Fiche : Chaînes de caractères

Une *chaîne de caractères* est une suite ordonnée de caractères délimitée par des guillemets `"` ou, plus rarement, par des apostrophes `'`. Une chaîne de caractères n'est pas **modifiable**.

Exemples de chaînes littérales :

- `'bonjour'` est une chaîne de caractères contenant 7 caractères.
- `''` est la chaîne de caractères vide contenant 0 caractère.

1 Caractère d'échappement

Le caractère \ est un caractère d'échappement car il permet de modifier le sens du caractère qui le suit.

Par exemple, il permet que le caractère `'` (ou `"`) ne soit pas considéré comme le délimiteur de fin de chaîne lorsque celle-ci est délimitée des apostrophes (ou guillemets).

[ ]:
```
message='L\'hiver approche, vite vite, il faut se couvrir des morsures du froid.'
print(message)
```

```
L'hiver approche, vite vite, il faut se couvrir des morsures du froid.
```

Remarque: Pour éviter l'utilisation de caractères d'échappement on peut utiliser les guillemets `"` pour délimiter la chaîne de caractères.

Le caractère \ devant le caractère `n` correspond à un retour à la ligne.

Remarque : Si l'on souhaite définir une chaîne de caractères littérales sur plusieurs lignes, on peut également écrire cette valeur sur plusieurs lignes en délimitant la chaîne par `"""` ou `'''`.

[ ]:
```
message = """Run!
Run!!
Run!!!"""
```

2 Concaténation de chaînes de caractères

L'opérateur `+` permet de concaténer des chaînes de caractères.

```
start = "hello "
name = "world"
message  = start + name + " !" # correspond à la chaîne "hello world !"
```

3 Les chaînes de caractères vues comme des tableaux

Même si les chaînes de caractères peuvent être soumises à une partie des opérations prévues pour les tableaux, elle ne sont pas de **type** tableau, c'est-à-dire que `'abc'` est différent de `['a, 'b', 'c']` : le second est modifiable, pas le premier.

[ ]:
```
message='L\'hiver approche, vite vite, il faut se couvrir des morsures du froid.'
print(message[4]) # affichage : v
print(message[3] + message[6]) # affichage : ir
message[1] = 'c'# erreur
```

4 Les chaînes sont comparables

La fonction `ord(car)` permet de connaître la valeur de l'unicode représentant le caractère `car`. Les chaînes peuvent être comparées alphabétiquement selon l'ordre de leur unicode. Tous les opérateurs de comparaison `(>, <..)` fonctionnent alors avec les chaînes de caractères. Cela est très utile pour trier des mots par ordre alphabétique (usuel) quand ils sont écrits en majuscules/minuscles :

[ ]:
```
mot = input("Entrez un mot quelconque : ")
if mot < "chat":
    place = "est avant"
elif mot > "chat":
    place = "est après"
else:
    place = "se confond avec"
print ("Le mot", mot, place, "le mot 'chat' dans l'ordre alphabétique")
```

5 Quelques fonctions sur les chaînes

- `len(ch)` : retourne le nombre de caractères de la chaîne `ch`.
- `find(mot)` : renvoie la position du début de la sous-chaîne `mot` dans la chaîne, en partant du début de la chaîne et `-1` si `mot`est absent de la chaîne.
- `ch.count(mot)` : renvoie le nombre d'occurrencs de la sous-chaîne `mot` qui apparaissent sans chevauchement dans la chaîne `ch`.
- `ch.lower()` : retourne une chaîne obtenue à partir de la chaîne `ch` en remplaçant les majuscules par des minuscules.
- `ch.upper()` : retourne une chaîne obtenue à partir de la chaîne `ch` en remplaçant les minuscules par majuscules.
- `ch.strip()` : retourne la chaîne obtenur après avoir enlèvé les espaces et `\n` éventuels au début et à la fin de la chaîne `ch`.
- `ch.replace(old, new)` : retoune la chaîne obtenue après avoir remplacé tous les séquences de caractères `old` par les séquences de caractères `new` dans la chaîne `ch`.
- `ch.split()` renvoie un tableau contenant les différents mots de la chaîne `ch` (on peut préciser le séparateur de mots entre les parenthèses).