# Chapitre 4
# Chaines de caractères



## 1 Chaines de caractères

Un **caractère** est un symbole quelconque du clavier : une lettre, un chiffre, un signe de ponctuation, etc. Une **chaine de caractères** est une suite de caractères : un texte.
En Python, les chaines de caractères sont notées entre guillemets, simples ou doubles, et sont de type `str` (pour *string* en anglais).

```
type("Bonjour !")

type(5)

type('5')
```

### 1.1 Opérations sur les chaines de caractères

Les opérations sur les chaines de caractères sont les suivantes.

| Opérations sur les chaines de caractères | |
|---|---|
| == | Égalité |
| len | **Longueur** (*length* en anglais) : nombre de caractères |
| + | **Concaténation** : juxtaposition |
| $n*$ | Concaténation $n$ fois avec elle-même (pour $n \in \mathbb{N}$) |

```
'conca' + 'ténation'

2 + 3
```

```
'2' + '3'

4 * "répétition"

"Identique" == 'Identique'

"Identique ?" == 'Identique?' # Les espaces comptent.

len("La typographie, c'est la vie.") # Les espaces sont comptées !
                                     # Les signes de ponctuation aussi.
```

Contrairement à ce que l'on a pu observer sur les entiers et les flottants, il n'y a pas de conversion automatique entre les types `int` et `str`.

```
1 + "4"
```

Pour convertir un entier en chaine de caractères, on peut lui appliquer la fonction **`str`**. Réciproquement, pour convertir en entier une chaine de caractères constituée uniquement de chiffres, on peut lui appliquer la fonction **`int`**.

```
str(14)

int('14')

str(int("1") + int("4"))
```

## 1.2 Accès aux caractères

Dans une chaine de caractères de longueur $n$, les caractères sont **indexés** de 0 à $n-1$, c'est-à-dire numérotés de 0 à $n-1$.

Pour tout $k \in [\![0; n-1]\!]$, on accède au **caractère d'indice k** d'une telle chaine de caractères `chaine` de la manière suivante :

```
chaine[k]
```

L'indice d'un caractère représente sa position dans la chaine.

⚠ Comme la numérotation des caractères démarre à zéro, le premier caractère de la chaine est celui d'indice 0 (et non celui d'indice 1).
Le dernier caractère de la chaine est celui d'indice `len(chaine)` $-1$.

```
"Bonjour"[3]

chaine = "L'informatique, c'est fantastique !"

len(chaine)

chaine[0]
```

```
chaine[4]

type(chaine[4])

chaine[len(chaine)]

chaine[len(chaine) - 1] # C'est le dernier caractère de la chaine.
                        # On peut aussi l'écrire chaine[-1].
```

## 1.3 Sous-chaines : tranches

Pour former une sous-chaine d'une chaine de caractères, on peut utiliser la technique du **slicing**, en spécifiant une **tranche** d'indices.
Dans une chaine de caractères `chaine`, le slicing s'écrit de la manière suivante :

- ⋆ si $i$ et $j$ sont deux entiers, alors

  ```
  chaine[i : j]
  ```

  désigne la chaine constituée des caractères de `chaine` dont l'indice est compris dans $[\![i; j-1]\!]$, c'est-à-dire entre $i$ inclus et $j$ exclu ;
- ⋆ si $i$ est un entier, alors

  ```
  chaine[i : ]
  ```

  désigne la chaine constituée des caractères de `chaine` dont l'indice est supérieur ou égal à $i$ : c'est la même sous-chaine que `chaine[i : len(chaine)]` ;
- ⋆ de même, si $j$ est un entier, alors

  ```
  chaine[ : j]
  ```

  désigne la chaine constituée des caractères de `chaine` dont l'indice est strictement inférieur à $j$ : c'est la même sous-chaine que `chaine[0 : j]` ;
- ⋆ si $i$, $j$ et $r$ sont trois entiers, alors

  ```
  chaine[i : j : r]
  ```

  est la chaine constituée des caractères de `chaine` dont les indices vont de $i$ inclus à $j$ exclu, en faisant des pas de $r$.

```
ch = "Sens dessus dessous"

ch[3 : 8]

ch[8 : 3]

ch[8 : 3 : -2]

ch[1 : : 4] # C'est la même commande que ch[1 : len(ch) : 4].

ch[ : : -1] # De droite à gauche.
```

## 2 Parcours d'une chaine de caractères

Il y a deux manières de parcourir tous les caractères d'une chaine de caractères.

La première consiste à écrire une boucle `for` dont la variable parcourt tous les entiers de 0 à sa longueur moins un, qui se trouvent être exactement les indices des caractères dans la chaine :

```
for k in range(len(chaine)) :
# L'entier k va de 0 à len(chaine) - 1,
# c'est-à-dire que k parcourt les indices des caractères de la chaine.
    bloc d'instructions
    dépendant du caractère chaine[k]
```

La seconde consiste à écrire une boucle `for` dont la variable est un caractère qui parcourt directement les caractères de la chaine un à un :

```
for lettre in chaine :
# La variable lettre prend successivement pour valeur
# chacun des caractères de la chaine chaine.
    bloc d'instructions
    dépendant du caractère lettre
```

⚠ On prendra soin de distinguer un caractère de son indice. On commentera soigneusement le code afin d'éviter les erreurs de type et de ne pas confondre les deux parcours précédents.

Les deux boucles suivantes affichent toutes les deux tous les caractères de la chaine `"Bonjour"`, un par un.

```
ch = "Bonjour"

for k in range(len(ch)): # Pour k allant de 0 à len(ch) - 1,
                         # c'est-à-dire pour k allant de de 0 à 6,
                         # c'est-à-dire pour k parcourant les indices
                         # des caractères de la chaine ch :
    print(ch[k])
    # on affiche le caractère d'indice k de la chaine ch.

for c in "Bonjour": # Pour c parcourant les caractères de la chaine ch :
    print(c)
    # on affiche le caractère c.
```

⚠ Remarquons que dans la boucle précédente, la variable `c` n'a rien à voir avec le caractère `"c"` : c'est une variable qui prend tour à tour pour valeur chaque caractère de la chaine `"Bonjour"`.

Comparons les trois scripts suivants.

```
chaine = "" # Initialement, la chaine est vide.

for lettre in "Bonjour":
    chaine = chaine + lettre
    print(chaine)
```

```
chaine = ""

for lettre in "Bonjour":
    chaine = chaine + lettre
print(chaine)
```

et

```
chaine = ""

for lettre in "Bonjour":
    chaine = lettre + chaine
    print(chaine)
```

# 3 Recherche d'un caractère dans une chaine

Un exemple très important de parcours de chaine de caractères est la recherche d'un caractère particulier dans une chaine de caractères.

Pour déterminer si une chaine de caractères donnée contient un caractère donné, on parcourt les différents caractères de la chaine.

- ⋆ Si le caractère parcouru est la lettre rechercée, on renvoie `True`, ce qui arrête la fonction (et donc le parcours).
- ⋆ Sinon, on continue le parcours.

Si à la fin du parcours, on n'a pas trouvé la lettre recherchée, alors on renvoie `False`.

- Avec une boucle `for` qui parcourt les indices de la chaine :

```
def recherche_for_indices(lettre_cherchee, chaine):
    """
    Entrées : un caractère lettre_cherchee,
              une chaine de caractères chaine.
    Sortie : True si le caractère lettre_cherchee appartient
             à la chaine chaine, False sinon.
    """
    for k in range(len(chaine)):
    # k va de 0 à len(chaine) - 1 :
    # k parcourt les indices des caractères de la chaine.
        if chaine[k] == lettre_cherchee:
            return True
            # Si le caractère d'indice k dans la chaine
            # est la lettre recherchée,
            # on renvoie True et la fonction s'arrête.
    return False
    # Si la fonction ne s'est pas arrêtée pendant le parcours,
    # c'est que le caractère n'est pas dans la chaine.
```

Testons cette fonction.

```
recherche_for_indices("o", "Bonjour")

recherche_for_indices("b", "Bonjour")
```

- Avec une boucle `for` qui parcourt les caractères de la chaine (c'est la version la plus élégante) :

```
def recherche_for_caracteres(lettre_cherchee, chaine):
    """
    Entrées : un caractère lettre_cherchee,
              une chaine de caractères chaine.
    Sortie : True si le caractère lettre_cherchee appartient
             à la chaine chaine, False sinon.
    """
    for caractere in chaine:
    # La variable caractere parcourt un à un
    # tous les caractères de la chaine.
        if caractere == lettre_cherchee:
            return True
    return False
```

En TP, on étendra le problème à la recherche d'un mot (et non d'un simple caractère) dans une chaine de caractères.