EXERCICE 3 (4 points)

Cet exercice porte sur la programmation en Python, la manipulation des chaînes de caractères, les arbres binaires de recherche et le parcours de liste.

1. On rappelle ici quelques notions sur la manipulation des chaînes de caractères en Python.

Une chaîne de caractères se comporte comme un tableau de caractères que l'on ne peut pas modifier.

Par exemple, on a le comportement suivant :

```
>>> une_chaine = 'Bonjour'
>>> une_chaine[3]
'j'
>>> une_chaine[3] = 'z'
 TypeError: 'str' object does not support item assignment
```

On peut aussi utiliser l'opérateur de concaténation +.

```
>>> une_chaine = 'a' + 'b'
>>> une_chaine
'ab'
>>> une_chaine = une_chaine + 'c'
>>> une_chaine
'abc'
```

On définit la fonction `bonjour` par le code suivant :

```
1  def bonjour(nom)
2      return 'Bonjour ' + nom + ' !'
```

a. Donner le résultat de l'exécution de `bonjour('Alan')`.

On exécute le programme suivant :

```
une_chaine='Bonjour'
x = (une_chaine[2] == une_chaine[3])
y = (une_chaine[4] == une_chaine[1])
```

b. Donner le type et les valeurs des variables `x` et `y`.

c. Écrire une fonction `occurrences_lettre(une_chaine,une_lettre)` prenant en paramètres une chaîne `une_chaine` et une lettre `une_lettre` et renvoyant le nombre d'occurrences de `une_lettre` dans `une_chaine`.

2. On rappelle qu'un arbre binaire de recherche est un arbre binaire pour lequel chaque nœud possède une étiquette dont la valeur est supérieure ou égale à toutes les étiquettes des nœuds de son fils gauche et strictement inférieure à celles des nœuds de son fils droit.
Sa taille est son nombre de nœuds ; sa hauteur est le nombre de niveaux qu'il contient.
On rappelle aussi que l'on peut comparer des chaînes de caractères en utilisant l'ordre alphabétique. On a par exemple :

```
>>> 'ab' < 'aa'
False
>>> 'abc' < 'acb'
True
```

On considère la liste de mots `animaux = ['python', 'chameau', 'pingouin', 'renard', 'gnou']`

a. Dessiner un arbre binaire de recherche contenant tous les mots de la liste `animaux` et de hauteur minimale.

b. Dessiner un arbre binaire de recherche contenant tous ces mots et de hauteur maximale.

3. On considère l'implémentation objet suivante d'un arbre binaire de recherche :
On dispose d'une classe `Abr` contenant notamment les méthodes et attributs suivants :

- Si `un_abr` est une instance d'`Abr` alors `un_abr.est_vide()` renvoie `True` si l'arbre est vide et `False` sinon.
- Si `un_abr` est une instance d'`Abr` et si `un_abr.est_vide()` renvoie `False`, alors `un_abr.valeur` contient une chaîne de caractères représentant la valeur de la racine de l'arbre.
- Si `un_abr` est une instance d'`Abr` et si `un_abr.est_vide()` renvoie `False`, alors `un_abr.sous_arbre_gauche` et `un_abr.sous_arbre_droit` contiennent chacun une instance d'`Abr`.

On considère que la variable `liste_mots_francais` est une liste de 336531 mots en français et que la variable `abr_mots_francais` est une instance d'`Abr` contenant les mots de la liste.

On considère la fonction suivante :

```
1  def mystere(un_abr):
2      if un_abr.est_vide():
3          return 0
4      else:
5          return 1 + mystere(un_abr.sous_arbre_gauche) \
6              + mystere(un_abr.sous_arbre_droit)
```

<u>Remarque</u> : on rappelle que le caractère \ en fin de ligne 5 indique que l'instruction se poursuit sur la ligne suivante.

a. Donner le résultat de `mystere(abr_mots_francais)`, en justifiant le résultat.

On veut calculer la hauteur de `abr_mots_francais`.

b. Donner le code d'une fonction `hauteur(un_abr)` permettant de faire ce calcul, en vous inspirant du code précédent.

4. Dans cette question, nous nous servirons uniquement de `liste_mots_francais` et plus de `abr_mots_francais`.

Pour aider à la résolution de mots croisés, on a décidé d'écrire une fonction `chercher_mots(liste_mots, longueur, lettre, position)` où `liste_mots` est une liste de mots français, `longueur` est la taille du mot recherché, `lettre` est une lettre du mot se trouvant à l'indice `position`.

Par exemple `chercher_mots(liste_mots_francais,3,'x',2)` renverra `['aux', 'box', 'dix', 'eux', 'fax', 'fox', 'lux', 'max', 'six']`.

a. Recopier et compléter la ligne 4 de la fonction ci-dessous :

```
1  def chercher_mots(liste_mots,longueur,lettre,position):
2      res = []
3      for i in range(len(liste_mots)):
4          if ..... and ..... :
5              res.append(liste_mots[i])
6      return res
```

b. Expliquer ce que donne la commande suivante.

```
>>> chercher_mots(chercher_mots(liste_mots_francais,3,'x',2),3,'a',1)
```

On cherche un mot de 5 lettres dont on connait la fin `'ter'`.

c. Ecrire la commande permettant de chercher les mots candidats dans `liste_mots_francais`.