TP de Langages de script n° 4 : mutations de listes

Les mutations de listes ou créations fonctionnelles de listes (*list comprehension* en Anglais, souvent traduit de façon assez moche par *compréhension de listes*) sont un des puissants outils de python qui simplifient la vie du programmeur. Les formes les plus simples des mutations de listes sont les suivantes:

```
[transformation(element) for element in liste]
[element for element in liste if propriete(element)]
```

Mais on peut parfaitement les combiner en:

```
[transformation(element) for element in liste if propriete(element)]
```

Par exemple:

```
>>> liste = [1, 34, 5, 4, 7, 8, 93]
>>> [2*i for i in liste]
[2, 68, 10, 8, 14, 16, 186]
>>> [i for i in liste if i%2 == 0]
[34, 4, 8]
>>> [2*i for i in liste if i%2 == 0]
[68, 8, 16]
```

On peut même faire encore plus fort:

```
>>> liste = 'ceci est une liste de mots'.split()
>>> liste
['ceci', 'est', 'une', 'liste', 'de', 'mots']
>>> [u+v for u in liste for v in liste if u != v]
['ceciest', 'ceciune', 'ceciliste', 'cecide', 'cecimots', 'estceci', 'estune',
'estliste', 'estde', 'estmots', 'unececi', 'uneest', 'uneliste', 'unede',
'unemots', 'listececi', 'listeest', 'listeune', 'listede', 'listemots',
'dececi', 'deest', 'deune', 'deliste', 'demots', 'motsceci', 'motsest',
'motsune', 'motsliste', 'motsde']
>>> alphabet = 'ab'
>>> mots_longueur_3 = [i+j+k for i in alphabet for j in alphabet for k in alphabet]
>>> mots_longueur_3
['aaa', 'aab', 'aba', 'abb', 'baa', 'bab', 'bba', 'bbb']
```

Exercice 1 : Premières mutations de listes.

Grâce à des mutations de listes, produisez les listes suivantes:

1. Liste des entiers pairs entre 0 et 10.
2. Sous-liste des éléments de rang pair d'une liste donnée.
   Indication: pour associer à chaque élément de liste son rang utilisez la fonction `enumerate`.
3. Programmez une fonction `diviseurs(n)` qui trouve tous les diviseurs d'un entier. Utilisez cette fonction pour créer la liste de tous les nombres premiers jusqu'à `n`.
4. Ecrivez une fonction `sans_e` qui prend en argument une liste de mots et affiche la liste de ceux qui ne contiennent pas la lettre `'e'`.

5. Ecrivez une fonction `anti_begue` qui prend en argument une chaîne de caractères et en crée une nouvelle dans laquelle ont été supprimés les mots consécutifs identiques. Votre fonction devra transformer le texte d'origine en liste de mots avant tout traitement et transformer la liste nouvellement obtenue en chaîne de caractères. Pour cela, regarder l'aide en ligne des méthodes `split` et `join` du type `str`. Par ailleurs, on remarque la chose suivante: si on zippe (`zip`) la liste des mots de la phrase avec la liste des mots décalée de 1, les mots répétés se retrouvent ensemble.

```
>>> begue
['ceci', 'est', 'un', 'un', 'test']
>>> begue_decale
['est', 'un', 'un', 'test']

>>> list(zip(begue, begue_decale))
[('ceci', 'est'), ('est', 'un'), ('un', 'un'), ('un', 'test')]
```

Il suffit alors de choisir les bons mots (attention au dernier mot de la phrase...).

Exercice 2 : On observe les types et on imprime en grand.

1. Créez une variable `x` de valeur `'toto'`. La fonction `type` vous donne le type de cette variable. Comparez le résultat de l'application de la fonction `dir` sur `x` et sur son type. Toutes les variables que vous déclarez (entier, chaînes de caractère, ...) sont en fait des objets d'une classe d'où le fait que `dir(x)` et `dir(str)` retournent la même chose. Pour utiliser une méthode il suffit d'utiliser la syntaxe `x.methode()`. Par exemple si `x` vaut `'toto'` alors `x.isupper()` va tester si `x` est bien écrit en majuscules. On peut également utiliser la syntaxe `str.isalpha('toto')` si on veut utiliser la méthode `isalpha` définie dans la classe `str` sans déclarer de variables.
2. Comment mettre une chaîne de caractères en minuscules ?
3. Téléchargez le fichier `grandes_lettres.py` et importez le module correspondant dans l'interpréteur python. Regardez l'aide en ligne de ce module.
4. À l'aide du module `grandes_lettres`, écrivez une fonction `grand_message` qui prend en argument une chaîne de caractères et réécrit le texte en grandes lettres. Pour cela il faudra imprimer les étoiles ligne à ligne.

   Exemple : 'toto' sera réécrit

```
*****  **  *****  **
  *   *  *   *   *  *
  *   *  *   *   *  *
  *   *  *   *   *  *
  *    **    *    **
```

5. À l'aide de l'instruction `input()` et du code que vous avez mis au point précédemment, créez un script `grand_message.py` demandant à l'utilisateur de rentrer une chaîne de caractère et l'imprimant en grand.

Exercice 3 : Mot sans cube.

On considère les mots sur un alphabet à deux lettres $\{a, b\}$. On dit qu'un mot (c'est-à-dire une suite de lettres) possède un cube s'il possède un facteur de la forme uuu où u est un mot non vide. Par exemple *baaa* et *bababa* possèdent un cube alors que *bababb* est sans cube. On peut montrer qu'il existe un mot infini sans cube (mais ce n'est pas l'objet de cet exercice). Nous allons être plus modestes et écrire une fonction qui teste si un mot est sans cube.

1. Que fait la méthode `endswith` de la classe `str` ? Trouvez la méthode symétrique.
2. Dans un interpréteur Python, regardez ce que donne `3*'ab'`.
3. Ecrivez une fonction `prefixes` qui renvoie la liste des préfixes d'un mot.
4. Écrivez une fonction `est_sans_prefixe_cube` qui prend en argument une chaîne de caractères et détermine si elle possède un cube comme préfixe.
5. Écrivez une fonction `est_sans_cube` qui prend en argument une chaîne de caractères et détermine si elle est sans cube. Pour cela vous pourrez tester les suffixes de l'argument grâce à la fonction précédente.

Exercice 4 : Codes secrets.

1. À partir de votre terminal (et sans avoir lancé l'interpréteur Python), consultez grâce à `pydoc` l'aide en ligne des fonctions `chr` et `ord`.
2. Créez la chaîne de caractères `message` contenant la valeur `'ceci est mon message a chiffrer'`. À l'aide d'une boucle `for` sur cette chaîne, chiffrez-la par un décalage de 3 (*chiffre de César*). Par exemple la lettre `a` sera chiffrée par la lettre `d` (et la lettre `x` par la lettre `a`).
3. Créez maintenant une chaîne de caractères `clef` dont la valeur est `'secret'`. Chiffrez la chaîne `message` par décalage grâce à la clef `clef` (*chiffre de Vigenère*). Ce chiffrement se fait en décalant la i^{e} lettre du message grâce à la i^{e} lettre de la clef (on reprend au début de la clef quand on a fini de lire celle-ci). Mathématiquement on peut écrire : lettre chiffrée = (lettre + clef) modulo 26. Pour simplifier, on ne modifiera pas les caractères blancs (espace, tabulation et retour chariot), la ponctuation...

Exercice 5 : un début de cryptanalyse

Une ancienne façon de chiffrer des messages consiste à procéder par substitutions de lettres. Par exemple, le chiffrement de César consiste à effectuer une rotation de 3 lettres sur l'alphabet. Mais cela peut se généraliser à une permutation quelconque sur l'alphabet.

Pour cryptanalyser un tel chiffrement on se base sur des tables d'occurrences des lettres dans la langue du message à décrypter. Un tel fichier `occurrences` est donné pour le français sous la forme `lettre:pourcentage` pour indiquer que la lettre `lettre` apparaît `pourcentage` fois sur 100 lettres dans cette langue.

Avant cela nous allons commencer par chiffrer le texte qui servira d'exemple.

1. Ecrivez une fonction qui enlève les espaces et les signes de ponctuation d'un texte. Vous pourrez adapter la fonction `decoupe` de la question 4 de l'exercice 1 du TP3.
2. Ecrivez une fonction `fic_to_dic` qui à partir d'un fichier associant bijectivement à chaque lettre une autre lettre sous la forme `lettre1:lettre2`, crée le dictionnaire correspondant. Vous pourrez adapter la fonction de de la question 1 de l'exercice 4 du TP2.
3. Chiffrez le texte à partir d'un fichier où chaque ligne est de la forme `lettre:traduction`.

4. Ecrivez une fonction `count_occurrences` qui, à partir d'un texte donné, crée un dictionnaire des couples (lettre, pourcentage).
5. Créez le dictionnaire des couples (lettre, pourcentage) à partir d'`occurrences`. Puis, à partir de ce dictionnaire et de celui de la question précédente, créez en un (lettre de `occurrences`, lettre du texte) qui permettrait de décrypter ce texte s'il suivait exactement les pourcentages de la langue.
6. À partir de ce dictionnaire, écrivez la fonction qui remplace les lettres du texte par celles qui leurs correspondent dans le dictionnaire.
7. Affichez le texte décrypté.

On voit bien que cette façon de cryptanalyser n'est malheureusement pas suffisante... Pour aller plus loin, il faut prendre en compte les bigrammes, c'est-à-dire les couples de lettres.