Niveau:AP1 **Durée : 1:30**	**Devoir surveillé n°2:Informatique**	**I.P.E.I.M.** **A.U.:2023/2024**

DOCUMENTS NON AUTORISÉS

Exercice 1 :

Prenez en compte les exceptions dans votre implémentation.

1. Écrivez une fonction nommée **`ligne_miroir`** qui prend un paramètre `nom_fichier`, inverse chaque ligne du fichier individuellement, enregistre ces nouvelles lignes inversées dans un nouveau fichier avec le suffixe "_mirror.txt" et retourne la liste des lignes du contenu.
2. Écrivez une fonction nommée **`inverser_lignes`** qui prend un paramètre `nom_fichier`, inverse l'ordre des lignes du fichier, puis remplace le contenu du fichier par la version inversée et retourne la liste des lignes du contenu.
3. En utilisant les fonctions ci-dessus, crivez une fonction nommée **`inverser`** qui prend un paramètre `nom_fichier`, inverse le contenu du fichier spécifié, remplace le contenu par la version inversée et retourne la liste des lignes du contenu.

Exercice 2 :

HTML (HyperText Markup Language) est le langage de base utilisé pour créer des pages web. Une balise en HTML est un élément de marquage encadré par des chevrons angulaires ('<' et '>'), utilisé pour définir la structure et le contenu d'une page web.

L'objectif de cet exercice est de comprendre les fondamentaux du langage HTML en se concentrant sur les éléments les plus simples en utilisant Python.

Nous allons nous intéresser aux éléments HTML les plus simples afin de simplifier les notions. Chaque élément HTML est défini par une balise ouvrante qui marque le début de l'élément, une balise fermante qui marque la fin de l'élément, et le contenu est placé entre ces balises.

À titre d'exemple :

Pour créer un paragraphe en HTML, on utilise la balise `<p>...</p>`. Les éléments HTML ont des composants clés :

- **Le nom de l'élément** : indique le type de balise utilisé.
- **La balise ouvrante** : marque le début de l'élément.
- **La balise fermante** : marque la fin de l'élément.
- **Le contenu** : le texte ou les autres éléments à l'intérieur des balises.
- **L'élément complet** : la balise ouvrante, le contenu et la balise fermante.

1. Écrivez une fonction `balise_ouverture` qui prend `txt` en argument. Elle renvoie une balise ouvrante pour ce `txt`, si le `txt` est alphanumérique et de moins de 10 caractères. Sinon, elle renvoie `<div>`.

On vous rappelle que la méthode **.isalnum()** permet de vérifier si une chaîne de caractères est composée uniquement de caractères alphanumériques (lettres et chiffres). Par exemple, **ch.isalnum()** renvoie **True** si tous les caractères de la chaîne **ch** sont alphanumériques, sinon elle renvoie **False**.

Exemple :

```
>>> balise_ouverture("titre")
"<titre>"
>>> balise_ouverture("balise_tres_longue")
'<div>'
```

1. De manière similaire, créez une fonction `balise_fermeture` qui prend un `txt` en paramètre et renvoie la balise HTML fermante correspondante, à condition que le `txt` respecte les mêmes critères de longueur et de composition alphanumérique.

   Exemple :

   ```
   >>> balise_fermeture("titre")
   "</titre>"
   >>> balise_fermeture("balise_tres_longue")
   '</div>'
   ```

2. Proposez une deuxième méthode pour la fonction `balise_fermeture` afin qu'elle utilise la fonction `balise_ouverture` pour créer la bonne balise fermante.

3. Écrivez la fonction `verif_ele` qui prend une chaîne de caractères en paramètre et renvoie `True` si la chaîne est valide en HTML, c'est-à-dire si elle s'ouvre par une balise et se ferme par la balise fermante correspondante. Sinon, elle doit renvoyer `False`.

   Exemple :

   ```
   >>> verif_ele('<p>Paragraphe valide</p>')
   True
   >>> verif_ele('<h1>Titre invalide</h2>')
   False
   ```

4. Écrivez une fonction **`html_en_liste`** qui prend un texte et renvoie une liste des textes valides à l'intérieur. Supposons que les éléments HTML sont continuellement adjacents, c'est-à-dire qu'il n'y a pas de caractères non imprimables entre les éléments HTML.

```
Indication:
   - Intercaler un marqueur '*' entre les elements HTML ch.replace('><','>*
<')
   - puis decouper selon ce marqueur
```

Exemple :

```
>>> html_en_liste('<h1>Titre 1</h1><p>Paragraphe 1</p><h2>Titre 2</h2>')
['<h1>Titre 1</h1>', '<p>Paragraphe 1</p>', '<h2>Titre 2</h2>']
```

6. Supposez qu'un texte ch valide puisse contenir plusieurs éléments HTML adjacents simples separees par un ou plusieurs caracteres Non imprimaples (espace ou tabulation ou retour de ligne)

Écrivez une fonction récursive **`html_to_list_general`** qui prend en argument une chaîne `ch` selon le principe suivant :

Écrivez une fonction récursive html_to_list_general qui prend en argument une chiane ch et renvoie une liste des elements valides selon le principe suivant :

- **Étape 1** : Initialisez une liste `L0` vide (resultat de la fonction).
- **Étape 2** : Vérifiez si la chaîne est de type `verif_ele` et contient exactement 1 élément HTML (c'est-à-dire que la chaîne ne contient plus ni '><' ni le caractère '>' suivi par 1 ou plusieurs espaces). Si oui, ajoutez `ch` à `L0`.
- **Étape 3** : Si `ch` contient plus d'un élément HTML :
  - Localisez la première balise ouvrante et, par conséquent, déduisez la balise fermante.
  - Divisez la chaîne selon la balise fermante. Vous obtenez une liste découpée, appelée `ch_decoupee`.
  - Utilisez `ch_decoupee` pour reconstituer le premier élément HTML et le garder dans la liste `L0`.
  - Utilisez `ch_decoupee` pour reconstituer le reste de la chaîne (du deuxième élément jusqu'à la fin).
- **Étape 4** : De façon récursive, refaites les mêmes étapes pour le reste de la chaîne (après le découpage) jusqu'au dernier élément.

7. Écrivez une fonction **`html_to_dict`** qui accepte comme paramètre `ch` contenant plusieurs

éléments adjacents simples et renvoie un dictionnaire dont les clés sont les tags et les valeurs sont des listes de ses éléments HTML.

Exemple :

```
>>> ch='<h1>Titre 1</h1><h2>Titre 2</h2><h1>Titre 3</h1>'
>>> html_to_dict(ch)
```

```
{'h1': ['<h1>Titre 1</h1>', '<h1>Titre 3</h1>'], 'h2': ['<h2>Titre
2</h2>']}
```