

# CHAÎNES DE CARACTÈRES

## 1. Notion de chaîne de caractère

### 1.1 Définition

**Définition.** Le type chaînes de caractères, `string` en anglais (et dans Python), désigne une suite finie de caractères élémentaires.

C'est celui qui permet de représenter des textes.

*Remarque.* Un caractère peut être toute lettre de l'alphabet mais aussi un signe de ponctuation, une espace voire d'autres symboles : `a,z; *' ?#`

### 1.2 Construction

Une chaîne de caractères se note généralement en écrivant directement la suite des caractères voulus et en entourant cette suite par des délimiteurs.

En Python, il y a plusieurs sortes de délimiteurs :

**Guillemets.** Elles autorisent des chaînes de caractères comportant des apostrophes.

```
"bonjour je m'appelle... "
```

**Apostrophes.** Elles autorisent des chaînes de caractères comportant des guillemets.

**Triples apostrophes ou triples guillemets.** Certes plus lourdes, elles ont l'avantage d'autoriser des chaînes de caractères sur plusieurs lignes :

```
"""Bonjour  
à tous"""
```

*Remarque.* La chaîne vide peut s'obtenir par `' '` ou `""`

## 2. Opérations sur les chaînes de caractères

### 2.1 Longueur

La longueur d'une chaîne de caractères s'obtient par la commande `len` :

```
>>> s="Bonjour "  
>>> len(s)  
7
```

### 2.2 Accès à un caractère

On peut accéder directement à un caractère en fonction de sa position avec les mêmes principes que pour les listes.

```
>>> s="Bonjour "  
>>> s[0]  
'B'  
>>> s[2]  
'n'  
>>> s[-1]  
'r'
```

*Remarque* (caractère non mutable). Les chaînes sont des objets non mutables.

```
>>> s[0]='A'
TypeError: 'str' object does not support item assignment
```

## 2.3 Concaténation

La concaténation est le procédé consistant à créer une nouvelle chaîne de caractères en juxtaposant deux chaînes données.

Elle s'effectue avec l'opérateur +.

```
>>> s="Bonjour "
>>> t=" à tous !"
>>> s+t
'Bonjour à tous !'
```

## 2.4 Conversions

### Variable vers chaîne

La commande `str` permet de convertir une variable en chaîne de caractères.

```
>>> a = 3
>>> type(a)
<class 'int'>
>>> b=str(a)
>>> b
'3'
>>> type(b)
<class 'str'>
```

*Remarque.* Si elle a un sens, la conversion d'une chaîne de caractères en un entier se fait avec la commande `int` et la conversion en un flottant avec `float`.

Voici un exemple de conversion possible.

```
>>> a = '785'
>>> type(a)
<class 'str'>
>>> b=int(a)
>>> type(b)
<class 'int'>
```

Voici une conversion impossible.

```
>>> a='huit'
>>> type(a)
<class 'str'>
>>> float(a)
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    float(a)
ValueError: could not convert string to float: 'huit'
```

## Chaîne vers liste

On peut convertir une chaîne de caractère en une **liste** de chaînes de caractères au moyen de la méthode `split()`

Le paramètre entré dans les parenthèses sert de séparateur.

On peut par exemple séparer la chaîne de caractères `s1` ci-dessous en coupant au niveau des virgules (caractère `,`) :

```
>>> s1 = "Nom, Prénom, Age"
>>> s1.split(',') # séparateur précisé en paramètre
['Nom', ' Prénom', ' Age']
# les espaces sont conservées ...
```

On peut par exemple séparer la chaîne de caractères `s2` ci-dessous en coupant au niveau des espaces (caractère `' '`) :

```
>>> s2 = "Bonjour les amis"
>>> s2.split(' ') # séparateur précisé en paramètre
['Bonjour', 'les', 'amis']
# On a récupéré les mots dans une liste .
```

*Remarque.* La commande `list` convertit, quand c'est possible, une variable en liste. Appliquée à une chaîne de caractères, elle fabrique la liste de **tous** les caractères de la chaîne.

```
>>> list("conversion !")
['c', 'o', 'n', 'v', 'e', 'r', 's', 'i', 'o', 'n', ' ', '!', '✓']
```

## 2.5 Caractères spéciaux

Certains caractères, dits *spéciaux*, peuvent être dissimulés dans un texte.

Deux fréquents sont

- `\n` pour le saut de ligne
- `\t` pour la tabulation (pour aligner verticalement)

Même si c'est rarement essentiel, vous pouvez les utiliser dans vos chaînes de caractères.

*Exemple.*

```
>>> s = "X\tY\n0.1\t3.1\n0.2\t4.2"
>>> print(s)
X      Y
0.1    3.1
0.2    4.2
```

## 2.6 Nettoyage de chaîne

Il est possible d'éliminer automatiquement les espaces classiques, les sauts de ligne et les tabulations *aux extrémités* d'une chaîne de caractères à l'aide de la méthode `strip()`.

```
>>> ligne = " trop d'espace au début et un saut de \n
ligne ici\n"
>>> print(ligne)
 trop d'espace au début et un saut de ligne à la fin

>>> print(ligne.strip())
trop d'espace au début et un saut de ligne à la fin
>>>
```

Cette méthode pourra être utilisée quand les chaînes de caractères seront extraites d'un fichier auxiliaire (voir le chapitre sur les manipulations de fichiers).

### 3. Sous-chaînes de caractère

#### 3.1 Généralités

**Définition** (sous-chaîne). Une sous-chaîne d'une chaîne de caractères  $s$  est une suite de caractères consécutifs de la chaîne  $s$ .

*Exemple.* Si  $s$  est la chaîne "Bonjour à tous" alors "Bonjour" ou encore "jour à t" en sont des sous-chaînes.

On peut extraire une sous-chaîne de la même manière qu'on peut extraire une sous-liste (attention aux principes de numérotation).

```
>>> s = "Bonjour à tous !"
>>> s[0:7]
'Bonjour'
>>> s[8:]
'à tous !'
```

#### 3.2 Recherche d'un mot

On cherche à déterminer si une chaîne de caractères  $s$  contient une sous-chaîne  $m$  donnée (aussi appelée « mot »).

##### Commande Python

L'appartenance d'un mot  $m$  à une chaîne  $s$  peut se tester au moyen de l'opérateur `in`.

```
>>> s="Bonjour à tous !"
>>> 'jou' in s
True
>>> 'jourà' in s
False
```

##### L'algorithme naïf

Même si une commande existe en Python, il est très instructif (et au programme) de chercher à écrire un algorithme effectuant la recherche d'un mot  $m$  dans une chaîne  $s$ .

Dans une version basique, on peut se contenter de retourner un booléen indiquant si la chaîne  $s$  contient le mot  $m$  ou non.

Nous définissons tout d'abord une fonction vérifiant si un mot  $m$  est présent à l'indice  $i$  de la chaîne  $s$ .

```
def sous_mot(m,s,i):
    """ Répond à la question : le mot m est-il
        présent dans s à l'indice i ? """
    j = 0
    long_m = len(m)
    long_s = len(s)
    while j < long_m and i + j < long_s and m[j] == s[i+j]:
        j += 1
    # on teste une à une la concordance des lettres
```

```
        j = j + 1
    return j == long_m
    # en sortie de boucle , ou bien j = long_m
    # càd toutes les lettres correspondent
    # ou bien m[j] est différent de s[i+j]
    # càd une lettre ne correspond pas.
```

Voici ensuite la fonction de recherche proprement dite :

```
def recherche_mot(m,s):
    """ Répond à la question : le mot m est-il
        un sous-mot de s ? """
    long_m = len(m)
    long_s = len(s)
    i = 0
    while i <= long_s - long_m and not(sous_mot(m,s,i)):
        i = i + 1
        # en sortie de boucle ,
        # i > long_s - long_m : on n'a pas trouvé m
        # ou bien
        # i <= long_s - long_m : sous_mot a retourné True
    return i <= long_s - long_m
```