```
# -*- coding: utf-8 -*-
"""
td 3_1 dictionnaire : algorithmes de recherche d'un élément
"""

lettre1 = 'B'
chaine1 = "INFORMATIQUE"

# Q1) lister dans un dictionnaire, les lettres présentes dans chaine : fonction lettres()
def lettres(chaine):
        d = {}              # on crée un dictionnaire d vide
        n= len(chaine)      # n est le nombre d'éléments dans chaine
        for i in range(n):  # on balaye tous les éléments de chaine
                if (chaine[ i ] in d)==False:   # si le ie élément de chaine n'est pas dans d
                        d[chaine[i]]=True  # alors on l'ajoute
        return d
# Q2) utilisation de lettres()
d1 =  lettres(chaine1)

# Q3) complexité de lettres()
"""
La complexité de lettres() est n car la boucle for réalise n itérations.
La durée d'exécution sera proportionnel à n.
"""

# Q4) recherche d'une lettre dans une chaine
try:
    print("Présence de la lettre",lettre1,"dans", chaine1,":", d1[lettre1],".")
except:
    print("Présence de la lettre",lettre1,"dans", chaine1,": False.")  # d1[lettre1] renvoie
une erreur si lettre1 n'est pas dans d1.


# Q5) complexité de l'algorithme de la question Q4)
"""
L'extraction d'une valeur dans un dictionnaire correspond à une opération donc à une
complexité unitaire en O(1). Cette complexité est donc indépendante de la taille n de la
liste.
(La complexité a été reporté à l'établissement du dictionnaire)
"""

# Q6) fonctions mots()
def mots(phrase):
        d = {}              # on crée un dictionnaire d vide
        n= len(phrase)      # n est le nombre d'éléments dans chaine
        for i in range(n):  # on balaye tous les éléments de chaine
                if (phrase[ i ] in d)==False:   # si le ie élément de chaine n'est pas dans d
                        d[phrase[i]]=True  # alors on l'ajoute
        return d

"""
Remarque : c'est le même algorithme qu'à la question Q1), seuls les noms des variables ont
changé.
"""

# Q7) d2 = mots()
```

```
mot2 = 'LE'
phrase2 = ['UN', 'DICTIONNAIRE', 'CONTIENT', 'DES', 'COUPLES', 'CONSTITUES', 'D', 'UNE',
'CLE', 'ET', 'DE','SA','VALEUR']

d2 = mots(phrase2)

try :
    print( "Présence de",mot2,"dans", phrase2, ":" , d2[mot2] )
except :
    print("Présence de la lettre",mot2,"dans", phrase2,": False.")  # d2[mot2] renvoie une
erreur si lettre1 n'est pas dans d1.

# Q8) complexité de mots
"""
Même réponse qu'à la question Q5) puisque c'est le même algorithme.
"""


# Q9) chercher un mot dans une chaine.
mot3 = ' DE '          # espace avant et après pour ne pas trouver un morceau de mot
phrase3 = "UN DICTIONNAIRE CONTIENT DES COUPLES CONSTITUES D UNE CLE ET DE SA VALEUR"

def chercher(x,L):
    nx = len(x)
    s = False   # on suppose par défaut que x n'est pas dans L
    n= len(L)   # n est le nombre d'éléments dans L
    i = 0               # 0 est le 1er indice de L (variant de boucle à 0)
    while (s==False and i < n-nx) :     # test  "x non trouvé" et "dernier indice de L non
atteint"
            if L[ i:i+nx ]==x:
                    s = True
            i =i +1                     # incrément du variant de boucle
    return s

print( "Présence de",mot3,"dans", phrase3, ":" , chercher(mot2,phrase3) )

# Q10) Complexité de chercher()
"""
La complexité de chercher() (dans le pire des cas, où x n'est pas présent dans L) est n-nx
--> O(n).
La durée d'exécution de chercher() va augmenter proportionnellement au nombre de caractères
de L
"""


# Q11) Comparaison des algorithmes
"""
La 2e méthodes est plus efficace pour une seule recherche car avec la première méthode, il
faudra également faire un premier traitement de complexité O(n) pour découper chaque mot par
les caractères qui ne sont pas des lettres.
Avec la 1ère méthode, on aura la possibilité de d'accéder à d'autres requêtes en 1 seul
extraction.
"""
```