Python – Tables de données

Attrapez-les tous !

Les Pokémons sont des créatures apparues dans la série de jeux éponyme sortie en 1996. Chaque Pokémon est défini par un ensemble de statistiques, certaines connues et d'autres secrètes. Nous représenterons un Pokémon à l'aide de son nom, ses types, son nombre de points de vie (HP), sa force d'attaque, sa force de défense, sa vitesse et la génération à laquelle il appartient. Chaque Pokémon sera représenté par un dictionnaire :

```
bulbizarre = {'Nom': 'Bulbizarre', 'Types': ['Plante', 'Poison'], 'HP': 45,
              'Attaque': 49, 'Défense': 49, 'Vitesse': 45, 'Génération': 1}
salameche = {'Nom': 'Salamèche', 'Types': ['Feu'], 'HP': 39,
             'Attaque': 52, 'Défense': 43, 'Vitesse': 65, 'Génération': 1}
```

On considère la fonction suivante qui permet d'obtenir le nom d'un Pokémon :

```
def nom(pokemon):
    return pokemon['Nom']
```

```
>>> nom(bulbizarre)
'Bulbizarre'
>>> nom(salameche)
'Salamèche'
```

Exercice 1 : Écrire une fonction `vie(pokemon)` qui renvoie le nombre de HP de `pokemon`.

```
>>> vie(bulbizarre)
45
>>> vie(salameche)
39
```

Exercice 2 : Écrire une fonction `types(pokemon)` qui renvoie la liste des types de `pokemon`.

```
>>> types(bulbizarre)
['Plante', 'Poison']
>>> types(salameche)
['Feu']
```

Exercice 3 : Écrire les 4 fonctions `attaque(p)`, `defense(p)`, `vitesse(p)` et `generation(p)` qui renvoient les valeurs correspondantes.

```
>>> attaque(bulbizarre)
49
>>> defense(bulbizarre)
49
>>> vitesse(bulbizarre)
45
>>> generation(bulbizarre)
1
```

Importation d'un Pokédex

Un pokédex est une liste de Pokémon. Nous allons créer un pokédex avec la liste de tous les Pokemons de la première génération à la sixième génération. Cette liste est contenue dans le fichier `pokemons.csv`. Nous pouvons utiliser le module `csv` pour importer ces données :

```
>>> import csv
>>> fichier = open("pokemons.csv", 'r', encoding='utf-8')
>>> pokedex = list(csv.DictReader(fichier, delimiter=','))
>>> fichier.close()
>>> pokedex[0]
OrderedDict([('Nom', 'Bulbizarre'), ('Type 1', 'Plante'), ('Type 2', 'Poison'),
         ('HP', '45'), ('Attaque', '49'), ('Défense', '49'), ('Vitesse', '45'),
         ('Génération', '1')])
```

On ouvre le fichier et ensuite, on utilise la commande `csv.DictReader(fichier, delimiter=',')` qui permet de lire un fichier CSV et renvoie une liste de dictionnaires correspondant à chaque ligne du tableau. Les descripteurs sont utilisés comme attributs des dictionnaires. On peut remarquer que dans le dictionnaire obtenu, il y n'y a pas le champs `'Types'`, mais au contraire deux champs différents. De même, toutes les valeurs numériques sont des textes. C'est là que l'étape de validation est importante.
Pour cela, nous utiliserons le code suivant :

```
import csv

def validation(pokemon):
    for k in ["HP", "Attaque", "Défense", "Vitesse", "Génération"]:
        pokemon[k] = int(pokemon[k])
    types = [pokemon["Type 1"]]
    if pokemon["Type 2"] != "":
        types.append(pokemon["Type 2"])
    pokemon["Types"] = types
    del pokemon["Type 1"]
    del pokemon["Type 2"]

def importation(nom_fichier):
    with open(nom_fichier, 'r', encoding='utf-8') as fichier:
        pokedex = []
        for pokemon in csv.DictReader(fichier, delimiter = ','):
            validation(pokemon)
            pokedex.append(pokemon)
    return pokedex

pokedex = importation("pokemons.csv")
```

Dans la fonction `validation`, les valeurs numériques sont converties en entiers et un nouvel attribut est ajouté à la place des deux attributs de types.

```
>>> pokedex[0]
OrderedDict([('Nom', 'Bulbizarre'), ('HP', 45), ('Attaque', 49), ('Défense', 49),
             ('Vitesse', 45), ('Génération', 1), ('Types', ['Plante', 'Poison'])])
```

Il est alors possible de faire des opérations sur le pokédex. Par exemple, la commande suivante répond **True** si, et seulement si, le Pokémon cherché se trouve dans le pokédex :

```
def est_dans(nom_pokemon, pokedex):
    for p in pokedex:
        if nom(p) == nom_pokemon:
            return True
    return False
```

```
>>> est_dans("Bulbizarre", pokedex)
True
>>> est_dans("Goldorak", pokedex) # Pas un Pokémon
False
```

Exercice 4 : Écrire une fonction `cherche(nom_pokemon, pokedex)` qui renvoit le dictionnaire correspondant au Pokémon demandé, s'il est dans le pokédex et **None** sinon.

```
>>> cherche("Bulbizarre", pokedex)
OrderedDict([('Nom', 'Bulbizarre'), ('HP', 45), ('Attaque', 49), ('Défense', 49),
             ('Vitesse', 45), ('Génération', 1), ('Types', ['Plante', 'Poison'])])
>>> cherche("Goldorak", pokedex) # None ne s'affiche pas
>>>
```

Extraction d'une partie des données

La fonction suivante permet d'extraire la liste des noms du pokedex :

```
def liste_noms(pokedex):
    noms = []
    for p in pokedex:
        noms.append(nom(p))
    return noms
```

Il est possible de l'écrire de façon plus simple en utilisant une définition par compréhension :

```
def liste_noms(pokedex):
    return [nom(p) for p in pokedex]
```

Pour les exercices suivants, vous pouvez utiliser la forme que vous préférez.

Exercice 5 : Écrire une fonction `selection_par_type(t, pokedex)` qui renvoie la liste de tous les Pokémons du pokédex qui ont le type `t`.

```
>>> selection_par_type("Fée", pokedex)
[OrderedDict([('Nom', 'Mélofée'), ('HP', 70), ('Attaque', 45),...
>>> len(selection_par_type("Fée", pokedex))
40
>>> liste_noms(selection_par_type("Fée", pokedex))
['Mélofée', 'Mélodelfe', 'Rondoudou', 'Grodoudou', 'M. Mime',...
```

Exercice 6 : Écrire une fonction `selection_par_generation(gen, pokedex)` qui renvoie la liste de tous les Pokémons qui sont de la génération donnée.

```
>>> liste_noms(selection_par_generation(1, pokedex))
['Bulbizarre', 'Herbizarre', 'Florizarre', 'Méga-Florizarre',...
```

Exercice 7 : Écrire une fonction `plus_fort(pokedex)` qui renvoie le Pokémon avec la valeur d'attaque maximale contenue dans le Pokédex. S'il y en a plusieurs, le premier rencontré est renvoyé.

```
>>> plus_fort(selection_par_generation(1, pokedex))
OrderedDict([('Nom', 'Méga-Mewtwo X'), ('HP', 106), ('Attaque', 190),
             ('Défense', 100), ('Vitesse', 130), ('Génération', 1),
         ('Types', ['Psy', 'Combat'])])
```

Exercice 8 : Écrire une fonction `plus_rapide_que(pokemon, pokedex)` qui renvoie la liste de tous les Pokémons du pokédex qui sont strictement plus rapides que `pokemon`.

Tri des données

Pour pouvoir exploiter plus facilement les données, il est souvent utile de les trier. Nous allons utiliser les fonctions prédéfinies par Python. Pour trier une liste, les deux possibilités sont `liste.sort()` qui modifie la liste pour que les éléments soient dans l'ordre croissant, et `sorted(liste)` qui renvoie une nouvelle liste triée.

```
>>> liste = [7, 9, 1, 8]
>>> liste.sort()
>>> liste
[1, 7, 8, 9]
>>> liste = [7, 9, 1, 8]
>>> sorted(liste)
[1, 7, 8, 9]
>>> liste
[7, 9, 1, 8]
```

Pour une liste de dictionnaires, il faut absolument donner la clef selon laquelle les données doivent être triées. Le paramètre donné pour l'argmuent `key` doit être une fonction qui prend n'importe quel élément de la liste et renvoie une valeur. Les éléments seront classés par rapport au résultat de cette fonction. Nous allns utiliser la syntaxe `lambda x: f(x)` qui veut dire qu'on associe à `x` la valeur `f(x)`.

```
>>> d = [{'Nom': 'Bob', 'Age': 56, 'Sexe': 'H'},\
         {'Nom': 'Lidia', 'Age': 35, 'Sexe': 'F'},\
         {'Nom': 'Hachim', 'Age': 26, 'Sexe': 'H'}]
>>> sorted(d, key=lambda p: p["Nom"]) # tri par rapport au nom
[{'Nom': 'Bob', 'Age': 56, 'Sexe': 'H'},
 {'Nom': 'Hachim', 'Age': 26, 'Sexe': 'H'},
 {'Nom': 'Lidia', 'Age': 35, 'Sexe': 'F'}]
>>> sorted(d, key=lambda p: p["Age"]) # par rapport à l'age
[{'Nom': 'Hachim', 'Age': 26, 'Sexe': 'H'},
 {'Nom': 'Lidia', 'Age': 35, 'Sexe': 'F'},
 {'Nom': 'Bob', 'Age': 56, 'Sexe': 'H'}]
>>> sorted(d, key=lambda p: p["Sexe"]) # par rapport au sexe
[{'Nom': 'Lidia', 'Age': 35, 'Sexe': 'F'},
 {'Nom': 'Bob', 'Age': 56, 'Sexe': 'H'},
 {'Nom': 'Hachim', 'Age': 26, 'Sexe': 'H'}]
```

```
 >>> sorted(d, key=lambda p: (p["Sexe"], p["Age"])) # (sexe, age)
[{'Nom': 'Lidia', 'Age': 35, 'Sexe': 'F'},
 {'Nom': 'Hachim', 'Age': 26, 'Sexe': 'H'},
 {'Nom': 'Bob', 'Age': 56, 'Sexe': 'H'}]
```

Dans le dernier exemple, vous pouvez remarquer qu'il est possible de donner un n-uplet pour définir plusieurs clefs de tri. En priorité, on trie par rapport à la première clef et en cas d'égalité, on regarde la suivante, et ainsi de suite. Cela permet d'affiner les tris de façons plus précises.
Enfin, il est possible de trier dans l'ordre décroissant en rajoutant le paramètre `reverse=True` :

```
>>> sorted(d, key=lambda p: p["Age"], reverse=True)
[{'Nom': 'Bob', 'Age': 56, 'Sexe': 'H'},
 {'Nom': 'Lidia', 'Age': 35, 'Sexe': 'F'},
 {'Nom': 'Hachim', 'Age': 26, 'Sexe': 'H'}]
```

Pour tous les exercices, nous utiliserons **sorted** pour renvoyer un nouveau pokédex et ne pas modifier l'original. Puisque nous avons déjà défini des fonctions qui renvoient le nom, l'attaque ou le nombre de HP d'un Pokémon, il est possible de les utiliser à la place de **lambda** :

```
def tri_par_nom(pokedex):
    return sorted(pokedex, key=nom)
```

```
>>> tri_par_nom(pokedex)
[OrderedDict([('Nom', 'Abo'), ('HP', 35), ...]),
 OrderedDict([('Nom', 'Abra'), ('HP', 25), ...]),
 OrderedDict([('Nom', 'Absol'), ('HP', 65), ...]),
 OrderedDict([('Nom', 'Aflamanoir'), ('HP', 85), ...]),
 ...
```

Exercice 9 : Écrire une fonction `tri_par_attaque(pokedex)` qui renvoie un nouveau pokédex avec les Pokémons classés par l'ordre décroissant de leur niveau d'attaque. Le premier est celui qui a la plus forte attaque et le dernier est celui qui a la plus faible.

Exercice 10 : Écrire une fonction `tri_par_attaque_defense_vie(pokedex)` qui renvoie un nouveau pokédex avec les Pokémons classés par l'ordre décroissant de leur niveau d'attaque, de défense et de vie. Vous pourrez soit faire une fonction **lambda** ou alors définir une nouvelle fonction qui renvoit le triplet (a,d,v) composé de l'attaque, la défense et la vie du Pokémon donné en paramètre.

Fusion et jointure de tables

Puisque les pokédex sont des listes, on peut les concaténer pour obtenir un nouveau pokédex. Mais cela peut créer des doublons. Pour éviter cela, on peut partir du premier pokédex et ajouter ceux du deuxième s'ils ne sont pas déjà dans le premier.

Exercice 11 : Écrire une fonction `fusion_pokedex(pokedex1, pokedex2)` qui renvoie un nouveau pokédex qui est la fusion des deux autres. Vous pouvez utiliser `pokedex1.copy()` pour obtenir une copie du premier pokédex et ainsi vous assurer que vous ne le modifiez pas.

```
>>> bulbizarre = cherche("Bulbizarre", pokedex)
>>> pikachu = cherche("Pikachu", pokedex)
>>> salameche = cherche("Salamèche", pokedex)
>>> fusion_pokedex([pikachu,bulbizarre], [salameche,pikachu])
[OrderedDict([('Nom', 'Pikachu'), ('HP', 35), ...]),
 OrderedDict([('Nom', 'Bulbizarre'), ('HP', 45), ...]),
 OrderedDict([('Nom', 'Salamèche'), ('HP', 39), ...])]
```

Parmi les Pokémons, certains sont légendaires. La liste de ces Pokémons est stockée dans `pokemons_legendaires.csv`. Les Pokémons légendaires peuvent être importés ainsi :

```
legendaires = importation("pokemons_legendaires.csv")
```

Exercice 12 : Écrire une fonction `ajout_legendaires(pokedex, legendaires)` qui modifie le pokédex en rajoutant le descripteur `"Légendaire"` à chaque Pokémon. S'il est légendaire, alors le champs vaut **`True`**, sinon il vaut **`False`**.

```
>>> ajout_legendaires(pokedex, legendaires)
>>> cherche("Mew", pokedex)
OrderedDict([('Nom', 'Mew'), ('HP', 100), ('Attaque', 100), ('Défense', 100),
('Vitesse', 100), ('Génération', 1), ('Types', ['Psy']), ('Légendaire', False)])
>>> cherche("Mewtwo", pokedex)
OrderedDict([('Nom', 'Mewtwo'), ('HP', 106), ('Attaque', 110), ('Défense', 90),
('Vitesse', 130), ('Génération', 1), ('Types', ['Psy']), ('Légendaire', True)])
```

Exercice 13 : Écrire une fonction `legendaire(pokemon)` qui renvoie le booléen indiquant si le Pokémon est légendaire ou pas.

```
>>> legendaire(cherche("Mew", pokedex))
False
>>> legendaire(cherche("Mewtwo", pokedex))
True
```

Exercice 14 : À l'aide des fonctions précédentes, ou de nouvelles, déterminer de quelle génération vient le Pokémon légendaire de type Feu le plus fort.

Table des types

Les types des Pokémons influencent l'effet des attaques et de la défense. Certains Pokémons sont sensibles à certains types et résistant à d'autres, ce qui peut amplifier ou réduire les effets d'une attaque. Par exemple, le type "Plante" est sensible aux attaques "Insecte", alors que "Spectre" est résistant à ces attaques. Considérons une attaque de type "Insecte" qui a une force de 42. Alors :

- Joliflor est de type Plante. La force de l'attaque sera doublée. Elle sera de : $42 \times 2 = 84$.
- Feuforêve est de type Spectre. La force sera divisée par deux. Elle sera de : $42 \times 0{,}5 = 21$.
- Desséliande est de type Plante et Spectre. Les effets se cumulent. La force de l'attaque sera donc : $42 \times 2 \times 0{,}5 = 42$.

Toutes les sensibilités aux différents types sont précisées dans le fichier `table_types.csv`. Le tableau ci-contre est un extrait de cette table.

Type cible	Insecte	Normal	Plante	Spectre
Insecte			0,5	
Normal				0
Plante	2		0,5	
Spectre	0,5	0		2

Chaque entrée indique la sendibilité du type de la cible par rapport aux différentes attaques. Ainsi, on voit que le type Plante est sensible au type Insecte et résistant au type Plante. on remarque également que les types Spectre et Normal sont insensibles l'un par rapport à l'autre. Toutes les cases vides correspondent à des combinaisons où il n'y a pas d'effet sur l'attaque.
Pour importer cette table, nous allons utiliser un dictionnaire de dictionnaires qui indiquera pour chaque type de la cible, quels sont les effets de chaque type d'attaque.

```
def importation_table_types(nom_fichier):
    with open(nom_fichier, 'r', encoding='utf-8') as fichier:
        sensibilites = dict()
        for entree in csv.DictReader(fichier, delimiter = ','):
            type_cible = entree["Type cible"]
            del entree["Type cible"]
            for k in entree.keys():
                if entree[k] == "":
                    entree[k] = 1
                else:
                    entree[k] = eval(entree[k])
            sensibilites[type_cible] = entree
    return sensibilites

sensibilites = importation_table_types("table_types.csv")
```

Pour chaque entrée, on enlève la clef `"Type cible"` mais on s'en sert pour définir le nouveau dictionnaire. Lorsqu'il n'y a pas de valeurs indiquées, on met 1, sinon on la convertit en entier ou en flottant, en fonction de la valeur.

```
>>> sensibilites["Plante"]
OrderedDict([('Acier', 1), ('Combat', 1), ('Dragon', 1), ('Eau', 0.5),
  ('Electrik', 0.5), ('Fée', 1), ('Feu', 2), ('Glace', 2), ('Insecte', 2),
  ('Normal', 1), ('Plante', 0.5), ('Poison', 2), ('Psy', 1), ('Roche', 1),
  ('Sol', 0.5), ('Spectre', 1), ('Ténèbres', 1), ('Vol', 2)])
>>> sensibilites["Plante"]["Insecte"] # sensibilité de Plante aux attaques Insecte
2
```

Exercice 15 : Écrire une fonction `coefficient_types(type_attaque, pokemon)` qui renvoie le coefficient par lequel est multiplié la force de l'attaque de type donnée sur le Pokémon. Cette fonction pourra utiliser le dictionnaire `sensibilites` défini au dessus.

```
>>> coefficient_types("Insecte", cherche("Joliflor", pokedex))
2
>>> coefficient_types("Insecte", cherche("Feuforêve", pokedex))
0.5
>>> coefficient_types("Insecte", cherche("Desséliande", pokedex))
1.0
```

Exercice 16 : Écrire une fonction `pokemons_sensibles(type_attaque, pokedex)` qui renvoie la liste de tous les Pokémons qui sont sensibles à l'attaque de type donné. C'est-à-dire, tous les Pokémons qui ont un coefficient multiplicateur strictement supérieur à 1 avec une attaque du type donné.

Pour aller plus loin

Afin d'exporter un pokédex dans un nouveau fichier, vous pouvez utiliser la fonction suivante :

```
def export_pokedex(nom_fichier, pokedex, descripteurs=None):
    with open(nom_fichier, 'w', newline='') as fichier:
        if not descripteurs:  # On prend les descripteurs du pokedex
            descripteurs = list(pokedex[0].keys())
        writer = csv.DictWriter(fichier, descripteurs,
                                delimiter=',',  # pour séparer les champs
                                extrasaction='ignore',
                                quoting=csv.QUOTE_MINIMAL) # pour éviter les "
        writer.writeheader() # Pour écrire les descripteurs
        for pokemon in pokedex:
            writer.writerow(pokemon) # Pour chaque ligne
```

Ainsi `export_pokedex('pokedex.csv', pokedex)` va exporter tout le pokédex dans le fichier `pokedex.csv`. Si on rajoute des descripteurs, on peut choisir ceux qui seront exportés. Par exemple `export_pokedex('pokedex.csv', pokedex, ['Nom', 'Génération'])` exportera uniquement les deux descripteurs indiqués. Le paramètre `extrasaction='ignore'` permet d'éviter les messages d'erreurs si tous les descripteurs ne sont pas exportés.
Vous pouvez utiliser cette fonction pour exporter vos propres pokédex.