Chaînes de caractères

I. Définition et création de chaînes

Définition 1 : *Chaîne de caractères*

Une chaîne de caractères est une suite finie de caractères (lettres, chiffres ou symboles de ponctuation) qui permet de représenter un texte. Pour définir une chaîne de caractères, on doit écrire ce texte entre guillemets `"..."` ou entre apostrophes `'...'`.

La chaîne de caractères vide se note `""` ou `''`.

Le type chaîne de caractères se nomme `str` en Python (cela vient de « string » en anglais qui signifie chaîne).

Remarque :

⋆ Si le texte lui-même contient des guillemets ou des apostrophes, on peut soit choisir l'autre délimiteur (les apostrophes si le texte contient des guillemets et vice-versa), soit échapper les guillemets et apostrophes du texte à l'aide du backslash `\`.

Échapper un caractère signifie ajouter `\` devant le caractère afin que Python ne le confonde pas avec un caractère spécial.

```
>>> "Bonjour !"
'Bonjour !'
>>> 'Hello!'
'Hello!'
>>> "Je m'appelle Victor."
"Je m'appelle Victor."
>>> 'Je m\'appelle Victor.'
"Je m'appelle Victor."
```

⋆ Pour écrire un texte sur plusieurs lignes, on doit tripler les délimiteurs `"""` ou `'''` :

```
>>> """Bonjour !
... Comment ça va ?"""
'Bonjour !\nComment ça va ?'
```

Le retour à la ligne est un caractère spécial noté `\n`. On peut aussi écrire nous-mêmes `\n` dans une chaîne de caractères.

II. Opérations et tests

1) Opérations sur les chaînes

Il existe deux opérations sur les chaînes de caractères :

- ⋆ la concaténation : `+`

```
>>> "Bonjour !" + ' ' + 'Comment ça va ?'
'Bonjour ! Comment ça va ?'
```

- ⋆ la multiplication par un entier : `*`

```
>>> 3 * "Salut"
'SalutSalutSalut'
>>> 0 * "Salut"
''
>>> -3 * "Salut"
''
```

Ainsi l'opération `3 * chaine` est équivalente à l'opération `chaine + chaine + chaine`. On remarque également que multiplier par un entier négatif ou nul donne la chaîne vide.

2) Tests et comparaisons

Comme pour les types numériques, il est possible de comparer deux chaînes de caractères avec les opérateurs de comparaison `==`, `<`, `<=`, `>`, `>=`.

L'opérateur `==` permet de tester l'égalité de deux chaînes. Les autres opérateurs permettent de déterminer quelle chaîne est avant ou après dans l'ordre alphabétique.

Exemple :

```
>>> "Bonjour" < "Salut"
True
>>> "Bonjour" == "Salut"
False
>>> "Bonjour" < "Salut"
True
>>> "Bonjour" <= "Salut"
True
>>> "Bonjour" >= "Salut"
False
>>> "Bonjour" > "Bon"
True
>>> "Bonjour" < "ah"
True
```

En ce qui concerne les comparaisons `<`, `<=`, `>`, `>=`, Python associe une valeur entière à chaque caractère puis compare les chaînes avec l'ordre lexicographique. Les valeurs associées aux caractères majuscules étant inférieures à celles associées aux lettres minuscules, cela explique le résultat du dernier exemple.

Complément

La valeur associée à un caractère peut être obtenue à l'aide de la fonction `ord` :

```
>>> ord("A")
65
>>> ord("B")
66
>>> ord("a")
97
>>> ord("!")
33
```

La fonction `chr` permet de récupérer le caractère associé à un nombre :

```
>>> chr(65)
'A'
>>> chr(66)
'B'
>>> chr(97)
'a'
>>> chr(33)
'!'
```

Enfin, il est possible de tester l'existence d'une sous-chaîne de caractères dans une chaîne plus longue avec l'opérateur `in`.

Exemple :

```
>>> chaine = "CCCTTGTTCACAGAAGCAAC"
>>> "ACAG" in chaine
True
>>> "ACAT" in chaine
False
>>> "ACAT" not in chaine
True
```

III. Fonctions et méthodes

Méthodes

Certains types de données possèdent des méthodes : ce sont des fonctions un peu spéciales qui s'appliquent directement à l'objet concerné. La syntaxe d'appel d'une méthode est différente de celle d'une fonction.

- ⋆ **Appel d'une fonction à un objet** :

```
fonction(objet)
```

- ⋆ **Appel d'une méthode à un objet** :

```
objet.methode(arg_1, arg_2, ..., arg_n)
```

1) Fonctions sur les chaînes

Fonction	Résultat	Type de retour
`len(chaine)`	Longueur (nombre de caractères) de la chaîne	`int`
`int(chaine)`	Convertit la chaîne en entier (si possible)	`int`
`float(chaine)`	Convertit la chaîne en flottant (si possible)	`float`
`str(objet)`	Convertit un objet en chaîne de caractères	`str`

Exemple :

```
>>> len("Bonjour !")
9
>>> len("")
0
>>> str(32)
'32'
>>> str(3.14)
'3.14'
>>> x = 135453431431
>>> len(str(x))     # Permet d'obtenir le nombre de chiffres de x
12
>>> int("32")
32
>>> int(2.72)
2
>>> float("3.14")
3.14
>>> float(12)
12.0
```

2) Méthodes sur les chaînes

Les méthodes suivantes ne sont pas à connaitre par cœur : elles sont données à titre informatif. Si un exercice nécessite l'utilisation de l'une de ces méthodes, celle-ci sera alors rappelée.

Méthode	Résultat	Type de retour
`chaine.replace(old, new)`	Remplace toutes les occurrences de `old` par `new` dans une copie	`str`
`chaine.lstrip(chars)`	Supprime les caractères de `chars` au début de `chaine` dans une copie	`str`
`chaine.rstrip(chars)`	Supprime les caractères de `chars` à la fin de `chaine` dans une copie	`str`
`chaine.split(sep)`	Retourne une liste de sous-chaînes de `chaine` en utilisant le séparateur `sep`	`list[str]`

Exemple :

```
>>> chaine = "Bonjour !"
>>> chaine.replace("jour", "soir")
'Bonsoir !'
```

```
>>> chaine.replace("o", "a")
'Banjaur !'
>>> chaine.lstrip("oB")
'njour !'
>>> chaine.rstrip("! ")
'Bonjour'
>>> chaine.rstrip("B")      # Pas de 'B' à la fin = aucun changement
'Bonjour !'
>>> chaine.split(" ")
['Bonjour', '!']
>>> chaine.split("o")
['B', 'nj', 'ur !']
>>> chaine
'Bonjour !'          # chaine n'est jamais modifiée (c'est une copie qui est
   renvoyée)
```

IV. Accès aux caractères d'une chaîne

1) Indices et caractères

Définition 2 : *Indices et caractères*

- ⋆ Dans une chaîne, les caractères sont numérotés par leurs indices. Les indices sont attribués de la façon suivante : le premier caractère de la chaîne a **toujours** pour indice 0, le deuxième a pour indice 1 et ainsi de suite jusqu'à la fin de la chaîne.
- ⋆ Les caractères d'une chaîne possèdent également un indice négatif attribué de la façon suivante : le dernier caractère de la chaîne a pour indice négatif -1, l'avant-dernier a pour indice négatif -2 et ainsi de suite jusqu'au début de la chaîne.
- ⋆ Pour accéder à un caractère particulier d'une chaîne de caractères, on utilise la syntaxe suivante :

  ```
  chaine[indice]
  ```

  où `chaine` est le nom de la variable de type `str` et `indice` est l'indice (positif ou négatif) du caractère auquel on veut accéder.

Par exemple, en considérant la chaîne `"Bonjour !"` qui a 9 caractères :

Caractères	B	o	n	j	o	u	r		!
Indice	0	1	2	3	4	5	6	7	8
Indice négatif	-9	-8	-7	-6	-5	-4	-3	-2	-1

Exemple :

```
>>> chaine = "Bonjour !"
>>> chaine[0]
'B'
>>> chaine[1]
'o'
>>> chaine[5]
'u'
```

```
>>> chaine[-1]
'!'
>>> chaine[-5]
'o'
>>> chaine[9]      # Indice trop grand ou trop petit = IndexError
Traceback (most recent call last):
  File "<console>", line 1, in <module>
IndexError: string index out of range
```

Remarque : Dans une chaîne de caractères `chaine`, les indices des caractères peuvent varier de `-len(chaine)` à `len(chaine)-1`.

Type immuable

Le type chaîne de caractère est dit immuable ou non mutable : cette notion sera étudiée plus en détail dans le chapitre sur les dictionnaires.
Une conséquence de la non mutabilité des chaînes de caractères est qu'il n'est pas possible de modifier un caractère en particulier d'une variable de type `str` de la façon suivante :

```
>>> chaine = "Bonjour !"
>>> chaine[3] = "g"
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

2) Tranches et sous-chaînes

Définition 3 : *Tranches (slicing)*

Pour extraire une sous-chaîne d'une chaîne de caractères, on peut utiliser l'une des syntaxes suivantes :

```
chaine[i:j]          # Chaîne composée des caractères d'indices i à j-1
chaine[:j]           # Chaîne composée des caractères d'indices 0 (premier) à j-1
chaine[i:]           # Chaîne composée des caractères d'indices i à -1 (dernier)
chaine[:]            # Copie complète de la chaîne de caractères
```

On parle alors de tranches (slicing en anglais). Les indices positifs ou négatifs peuvent être utilisés.

Exemple :

```
>>> chaine = "Bonjour !"
>>> chaine[2:5]
'njo'
>>> chaine[2:-1]
'njour '
>>> chaine[5:3]    # Chaîne vide si le premier caractère est après le dernier
''
>>> chaine[:4]     # Les 4 premiers caractères
'Bonj'
>>> chaine[-3:]    # Les 3 derniers caractères
'r !'
>>> chaine[:]      # Copie de la chaîne
'Bonjour !'
```

Remarque :

- ⋆ Dans la syntaxe `chaine[i:j]`, le caractère d'indice `j` **est exclu**.
- ⋆ Il est très pratique de retenir les longueurs des chaînes de caractères obtenues par ces syntaxes. Si $0 \leqslant$ `i` $\leqslant$ `j` $\leqslant$ `len(chaine)`, alors
  - `len(chaine[i:j]) = j - i`
  - `len(chaine[:j]) = j`
  - `len(chaine[i:]) = len(chaine) - i`
  - `len(chaine[:]) = len(chaine)`

3) Parcours d'une chaîne de caractères

Pour parcourir une chaîne de caractères, c'est-à-dire examiner les caractères de la chaîne un par un à l'aide d'une boucle, on dispose de deux moyens :

- ⋆ **Parcourir les indices de la chaîne :**

```
for i in range(len(chaine)):
    print(f"Le caractère d'indice {i} de la chaîne est {chaine[i]}.")
```

- ⋆ **Parcourir les caractères de la chaîne :**

```
for c in chaine:
    print(f"{c} est un caractère de la chaîne mais on n'a pas accès à son indice.
   ")
```

Pour certains algorithmes, un des deux parcours est plus adapté que l'autre mais dans la plupart des cas, on peut choisir entre les deux parcours. On peut également retenir que le parcours des indices fonctionne toujours ce qui n'est pas le cas du parcours des caractères.

Exemple :

- ⋆ **Code :**

```
chaine = "Bonjour !"
for i in range(len(chaine)):
    print(chaine[i])
```

Résultat :

```
B
o
n
j
o
u
r

!
```

- ⋆ **Code :**

```
chaine = "Bonjour !"
for i in range(len(chaine)):
    print(i)
```

Résultat :

```
0
1
2
3
4
5
6
7
8
```

⋆ **Code :**

```
chaine = "Bonjour !"
for c in chaine:
    print(c)
```

Résultat :

```
B
o
n
j
o
u
r

!
```

V. Algorithmes sur les chaînes de caractères

Étudions un problème classique sur les chaînes de caractères : celui de rechercher la présence (et éventuellement la position s'il est présent) d'un caractère dans une chaîne.

Commençons par décrire l'algorithme avec des mots avant de passer au code.

⋆ Entrées : `caractere` et `chaine`

⋆ On parcourt les caractères de `chaine` un par un. Pour chacun de ces caractères :

- si ce caractère est celui que l'on cherche, alors on peut arrêter la fonction et renvoyer `True` (le caractère est bien présent).
- si ce caractère n'est pas celui que l'on cherche, alors **on ne fait rien**, c'est-à-dire on continue le parcours de la chaîne.

⋆ Si à la fin du parcours, le caractère recherché n'a toujours pas été trouvé, on renvoie alors `False` (le caractère est absent).

Pour la mise en œuvre de cet algorithme, il y a plusieurs choix à faire : il faut choisir entre utiliser une boucle `for` ou une boucle `while` et, dans le cas d'une boucle `for`, choisir entre utiliser un parcours des indices ou un parcours des caractères.

⋆ **Avec une boucle `for` et un parcours des indices** :

```
def est_present(caractere, chaine):
    """
    Entrées : un caractère caractere
              une chaîne de caractères chaine
    Sortie : True si caractere est présent dans chaine
             False sinon
    """
    for i in range(len(chaine)):
        if caractere == chaine[i]:
            return True    # Boucle et fonction interrompues par return
        # Surtout pas de else !
    return False           # On renvoie False après la fin du parcours
```

⋆ **Avec une boucle `for` et un parcours des caractères** :

```
def est_present(caractere, chaine):
    """
    Entrées : un caractère caractere
              une chaîne de caractères chaine
    Sortie : True si caractere est présent dans chaine
             False sinon
    """
    for c in chaine:
        if caractere == c:  # c est directement un caractère de chaine
            return True
    return False
```

⋆ **Avec une boucle `while` :**

```
def est_present(caractere, chaine):
    """
    Entrées : un caractère caractere
              une chaîne de caractères chaine
    Sortie : True si caractere est présent dans chaine
             False sinon
    """
    i = 0   # Indice du premier caractère
    while i < len(chaine) and caractere != chaine[i]:
        i += 1
        # Tant qu'on n'est pas arrivé au bout de la chaîne et qu'on n'a pas
        # trouvé le caractère, on avance
    return i < len(chaine)
    # - Si i < len(chaine), alors la boucle while s'est arrêtée avant la fin du
    #   parcours et donc parce qu'on a trouvé le caractère cherché
    # - Si i = len(chaine), alors la boucle while s'est arrêtée parce qu'on a
    #   atteint la fin de la chaîne sans avoir trouvé le caractère
```

Remarque : Déterminer la position (l'indice) d'un caractère ou compter le nombre d'occurrences d'un caractère dans une chaîne de caractères sont des variantes de l'algorithme de recherche d'un caractère. Il suffit d'adapter l'un des programmes précédents.