Chaînes de caractères : manipulation, méthodes...

Algorithmique et structuration de données

Léna Gaubert
lena.gaubert@proton.me

L3 TIM/TAL @ INALCO

Manipulation de chaîne de caractères

Précédemment...

Une chaîne de caractère, ou `string` (abbréviation : `str`), est un type d'objet (*data type*).

- **caractères** : alphanumériques, caractères spéciaux, espace... (Unicode)
- entre **guillemets** doubles ou simples
- **opération possibles** :
  - addition/concaténation (`"hello" + " " + "world"`)
  - multiplication d'une chaîne par un entier (`"hello" * 3`)

Chaîne de caractères

Une chaîne de caractères est une *séquence* de caractères Unicode.

> Une **séquence** est une collection d'objets *ordonnés*.

Une chaîne de caractères est donc : ordonnée, indexable, trancheable (*sliceable*), itérable.

Assignation à une variable

- Entre guillemets simples ou doubles :

```
word = "magic"
word2 = 'clouds'
```

- *Multiline string* : avec des triples guillemets (simples ou doubles)

```
txt = """Echo, echo, echo,
The lights they go
"""
```

Longueur d'une chaîne

Pour trouver la longueur d'une chaîne, vous pouvez utiliser la fonction `len()` (*length*). Le résultat de la fonction `len()` correspond au nombre de caractères dans la chaîne.

- input : `string`
- output : `int`

```
txt = "Python is amazing!"
length = len(txt)
print(length)  # 17
```

Indexation

Une chaîne de caractères étant une *séquence*, on peut accéder à ses éléments par *indexation*.

```
word = "fontaines"
print(word[1])
print(word[3])
print(word[-1])
print(word[9])
```

Indexation

En Python, l'indexation commence à 0. La syntaxe est la suivante : `sequence[index]`, avec `sequence` une séquence (chaîne de caractère, liste...) et `index` un entier (`int`).

- Les index (ou indices) *négatifs* partent de la fin de la chaîne :
  - -1 renvoie le dernier élément
  - -2 l'avant dernier, etc
- Accéder à un indice qui dépasse la longueur de la séquence renvoie une erreur `IndexError`.

Slicing : extraire une sous-chaîne

La syntaxe générale est la suivante : `sequence[start:stop:step]`

- `start` : indice à partir duquel la tranche commence (inclusif, par défaut, `start=0`)
- `end` : indice à partir duquel la tranche se termine (exclusif, par défaut, `end=len(str)-1`)
- `step` : le pas (optionnel, par défaut, `step=1`)

Slicing

Exercice : tester les lignes de code suivantes dans votre terminal.

```
txt = "Bonjour"
print(len(txt))
print(txt[len(txt)])
print(txt[3:6])
print(txt[1:5:2])
print(txt[1:])
print(txt[:5:2])
print(txt[::-1])
```

Slicing

Sauter des lettres dans une chaîne

On utilise le troisième argument dans le découpage de chaîne pour spécifier un pas.

```
txt = "Bonjour"
print(txt[::2])  # Bnour
```

Inverser une chaîne

Pour inverser une chaîne, on utilise un pas *négatif*.

```
txt = "Bonjour"
print(txt[::-1])  # ruojnoB
```

Slicing

Extraire une tranche d'une chaîne

Utiliser le premier et deuxième argument (et le troisième, si on souhaite changer le pas).

```
txt = "Bonjour"
print(txt[:4])  # Bonj
print(txt[1:])  # onjour
print(txt[2:5])  # njo
```

Formatage

f-strings, écriture scientifique...

Formatage

Plutôt que de faire appel à la concaténation, on peut **formater des chaînes de charactères** à l'aide des *f-strings*. Plus globalement, les *f-strings* permettent d'afficher des variables avec un **format précis**.

```
name = "Grian"
age = 31
msg = f"My name is {name} and I am {age}."
print(msg)
```

```
name = "Ada"
age = 43
msg = """I am {name} and
I am {age} years old.
""".format(name=name, age=age)
```

Formatage

Pour les *f-strings*, il existe deux syntaxes à connaître :

- `f"Message {variable}"` : `variable` est définie au préalable. Le `f` en amont des guillemets indique à l'interpréteur Python qu'il s'agit d'une *f-string*.
- `"Message {variable}".format(variable=var)` : la méhode `.format()` permet de formatter la chaîne de caractère. `var` est définie en amont.

Affichage avec une précision numérique

Exécuter les lignes de code ci-dessous. Que se passe-t-il ?

```
x = 3.14159265
print(f"{x:.2f}")
print(f"{x:.4f}")
print(f"{x:8.3f}")
```

Écriture scientifique

Exécuter les lignes de code ci-dessous. Que se passe-t-il ?

```
x = 12345.6789
print(f"{x:e}")
print(f"{x:.2e}")
```

Écriture scienfitique : rappel

Soit x un nombre réel, a un nombre décimal (en général : $1 \leq |a| < 10$), et n un entier. On peut exprimer x sous la forme :

$$x = a \times 10^n$$

Exemple :

- $12345 = 1.2345 \times 10^4$
- $0.00056 = 5.6 \times 10^{-4}$
- $7 = 7.0 \times 10^0$

En résumé

Dans une *f-string*, on écrit :

- `{variable:.nf}` pour afficher un flottant (*float*) avec `n` décimales,
- `{variable:.ne}` pour la notation scientifique avec `n` décimales

Méthodes

Opérations courantes sur les chaînes de caractères

Méthodes

Python propose de nombreuses méthodes pour réaliser des opérations courantes sur les chaînes.

- `.upper()`
- `.lower()`
- `.strip()`

```
txt = "  Python JavaScript  "
stripped = txt.strip()
upper_txt = txt.upper()
print(stripped)
print(upper_txt)
```

Recherche

A l'aide des opérateurs logiques, on peut vérifier si une sous-chaîne existe dans une chaîne.

```
txt = "Python is a great language.
if "Python" in txt:
    print("Python is in the text")
```

time to practice!! (๑>ᴗ<)๑