International Online ESTYA Hackathon: Python programming Contest

Paris 8-9 of May 2021

Problème A

Dans une première pratique, nous chercherons à calculer la mention d'un étudiant à partir d'une valeur de moyenne donnée.

Nous demandons pour cela à l'utilisateur de saisir la moyenne d'un étudiant donné. Cette valeur doit évidemment être comprise entre 0 et 20.

En fonction de l'intervalle auquel appartient cette valeur, la mention correspondante sera déduite selon le schéma suivant :

- **Refusé** : si la moyenne est inférieure à 10.00
- **Passable** : pour les moyennes comprises entre 10.00 et 11.99
- **Assez Bien** : pour les moyennes comprises entre 12.00 et 13.99
- **Bien** : pour les moyennes comprises entre 14.00 et 15.99
- **Très Bien** : pour les moyennes comprises entre 19.00 et 17.99
- **Excellent** : pour les moyennes supérieures ou égales à 18.00

Question

Ecrire un script Python qui permet de :

- **Saisir une valeur réelle qui correspond à la moyenne d'un étudiant (appliquez le contrôle de saisie)**
- **Afficher la mention correspondante à cette moyenne**

Problème B : saisie répétitive et calculs élémentaires

Soit à saisir une suite de valeurs entières pour lesquelles nous demandons d'en déduire la plus petite valeur, la plus grande ainsi que la moyenne de l'ensemble des valeurs.

Pour cela, nous demandons à l'utilisateur de saisir ces valeurs de façon séquentielle. La fin de la saisie est marquée par **''done''**.

Une fois la saisie terminée, le programme devra afficher trois résultats (plus petite valeur, plus grande valeur et la moyenne des valeurs).

Exemple :

- Donnez une valeur : 10
- Donnez une valeur : 30
- Donnez une valeur : 5
- Donnez une valeur : 40
- Donnez une valeur : done

Le programme affichera les résultats suivants :

La plus **grande** valeur est : 40

La plus **petite** valeur est : 5

La moyenne des valeurs est : 21.5

Question

Ecrire un script Python qui permet de :

- **Saisir une suite de valeurs (pour signaler la fin de la saisie, l'utilisateur indique la mention''done'')**
- **Afficher la plus grande, la plus petite ainsi que la moyenne de ces valeurs**

Problème C : Force des mots de passe

L'administrateur du site ESTYA Education souhaite hausser le niveau de sécurité pour ses utilisateurs. Il vous demande pour cela développer un module qui permet d'évaluer la force du mot de passe proposé par les utilisateurs lors de la création de leurs comptes.

En pratique, il n'existe pas d'algorithme standard pour le calcul de la force d'un mot de passe. Il existe, cependant, des règles telles que la longueur du mot, la présence des caractères spéciaux, etc.

L'administrateur souhaite pour cela retenir les critères suivants :

- **C1** : Longueur du mot
- **C2** : Nombre de caractères alphabétiques
- **C3** : Nombre de caractères numériques
- **C4** : Présence de séquence de nombre : par exemples 789 ou 123
- **C5** : Présence de caractères consécutifs dupliqués

On vous demande de définir les **fonctions** suivantes pour aider l'administrateur à évaluer la force d'un mot de passe selon les critères précédents :

1. Une première fonction permettant de saisir un mot de passe : les deux contrôles suivants seront exigés :
    a. la taille du mot de passe est au minimum égale à 8 caractères,
    b. le mot de passe commence obligatoirement par un caractère alphabétique

2. Fonction qui permet de retourner la taille d'un mot de passe

3. Fonction qui permet de retourner le nombre de caractères alphabétiques dans un mot de passe

4. Fonction qui permet de retourner le nombre de caractères numériques dans un mot de passe

5. Fonction qui permet de vérifier si un mot de passe contient des séquences

6. Fonction qui permet de vérifier si un mot de passe contient des caractères consécutifs dupliqués

Exemple :

Soit le mot de passe suivant saisi par l'utilisateur : ''H@chathonn_123_fr''

- Longueur du mot de passe : 17
- Nombre de caractère alphabétiques : 11
- Nombre de caractères numériques : 3
- Présence de séquence : Vrai (123)
- Présence de caractères consécutifs et dupliqués : Vrai (''H@chatho**nn**_123_fr'')

Problème D : Indexation automatique du texte

L'indexation automatique du texte tend à rechercher les mots qui correspondent au mieux au contenu informationnel d'un document. On admet généralement qu'un mot qui apparaît souvent dans un texte représente un concept important. Ainsi, la première approche consiste à déterminer les mots représentatifs par leur fréquence.

Ainsi, pour un texte donné, un index sera constitué par l'ensemble des mots qui apparaissent dans le texte et pour chacun on retient la fréquence de sa répétition, soit l'ensemble des couples (**mot : fréquence**).

Exemple : soit un fichier texte dont le contenu est comme suit :

```
La programmation Python
Hackathon de programmation Python
la programmation Python boucles et tests
```

L'index dans ce cas sera le suivant :

{(**La** : 2), (**programmation** :3), (**Python** :3), (**Hackathon** : 1), (**de** :1), (**boucles** :1), (**et** :1), (**tests** :1)}

Question 1 :

Ecrire une fonction appelée **TermFrequency** qui étant donnée un fichier texte, elle retournera son index comme indiqué dans l'exemple précédent.

On s'aperçoit par la suite que les mots les plus fréquents sont des mots fonctionnels (ou mots outils, mots vides). En français, les mots tels que « de », « La» et « les » sont les plus fréquents.

Il est évident que l'on ne peut pas garder ces mots à haute fréquence mais peu porteur de sens en terme. C'est pourquoi nous chercherons dans un deuxième temps à éviter ces mots dans l'index du document textuel. L'idée sera de définir une liste constituée par l'ensemble de ces mots ; alors on se limitera à la liste suivante :

mots_vides = ['le', 'la','les','de','dans', 'à', 'après','assez','aucun','celui', 'ces,'cependant','devant']

Dans l'exemple du fichier précédent, l'index sera comme suit :

{(**programmation** :3), (**Python** :3), (**Hackathon** : 1), (**boucles** :1), (**tests** :1)}

Question 2 :

Améliorer la fonction précédente pour ne pas considérer les mots vides dans l'index final.

Nous cherchons finalement à généraliser la fonction précédente sur un ensemble de fichiers contenus dans un répertoire. Le résultat de l'indexation sera un fichier dont le format des lignes est indiqué par l'exemple suivant :

Exemple

File1.txt Python 2

File1.txt programmation 1

…

File2.txt Python 5

File2.txt compétition 2

…

Question 3 :

Développez une fonction **indexer_Repertoire** qui aura un seul paramètre qui indique l'intitulé répertoire contenant les fichiers à indexer.

Cette fonction devra par la suite générer en résultat un fichier intitulé "**Index.tx**" dont le format est précisé dans l'exemple précédent

Problème E : Cryptographie

Une unité de recherche se propose sécuriser l'échange de documents entre ses intervenants. Pour cela, on demande au responsable du service informatique de développer une solution propriétaire pour rendre l'accès aux données plus sécurisé.

Après réflexion, l'ingénieur propose dans un premier temps une manière basique de chiffrement qui consiste à décaler chaque lettre du texte à crypter régulièrement de **10** positions de l'alphabet. Alors on suppose que le texte à chiffrer est **composé uniquement de lettre en majuscules**.

Illustration

- A →K
- B→L
- …
- Q→A
- R→B
- …
- Y→I
- Z→J

Par la suite le texte ''**BONJOUR EITECH**'' sera crypté en ''**LYXTYB 0VSDOMR**''

Question 1 :

Ecrire une fonction qu'on appellera **crypter** qui admet un paramètre le texte à chiffrer et retourne en résultat le texte ainsi chiffré selon la méthode de décalage décrite ci-dessus.

Une fois cette première version implémentée, une série de tests montre que cette méthode de chiffrement n'est pas assez robuste.

L'ingénieur propose par la suite d'améliorer l'algorithme de chiffrement précédent en introduisant la notion de clé.

Ceci consiste à utiliser une clé dont le rôle est de rendre le décalage variable comme l'indique l'exemple suivant :

L'exemple ci-dessous effectue le chiffrement du texte "**ELITECH CONTEST**" avec la clé "**ANIS**" :

1ère ETAPE : Le texte à crypter ainsi que la clé sont représentés par deux tableaux comme suit :

Texte à crypter : A chaque lettre, on associe sa position dans l'alphabet (0-25)

E	**L**	**I**	**T**	**E**	**C**	**H**		**C**	**O**	**N**	**T**	**E**	**S**	**T**
4	11	8	19	4	2	7		2	14	13	19	4	18	19

Clé : A chaque lettre, on associe sa position dans l'alphabet (0-25) ; le tableau devra avoir la même taille que le tableau du texte à crypter…

A	**N**	**I**	**S**											
0	13	8	18	0	13	8	18	0	13	8	18	0	13	8

2ème ETAPE :

Par la suite, chaque caractère du texte initial sera décalé d'autant de positions que la valeur correspondante dans le tableau Clé.

Exemples :

- Position 1 (lettre 'E') : la même position dans le tableau Clé est la lettre 'A' (ordre=0), par la suite la lettre 'E' ne sera pas décalée.
- Position 2 (lettre 'L') : la même position dans le tableau Clé est la lettre 'N' (ordre=13), par la suite la lettre 'L' sera décalée de 13 positions pour avoir la lettre 'Y'.
- Position 4 (lettre 'T') : la même position dans le tableau Clé est la lettre S (ordre=18), par la suite la lettre 'T' sera décalée de 18 positions pour avoir la lettre 'L'.

Le tableau final du texte chiffré sera le suivant :

Texte chiffré : Résultat

4	24	16	11	4	15	15		2	1	21	11	4	5	1
E	**Y**	**Q**	**L**	**E**	**P**	**P**		**C**	**B**	**V**	**L**	**E**	**F**	**B**

Ainsi le texte chiffré est "**EYQLEPP CBVLEFB**"

Question 2 :

Ecrire une fonction qu'on appellera **crypter_cle** qui admet deux paramètres (le texte à chiffrer et la valeur de la clé) et retourne en résultat le texte ainsi chiffré selon la méthode de décalage utilisant la clé comme l'indique l'exemple précédent.