TP 03

Structures algorithmiques

Lien Capytale https://capytale2.ac-paris.fr/web/c/f807-628160/mcer

Savoirs et compétences :

❒ .

Soit la liste suivante pour effectuer les tests :

```
les_notes=[13.5, 7.1, 14.0, 9.7, 5.9, 5.8, 6.5, 6.2, 9.1, 11.7, 8.6, 16.7, 12.0, 12.8, 9.8, ↙
    10.1, 8.3, 6.5, 11.4, 12.5, 7.0, 6.9, 7.9, 7.0, 10.1, 10.8, 9.1, 5.6, 8.2, 3.4, 10.8, 8.2,↙
     13.3, 8.0, 14.9, 8.0, 8.2, 4.1, 6.5, 8.0, 8.2]
```

Moyenne et variance

Soit $a = [a_0, a_1, \ldots, a_{n-1}]$ une liste de nombres. On rappelle les définitions de la moyenne m et de la variance v :

$$m = \frac{1}{n}\sum_{i=0}^{n-1} a_i \qquad v = \frac{1}{n}\sum_{i=0}^{n-1}(a_i - m)^2.$$

Question 1 *Ecrire une fonction* `moyenne(a:list) -> float` *qui prend pour argument une liste de nombres* `a` *et qui renvoie la moyenne de* `a`.

Question 2 *Faire un test avec la liste* `a=[1,2,3,4,5]`.

Question 3 *Ecrire une fonction* `variance(a:list)` *qui prend pour argument une liste de nombres* `a`, *et qui renvoie la variance de* `a`.

Question 4 *Faire un test avec la liste* `a=[1,2,3,4,5]`.

Question 5 *Calculer la moyenne et l'écart-type (racine carrée de la variance) de la liste* `les_notes`.

R On trouvera une moyenne d'environ 9.08 et un écart-type d'environ 2.93.

On souhaite évaluer la moyenne par élément de deux listes, de même taille, de nombres comme présenté ci-dessous :

Liste a	a_0	a_1	a_2	a_3	...	a_{n-1}
Liste b	b_0	b_1	b_2	b_3	...	b_{n-1}
Liste moyennes	$\frac{a_0+b_0}{2}$	$\frac{a_1+b_1}{2}$	$\frac{a_2+b_2}{2}$	$\frac{a_3+b_3}{2}$	...	$\frac{a_{n-1}+b_{n-1}}{2}$

Question 6 *Ecrire une fonction* `moyennes(a:list,b:list)->list` *qui prend en entrée deux listes* `a` *et* `b` *de même taille (condition qui ne doit* `pas` *être vérifiée) et renvoie une liste de même taille contenant dans la case d'indice* `i` *la valeur moyenne des valeurs des flottants stockés dans les deux listes* `a` *et* `b` *à l'indice* `i`.

Question 7 *Tester le bon fonctionnement de votre fonction.*

Pour aller plus loin, évaluation de la moyenne glissante sur n éléments consécutifs.
Exemple avec n=3 :

Liste	12	13.5	11.2	11.7	12.1	...	...
Moyenne glissante	12	13.5	$\frac{12+13.5+11.2}{3}$	$\frac{13.5+11.2+11.7}{3}$	$\frac{11.2+11.7+12.1}{3}$	...	...

Question 8 *Ecrire une fonction* `moyenneGlissante(a:list, n:int)->list` *d'argument une liste* `a` *d'entiers ou flottants et un entier* `n` *et renvoyant la liste des moyennes glissantes sur* `n` *éléments consécutifs. Vous pourrez avantageusement utiliser le slicing.
 Faire un test avec la listes* `les_notes`.

Le maximum

Question 9 *Ecrire une fonction* `maximum(L:list)->float` *qui à partir d'une liste de flottants ou entiers renvoie le max de cette liste. La fonction prédéfinie en Python* `max` *ne doit pas être utilisée.*

Seuil

On souhaite maintenant, à partir d'une liste `a` d'entiers (ou flottants) et d'un entier (ou flottant) appelé `seuil` obtenir le nombre d'éléments de `L` majorés, au sens strict, par `seuil`.

Question 10 *Ecrire une fonction* `majores_par(L:list,x:float)->int` *réalisant cette opération.*

Question 11 *Tester le bon fonctionnement de votre fonction.*

Question 12 *Modifier la fonction précédente pour obtenir une fonction* `elements_majores_par(L:list,x:float)->l` *retournant la liste des éléments majorés par le seuil* `x`.

Question 13 *Tester le bon fonctionnement de votre fonction.*

Recherche séquentielle dans une liste

Question 14 *Définir une fonction* `sequentielle(a:list,x:float)->bool`*, d'arguments une liste* `a` *et un entier ou un flottant* `x`*, et qui renvoie le booléen* `True` *ou* `False` *selon que l'élément* `x` *est dans la liste* `x` *ou non.
  On fera attention à arrêter la recherche dès que l'élément est trouvé.
 Faire vos tests avec la liste* `les_notes` *en prenant pour* `x` *la première valeur de* `les_notes`*, puis la dernière et enfin une valeur au milieu.*

Question 15 *Tester le bon fonctionnement de votre fonction.*

Question 16 *Définir une fonction* `occurrenceElement(a:list,x:float)->int`*, d'arguments une liste* `a` *et un flottant* `x`*, et qui renvoie le nombre d'occurrence de l'entier ou flottant* `x`.

Question 17 *Tester le bon fonctionnement de votre fonction.*

Question 18 *Définir une fonction* `occurrenceListe(a:list)->list`*, d'argument une liste* `a` *d'entiers compris dans l'intervalle [0,k] et de longueur* `n` *tel que* $k < n$ *et qui renvoie la liste des nombres d'occurrence des entiers de la liste* `a`*. Cette valeur est 0 si le nombre n'est pas dans la liste.*

Question 19 *Tester le bon fonctionnement de votre fonction.*

Création de listes aléatoires

Pour tester vos algorithmes, il peut être utile de créer des listes quelconques de nombres. On peut écrire une fonction qui crée des listes aléatoires à partir de la bibliothèque `random`.

```
# import de la bibliothèque random
import random as r
# r est un alias
```

Question 20 *Comprendre le fonctionnement de la fonction* `randrange` *de cette bibliothèque (en utilisant l'aide et avec des exemples).*

```
help (r.randrange)
```

Question 21 *Ecrire une fonction* `hasard_liste(n:int,k:int)->list` *d'arguments deux entiers* `n` *et* `k` *permettant de générer une liste de* `n` *entiers aléatoires appartenant à* `range(k)`.

Question 22 *Tester votre fonction avec* $n = 10$ *et* $k = 7$.

Question 23 *Créer une liste à partir de la fonction* `hasard_liste(n:int,k:int)` *et tester votre fonction* `occurrenceListe(a:list)`.