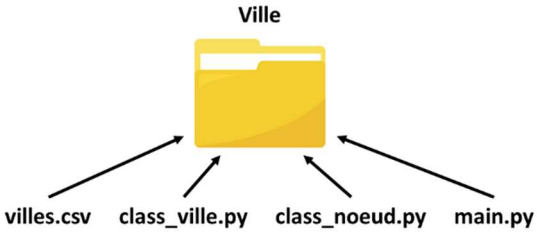
Classement de villes

Le but est d'utiliser un ABR pour classer des villes selon différents critères.

1. Structure des fichiers

Tous les fichiers créés sont à mettre dans un répertoire Ville. Il devra contenir 4 fichiers.



Dans un premier temps on récupère le fichier **villes.csv**.
Qui contient la liste des 200 plus grandes villes de France dans un ordre aléatoire.
Chaque ligne de ce fichier contient les 5 informations suivantes :

- Nom de la ville.
- Numéro du département.
- Nombre d'habitants.
- Superficie (en km²)
- Rang au niveau national (critère : nombre d'habitants)

Voici à quoi ressemblent les deux premières lignes de ce fichier :

```
Saint-Priest,69,40944,29.7,155
Versailles,78,85761,26.2,46
```

2. Création de la classe Ville

✍ Implémenter la classe **Ville** ci-dessous et enregistrer votre script nommé **class_ville.py** dans le répertoire Ville.

```
class Ville:
    """classe définissant une ville de France

    Chaque ville est définie par 4 attributs :
    -nom
    -numéro de département
    -population (nbre d'habitants)
    -superficie (km^2)
    -rang au niveau national (place)
    """
    def __init__(self, liste):
        self.nom = liste[0]
        self.departement = int(liste[1])
        self.population = int(liste[2])
        self.superficie = float(liste[3])
        self.rang = int(liste[4])


# création d'un objet pour test
liste = ['Nice', '6', '343123', '71.9', '5']
ville = Ville(liste)
```

✍ Vérifier l'accès à tous les attributs de l'objet instancié **ville**.

✍ Écrire une méthode **get_rang(self)** qui retourne le rang de la ville.

✍ Écrire une méthode **get_superficie(self)** qui retourne la superficie de la ville.

✍ Écrire une méthode **affiche_ville(self)** qui affiche les cinq données de la ville.

✍ Écrire une méthode **affiche_nom(self)** qui affiche le nom de la ville.

3. Création de la classe Noeud

✍ Créer une classe **Noeud** sur le format des ABR sachant que la valeur d'un Nœud est une instance de la classe **Ville**. Le constructeur est le suivant :

```
def __init__(self, liste, fils_g=None, fils_d=None):
    self.ville = Ville(liste)
    self.fils_g = fils_g
    self.fils_d = fils_d
```

L'attribut **ville** est un objet de type **Ville**.

✍ Enregistrer votre script nommé **class_noeud.py** dans le répertoire Ville.

✍ Récupérer et adapter la méthode **insert(self, value)** en **insert(self, liste)** de manière à ranger en fonction du rang de la ville.

4. Création de l'arbre

✍ Ouvrir le fichier **ville.csv**, qui est à placer dans le répertoire Ville, avec un éditeur de texte (Notepad+…) afin de savoir quel caractère délimite les données.

✍ Dans le script principal, que l'on nommera **main.py** et placé dans le répertoire Ville, écrire le code ci-dessous permettant de créer une liste contenant les lignes du fichier **ville.csv** sous formes de listes (compléter le délimitateur).

```
import csv

liste_villes=[]

file=open("villes.csv",'r',encoding='utf-8')
lecteur=csv.reader(file,delimiter='…')
for ligne in lecteur:
    liste_villes.append(ligne)
```

✍ Pour obtenir un arbre équilibré (gauche, droite), il faut choisir la racine avec un rang de 100. Pourquoi ?

✍ En quoi est-ce important pour une recherche dans l'arbre ?

✍ Compléter le script principal en écrivant le code suivant afin de construire l'arbre.

```
liste = ["Maisons-Alfort", '94', '51091', '5.4', '100']
abr = Noeud(liste)
for el in liste_villes:
    ville = Ville(el)
    abr.insert(el)
```

5. Exploitation de l'arbre

✍ Adapter la méthode **`parcours_infixe(self)`** au contexte afin d'afficher les villes par ordre croissant de leur rang.

✍ Apporter les modifications afin de faire afficher les villes par ordre croissant de leur superficie.

✍ En utilisant cette nouvelle construction, écrire une méthode **`recherche_max(self)`** qui affiche la ville ayant la plus grande superficie.

✍ Écrire une méthode **`recherche_rang(self, rang)`** qui affiche la ville dont le rang est rang.