

RECHERCHE D'UN SOUS-MOT

I. Chaînes de caractères : rappels et compléments

Une chaîne de caractères est une suite de caractères consécutifs notée entre apostrophes (ou entre guillemets) :

```
>>> s = 'anticonstitutionnellement'
>>> type(s)
<class 'str'>
>>> print(s)
anticonstitutionnellement
>>> len(s) #Longueur de la chaîne de caractères
25
>>> s[0],s[1],s[24] #Lettres indexées à partir de 0
('a', 'n', 't')
>>> s[25]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> 'bon' + 'jour' #Concaténation de deux chaînes de caractères
'bonjour'
```

⚠ ATTENTION ! Les chaînes de caractères ne sont pas des objets mutables : on ne peut pas modifier les caractères à l'intérieur d'une chaîne de caractères.

Plus généralement, les chaînes de caractères ne sont pas des listes.

```
>>> s[0] = 'A'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

EXERCICE 1

Écrire une fonction `nombre_a` qui prend en argument une chaîne de caractères et qui renvoie le nombre de `a` que contient cette chaîne.

EXERCICE 2

Écrire une fonction `est_palindrome` qui teste si la chaîne de caractères passée en argument est un palindrome.

EXERCICE 3

Écrire une fonction `miroir` qui prend en argument une chaîne de caractères et qui renvoie son miroir (par exemple, le miroir de "miroir" et "riorim").

II. Recherche d'un mot dans un texte

On souhaite écrire un algorithme permettant de tester si une chaîne de caractères est un sous-mot d'une autre chaîne de caractères, par exemple si "ATGCCTG" est dans "TTCGGAGCGTAAGTATGCCTGTAAT".

En Python :

```
def sous_mot(m,t):
    for i in range(len(t)-len(m)+1):
        j = 0
        while j < len(m) and m[j] == t[i+j]:
            j = j + 1
        if j == len(m):
            return True
    return False
```

Étudions la complexité de cette fonction. Notons L la longueur de la chaîne de caractères $\mathfrak{t}$ et ℓ la longueur de la chaîne de caractères $\mathfrak{m}$.

- Le nombre d'itérations de la boucle `while` est majoré par ℓ .
- Le nombre d'itérations de la boucle `for` est majoré par $1 + L - \ell$

Par conséquent, la complexité est en $O(L\ell)$. Cette borne est atteinte, par exemple lorsqu'on recherche "aa...aab" dans "aaaa...aaaa".