Chapitre (S1) 5 Chaînes de caractères

Objectifs
- Connaître l'objet *str* de python.
- Savoir manipuler les chaînes.

Dans une première approche, une chaîne de caractères peut être vue comme une suite de caractères indicés en commençant par l'indice 0, ce n'est cependant pas une liste de caractères. La structure de données en python pour les chaînes est le type « str » . Depuis la version 3, python gère nativement l'unicode (en particulier l'utf-8), ce qui permet la gestion des caractères accentués (et même d'autres alphabets que le nôtre).

1. CRÉATION ET AFFICHAGE D'UNE CHAÎNE

1.1. Création

Une chaîne de caractères en python peut être délimitée par des apostrophes (simple quotes) ou bien par des guillemets (double quotes). Exemple :

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

On peut créer une chaîne vide de deux manières différentes :

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

Il est également possible de délimiter une chaîne par des triples apostrophes (ou guillemets), ce qui permet de l'écrire sur plusieurs lignes. Ceci est souvent utilisé pour documenter les fonctions lors de leur définition.

1.2. Affichage

Pour afficher une chaîne à l'écran on utilise la fonction `print` :

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

La fonction `print` insère un espace par défaut, entre chaque chaîne. Dans la fonction `print`, on peut préciser deux arguments particuliers, `end =` et `sep =`, pour indiquer la fin de la chaîne, ou la partie qui sépare chaque argument à l'affichage.

```
>>> print(1,2,3,end='=',sep="+"); print(6)
1+2+3=6
```

1.3. Cas particuliers

Grâce au caractère d'échappement \, python sait que ce qui suit est un caractère faisant partie de la chaîne, et non la fin de la chaîne. On peut le rencontrer dans plusieurs circonstances à l'intérieur d'une chaîne, notamment :

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

Pour insérer le caractère backslash il suffit de le « doubler » :

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

Alors la chaîne `c1` donne dans un affichage :

```
C' est une chaîne avec un guillemet interne
```

Il existe des caractères de formatage de chaîne :

- `\n` : pour insérer un saut de ligne,
- `\t` : pour insérer une tabulation.

Ces caractères interviennent lors de l'affichage des chaînes, ainsi :

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

2. OPÉRATIONS ÉLÉMENTAIRES SUR LES CHAÎNES

2.1. Longueur

La fonction `len(chaine)` renvoie la longueur d'une chaîne, c'est à dire le nombre de caractères.

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

2.2. Comparaison, inclusion

Deux chaînes peuvent être comparées avec les opérateurs == (égalité), <, <=, >, >= (comparaisons selon l'ordre lexicographique).

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

Noter le résultat du dernier test, qui diffère bien sûr de celui réalisé entre entiers :

```
>>> 25 < 156
True
```

Pour savoir si un caractère (où une sous chaîne) est inclus(e) dans une chaîne, on peut utiliser 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
```

Remarque 1 Cette fonction de comparaison est une fonction « avancée » de python qui cache en réalité un ensemble de comparaisons élémentaires. Nous verrons ultérieurement comment cette fonction est construite.

2.3. Concaténation, répétition

La concaténation est l'opération qui consiste à juxtaposer deux chaînes pour en faire une seule. En python c'est l'opérateur `+` et les deux opérandes doivent être de même type :

```
>>> 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'
```

2.4. Conversion

Il est possible de convertir une valeur numérique en chaîne de caractères avec la fonction `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'
```

À l'inverse, il est possible de transformer une chaîne représentant une valeur numérique en nombre avec les fonctions `int` ou `float`.

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

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]
```

3. MANIPULATIONS DES CARACTÈRES D'UNE CHAÎNE

3.1. 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]`. On peut représenter la chaîne `ch`, contenant n caractères par : `ch = "ch[0] ... ch[n-1]"`

Exemple 1

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

Un indice négatif peut être utilisé, il désigne alors le caractère situé « avant » le premier caractère, en imaginant les caractères disposés de manière cyclique. L'indice -1 désigne le dernier caractère, -2 l'avant dernier et ainsi de suite.

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

Attention **Non mutabilité**

Il n'est pas possible de modifier isolément un caractère d'une chaîne. On dit que 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
```

3.2. Slicing

Il est possible d'extraire une partie d'une chaîne en utilisant le « slicing » (de l'anglais *to slice* : découper en tranches), le résultat sera une chaîne. Comme pour les listes, la syntaxe générale 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. On remarque l'analogie avec la commande `range(début,fin,pas)`. Par défaut le pas est de 1. 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.

Exemple 2

```
>>> 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'
```

Remarque 2 Un autre façon d'envisager le slicing est de considérer que ce sont les espaces entre caractères qui sont indicés (l'espace indice 0 précédant le premier caractère), et l'on sélectionne les caractères compris entre les espaces indiqués par les indices de début et de fin, les deux étant inclus dans cette vision.

Exercice 1 [Sol 1] On suppose la déclaration :

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

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

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

3.3. Parcours de chaîne

Comme pour les listes, on peut parcourir une chaîne à l'aide d'une boucle `for` en accédant aux différents caractères, par deux méthodes distinctes. 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
```

Cette méthode est un peu longue à écrire mais permet d'avoir une information sur l'indice des différents caractères. La deuxième méthode est un parcours par caractère (une chaîne de caractère est un **itérable**), 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
```

Bien évidement, ici on ne peut accéder simplement à l'indice des différents caractères.

Exercice 2 [Sol 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, car)` qui renvoie le nombre d'occurrences du caractère `car` dans la `chaine`.

3) Écrire la fonction `pos(chaine, car)` qui renvoie l'indice de la première occurrence du caractère `car` dans la chaîne.

Remarque 3 De manière un peu hybride, la fonction `enumerate(<objet>)` permet de parcourir un itérable en renvoyant, pour chaque élément, un couple (indice,élément), la numérotation des indices commençant par défaut à 0.

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

3.4. Pour aller plus loin : méthodes spécifiques aux chaînes

Dans une console on tape `dir(str)` :

```
>>> dir(str)
```

```
['__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__', '__setattr__', '__sizeof__', '__str__', \
↪ '__subclasshook__', 'capitalize', 'casefold', 'center', \
↪ 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', \
↪ 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', \
↪ 'isdecimal', 'isdigit', 'isidentifier', 'islower', \
↪ 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', \
↪ 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', \
↪ 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', \
↪ 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', \
↪ 'splitlines', 'startswith', 'strip', 'swapcase', 'title', \
↪ 'translate', 'upper', 'zfill']
```

On a ainsi la liste des méthodes applicables au type « str » , puis, pour avoir de l'aide sur une méthode particulière, on utilise la commande `help` :

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

replace(self, old, new, count=-1, /)
    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.
```

On appuie sur la lettre q pour quitter le mode d'aide.

SOLUTIONS DES EXERCICES

Solution 1 On peut utiliser le slicing pour éviter les recopies :

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

Solution 2

1) On définit c = 'NOM Prénom' On peut utiliser le parcours par indice :

```
for i in range(0,len(c),2):
    print(c[i])
```

ou bien le le parcours de chaîne et le slicing :

```
for car in c[::2]:
    print(car)
```

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

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

3) Il y a la méthode `index` qui fait déjà cela, mais voici comment on pourrait la définir :

```
def pos(ch,car):
    for i in range(len(ch)):
        if ch[i] == car:
            return i
```

Cette version est plutôt à éviter (`return` dans une boucle). On utilisera donc plutôt la nouvelle version ci-après utilisant un `while`.

```
def pos(ch,car):
    i = 0
    trouve = False
    while i < len(ch) and trouve == False:
        if ch[i] == car:
            trouve = True
        i += 1
    return i-1
```

```
>>> pos('Supercalifragilisticexpialidocious', 'p')
2
```