

Chapitre 4

Chaînes de caractères et autres séquences

1 Opérations communes

Les types `int`, `float`, `bool` sont des *types scalaires*.

Les types `list` (listes) et `str` (chaînes de caractère) sont des structures de données de *type séquentielles*.

Tous les objets de type séquentiel ont en commun :

Opération	Résultat
<code>s[i]</code>	élément d'indice <code>i</code> de <code>s</code>
<code>s[i:j]</code>	Extraction de <code>i</code> (inclus) à <code>j</code> (exclus)
<code>s[i:j:k]</code>	Extraction de <code>i</code> à <code>j</code> par pas de <code>k</code>
<code>len(s)</code>	Longueur de <code>s</code>
<code>x in s</code>	<code>True</code> si <code>x</code> est dans <code>s</code> , <code>False</code> sinon
<code>x not in s</code>	<code>True</code> si <code>x</code> n'est pas dans <code>s</code> , <code>False</code> sinon
<code>s+t</code>	Concaténation de <code>s</code> et <code>t</code>
<code>s*n, n*s</code>	Concaténation de <code>n</code> copies de <code>s</code>
<code>s.index(x)</code>	Indice de la 1 ^{ère} occurrence de <code>x</code> dans <code>s</code>

où `s` et `t` sont des objets séquentiels de même type, et `i`, `j`, `k`, `n` sont des entiers.

1.1 Exemples : concaténation et duplication

```
In [1]: L = [1,2,3] + [4,5]
In [2]: print(L)
[1,2,3,4,5]
In [3]: L * 2
[1,2,3,4,5,1,2,3,4,5]
```

- Pour une liste `L`, un élément `x` et une liste `L2` :
`L.append(x)` a même effet que `L += [x]`
`L.extend(L2)` a même effet que `L += L2`

- concaténation et duplication marchent aussi avec une chaîne de caractère :

```
In [4]: ch = 'To' + 'to' ; print(ch)
Toto
In [5]: print(ch*3)
TotoTotoToto
```

1.2 les chaînes de caractères

Les objets de type `str`, chaînes de caractère, sont des structures de données séquentielles :

```
In [1]: chaine = 'Amanda'
In [2]: len(chaine)
In [3]: print(chaine[::-1])
adnamA
In [4]: print(chaine*2)
AmandaAmanda
```

- On accède à leurs éléments par leurs indices entre crochets :

```
In [1]: ch = 'Amanda'
In [2]: print(ch[0], ch[-1])
A a
```

- On peut les parcourir à l'aide d'une boucle `for` :

```
In [1]: for c in ch[:3]:
...     print(c)

A
m
a
```

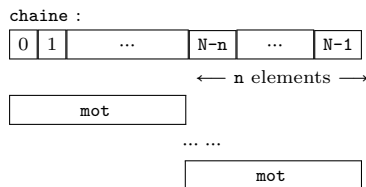
- Attention : les chaînes sont **non-modifiables** : changer un élément produit une erreur `TypeError` :

```
In [1]: chaine[0] = 'Z'
TypeError: 'str' object does not support item assignment
```

1.3 Exemple : Recherche d'un mot dans une chaîne

La recherche d'un mot dans une chaîne de caractère est un problème essentiel en informatique qui admet des algorithmes élaborés et efficaces. Donner une solution naïve, pas très efficace n'est pas difficile :

```
def contient(chaine, mot):
    N = len(chaine)
    n = len(mot)
    for i in range(N-n+1):
        if (mot == chaine[i:i+n]):
            return True
    return False
```



Le résultat est lent et coûteux en mémoire car la fonction doit effectuer jusqu'à $(N-n)$ copies d'une liste de longueur n , et pour chacune n comparaisons, soit de l'ordre de $(N-n) \times 2n$ opérations et $(N-n) \times n$ fois l'emplacement nécessaire au stockage d'un caractère utilisé.

Une meilleure solution, (mais toujours naïve) est :

```
def cherche(chaine, mot):
    N = len(chaine)
    n = len(mot)
    for i in range(N-n+1):
        k = 0
        while k < n:
            if mot[k] != chaine[i+k]:
                break
            k += 1
        if k == n:
```

```
        return True
    return False
```

On compare à chaque position, mais on passe à la position suivante dès que 2 caractères diffèrent. ON n'effectue plus les copies en mémoire. Dans le pire des cas on effectue de l'ordre de $(N-n) \times n$ opérations (de comparaisons).

Il existe des algorithmes beaucoup plus efficaces (et plus compliqués : Boyer-Moore, Knuth-Bendix-Pratt).

2 Les tuple

En français **t-uplet**. Ce sont des objets séquentiels. On les définit par la liste de leurs éléments entre parenthèses (`.`).

```
In [1]: seq = (0,1,2,3) ; print(seq)
(0, 1, 2, 3)
In [2]: print(seq[0])
0
In [3]: print(seq*2)
(0, 1, 2, 3, 0, 1, 2, 3)
```

Ils diffèrent des listes surtout en ce qu'ils sont non-modifiables.

```
In [1]: seq[0] = 1
[...]
TypeError: tuple object does not support item assignment
```

Les parenthèses sont optionnelles :

```
In [1]: a = 2 ; 2*a, 3*a, 4*a # retourne un t-uplet
(4, 6, 8)
```