Trier une table

sorted() et .sort()

Pour trier en ordre croissant de façon simple et facile, le type `list` offre une méthode `.sort()` qui permet de modifier un tableau en le triant.

```
>>> tab = [5, 2, 3, 1, 4]
>>> tab.sort()
>>> tab
[1, 2, 3, 4, 5]
```

Notr que `tab.sort()` a modifié le tableau `tab` et a renvoyé `None`.

Dans cet exemple, on a trié un tableau de nombres entiers. On peut faire la même chose avec un tableau de nombres decimaux (`float`) ou de chaines de caractères. ⚠️ Les chaines de caractères sont triées par ordre lexicographique[1].

```
>>> tab = ['pomme', 'banane', 'orange', 'fraise']
>>> tab.sort()
>>> tab
['banane', 'fraise', 'orange', 'pomme']
```

Noter que `.sort()` est une méthode reservée aux tableaux, c'est-à-dire aux variables de type `list`. Elle ne s'applique pas aux vp-uplets ou aux dictionnaires :

```
>>> puplet  = (5, 2, 3, 1, 4)
>>> puplet.sort()
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'sort'
>>> dico = {1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'}
>>> dico.sort()
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
AttributeError: 'dict' object has no attribute 'sort'
```

Au contraire de `.sort()`, la fonction `sorted()` accepte n'importe quel itérable et renvoie un nouveau tableau trié :

```
>>> sorted((5, 2, 3, 1, 4))
[1, 2, 3, 4, 5]
>>> sorted([5, 2, 3, 1, 4])
[1, 2, 3, 4, 5]
>>> sorted({'un':1, 'deux':2, 'trois':3 })
['deux', 'trois', 'un']
```

Noter que `sorted()` renvoie toujours un tableau, même pour trier un p-uplet ou un dictionnaire. Et dans le cas d'un dictionnaire, ce sont les clés qui sont triées et renvoyées.

⚠️ Attention pour les tableaux, à la différence de `.sort()`, le tableau trié n'est pas modifié, c'est un nouveau tableau trié qui renvoyé par `sorted()` :

```
>>> tab = [5, 2, 3, 1, 4]
>>> sorted(tab)
[1, 2, 3, 4, 5]
>>> tab
[5, 2, 3, 1, 4]
>>> tab = [5, 2, 3, 1, 4]
>>> tab_trie = sorted(tab)
>>> tab
[5, 2, 3, 1, 4]
>>> tab_trie
[1, 2, 3, 4, 5]
```

On peut trier sur des types construits, dans ce cas le tri est fait par ordre des éléments. Par exemple, pour trier notre tableau de tableaux `pays`.

```
>>> pays
[['France', 'Paris', '68'],
 ['Espagne', 'Madrid', '48'],
 ['Italie', 'Rome', '60']]
>>> sorted(pays)
['Espagne', 'Madrid', '48'], 'France', 'Paris', '68'], ['Italie', 'Rome', '60']]]
```

Dans ce cas, le tri se fait en comparant les premières valeurs de chaque sous-tableau : `'Espagne' < 'France'< 'Italie'`.

Par contre, on ne peut pas trier un tableau de dictionnaires :

```
>>> pays = [{'Capitale': 'Paris', 'Pays': 'France', 'Population (ml)': '68'},
    {'Capitale': 'Madrid', 'Pays': 'Espagne', 'Population (ml)': '48'},
    {'Capitale': 'Rome', 'Pays': 'Italie', 'Population (ml)': '60'}]
>>> sorted(pays)
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
TypeError: '<' not supported between instances of 'dict' and 'dict'
```

On ne peut pas comparer les dictionnaire, il faut préciser une clé de tri.

Paramètre key

`.sort()` et `sorted()` acceptent un paramètre nommé `key` permettant de spécifier une fonction à appeler sur chaque élément du tableau afin d'effectuer des comparaisons.

Par exemple, on peut modifier l'ordre de tri d'un tableau de nombres au format `str` :

```
>>> sorted(['5', '3', '1', '11', '21'])
['1', '11', '21', '3', '5']
```

en précisant que les données doivent être converties en entier par la fonction `int()` avant d'être triées :

```
>>> sorted(['5', '3', '1', '11', '21'], key=int)
['1', '3', '5', '11', '21']
```

De la même façon, le paramètre `key` permet de trier une table en précisant les colonnes selon lesquelles on veut trier. Par exemple, si on veut trier le tableau de tableaux des pays selon leur population :

```
>>> pays
[['France', 'Paris', '68'],
 ['Espagne', 'Madrid', '48'],
 ['Italie', 'Rome', '60']]
```

On peut écrire une fonction `popul` qui renvoie le champs population de chaque pays converti en nombre entier :

```
def popul(x):
    return int(x[2])
```

et qui sert de clé de `sorted()` :

```
>>> sorted(pays, key=popul)
[['Espagne', 'Madrid', '48'], ['Italie', 'Rome', '60'],  ['France', 'Paris', '68']]
```

Ou bien l'écrire directement dans une fonction lambda :

```
>>> sorted(pays, key=lambda x:int(x[2]))
[['Espagne', 'Madrid', '48'], ['Italie', 'Rome', '60'],  ['France', 'Paris', '68']]
```

De la même façon, une fonction lambda va permettre de trier le tableau de dictionnaires en ordre croissant de population :

```
>>> pays = [{'Capitale': 'Paris', 'Pays': 'France', 'Population (ml)': '68'}, \
            {'Capitale': 'MAdrid', 'Pays': 'Espage', 'Population (ml)': '48'}, \
            {'Capitale': 'Rome', 'Pays': 'Italie', 'Population (ml)': '60'}]

>>> sorted(pays, key=lambda x:int(x['Population (ml)']))
[{'Capitale': 'Madrid', 'Pays': 'Espagne', 'Population (ml)': '48'}, {'Capitale': 'Rome', 'Pays':
'Italie', 'Population (ml)': '60'}, {'Capitale': 'Paris', 'Pays': 'France', 'Population (ml)':
'68'}]
```

Paramètre reverse

`.sort()` et `sorted()` acceptent aussi un paramètre nommé `reverse` avec une valeur booléenne. Par défaut, `reverse` est `False`, c'est-à-dire qu'on tri en ordre croissant, mais on peut le changer pour indiquer un ordre décroissant des tris. Par exemple, pour avoir les pays dans par population décroissante :

```
>>> sorted(pays, key=lambda x:int(x[2]), reverse=True)
[['France', 'Paris', '68'], ['Italie', 'Rome', '60'], ['Espagne', 'Madrid', '48']]
```

Exercice corrigé

On a importé un tableau de dictionnaires des codes postaux avec :

```
with open('laposte_hexasmal.csv', 'r', encoding='utf-8-sig') as f:
    codes = list(csv.DictReader(f, delimiter=';'))
```

1. Ecrire les fonctions `plus_petit_code` et `plus_grand_code` qui renvoient la commune qui a le plus petit code postal et celle qui a le plus grand code postal.
2. Ecrire les fonctions `plus_grande_latitude` qui renvoie la commune qui a la plus grande latitude.

Réponse

Exercice corrigé - Pour aller plus loin

1. Ecrire les fonctions `plus_loin(longA, latA)` qui renvoie la commune la plus éloignée du point GPS de coordonnées (longA, latA).

Exemple:

```
>>> plus_loin(0,0)
{'code_commune_insee': '98612',
'code_postal': '98620',
'coordonnees_gps': '-14.270411199, -178.155263035',
'libelle_d_acheminement': 'SIGAVE',
'ligne_5': '',
'nom_de_la_commune': 'SIGAVE'}
```

Note : La distance en mètres entre les points de coordonnées (;) et (;) est donnée par la formule de Pythagore :

-
-
-
-

Source : http://villemin.gerard.free.fr/aGeograp/Distance.htm

Réponse

---

1. On commence par comparer les codes Unicode du premier caractère de chaque chaîne, puis en cas d'égalité le second caractère, et ainsi de suite comme dans un dictionnaire. Attention aux majuscules et aux nombres, '11' est plus petit que '2' ! ↩