Chapitre n° 14 : tables de données, tri et fusion

Compétences attendues

— Trier une table suivant une colonne.
— Construire une nouvelle table en combinant les données de deux tables.

1 Tri d'une table

1.1 Présentation

Nous avons vu précédemment l'intérêt de trier des données, ainsi que des exemples d'algorithmes pour le faire. Dans le cas de tables de données, il peut, par exemple, être intéressant d'afficher une liste d'élèves par ordre alphabétique ou en fonction d'une note obtenue à un devoir.

Pour cela, il est possible d'adapter les programmes de tris pour les appliquer à des tableaux de dictionnaires. Mais, à chaque fois que le critère de tri change (le nom, une note, etc.), le programme doit être modifié.

Pour la suite du chapitre, nous utiliserons les fonctions de tris présentes dans le langage Python. Si t est un tableau (par exemple de nombres) à trier :

— *t.sort()* modifie le tableau t ;
— *sorted(t)* crée un nouveau tableau (sans modifier le tableau t d'origine), ce nouveau tableau doit être affecté à une nouvelle variable.

1.2 Trier des données en fonction d'une clef

La fonction *sorted(t)* fonctionne lorsque t est un tableau de nombres (tri par ordre croissant) ou un tableau de chaînes de caractères (tri par ordre alphabétique). En revanche, on obtient une erreur si on essaye de trier un tableau contenant ces deux types de données, ou si on essaye de trier un tableau de dictionnaires.

Exemples :

```
t = [5, 6, 2, 3, 1, 4]
u = ["bac", "cab", "abc"]
v = t + u
w = [{"nom": "Alice", "role": 1}, {"nom": "Bob", "role": 2}]

t_trie = sorted(t) # [1, 2, 3, 4, 5, 6]
u_trie = sorted(u) # ['abc', 'bac', 'cab']
v_trie = sorted(v) # erreur '<' non supporté entre 'int' et 'str'
w_trie = sorted(w) # erreur '<' non supporté entre 'dict' et 'dict'
```

Pour utiliser la fonction *sorted()* sur des tables de données, il faut lui indiquer comment se ramener à des valeurs comparables (nombres, chaînes de caractères, n-uplets). Pour cela, il faut définir une fonction qui prend un dictionnaire en argument et renvoie la valeur à comparer.

Exemples : on s'appuie à nouveau sur la table de données *eleves.csv* vue dans le chapitre précédent. Notre but est de trier les données en fonction du prénom.

```
def prenom(x):
    return x["prénom"]
```

On peut ensuite utiliser la fonction *sorted* sur le tableau de dictionnaires *eleves*en précisant qu'il faut utiliser la fonction *prenom* chaque fois que deux éléments doivent être comparés. On l'indique en passant *key=prenom* à la fonction *sorted*.

```
tri_eleves = sorted(eleves, key=prenom)
```

Remarque : il est possible de trier par ordre décroissant en passant l'option *reverse=True* à la fonction *sorted*.

```
tri_eleves = sorted(eleves, key=prenom, reverse=True)
```

1.3 Ordre lexicographique et stabilité

Définition : *on appelle ordre lexicographique un tri prenant en compte plusieurs critères. Le tri se fait selon un premier critère, en cas d'égalité sur ce critère, le tri est fait sur le second, etc..*

Par exemple, si on fait une liste des élèves du lycée en fonction de leur classe, il peut être intéressant de trier aussi par ordre alphabétique le nom des élèves faisant parti de la même classe.

La comparaison de Python réalise un ordre lexicographique lorsqu'on l'applique à un tuple.

Exemples :

```
(1, 3) < (1, 7) # True
(1, 7) < (1, 4) # False
(1, 7) < (2, 4) # True
(1, 'bac') < (2, 'cab') # True
```

On peut utiliser cette propriété pour trier notre table de données selon plusieurs critères.

Exemples : on souhaite trier les données *eleves.csv* en fonction du prénom, puis (comme il y a deux « .Alan ») en fonction de l'année.

```python
def prenom_puis_annee(x):
    return x["prénom"], x["année"]

liste_triee = sorted(eleves, key=prenom_puis_annee)
```

Remarque : supposons que la liste ait déjà été triée en fonction de l'année. Si on trie ensuite selon le prénom, on dira que le tri est *stable* si élèves ayant le même prénom gardent *leur position relative*, c'est à dire que les élèves ayant le même prénom restent trié en fonction de l'année.
Le tri proposé par Python est stable. Le tri par insertion est également stable, mais le tri par sélection ne l'est pas.

Dans le cas où le tri est stable, il est possible d'obtenir un ordre lexicographique en faisant des tris successifs par ordre inverse de priorité.

1.4 Exemple d'application

L'algorithme des plus proches voisins travaille à partir d'une table de données et nécessite de déterminer les k lignes « les plus proches » de point cherché. Une manière simple consiste à trier l'intégralité de la table par ordre de proximité avec le point cherché, pour ensuite prendre les k premières lignes. (Cette méthode n'est pas très efficace car on doit trier probablement un grand nombre de lignes pour n'en conserver qu'un nombre très limité, k).

2 Fusion de tables

2.1 Présentation

Il est fréquent de se retrouver avec plusieurs jeux de données, qu'on souhaiterait combiner en une seule table, afin de pouvoir ensuite utiliser les opérations définies précédemment : filtrage, sélection de colonnes, agrégation et tris.

Il existe plusieurs façon de « fusionner » des données en tables. Selon l'opération que l'on souhaite effectuer, des précautions particulières sont à prendre pour ne pas introduire d'incohérences dans les données.

2.2 Réunion de tables

Une première opération naturelle est de vouloir mettre dans une même table les données de deux tables existantes. *Lorsque toutes les tables possèdent la même structure, c'est à dire qu'elles ont les mêmes attributs (en nom et en nombre) et que les attributs de même nom sont de même type, alors il est possible d'effectuer la réunion de ces tables.* Cette opération de réunion est une simple concaténation des tableaux de données.

Exemple : sur la plate-forme des données publiques françaises, on peut trouver des fichiers CSV recensant les listes des prénoms d'enfants nés dans certaines villes. Par exemple, on peut trouver les données concernant Rennes à l'adresse suivante : https ://www.data.gouv.fr/fr/datasets/prenoms-a-rennes/
Une fois décompressé, on obtient un fichier CSV pour chacune des années de 2007 à 2015. Les attributs sont l'année de naissance, le code de la commune, le nom de la commune, le sexe, le prénom et le nombre d'enfants nés avec ce prénom.

ANNAISS	CODCOM	LBCOM	SEXE	PRN	NBR
2007	35238	RENNES	FEMME	Emma	50
...	...	...	...	...	...

Remarque : l'attribut « année de naissance » est noté différemment pour le fichier de 2008 et les fichiers de 2007 à 2014 font figurer l'attribut « mois de naissance », mais sans les données correspondantes. Il est facile de modifier ces fichiers pour obtenir les fichiers *prenoms2007.csv*, *prenoms2008.csv*, etc. ayant tous exactement la mise en forme présentée ci-dessus.

Le code suivant permet alors de charger les données des années 2007 et 2008 et de les réunir dans une même table.

```
import csv
f2007 = open("prenoms2007.csv")
f2008 = open("prenoms2008.csv")
t2007 = list(csv.DictReader(f2007, delimiter = ";"))
t2008 = list(csv.DictReader(f2008, delimiter = ";"))
f2007.close()
f2008.close()

t20078 = t2007 + t2008 # concaténation des données de 2007 et des
    données de 2008
```

L'opérateur de concaténation « + » est toujours autorisé entre deux tableaux en Python. Mais dans le cas des tables de données, la réunion n'a de sens que si les opérations sur les tables sont toujours possibles. Par exemple, il est possible de sélectionner les lignes correspondant aux prénoms des filles pour les tableaux *t2007*, *t2008* mais également *t20078*.

```
filles2007 = [f for f in t2007 if f["SEXE"] == "FEMME"]
filles2008 = [f for f in t2008 if f["SEXE"] == "FEMME"]
filles20078 = [f for f in t20078 if f["SEXE"] == "FEMME"]
```

En revanche, si on avait concaténé les tables *t2007* et *eleves* (du chapitre précédent), on aurait obtenu une erreur *KeyError : 'SEXE'*.

```
mauvaise_table = t2007 + eleves # concaténation des données de 2007
    et des données de 2008

test = [f for f in mauvaise_table if f["SEXE"] == "FEMME"]
```

Il faut donc retenir que *réunir les valeurs de deux tables est une opération légitime, à condition que les tables aient les mêmes attributs et les mêmes domaines de valeurs pour leurs attributs.*

2.3 Opération de jointure

Lorsque des tables ont des attributs différents, mais au moins un attribut en commun, il est possible de réaliser une opération de jointure.

Exemple : on reprend le tableau *eleves* vu au chapitre précédent.

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.

On considère que *Brian*, *Linus* et *Blaise* ont rendu leur projet, qui a été noté. Les données sont stockées dans une autre table.

prénom	note
Linus	16
Brian	17
Blaise	14

On souhaite maintenant éditer un relevé de note, c'est à dire une table donnant pour chaque élève qui a eu une note son prénom, sa date de naissance, le nom du projet et sa note dans le projet. La première chose à faire est de définir une fonction qui, étant donnée une ligne de la table *eleves* et une ligne de la table *notes*, renvoie un nouveau dictionnaire représentant la ligne du résultat.

```
def fusion(e, n):
    return {"prénom": e["prénom"], "jour": e["jour"], "mois": e["
    mois"], "année": e["annee"], "projet": e["projet"], "note": n["
    note"]}
```

Ici, on choisit d'ignorer la clef *prénom* du dictionnaire n car nous supposons que les deux dictionnaires portent sur le même élève.

L'algorithme de *fusion de tables* que l'on souhaite appliquer est une double boucle.

```
t = []
for e in eleves:
    for n in notes:
        if e["prénom"] == n["prénom"]:
            t.append(fusion(e, n))
```

L'application de cet algorithme crée le tableau t comprenant trois lignes.

prénom	jour	mois	année	projet	note
Brian	1	1	1942	Programmer avec style.	17
Linus	28	12	1969	Un petit système d'exploitation.	16
Blaise	19	6	1623	Machine arithmétique.	14

Remarque : cet algorithme peut également être codée de façon plus compacte et élégante à l'aide d'une double compréhension.

```
t = [ fusion(e, n) for e in eleves for n in notes if e["prénom"] ==
     n["prénom"] ]
```

La jointure de table est un outil très puissant dans la gestion des données tabulées. Par exemple, si on revient sur les tables des prénoms de 2007 et 2008, on peut facilement calculer la table suivante qui montre lenombre de naissances pour chaque prénom en 2007 et 2008, ce qui permet d'étudier l'évolution.

```
evolution = [ {"PRN": t1["PRN"], "NBR2007": t1["NBR"], "NBR2008":
    t2["NBR"]} for t1 in t2007 for t2 in t2008 if t1["PRN"] == t2["
    PRN"] ]
```

2.4 Utilisation d'un identifiant unique

Lors de la production du relevé de notes ou lors de l'évolution des prénoms dans la partie précédente, nous avons supposé implicitement que les prénoms

étaient uniques. C'est vrai pour les fichiers *prenoms2007.csv* et *prenoms2008.csv*, mais par pour le fichier *eleves.csv* où le prénom *Alan* est présent deux fois. Considérons la petite table *note* suivante :

prénom	note
Alan	15

La jointure entre la table *eleves* et la table *note* donnera la table suivante :

prénom	jour	mois	année	projet	note
Alan	23	6	1912	Déchiffrer des codes secrets.	15
Alan	1	4	1922	Ce qu'un programmeur doit savoir.	15

Alors qu'un seul des projets avait été noté à 15, la jointure comporte deux lignes car le test *if e["prénom"] == n["prénom"]* est vrai deux fois. Cette situation n'est pas satisfaisante.

De façon générale, il est possible que des données « réalistes » ne suffisent pas à identifier les éléments de manière unique (personnes ayant le même nom et prénom, voire date de naissance, l'IBSN permet d'identifier un livre, mais il peut être en plusieurs exemplaires dans une bibliothèque, etc.).

Une manière de résoudre ce problème est d'associer au moment du chargement un entier unique pour chaque entrée de la table. Le code suivant permet de rajouter un attribut *id* (pour « identifiant ») dont la valeur est un entier incrémenté à chaque ligne.

```
compteur = 0
for e in eleves:
    e["id"] = compteur
    compteur += 1
```

On obtient alors le tableau suivant :

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

Le professeur peut alors rendre sa note en évitant les ambiguïté en utilisant l'attribut *id*.

id	note
7	15

La jointure peut alors se faire sur l'attribut id (qui devra avoir été ajouté aussi à la table *notes*).

```
1  t = [ fusion(e, n) for e in eleves for n in notes if e["id"] == n["
       id"] ]
```

Le tableau obtenu ne possède bien qu'une seule ligne.

id	prénom	jour	mois	année	projet	note
7	Alan	1	4	1922	Ce qu'un programmeur doit savoir.	15