Numérique et sciences informatiques
Classe de première

E Traitement de données en table
1. Indexation de tables

I Représentation des données en table

I.1 Un premier exemple

• Considérons le tableau suivant représentant une commande dans un magasin :

Référence	Désignation	Prix	Quantité	Disponible
18635	Lot crayon HV	2,30	1	Oui
17986	Stylo rouge	1,50	3	Non
20165	Cahier A4 petits carreaux	3,50	5	Oui
31254	Règle 30 cm	2,75	1	Non

Ce type de représentation des informations est utilisé depuis très longtemps. On dit qu'il s'agit d'une **représentation des données en table**. Cette façon de structurer les informations est massivement utilisée en informatique et il est important de connaître les procédés informatiques qui permettent d'exploiter cette organisation des données.

• La représentation en table suit les principes suivants :
  * Un tableau représente une **collection d'objets** ;
  * la première ligne joue un rôle particulier : elle présente la liste de **noms des attributs** des « objets » décrits dans le tableau ;
  * toutes les lignes suivantes ont la même fonction : chaque ligne correspond à un « objet » désigné et contient la succession des **valeurs des attributs** de l'objet en question ;
  * chaque colonne correspond à un'attribut différent ;
  * il y a *toujours le même nombre de colonnes pour chaque ligne* et les valeurs associées à un attribut donné sont *d'un type fixe*. Dans cet exemple, la référence et la quantité sont toujours exprimées par un entier, la désignation par une chaîne de caractère, le prix par un « nombre à virgule », la disponibilité par « oui » ou « non .

I.2 Les fichiers CSV

• En informatique, les données représentées sont conservées dans des fichiers. Si ces données à conserver sont représentées par un tableau, on utilisera très souvent un **fichier au format CSV** (pour « *Comma Separated Values* » : valeurs séparées par des virgules). Ces fichiers CSV seront repérés par le fait que leur nom comporte l'extension « `.csv` ».

• Les règles respectées un fichier au format csv sont les suivantes :
  * un fichier CSV est un **fichier texte** : les informations qu'ils contiennent sont interprétés comme une succession de caractères ;
  * la première ligne du tableau contient le nom des différents attributs associés à chaque objet de la collection à enregistrer ;
  * chacune des lignes suivantes contient les valeurs de chaque attribut d'un des objets de la collection ;
  * chaque ligne comporte exactement le même nombre de champs (même dans le cas d'attributs vides ;

* à l'intérieur d'une ligne, les valeurs des différents attributs (les « champs ») sont séparées par le caractère virgule ‘,’.

Remarque : En fait certains fichiers CSV peuvent utiliser un autre caractère comme séparateur : par exemple le point-virgule ou le caractère de tabulation.

• Par exemple, le tableau que nous avons vu plus haut pourra être enregistré dans un fichier CSV nommé « commande.csv » de la façon suivante :

```
Référence,Désignation,Prix,Quantité,Disponible
18635,Lot crayon HV,2.30,1,Oui
17986,Stylo rouge,1.50,3,Non
20165,Cahier A4 petits carreaux,3.50,5,Oui
31254,Règle 30 cm,2.75,1,Non
...
```

• *Remarque :* Il est utile de savoir que dans la plupart des langages de programmation le caractère ‘EOL’ de fin de ligne (qui marque donc la fin d'une ligne) est représenté par ‘\n’ et le caractère de tabulation par ‘\t’.

• Pour le cas où la valeur associée à certains attributs comporte le caractère ‘,’ ou le caractère ‘EOL’, on peut délimiter certain champ par un guillement droit ‘"’. De plus, si la valeur d'un attribut comporte le caractère ‘"’, on convient que, à l'intérieur d'un champ, le caractère ‘"’ est désigné par *deux* guillemets droits ‘""’.

I.3 Structures de données adaptées à la manipulation des données en table

• Pour manipuler les données en table dans un programme informatique, on a besoin d'une structure de données adaptée à cette organisation de l'information. On peut distinguer deux possibilités

a) Un tableau de tableaux ou un tableau de tuples

b) Un tableau de dictionnaires

II Fonctions et méthodes générales pour manipuler les fichiers en Python

II.1 Un exemple de lecture d'un fichier

Supposons que l'on dispose du roman « Notre-Dame de Paris » dans un fichier texte “NotreDameParis.txt“ et que l'on souhaite faire des statistiques sur ce texte à l'aide d'un programme python. On souhaite savoir combien de phrases du roman contiennent les chaînes de caractère “Quasimodo“ et “Notre-Dame“ en servant d'un petit programme. On peut alors écrire le programme suivant :

```
F = open("NotreDameParis.txt", 'r', encoding="utf-8")
contenu = F.read()
F.close()

compt1= 0
```

```
compt2= 0
tabPhrases = contenu.split(sep=".")
for phrase in tabPhrases :
    if "Quasimodo" in phrase :
        compt1 += 1
    if "Notre-Dame" in phrase :
        compt2 += 1
print(compt1)
print(compt2)
```

• A l'exécution il affichera :

```
238
147
```

Ceci signifie que le roman comporte 238 phrases avec "`Quasimodo`" et 147 phrases avec "`Notre-Dame`".

• Les trois premiers lignes du programme consiste à :
  * « ouvrir » un fichier nommé `NotreDameParis.txt ;`
  * récupérer le texte placé dans ce fichier dans la variable `contenu` ;
  * « fermer » le fichier.

II.2 Ouvrir et fermer un fichier

a) Ouvrir un fichier

• Pour « ouvrir » un fichier on utilise la sytaxe suivante :

```
F = open(nomFichier, mode, encodage)
```

b) Les arguments de la fonction `open`

Détaillons le role des différents argument de la fonction `open`.

• Le premier argument *`nomFichier`* est le **nom du fichier** auquel on veut accéder. Si ce fichier se trouve dans le même dossier que le programme, on se contentera d'indiquer son nom (extension comprise) comme une chaîne de caractères (donc entre guillemet ou entre apostrophes). Sinon, on devra aussi avant le « chemin d'accès » relatif ou absolu pour trouver le dossier où il est placé. Dans les exercices que nous ferons, on simplifiera la situation et le fichier sera toujours placé dans le même dossier que le programme qui le manipule.

• L'argument `mode` représente le **mode d'ouverture**. Nous ne présentons ici les valeurs les plus courantes de cet argument :
  * `'r'` signifie que l'on veut ouvrir le fichier « *en lecture* » (*'read'* en anglais). Il s'agit de *récupérer les informations contenues dans le fichier* pour les pouvoir les exploiter mais *sans modifier le fichier* ;
  * `'w'` signifie que l'on veut ouvrir le fichier *« en écriture »* (*'write'* en anglais). Il s'agit alors d'*y enregistrer de nouvelles informations* et non d'exploiter les informations qui s'y trouvent.

Il faut être très prudent lorsque l'on utilise le mode 'écriture' !
* En effet, si le nom de fichier indiqué en premier argument est celui d'un fichier qui existe déjà alors l'ouverture de ce fichier en écriture avec 'w' *effacera toutes les données déjà présentes dans le fichier* !

* En revanche, si le nom indiqué ne correspond à aucun fichier existant, alors un fichier portant ce nom est crée.

  * '`a`' (comme *'append'* en anglais) signifie que l'on veut ouvrir le fichier « en écriture » mais que *les nouvelles données seront placées à la fin du fichier* à la suite des données déjà présentes dans le fichier. Ces anciennes données ne seront donc pas effacées à l'ouverture.

• Par ailleurs, il faut savoir qu'un fichier peut être ouvert en mode 'binaire' ou en mode 'texte'. Ceci signifie que les informations extraites du fichiers pourront se présenter :

  * soit comme une suite de caractères (en mode texte) ;
  * soit comme une suite d'octets écrits en binaire (en mode binaire).

Par défaut, c'est-à-dire sans mention contraire explicite, le fichier sera ouvert en mode texte. Pour ouvrir un fichier en binaire, il faut ajouter un '`b`' dernière le mode choisi : par exemple, '`rb`' ou '`ab`'.

• Le troisième argument *`encodage`* ne sera utilisé que si le fichier est ouvert en mode texte. Il précise l'encodage utilisé pour les caractères. Par exemple, on écrira

```
encoding = "utf8"
```

pour indiquer que l'encodage est UTF 8.

c) Fermer un fichier

• L'instruction

```
F.close()
```

qui appelle la méthode `close()` associée à la variable `F` permet de « fermer » le fichier, c'est-à-dire de signaler que le programme n'a plus besoin du fichier qu'il a ouvert précédemment et donc de le rendre disponible pour un autre programme.

c) Une méthode alternative

• On peut aussi ouvrir un fichier avec la syntaxe suivante :

```
with open(nomFichier, mode, codage) as F :
    ...
```

Dans ce cas, le programme ne comportera pas d'instruction explicite pour fermer le fichier car le fichier sera automatiquement fermé après exécution de la dernière instruction du bloc ou bien si une erreur interromp le programme avant de l'avoir atteinte.

II.3 Déplacement du curseur

• Lorsque l'on lit ou écrit dans un fichier, on déplace un **curseur** qui indique « où on se trouve » dans le fichier. A chaque opération de lecture ou d'écriture dans le fichier, le curseur est déplacé de façon à lire les parties suivantes ou écrire à la suite de ce qui a dèjà été écrit. Par exemple, si le curseur est placé au début du fichier et si on demande à lire la première ligne, le curseur sera ensuite placé juste après le premier caractère de fin de ligne.

• A l'ouverture, le curseur n'est pas toujours situé au même endroit du fichier. Ceci dépent en effet du mode d'ouverture choisi :

  * Lorsqu'on ouvre un fichier en mode '`r`' ou '`w`' le curseur est placé au début du fichier ;
  * Lorsqu'on ouvre un fichier en mode '`a`' le curseur est placé à la fin du fichier.

• L'instruction `F.tell()` nous indiquera où le curseur est situé par rapport au début du fichier.

II.4 Quelques méthodes de lecture

Nous supposerons ici que le fichier a été ouvert en mode texte. Pour *lire* dans un fichier que l'on a préalablement ouvert en lecture, on peut utiliser les instructions suivantes.

a) La méthode `read()`

• Dans l'exemple de la partie II.1, nous avons placé la totalité du texte contenu dans le fichier dans la variable `contenu` à l'aide de l'instruction suivante :

```
contenu = F.read()
```

Puisque le fichier est ouvert en mode texte, cette variable `contenu` est alors de type **`str`**.

* Si `n` est un entier (de type **`int`**), l'instruction suivante :

```
texte = F.read(n)
```

placera les `n` *caractères* situés devant le curseur dans la variable `texte`. Si la fin du fichier se situe moins de `n` caractères après le curseur, la variable recevra tous les caractères situés après le curseur.

b) La méthode `readline()`

• L'instruction suivante

```
ligne = F.readline()
```

place dans la variable `ligne` tous les caractères situés entre le curseur et le premier caractère de fin de ligne qui le suit (en incluant ce caractère '`EOL`' de fin de ligne). Si on arrive à la fin du fichier (c'est-à-dire que l'on rencontre le caractère '`EOF`' de fin de fichier) *avant* de rencontrer un caractère de fin de ligne, alors on placera tous les caractères situés entre le curseur et la fin du fichier dans la variable `ligne`.

c) La méthode `readlines()`

• L'instruction

```
tabLignes = F.readlines()
```

placera le contenu du fichier dans un tableau de chaines de caractères et chaque élément du tableau contiendra une ligne du fichier.

II.5 Une méthode d'écriture

Nous supposerons toujours que le fichier a été ouvert en écriture en mode texte. Pour *écrire* dans un fichier que l'on a préalablement ouvert en écriture, on peut utiliser l'instruction suivante.

```
F.write(texte)
```

En supposant ici que la variable `texte` est de type **`str`** cette instruction permet de placer le contenu de la variable `texte` dans le fichier à partir de la position du curseur.

III Fonctions du module CSV pour manipuler des fichiers CSV en python

III.1 Le résultat souhaité

• **L'indexation de données en table** est la création d'une structure de données qui contiennent toutes les informations contenues dans un fichier au format CSV en choisissant une structure qui la rende facilement manipulable dans un programme python. Nous allons choisir d'utiliser un tableau de dictionnaires.

• On souhaite écrire un programme en python qui récupère les données contenues dans le fichier `commande.csv` et qui les place dans une variable **`commande`** de type **`list`** qui sera organisée de la façon suivante :

```
[{"Référence" : 18635,"Désignation" : "Lot crayon HB", "Prix" : 2.30,
                "Quantité" : 1, "Disponible" : True},
                                ...,
 {"Référence" : 31254,"Désignation" : "Règle 30 cm", "Prix" : 2.75,
                "Quantité" : 1, "Disponible" : False}]
```

• Chaque élément de ce tableau est donc un objet de type **`dict`** ;

• Les *clés* de chacun des dictionnaires de ce tableau sont exactement les mêmes et *correspondent aux noms des attributs* du tableau de départ : `"Référence"`, `"Désignation"`, `"Prix"`, `"Quantité"`, `"Disponible"`.

• Les valeurs associées à chaque attribut sont d'un type donné et adapté à la nature de l'information à exploiter. Dans cet exemple :

* de type **`str`** pour la « désignation » ,
* de type **`int`** pour la « référence » et la « quantité » ;
* de type **`float`** pour le « prix » ;
* de type **`bool`** pour la « disponibilité ».

• La longueur de ce tableau correspond donc au nombre d'objets enregistrés dans la table.

III.2 Lecture d'un fichier CSV avec la fonction `DictReader`

• Pour lire un fichier CSV, on peut utiliser le module `csv` de la bibliothèque standard de Python. Par exemple, pour lire le fichier `commande.csv`, on peut écrire le programme suivant :

```
import csv
F = open("commande.csv", "r", encoding="utf8")
commande = list(csv.DictReader(F))
F.close()
```

• Si l'on souhaite imprimer à l'écran le contenu de la variable `commande` en executant le code suivant :

```
    for e in commande :
        print(e)
```

on obtiendra bien le résultat suivant :

```
{'Référence': '49514', 'Désignation': 'Lot 5 crayons HB',
'Prix': '4.3', 'Quantité': '6', 'Disponible': 'False'}
...
```

• Dans un fichier CSV, il est possible que le caractère de séparation des champs ne soit pas une virgule. Ce peut aussi être le caractère de tabulation '\t', les deux points ':', le point-virgule ';', etc. Ceci peut être préciser dans l'appel de la fonction `DictReader` en indiquant une valeur à un argument `delimiter` de la façon suivante :

```
commande = list(csv.DictReader(F, delimiter=";"))
```

• Nous voyons donc que, grâce à la fonction **`list(csv.DictReader(...))`** du module CSV, il est aisé de récupérer une liste de dictionnaires à partir d'un fichier CSV.

III.3 Ajustements sur la variable obtenue

• Le résultat obtenu n'est cependant pas encore entièrement satisfaisant. En effet, les valeurs associées à chaque clé des dictionnaires sont toujours de type **`str`** puisqu'elles sont extraites d'un fichier texte. Il nous faut donc effectuer des conversions de type de façon à ce que :

* les valeurs associées aux clés "`Référence`" et "`Quantité`" soient de type **`int`** ;
* les valeurs associées à la clé "`Prix`" soit de type **`float`** ;
* les valeurs associées à la clé "`Disponible`" soit de type **`bool`** ;
* les valeurs associées à la clé "`Désignation`" reste de type **`str`**.

• Pour procéder à ces conversions de type, on peut utiliser les fonctions suivantes :

```
def convertir(dico) :
    dico["Référence"] = int(dico["Référence"])
    dico["Prix"] = float(dico["Prix"])
    dico["Quantité"] = int(dico["Quantité"])
    dico["Disponible"] = (dico["Disponible"]== "Oui")

def convertirTab(tab) :
    for d in tab :
        convertir(d)

convertirTab(commande)
```

On pourra alors vérifier que les valeurs associées aux différentes clés sont alors bien toutes du type souhaité.

III.4 Ecriture dans un fichier CSV avec la fonction `DictWriter`

• Le module **`csv`** nous permet aussi de créer un fichier CSV à partir d'une variable `table` qui est une liste de dictionnaires ou de dictionnaires ordonnées comme celle que nous venons de voir. Il suffit d'écrire les lignes de code suivantes :

```
sortie = open("commandeModif.csv", "w", encoding="utf8")
w = csv.DictWriter(sortie, list(table[0].keys()))
w.writeheader()
w.writerows(table)
sortie.close()
```

Attention ! On suppose ici que le fichier `commandeModifiee.csv` *n'existe pas encore* dans le dossier où se trouve le programme. Dans ce cas, l'instruction d'ouverture de ce fichier en écriture (le deuxième argument est **`'w'`**) va engendrer *la création de ce fichier* dans ce même dossier. Si le fichier existe déjà, cette instruction d'ouverture en écriture va entraîner *l'effacement des données contenues dans le fichier*.

• Une fois la séquence d'instructions exécutée, le fichier `commandeModif.csv` aura pour contenu :

```
Référence,Désignation,Prix,Quantité,Disponible
18635,Lot crayon HV,2.3,1,True
17986,Stylo rouge,1.5,3,False
20165,Cahier A4 petits carreaux,3.5,5,True
31254,Règle 30 cm,2.75,1,False
```