Le formatage des chaînes de caractères

La concaténation avec l'opérateur +

Avant de parler de formatage, on parle souvent de concaténation.

La concaténation consiste à mettre bout à bout plusieurs chaînes de caractères.

La concaténation peut se faire très simplement avec l'opérateur mathématique `+` :

```
protocole = "https://"
nom_du_site = "Docstring"
extension = "fr"

url = protocole + "www." + nom_du_site + "." + extension
```

Ça fonctionne, mais très rapidement vous allez observer deux problèmes :

- Ce n'est pas très agréable à lire.
- Vous ne pouvez concaténer que des objets du même type (car Python est un langage fortement typé).

Ainsi, le code suivant retournera une erreur car la variable `age` est un nombre entier :

```
>>> age = 26
>>> phrase = "J'ai " + age + " ans."
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
```

Pour palier à cette erreur, il faut convertir la variable `age` en chaîne de caractères :

```
>>> age = 26
>>> phrase = "J'ai " + str(age) + " ans."
"J'ai 26 ans."
```

Et on retombe sur le premier problème : ça devient très vite difficile à lire et c'est le meilleur moyen de se retrouver avec des `TypeError`.

La concaténation avec les f-string

Depuis la version 3.6 de Python, une fonctionnalité très intéressante a fait son apparition : **les f-string**.

Derrière ce nom un peu bizarre se trouve une façon très efficace de formater des chaînes de caractères.

C'est vrai autant pour la lisibilité que pour les performances, qui seront meilleures qu'avec la méthode format (ça reste de l'ordre du millième de milliseconde mais tout de même).

Pour utiliser les f-string, il suffit de rajouter la lettre **f** devant votre chaîne de caractères :

```
phrase = f"Je suis un f-string."
```

On peut ensuite, comme avec la méthode format, insérer des données dans la chaîne de caractères avec les accolades.

La différence majeure réside dans le fait qu'on peut écrire du code Python directement dans les accolades.

On retrouve donc la flexibilité de la méthode format mais avec quelque chose d'encore plus facile à lire.

Reprenons l'exemple de l'URL :

```
protocole = "https://"
nom_du_site = "Docstring"
```

```
extension = "fr"

url = f"{protocole}www.{nom_du_site}.{extension}"  # https://www.Docstring.fr
```

Vous souhaitez mettre le nom du site en minuscule ?

Aucun problème, on peut utiliser la méthode `lower` directement à l'intérieur du f-string :

```
protocole = "https://"
nom_du_site = "Docstring"
extension = "fr"

url = f"{protocole}www.{nom_du_site.lower()}.{extension}"  # https://www.docstring.fr
```

On peut insérer n'importe quel type d'objet dans un f-string sans que cela ne pose de problème à Python :

```
>>> liste = ['Pommes', 'Poires', 'Bananes']
>>> print(f"Voici votre liste de courses : {liste}")
Voici votre liste de courses : ['Pommes', 'Poires', 'Bananes']
```

Et comme toujours, on peut écrire du code Python directement à l'intérieur des accolades :

```
>>> liste = ['Pommes', 'Poires', 'Bananes']
>>> print(f"Voici votre liste de courses : {', '.join(liste)}")
Voici votre liste de courses : Pommes, Poires, Bananes
```

La méthode format

Pour éviter les erreurs de type et insérer des objets dans une chaîne de caractères, on peut utiliser la méthode `format`.

Cette méthode permet d'insérer des objets à l'intérieur d'emplacements spécifiés dans la chaîne de caractères par des **accolades** :

```
age = 26
phrase = "J'ai {} ans".format(age)
```

Ici, pas besoin de convertir la variable `age` en chaîne de caractères. La méthode format s'en charge pour nous et nous évitons ainsi le `TypeError`.

Cela permet également de garder une certaine continuité dans notre chaîne de caractères qui ne se retrouve pas entrecoupée d'opérateurs `+`.

On peut également spécifier à l'intérieur des accolades un nom qu'on utilisera comme paramètre dans la méthode format pour obtenir une phrase encore plus claire :

```
age_de_lutilisateur = 26
phrase = "J'ai {age} ans".format(age=age_de_lutilisateur)
```

On peut mettre autant d'emplacements dans une chaîne de caractères que nécessaire.

Il faudra cependant faire attention de passer autant d'éléments à la méthode format qu'il y a d'emplacements dans notre chaîne de caractères.

Par exemple, le code suivant fonctionne :

```
prenom = "Pierre"
age = 26
phrase = "Je m'appelle {} et j'ai {} ans.".format(prenom, age)
```

Mais ci-dessous nous aurons une erreur (`IndexError: tuple index out of range`) car il y a deux emplacements dans la chaîne de caractères mais un seul élément est passé à la méthode format :

```
prenom = "Pierre"
age = 26
phrase = "Je m'appelle {} et j'ai {} ans.".format(prenom)
```

On peut également spécifier à l'intérieur des accolades un indice, qui fera référence à la position de l'argument de la méthode format à utiliser.

👉 C'est très pratique si vous souhaitez réutiliser un argument :

```
prenom = "Pierre"
age = 26
langage = "Python"
phrase = "Je m'appelle {0} et j'ai {1} ans. {0} apprends le langage {2}.".format(prenom, age, langage)
# Je m'appelle Pierre et j'ai 26 ans. Pierre apprends le langage Python.
```

On peut également remettre des noms à l'intérieur des accolades afin d'obtenir quelque chose de plus agréable à lire :

```
prenom = "Pierre"
age = 26
langage = "Python"
phrase = "Je m'appelle {prenom} et j'ai {age} ans. {prenom} apprends le langage {langage}.".format(prenom=prenom, age=age, langage=l
```

Le nom que vous mettez à l'intérieur des accolades n'a pas besoin d'être le même que la variable que vous passez à la méthode format. On aurait très bien pu faire :

```
prenom = "Pierre"
age = 26
langage = "Python"
phrase = "Je m'appelle {user} et j'ai {age} ans. {user} apprends le langage {learning}.".format(user=prenom, age=age, learning=langa
```

👉 La méthode format permet donc d'obtenir un code beaucoup plus facile à lire et évite de s'embêter avec les fonctions de conversion.

Si on reprend l'exemple de l'URL du début de l'article :

```
protocole = "https://"
nom_du_site = "Docstring"
extension = "fr"

# Avec l'opérateur +
url = protocole + "www." + nom_du_site + "." + extension

# Avec la méthode format
url = "{}www.{}.{}".format(protocole, nom_du_site, extension)
url = "{protocole}www.{domaine}.{extension}".format(protocole=protocole,
                                                    domaine=nom_du_site,
                                                    extension="fr")
```

L'avantage de format sur les f-string

La plupart du temps, les f-string sont **beaucoup plus agréables à lire et facile à utiliser** que la méthode format.

J'utilise **95% du temps** les f-string mais il reste encore quelques cas de figure dans lesquels la méthode format est utile.

👉 Le principal avantage de la méthode format est que vous pouvez définir votre chaîne de caractères à un endroit de votre script et ne l'utiliser que plus tard.

Les f-string nécessitent que les variables que vous insérez dans votre chaîne de caractères soient disponible immédiatement.

Si vous essayez d'insérer dans un f-string une variable qui n'a pas encore été déclarée, vous obtiendrez évidemment une erreur :

```
>>> prenom = "Pierre"
>>> phrase = f"Bonjour {prenom}, cette semaine vous avez regardé {nombre} vidéos."
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'nombre' is not defined
```

Or, il arrive que l'on définisse des chaînes de caractères à utiliser dans un fichier de configuration et que l'on souhaite utiliser ces phrases à différents endroits de notre script avec des variables qui seront définies par la suite.

```
# constants.py
BONJOUR = "Bienvenue {prenom}, vous avez regardé {nombre_videos} vidéos cette semaine."
AU_REVOIR = "À bientôt {prenom} !"
```

```
# main.py
from constants import BONJOUR

user = input("Entrez votre nom d'utilisateur : ")
progression = get_weekly_progress(user)

message_de_bienvenue = BONJOUR.format(prenom=user, nombre_videos=progression)
print(message_de_bienvenue)
```

Ici, on définit des phrases à afficher dans un fichier `constants.py`.

On utilise ensuite ces phrases en les formatant dans le script `main.py`.

On pourrait ainsi ré-utiliser ces phrases dans plusieurs scripts sans avoir besoin de les réécrire à chaque fois.