Chapitre n° 13 : tables de données, indexation et recherche

Compétences attendues

— Importer une table depuis un fichier texte tabulé ou un fichier CSV.

— Rechercher les lignes d'une table vérifiant des critères exprimés en logique propositionnelle.

1 Indexation de tables

1.1 Notion de table de données

1.1.1 Présentation

L'organisation tabulaire des données est très répandue. Par exemple, un bulletin scolaire est organisé en table et indique, pour chaque matière, la note de l'élève, la moyenne de la classe, la note la plus basse et la note la plus haute de la classe, ainsi que l'appréciation du professeur. Les résultats d'un match de tennis sont eux-aussi présentés comme une table : les lignes correspondent aux joueurs et les colonnes indiquent le nombre de jeux gagnés dans chaque *set*.

L'organisation tabulaire est également très ancienne. Les tables documentés les plus anciennes sont des livres. Elles sont mentionnées dans l'Égypte ancienne !

Les données tabulées ont évidemment une place très importantes en informatique. Avec l'introduction du *modèle relationnel*, proposé par Edgar F. Codd dans les années 1970 alors qu'il est employé par IBM, les tables de données, stockées dans des bases de données, deviennent rapidement le principal moyen de stocker de l'information structurée. Même sans utiliser de tels systèmes, la manipulation des données en tables depuis un langage de programmation est un outil important, utilisé dans de nombreux domaines : calcul scientifique, intelligence artificielle, programmation Web, bio-informatique, informatique financière, *etc.*.

1.1.2 Définition

Nous ferons dans ce cours des hypothèses sur les données en table. ces tables représentent des *collections d'éléments*.
Chaque *ligne* représente un élément de la collection.
Les *colonnes* représentent les *attributs* d'un élément. Pour un attribut donné, les valeurs sont toutes du *même type* (*int*, *float*, *str*, *etc.*).

Exemple : dans le tableau ci-dessous, chaque ligne (autre que la ligne contenant les entêtes des colonnes) représente un élève.Les attributs de chaque élève, et par extension les attributs de la table, sont les noms de colonnes (*prénom*, *jour*,

mois, *année*, *projet*). Les prénoms et les projets sont des chaînes de caractères, les autres attributs sont des entiers.

prénom	jour	mois	année	projet
Brian	1	1	1942	Programmer avec style.
Grace	9	12	1906	Production de code machine.
Linus	28	12	1969	Un petit système d'exploitation.
Donald	10	1	1942	Tout sur les algorithmes.
Alan	23	6	1912	Déchiffrer des codes secrets.
Blaise	19	6	1623	Machine arithmétique.
Margaret	17	8	1936	Atterrissage d'un module lunaire.
Alan	1	4	1922	Ce qu'un programmeur doit savoir.
Joseph Marie	7	7	1752	Programmer des dessins.

Le terme *indexation de table* signifie qu'il y a création d'une structure de données à partir de données tabulées. En bases de données, un *index* est une structure de données auxiliaire permettant un accès rapide aux données de la table

1.2 Lire un fichier au format CSV

1.2.1 Présentation des fichiers CSV

La première opération concernant les données en table est le *chargement de données*. En effet, le programme qui analyse les données n'est généralement pas celui qui les a produites initialement. Un moyen simple d'échange de données est la transmission d'un *fichier*. Les deux programmes doivent utiliser le même format de stockage. Pour les données tabulées, l'usage est d'utiliser le format CSV (*comma-separated values*, cce qui signifie « données séparées par des virgules »). Ce format est relativement simple :

- les fichiers CSV sont de simples fichiers textes ;
- chaque ligne du fichier correspond à une ligne de la table ;
- chaque ligne est séparées en champs au moyen du caractère « , » (en français, comme la notation des nombres utilise la virgule, on utilise plutôt le point virgule comme séparateur « ; ») ;
- toutes les lignes du fichier ont le même nombre de champs ;
- la première ligne peut représenter des noms d'attributs ou commencer directement avec les données ;
- on peut utiliser des guillemets droits « " » pour délimiter le contenu des champs.

Exemple : le tableau précédent peut être représenté par le fichier texte suivant.

"prénom","jour","mois","année","projet"

"Brian",1,1,1942,"Programmer avec style."

"Grace",9,12,1906,"Production de code machine."

"Linus",28,12,1969,"Un petit système d'exploitation."

"Donald",10,1,1942,"Tout sur les algorithmes."

"Alan",23,6,1912,"Déchiffrer des codes secrets."

"Blaise",19,6,1623,"Machine arithmétique."

"Margaret",17,8,1936,"Atterrissage d'un module lunaire."

"Alan",1,4,1922,"Ce qu'un programmeur doit savoir."

"Joseph Marie",7,7,1752,"Programmer des dessins."

Remarque : il existe des règles pour pouvoir insérer les caractères spéciaux (virgules, retours à la ligne et guillemets droits) dans les champs.

1.2.2 Fichiers CSV et Python

La bibliothèque standard Python propose un module *csv* contenant des fonctions utilitaires pour lire et écrire des fichiers CSV.

Exemple : le code suivant permet de charger le fichier *eleves.csv* dans la variable *table*.

```
1 import csv
2 fichier = open("eleves.csv")
3 table = list(csv.reader(fichier))
4 fichier.close()
```

Après l'import de la bibliothèque, on ouvre le fichier *eleves.csv* et on l'attribut à l'objet *fichier*.

La fonction *reader* du module *csv* prend en argument un fichier ouvert et renvoie une valeur spéciale représentant le fichier csv. Cette valeur peut être convertie en tableau grâce à la fonction *list*.

A ce stade, la variable *table* contient un tableau de tableaux de chaînes de caractères.
[['prénom', 'jour', 'mois', 'année', 'projet'],
['Brian', '1', '1', '1942', 'Programmer avec style.'],
['Grace', '9', '12', '1906', 'Production de code machine.'], ['Linus', '28', '12', '1969', "Un petit système d'exploitation."],
...
['Joseph Marie', '7', '7', '1752', 'Programmer des dessins.']]

L'inconvénient d'utiliser un tableau de de tableaux est que pour accéder à une ligne, ou dans une ligne accéder à un attribut donné, il faut connaître les indices. De plus, la première ligne, qui contient les noms des attributs, a été

considérée comme les autres lignes. Une alternative est d'utiliser la fonction *DictReader* du module *csv*.

Exemple :

```
import csv
fichier = open("eleves.csv")
table = list(csv.DictReader(fichier))
fichier.close()
```

Dans ce cas, la variable *table* contient un tableau de dictionnaires ordonnés :
[OrderedDict([('prénom', 'Brian'), ('jour', '1'), ('mois', '1'), ('année', '1942'), ('projet', 'Programmer avec style.')]),
OrderedDict([('prénom', 'Grace'), ('jour', '9'), ('mois', '12'), ('année', '1906'), ('projet', 'Production de code machine.')]),
...
OrderedDict([('prénom', 'Joseph Marie'), ('jour', '7'), ('mois', '7'), ('année', '1752'), ('projet', 'Programmer des dessins.')])]

Ce sont des dictionnaires qui stockent un ordre particulier pour leurs clefs. Nous les utiliserons comme des dictionnaires standards. On peut remarquer que cette fois, *la première ligne a été utilisée pour créer les clefs des dictionnaires.*

1.3 Validation des données

L'utilisation de la fonction *DictReader* a permis de créer une liste de dictionnaires plutôt qu'une liste de liste. En revanche, les jours, mois et années sont chargés comme des chaînes de caractères. Il faut donc changer leur type, nous souhaiterions que ce soit des entiers.

Cet exemple est généralisable : *il faut pouvoir tester que le tableau contient des données valides.*

Voici deux façons de créer une table valide, qui demandent de connaître le type des attributs *a priori*.

La première crée une copie de la table d'origine. On définit en premier lieu une fonction de validation (appelée *valide* ci-dessous). Cette fonction attend un dictionnaire en argument. Elle en extrait les valeurs puis effectue des vérifications et des conversions. Elle retourne un dictionnaire avec les données validées.

```
def valide(x):
    prenom = x["prénom"]
    jour = int(x["jour"])
    mois = int(x["mois"])
    annee = int(x["année"])
    projet = x["projet"]
    if mois < 1 or mois > 12:
        exit("mois invalide dans le fichier CSV")
    return {"prénom": prenom, "jour": jour, "mois": mois, "année":
    annee, "projet": projet}

```

```
11  table_validee = [valide(x) for x in table]
```

Remarque : on a utilisé la notation par compréhension pour créer la table validée.

Dans cet exemple, la fonction quittera le programme avec une erreur si l'une des conditions suivantes se réalise :

— le dictionnaire x ne contient pas l'une des cinq clefs attendue ;

— un appel à *int* échoue (parce que la chaîne de caractères stockée dans le fichier ne correspond pas à un entier) ;

— le mois n'est pas compris entre 1 et 12.

Une seconde façon de créer une table valide consiste à modifier la table d'origine.

```
1  for x in table:
2      x["jour"] = int(x["jour"])
3      mois = int(x["mois"])
4      if mois < 1 or mois > 12:
5          exit("mois invalide dans le fichier CSV")
6      x["mois"] = int(x["mois"])
7      x["année"] = int(x["année"])
```

1.4 Écrire un fichier au format CSV

Le module *csv* de Python propose également des fonctions utilitaires pour écrire le contenu d'un tableau de dictionnaires dans un fichier CSV.

Exemple :

```
1  sortie = open("nouveau.csv", "w")
2  w = csv.DictWriter(sortie, ["prénom", "jour", "mois", "année", "
       projet"])
3  w.writeheader()
4  w.writerows(table)
5  sortie.close()
```

Ce code commence par ouvrir un fichier en écriture (grâce au paramètre "w" passé à *open*). On appelle ensuite la fonction *DictWriter* en lui passant en argument le fichier ouvert en écriture et la liste des attributs. La fonction *DictWriter* renvoie un objet w permettant d'écrire des lignes dans le fichier. L'instruction *w.writeheader()* doit être appelée en premier et écrit la ligne d'entêtes. L'instruction *w.writerows(table)* prend la table en argument et écrit les lignes correspondantes dans le fichier.

2 Recherche dans une table

2.1 Présentation

Une fois qu'un ensemble de données est chargé dans une table, il devient possible d'exploiter ces données à l'aide des opérations de manipulation de tableaux. Par exemple, il est possible d'extraire des données cibles, tester la présence de certaines données, de faire des statistiques, etc. Ces opérations sont appelées des *requêtes*. Nous allons voir dans cette partie quelques exemples de manipulations usuelles.

2.2 Recherche d'un élément

2.2.1 Recherche simple

Nous avons déjà vu un exemple d'algorithme de recherche d'un élément dans un tableau. La fonction *appartient* ci-dessous renvoie *True* si l'élément v est dans le tableau t et *False* sinon.

```
1  def appartient(v, t):
2      for x in t:
3          if v == x:
4              return True
5      return False
```

Cette fonction est bien adaptée pour des tableaux simples. En revanche, les données chargées en table constituent un tableau de dictionnaires : l'élément recherché est donc un dictionnaire, ce qui est contraignant.

Par exemple, dans les données chargées depuis le fichier *eleves.csv*, on pourrait rechercher l'élément :
{"prénom" : "Donald", "jour" : 10, "mois" : 1, "année" : 1942, "projet" : "Tout sur les algorithmes."}

2.2.2 Recherche en fonction d'un attribut clef

Il est possible d'adapter la fonction précédente pour mener une recherche à partir d'un seul attribut. Dans notre exemple, la recherche pourrait être menée sur le prénom.

```
1  def appartient_prenom(p, eleves):
2      for e in eleves:
3          if e["prénom"] == p:
4              return True
5      return False
```

L'algorithme parcourt à nouveau l'ensemble des éléments du tableau, c'est à dire les lignes de la table, mais le test d'égalité est fait sur l'attribut qui nous intéresse plutôt que sur l'intégralité de la ligne.

2.2.3 Récupération d'une donnée simple

À partir de cette base, on peut déduire d'autres fonctions qui, au lieu de seulement indiquer la présence ou l'absence d'un élément, renvoient certaines des informations associées.

Par exemple, la fonction *projet_de* ci-dessous renvoie le projet de l'élève ayant le prénom p, ou renvoie *None* si aucun élève n'a le prénom p.

```
def projet_de(p, eleves):
    for e in eleves:
        if e["prénom"] == p:
            return e["projet"]
```

Remarque : si plusieurs élèves ont le même prénom (comme *Alan* dans notre exemple), seul le premier projet est renvoyé.

Il est parfois utile de préciser la recherche, par exemple en tenant à la fois compte du prénom et de l'année de naissance.

```
def projet_de(p, a, eleves):
    for e in eleves:
        if e["prénom"] == p and e["année"] == a:
            return e["projet"]
```

2.3 Agrégation

2.3.1 Présentation

Les opérations d'*agrégation* combinent les données de plusieurs lignes pour produire un résultat, typiquement une statistique, sur ces données.

2.3.2 Exemples

Comptage d'occurrences.
Par exemple, la fonction suivante compte le nombre d'élèves dont l'année de naissance est a.

```
def eleves_nes_en(a, eleves):
    nb = 0
    for e in eleves:
        if e["année"] == a:
            nb += 1
    return nb
```

Sommes et moyennes.
De façon générale, les opérations d'agrégation peuvent être réalisées en utilisant des accumulateurs qui enregistrent progressivement un bilan du parcours de la table.

Par exemple, la fonction suivante calcule l'âge moyen de la classe à la fin de l'année a en utilisant un accumulateur pour la somme des âges.

```
def age_moyen(a, eleves):
    somme = 0
    for e in eleves:
        somme += a - e["année"]
    return somme / len(eleves)
```

Remarque : comme précédemment, il est possible d'ajouter des conditions dans le code précédent pour ne faire la moyenne que sur une catégorie d'élèves.

2.4 Sélection de lignes

2.4.1 Principe

Nous avons vu comment utiliser la table de données pour produite un résultat simple (valeur d'un attribut, valeur agrégée, etc.). Une autre opération courante est la *sélection*, qui consiste à produire une nouvelle table en extrayant de la table d'origine toutes les lignes vérifiant une certaine condition.

2.4.2 Exemple

Nous allons construire une table comprenant tous les élèves nés entre le premier janvier et le 20 mai inclus.

Une première méthode consiste à créer d'abord un tableau t vide, puis d'ajouter une à une les lignes sélectionnées.

```
t = []
for e in eleves:
    if e["mois"] < 5 or (e["mois"] == 5 and e["jour"] <= 20):
        t.append(e)
```

Une autre méthode, plus rapide, consiste à construire le tableau par compréhension.

```
t = [e for e in eleves if e["mois"] < 5 or (e["mois"] == 5 and e["
    jour"] <= 20)]
```

2.4.3 Application : sélection de lignes valides

Dans un fichier tel que *eleves.csv*, la colonne *jour* n'est censé ne contenir que des valeurs comprises entre 1 et 31 (voire 28, 29 ou 30 suivant le mois et l'année), la colonne *mois* n'est censé contenir que des valeurs comprimes entre 1 et 12 et la colonne *année* n'est censé ne contenir que des valeurs inférieures ou égales à 2020. Néanmoins, rien ne garantit que ça soit bien le cas au moment où on charge le fichier, qui peut être corrompu ou invalide pour différentes raisons.

Plutôt que de stopper le programme à la première ligne invalide rencontrée, comme dans la première partie, il est possible d'utiliser une opération de sélection pour construire une table contenant chaque ligne valide du fichier de données et ignorant les autres.

```
t = [e for e in eleves if 1 <= int(e["jour"]) <= 31 and 1 <= int(e["mois"]) <= 12 <  and int(e["année"]) <= 2020)]
```

2.5 Sélection de lignes et colonnes

La construction par compréhension permet également d'effectuer des opérations. Ainsi, il est possible de ne sélectionner qu'une colonne particulière plutôt que l'intégralité des lignes.

Exemple : le code ci-dessous construit un tableau contenant les prénoms de chaque élève de la classe.

```
t = [e["prénom"] for e in eleves]
```

On appelle cette opération une *projection*.

Il est naturellement possible de combiner une *projection* à la sélection d'un ensemble de lignes.

Exemple : le code ci-dessous construit un tableau contenant les prénoms de chaque élève de la classe né entre le premier janvier et le 20 mai inclus.

```
t = [e["prénom"] for e in eleves if e["mois"] < 5 or (e["mois"] == 5 and e["jour"] <= 20)]
```

Il est possible d'utiliser cette opération de projection pour générer une table conservant plusieurs des attributs et même en ajouter des nouveaux avec ou sans sélection.

Exemple : le code ci-dessous construit une table comprenant pour chaque ligne l'attribut "prénom", ajouter un nouvel attribut "âge" (dont la valeur est calculée) et comprenant enfin l'attribut "projet".

```
t = [{"prénom": e["prénom"], "âge":2020 - e["année"], "projet": e["projet"]} for e in eleves]
```