1 Correction du TD 3

Correction de la partie A (exercice 1) :

```
def lettre_suivante(lettre) :
    c = ord(lettre) - 97
    c = (c + 1) % 26
    return chr (c + 97)

print (lettre_suivante('m'), lettre_suivante('z'))
```

Correction de la partie B (exercice 2) :

```
mots = ['eddard', 'catelyn', 'robb', 'sansa', 'arya', 'brandon',
        'rickon', 'theon', 'rorbert', 'cersei', 'tywin', 'jaime',
        'tyrion', 'shae', 'bronn', 'lancel', 'joffrey', 'sandor',
        'varys', 'renly', 'a' ]

def mots_lettre_position (liste, lettre, position) :
    res = [ ]
    for mot in liste :
        if position < len(mot) and mot[position] == lettre :
            res.append (mot)
    return res

r = mots_lettre_position ( mots, 'y', 1)
print (r)
```

Correction de la partie C (exercice 3) :

```
def dictionnaire_choisi (liste) :
    d = { }
    for mot in liste :
        for i,c in enumerate(mot) :
            d [i,c] = d.get ((i,c), []) + [ mot ]
    return d

def mots_lettre_position (d, lettre, position) :
    return d.get ( (position, lettre), [] )

d = dictionnaire_choisi(mots)
r = mots_lettre_position ( d, 'y', 1)
print (r)
```

S'il permet d'aller beaucoup plus vite pour effectuer une recherche, le dictionnaire `d` contient beaucoup plus de mots que la liste initiale. Si on suppose que tous les mots sont uniques, il en contient exactement autant que la somme des longueurs de chaque mot.

A quoi ça sert : tout dépend du nombre de fois qu'on n'effectue ce type de recherche. Si on décompose les deux méthodes en coût fixe (préparation du dictionnaire) et coût recherche, on obtient pour les deux méthodes :

méthode	coût fixe	coût variable
liste	0	$O(N)$
dictionaire	$O(Lln(26 * M))$	$O(ln(26 * M))$

Où :
– N est le nombre de mots,
– L est la somme des nombres de lettres de chaque mot,
– M est la longueur maximale d'un mot.
Si on effectue cette recherche un grand nombre de fois, l'utilisation d'un dictionnaire permet d'être beaucoup plus rapide même si on doit créer une structure intermédiaire. Ce schéma revient régulièrement : représenter autrement les données pour accélérer un traitement effectué un grand nombre de fois.

Code de César partie D (exercice 4) :

```
m = "JENESUISPASCODE"
s = "".join( [ chr((ord(l)-65+3)%26+65) for l in m ] )
print (s)
```

Solution pour le codage de Vigenère :

```
def code_vigenere ( message, cle) :
    message_code = ""
    for i,c in enumerate(message) :
        d = cle[ i % len(cle) ]
        d = ord(d) - 65
        message_code += chr((ord(c)-65+d)%26+65)
    return message_code

print ( code_vigenere ("JENESUISPASCODE", "DOP") )
```

Solution pour le décodage de Vigenère (exercice 6) :

```
def code_vigenere ( message, cle, decode = False) :      # ligne changée
    message_code = ""
    for i,c in enumerate(message) :
        d = cle[ i % len(cle) ]
        d = ord(d) - 65
        if decode : d = 26 - d                           # ligne ajoutée
        message_code += chr((ord(c)-65+d)%26+65)
    return message_code

print ( code_vigenere ("MSCHGJLGEDGRRRT", "DOP", True) )
```

fin correction TD ?? □