III. Les chaines de caractères

Les chaines de caractères sont des listes de caractères. On parle de chainage car les caractères se suive et chaque caractère a sa place comme les maillons d'une chaine. Il est ainsi possible de faire apparaitre plusieurs fois le même caractère dans une chaine.

Les valeurs littérales de chaînes de caractères s'écrivent de plusieurs manières en Python. Il est possible d'utiliser indifféremment

- Des apostrophes ('),
- Des guillemets ("),
- Les triples apostrophes (' ' ')
- Ou des triples guillemets (" " ").

Le choix est souvent imposé par le contenu de la chaîne : une chaîne contenant des guillemets simples sera déclarée avec des guillemets doubles et réciproquement. Pour les autres cas, la chaine s'étale sur plusieurs lignes.

```
>>>  x = 'Hello '
>>> x
'Hello '
>>> print (x)
Hello
>>> y = "world!"
>>> y
'world!'
>>> print (y)
world!
>>> z = '''hello
world'''
>>> z
'Hello \nworld!'
>>> print (z)
Hello
world!
```

1. *Caractères spéciaux :*

Le dernier exemple ci-dessus (variable `z`) montre l'introduction dans une chaîne d'un *caractère spécial* désigné par **`\n`**. Ce caractère désigne le passage à la ligne, ce que montre très bien l'instruction **`print(z).`**

Voici quelques exemples de caractères spéciaux :

- `\n` permet d'insérer des marques de *passage à la ligne* dans une chaîne non délimitée par des quotes simples.

```
>>> print ("Une chaîne\nsur plusieurs\nlignes.")
Une chaîne
sur plusieurs
lignes
```

➲ `\t` permet d'insérer des marques de *tabulation* dans une chaîne.

```
>>> print ("Une chaîne\n\tsur plusieurs lignes\n\t\tavec tabulations")
Une chaîne
        Sur plusieurs lignes
                Avec tabulations
```

Même s'ils sont écrits avec deux symboles, les caractères spéciaux comptent pour un seul caractère.

2. Opérations sur les chaînes

2.1. Longueur

La longueur d'une chaîne de caractères est le nombre de caractères qu'elle contient. En Python, c'est la fonction **`len`** qui permet d'obtenir la longueur d'une chaîne.

```
>>>  x = 'Hello '
>>> len(x)
5
>>>  y = 'Hello\n '
>>> len(y)
6
>>>  z = 'Hello  world'
>>> len(z)
11
```

2.2. Conversion numériques/chaînes

Pour convertir une chaîne ne contenant que des caractères numériques en un nombre entier ou flottant, on utilise les fonctions `int` et `float`.

```
>>> int ("34")
34
>>> float ("34")
34.0
```

La chaîne doit contenir une chaîne représentant un nombre. Par exemple si des caractères sont ajoutés à la chaîne, une erreur sera déclenchée.

```
>>> int ("334R")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    int ("334R")
ValueError: invalid literal for int() with base 10: '334R'
```

La fonction `str` est la fonction réciproque des précédentes. On utilise pour effectuer la conversion en sens inverse

```
>>> str (45.7)
'45.7'
```

2.3. Notation indicielle

Python offre une méthode simple pour accéder aux caractères contenus dans une chaîne : une chaîne est manipulée comme une séquence indexée de caractères. Ainsi, chaque caractère est accessible directement par son ***index*** ou ***indice***.

Le premier caractère (le plus à gauche) d'une chaîne de longueur n à pour indice **0** et le dernier l'indice **n-1**.

L'indice est mis entre crochets.

```
>>> x = "hello world!"
>>> x[0]
'h'
>>> x[4]
'o'
>>> x[11]
'!'
```

L'indice peut être négatif. Dans ce cas, il désigne un caractère repéré depuis la fin de la chaîne.

```
>>> x[-1]
'!'
>>> x[-2]
'd'
>>> x[-12]
'h'
```

2.4. Sous-chaînes

En plus de cet accès unitaire aux caractères, il est possible d'accéder à des sous-chaînes en précisant la tranche souhaitée par l'indice de son premier caractère et l'indice du caractère suivant le dernier caractère de cette tranche.

```
>>> x = 'Hello World!'
>>> x[2:4]
'll'
```

Si l'index de début est manquant, Python considère que le début est 0,

```
>>> x[:5]
'Hello'
>>> x[:5] == x[0:5]
True
```

si l'index de fin est manquant, la longueur de la chaîne est prise.

```
>>> x[3:]
'llo World!'
>>> x[3:] == x[3:len(x)]
True
>>> x[:]
'Hello World!'
>>> x[:] == x
True
```

Enfin, un index négatif précise que le calcul s'effectue depuis la fin de la chaîne.

```
>>> x[-3:]
'ld!'
>>> x[1:-1]
'ello World'
```

2.5. Autres fonctions de manipulation des chaines

Fonction/Expression	Description
mot1 + mot2	Colle les deux chaînes mots1 et mots2. On dit **concaténer**
mot *nb	Recopie nb fois la chaîne mot
mot.upper()	Renvoie la chaîne mot en **majuscule**
mot.lower()	Renvoie la chaîne mot en **minuscule**
c in ch	Renvoie True si le caractère c est dans la chaîne ch, et False sinon
ch.count('text')	Compte le nombre d'occurrence de *text* dans la chaîne ch
ch.find('text')	Renvoie la position de *text* dans la chaîne ch, et –1 si *text* n'y est pas
ch.replace ('tx1','tx2')	Remplace chaque *tx1* par *tx2* dans la chaîne ch
ch.capitalize()	Met la première lettre en majuscule
ch.strip()	Copie de s en retirant les "blancs" en début et fin ,et supprime la retour à ligne

Opération sur les chaines de caractères

⇒ ***Exemple***

```
>>> s = "salut, Je m'appelle Sara..."
>>> print(s.lower())
salut, je m'appelle sara..."
>>> print(s.upper())
SALUT, JE M'APPELLE SARA..."
>>> print(s.title())
Salut, Je M'Appelle Sara...
>>> print(s.capitalize())
Salut, je m'appelle sara...
>>> s = "abracadabra"
>>> print(s.count("bra"), "et", s.count("a"))
2 et 5
>>> "bra" in s
True
>>> "test" in s
False
>>> print(s.find("bra"))
1
>>> print(s.replace("bra","ta"))
atacadata
>>> s = "      salut \n Je m'appelle Sara...   \n"
>>> s.strip()
"salut \n Je m'appelle Sara.
```