
Images (photos, diagrammes)	.gif, .jpg, .bmp, .png, ...
Audio (chanson, musique)	.wav, .ra, .ram, ...
Vidéo (film)	.avi, .mpg, ...

Introduction

Quelques catégories de fichiers :

Catégories de fichiers	Extensions
Documents : destinés à être imprimés et lus. Texte + informations de typographie (polices de caractères, couleurs)	.docx, .odt, .html, .doc, ...
Texte : texte *brut* sans indications de typographie. Exemple : code source ou bien des données pour un programme.	.txt, .html, .ini, .log, .conf, .c, ...

Fichier texte

Plan

1. Fichier Texte :
    1. Différence entre fichier texte brut et fichier binaire
    2. Écrire des données dans un fichier texte
    3. Lecture des données depuis un fichier texte

Fichier texte : Définition

Un fichier texte : [Définition]

- En informatique, un **fichier texte** ou **fichier texte brut** ou **fichier texte simple** est un fichier dont le contenu représente uniquement une suite de caractères;
- Les caractères considérés sont généralement :
  - des caractères imprimables,
  - d'espaces
  - et de retours à la ligne.

Fichier texte : Définition

Un fichier binaire: [Définition]

- Un fichier « non texte » est appelé « fichier binaire », dans le sens où les bits contenus dans le fichier ne représentent pas une simple suite de caractères imprimables, d'espaces et de retours à la ligne.
- On appelle « binaire » tout fichier qui n'est pas interprétable sous forme de texte : une image, un son ou encore un fichier compressé,...
- C'est aussi un terme spécifique à l'informatique où l'on part d'un code source (texte) pour générer un logiciel sous forme de fichier exécutable (binaire).

Fichier texte : Définition

Différence entre fichier texte et fichier binaire :

MyText.txt - Notepad

File Edit Format Help

```
PROCESS DATE:   9/15/03    BLOWFISH AND ASSOCIATES          REPORT ID:     AR100
RUN DATE....:   9/29/03    BILLING REPORT                   PAGE.....:         1


CUSTOMER NBR          NAME                DATE     AMOUNT
1001                  Anderson, Ed        90103    752.00
2002                  Garcia, Hector      90103    930.00
3003                  Takagaki, Mary      90103    102.55
4004                  Todd, William       90103    455.00
5005                  Daniels, Monica     90103    718.25
6006                  Bailey, Authur      90103     97.50
7007                  Sellers, Steve      90103    842.00
8008                  Baker, David        90103    471.00
```

```
0000000  0000 0001 0001 1010 0010 0001 000
0000010  0000 0010 0000 0028 0000 0010 000
0000020  0000 0001 0004 0000 0000 0000 000
0000030  0000 0000 0000 0010 0000 0000 000
0000040  0004 8384 0084 c7c8 00c8 4748 004
0000050  00e9 6a69 0069 a8a9 00a9 2828 002
0000060  00fc 1819 0019 9898 0098 d9d8 00d
0000070  0057 7b7a 007a bab9 00b9 3a3c 003
0000080  8888 8888 8888 8888 288e be88 888
0000090  3b83 5788 8888 8888 7007 778e 882
00000a0  d61f 7abd 8818 8888 407c 585f 881
00000b0  8b06 e8f7 88aa 8388 8b3b 88f3 88b
00000c0  8a18 880c e841 c988 b328 0871 088
00000d0  a948 5862 5884 7e81 3788 1ab4 5a8
00000e0  3d86 dcb8 5cbb 8888 8888 8888 888
00000f0  8888 8888 8888 8888 8888 8888 888
0000100  0000 0000 0000 0000 0000 0000 000
*
0000130  0000 0000 0000 0000 0000 0000 000
000013e
```

Hex Editor Neo

Source : http://www.hhdsoftware.com/free-hex-editor

Fichier texte : avant de commencer

- Au lancement de l'interpréteur, le répertoire de travail courant est celui dans lequel se trouve l'exécutable de l'interpréteur.
- Sous Windows, c'est C:\Python3X, le X étant différent en fonction de votre version de Python.
- Quand vous lancez un programme Python directement, par exemple en faisant un double-clic dessus, le répertoire courant est celui d'où vous lancez le programme.

```
>>> import os # os : operating system
>>> cur_dir = os.getcwd()# get current working directory
>>> print(cur_dir) #Afficher le répertoire courant
>>> os.chdir("C:/tests") # chdir : change directory
```

- Le répertoire **C:\tests** doit exister avant

Fichier texte : avant de commencer

Chemins relatifs et absolus

Les chemins absolus et relatifs sont deux moyens de décrire le chemin menant à des fichiers ou répertoires.

Pour décrire l'arborescence d'un système, on a deux possibilités :

1. les chemins absolus :

Un chemin absolu permet d'accéder à un endroit dans le disque quel que soit le répertoire de travail courant. Pour ce la, on décrit la suite des répertoires menant au fichier. Sous Windows, on partira du nom de volume (C:\, D:\…).

Exemple : C:\test\mon_fichier.txt

Remarque :

- Windows utilise des antislashs « \ » pour décrire l'arborescence de ses répertoires
- Mais, dans le code python, on utilise un slash « / » ou 2 antislashs « \\ »

Fichier texte : avant de commencer

2. Chemins relatifs :

- Quand on décrit la position d'un fichier grâce à un chemin relatif, cela veut dire que l'on tient compte du dossier dans lequel on se trouve actuellement.
- Quand on décrit un chemin relatif, on utilise parfois le symbole **..** (deux points) qui désigne le répertoire parent.
- Voici quelques exemples : On souhaite accéder à **fichier1.txt**

1. Si on se trouve dans le dossier **C:\test** notre chemin relatif sera donc : **rep1\fichier1.txt**
2. Si le répertoire de travail courant est **rep2** , notre chemin relatif sera donc : **..\rep1\fichier1.txt**
3. Maintenant, si on se trouve dans **C:** notre chemin relatif sera **test\rep1\fichier1.txt**

- C:
  - test
    - rep1
      - fichier1.txt
    - rep2
      - fichier2.txt
      - fichier3.txt

Fichier texte

Un fichier texte peut être utilisé en 3 modes :

1. Le mode Ecriture (mode write : w)
2. Le mode Ecriture à la fin (mode append : a)
3. Le mode Lecture (mode read : r)

Écrire des données dans un fichier texte

- Pour écrire dans un fichier, on commence par
  - l'ouvrir en écriture avec l'option « w » (pour « *write* »):
  - Ce qui aura pour effet de l'écraser, s'il existait déjà, sinon il sera crée !

```
>>> f= open ("fichier.txt", "w") # Ouvrir un fichier en écriture
```

  - Les données ne seront réellement enregistrées qu'après la fermeture de fichier par l'exécution de f.close ()

```
>>> f.close()
```

- Vérifiez le contenu de **fichier.txt** dans le répertoire courant après chaque modification !

Écrire des données dans un fichier texte

- Ensuite, on y écrit des chaînes de caractères via la méthode **write.**
- La méthode **write (chaine)** n'accepte en paramètre que des chaînes de caractères et renvoie le nombre de caractères qui ont été écrits. (Pour écrire des entiers, il faut les convertir en type 'str' et pour les lires les reconvertir en 'int')

```
>>> f.write("Première ligne écrite dans un fichier via Python ")
>>> f.write ("Premier ligne\n Deuxième ligne\t après 4 espaces")
```

- **writelines()** permet d'écrire directement le contenu d'une liste dans le fichier.

```
>>> f.writelines (["Premier ligne\n", "Deuxième ligne\n"])
>>> f.close()
```

Écrire des données dans un fichier texte

Exercice n° 1 :

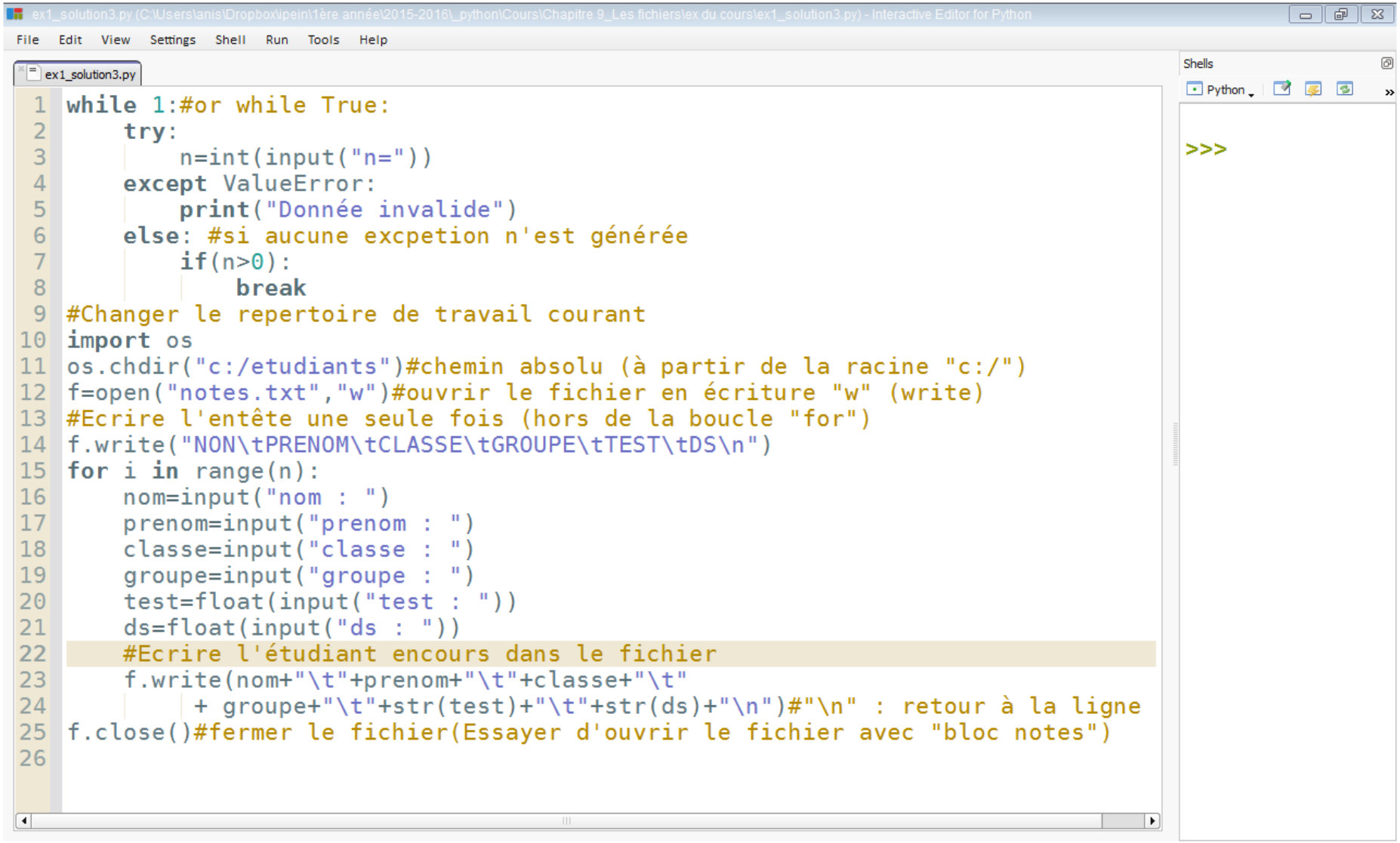
Écrire un programme python qui permet de lire le nom, le prénom, la classe, le groupe, la note de test et la note de de DS de n étudiants et les enregistrer dans un fichier texte nommé notes.txt dans le répertoire c:\etudiants.

Exemple :

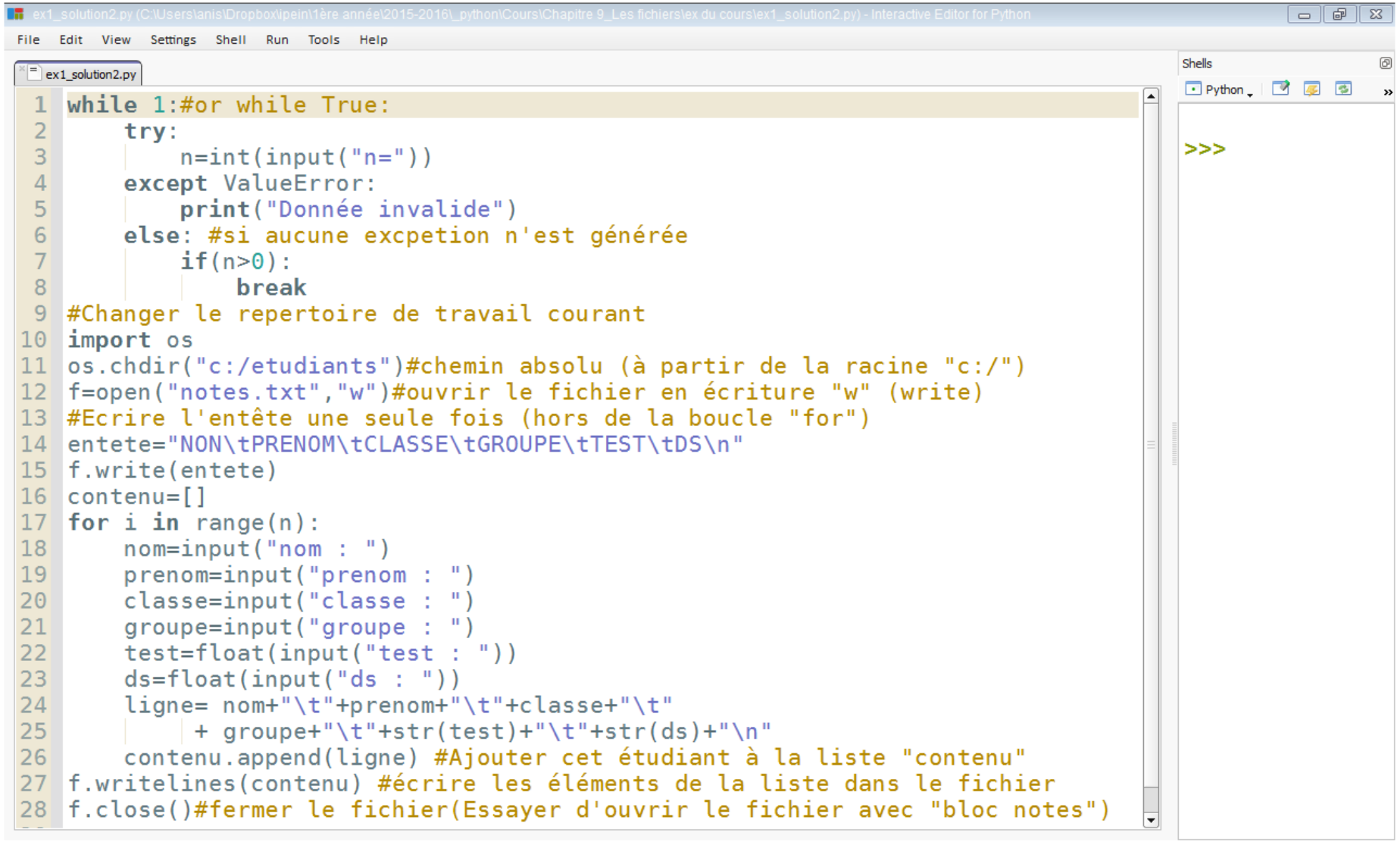
Nom	Prénom	Classe	Groupe	TEST	DS
BAHIJ	CHAHER	T3	A	19.00	15.50
BALLOUM	EYA	M3	A	10.50	16.00
MOHAMED	HACHEM	T4	B	14.00	16.00
ATOUI	OUMAYMA	P3	A	17.00	14.50
AMAIDI	WAFA	M4	A	17.50	18.00

Durée estimée : 10 min

Écrire des données dans un fichier texte



Écrire des données dans un fichier texte



Lecture des données depuis un fichier texte

On peut lire le contenu de fichier de deux manières : Lire le fichier ligne par ligne ou lire tout le fichier en bloc;

1. Lire l'intégralité de fichier : méthode **read()**

Exemple :

```
>>> f = open('fichier.txt','r') # r (read) : mode lecture
# et pointer le curseur au début du fichier
>>> contenu = f.read() #lire l'intégralité de fichier (le curseur sera positionné à la fin du fichier)
>>> print(contenu) #essayer de répéter les deux dernières lignes.
>>> f.close()#libérer les ressources non utilisées.
```

Lecture des données depuis un fichier texte

2. Lire le fichier ligne par ligne :

- Soit pas à pas via la méthode **readline( )**, qui lit la ligne courante et passe à la suivante,
- Soit globalement via la méthode **readlines( )**, qui crée l'itérateur de toutes les lignes (liste) et permet ainsi de traiter ces dernières dans une boucle. Le contenu du fichier est donc stocké dans une liste où chaque élément de la liste (de type chaîne de caractères) correspond à une ligne du fichier.

Exemple :

```
>>> f= open('fichier.txt','r') #'r' : ouvrir en mode lecture et
placer pointeur au début de la première ligne du fichier
>>> print(f.readline()) #Affiche la ligne en cours et placer le
pointeur au début de la ligne suivante (essayez encore)
>>> s = 0
>>> for ligne in f.readlines(): # méthode 1 (parcourir une liste)
...     s += len(ligne) # s = nombre de caractères dans f
>>> for ligne in f: print(ligne) # méthode 2
>>> f.close()
```

Ecrire à la fin dn fichier texte

- Pour écrire à la fin d'un fichier, on utilise la fonction open en mode ajout (append), donc contrairement au mode écriture, le contenu du fichier ne sera pas écrasé.

Exemple :

```
>>> f= open('fichier.txt','a') #'a' : mode append
```

- La ligne suivante sera ajoutée au fichier sans écrasement de son contenu

```
>>> f.write("\nnouvelle ligne ajoutée en mode append")
>>> f.close() # Enregistrer les informations et fermer le fichier
```

Extraire les données d'une ligne dans des variables

- La méthode **split( )** permet de séparer des informations dans les lignes lues selon un caractère qui lui est passé en paramètre et de retourner une liste de chaînes de caractères.

Exemple :

```
>>> 'NOM,PRENOM,Classe,DS'.split(',')#['NOM', 'PRENOM', 'Classe', 'DS']
```

- La méthode **strip( )** permet d'enlever les **espaces** et **caractères d'échappement** (tabulation « \t », un retour à la ligne « \n »,…) en début et fin de ligne.

Exemple :

```
>>>' \n \t une \t chaine\n'.strip()#Suppression en début et fin de ligne
        'une \t chaine'
>>>' \n \t une \t chaine \n'.lstrip()# Suppression en début de ligne
         'une \t chaine \n'
>>>' \n \t une \t chaine \n'.rstrip()# Suppression en fin de ligne
         ' \n \t une \t chaine'
```

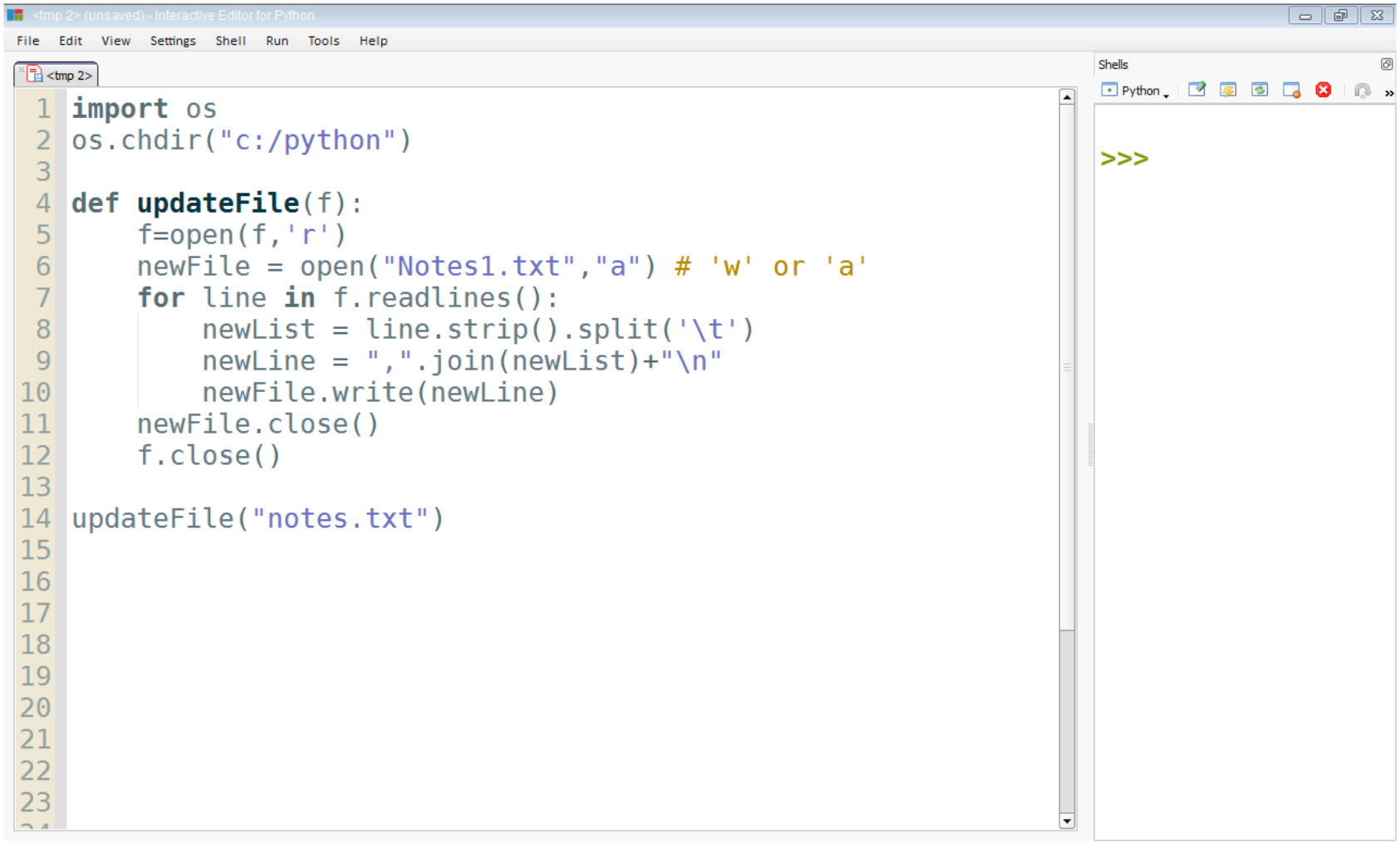
Exercice N°2 :

Écrire un programme python qui permet d'ouvrir en lecture le fichier **notes.txt** situé dans le répertoire **c:\etudiants** et extraire les différentes informations situées dans chaque ligne du fichier dans les variables suivantes : *nom, prénom, classe, groupe, noteTest et noteDS* et les enregistrer dans un nouveau fichier **notes1.txt** dans lequel on écrit les mêmes données, mais formatées différemment : Les données sont séparées par des ***virgules*** au lieu de ***tabulations***.

Exemple : Contenu du fichier : notes1.txt après modifications

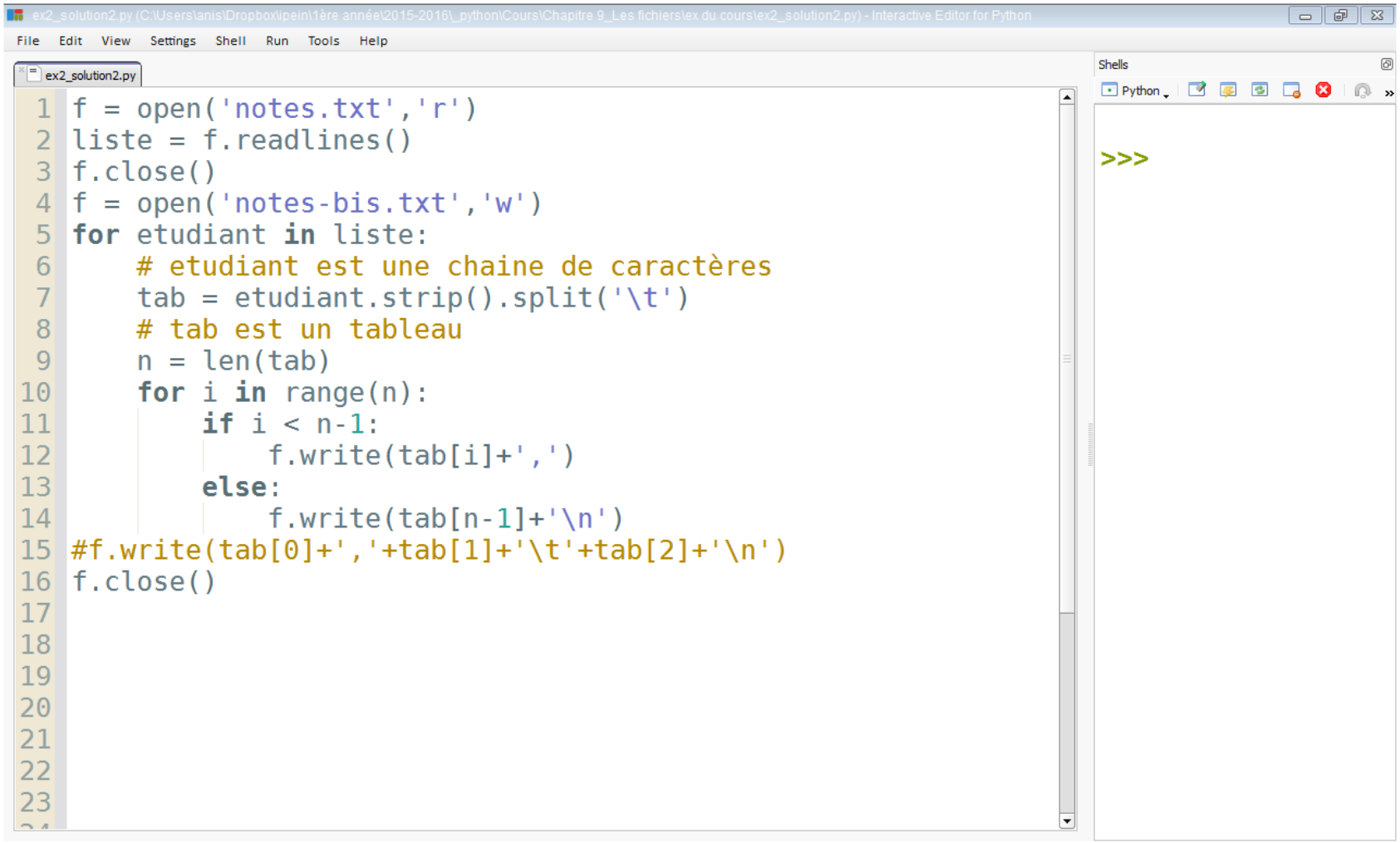
```
Nom,Prénom,Classe,Groupe,TEST,DS
BAHIJ,CHAHER,T3,A,19.00,15.50
BALLOUM,EYA,M3,A,10.50,16.00
MOHAMED,HACHEM,T4,B,14.00,16.00
ATOUI,OUMAYMA,P3,A,17.00,14.50
AMAIDI,WAFA,M4,A,17.50,18.00
```

Exercice N°2 : Solution 1



said_anis@hotmail.com

Exercice N°2 : Solution 2



Gérer les erreurs lors de manipulation des fichiers

- Lors de la manipulation des fichiers, il se peut qu'une erreur se produit quand on lit, ou quand on écrit…
- La commande **with** permet de fermer le fichier à la fin du bloc des instructions ou si une exception se produit.

Exemple :

```
>>> with open("fichier.txt","r") as f: #'r':mode lecture
...     s=f.read()
...     print(s)
```

- Cela signifie simplement que, si une exception se produit, le fichier sera tout de même fermé à la fin du bloc.
- Il est inutile, par conséquent, de fermer le fichier à la fin du bloc with. Python va le faire tout seul, qu'une exception soit levée ou non.
- Il est recommandé d'utiliser cette syntaxe, elle est plus sûre et facile à comprendre.

Fichier binaire

1. Fichier binaire :
    1. Écrire des données dans un fichier binaire
    2. Lecture des données depuis un fichier binaire

Écrire des données dans un fichier binaire

Sérialisation des données avec le module Pickle

- Le module Pickle est extrêmement pratique pour sauvegarder dans un fichier des structures de données comme les listes (type list) ou les dictionnaires (type dict).

Exemple : sauvegarde d'un dictionnaire :

```
>>> import pickle
# création d'un dictionnaire
>>> coefficients = {"coeffanalyse":10,"coeffalgebe":5,"coeffchimie":5}
# enregistrement du dictionnaire dans un fichier
>>> f = open("data.txt","wb")#ouvrir le fichier en mode écriture binaire
# sérialisation: écrire les données sous frome binaire
>>> pickle.dump(coefficients,f)
>>> f.close()
```

- Vérifiez le contenu du fichier data.txt !

Lire les données depuis un fichier binaire

Désérialisation des données avec le module Pickle

Exemple : récupération de contenu du dictionnaire :

```
>>> import pickle
# récupération du dictionnaire contenu dans le fichier data.txt
>>> f = open("data.txt","rb")#ouvrir le fichier en mode lecture binaire
 # dé sérialisation: récupérer les données sous leur forme normale
>>> coefficients = pickle.load(f)
>>> f.close()
>>> print(coefficients)
```

Exercice N°3 : fichiers binaires

Écrire un script python qui permet de définir et d'appeler les fonctions suivantes :

1. Créer une fonction **saisie_listes ( )** qui saisie et retourne une liste de n sous-listes chacune de m entiers positifs (m>4). (1**0 min)**
2. Créer une fonction **remplir(fichier, listes)** qui enregistre les n listes dans un fichier binaire nommé fichier1.txt dans le répertoire C:\python\scripts. (5 **min)**
3. Créer une fonction **trierListesFichier(fichier_source, fichier_distination)** qui permet de trier les entiers de chaque liste (ligne) dans l'ordre croissant et enregistrer les résultats dans le fichier binaire C:\python\scripts\fichier2.txt. (1**0 min)**
4. Créer une fonction **trierLignesFichier(fichier_source, fichier_distination)** qui permet de trier les listes d'entiers du fichier binaire C:\python\scripts\fichier2.txt dans l'ordre croissant selon le premier entier de chaque liste et enregistrer les résultats dans le fichier binaire C:\python\scripts\fichier3.txt. (1**0 min)**
5. Créer une fonction **afficher(fichier)** qui affiche le contenu de fichier binaire fichier3.txt en mode naturel (décimales). (5 **min)**
6. Créer une fonction **recherche (n,fichier)** qui retourne les numéros de lignes et numéros de colonnes de chaque occurrence de l'entier n dans le fichier3.txt ou bien elle retourne None s'il n'existe pas. (1**0 min)**

Exercice N°3 : Solution 1

said_anis@hotmail.com