Python

Statistiques

05 septembre 2022

Version 4

Fabrice BOISSIER <fabrice.boissier@epita.fr>

Lisa MONPIERRE <lisa.monpierre@epita.fr>

Copyright

Table des matières

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][TP1]**
Nombre d'étudiant(s) par rendu :	1
Procédure de rendu :	Devoir/Assignment sur Teams
Nom du répertoire :	`login-Python-Stats`
Nom de l'archive :	`login-Python-Stats.tar.bz2`
Date maximale de rendu :	02/10/2022 23h42
Durée du projet :	1 mois
Architecture/OS :	Linux - Ubuntu (x86_64)
Langage(s) :	Python
Compilateur/Interpréteur :	**/usr/bin/python3.10**
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) :

```
python3.10 script.py arg1 arg2 [...]
```

L'arborescence attendue pour le projet est la suivante :

```
login-Python-Stats/
login-Python-Stats/AUTHORS
login-Python-Stats/README
login-Python-Stats/src/
login-Python-Stats/src/class/
login-Python-Stats/src/class/MyOwnStats.py
login-Python-Stats/src/module/
login-Python-Stats/src/module/MyOwnStats.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 - Module statistiques

Nom du(es) fichier(s) :	**`MyOwnStats.py`**
Répertoire :	**`login-Python-Stats/src/module/`**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Fonctions externes autorisées :	**`math.sqrt`**, **`random.random`**, **`random.randint`**

Objectif : Le but de l'exercice est d'écrire un module python contenant plusieurs fonctions permettant de calculer des statistiques sur une distribution.

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.

Vous ne devez pas implémenter de classe dans cet exercice, mais bel et bien un module. Vous ne devez pas non plus importer la classe de l'exercice 2.

Pour construire ce module, vous avez le droit d'importer exclusivement 3 fonctions de modules externes :

- **`math.sqrt`** du module `math`
- **`random.random`** du module `random`
- **`random.randint`** du module `random`

Cependant, vous pouvez parfaitement utiliser les fonctions, types, et classes internes à Python (tels que **`print()`**, **`int()`**, **`str()`**, **`range()`**, **`list()`**, et autres).

Titre

Notez bien qu'aucune fonction de ce module ne doit modifier la distribution donnée en paramètre ! Pas même les fonctions de tri, d'ajout, de suppression, ou de fusion, ni les autres fonctions ayant besoin que la distribution soit triée.

Vous devez implémenter les fonctions suivantes :

```
GenerateDistribution(DistrNbValues)
SortDistribution(Distribution, [asc=True])
PrintDistribution(Distribution)

AddValueDistribution(Distribution, Value)
AddValuePositionDistribution(Distribution, Value, Position)
DeleteValueDistribution(Distribution, Value)
DeletePositionDistribution(Distribution, Position)
MergeDistribution(Distribution, NewDistribution)

Min(Distribution)
Max(Distribution)
Cardinality(Distribution)
Range(Distribution)

Mean(Distribution)
Median(Distribution)
GetQuartile(Distribution, Quartile)

InterQuartileRange(Distribution)
Variance(Distribution)
StandardDeviation(Distribution)
RelativeStandardDeviation(Distribution)
GiniCoefficient(Distribution)
```

Liste des fonctions pour le module de statistiques

`GenerateDistribution(DistrNbValues)`

Cette fonction crée et renvoie une liste d'entiers aléatoires représentants une distribution, ainsi que les valeurs minimale et maximale. Les valeurs doivent être renvoyées exactement dans cet ordre : la valeur minimum, la valeur maximum, et la liste contenant la distribution. La distribution créée peut contenir plusieurs fois la même valeur. Le paramètre *DistrNbValues* est un entier indiquant le nombre d'entiers que la distribution doit contenir. Si *DistrNbValues* est nul, alors vous renverrez une liste vide, avec le minimum et le maximum à 0. Si *DistrNbValues* est négatif, alors vous renverrez une distribution contenant 10 entiers.

`SortDistribution(Distribution, [asc=True])`

Cette fonction renvoie la version triée de la distribution donnée en paramètre. Le paramètre *Distribution* est la distribution à trier (il doit s'agir d'une liste d'entiers). Le paramètre *asc* est un booléen indiquant si la liste doit être triée par ordre ascendant ou descendant. Par défaut, le tri doit être ascendant (à la position 0 doit se trouver l'entier

le plus petit de la distribution, et à la dernière position on doit trouver l'entier le plus grand).

Attention : cette fonction ne doit pas modifier le paramètre donné, mais bien renvoyer une copie triée !

`PrintDistribution(Distribution)`

Cette fonction écrit le contenu de la distribution sur la sortie standard. Le format attendu est simple : un seul élément et sa position doivent être affichés par ligne.

`[position] : valeur`

Ce qui donnerait pour la distribution $[10, 5, 42]$:

```
$ python3.10 example.py
[0] : 10
[1] : 5
[2] : 42
$
```

Si la distribution est vide, vous n'afficherez rien.

`AddValueDistribution(Distribution, Value)`

Cette fonction renvoie une distribution contenant la valeur ajoutée à la distribution donnée en paramètre. Le paramètre *Value* est la valeur à ajouter. La valeur supplémentaire s'ajoute en fin de liste. On peut ajouter une valeur déjà existante. Si la valeur n'est pas un entier, vous déclencherez une exception **`ValueNotInteger`** sans afficher de message.

Ainsi, pour la distribution $[10, 5, 42]$, lorsque l'on ajoute la valeur 6, on doit obtenir la distribution suivante : $[10, 5, 42, 6]$.

`AddValuePositionDistribution(Distribution, Value, Position)`

Cette fonction renvoie une distribution contenant la valeur ajoutée à la position précise dans la distribution donnée en paramètre. Le paramètre *Value* est la valeur à ajouter. On peut ajouter une valeur déjà existante. Si la valeur n'est pas un entier, vous déclencherez une exception **`ValueNotInteger`** sans afficher de message. Le paramètre *Position* indique la position dans la liste où ajouter l'élément (l'ancien élément présent à cette position sera décalé à la position suivante : si on ajoute en position 0, l'élément qui y était sera poussé en position 1). Si la position est négative, on ajoute l'élément en position 0. Si la position est supérieure à la dernière position de la liste, on ajoute l'élément en fin de liste/après la dernière position.

`DeleteValueDistribution(Distribution, Value)`

Cette fonction renvoie une copie de la distribution donnée en paramètre dont on a supprimé une occurrence de la valeur indiquée (la première occurrence trouvée depuis la

position 0). Le paramètre *Value* est la valeur à supprimer. Si la valeur n'est pas un entier, vous déclencherez une exception **`ValueNotInteger`** sans afficher de message. Si la valeur n'est pas trouvée dans la liste, vous déclencherez une exception **`ValueNotFoundInDistribution`** sans afficher de message. Si la distribution ne contient plus aucune valeur au moment de l'appel à cette méthode, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message. L'exception **`ValueNotInteger`** prime sur l'exception **`DistributionEmpty`** qui elle-même prime sur l'exception **`ValueNotFoundInDistribution`**.

`DeletePositionDistribution(Distribution, Position)`

Cette fonction renvoie une copie de la distribution donnée en paramètre dont on a supprimé la valeur située à la position donnée en paramètre. Le paramètre *Position* est la position de la valeur à supprimer. Si la position est négative, vous supprimerez l'élément en position 0. Si la position est supérieure à la dernière position de la liste, vous supprimerez le dernier élément de la liste. Si la distribution ne contient plus aucune valeur au moment de l'appel à cette méthode, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`MergeDistribution(Distribution, NewDistribution)`

Cette fonction renvoie une liste issue de la fusion des deux distributions données en paramètre. Le paramètre *NewDistribution* est la distribution à ajouter après la liste en premier paramètre.

`Min(Distribution)`

Cette fonction renvoie la valeur minimale de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`Max(Distribution)`

Cette fonction renvoie la valeur maximale de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`Cardinality(Distribution)`

Cette fonction calcule et renvoie le *cardinal* de la distribution.

`Range(Distribution)`

Cette fonction calcule et renvoie l'*étendue* de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`Mean(Distribution)`

Cette fonction calcule et renvoie la *moyenne* de la distribution (il doit s'agir d'un flottant). Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`Median(Distribution)`

Cette fonction calcule et renvoie la *médiane* de la distribution. Si la distribution a un nombre impair d'éléments, vous renverrez l'élément à la position par défaut/obtenue par la partie entière inférieure (par exemple, pour une distribution composée de 3 éléments, vous renverrez l'élément en position 1 et non pas en 2). Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

Rappel : cette fonction ne doit pas modifier la distribution donnée en paramètre !

`GetQuartile(Distribution, Quartile)`

Cette fonction calcule et renvoie le *quartile* demandé de la distribution. Le paramètre **`Quartile`** correspond à un entier entre 0 et 4 (inclus). Les quartiles 1, 2, et 3 correspondent aux quartiles séparant les différentes parties de la distribution. Le quartile 0 correspond à la plus petite valeur de la distribution, et le quartile 4 correspond à la plus grande valeur de la distribution. Si le paramètre **`Quartile`** est inférieur à 0 ou supérieur à 4, vous déclencherez une exception **`IncorrectQuartileParameter`** sans afficher de message. En cas de position non entière, vous prendrez la valeur par défaut en retirant juste la virgule. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message. L'exception **`IncorrectQuartileParameter`** prime sur l'exception **`DistributionEmpty`**.

Rappel : cette fonction ne doit pas modifier la distribution donnée en paramètre !

`InterQuartileRange(Distribution)`

Cette fonction calcule et renvoie l'*écart interquartile* de la distribution. C'est-à-dire la différence entre le quartile 3 et le quartile 1. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`Variance(Distribution)`

Cette fonction calcule et renvoie la *variance* de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`StandardDeviation(Distribution)`

Cette fonction calcule et renvoie l'*écart type* de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`RelativeStandardDeviation(Distribution)`

Cette fonction calcule et renvoie le *coefficient de variation* de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`GiniCoefficient(Distribution)`

Cette fonction calcule et renvoie le *coefficient de Gini* de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

5 Exercice 2 - Classe statistiques

Nom du(es) fichier(s) :	**`MyOwnStats.py`**
Répertoire :	**`login-Python-Stats/src/class/`**
Droits sur le répertoire :	750
Droits sur le(s) fichier(s) :	640
Fonctions externes autorisées :	**`math.sqrt`**, **`random.random`**, **`random.randint`**

Objectif : Le but de l'exercice est d'écrire une classe python contenant une distribution et plusieurs méthodes permettant de calculer des statistiques dessus (comme le module de l'exercice 1, mais cette fois dans le paradigme objet).

Vous devez implémenter une classe **`MyDistribution`** contenant les méthodes décrites plus bas, si nécessaire en déclarant vos propres méthodes et attributs. Ce code sera testé en étant lui-même appelé par un autre script, vous devez donc absolument respecter les prototypes décrits ici.

Vous ne devez pas implémenter les fonctions d'un module dans cet exercice, mais bel et bien une classe et ses méthodes. Vous ne devez pas non plus importer le module de l'exercice 1, pour la bonne raison que vous pourrez cette fois optimiser votre code (par exemple en déclenchant automatiquement le re-calcul des indicateurs lors de l'ajout de nouvelles valeurs, ou en stockant dans un attribut dédié la somme des valeurs de la distribution courante, voire d'autres sommes ou calculs communs à plusieurs indicateurs). Vous pouvez donc ajouter des méthodes ou attributs supplémentaires utiles. Cependant, vous ne devez toujours pas trier la distribution sans qu'un appel à la méthode de tri ait été réalisé par l'utilisateur de votre classe.

Pour construire cette classe, vous avez le droit d'importer exclusivement 3 fonctions de modules externes :

- **`math.sqrt`** du module `math`
- **`random.random`** du module `random`
- **`random.randint`** du module `random`

Cependant, vous pouvez parfaitement utiliser les fonctions, types, et classes internes à Python (tels que **`print()`**, **`int()`**, **`str()`**, **`range()`**, **`list()`**, et autres).

Titre

Notez bien qu'aucune méthode de ce module ne doit modifier la distribution donnée en paramètre, exceptée la méthode de tri ! Aucune méthode d'ajout, de suppression, ou de fusion, ni les autres méthodes ayant besoin que la distribution soit triée.

Vous devez implémenter les méthodes suivantes :

```
__init__(self, DistrNbValues)
GetDistribution(self)
SortDistribution(self, [asc=True])
PrintDistribution(self)

AddValueDistribution(self, Value)
AddValuePositionDistribution(self, Value, Position)
DeleteValueDistribution(self, Value)
DeletePositionDistribution(self, Position)
MergeDistribution(self, NewDistribution)

Min(self)
Max(self)
Cardinality(self)
Range(self)

Mean(self)
Median(self)
GetQuartile(self, Quartile)

InterQuartileRange(self)
Variance(self)
StandardDeviation(self)
RelativeStandardDeviation(self)
GiniCoefficient(self)
```

Liste des méthodes pour la classe de statistiques

`__init__(self, DistrNbValues)`

Le constructeur de la classe crée une liste d'entiers aléatoires représentants une distribution, et la conserve dans un attribut. La distribution créée peut contenir plusieurs fois la même valeur. Le paramètre *DistrNbValues* est un entier indiquant le nombre d'entiers que la distribution doit contenir. Si *DistrNbValues* est nul, alors vous allouerez une liste vide. Si *DistrNbValues* est négatif, alors vous créerez une distribution contenant 10 entiers.

`GetDistribution(self)`

Cette méthode renvoie une liste contenant les entiers de la distribution.

`SortDistribution(self, [asc=True])`

Cette méthode trie la distribution de l'objet. Le paramètre *asc* est un booléen indiquant si la liste doit être triée par ordre ascendant ou descendant. Par défaut, le tri doit être ascendant (à la position 0 doit se trouver l'entier le plus petit de la distribution, et à la dernière position on doit trouver l'entier le plus grand).

`PrintDistribution(self)`

Cette méthode écrit le contenu de la distribution sur la sortie standard. Le format attendu est simple : un seul élément et sa position doivent être affichés par ligne.

`[position] : valeur`

Ce qui donnerait pour la distribution $[10, 5, 42]$:

```
$ python3.10 example.py
[0] : 10
[1] : 5
[2] : 42
$
```

Si la distribution est vide, vous n'afficherez rien.

`AddValueDistribution(self, Value)`

Cette méthode ajoute la valeur donnée en paramètre à la distribution. Le paramètre *Value* est la valeur à ajouter. La valeur supplémentaire s'ajoute en fin de liste. On peut ajouter une valeur déjà existante. Si la valeur n'est pas un entier, vous déclencherez une exception **`ValueNotInteger`** sans afficher de message.

Ainsi, pour la distribution $[10, 5, 42]$, lorsque l'on ajoute la valeur 6, on doit obtenir la distribution suivante : $[10, 5, 42, 6]$.

`AddValuePositionDistribution(self, Value, Position)`

Cette méthode ajoute une valeur à la position précise dans la distribution. Le paramètre *Value* est la valeur à ajouter. On peut ajouter une valeur déjà existante. Si la valeur n'est pas un entier, vous déclencherez une exception **`ValueNotInteger`** sans afficher de message. Le paramètre *Position* indique la position dans la liste où ajouter l'élément (l'ancien élément présent à cette position sera décalé à la position suivante : si on ajoute en position 0, l'élément qui y était sera poussé en position 1). Si la position est négative, on ajoute l'élément en position 0. Si la position est supérieure à la dernière position de la liste, on ajoute l'élément en fin de liste/après la dernière position.

`DeleteValueDistribution(self, Value)`

Cette méthode supprime une occurrence de la valeur indiquée (la première occurrence trouvée depuis la position 0). Le paramètre *Value* est la valeur à supprimer. Si la valeur n'est pas un entier, vous déclencherez une exception **`ValueNotInteger`** sans afficher de message. Si la valeur n'est pas trouvée dans la liste, vous déclencherez une exception **`ValueNotFoundInDistribution`** sans afficher de message. Si la distribution ne contient plus aucune valeur au moment de l'appel à cette méthode, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message. L'exception **`ValueNotInteger`** prime sur l'exception **`DistributionEmpty`** qui elle-même prime sur l'exception **`ValueNotFoundInDistribution`**.

`DeletePositionDistribution(self, Position)`

Cette méthode supprime la valeur située à la position donnée en paramètre. Le paramètre *Position* est la position de la valeur à supprimer. Si la position est négative, vous supprimerez l'élément en position 0. Si la position est supérieure à la dernière position de la liste, vous supprimerez le dernier élément de la liste. Si la distribution ne contient plus aucune valeur au moment de l'appel à cette méthode, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`MergeDistribution(self, NewDistribution)`

Cette méthode fusionne la distribution donnée en paramètre à celle gérée par la classe. Le paramètre *NewDistribution* est la distribution à ajouter à la fin de la distribution déjà contenue dans l'objet.

`Min(self)`

Cette méthode renvoie la valeur minimale de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`Max(self)`

Cette méthode renvoie la valeur maximale de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`Cardinality(self)`

Cette méthode calcule et renvoie le *cardinal* de la distribution.

`Range(self)`

Cette méthode calcule et renvoie l'*étendue* de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`Mean(self)`

Cette méthode calcule et renvoie la *moyenne* de la distribution (il doit s'agir d'un flottant). Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`Median(self)`

Cette méthode calcule et renvoie la *médiane* de la distribution. Si la distribution a un nombre impair d'éléments, vous renverrez l'élément à la position par défaut/obtenue par la partie entière inférieure (par exemple, pour une distribution composée de 3 éléments, vous renverrez l'élément en position 1 et non pas en 2). Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

Rappel : cette fonction ne doit pas modifier la distribution contenue dans l'objet !

`GetQuartile(self, Quartile)`

Cette méthode calcule et renvoie le *quartile* demandé de la distribution. Le paramètre **`Quartile`** correspond à un entier entre 0 et 4 (inclus). Les quartiles 1, 2, et 3 correspondent aux quartiles séparant les différentes parties de la distribution. Le quartile 0 correspond à la plus petite valeur de la distribution, et le quartile 4 correspond à la plus grande valeur de la distribution. Si le paramètre **`Quartile`** est inférieur à 0 ou supérieur à 4, vous déclencherez une exception **`IncorrectQuartileParameter`** sans afficher de message. En cas de position non entière, vous prendrez la valeur par défaut en retirant juste la virgule. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message. L'exception **`IncorrectQuartileParameter`** prime sur l'exception **`DistributionEmpty`**.

Rappel : cette fonction ne doit pas modifier la distribution contenue dans l'objet !

`InterQuartileRange(self)`

Cette méthode calcule et renvoie l'*écart interquartile* de la distribution. C'est-à-dire la différence entre le quartile 3 et le quartile 1. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`Variance(self)`

Cette méthode calcule et renvoie la *variance* de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`StandardDeviation(self)`

Cette méthode calcule et renvoie l'*écart type* de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`RelativeStandardDeviation(self)`

Cette méthode calcule et renvoie le *coefficient de variation* de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

`GiniCoefficient(self)`

Cette méthode calcule et renvoie le *coefficient de Gini* de la distribution. Si la liste est vide, vous déclencherez une exception **`DistributionEmpty`** sans afficher de message.

Optionnel : si vous souhaitez augmenter la difficulté de l'exercice, n'hésitez pas à optimiser votre code de différentes manières. Vous pouvez typiquement recalculer certaines valeurs intermédiaires utiles lors de la mise à jour de la distribution (ajouter ou supprimer chaque valeur au fur et à mesure), et faire les calculs les plus lourds uniquement lorsqu'un développeur tiers demande explicitement une métrique. Vous pouvez également implémenter plusieurs fois cette classe et comparer dans quels scénarios d'usage chaque version sera la plus rapide : de nombreux ajouts et suppressions d'une valeur à chaque fois formant une grande liste avec quelques demandes de métriques dans l'ensemble du scénario, ou quelques fusions de grandes listes avec de nombreux accès à toutes les métriques possibles entre chaque fusion.