

# Corrigé sujet 17 - Année : 2022

[Sujet 17 - 2022](#) ↴

## 1. Exercice 1



```

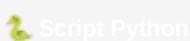
1 def nombre_de_mots(phrase):
2     nb_espace = 0
3     for caractere in phrase:
4         if caractere==" ":
5             nb_espace+=1
6         if phrase[-1]==".":
7             return nb_espace+1
8         else:
9             return nb_espace

```

### 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 chaînes de caractères.

## 2. Exercice 2



```

1 class Noeud:
2     """
3         Classe implémentant un noeud d'arbre binaire
4         disposant de 3 attributs :
5             - valeur : la valeur de l'étiquette,
6             - gauche : le sous-arbre gauche.
7             - droit : le sous-arbre droit.
8     """
9     def __init__(self, v, g, d):
10         self.valeur = v

```

```

11         self.gauche = g
12         self.droite = d
13
14 class ABR:
15     """
16         Classe implémentant une structure
17         d'arbre binaire de recherche.
18     """
19
20     def __init__(self):
21         '''Crée un arbre binaire de recherche vide'''
22         self.racine = None
23
24     def est_vide(self):
25         '''Renvoie True si l'ABR est vide et False sinon.'''
26         return self.racine is None
27
28     def parcours(self, tab = []):
29         """
30             Renvoie la liste tab complétée avec tous les
31             éléments de
32             l'ABR triés par ordre croissant.
33         """
34         if self.est_vide():
35             return tab
36         else:
37             self.racine.gauche.parcours(tab)
38             tab.append(self.racine.valeur) #(1)
39             self.racine.droite.parcours(tab)
40         return tab
41
42     def insere(self, element):
43         '''Insère un élément dans l'arbre binaire de recherche.'''
44         if self.est_vide():
45             self.racine = Noeud(element, ABR(), ABR())
46         else:
47             if element < self.racine.valeur:
48                 self.racine.gauche.insere(element)
49             else :
50                 self.racine.droite.insere(element)
51
52     def recherche(self, element):
53         """
54             Renvoie True si element est présent dans l'arbre
55             binaire et False sinon.
56         """
57         if self.est_vide():
58             return False #(2)
59         else:
60             if element < self.racine.valeur:
61                 return self.racine.gauche.recherche(element) #(3)
62             elif element > self.racine.valeur:
63                 return self.racine.droite.recherche(element)
64             else:
65                 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](#)