Python

Statistiques Graphiques

13 septembre 2023

Version 3

Fabrice BOISSIER <fabrice.boissier@epita.fr>

Copyright

Table des matières

1 Consignes Générales IV

2 Format de Rendu V

3 Aide Mémoire VI

4 Exercice 1 - Statistiques population & gendarmerie par départements 1

5 Exercice 2 - Diagrammes statistiques 7

1 Consignes Générales

Les informations suivantes sont très importantes :

Le non-respect d'une des consignes suivantes entraînera des sanctions pouvant aller jusqu'à la multiplication de la note finale par 0.

Ces consignes sont claires, non-ambiguës, et ont un objectif précis. En outre, elles ne sont pas négociables.

N'hésitez pas à demander si vous ne comprenez pas une des règles.

Consigne Générale 0 : Vous devez lire le sujet.
Consigne Générale 1 : Vous devez respecter les consignes.
Consigne Générale 2 : Vous devez rendre le travail dans les délais prévus.

Consigne Générale 3 : Le travail doit être rendu dans le format décrit à la section Format de Rendu.

Consigne Générale 4 : Le travail rendu ne doit pas contenir de fichiers binaires, temporaires, ou d'erreurs (`*~`, `*.o`, `*.a`, `*.so`, `*#*`, `*core`, `*.log`, `*.exe`, binaires, ...).

Consigne Générale 5 : Dans l'ensemble de ce document, la casse (caractères majuscules et minuscules) est très importante. Vous devez strictement respecter les majuscules et minuscules imposées dans les messages et noms de fichiers du sujet.

Consigne Générale 6 : Dans l'ensemble de ce document, **`login`** correspond à votre login.

Consigne Générale 7 : Dans l'ensemble de ce document, **`nom1-nom2`** correspond à la combinaison des deux noms de votre binôme (par exemple pour Fabrice BOISSIER et Mark ANGOUSTURES, cela donnera **`boissier-angoustures`**).

Consigne Générale 8 : Dans l'ensemble de ce document, le caractère ␣ correspond à une espace (s'il vous est demandé d'afficher ␣␣␣, vous devez afficher trois espaces consécutives).

Consigne Générale 9 : Tout retard, même d'une seconde, entraîne la note non négociable de 0.

Consigne Générale 10 : La triche (échange de code, copie de code ou de texte, ...) entraîne **au mieux** la note non négociable de 0.

Consigne Générale 11 : En cas de problème avec le projet, vous devez contacter le plus tôt possible les responsables du sujet aux adresses mail indiquées.

Conseil : N'attendez pas la dernière minute pour commencer à travailler sur le sujet.

2 Format de Rendu

Responsable(s) du projet :	**Fabrice BOISSIER** <fabrice.boissier@epita.fr> **Lisa MONPIERRE** <lisa.monpierre@epita.fr>
Balise(s) du projet :	**[PYTHON][TP2]**
Nombre d'étudiant(s) par rendu :	1
Procédure de rendu :	Devoir/Assignment sur Teams
Nom du répertoire :	`login-Python-Graphs`
Nom de l'archive :	`login-Python-Graphs.tar.bz2`
Date maximale de rendu :	29/10/2022 23h42
Durée du projet :	1,5 mois
Architecture/OS :	Linux - Ubuntu (x86_64)
Langage(s) :	Python
Compilateur/Interpréteur :	**/usr/bin/python3**
Options du compilateur/interpréteur :	

Les fichiers suivants sont requis :

`AUTHORS`	contient le(s) nom(s) et prénom(s) de(s) auteur(s).
`README`	contient la description du projet et des exercices, ainsi que la façon d'utiliser le projet.

Votre code sera testé automatiquement, vous devez donc scrupuleusement respecter les spécifications pour pouvoir obtenir des points en validant les exercices. Votre code sera testé en appelant chaque script avec l'interpréteur python (et éventuellement un ou des arguments) :

```
python3 script.py arg1 arg2 [...]
```

L'arborescence attendue pour le projet est la suivante :

```
login-Python-Graphs/
login-Python-Graphs/AUTHORS
login-Python-Graphs/README
login-Python-Graphs/src/
login-Python-Graphs/src/PopStats.py
login-Python-Graphs/src/GraphStats.py
```

3 Aide Mémoire

Le travail doit être rendu au format ***.tar.bz2***, c'est-à-dire une archive **bz2** compressée avec un outil adapté (voir **`man 1 tar`** et **`man 1 bz2`**).
Tout autre format d'archive (zip, rar, 7zip, gz, gzip, ...) ne sera pas pris en compte, et votre travail ne sera pas corrigé (entraînant la note de 0).

Pour générer une archive *tar* en y mettant les dossiers *folder1* et *folder2*, vous devez taper :

```
tar cvf MyTarball.tar folder1 folder2
```

Pour générer une archive *tar* et la compresser avec GZip, vous devez taper :

```
tar cvzf MyTarball.tar.gz folder1 folder2
```

Pour générer une archive *tar* et la compresser avec BZip2, vous devez taper :

```
tar cvjf MyTarball.tar.bz2 folder1 folder2
```

Pour lister le contenu d'une archive *tar*, vous devez taper :

```
tar tf MyTarball.tar.bz2
```

Pour extraire le contenu d'une archive *tar*, vous devez taper :

```
tar xvf MyTarball.tar.bz2
```

Dans ce sujet précis, vous ferez du code en Python, qui affichera les résultats dans le terminal (donc des flux de sortie qui pourront être redirigés vers un fichier texte).

4 Exercice 1 - Statistiques population & gendarmerie par départements

Nom du(es) fichier(s) :	**PopStats.py**
Répertoire :	**login-Python-Graphs/src/**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Modules recommandés :	**sys**, **requests**, **numpy**

Objectif : Le but de l'exercice est d'écrire un script python calculant des statistiques sur la population française par département et le nombre de pré-plaintes déposées sur la plateforme en ligne de la gendarmerie, le tout en utilisant des API publiques.

Vous pouvez utiliser les modules Python de statistiques pour effectuer les calculs. De plus, il est recommandé d'utiliser le module **requests** pour effectuer les appels aux APIs en ligne, mais vous pouvez tout à fait utiliser **urllib** et **json**, voire d'autres modules externes.

La documentation pour les différents modules externes est accessibles via ces liens :

- Requests : https://requests.readthedocs.io
- Numpy : https://numpy.org/doc/
- API Gendarmerie : https://www.gendarmerie.interieur.gouv.fr/barometre-numerique/api/docs/
- API OpenDataSoft : https://public.opendatasoft.com/api/explore/v2.1/console
  - Catalogues des ensembles de données : https://public.opendatasoft.com/explore/?sort=modified
  - Populations légales Départements - Millésimé - France : https://public.opendatasoft.com/explore/dataset/demographyref-france-pop-legale-departement-millesime

Vous devez implémenter les fonctions décrites plus bas, si nécessaire en déclarant vos propres fonctions. Ce code sera testé en étant lancé par un script externe : vous pouvez adapter les types et valeurs donnés en paramètre, mais veillez à respecter les prototypes. Néanmoins, les messages texte affichés doivent être strictement les mêmes (à l'espace près) que ceux indiqués dans le sujet.

Vous devez implémenter les fonctions suivantes :

```
GetPopulationDepartements()
GetPrePlaintesDepartement(DepartementISOCode)

GetAndFilterPopulationDepartementsPerYear(year)
GetAndCleanPrePlaintesDepartement(DepartementCode)

SortListColumn(L, column)
SortListColumnReverse(L, column)

PrintThreeFirstsElt(L)
PrintStats(year)
main()
```

Liste des fonctions à implémenter

`GetPopulationDepartements()`

Cette fonction interroge l'API d'OpenDataSoft pour demander des informations dans le catalogue « Populations légales Départements - Millésimé - France » et renvoyer les résultats sous forme de dictionnaire.
L'API est accessible à l'adresse suivante, et ne nécessite aucun paramètre.
URL de base : https://public.opendatasoft.com/api/explore/v2.1/catalog/datasets/
Complément d'URL : /demographyref-france-pop-legale-departement-millesime/records/

De plus, il faut noter que la réponse renvoyée par l'API doit être désérialisée (*unmarshalled* en anglais) depuis JSON vers un dictionnaire avant d'être retournée.

`GetPrePlaintesDepartement(DepartementISOCode)`

Cette fonction interroge l'API de la gendarmerie pour demander la quantité de préplaintes reçues. L'API est accessible à l'adresse suivante, et nécessite un paramètre : l'indicatif du département au format ISO 3166-2. De plus, l'API de la gendarmerie ne transmet pas ce paramètre dans l'en-tête de la requête, mais directement dans l'URL avec le *path*. Il faut donc ajouter le département voulu en fin d'URL.
URL de base : https://www.gendarmerie.interieur.gouv.fr/barometre-numerique/
Complément d'URL : /api/pre-plainte-en-ligne/map-detail/

Une autre information importante : les API ne disposent pas toujours de toutes les informations. Il est possible que certains départements soient mis à 0.

De plus, il faut noter que la réponse renvoyée par l'API doit être désérialisée (*unmarshalled* en anglais) depuis JSON vers un dictionnaire avant d'être retournée.

Très succinctement, la norme **ISO 3166-2** implique d'utiliser le numéro du département en le faisant précéder de « **`FR-`** ». Un seul cas est exceptionnel : l'indicatif de Paris ne

doit pas simplement utiliser **75** précédé de **FR-**, mais doit ajouter un **C** à la fin (donc « **FR-75C** »).

`GetAndFilterPopulationDepartementsPerYear(year)`

Cette fonction appelle l'API publique pour obtenir la population par département lors de l'année donnée en paramètre. La fonction doit utiliser l'année contenue dans le champs **`start_year`**.
Deux stratégies sont possibles pour écrire cette fonction :

- Soit vous appelez directement **`GetPopulationDepartements()`** afin de filtrer ses résultats (le plus simple)
- Soit vous réécrivez l'appel à l'API en ajoutant des paramètres dans l'en-tête pour demander à recevoir exclusivement les résultats qui nous intéressent (plus difficile, mais beaucoup plus optimal en bande passante)

`GetAndCleanPrePlaintesDepartement(DepartementCode)`

Cette fonction appelle l'API publique pour faire la requête auprès de la gendarmerie avec le code du département, puis filtre les résultats pour ne renvoyer que le nombre de pré-plaintes sous forme d'entier.

`SortListColumn(L, column)`

Cette fonction trie en ordre croissant la liste donnée en paramètre selon les valeurs contenues dans une colonne particulière. Cette fonction doit trier dans l'ordre croissant : la valeur de la case 0 doit être plus petite ou égale à cette de la case 1, et ainsi de suite.

`SortListColumnReverse(L, column)`

Cette fonction trie en ordre décroissant la liste donnée en paramètre selon les valeurs contenues dans une colonne particulière. Cette fonction doit trier dans l'ordre décroissant : la valeur de la case 0 doit être plus grande ou égale à cette de la case 1, et ainsi de suite.

`PrintThreeFirstsElt(L)`

Cette fonction écrit sur la sortie standard les 3 premiers éléments de la liste donnée en paramètre. Le format attendu est simple : un seul élément doit être affichés par ligne.

valeur

Ce qui donnerait pour la liste $[('a','b'),('c','d'),('e','f'),('g','h')]$:

```
$ python3 example.py
('a', 'b')
('c', 'd')
('e', 'f')
$
```

Si la distribution contient moins de 3 éléments, vous n'afficherez que ceux disponibles. Si la distribution est vide, vous n'afficherez rien.

`PrintStats(year)`

Cette fonction lance les appels aux APIs puis calcule plusieurs statistiques simples sur une année. Vous pouvez utiliser **`numpy`** pour calculer ces statistiques (et cela est vivement recommandé).
Cette fonction devra calculer plusieurs statistiques utiles concernant les départements durant l'année donnée en paramètre, et les croiser avec le nombre de pré-plaintes en ligne. L'affichage attendu est le suivant dans cet ordre précis : *(cf la page suivante également)*

1. Vous devrez tout d'abord afficher la liste des départements avec leur code, leur nom, la population totale du département, et le nombre de pré-plaintes associées. Chaque département s'affichera sur une seule ligne
2. Puis, vous devrez sauter une ligne
3. Vous afficherez ensuite une ligne avec marquée :
   « **`- 3 Tops Most Populations:`** »
4. Vous afficherez la liste des 3 département les plus peuplés dans l'ordre décroissant (dans le même format que la liste complète des départements)
5. Vous afficherez ensuite une ligne avec marquée :
   « **`- 3 Tops Least Populations:`** »
6. Vous afficherez la liste des 3 département les moins peuplés dans l'ordre croissant (dans le même format que la liste complète des départements)
7. Puis, vous devrez sauter une ligne
8. Vous afficherez ensuite une ligne avec marquée :
   « **`- 3 Tops Most PrePlaintes:`** »
9. Vous afficherez la liste des 3 département ayant produit le plus de préplaintes dans l'ordre décroissant (dans le même format que la liste complète des départements)
10. Vous afficherez ensuite une ligne avec marquée :
    « **`- 3 Tops Least PrePlaintes:`** »
11. Vous afficherez la liste des 3 département ayant produit le moins de préplaintes dans l'ordre croissant (dans le même format que la liste complète des départements)
12. Puis, vous devrez sauter une ligne
13. Puis vous afficherez dans cet ordre chaque statistique suivante concernant la population en la précédant d'un message : la moyenne, la médiane, la variance, l'écart type.
    La moyenne sera précédée du message : « **`- Population average:`** »
    La médiane sera précédée du message : « **`- Population median:`** »
    La variance sera précédée du message : « **`- Population variance:`** »
    L'écart type sera précédé du message : « **`- Population standard deviation:`** »

14. Puis, vous devrez sauter une ligne
15. Puis vous afficherez dans cet ordre chaque statistique suivante concernant les préplaintes en la précédant d'un message : la moyenne, la médiane, la variance, l'écart type.
    La moyenne sera précédée du message : « **`- Preplaintes average:`** »
    La médiane sera précédée du message : « **`- Preplaintes median:`** »
    La variance sera précédée du message : « **`- Preplaintes variance:`** »
    L'écart type sera précédé du message : « **`- Preplaintes standard deviation:`** »

Le format attendu devrait lister les éléments dans ce format pour l'année 2019 :

```
$ python3 example.py
('01', 'Ain', 655171, 9077)
('02', 'Aisne', 549587, 4962)
('2A', 'Corse-du-Sud', 156958, 1193)
('2B', 'Haute-Corse', 179037, 1443)
[...]
('974', 'La Réunion', 862814, 3713)

- 3 Tops Most Populations:
('59', 'Nord', 2639070, 11063)
('75', 'Paris', 2210875, 8)
('13', 'Bouches-du-Rhône', 2047433, 15898)
- 3 Tops Least Populations:
('48', 'Lozère', 80141, 428)
('23', 'Creuse', 123500, 753)
('05', 'Hautes-Alpes', 146148, 1253)

- 3 Tops Most PrePlaintes:
('33', 'Gironde', 1595903, 29610)
('44', 'Loire-Atlantique', 1415805, 23312)
('31', 'Haute-Garonne', 1373626, 21134)
- 3 Tops Least PrePlaintes:
('69', 'Rhône', 1864962, 0)
('92', 'Hauts-de-Seine', 1622143, 0)
('93', 'Seine-Saint-Denis', 1616311, 0)

- Population average:
677518.38
- Population median:
548026.5
- Population variance:
259634302703.77563
- Population standard deviation:
509543.2294749638

- Preplaintes average:
5593.23
- Preplaintes median:
3748.0
- Preplaintes variance:
29523226.6771
- Preplaintes standard deviation:
5433.528013832265
```

`main()`

Cette fonction appelle simplement **`PrintStats()`** en lui donnant en paramètre le premier argument donné à la lgine de commande. Si le premier argument donné au script n'est pas un entier, vous quitterez le programme en renvoyant **`-1`** et en affichant ce message :

`First argument must be an integer`

La fonction **`main()`** doit être appelée par votre script pour qu'il affiche toutes les statistiques demandées lorsqu'il est lancé.

5 Exercice 2 - Diagrammes statistiques

Nom du(es) fichier(s) :	**GraphStats.py**
Répertoire :	**login-Python-Graphs/src/**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Modules recommandés :	**sys**, **datetime**, **csv**, **pandas**, **matplotlib.pyplot**

Objectif : Le but de l'exercice est d'écrire une classe faisant des appels aux APIs, lisant un CSV, et produisant des diagrammes statistiques.

Vous devrez réutiliser une partie du code écrit dans l'exercice 1, mais de façon plus précise.

La documentation pour les différents modules externes est accessibles via ces liens :
- DateTime : https://docs.python.org/fr/3/library/datetime.html
- CSV : https://docs.python.org/fr/3/library/csv.html
- Pandas : https://pandas.pydata.org/docs/user_guide/index.html
- Matplotlib : https://matplotlib.org/stable/tutorials/introductory/usage.html

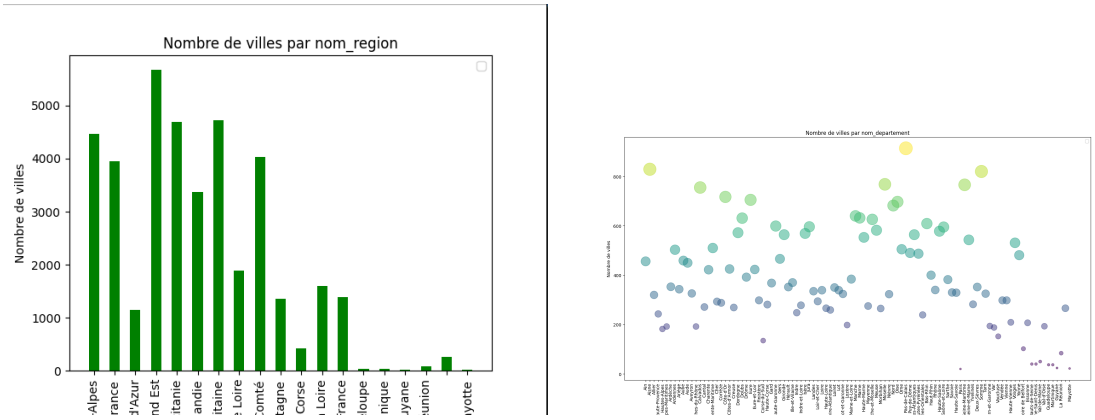

FIGURE 1 – Exemples de graphiques statistiques avec Matplotlib

Les différents appels aux APIs publiques que vous avez réalisés vous ont permis de récupérer les données brutes, et de les afficher de façon très rudimentaires dans le terminal. Néanmoins, un affichage graphique améliorera non seulement l'esthétique, mais également la lisibilité et la capacité à comprendre les informations contenues dans les données.

Afin d'étendre un peu plus les analyses, vous devrez récupérer un fichier CSV en ligne et le lire pour en tirer des informations que vous croiserez avec les données sur la population. Le fichier CSV contient une liste de villes et leur département ainsi que leur région d'appartenance. Il peut être directement fourni en paramètre aux modules **csv** et **Pandas** sans être téléchargé préalablement. Celui-ci est accessible ici : https://www.data.gouv.fr/fr/datasets/r/dbe8a621-a9c4-4bc3-9cae-be1699c5ff25

Vous devez implémenter les fonctions décrites plus bas, si nécessaire en déclarant vos propres fonctions. Ce code sera testé en étant lui-même appelé par un autre script, vous devez donc absolument respecter les prototypes décrits ici.

Fonction `main()`

Cette fonction sera celle appelée lorsque le script est lancé. Elle ne prend pas de paramètre, mais elle lit les arguments donnés en ligne de commande puis instancie la classe GraphStat et appelle la méthode **`ProcessAndDraw()`**. Si tout se passe bien, le script doit renvoyer **`0`**. Deux paramètres, dont le deuxième est optionnel, doivent être testés :

1. Le premier argument est l'année de recensement qui sera demandée à l'API des populations millésimées par département.
2. Le deuxième argument est le format de graphique attendu.
   — L'argument **`bar`** indique que l'utilisateur souhaite un diagramme en barres (le format « *bâtonnets* », ou *bar chart* en anglais).
   — L'argument **`pie`** indique que l'utilisateur souhaite un diagramme circulaire (le format « *camembert* », ou *pie chart* en anglais).
   — Si aucun de ces choix n'est sélectionné, alors l'option **`bar`** sera sélectionnée par défaut.

Si le premier argument (année du recensement) est donné mais n'est pas un entier positif entre **`1800`** et l'année courante, le script doit se terminer en renvoyant **`-1`** et en écrivant :

```
$ python GraphStats.py 42 bar
Year must be greater than "1800" and at most current year
   (2023)
./GraphStats.py year [default:bar | pie]
$ echo $?
255
```

Si un deuxième argument (format du graphique) est donné mais n'est ni **`bar`** ni **`pie`**, le script doit se terminer en renvoyant **`-2`** et en écrivant :

```
$ python GraphStats.py 2019 wololooo
Graphic format must be "bar" or "pie" (or can be omitted)
./GraphStats.py year [default:bar | pie]
$ echo $?
254
```

Si les deux arguments sont problématiques, celui de l'année prime sur celui du format de graphique :

```
$ python GraphStats.py 4 chan
Year must be greater than "1800" and at most current year
   (2023)
./GraphStats.py year [default:bar | pie]
$ echo $?
255
```

Enfin, si le premier argument est un **-h**, **-help**, ou qu'il n'y a aucun argument, le script doit afficher ce message et renvoyer **0** sans rien faire d'autre :

```
$ python GraphStats.py -h
./GraphStats.py year [default:bar | pie]
$ echo $?
0
```

```
$ python GraphStats.py --help
./GraphStats.py year [default:bar | pie]
$ echo $?
0
```

```
$ python GraphStats.py
./GraphStats.py year [default:bar | pie]
$ echo $?
0
```

Classe **GraphStat**

Cette classe permet de faire les appels aux APIs, calculer les ratios, et dessiner les graphiques associés. Elle contient au moins ces méthodes et attributs :

```
__init__(self, year, graph_type)
__GetPopulation(self, year)
__GetVilles(self)
__BuildPopulationDict(self, Dict_API)
__BuildCityDict(self, data)
__BuildDepartementDict(self, data)
__CalculateStats(self, Dict_Pop, Dict_Cities)
__DrawBar(self, Dict_Ratio, Dict_ID_Dept, asc=False)
__DrawPie(self, Dict_Ratio, Dict_ID_Dept, asc=False)
ProcessAndDraw(self)

year
graph_type
```

__init__(self, year, graph_type)

Cette méthode récupère les paramètres fournis par la fonction **main()** et les assigne aux attributs **year** et **graph_type** contenus dans la classe **GraphStat**. Attention, cette méthode n'exécute rien d'autre.

__GetPopulation(self, year)

Cette méthode exécute l'appel à l'API pour obtenir la *population totale* par département pour l'année donnée en paramètre, puis elle renvoie la réponse donnée par l'API. L'année correspond à l'attribut/champs **strat_year** de l'API. L'URL de l'API est la suivante : https://public.opendatasoft.com/explore/dataset/demographyref-france-pop-legale-departement-millesime
Si la réponse de l'API est vide, cette méthode termine le script en renvoyant **-3** et en écrivant le message suivant :

```
$ python GraphStats.py 2000 bar
Sadly, no data was available for year "2000".
Try again with another date.
$ echo $?
253
```

__GetVilles(self)

Cette méthode récupère le fichier CSV depuis l'API publique, puis elle renvoie le *dataframe* qui en est extrait (dans le cas d'utilisation du module *Pandas*) ou la matrice représentant le CSV (dans le cas d'utilisation du module *csv*).
L'URL pour accéder au dataset est la suivante : https://www.data.gouv.fr/fr/datasets/r/dbe8a621-a9c4-4bc3-9cae-be1699c5ff25

__BuildPopulationDict(self, Dict_API)

Cette méthode construit un dictionnaire dont les clés sont les codes des départements, et les valeurs sont la population totale de chaque département. Le paramètre d'entrée est le dictionnaire de données renvoyée par l'API traitant de la démographie.

__BuildCityDict(self, data)

Cette méthode construit un dictionnaire dont les clés sont les codes des départements, et les valeurs sont la quantité de villes pour chaque département. Le paramètre d'entrée est le tableau contenant la liste des villes par département (soit un *dataframe* si vous avez utilisé le module *Pandas*, soit une matrice si vous avez utilisé le module *csv*).

__BuildDepartementDict(self, data)

Cette méthode construit un dictionnaire dont les clés sont les codes des départements, et les valeurs sont les noms complets de chaque département. Le paramètre d'entrée peut aussi bien être le dictionnaire de données renvoyée par l'API traitant de la démographie, ou le tableau contenant la liste des villes par département.

__CalculateStats(self, Dict_Pop, Dict_Cities)

Cette méthode calcule le ratio entre la population de chaque département et le nombre de villes contenus dans chacun d'entre eux. La fonction prend en premier paramètre un dictionnaire dont les clés sont les codes des départements et les valeurs sont la population contenue dans chacun d'eux, et le deuxième paramètre est un dictionnaire dont les clés sont les codes des départements et les valeurs sont la quantité de villes dans chacun des départements.
La méthode doit renvoyer un dictionnaire dont les clés sont les codes des départements, et les valeurs sont les ratios.

__DrawBar(self, Dict_Ratio, Dict_ID_Dept, [asc=False])

Cette méthode dessine le diagramme en barres appliqué aux données en paramètres.
Le premier dictionnaire en paramètre contient en clés les codes des départements et en valeurs les ratios des population/villes. Le deuxième dictionnaire en paramètre contient en clés les codes des départements et en valeurs les noms complets des départements. Le troisième paramètre est optionnel et sert à indiquer si l'on souhaite afficher les 10 ratios les plus élevés (en ordre décroissant), ou les 10 ratios les plus faibles (en ordre croissant).

Pour ce graphique, la couleur des barres doit être **`green`** et leur largeur de **`0.5`**.
Le diagramme à afficher doit indiquer en hauteur le ratio, et à la base les noms complets des départements.

`__DrawPie(self, Dict_Ratio, Dict_ID_Dept, [asc=False])`

Cette méthode dessine le diagramme circulaire appliqué aux données en paramètres.
Le premier dictionnaire en paramètre contient en clés les codes des départements et en valeurs les ratios des population/villes. Le deuxième dictionnaire en paramètre contient en clés les codes des départements et en valeurs les noms complets des départements. Le troisième paramètre est optionnel et sert à indiquer si l'on souhaite afficher les 5 ratios les plus élevés, ou les 5 ratios les plus faibles.
Afin d'obtenir un diagramme comparant correctement les ratios de l'ensemble de la base de données, vous afficherez en gris dans un seul secteur l'ensemble des ratios non pris en compte, et les 5 sélectionnés (les plus élevés ou les plus faibles) en couleurs :

1. rouge pour le plus grand,
2. orange pour le suivant,
3. jaune pour le suivant,
5. vert pour le suivant,
6. et bleu pour le dernier.

Le diagramme à afficher doit indiquer les ratios et les noms complets des départements.

`ProcessAndDraw(self)`

Cette méthode est la seule méthode publique (au sens POO) : c'est celle-ci qui doit être appelée par le code externe pour déclencher la récupération des données auprès des APIs externes, faire tous les calculs de ratios et préparer les dictionnaires, puis choisir la bonne méthode pour dessiner le graphique de sortie à partir des attributs enregistrés dans la classe.