UNIVERSITÉ DE SHERBROOKE
DÉPARTEMENT D'INFORMATIQUE

IFT 211 / IFT 776

Les structures de contrôle en Python

Hiver 2016

Laboratoire #2 : exercices 1 à 5
Travail pratique #2 : exercice 6

Le but de ce laboratoire est de vous familiariser avec la programmation, principalement celle reliée à l'utilisation des énoncés de sélection et d'itérations.
Ce laboratoire et le devoir devront être complétés avant le 22 janvier 2016 à 23h59

1 Corrigez les erreurs

Pour les programmes `pgm1.py` à `pgm6.py` fournies sur la page WEB pour le laboratoire, déterminez quelles sont les erreurs présentes dans le programme et corrigez-les.

2 La plus grande valeur d'une liste

Complétez le programme `pgm8.py` afin qu'il produise le plus grand nombre d'une liste de valeurs lues en entrée.

3 Matplotlib

Complétez le programme `pgm9.py` afin qu'il affiche sur un graphe le sinus et le cosinus d'une liste de valeurs lues en dans le fichier `pgm9.donnees`.

4 Compteur de voyelles

Modifiez le programme `pgm7.py` afin qu'il encrypte un message selon le code de César (plutôt que de trouver les voyelles). Avec ce code, inventé par Jules César, on remplace chaque lettre du texte par la lettre qui se situe trois positions plus loin dans l'alphabet. Par exemple, la lettre a est remplacée par la lettre d. Si le caractère à remplacer se trouve à la fin de l'alphabet vous devez prévoir la remplacer par une lettre au début de l'alphabet. Ainsi si la dernière lettre de l'alphabet est "z" sa lettre de substitution est "c" (considérant l'alphabet comme une liste circulaire). L'alphabet à utiliser est déjà défini dans `pgm7.py`.

5 Conversion de notes

Dans les pays occidentaux, il y a deux systèmes qui sont utilisés pour représenter les notes de musique. Il y a la notation basée sur les sept premières lettres de l'alphabet héritée de la Grèce antique et conservée par les pays dits protestants, et celle basée sur les notes « do ré mi fa sol la si » inventée par Guido d'Arezzo et utilisée par les pays dits catholiques. Les équivalences entre les notes sont : A (la), B (si), C (do), D (ré), E (mi), F (fa) et G (sol).

Écrivez un programme (`note.py`) qui, selon le choix de l'utilisateur, transforme une série de notes de musique d'une des deux notations existantes vers la seconde. Le programme lira de façon répétitive le choix de système de notation («a» pour la représentation alphabétique, «s» pour la représentation syllabique, «q» pour terminer) suivie de la note à transformer.

Un exemple de données en entrée serait :	Le résultat de l'exécution du programme avec ces données serait :
a B s la q	si A

6 Travail pratique #2 : Correcteur d'orthographe

Le travail peut être fait en équipe de deux personnes.
Le travail est à remettre le 22 janvier 2016 avant 23h59.

6.1 Énoncé du problème

On veut bâtir un programme (`correcteur.py`) qui implante un correcteur d'orthographe. Ce programme doit lire des mots en entrée et suggérer des corrections si, évidemment, le mot n'apparaît pas dans le dictionnaire utilisé.

Le programme doit donc d'abord se procurer un dictionnaire. Celui-ci se trouve dans un fichier, il doit demander le nom du fichier contenant le dictionnaire. Une fois le dictionnaire en place, le programme lit des mots jusqu'à ce qu'il rencontre le mot clé «**fin...**». Lorsqu'il rencontre ce mot clé, le programme termine son exécution.

Lorsqu'il lit un mot, le programme vérifie s'il est présent dans le dictionnaire. S'il y est, il indique que le mot est bien orthographié et il demande un autre mot. S'il n'y est pas, il suggère une liste de mots. Afin de construire cette liste, le programme doit trouver la *distance d'édition* entre le mot à corriger et tous les mots du dictionnaire. La liste contiendra tous les mots dont la distance d'édition est inférieure à trois. Cette liste peut être affichée par ordre alphabétique ou par ordre de distance. Le programme s'informe auprès de l'usager de l'ordre d'affichage. Il trie et affiche la liste autant de fois que nécessaire. Une fois que l'usager en a terminé avec l'affichage de la liste (en entrant le critère «**suivant**»), on passe au mot suivant.

Les critères permettant d'établir l'ordre de tri pour l'affichage de la liste sont :

— l'ordre alphabétique («**alpha**») ;

— la distance («**distance**»).

Le programme lit donc premièrement le nom du fichier contenant le dictionnaire. Il lit ensuite les mots à corriger. Pour chaque mot à corriger il détermine s'il est syntaxiquement correct. S'il ne l'est pas il demande l'ordre d'affichage de la liste des mots suggérés. Le critère «**suivant**» suspend l'affichage des suggestions. Le mot «**fin...**» arrête le programme (on ajoute un «...» à la fin pour le distinguer du mot «**fin**» qui est un mot valide du dictionnaire).

6.2 Notes importantes

— numpy
Pour déterminer la distance entre deux mots, vous avez besoin de matrices. Pour définir des matrices en Python, vous devez utiliser la bibliothèque `numpy`. Le bout de code suivant montre comment définir et initialiser une matrice avec `numpy`.

```
import numpy as np
...
# définition d'une matrice 10 x 10 initialisée à 0
matrice = np.zeros(shape=(10,10),dtype=int)

# initialisation de la ligne 0
for i in range(10):
    matrice[0, i] = i
# initilisation de la colonne 0
for i in range(10):
    matrice[i, 0] = i
```

— Utilisation du type dictionnaire
Pour emmagasiner les suggestions de mots, il est recommandé d'utiliser une variable de type dictionnaire de Python. L'indice sera le mot suggéré et le second élément sera la distance d'édition.
Le bout de code suivant montre comment utiliser un dictionnaire et le trier.

```
# Définition d'un dictionnaire Python
mon_dico = {}

# Ajout de plusieurs couples mot:valeur
mon_dico.update({"bonjour":7})
mon_dico.update({"vrai":4})
mon_dico.update({"blabla":6})
mon_dico.update({"inconditionnellement":20})

# Affichage du dictionnaire
print(mon_dico)

# Affichage du de la valeur associée à "bonjour"
print(mon_dico["bonjour"])

# Affichage du dictionnaire en ordre alphabétique
mots_triés = sorted(mon_dico)
for i in mots_triés:
    print(i, mon_dico[i])

# Affichage du dictionnaire selon l'ordre des valeurs
valeurs_triées = sorted(mon_dico, key=mon_dico.__getitem__)
for i in valeurs_triées:
    print(mon_dico[i], i )
```

— Affichage sur un nombre de colonnes fixe d'une chaîne de caractères
Afin que la liste de suggestion soient bien lisible, il est suggéré d'afficher les mots sur le même nombre de colonne. Le code suivant montre comment y parvenir.

```
# Affichage structuré (sur 20 colonnes) du dictionnaire et de la valeur
for mot in mon_dico:
    print( '{0: <20}'.format(mot), " :   ", mon_dico[mot])
```

6.3 Exemple de jeu de données

```
dictionnaire          // nom du fichier contenant le dictionnaire
bonjour               // premier mot à corriger
bonyour               // second mot à corriger
alpha                 // on veut afficher les suggestions en ordre alphabétique
distance              // on veut afficher les suggestions selon la distance
suivant               // on passe au mot suivant
boyor                 // troisième mot à corriger
alpha                 // on veut afficher les suggestions en ordre alphabétique
suivant               // on passe au mot suivant
fin...                // on termine le programme
```

6.4 Exemple de dictionnaires

Un dictionnaire est tout simplement une suite de mots. Voici une exemple de dictionnaire :

```
ajax
bonjour
retour
bien
long
longueur
silence
silencieux
tour
toujours
jour
alphabet
soleil
lune
chat
chien
kayak
canot
equestre
cheval
mot
saut
sceau
sous
dollar
argent
monnaie
bilan
statistique
probabilite
anticonstitutionnellement
```

Vous trouverez sur le site WEB du cours deux exemples de dictionnaires que vous pourrez utiliser pour créer votre propre dictionnaire.

7 La distance d'édition

La distance d'édition (ou distance de Levenshtein) est une mesure de similitude entre deux chaînes de caractères. La distance correspond aux nombres de caractères que l'on doit modifier, détruire ou insérer pour transformer une chaîne C1 en une chaîne C2.

Par exemple, la distance entre la chaîne $C1$ = «`test`» et la chaîne $C2$ = «`test`» est 0. La distance entre $C1$ = «`test`» et $C2$ = «`tent`» est 1 car en modifiant un seul caractère de $C2$ on peut obtenir $C1$. La distance entre $C1$ = «`bonjour`» et $C2$ = «`boyour`» est 2, car on doit insérer une lettre (`n`) après la première lettre `o` et changer la lettre `y` pour un `j`.

Le premier algorithme servant à calculer cette distance a été proposé en 1965 par le scientifique russe Vladimir Levenshtein.

Pour calculer la distance entre deux mots, on utilise l'algorithme suivant :

1. Soit $C1$ et $C2$ deux mots de longueur $l1$ et $l2$ respectivement.
2. Si $l1 = 0$, on retourne $l2$.
3. Si $l2 = 0$, on retourne $l1$.
4. On construit une matrice contenant $l2 + 1$ lignes et $l1 + 1$ colonnes.
5. On initialise la première ligne de la matrice à $0, 1, 2, ..., l1$.
6. On initialise la première colonne de la matrice à $0, 1, 2, ..., l2$.
7. Pour chaque caractère de C1 (indice i).
   (a) Pour chaque caractère de C2 (indice j).
   - Si $C1[i] = C2[j]$, le coût est 0.
   - Si $C1[i] \neq C2[j]$, le coût est 1.
   - On initialise la cellule $c[i, j]$ de la matrice au minimum entre :
     i. la valeur de la cellule immédiatement au-dessus $+1$ $(c[i-1, j] + 1)$;
     ii. la valeur de la cellule immédiatement à gauche $+1$ $(c[i, j-1] + 1)$;
     iii. la valeur de la cellule $c[i-1, j-1]$+le coût.
8. Après avoir parcouru toute la matrice, la distance se trouve dans la cellule en position $[l1, l2]$;

Exemple

Voici une exemple qui calcule la distance entre les mots «`bonjour`» (*C1* de longueur $l1 = 7$) et «`boyour`» (*C2* de longueur $l2 = 6$). On construit donc une matrice contenant 8 colonnes et 7 lignes.

La matrice de départ, une fois initialisée, sera donc :

		b	o	n	j	o	u	r
	0	1	2	3	4	5	6	7
b	1							
o	2							
y	3							
o	4							
u	5							
r	6							

Les étapes subséquentes produiront :

1 →

		b	o	n	j	o	u	r
	0	1	2	3	4	5	6	7
b	1	0						
o	2	1						
y	3	2						
o	4	3						
u	5	4						
r	6	5						

2 →

		b	o	n	j	o	u	r
	0	1	2	3	4	5	6	7
b	1	0	1					
o	2	1	0					
y	3	2	1					
o	4	3	2					
u	5	4	3					
r	6	5	4					

3 →

		b	o	n	j	o	u	r
	0	1	2	3	4	5	6	7
b	1	0	1	2				
o	2	1	0	1				
y	3	2	1	1				
o	4	3	2	2				
u	5	4	3	3				
r	6	5	4	4				

4 →

		b	o	n	j	o	u	r
	0	1	2	3	4	5	6	7
b	1	0	1	2	3			
o	2	1	0	1	2			
y	3	2	1	1	2			
o	4	3	2	2	2			
u	5	4	3	3	3			
r	6	5	4	4	4			

5 →

		b	o	n	j	o	u	r
	0	1	2	3	4	5	6	7
b	1	0	1	2	3	4		
o	2	1	0	1	2	2		
y	3	2	1	1	2	3		
o	4	3	2	2	2	2		
u	5	4	3	3	3	3		
r	6	5	4	4	4	4		

6 →

		b	o	n	j	o	u	r
	0	1	2	3	4	5	6	7
b	1	0	1	2	3	4	5	
o	2	1	0	1	2	2	3	
y	3	2	1	1	2	3	3	
o	4	3	2	2	2	2	3	
u	5	4	3	3	3	3	2	
r	6	5	4	4	4	4	3	

7 →

		b	o	n	j	o	u	r
	0	1	2	3	4	5	6	7
b	1	0	1	2	3	4	5	6
o	2	1	0	1	2	2	3	4
y	3	2	1	1	2	3	3	4
o	4	3	2	2	2	2	3	4
u	5	4	3	3	3	3	2	3
r	6	5	4	4	4	4	3	2

Dans la coin inférieur droit de la matrice, on retrouve la distance entre les deux mots. La distance entre «`boyour`» et «`bonjour`» est de 2.

Cette mesure de distance sert évidemment dans la correction d'orthographe Toutefois elle est

aussi utilisée dans plusieurs autres domaines :

- Dans le traitement de fichiers, on établit la différence entre deux fichiers afin d'éviter le transfert de copies entières de fichiers. On transfert seulement les différences (la commande «`diff`» de Unix utilise un algorithme spécial de distance d'édition).
- Lors de la mise à jour d'écran éloigné (comme lors d'une session à distance par l'intermédiaire d'outils tels *ssh* et *telnet*). Cette approche permet la mise à jour seulement des zones modifiées de l'écran. Les modifications sont établies par une mesure de distance.
- Des algorithmes établissant la distance d'édition sont aussi utilisés en reconnaissance de la parole. On tente de trouver une correspondance entre un nouvel énoncé vocal (utterance) et un autre emmagasiné dans une bibliothèque.
- Des algorithmes de distance d'édition sont à la base de la bioinformatique. En effet, on tente d'établir la distance entre deux séquences d'ADN, d'ARN ou de protéines afin d'en déduire certaines fonctions ou propriétés communes, ou d'établir une relation de descendance entre différents organismes. De plus, le concept de distance d'édition est très similaire à d'autres concepts utilisés en biologie moléculaire tels l'alignement de séquence et la recherche de la plus longue sous-chaîne commune.
- Pour la détection de cas de plagiat on se sert aussi de la distance d'édition.

8 Soumission.

8.1 Soumission du laboratoire 2

1. Connexion sur Turnin (http ://opus.dinf.usherbrooke.ca)
2. Soumettre les fichiers pgm1.py à pgm9.py et note.py dans le projet **labo2**

8.2 Soumission du travail patique 2

1. Connexion sur Turnin (http ://opus.dinf.usherbrooke.ca)
2. Soumettre le fichier correcteur.py dans le projet **tp2**