Chapter 1

Données en table

1.1 Manipuler les fichiers

1.1.1 Ouverture d'un ficher

Noms de fichier: bonnes pratiques

Nous nous tiendrons aux consignes données dans le cours SHELL/système. Les caractères autorisés pour les noms de fichier sont:

- Les caractères alphanumériques.
- Le tiret-bas, underscore ou tiret du 8.
- Le trait d'union sauf en début de nom.

✎ Tout le reste est proscrit.

Ecriture dans un ficher

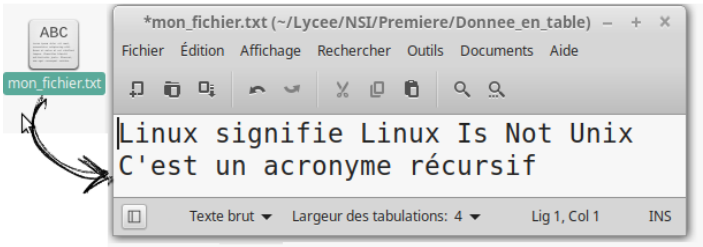
En python, on manipule les fichiers par l'intermédiaire d'un *objet-fichier* que l'on crée avec la fonction intégrée (ou *built-in*) `open`. Voici un exemple où on ouvre un ficher texte pour y écrire :

```
fic = open("mon_fichier.txt","w")
fic.write("Linux signifie Linux Is Not Unix\n")
ligne = "C'est un acronyme récursif\n"
fic.write(ligne)
fic.close()
```

Explications:

1. La commande open permet d'ouvrir un fichier. Le premier argument est le nom du fichier. Le second argument est ici `"w"` pour dire que l'on veut écrire dans le fichier (write en anglais).

2. On travaille pas avec le nom du fichier, mais avec la valeur renvoyée par la fonction open. Ici nous avons nommé `fic` ce fichier-objet. C'est avec cette variable `fic` que l'on travaille désormais.
3. L'instruction est `fic.write()` où l'argument est une chaîne de caractères.
4. Pour passer à la ligne, il faut ajouter le caractère de fin de ligne `\n`.
5. Il est important de fermer son fichier quand on a fini d'écrire. La commande est `fic.close()`.
6. Les données à écrire sont des chaînes, donc pour écrire un nombre, il faut d'abord le transformer par `str(nombre)`.



lecture d'un ficher

```
fic = open("mon_fichier.txt","r")
for ligne in fic:
    print(ligne, "\n")
fic.close()
```

Explications.:

1. La commande `open` est cette fois appelée avec le paramètre `"r"` (pour read), elle ouvre le fichier en lecture.
2. On travaille de nouveau avec un fichier-objet nommé ici *fic*.
3. Une boucle parcourt tout le fichier ligne par ligne. Ici on demande juste l'affichage de chaque ligne.
4. On ferme le fichier avec `fic.close()`.
5. Les données lues sont des chaînes, donc pour obtenir un nombre, il faut d'abord le transformer par `int(chaine)` (pour un entier) ou `float(chaine)` (pour un nombre à virgule).

Voici le résultat:

>>> Linux signifie Linux Is Not UnixC'est un acronyme récursif

1.1.2 Le mot clé `with`

Le mot clé **`with`** permet de gérer automatiquement la fermeture du fichier qui a été ouvert. Voici sa syntaxe :

```
with open(mon_fichier, mode_ouverture) as variable:
    # Opérations sur le fichier
```

On trouve dans l'ordre :

Le mot-clé **`with`**, prélude au bloc dans lequel on va manipuler notre fichier (on peut trouver **`with`** dans la manipulation d'autres objets).

Notre objet. Ici, on appelle **`open`** qui va renvoyer un objet **`notre fichier`**.

Le mot-clé **`as`** que nous avons déjà vu dans le mécanisme d'importation et dans les exceptions. Il signifie toujours la même chose : *en tant que* .

Notre variable qui contiendra notre objet. Si la variable n'existe pas, Python la crée.

Exemple:

```
>>> with open('mon_fichier.txt', 'r') as mon_fichier:
...     texte = mon_fichier.read()
...     print(texte)
...
Linux signifie Linux Is Not UnixC'est un acronyme récursif
```

Cela ne veut pas dire que le bloc d'instructions ne lèvera aucune exception.

Cela signifie simplement que, si une exception se produit, le fichier sera tout de même fermé à la fin du bloc.

Le mot-clé **`with`** permet de créer un "*context manager*" (gestionnaire de contexte) qui vérifie que le fichier est ouvert et fermé, même si des erreurs se produisent pendant le bloc. En résumé, ces deux blocs de code sont équivalents:

Avec `with` :

```
with open(fichier, mode) as fichier:
    # manipulation du fichier
```

Sans `with`

```
fichier = open(fichier, mode)
try:
    # manipulation du fichier
finally:
    fichier.close()
```

1.1.3 Les méthodes `readline` et `read`.

A:readline

Une fois le fichier ouvert et la première fois qu'elle est appelée, la méthode `readline` lit la première ligne du fichier. Au deuxième appel, elle lit la deuxième ligne et ainsi de suite jusqu'à ce qu'elle retourne la chaîne vide.

Exemple On a écrit un fichier texte contenant trois lignes:

```
première ligne du fichier
deuxième ligne du fichier
troisième ligne du fichier
```

On ouvre ce fichier et on lit toutes les lignes une par une :

```
>>> fic = open('exemple.txt', 'r')
>>> ligne = fic.readline()
>>> while ligne != '':
...     print(ligne)
...     ligne=fic.readline()
...
première ligne du fichier

deuxième ligne du fichier

troisième ligne du fichier
```

B:read

Il est aussi possible de lire l'intégralité du fichier avec la méthode `read` qui retourne ce contenu sous forme d'une chaîne de caractère comportant les fins de ligne `\n`.

```
>>> fic = open('exemple.txt', 'r')
>>> lignes = fic.read()
>>> lignes
'première ligne du fichier\ndeuxième ligne du fichier\ntroisième lig
```

1.1.4 Chaîne de caractère: méthode `split()` et `join()`

En manipulant les fichiers, vous serez amenés à gérer les chaînes de caractères. Il existe deux méthodes à connaitre.

`split()` Cette méthode sépare une chaîne de caractère **en une liste** de ses caractères, éventuellement en précisant le séparateur (espace par défaut).

Exemple:

```
>>> chaine = "Linux is not unix"
>>> chaine.split()
['Linux', 'is', 'not', 'unix']
```

`join()` Recolle une liste de caractère. c'est l'opération inverse de `split()`. Son usage est un peu spécial: `separateur.join(liste_à_recoller`

Exemple:

```
>>> liste = ["Linux", "is" ,"not" ,"unix"]
>>> ";".join(liste)
'Linux;is;not;unix'
```

1.2 Les fichiers CSV

Indexation de tables Une table de données représente une collection d'éléments. Chaque ligne représente un élément de la collection. Les colonnes représentent les attributs des éléments. Indexer une table, c'est créer une structure de données à partir de données dans un

tableau (tabulées). En base de données, un index est une structure de données auxiliaire permettant un accès rapide aux données. Nous utiliserons le format CSV pour les données tabulées.

1.2.1 Définition

Le format CSV (pour comma separated values, soit en français valeurs séparées par des virgules) est un format très pratique pour représenter des données structurées. Un fichier CSV est un fichier texte, par opposition aux formats dits binaires. Dans ce format, chaque ligne représente un enregistrement et, sur une même ligne, les différents champs de l'enregistrement sont séparés par une virgule (d'où le nom).

Figure 1.1: Le fichier CSV sur tableur et en fichier texte (*source:*`data.gouv.fr`).

	A	B	C	D
1	Cours d'eau	Embouchure	Bassin	Longueur (km)
2	Aa	Mer du Nord	Aa	89
3	Canche	Manche	Canche	88
4	Ternoise	Canche	Canche	43
5	Lawe	Lys	Escaut	41
6	Clarence	Lys	Escaut	33
7	Lys	Escaut	Escaut	119
8	Authie	Manche	Authie	103
9	Liane	Manche	Liane	37
10	Deûle	Lys	Escaut	59
11	Scarpe	Escaut	Escaut	94

```
Cours d'eau,Embouchure,Bassin,Longueur (km)
Aa,Mer du Nord,Aa,89
Canche,Manche,Canche,88
Ternoise,Canche,Canche,43
Lawe,Lys,Escaut,41
Clarence,Lys,Escaut,33
Lys,Escaut,Escaut,119
Authie,Manche,Authie,103
Liane,Manche,Liane ,37
Deûle,Lys,Escaut,59
Scarpe,Escaut ,Escaut,94
```

Exemple: Rivières du *Pas-de-Calais* Le bassin-versant est l'espace drainé par un cours d'eau et ses affluents. Autrement dit, d'après le tableau, la *Ternoise* est un affluent de la Canche.

En pratique, on peut spécifier le caractère utilisé pour séparer les différents champs et on utilise fréquemment un point-virgule, une tabulation ou deux points pour cela.

Notons enfin que la première ligne d'un tel fichier est souvent utilisée pour indiquer le nom des différents champs. Dans ce cas, le premier enregistrement apparaîssant en deuxième ligne du fichier.

1.2.2 Manipulation du fichier CSV

On choisit de représenter les fichiers CSV par des listes de dictionnaires. Une table est une liste de dictionnaire, à chaque ligne du fichier correspond un dictionnaire. Les clés sont les colonnes du fichier CSV, donc les attributs des éléments.

Exemple sur une partie du fichier précédent

```
tab_fleuve = [{'Nom':'Aa', 'Embouchure': 'Mer du Nord', 'longueur':89, 'Bassin':'Aa'},
              {'Nom':'Canche', 'Embouchure': 'Manche','longueur':88, 'Bassin':'Canche'},
              {'Nom':'Authie', 'Embouchure': 'Manche','longueur':103, 'Bassin':'Authie'}
              {'Nom':'Liane', 'Embouchure': 'Manche','longueur':37, 'Bassin':'Liane'}]
```

Import du fichier

Le module csv sera utilisé pour faciliter la manipulation des fichiers. Voici les méthodes du module qui nous seront utiles.

```
>>> import csv
>>> fichier = open("fleuve.csv")
>>> variable = csv.DictReader(fichier)
>>> table= [dict(ligne) for ligne in variable]
>>> #Affichage ligne par ligne de la liste table
>>> for lig in table:
...     print(lig)
...     print("")
...
{'Nom': 'Aa', 'Mer': 'Mer du Nord', 'Bassin': 'Aa', 'Longueur (km)': '89'}

{'Nom': 'Canche', 'Mer': 'Manche', 'Bassin': 'Canche', 'Longueur (km)': '88'}

{'Nom': 'Authie', 'Mer': 'Manche', 'Bassin': 'Authie', 'Longueur (km)': '103'}

{'Nom': 'Liane', 'Mer': 'Manche', 'Bassin': 'Liane\xa0', 'Longueur (km)': '37'}
```

`table` est une liste de dictionnaire dont les clés sont les attributs.

Export du fichier

1.3 Opérations sur les tables

Dans le paragraphe précédent, nous avons vu comment charger les données en utilisant un fichier. Nous avons obtenu une table de données qui représentent une collection d'éléments.

Chaque ligne est un élément de la collection. Les colonnes représentent les attributs d'un éléments.

Exemple: Dans la table `riviere`, chaque ligne représente un cours d'eau (sauf celle contenant les entêtes), les attributs de la table sont les noms des colonnes: nom, embouchure, longueur, bassin.

1.3.1 Récupération d'une donnée

On peut utiliser les opérations de manipulation des listes et dictionnaires pour exploiter les données. Ecrire une fonction `existe(nom, riviere)` vérifiant si pour un nom donné, le cours d'eau est présent dans la table, `existe` retournera un booléen.

1.3.2 Agrégation

Si on combine plusieurs lignes pour produire un résultat comme un calcul d emoyenne ou un comptage, on parle d'`agrégation`.

1. Ecrire une fonction `longueur(long , riviere)` retournant le nombre de cours d'eau dont la longueur est plus grande que `long`.
2. Ecrire une fonction `longueur_moy(riviere)` retournant la longueur moyenne des affluents de la Canche.

1.3.3 Sélection dans une table

Sélection de lignes Si on produit une nouvelle table avec des lignes de la table d'origine, on effectue une **sélection**. Ecrire une fonction ou une commande python retournant la liste des cours d'eau de plus de 50 km.

Un cours d'eau qui qui se jette dans la mer est un fleuve, écrire une commande python retournant la liste des cours d'eau qui sont des fleuves.

Sélection de lignes et de colonnes Si on veut sélectionner une colonne particulière plutôt que l'intégralité de la ligne, on parle de **projection**. Reprendre l'exemple précédent en ne retournant que les noms des cours d'eau concernés.

1.3.4 Tri de la table

Pour exploiter les données, il peut être intéressant de les trier. Avec la méthode `sorted` de Python, trier cette tableau du cours d'eau du plus court au plus long. Cette fonction a besoin d'une clé selon laquelle effectuer le tri.

1. Ecrire un fonction `critere(ligne)` retournant l'attribut pour une ligne donnée. Par exemple,
2. Trier la table riviere par ordre alphabétique selon la clé "nom".

Fusion/jointure

1.4 Exercices