TP d'informatique 11

Résumé de texte en 200 mots +/- 10%...

On propose de coder quelques fonctions pouvant aider monsieur V, professeur de Lettres, à gagner du temps pour la tâche de correction des résumés de texte de ses étudiants...

On va concevoir un programme demandant aux élèves de saisir leurs résumés, puis qui effectuera quelques opérations de routine : comptage du nombre de mots, vérification de l'utilisation d'un mot imposé, recherche de plagiat...

Partie A — Recherche de caractères, et de chaînes de caractères

On se propose dans un premier temps de coder quelques fonction "simples" sur les chaînes de caractères.

1. Proposer une fonction `test(résumé, caractère)` prenant en argument une chaîne de caractères appelée `résumé` et un caractère nommé `caractère`, et retournant True si ce caractère est présent dans le résumé.

2. On souhaite créer une fonction `positioncaractère` prenant en argument une chaîne de caractère appelée `résumé` et un caractère, et renvoyant la liste des positions où apparaît le caractère dans le résumé.

Par exemple : `positioncaractère('Quel beau texte', 'u')` doit retourner `[1,8]`.

Compléter le code ci-dessous :

```
def positioncaractère (résumé, carac):
        ListPos =[]
        for i in range(len(résumé)):
        # A compléter

        return ListPos
```

3. En s'inspirant des fonctions précédentes, proposer une fonction `nombremots` prenant en argument une chaîne de caractères appelée `résumé` et renvoyant le nombre de mots de cette chaîne de caractères (pour simplifier, on pourra supposer que la chaîne de caractères `résumé` ne contient pas d'apostrophe).

Monsieur V a donné comme consigne aux élèves d'utiliser au moins une fois le mot "enfance" dans leur résumé. On propose ici de coder une fonction permettant de vérifier que la consigne a été respectée.

On souhaite tout d'abord coder une fonction `appartient1(résumé,mot)` prenant en argument une chaîne de caractère appelée `résumé` et un mot de deux lettres et qui retourne True si le mot apparaît bien dans le texte.

4. Parmi les 3 fonctions ci-dessous, quelle(s) sont la (les) fonction(s) pouvant convenir ? Pour les fonctions non valides, préciser les problèmes identifiés.

```
# PROPOSITION 1
def appartient1(résumé,mot) :
        result =False
        if mot in résumé :
                result =True
        return result
```

```python
# PROPOSITION 2
def appartient1(résumé,mot) :
    result =False
    if mot[0] in résumé :
        for j in positioncaractère (résumé,mot[0]):
            if j ==0:
                if mot ==résumé[0]+résumé[1] and résumé[2] in '. ?! :;':
                    result =True
            if j >0:
                if mot[1] ==résumé[j+1] and résumé[j-1] ==" " and résumé[j+2] in '.?! :;':
                    result =True
    return result
```

```python
# PROPOSITION 3
def appartient1(résumé,mot) :
        result =True
        if len( positioncaractère (résumé,mot[0])) ==len( positioncaractère (résumé,mot[1])) :
                for i in range(len( positioncaractère (résumé,mot[0]))) :
                        if positioncaractère (résumé,mot[0])[1] != positioncaractère (résumé,mot[0])[0] +1 :
                                result =False
        else :
                result =False
        return result
```

Partie B — Recherche de plagiat

Dans un monde idéal les élèves ne trichent pas mais malheureusement Monsieur V a depuis longtemps perdu ses illusions... Il cherche donc à vérifier si des élèves ont recopié une partie de leur résumé les uns sur les autres.

Parmi les algorithmes permettant de détecter les plagiats on peut citer *l'algorithme de Levenshtein*. Cet algorithme permet de comparer deux chaînes de caractères en déterminant le plus petit nombre de modifications nécessaires pour qu'elles soient égales. On appelle *distance de Levenshtein* entre deux mots M et P le coût minimal pour transformer M en P en effectuant les seules opérations élémentaires suivantes :

- ➢ *substitution d'un caractère de M par un caractère différent de P*
- ➢ *insertion (ou ajout) dans M d'un caractère de P*
- ➢ *suppression (ou effacement) d'un caractère de M.*

On associe à chacune de ces opérations un coût. Généralement, le coût est égal à 1 pour chacune des trois opérations.

5. Pour deux mots M et P donnés, comment interpréter une distance de Levenshtein proche de 0 ?

6. On propose de coder une fonction calculant la *distance de Levenshtein simplifiée* entre deux mots de même longueur, et uniquement basée sur l'opération de substitution : quand on compare deux mots M et P, la distance augmente d'une unité dès que l'on repère qu'il suffit de substituer un caractère de P pour retrouver le mot M.

Exemple : M : "bonjour" et P : "bongour" la distance de Levenshtein simplifiée est égale à 1.

Ecrire une fonction `distance` prenant en argument deux chaînes de caractères nommées `résumé1` et `résumé2` de même longueur et qui retourne la distance de Levenshtein simplifiée.

Partie C — Programme principal

7. Proposer un programme principal effectuant les opérations suivantes.

- ➢ Demande à deux élèves de saisir leurs résumés que l'on stockera dans les variables résumé1 et résumé2 (on supposera que ces deux résumés possèdent la même longueur) ;
- ➢ Décompte des mots et vérification que la consigne 200 mots +/- 10% est respectée. Affichage d'un message précisant si cette consigne a été respectée ou pas.
- ➢ Vérification pour les deux résumés que le mot "enfance" apparaît au moins une fois dans le texte. Affichage d'un message précisant si la consigne a été respectée ou pas. A cet effet, on pourra supposer qu'une fonction `appartient2` a été codée, qui prend en argument une chaîne de caractère appelée `résumé` et un mot quelconque, et qui précise si ce mot apparaît dans le résumé ainsi que le nombre de fois où il apparaît.
- ➢ Calcul de la distance simplifiée de Levenshtein et affichage d'un message d'alerte si cette distance franchit la valeur seuil 50.

Dans cet algorithme principal, on pourra réutiliser les fonctions précédentes même si le codage exact de ces fonctions n'a pas été réussi.

Partie D — Post-traitement

Sensible à la syntaxe et à la grammaire, Monsieur V souhaite corriger une erreur courante, celle consistant à débuter une phrase par une lettre minuscule. A cette fin, il cherche à écrire un programme repérant ces erreurs, et les rectifiant de manière systématique.

Pour réaliser ce programme, il peut être utile de savoir qu'en informatique, à chaque caractère est associé un nombre appelé **code ASCII**, qui est un entier compris entre 0 et 255. En voici quelques exemples :

Caractère	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Code ASCII	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90

Caractère	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Code ASCII	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122

Caractère	.	!	?
Code ASCII	46	33	63

En outre, la fonction `ord` permet d'obtenir le code ASCII d'un caractère. Sa bijection réciproque est la fonction `chr`, qui permet d'associer à un entier de $[\![0, 255]\!]$ le caractère ayant cet entier pour code ASCII.

Par exemple : `ord('A')=65`, et `chr(65)='A'`.

8. Ecrire une fonction `MAJ(carac)`, qui reçoit comme paramètre un caractère `carac`, qui retourne la lettre majuscule correspondante lorsque `carac` est une lettre minuscule de l'alphabet, et qui retourne `carac` sinon.

Par exemple : `MAJ('a')='A'` ; `MAJ('A')='A'`, et `MAJ('?')='?'`.

9. Ecrire une fonction `MAJCORR(résumé)`, qui reçoit comme paramètre une chaîne de caractères nommée `résumé`, qui retourne la chaîne de caractères où les erreurs évoquées en préambule de cette partie ont été corrigées.

Par exemple, si la chaîne de caractères `résumé` est :

```
il fait chaud! il fait beau. que se passe t-il? c'est l'été.
```

l'instruction `MAJCORR(résumé)` doit produire le résultat :

```
Il fait chaud! Il fait beau. Que se passe t-il? C'est l'été.
```