Corrigé sujet 17 - Année : 2022

Sujet 17 - 20222

1. Exercice 1

Script Python

```python
def nombre_de_mots(phrase):
    nb_espace = 0
    for caractere in phrase:
        if caractere==" ":
            nb_espace+=1
    if phrase[-1]==".":
        return nb_espace+1
    else:
        return nb_espace
```

Commentaire

1. Il faut avoir remarqué que le nombre de mots est égal:
   - au nombre d'espace si la phrase se termine par "!" ou "?",
   - au nombre d'espace plus un si la phrase se termine par un ".".
2. Les exemples de l'énoncé ne testent que la fin avec un point d'exclamation.
3. Dans le corrigé, on a compté le nombre d'espace en effectuant un parcours de la phrase, on pouvait aussi utiliser la méthode `count` des chaines de caractères.

2. Exercice 2

Script Python

```python
class Noeud:
    '''
    Classe implémentant un noeud d'arbre binaire
    disposant de 3 attributs :
    - valeur : la valeur de l'étiquette,
    - gauche : le sous-arbre gauche.
    - droit : le sous-arbre droit.
    '''
    def __init__(self, v, g, d):
        self.valeur = v
```

```
        self.gauche = g
        self.droite = d

class ABR:
    '''
    Classe implémentant une structure
    d'arbre binaire de recherche.
    '''

    def __init__(self):
        '''Crée un arbre binaire de recherche vide'''
        self.racine = None

    def est_vide(self):
        '''Renvoie True si l'ABR est vide et False sinon.'''
        return self.racine is None

    def parcours(self, tab = []):
        '''
      Renvoie la liste tab complétée avec tous les
        éléments de
        l'ABR triés par ordre croissant.
        '''
        if self.est_vide():
            return tab
        else:
            self.racine.gauche.parcours(tab)
            tab.append(self.racine.valeur) #(1)
            self.racine.droite.parcours(tab)
            return tab

    def insere(self, element):
        '''Insère un élément dans l'arbre binaire de recherche.'''
        if self.est_vide():
            self.racine = Noeud(element, ABR(), ABR())
        else:
            if element < self.racine.valeur:
                self.racine.gauche.insere(element)
            else :
                self.racine.droite.insere(element)

    def recherche(self, element):
        '''
        Renvoie True si element est présent dans l'arbre
        binaire et False sinon.
     '''
        if self.est_vide():
            return False #(2)
        else:
            if element < self.racine.valeur:
                return self.racine.gauche.recherche(element) #(3)
            elif element > self.racine.valeur:
                return self.racine.droite.recherche(element)
            else:
                return True
```

1. On parcours à gauche, on ajoute la valeur de la racine puis on parcourt à droite.
2. Si l'arbre est vide alors l'élément ne s'y trouve pas !

3. Si l'arbre n'est pas vide, on compare avec la valeur de la racine. Si ce n'est pas la valeur cherchée on recherche à droite ou à gauche suivant les cas.

Attention

- Le code fourni utilise un objet mutable (une liste) comme paramètre par défaut de la méthode de parcours :

  Script Python

  ```
  def parcours(self, tab = []):
  ```

  C'est une très mauvaise pratique car source d'erreurs, en effet la variable `tab` étant mutable elle est modifiée par la fonction lors d'un premier appel et ne sera donc plus vide lors des appels suivants. Pour une solution à ce problème, on pourra par exemple consulter ce site