TP 6 : Entrées/sorties: Lecture/écriture de fichiers

1 Fonctions pour les fichiers

On a vu au TP 4 les fonctions `print` et `input` pour écrire à l'écran ou demander à l'utilisateur d'entrer quelque chose au clavier. En pratique, on utilise assez peu les fonctions `print` et `input` depuis un clavier ou vers l'écran : si on veut traiter une grande quantité de données (par exemple pour faire une étude statistique pour un TIPE...), ces données sont en général stockées dans des fichiers ad hoc, qu'on veut pouvoir lire pour ensuite en manipuler le contenu. Il est assez maladroit de copier le contenu du fichier dans un script Python, il vaut mieux séparer les données du script qui les transforme. En particulier, même si les données changent (à la suite d'une nouvelle série de mesures, par exemple), le script Python reste identique. Après manipulation, on peut ensuite vouloir réécrire nos données transformées vers un autre fichier. Cette sous-section décrit la manipulation des entrées/sorties vers des fichiers, qui existent en dehors de notre programme Python.

Du point de vue du programmeur, un fichier ouvert *en lecture* doit être vu comme un tube (*pipe* en anglais) par lequel arrivent des données extérieures chaque fois que le programme les demande, de même qu'il faut voir un fichier ouvert *en écriture* comme un tube par lequel s'en vont les données que le programme envoie. Remarquez qu'ainsi, le clavier ou l'écran ne sont que des tubes particuliers.

Pour les fonctions Python, un fichier est une séquence de caractères : une fois qu'un fichier a été ouvert par un programme, celui-ci maintient un marqueur (fictif) à la position courante, qui indique à tout moment où sera lu/écrit le prochain octet. Toute opération de lecture ou d'écriture fait bouger ce pointeur vers l'avant.

Voici les principales fonctions pour le traitement des fichiers. Dans ce qui suit, `f` désigne un tube, qu'on pourra considérer comme étant un fichier ouvert en lecture ou en écriture.

fonction	desciption
`f=open(nom_du_fichier,'r')`	Ouvre le fichier `nom_de_fichier` (donné sous la forme d'une chaîne de caractères indiquant son emplacement) en lecture (`r` comme read). Le fichier doit exister et seule la lecture est autorisée.
`f=open(nom_du_fichier,'w')`	Ouvre le fichier `nom_de_fichier` en écriture (`w` comme write). Si le fichier n'existe pas, il est créé, sinon il est écrasé (vidé avant utilisation).
`f=open(nom_du_fichier,'a')`	Ouvre le fichier `nom_de_fichier` en ajout (`a` comme append). Identique au mode `'w'`, sauf que si le fichier existe, il n'est pas écrasé et ce qu'on écrit est ajouté à partir de la fin du fichier.
`f.close()`	Sur un fichier ouvert comme précédemment, le ferme. Cette ligne est impérative pour les fichiers ouverts en écriture, puisque le fichier n'est réellement écrit complètement qu'à la fermeture.
`f.read()`	Lit tout le fichier d'un coup et le renvoie sous forme de chaîne de caractères (à ne réserver qu'aux fichiers de taille raisonnable).
`f.readlines()`	Pareil que précédemment, mais le résultat est une liste de chaînes de caractères, avec un élément de la liste par ligne. Attention, le saut de ligne `\n` est présent à la fin de la ligne.
`f.readline()`	Lit une unique ligne du fichier et la renvoie sous forme de chaîne (avec `\n` au bout). Le curseur de lecture (virtuel!) est positionné au début de la ligne suivante. Si on est arrivé en fin de fichier, le résultat est une chaîne vide `""`.
`f.write(s)`	Écrit la chaîne `s` à la suite du fichier.
`f.writelines(T)`	Écrit l'ensemble des éléments de `T` dans le fichier `f` comme des lignes successives. `T` est une liste, une séquence, un tuple... bref, un itérable. Il faut que le marqueur de saut de ligne `\n` soit présent à la fin des éléments de `T` pour que ce soit effectivement des lignes successives qui soient écrites.

Enfin, un fichier ouvert en lecture peut-être vu comme un itérable : on peut parcourir ses lignes avec une boucle `for` sans passer par `readlines` :

```
f=open('nom_du_fichier','r')
for ligne in f:
    [instructions]
```

2 Exercices de mise en pratique

Dans les exercices qui suivent, on ouvrira les fichiers comme expliqué ci-dessus, on ne recopiera pas les données dans un fichier Python. Petit rappel : les données vont être récupérées sous forme de chaînes de caractères. Il faudra parfois utiliser des fonctions de conversions pour les traiter (`int(x)` renvoie un entier, `float(x)` un flottant). Inversement, `str(x)` renvoie la conversion de l'objet `x` en chaîne de caractères.

Prérequis. Téléchargez l'archive `TP6.zip` sur le site web. Si ce n'est pas déja fait, créez un répertoire dédié au TP, positionnez l'archive dedans et dézippez là, et créez un script Python (`TP6.py`, par exemple) dans le répertoire. Ouvrez-le avec `Pyzo`.

Rappel sur les chaînes de caractères. On rappelle que `+` permet la concaténation de chaînes de caractères. Deux méthodes utiles sur les chaînes : `split` et `join` :

```
>>> "abc def ghi   klmno pq".split()
['abc', 'def', 'ghi', 'klmno', 'pq']
>>> "abbbabbbbbbaba".split("a")
['', 'bbb', 'bbbbbb', 'b', '']
>>> "".join(["abc", "def", "ghij"])
'abcdefghij'
>>> " xx ".join(["abc", "def", "ghij"])
'abc xx def xx ghij'
```

`s.split(c)` permet de séparer une chaîne `s` autour d'un caractère particulier `c` (par défaut le caractère d'espacement `' '`), tandis que `c.join(L)` permet de concaténer les éléments d'une liste de chaînes avec `c` comme séparateur. Dans les deux cas un objet est *renvoyé*.

Le caractère spécial `\n` (saut de ligne) compte comme un seul caractère, et si la chaîne `s` termine par `\n`, alors `s[:-1]` est une chaîne constituée des mêmes éléments que `s`, sans `\n` (c'est utile dans la suite).

Exercice 1. *Fichier contenant des entiers.* Le fichier `liste_entiers.txt` contient une liste d'entiers (un seul par ligne). Faire la somme (vous devez trouver `5301585393018010060 3592`).

Exercice 2. *Fichier contenant des flottants.* Le fichier `liste_couples.txt` contient des lignes de la forme `x;y` où `x` et `y` sont des flottants. Tracer le graphe correspondant aux `x` en abscisse et `y` en ordonnée. Petit rappel d'utilisation de `matplotlib` :

```
import matplotlib.pyplot as plt
plt.plot(X,Y) #pour tracer
plt.show() #pour afficher
```

où `X` et `Y` sont deux listes de flottants de même taille. On pourra utiliser `s.split(';')` pour séparer une chaîne autour du caractère `;`. Le résultat est une liste de chaînes.

Exercice 3. *Écriture dans un fichier.* Écrire une fonction `creation(f,n)` prenant en entrée un fichier `f` ouvert en écriture, et un entier `n`, et écrivant dans le fichier les entiers de 1 à n (un entier par ligne). En fin de fonction, le fichier est refermé. Par exemple, le script suivant crée un fichier `essai.txt` contenant les entiers de 1 à 1000 :

```
h=open('essai.txt','w')
creation(h,1000)
```

Indication : il est nécessaire d'utiliser la méthode `write` pour écrire dans un fichier. Ne pas oublier de convertir les entiers en chaînes et de rajouter des sauts de ligne !

3 Exercices plus conséquents

En période de conseils de classe, il est fatigant de remplir des bulletins : il vaut mieux tout générer automatiquement.

Exercice 4. *Génération des notes du trimestre.* Pour éviter le travail fastidieux de la correction de copies, il vaut mieux choisir les notes au hasard. La fonction `randint` du module `random` prend en entrée deux entiers a et b et retourne un entier aléatoire de l'intervalle $[\![a, b]\!]$.

1. Écrire une fonction `genere_note()` sans argument, retournant une note entre 0 et 20. Celle-ci sera entière ou demi-entière (comme 7 ou 15.5). Indication : il suffit de générer un entier qu'on divisera par 2 pour obtenir un demi-entier.
2. Écrire une fonction `genere_notes(f1,f2)` prenant en entrée un fichier ouvert en lecture et un en écriture, tel que si les lignes de `f1` sont de la forme `nom,prenom`, la fonction écrive dans le fichier `f2` des lignes de la forme `nom,prenom,note1,note2`, avec `note1` et `note2` des notes générées aléatoirement.

Testez votre script avec le fichier `classe.txt` (ce sont les PC* de la promotion 2014), pour produire `notes_eleves.txt` :

```
f=open('classe.txt','r')
g=open('notes_eleves.txt','w')
genere_note(f,g)
f.close()
g.close()
```

Exercice 5. *Calcul de la moyenne.* Il faut maintenant rentrer les moyennes sur Pronote. Avant ça, on va les calculer.

1. Écrire une fonction `moyenne(note1,note2)` prenant en entrée deux notes et retournant la moyenne de ces deux notes.
2. Écrire un script `moyennes(f1,f2)`, fonctionnant comme précédemment. Les lignes de `f1` sont de la forme `nom,prenom,note1,note2`, les lignes du fichier de sortie `f2` de la forme `nom,prenom,moyenne`, où `moyenne` est bien sûr la moyenne des deux notes.

Testez votre script avec le fichier `notes_eleves.txt`, pour produire `moyennes_eleves.txt`.

Exercice 6. *La moyenne de la classe.* Écrire une fonction `moyenne_classe(f)` prenant en entrée un fichier ouvert en lecture de la même forme que `moyennes_eleves.txt` créé à la question précédente, et affichant à l'écran la moyenne de la classe.

Exercice 7. *Rang.* Reprendre en lecture un fichier de la forme `moyenne_eleves.txt` pour produire `moyenne_rang.txt`. Le fichier contient des lignes de la forme `nom,prenom,moyenne,rang`. Pour obtenir le rang, il est nécessaire de former une liste contenant les moyennes et de trier cette liste. On utilisera pour cela la *méthode* `sort` pour les listes. Si `L` est une liste, `L.sort()` modifie la liste de façon à la trier.

On peut préciser la façon de trier la liste en rajoutant un argument à la méthode, ce qui est utile ici. Par défaut, si `L` contient des chaînes de caractères, la liste va être triée par ordre alphabétique. Par exemple :

```
>>> L=["de", "abc"]
>>> L.sort()
>>> L
["abc", "de"]
```

C'est le cas aussi si la liste contient des choses plus complexes, comme des couples (chaîne de caractères, entier).

```
>>> L=[("bc", 0), ("a", 8), ("a", 4)]
>>> L.sort()
>>> L
[('a', 4), ('a', 8), ('bc', 0)]
```

La liste est triée d'abord par l'ordre naturel sur le premier élément de chaque couple, puis en regardant le deuxième élément du couple en cas d'égalité. On peut changer ce comportement par défaut en donnant une *fonction clé* en paramètre :

```
>>> def f(x): #fonction de tri: l'opposé de la deuxième composante du couple (qui doit être un entier).
...  return -x[1]
...
>>> L.sort(key=f)
>>> L
[('a', 8), ('a', 4), ('bc', 0)]
```

Pour le calcul du rang, on ne s'embêtera pas avec les cas d'égalité dans les moyennes. Indications :

— former les triplets (nom, prenom, moyenne), à partir du fichier initial ;
— trier une première fois par moyenne décroissante ;
— rajouter les rangs ;
— trier à nouveau dans l'ordre alphabétique les triplets (le tri par défaut fait l'affaire) ;
— formet le fichier final.

4 Un projet plus complexe : création automatique d'un trombinoscope en HTML

On souhaite créer automatiquement un trombinoscope. On dispose d'un dossier contenant entre autres des photos au format `jpg`, par exemple les photos d'une classe. On souhaite créer automatiquement un fichier `trombinoscope.html` ressemblant à ceci :

```
<html><head><title>Trombinoscope de la classe</title></head>
<body>
<h1 align='center'>Trombinoscope de la classe</h1>
<table cols=4 align='center'>
<tr>
<td><img src="Abega_Thomas.jpg" width=150 height=200/><br/>Thomas Abega</td>
<td><img src="Akouka_Liora.jpg" width=150 height=200/><br/>Liora Akouka</td>
<td><img src="Armaut_Elisabeth.jpg" width=150 height=200/><br/>Elisabeth Armaut</td>
<td><img src="Bignotti_Romain.jpg" width=150 height=200/><br/>Romain Bignotti</td>
</tr>
<tr>
<td><img src="Tocut_Laetitia.jpg" width=150 height=200/><br/>Laetitia Tocut</td>
<td><img src="Vanbemmel_Thibaud.jpg" width=150 height=200/><br/>Thibaud Vanbemmel</td>
<td><img src="Vial_Nicolas.jpg" width=150 height=200/><br/>Nicolas Vial</td>
</tr>
</table>
</body>
</html>
```

Ouvert dans un navigateur (Firefox par exemple), ce fichier donne bien le trombinoscope de la classe, si les photos sont dans le même répertoire que le fichier. Le nombre de lignes dépend du nombre d'images dans le répertoire, qui ont pour forme `Nom_Prenom.jpg`. Le but de cette section est de créer automatiquement le trombinoscope à l'aide d'un script Python. Outre la manipulation de fichiers présentés à la section précédente, vous pourrez utiliser le module `os` pour lister les éléments présents dans le répertoire courant :

```
>>> import os
>>> os.listdir() #placez votre script Python directement dans le répertoire des photos !
['Callea_Elodie.jpg', 'Negro_Andrea.jpg', 'Loegel_Olga.jpg', 'Bignotti_Romain.jpg' ...]
```

La sortie, trop longue à afficher, à été tronquée. Mais en tout cas on peut donner la liste des éléments présents dans le répertoire à l'aide de `os.listdir()`, le format de retour est une liste de chaînes de caractères. Débrouillez-vous pour créer automatiquement un fichier `trombinoscope.html` de la forme précédente :

- la liste des fichiers d'extension `jpg` doit être récupérée automatiquement avec `listdir` ;
- le script fonctionne quel que soit le nombre de photos présentes dans le dossier `photos`.
- toutes les lignes sont écrites par le script, en particulier celles de la forme `<td><img src=...</td>` dépendent de la photo examinée. Remarquez qu'on part des photos `Nom_Prenom.jpg` et qu'on écrit `Prenom Nom` à droite ;
- faites attention à fermer/ouvrir les balises `<tr>` toutes les 4 photos ;
- si la dernière ligne en contient moins de 4, veillez quand même à ne pas oublier la balise fermante `</tr>` ;
- utiliser `T.sort()` sur une liste `T` composé de chaînes de caractères permet de le trier par ordre alphabétique. Le trombinoscope doit être ordonné alphabétiquement (suivant le nom).

Exercice 8. Écrire le script et tester avec l'archive `photos_eleves.zip`. Ce sont les MPSI de l'année 2015/2016 (mais ce ne sont pas leurs têtes !)