```
## 1 Recherche dans une chaîne de caractères
#1.1
#1.1.1
fichier=open("le_tour_du_monde_en_80_jours_utf8.txt",'r',encoding='utf8')
texte=fichier.read()#on crée une unique chaîne de caractères
fichier.close()
essai='Physique et cie' #texte d'essai pour ne pas lancer
#un algorithme défaillant sur une grande chaine de caractère

def rech_lettre(lettre,chaine):
    """La fonction renvoie le nombre d'occurrences du caractère lettre dans
    la chaîne de caractère chaine. Elle teste lettre par lettre en balayant
    la chaîne, à la manière d'une liste"""
    compteur=0
    for k in range(len(chaine)):
        if chaine[k]==lettre:
            compteur+=1
    return compteur

##print(rech_lettre('e',essai)) #renvoie 3
##print(rech_lettre('y',texte)) #renvoie 968

#1.1.1
def rech_lettre2(lettre,chaine):
    """La fonction renvoie le nombre d'occurrences et la liste des positions
    du caractère lettre dans la chaîne de caractère chaine, en tuple.
    Elle teste lettre par lettre en balayant la chaîne, à la manière d'une liste"""
    compteur=0
    listepos=[]
    for k in range(len(chaine)):
        if chaine[k]==lettre:
            compteur+=1
            listepos.append(k)
    return (compteur,listepos)#tuple demandé

##print(rech_lettre2('e',essai)) #renvoie 3
##print(rech_lettre2('y',texte)) #renvoie 968


#1.2
#1.2.1
def rech_motif(motif,chaine):
    """La fonction renvoie le nombre d'occurrences d'un motif dans la chaîne de
    caractère chaine : le principe et de balayer la chaine, et tester si les lettres
    correspondent. L'usage de while est préconisé, pour ne pas balayer inutilement
    la chaine de la longueur du motif si une lettre ne correspond pas."""
    compteur=0
    for i in range(len(chaine)-len(motif) + 1):
        j=0
        while (j<len(motif)) and (motif[j]==chaine[i+j]):
            j+=1
        if j==len(motif):
            compteur+=1
    return compteur

##print(rech_motif('cie',essai))

#1.2.2

import time
##debut=time.time()
##print(rech_motif('ent',texte))
##fin=time.time()
##print(fin-debut)


##2 Recherche dans une liste
#2.1 Recherche d'un maximum

#2.1.1

##fichier2=open("signal_audio.txt",'r')
##signal=[]
##for ligne in fichier2:
```

```
##    signal.append(int(ligne)) #on n'oublie pas de convertir en entier, car sinon chaîne de caractère
##
##print(signal[3]) #pour tester le fonctionnement, on ne print pas toute la liste...

#2.1.2

def val_max(liste):
    maxi=liste[0] #on initialise avec la première valeur de la liste
    for nombre in liste:
        if nombre>maxi:
            maxi=nombre
    return maxi
##maxsignal=val_max(signal)
##print(maxsignal)

#2.1.3

import matplotlib.pyplot as plt

abs=[x for x in range(10000)] #pour vous rappeler l'existence des listes en compréhension
##plt.plot(abs,signal[0:10000]) #slice de liste)
##plt.axhline(maxsignal,color='r')
##plt.show()

#2.1.4
def moyenne(liste):
    somme=0
    for k in liste:
        somme+=k
    return somme/len(liste)

##moysignal=moyenne(signal)
##print(moysignal)

##plt.plot(abs,signal[0:10000]) #slice de liste)
##plt.axhline(maxsignal,color='r')
##plt.axhline(moysignal,color='b',linestyle='--')
##plt.show()


#2.2.2
def affiche_ip(n):
    n1=n%1000 ; n=n//1000 ; n2 = n%1000 ; n=n//1000
    n3=n%1000; n4 = n//1000
    ip = "{0:}.{1:03}.{2:03}.{3:03}".format(n4,n3,n2,n1)
    return ip

print(affiche_ip(192168001001))

#2.2.2.1
def lecture_fichier(nom):
    """ Fonction qui renvoie une liste issue des lignes du fichier dont le nom est fourni"""
    fichier=open(nom,'r')
    liste=[]
    for ligne in fichier:
        liste.append(int(ligne)) #on n'oublie pas de convertir en entier, car sinon chaîne de caractère
    return liste

##adresses=lecture_fichier('unsorted_IP_adresses_list.txt')
##print(adresses[2])#test

#2.2.2.2 et 3
def lecture_fichier2(nom):
    """ Fonction qui renvoie une liste triée issue des lignes du fichier dont le nom est fourni"""
    fichier=open(nom,'r')
    liste=[]
    for ligne in fichier:
        liste.append(int(ligne)) #on n'oublie pas de convertir en entier, car sinon chaîne de caractère
    liste.sort()
    return liste

adressestriees=lecture_fichier2('unsorted_IP_adresses_list.txt')
print(affiche_ip(adressestriees[234567]))
```

```python
#2.2.2.4 5 et 6
def rech_dicho(liste_triee,cible):
    """ Recherche par division d'intervalle, en utilisant l'invariant que
    liste_triee[g]<=cible<liste_triee[d] à chaque division. A la fin de la boucle
    il reste deux éléments : celui de gauche est cible, sauf s'il n'est pas présent
    dans liste_triee. On renvoit donc l'élément cible le plus à droite de la liste"""
    #initialisation
    g,d=0,len(liste_triee)-1
    #robustesse : vérifie l'invariant en début de boucle
    if cible>liste_triee[d] or cible<liste_triee[g]:
        return None
    if cible==liste_triee[d]:
        return d
    while g<d-1:
        m=(g+d)//2
        if liste_triee[m]>cible:
            d=m
        else:
            g=m
    if liste_triee[g] == cible:
        return g
    else:
        return None

##print(rech_dicho(adressestriees,78192004194))
##print(rech_dicho(adressestriees,127198224147))

#2.2.2.7
def rech_dicho(liste_triee,cible):
    """ Recherche par division d'intervalle, en utilisant l'invariant que
    liste_triee[g]<=cible<liste_triee[d] à chaque division. A la fin de la boucle
    il reste deux éléments : celui de gauche est cible, sauf s'il n'est pas présent
    dans liste_triee. On renvoit donc l'élément cible le plus à droite de la liste"""
    #initialisation
    g,d=0,len(liste_triee)-1
    #robustesse : vérifie l'invariant en début de boucle
    if cible>liste_triee[d] or cible<liste_triee[g]:
        return -1
    if cible==liste_triee[d]:
        return d
    while g<d-1:
        m=(g+d)//2
        if liste_triee[m]>cible:
            d=m
        else:
            g=m
    if liste_triee[g] == cible:
        return g
    else:
        return -1

##print(rech_dicho(adressestriees,236018142100))
##print(rech_dicho(adressestriees,96151048114))


#2.2.2.9

def rech_naive(liste,cible):
    for k in range(len(liste)):
        if liste[k]==cible:
            return k

debut=time.time()
print(rech_naive(adressestriees,236018142100))
fin=time.time()
print(fin-debut)

debut=time.time()
print(rech_dicho(adressestriees,236018142100))
fin=time.time()
print(fin-debut)#affiche 0, trop rapide :D
```