Semestre 1 / Cours 5 - Chaînes de caractères

ITC MPSI & PCSI – Année 2025-2026

Sommaire

Création et affichage d'une chaîne

Création d'une chaîne

Une chaîne de caractères est une suite de caractères indicés en commençant par l'indice 0, mais ce n'est pas une liste de caractères.

Création d'une chaîne

Une chaîne de caractères est une suite de caractères indicés en commençant par l'indice 0, mais ce n'est pas une liste de caractères.

- Une chaîne peut être délimitée par des simples quotes ou bien par des doubles quotes.

Création d'une chaîne

Une chaîne de caractères est une suite de caractères indicés en commençant par l'indice 0, mais ce n'est pas une liste de caractères.

- Une chaîne peut être délimitée par des simples quotes ou bien par des doubles quotes.

```
>>> chaine1 = "C'est un premier message."
>>> type(chaine1)
<class 'str'>
>>> chaine2 = 'Message "deux".'
```

Création d'une chaîne

Une chaîne de caractères est une suite de caractères indicés en commençant par l'indice 0, mais ce n'est pas une liste de caractères.

- Une chaîne peut être délimitée par des simples quotes ou bien par des doubles quotes.

```
>>> chaine1 = "C'est un premier message."
>>> type(chaine1)
<class 'str'>
>>> chaine2 = 'Message "deux".'
```

- Chaîne vide :

```
>>> c = ''
>>> d = str()
```

Création d'une chaîne

Une chaîne de caractères est une suite de caractères indicés en commençant par l'indice 0, mais ce n'est pas une liste de caractères.

- Une chaîne peut être délimitée par des simples quotes ou bien par des doubles quotes.

```
>>> chaine1 = "C'est un premier message."
>>> type(chaine1)
<class 'str'>
>>> chaine2 = 'Message "deux".'
```

- Chaîne vide :

```
>>> c = ''
>>> d = str()
```

- Il est possible de délimiter une chaîne par des triples apostrophes (ou guillemets), ce qui permet de l'écrire sur plusieurs lignes.

PRINT

- Pour afficher une chaîne à l'écran on utilise la fonction `print` dans l'éditeur, ou directement la console :

```
>>> chaine1 = "C'est un premier message."
>>> type(chaine1)
<class 'str'>
>>> chaine2 = 'Message "deux".'
>>> type(chaine2)
<class 'str'>
>>> chaine1
"C'est un premier message."
>>> chaine2
'Message "deux".'
```

PRINT

- Pour afficher une chaîne à l'écran on utilise la fonction `print` dans l'éditeur, ou directement la console :

```
>>> chaine1 = "C'est un premier message."
>>> type(chaine1)
<class 'str'>
>>> chaine2 = 'Message "deux".'
>>> type(chaine2)
<class 'str'>
>>> chaine1
"C'est un premier message."
>>> chaine2
'Message "deux".'
```

- La fonction `print` insère un espace par défaut, entre chaque chaîne.

- On définit une chaîne délimitée par deux quotes et contenant le caractère apostrophe comme ceci :
  `chaine = 'C\'est un message.'`

- On définit une chaîne délimitée par deux quotes et contenant le caractère apostrophe comme ceci :
  `chaine = 'C\'est un message.'`
- Grâce au caractère d'échappement \ (`backslash`), python sait que ce qui suit est une caractère faisant partie de la chaîne, et non la fin de la chaîne.

Caractère d'échappement \

- `\'` : pour insérer l'apostrophe dans une chaîne délimitée par deux apostrophes,

Caractère d'échappement \

- `\'` : pour insérer l'apostrophe dans une chaîne délimitée par deux apostrophes,

```
>>> c1 = 'C\' est une chaîne avec un guillemet \
↪ interne'
```

- `\"` : pour insérer un guillemet dans une chaîne délimitée par deux guillemets,

Caractère d'échappement \

- `\'` : pour insérer l'apostrophe dans une chaîne délimitée par deux apostrophes,

```
>>> c1 = 'C\' est une chaîne avec un guillemet \
↪ interne'
```

- `\"` : pour insérer un guillemet dans une chaîne délimitée par deux guillemets,
- Pour insérer le caractère \ il suffit de le « doubler ».

Caractère d'échappement \

- `\'` : pour insérer l'apostrophe dans une chaîne délimitée par deux apostrophes,

```
>>> c1 = 'C\' est une chaîne avec un guillemet \
↪ interne'
```

- `\"` : pour insérer un guillemet dans une chaîne délimitée par deux guillemets,
- Pour insérer le caractère \ il suffit de le « doubler ».
- Caractères de formatage de chaîne (pour l'affichage).
  - `\n` : pour insérer un saut de ligne,
  - `\t` : pour insérer une tabulation.

Caractère d'échappement \

- `\'` : pour insérer l'apostrophe dans une chaîne délimitée par deux apostrophes,

```
>>> c1 = 'C\' est une chaîne avec un guillemet \
↪ interne'
```

- `\"` : pour insérer un guillemet dans une chaîne délimitée par deux guillemets,
- Pour insérer le caractère \ il suffit de le « doubler ».
- Caractères de formatage de chaîne (pour l'affichage).
  - `\n` : pour insérer un saut de ligne,
  - `\t` : pour insérer une tabulation.

```
>>> chaine4 = 'Nom \t prénom \n Iso \t luc'
>>> print(chaine4)
Nom      prénom
 Iso      luc
```

Opérations élémentaires sur les chaînes

Longueur, comparaison, inclusion

- La fonction `len(<chaîne>)` renvoie le nombre de caractères.

```
>>> ch1 = 'anticonstitutionnellement'
>>> len(ch1)
25
```

Longueur, comparaison, inclusion

- La fonction `len(<chaîne>)` renvoie le nombre de caractères.

```
>>> ch1 = 'anticonstitutionnellement'
>>> len(ch1)
25
```

- Comparaison de chaînes : == (égalité), <, <=, >, >= (ordre lexicographique).

Longueur, comparaison, inclusion

- La fonction `len(<chaîne>)` renvoie le nombre de caractères.

```
>>> ch1 = 'anticonstitutionnellement'
>>> len(ch1)
25
```

- Comparaison de chaînes : == (égalité), <, <=, >, >= (ordre lexicographique).

```
>>> 'a' < 'b'
True
>>> 'arbre' < 'bis'
True
>>> '1' < 'a'
True
>>> '25' < '156'
False
```

Longueur, comparaison, inclusion

- Pour savoir si un caractère (ou une sous chaîne) est inclus(e) dans une chaîne, on utilise l'opérateur **`in`**, ou sa négation **`not in`**.

Longueur, comparaison, inclusion

- Pour savoir si un caractère (ou une sous chaîne) est inclus(e) dans une chaîne, on utilise l'opérateur **in**, ou sa négation **not in**.

```
>>> ch1 = 'phrase à tester'
>>> 'h' in ch1
True
>>> 'ase' in ch1
True
>>> 'axe' in ch1
False
>>> 'axe' not in ch1
True
```

Concaténation, répétition

- La concaténation est l'opération qui consiste à juxtaposer deux chaînes pour en faire une seule. En python c'est l'opération `+` entre deux chaînes.

Concaténation, répétition

- La concaténation est l'opération qui consiste à juxtaposer deux chaînes pour en faire une seule. En python c'est l'opération `+` entre deux chaînes.

```
>>> ch1 = "Bonjour "
>>> ch2 = "vous!"
>>> ch3 = ch1 + ch2  # concaténation
>>> ch4 = ch1 * 4  # répétition
>>> ch3
'Bonjour vous!'
>>> ch4
'Bonjour Bonjour Bonjour Bonjour '
>>> ch5 = 'votre note est : '
>>> ch6 = ch1 + ch5 + '15'
>>> ch6
'Bonjour votre note est : 15'
```

Concaténation, répétition

- La répétition est l'opération qui consiste à reproduire plusieurs copies d'une chaîne et de les concaténer. En python c'est l'opération * entre une chaîne et une entier.

Concaténation, répétition

- La répétition est l'opération qui consiste à reproduire plusieurs copies d'une chaîne et de les concaténer. En python c'est l'opération `*` entre une chaîne et une entier.

```
>>> ch1 = "Bonjour "
>>> ch2 = ch1 * 4  # répétition
>>> ch2
'Bonjour Bonjour Bonjour Bonjour '
```

Conversion

- Conversion d'une valeur numérique en chaîne de caractères : `str()`.

Conversion

- Conversion d'une valeur numérique en chaîne de caractères : `str()`.

```
>>> phrase = 'votre note est : '
>>> phrase + str(15)
'votre note est : 15'
```

Conversion

- Conversion d'une valeur numérique en chaîne de caractères : `str()`.

```
>>> phrase = 'votre note est : '
>>> phrase + str(15)
'votre note est : 15'
```

- Noter la différence entre ces deux syntaxes :

```
>>> phrase = 'votre note est : '
>>> note = 15
>>> phrase + 'note'
'votre note est : note'
>>> phrase + str(note)
'votre note est : 15'
```

Conversion

- Conversion d'une chaîne représentant une valeur numérique en nombre avec `int` ou `float`.

Conversion

- Conversion d'une chaîne représentant une valeur numérique en nombre avec `int` ou `float`.

```
>>> a = '123'
>>> b = '456'
>>> a + b
'123456'
>>> int(a) + int(b)
579
>>> float(a) + float(b)
579.0
```

Conversion

La conversion peut être redoutablement efficace pour diverses usages, par exemple récupérer la liste des chiffres d'un entier sans utilisation de la division euclidienne :

```
>>> str(123)
'123'
>>> list(str(123))
['1', '2', '3']
>>> [int(x) for x in str(123)] # si on préfère des \
↪ entiers
[1, 2, 3]
```

Manipulations des caractères d'une chaîne

Accès aux caractères

- Les caractères d'une chaîne sont indicés à partir de 0. Si `ch` désigne une chaîne, le caractère c_i d'indice i est `ch[i]`.

Accès aux caractères

- Les caractères d'une chaîne sont indicés à partir de 0. Si `ch` désigne une chaîne, le caractère c_i d'indice i est `ch[i]`.

```
>>> nom = 'Le corbeau'
>>> nom[0], nom[3], nom[5]
('L', 'c', 'r')
```

Accès aux caractères

- Les caractères d'une chaîne sont indicés à partir de 0. Si `ch` désigne une chaîne, le caractère c_i d'indice i est `ch[i]`.

```
>>> nom = 'Le corbeau'
>>> nom[0], nom[3], nom[5]
('L', 'c', 'r')
```

- Un indice négatif peut être utilisé. L'indice -1 désigne le dernier caractère, -2 l'avant dernier et ainsi de suite.

Accès aux caractères

- Les caractères d'une chaîne sont indicés à partir de 0. Si `ch` désigne une chaîne, le caractère c_i d'indice i est `ch[i]`.

```
>>> nom = 'Le corbeau'
>>> nom[0], nom[3], nom[5]
('L', 'c', 'r')
```

- Un indice négatif peut être utilisé. L'indice -1 désigne le dernier caractère, -2 l'avant dernier et ainsi de suite.

```
>>> nom[-1]
'u'
>>> nom[-3]
'e'
```

Accès aux caractères

- **Important** : il n'est pas possible de modifier isolément un caractère d'une chaîne. Les chaînes de caractères sont des objets **non mutables** (contrairement aux listes).

Accès aux caractères

- **Important** : il n'est pas possible de modifier isolément un caractère d'une chaîne. Les chaînes de caractères sont des objets **non mutables** (contrairement aux listes).

```
>>> nom[1] = 'E'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'str' object does not support item \
↪ assignment
```

Slicing

- Pour extraire une partie d'une chaîne on utilise le « slicing », le résultat est une chaîne. Comme pour les listes, la syntaxe est la suivante :

```
<chaîne>[indice début:indice fin:pas]
```

Slicing

- Pour extraire une partie d'une chaîne on utilise le « slicing », le résultat est une chaîne. Comme pour les listes, la syntaxe est la suivante :

  `<chaîne>[indice début:indice fin:pas]`

- Le résultat est une chaîne constituée des caractères dont les indices sont compris entre l'indice de début (inclus) et l'indice de fin (exclus), avec le pas spécifié.

Slicing

- Pour extraire une partie d'une chaîne on utilise le « slicing », le résultat est une chaîne. Comme pour les listes, la syntaxe est la suivante :

  `<chaîne>[indice début:indice fin:pas]`

- Le résultat est une chaîne constituée des caractères dont les indices sont compris entre l'indice de début (inclus) et l'indice de fin (exclus), avec le pas spécifié.
- Si l'indice de début est absent cela signifie « depuis le début », si l'indice de fin est absent cela signifie « jusqu'à la fin ».

Slicing

- Pour extraire une partie d'une chaîne on utilise le « slicing », le résultat est une chaîne. Comme pour les listes, la syntaxe est la suivante :

  ```
  <chaîne>[indice début:indice fin:pas]
  ```

- Le résultat est une chaîne constituée des caractères dont les indices sont compris entre l'indice de début (inclus) et l'indice de fin (exclus), avec le pas spécifié.
- Si l'indice de début est absent cela signifie « depuis le début », si l'indice de fin est absent cela signifie « jusqu'à la fin ».
- Le pas peut être négatif pour un parcours de la chaîne de la fin vers le début.

Slicing

- Exemple :

```
>>> nom = 'Hélène'
>>> nom[:3]  # les 3 premiers caractères
'Hél'
>>> nom[1::2]  # de 2 en 2 à partir du deuxième
'éèe'
>>> nom[1:]  # à partir du deuxième
'élène'
>>> nom[-3:]  # les 3 derniers
'ène'
>>> nom[4:0:-1]  # du 5ième au 2ième ordre inverse
'nèlé'
```

Exercice 1 :

On suppose la déclaration :

```
>>> ch = "Supercalifragilisticexpialidocious"
```

Sans retaper le mot dans son intégralité, affecter à `ch` la même chaîne, mais en remplaçant la première lettre par un `s` minuscule.

Faire de même en remplaçant le `f` par un `F` majuscule.

Exercice 1 :

On suppose la déclaration :

```
>>> ch = "Supercalifragilisticexpialidocious"
```

Sans retaper le mot dans son intégralité, affecter à `ch` la même chaîne, mais en remplaçant la première lettre par un `s` minuscule.

Faire de même en remplaçant le `f` par un `F` majuscule.

```
>>> ch = 'Supercalifragilisticexpialidocious'
>>> ch = 's' + ch[1:]
>>> ch
'supercalifragilisticexpialidocious'
>>> ch = ch[:9] + 'F' + ch[10:]
>>> ch
'supercaliFragilisticexpialidocious'
```

Parcours de chaîne

- Comme pour les listes, on peut parcourir une chaîne à l'aide d'une boucle `for`, de deux façons.

Parcours de chaîne

- Comme pour les listes, on peut parcourir une chaîne à l'aide d'une boucle `for`, de deux façons.
- La première méthode consiste à utiliser une variable entière correspondant aux indices des caractères auxquels on souhaite accéder (parcours par indice).

Parcours de chaîne

- Comme pour les listes, on peut parcourir une chaîne à l'aide d'une boucle **for**, de deux façons.
- La première méthode consiste à utiliser une variable entière correspondant aux indices des caractères auxquels on souhaite accéder (parcours par indice).

```
>>> nom = 'Hélène'
>>> for i in range(len(nom)):
...     print(i,nom[i])
...
0 H
1 é
2 l
3 è
4 n
5 e
```

Parcours de chaîne

- La deuxième méthode est un parcours par caractère, dans ce cas la variable de la boucle **`for`** reçoit successivement les caractères de la chaîne.

Parcours de chaîne

- La deuxième méthode est un parcours par caractère, dans ce cas la variable de la boucle **for** reçoit successivement les caractères de la chaîne.

```
>>> nom = 'Hélène'
>>> for z in nom:
...     print(z)
...
H
é
l
è
n
e
```

Parcours de chaîne

- La deuxième méthode est un parcours par caractère, dans ce cas la variable de la boucle **for** reçoit successivement les caractères de la chaîne.

```
>>> nom = 'Hélène'
>>> for z in nom:
...     print(z)
...
H
é
l
è
n
e
```

- Ici on ne connaît pas l'indice des différents caractères.

Parcours de chaîne

- La fonction `enumerate(<chaîne>)` permet de parcourir une chaîne en renvoyant, pour chaque caractère, un couple (indice, caractère).

Parcours de chaîne

- La fonction `enumerate(<chaîne>)` permet de parcourir une chaîne en renvoyant, pour chaque caractère, un couple (indice, caractère).

```
>>> nom = 'Hélène'
>>> for (i,car) in enumerate(nom):
...     print(i,car)
...
0 H
1 é
2 l
3 è
4 n
5 e
```

Exercice 2 :

1. Définir la chaîne de caractère `"NOM Prénom"` vous correspondant. De deux manières différentes, parcourir cette chaîne et faire afficher le premier caractère, le troisième, le cinquième ...
2. Écrire la fonction `nb(chaîne:str, car)->int` qui renvoie le nombre d'occurrences du caractère `car` dans la `chaine`.
3. Écrire la fonction `pos(chaine:str, car)->int` qui renvoie l'indice de la première occurrence du caractère `car` dans la chaîne.

Exercice 2 :

Par exemple par slicing :

```
>>> c = "BLANQUER Jean-Michel"
>>> for car in c[::2]:
...     print(car)
...
B
A
Q
E

e
n
M
c
e
```

Exercice 2 :

Ou en parcourant les bons indices :

```
>>> c = "BLANQUER Jean-Michel"
>>> for i in range(0, len(c), 2):
...     print(c[i])
...
B
A
Q
E

e
n
M
c
e
```

Exercice 2 :

1. Définir la chaîne de caractère `"NOM Prénom"` vous correspondant. De deux manières différentes, parcourir cette chaîne et faire afficher le premier caractère, le troisième, le cinquième ...
2. Écrire la fonction `nb(chaîne:str, car)->int` qui renvoie le nombre d'occurrences du caractère `car` dans la `chaine`.
3. Écrire la fonction `pos(chaine:str, car)->int` qui renvoie l'indice de la première occurrence du caractère `car` dans la chaîne.

Il s'agit de parcourir la chaîne et d'incrémenter un compteur le cas échéant :

```
def nb(ch:str,car)->int:
    n = 0
    for c in ch:
        if c == car:
            n += 1
    return n
```

Exercice 2 :

```
def pos(ch:str,car:str)->int:
    i = 0
    trouve = False
    while i < len(ch) and trouve == False:
        if ch[i] == car:
            trouve = True
        i += 1
    if i < len(ch):
        return i-1
    else:
        return False
>>> pos('Supercalifragilisticexpialidocious', 'p')
2
>>> pos('Supercalifragilisticexpialidocious', 'z')
False
```

Pour aller plus loin : méthodes spécifiques aux chaînes

- Dans une console on tape `dir(str)`.

Pour aller plus loin : méthodes spécifiques aux chaînes

- Dans une console on tape `dir(str)`.

```
>>> dir(str)[:30] # on s'arrête aux 30 premières sur \
↪ la diapo
['__add__', '__class__', '__contains__', \
↪ '__delattr__', '__dir__', '__doc__', '__eq__', \
↪ '__format__', '__ge__', '__getattribute__', \
↪ '__getitem__', '__getnewargs__', '__getstate__', \
↪ '__gt__', '__hash__', '__init__', \
↪ '__init_subclass__', '__iter__', '__le__', \
↪ '__len__', '__lt__', '__mod__', '__mul__', \
↪ '__ne__', '__new__', '__reduce__', \
↪ '__reduce_ex__', '__repr__', '__rmod__', \
↪ '__rmul__']
```

Pour aller plus loin : méthodes spécifiques aux chaînes

Pour avoir de l'aide sur une méthode, on utilise la commande `help(<méthode>)`. On appuie sur la lettre `q` pour quitter le mode d'aide.

Pour aller plus loin : méthodes spécifiques aux chaînes

Pour avoir de l'aide sur une méthode, on utilise la commande `help(<méthode>)`. On appuie sur la lettre `q` pour quitter le mode d'aide.

```
>>> help(str.replace)
Help on method_descriptor:

replace(self, old, new, /, count=-1) unbound builtins.str method
    Return a copy with all occurrences of substring old replaced by new.

      count
        Maximum number of occurrences to replace.
        -1 (the default value) means replace all occurrences.

    If the optional argument count is given, only the first count \
    ↪ occurrences are
    replaced.
```