Les données tabulées (organisées en tableaux) sont omniprésentes en informatique. On les retrouve dans les tableurs (par exemple Excel) mais aussi dans les sytème de gestion de bases de données (par exemple Mysql).

Le fait de stocker des informations sous forme de tableaux permet de les structurer de faciliter leur manipulation.

Dans ce chapitre, nous allons voir comment représenter en Python de telles données ainsi que le format le plus couramment utiliser pour les stocker.

# 1 Table de données

Une table de données est un tableau contenant des informations sur une collection d'entités. On l'organise de la manière suivante :

— chaque ligne du tableau représente une **entité** ;

— chaque colonne du tableau représente une information sur les entités correspondantes, on parle d'**attribut**.

| **Nom** | **Âge** | **Taille** | **Ville** |
|---|---|---|---|
| Tata | 34 | 1.74 | Lyon |
| Titi | 25 | 1.78 | Lille |
| Tutu | 67 | 1.75 | Paris |
| Tete | 46 | 1.57 | Caen |
| Toto | 12 | 1.34 | Tourcoing |

On a vu dans le chapitre précédent que les n-uplets pouvaient être utilisés pour stocker des données (potentiellement de natures différentes) concernant une certaine entité.

```
u = ("toto", 12, 1.35, "Tourcoing")
```

Le problème de cette approche est qu'il est nécessaire de se souvenir à quelle position dans le n-uplet correspond chaque information (nom à l'indice 0, âge en 1, etc.).

Pour y remédier, on peut en Python utiliser à la place un dictionnaire dans lequel les clés sont les noms des informations (`"nom"`, `"age"`) et les valeurs les valeurs de ces informations (`"toto"`,12).

```
u = {"nom":"toto", "age":12, "taille":1.35, "ville":"Tourcoing"}
```

On parle dans ce cas de **n-uplet nommé**.

Une solution pour représenter une table de données en Python est alors d'utiliser un **tableau de n-uplets nommés** :

```
table = [
{"nom":"Tata", "age":34, "taille":1.74, "ville":"Lyon"},
{"nom":"Titi", "age":25, "taille":1.78, "ville":"Lille"},
{"nom":"Tutu", "age":67, "taille":1.75, "ville":"Paris"},
{"nom":"Tete", "age":46, "taille":1.57, "ville":"Caen"},
{"nom":"Toto", "age":12, "taille":1.34, "ville":"Tourcoing"}
]
```

Cette représentation est utile pour la manipulation de données à l'intérieur d'un programme Python. Cependant, de telles données sont généralement stockées dans un fichier externe de sorte à pouvoir être générées ou modifiées par d'autres programmes. Pour cela, on utilise généralement le format csv.

# 2 Le format csv

Le format csv (*coma separated values*, données séparées par des virgules) est le format standard pour sauvegarder des données tabulées. Un fichier csv est un fichier texte dans lequel on organise les données comme suit :

— chaque ligne du fichier correspond à une ligne de la table ;

— chaque ligne est séparée en champs (les attributs) par un `","` ;

— toutes les lignes ont le même nombre de champs ; la première ligne peut représenter les noms des attributs ou commencer directement avec les données ;

— on peut utiliser des guillemets droits (`""`) pour délimiter le contenu des champs.

Ainsi la table vue dans la partie précédente peut être sauvegardée dans un fichier `"table.csv"` :

```
nom, age, taille, ville
Tata, 34, 1.74, Lyon
Titi, 25, 1.78, Lille
Tutu, 67, 1.75, Paris
Tete, 46, 1.57, Caen
Toto, 12, 1.34, Tourcoing
```

Ce fichier pourra alors être lu ou modifié par divers programmes.

# 3 Lecture et validation des données

La bibliothèque `csv` de Python propose des outils pour manipuler des fichier csv. Pour lire un fchier, on utilise les commandes suivantes :

```
import csv                          #importation du module
f = open("table.csv", 'r')          #ouverture du fichier
table = list(csv.DictReader(f))     #creation de la table
f.close()                           #fermeture du fichier
```

La variable `table` contient après exécution de ces commandes le tableau contenu dans le fichier `"table.csv"` sous la forme d'un tableau de n-uplets nommés comme vu précédemment.

**Attention :** Pour utiliser cette méthode, il faut que la première ligne du fichier csv comporte les noms des attributs de la table. Sans cela, les clés des dictionnaires correspondrons aux valeurs des attributs de la première entité.

Un problème persiste encore ici après l'import des données : les types des attributs. En effet, la méthode présentée génère un tableau de n-uplets nommés dont tous les attributs sont des chaînes de caractère. Cela est normal car un fichier csv est un simple fichier texte.

Il est donc nécessaire si l'on veut manipuler correctement les données importer, d'effectuer un certain nombre de conversions. On parle alors le **validation des données**.

Cette validation doit être effectuée en fonction des types de données stockées dans le fichier csv. Dans notre exemple, les types seront les suivants :

| **Attribut** | nom | age | taille | ville |
|---|---|---|---|---|
| **Type** | str | int | float | str |

Définissons une fonction permettant de valider une ligne la table :

```
def valide(x):
    nom = x["nom"]
    age = int(x["age"])
    taille = float(x["taille"]})
    ville = x["ville"]
    return {"nom":nom, "age":age, "taile":taille, "ville":ville}
```

On peut alors utiliser cette fonction pour générer une table valide :

```
table_valide = [valide(x) for x in table]
```

**Remarque :** Des erreurs peuvent survenir si le fichier csv est mal formé. On peut penser à des conversions impossibles (`int("bonjour")`).

# 4 Écriture

Pour écrire un nouveau fichier ou écraser un fichier existant à partir d'une table python `table` sous la forme d'un tableau de n-uplets nommés, la syntaxe est la suivante :

```
import csv
f = open("fichier.csv", 'w') #ouverture du ficher
w = csv.DictWriter(f, ["nom", "age", "taille", "ville"])
#creation d un objet du module csv permettant d ecrire
#dans le fichier, on precise les noms des attributs
w.writeheader()             #ecriture des noms des attributs
w.writerows(table)          #ecriture des lignes du fichier
f.close()                   #fermeture du fichier
```