PYTHON – Chaines de caractères

Introduction

Voici un exemple de chaînes de caractères :

```
chaine = "CECI EST UNE CHAINE"
print(type(chaine))
print(chaine.lower()) # chaîne en minuscule
```

```
<class 'str'>
ceci est une chaine
```

- Si vous tapez type(chaine) dans l'interpréteur, vous obtenez <class 'str'>
  Les développeurs de Python ont créé le type **str** qui est utilisée pour créer des chaînes de caractères
- Le type **str**, possède **plusieurs méthodes**, comme lower.
  La fonction lower est propre aux chaînes de caractères. Toutes les chaînes peuvent y faire appel.

Mettre en forme une chaîne

- chaine.lower() : renvoie la chaîne en minuscules mais ne modifie pas la chaîne
- str() : crée un objet *chaîne de caractères*
- chaine.upper() : renvoie la chaine en majuscules
- chaine.capitalize() : renvoie La première lettre en majuscule
- chaine.strip() : retire les espaces au début et à la fin de la chaîne
- chaine.center(20) : On lui passe en paramètre la taille de la chaîne que l'on souhaite obtenir. La méthode va ajouter alternativement un espace au début et à la fin de la chaîne, jusqu'à obtenir la longueur demandée

On peut « chaîner » les méthodes : Exemple :

```
titre = "introduction"
print(titre.upper().center(20))
```

```
'    INTRODUCTION    '
```

Vous pouvez en voir la liste exhaustive des méthodes dans l'aide, en tapant, dans l'interpréteur : help(str)

Formater et afficher une chaîne : méthode format

La méthode **format** est une méthode de la classe str, utilisée pour formater une chaîne

```
prenom = "Jean"
nom = "Durant"
age = 21
print("Je m'appelle {0} {1} et j'ai {2} ans.".format(prenom, nom, age))
```

```
Je m'appelle Jean Durant et j'ai 21 ans.
```

De gauche à droite, nous avons :

- une chaîne de caractères qui ne présente rien de particulier, sauf ces accolades entourant des nombres, d'abord 0, puis 1, puis 2 ;
- nous appelons la méthode format de cette chaîne en lui passant en paramètres les variables à afficher, dans un ordre bien précis ;
- quand Python exécute cette méthode, il **remplace dans notre chaîne {0} par la première variable passée à la méthode format (soit le prénom)**, {1} par la deuxième variable… et ainsi de suite.

On peut utiliser la méthode format pour formater des chaînes :

```
nouvelle_chaine = "Je m'appelle {0} {1} et j'ai {2} ans.".format(prenom, nom, age)
```

On peut placer les variables dans l'ordre que l'on veut :

```
prenom = "Jean"
nom = "Durant"
age = 21
print("Je m'appelle {0} {1} ({3} {0} pour l'administration) et j'ai {2} ans.".format(prenom, nom,
age, nom.upper()))
```

```
Je m'appelle Jean Durant (DURANT Jean pour l'administration) et j'ai 21 ans.
```

on ne précise pas forcément le numéro de la variable entre accolades :

```
date = "Dimanche 24 juillet 2011"
heure = "17:00"
print("Cela s'est produit le {}, à {}.".format(date, heure))
```

```
Cela s'est produit le Dimanche 24 juillet 2011, à 17:00.
```

Mais cela ne fonctionne que si vous donnez les variables dans le bon ordre dans format

La concaténation de chaînes

La concaténation cherche à regrouper deux chaînes en une, en mettant la seconde à la suite de la première.

Le signe « + » est le signe utilisé pour **concaténer** deux chaînes

```
prenom = "Jean"
message = "Bonjour"
chaine_complete = message + " " +  prenom # On utilise le symbole '+' pour concaténer deux chaînes
print(chaine_complete)
```

```
Bonjour Jean
```

Si vous voulez concaténer des chaînes et des nombres, il faudra **convertir** les nombres en chaînes pour pouvoir le concaténer aux autres chaînes, sinon vous obtiendrez une erreur

```
age = 21
message = "J'ai " + str(age) + " ans."
print(message)
```

```
J'ai 21 ans.
```

On appelle str pour convertir un objet en une chaîne de caractères, comme nous avons appelé int pour convertir un objet en entier

Vérifier si un élément existe dans une chaîne de caractères

Pour déterminer si un élément spécifié est présent dans une chaîne de caractères, utilisez le mot-clé *in* :

```
prenom = "Jean"
if "p" in prenom :
    print("Oui, 'p' est dans le prénom")
```

```
Oui, 'p' est dans le prénom
```

Parcours de chaînes

Accéder aux caractères d'une chaîne

Pour accéder aux lettres constituant une chaîne, nous allons utiliser un indice : On précise entre crochets [] l'indice, c'est-à-dire la position du caractère auquel on souhaite accéder.

```
chaine[position_dans_la_chaine]
```

Par exemple, nous souhaitons sélectionner la première lettre d'une chaîne.

```
chaine = "Salut !"
print(chaine[0]) # Première lettre de la chaîne
print(chaine[2]) # Troisième lettre de la chaîne
print(chaine[-1]) # Dernière lettre de la chaîne
```

```
S
l
!
```

- On commence à compter à partir de 0. La première lettre est donc à l'indice 0.
- On peut accéder aux lettres en partant de la fin à l'aide d'un indice négatif. Quand vous tapez chaine[-1], vous accédez ainsi à la dernière lettre de la chaîne.

On peut obtenir la longueur de la chaîne (le nombre de caractères qu'elle contient) grâce à la fonction len.

```
chaine = "Salut"
len(chaine) # Résultat : 5
```

Méthode de parcours par while

on peut parcourir une chaîne grâce à la boucle while.

```
chaine = "Salut"
i = 0 # On appelle l'indice 'i' par convention
while i < len(chaine):
    print(chaine[i]) # On affiche le caractère à chaque tour de boucle
    i += 1
```

N'oubliez pas d'incrémenter i.

Si vous essayez d'accéder à un indice qui n'existe pas (par exemple 25 alors que votre chaîne ne fait que 20 caractères de longueur), Python lèvera une exception de type **IndexError**.

Méthode de parcours par For

on peut parcourir une chaîne grâce à la boucle for.

Par exemple, nous allons parcourir la séquence "Bonjour les amis"

```
chaine = "Bonjour !"
for lettre in chaine:
    print(lettre)
```

```
B
o
n
j
o
u
r

!
```

1. **l'interpréteur va créer une variable lettre** qui contiendra le premier élément de la chaîne (autrement dit, la première lettre).
2. la variable lettre prend successivement la valeur de chaque lettre contenue dans la chaîne de caractères (d'abord B, puis o, puis n…).
3. On affiche ces valeurs avec print et cette fonction revient à la ligne après chaque message, ce qui fait que toutes les lettres sont sur une seule colonne.

Notez bien qu'il est **inutile d'incrémenter la variable lettre**.

Python se charge de l'incrémentation, c'est l'un des grands avantages de l'instruction for.

Sélection de chaînes

La sélection consiste à extraire une partie de la chaîne.

Cette opération renvoie le morceau de la chaîne sélectionné, sans modifier la chaîne d'origine.

```
chaine[indice_debut:indice_fin]
```

Par exemple

```
presentation = "salut"
presentation[0:2] # On sélectionne les deux premières lettres : 'sa'
presentation[2:len(presentation)] # On sélectionne la chaîne sauf les deux 1ères lettres : 'lut'
```

on peut sélectionner du début de la chaîne jusqu'à un indice, ou d'un indice jusqu'à la fin de la chaîne, sans préciser autant d'informations :

```
presentation[:2] # Du début jusqu'à la troisième lettre non comprise : 'sa'
presentation[2:] # De la troisième lettre (comprise) à la fin : 'lut'
```

on peut sélectionner à partir de la fin de la chaîne:

```
presentation[-1] # 't'
```

On peut constituer une nouvelle chaîne, en remplaçant une lettre par une autre :

```
mot = "lac"
mot = "b" + mot[1:]
print(mot)
```

```
bac
```

Pour rechercher/remplacer des lettres, nous avons aussi à notre disposition les méthodes **count**, **find** et **replace** de la classe str, à savoir « compter », « rechercher » et « remplacer ».

Comparaisons de chaines de caractères

On peut comparer, comme pour les nombres, des chaines de caractères.

Le résultat de la comparaison est True ou False et peuvent donc s'utiliser comme condition avec if .

Voici les différentes comparaisons possibles :

- texte1 == texte2 : Renvoie True si les deux textes sont parfaitement identiques.
- texte1 != texte2 : Renvoie True si les deux textes ont au moins un caractère de différent.
- texte1 < texte2 : Renvoie True si le texte1 est strictement avant texte2 dans l'ordre lexicographique (l'ordre du dictionnaire).
- texte1 <= texte2 : Comme < mais les deux textes peuvent être les mêmes.
- texte1 > texte2 : Renvoie True si le texte1 est strictement après texte2 dans l'ordre lexicographique (l'ordre du dictionnaire).
- texte1 >= texte2 : Comme < mais les deux textes peuvent être les mêmes.

Pour ranger dans l'ordre lexicographique, on compare les deux premiers caractères de chaque texte. S'ils sont égaux, on compare le second etc.

Par exemple : "azerty"<"azfa"

Ça fonctionne aussi pour les nombres dans une chaine de caractères.

Par exemple : "1234"<"2". Car on regarde le premier caractère : "1"<"2" donc "1234"<"2".