TP 3 : Algorithmique du texte (suite)

***Remarques pratiques* :** tout d'abord, s'assurer que :
- les fonctions sont bien nommées exactement comme dans l'énoncé (donc `word_occurrences` avec deux `r`s, par exemple) ;
- les arguments des fonctions sont dans le même ordre que dans l'énoncé ;
- les fonctions renvoient bien une valeur avec `return` quand il faut retourner une valeur (pas juste un `print`).

Mettre les fonctions du TP2 et TP3 dans un même fichier nommé `tp23_prenom1_nom1__prenom2_nom2.py` (s'il y a des espaces dans le nom, remplacer par des underscores `_` aussi, et n'utiliser que des caractères ascii, i.e. pas d'accents, entre autres).

Mettre tous les tests effectués à la fin du fichier dans un bloc :

```
if __name__ == "__main__":
    # mettre ici tous les tests indentés
```

(autrement dit, pas d'appels de fonctions ou de `print` qui traînent en dehors de tout bloc).

Et tout ceci dans le but de faciliter le test automatique des fichiers et trouver rapidement ce qui va ou ne va pas.

Quelques conseils :
- tester sur quelques exemples chaque fonction (pas juste un exemple : chercher les cas limites) ;
- `Syntax error` signifie en python le plus souvent une parenthèse manquante à la ligne précédente à celle indiquée ;
- Une valeur de retour `None` signifie souvent qu'un `return` a été oublié.

1 Adaptation de Boyer-Moore : substitution de mot dans un texte

Adapter la fonction `BMsimple_with_table` du TP 2 en une fonction `BMsimple_with_table_gsub(word, text, replacement)`, qui renvoie le texte `text` dans lequel on a remplacé chaque occurrence de `word` par `replacement`.

On pourra utiliser une chaîne auxiliaire `new_text`, initialisée à `""` et que l'on agrandit avec l'opérateur `+=` au fur et à mesure.

Exemples :

```
print(BMsimple_with_table_gsub("Toto", "Toto, Toto, et Toto", "Titi"))
    # => "Titi, Titi, et Titi"
```

2 Un peu de récursivité

2.1 Recherche dichotomique

Écrire une fonction `recherche_dichotomique(tab, n)`, qui prend en entrée un tableau d'entiers trié par ordre croissant, et un entier `n`, et renvoie `True` si `n` est dans le tableau, et `False` sinon.

Il faut utiliser un algorithme dichotomique en $O(\log(\texttt{len(tab)}))$ qui utilise le fait que le tableau soit déjà trié.

Exemples :

```
print(recherche_dichotomique([1,3,4,6,9], 5)) # => False
print(recherche_dichotomique([1,3,4,6,9], 4)) # => True
```

2.2 Permutations

Écrire une fonction `permutations(word)`, qui prend en entrée un mot constitué de lettres distinctes et renvoie la liste des mots obtenus par toutes les permutations de ses lettres. On pourra utiliser une fonction auxiliaire récursive `permutations_aux(perm, word, perms)` qui construit la permutation courante dans `perm` et ajoute les permutations à un tableau `perms` (initialisé préalablement dans `permutations`).

Exemple :

```
print(permutations("abc"))
  # => ['abc', 'acb', 'bac', 'bca', 'cab', 'cba']
```