Série 2

Deuxième série : opérations sur les mots et les langages

1. Éléments nécessaires pour la réalisation des exercices

Afin de réaliser les différents activités de cette série de TP, il est indispensable de d'avoir :

— Connaissance du type `string` en Python.
  — Ne pas hésiter à consulter la documentation officielle
— Notions sur la récursivité
— Suivi le cours/TD du premier chapitre

Exercice 1

Écrire des les fonctions Python permettant de calculer :

Activité 1

La fonction `flatten_word(w_tuple)` transforme une liste de tuples représentant les symboles et leurs nombres en une chaîne de caractères. Par exemple, `flatten_word(("a",2),("b",4),("c",3))` renvoie la chaîne "aabbbbccc".

En-tête

```
def flatten_word(w_tuple)
```

Test

```
flatten_word(("a",2),("b",4),("a",2),("b",6),("c",1))
```

Activité 2

La fonction `unflatten_word(word)` effectue la transformation inverse de la fonction précédente. Par exemple, pour le mot "aabbbbccc", la fonction renvoie `("a",2),("b",4),("c",3)` .

En-tête

```
unflatten_word(word)
```

Test

```
unflatten_word("aabbbbccc")
```

Exercice 2

Écrire des les fonctions Python permettant de calculer :

Activité 1

La fonction `prefixes(word)` retourne la liste de tous les préfixes du mot *word*.

En-tête

```
def prefixes(word)
```

Test

```
prefixes("aaabbbccc")
```

Activité 2

La fonction `factors(word)` retourne la liste de tous les facteur du mot *word*.

En-tête

```
def factors(word)
```

Test

```
factors("aaabbbccc")
```

Activité 3

Une fonction permettant de calculer l'entrelacement d'un mot avec un symbole.

En-tête

```
def shuffle_symbol(word,c)
```

Test

```
shuffle_word("abba","c")
```

Activité 4

Une fonction permettant de calculer l'entrelacement de deux mots.

En-tête

```
def shuffle(w1,w2)
```

Test

```
shuffle_word("aaa","bbb")
```

Exercice 3

Pour chacun des langages suivants, écrire une fonction permettant d'accepter le langage. En d'autres termes, étant donné un mot du langage, la fonction doit décider si le mot appartient au langage ou non.

Activité 1

L_1 : les mots sur $\{a, b, c\}$ tel que le nombre de a est pair, le nombre de b est multiple de 3 et le nombre de c est multiple de 6. La lecture du mot se fait de gauche à droite uniquement (chaque symbole est lu une seule fois).

En-tête

```
def accept_l1(word)
```

Test

`accept_l1("ccabbaccbcc")` et `accept_l1("ccabbc")`

Activité 2

$L_2 = \{a^{2p}b^{3q}c^{2p+3q} | p, q \geqslant 0\}$. La lecture du mot se fait de gauche à droite uniquement (chaque symbole est lu une seule fois).

En-tête

```
def accept_l2(word)
```

Test

`accept_l2("aabbbccccccc")` et `accept_l2("aabbbcc")`

Activité 3

$L_3 = \{ww^R | w$ est un mot quelconque sur $\{"a", ..., "z"\}\}$. La lecture du mot se fait de gauche à droite uniquement (chaque symbole est lu une seule fois).

En-tête

```
def accept_l3(word)
```

Test

`accept_l3("abba")` et `accept_l3("aaabbcaa")`

Activité 4

$L_4 = \{wuw | w$ et u sont des mots quelconques sur $\{"a", ..., "z"\}\}$. w a une longueur non nulle.

En-tête

```
def accept_l4(word)
```

Test

`accept_l4("abcdabcd")` et `accept_l4("aababb")`

Activité 5

$L_5 = \{a^{2^n} | n \geqslant 0$. Il n'est pas autorisé de compter le nombre de a puis de le tester.

En-tête

```
def accept_l5(word)
```

Test

`accept_l5("aaaaaaaa")` et `accept_l5("aaaaaa")`

Activité 6

$L_6 = \{a^{n^2} | n \geqslant 0$. Il n'est pas autorisé de compter le nombre de a puis de le tester.

En-tête

```
def accept_l6(word)
```

Test

`accept_l6("aaaaaaaaa")` et `accept_l6("aaaaaa")`

Exercice 4

Soit la grammaire dont les règles de production sont données par : $S \rightarrow aSbS|\varepsilon$.

Activité 1

Écrire une fonction Python permettant de générer tous les mots dont la longueur est inférieure à une certaine limite.

En-tête

```
def generate(lim)
```

Test

```
generate(5)
```

Activité 2

Ecrire une deuxième fonction permettant de vérifier si un mot est généré par cette grammaire.

En-tête

```
def is_generated(w)
```

Test

```
is_generated("aabb") et is_generated("ababba")
```