Opendata et données structurées

I – Définitions

OPENDATA : (**données ouvertes en français)**. Concept selon lequel les données et l'information sont un bien commun à partager et diffuser.

Donnée structurée : une donnée structurée est une donnée présentée selon un format connu et fixe. Cela permet d'échanger, de stocker et de retrouver facilement l'information.

Identifiant	Type	Utilisation	Couleur
0	Voiture	Sport	Rouge
1	Hélico	Tourisme	Marron
2	Voiture	Sport	Rouge

Attention, le vocabulaire suivant n'est pas forcément fixé exactement de la même façon par tous.

Descripteur : les différentes colonnes, On parle aussi d'**attributs**.

En-tête : la première ligne (celle qui fournit l'ordre et le nom des attributs).

Enregistrement : l'une des lignes décrivant l'un des objets enregistrés. On parle aussi de **p-uplet**.

Champ : l'une des cases d'un enregistrement.

Valeur : le contenu d'un des champs.

Collection : l'ensemble de l'en-tête et des enregistrements, le « tableau » dans sa totalité.

II – Collection sous forme d'un tableau de tableaux

2.1 Collection crée par extension :

```
c = [
    ['0', 'voiture', 'sport', 'rouge'],
    ['1', 'helico', 'tourisme', 'marron'],
    ['2', 'voiture', 'sport', 'rouge']
 ]
```

2.2 Collection crée par extension et ajouts :

```
c = []
c.append(['0', 'voiture', 'sport', 'rouge'])
c.append(['1', 'helico', 'tourisme', 'marr.'])
c.append(['2', 'voiture', 'sport', 'rouge'])
```

→ Avec la deuxième façon, est-ce un tableau statique ou un tableau dynamique ?

→ Comment obtenir l'enregistrement Id1 :

→ Comment obtenir la couleur de l'Id1 :

2.3 Visualiser tous les enregistrements :

```
def visualiser(c):
    '''affiche la collection c'''
    for e in c:
        print(f"{e[1]:20}{e[2]:20}{e[3]:20}")
```

→ Que va contenir **e** successivement ?

→ Que veut dire **f"{e[1]:20}"** ?

```
def visualiser(c):
    '''affiche la collection c'''
    for i in range(len(c)):
        e = c[i]
        print(f"{e[1]:20}{e[2]:20}{e[3]:20}")
```

→ Pourquoi peut-on utiliser les deux « for » ?

→ Quel est le problème du tableau de tableaux ?

III – Collection sous forme d'un tableau de tuples

```
c = [
    ('0', 'voiture', 'sport', 'rouge'),
    ('1', 'helico', 'tourisme', 'marron'),
    ('2', 'voiture', 'sport', 'rouge')
 ]
```

Quelle est la seule différence de syntaxe ?

Avantages :

→ Plus de nécessité d'avoir le même type dans chaque case pour profiter d'une exécution rapidement

→ En Python, le tuple est immuable : pas de risque de modification involontaire.

IV – Collection sous forme d'un tableau de dictionnaires

Pour chaque enregistrement, il faudra cette fois associer la clé-attribut avec la valeur voulue.

4,1 Création manuelle

```
enr = {}
enr['id'] = 0
enr['type'] = 'voiture'
enr['utilisation'] = 'sport'
enr['couleur'] = 'rouge'
```

C'est long, il serait utile de créer une fonction qui reçoit un tableau **val** contenant les valeurs dans le bon ordre et qui renvoie le dictionnaire **d** associé :

```
def creer_enregistrement(val, att):
    d = {}
    d[ att[0] ] = val[0]
    d[ att[1] ] = val[1]
    d[ att[2] ] = val[2]
    d[ att[3] ] = val[3]
    return d
```

→ Que va renvoyer l'appel suivant ?

```
>>> a = ['id','type','utilisation','couleur']
>>> v = [0, 'voiture', 'sport', 'rouge']
>>> creer_enregistrement(v, a)
```

4,2 Créer une collection-tableau c de dictionnaires à partir d'une collection de base organisée en tableau de tableaux

```
att = ['id','type','utilisation','couleur']
base = [
    ['0', 'voiture', 'sport', 'rouge'],
    ['1', 'helico', 'tourisme', 'marron'],
    ['2', 'voiture', 'sport', 'rouge']
 ]


c = []
for val in base:
    c.append( creer_enregistrement(val, att) )
```

→ Que va renvoyer **c[1]** ?

→ Que va renvoyer **c[1]['couleur']** ?

On peut aussi utiliser ceci bien entendu :

```
c = []
for i in range(len(base)):
    c.append(creer_enregistrement(base[i], att))
```

V – Fichier CSV

CSV veut dire **Comma Separated Values** : littéralement, des valeurs séparées par des virgules.
Il s'agit d'un format de fichier-texte qui peut être correctement ouvert par un tableur.
→ Chaque ligne est un enregistrement.
→ Chaque enregistrement est un texte contenant les valeurs séparées par une « virgule » et finissant par \n.
→ La première ligne peut éventuellement être l'en-tête contenant les noms des attributs.

Exemple :

```
#,Name,Type 1,Type 2,Total,HP,Attack,Defense,Sp. Atk,Sp. Def,Speed,Generation,Legendary
1,Bulbasaur,Grass,Poison,318,45,49,49,65,65,45,1,False
```

L'intérêt est de stocker les données de façon permanente. Un script Python peut alors ensuite :
→ Lire la première ligne d'en-tête pour détecter les noms des attributs
→ Lire chaque ligne pour créer un dictionnaire **enr** et le mettre dans la collection **c**.

```
def creer_collection(fichierCSV):
    '''Fonction qui renvoie un tableau de dictionnaire à partir d'un fichier CSV
    :: param fichierCSV(str) :: le nom du fichier CSV correct à traiter
    :: return (list) :: le tableau de dictionnaires
    '''

    c = []
    f = open(fichierCSV, 'r', encoding='utf-8')
    entete = f.readline()
    att = decomposer(entete, ',')
    for ligne in f:
        val = decomposer(ligne, ',')
        enr = creer_enregistrement(val, att)
        c.append(enr)
    f.close()
    return c

def decomposer(ligne) :
      ligne.replace('\n', '')
      return ligne.split(',')
```