Programmation Python

Guide pratique : tranches (slcing) et les chaîne de caractère Python et

Solemane Coulibaly (solemane.coulibaly@gmail.com)

decembre 2024

Les Listes et les tranches (slcing) en Python

1. Qu'est-ce qu'une Liste en Python ?

Une **liste** est une structure de données qui peut contenir plusieurs éléments (entiers, chaînes, objets, etc.).

Exemple :

```
nombres = [10, 20, 30, 40, 50]
```

2. Accès aux Éléments

Chaque élément a une **position (indice)**, en commençant par 0.

Élément	10	20	30	40	50
Indice	0	1	2	3	4

Exemple :

```
print(nombres[0])  # Affiche 10
print(nombres[2])  # Affiche 30
```

3. Les Tranches (slicing)

3.1 Syntaxe :

```
liste[start:end:step]
```

- **start** : l'indice de départ (inclus)
- **end** : l'indice de fin (exclu)
- **step** *(optionnel)* : le pas (par défaut 1)

Exemple :

```
nombres = [10, 20, 30, 40, 50, 60]
```

```
print(nombres[1:4])  # [20, 30, 40]
print(nombres[:3])   # [10, 20, 30]  (du début jusqu'à l'indice 3 exclu)
print(nombres[3:])   # [40, 50, 60]  (de l'indice 3 à la fin)
```

3.2 Slicing avec Pas

```
print(nombres[::2])  # [10, 30, 50] (1 élément sur 2)
print(nombres[1:5:2])  # [20, 40] (de l'indice 1 à 4 avec un pas de 2)
```

3.3 Tranches avec Indices Négatifs

- Un indice négatif compte **à partir de la fin**.
- `-1` est le dernier élément, `-2` l'avant-dernier, etc.

```
print(nombres[-3:])  # [40, 50, 60]
print(nombres[:-2])  # [10, 20, 30, 40]
print(nombres[::-1])  # [60, 50, 40, 30, 20, 10] (liste inversée)
```

3.4 Copie d'une Liste

```
copie = nombres[:]
```

Cette syntaxe permet de copier **tout le contenu** de la liste.

4. Opérations Utiles sur les Listes

4.1 Modification

```
nombres[2] = 100  # Remplace l'élément d'indice 2
```

4.2 Remplacement par Tranche

```
nombres[1:4] = [200, 300, 400]  # Remplace 20, 30, 40 par 200, 300, 400
```

4.3 Suppression par Tranche

```
del nombres[2:4]  # Supprime les éléments d'indice 2 et 3
```

5. Exemples Pratiques

Exemple 1 : Afficher tous les éléments sauf le premier et le dernier

```
liste = [5, 10, 15, 20, 25]
print(liste[1:-1])  # [10, 15, 20]
```

Exemple 2 : Inverser une liste avec slicing

```
liste = [1, 2, 3, 4, 5]
print(liste[::-1])  # [5, 4, 3, 2, 1]
```

Exemple 3 : Prendre un élément sur deux

```
nombres = [0, 1, 2, 3, 4, 5, 6]
print(nombres[::2])  # [0, 2, 4, 6]
```

Chaîne de caractère en Python

1. Définition

Une **chaîne de caractères** (*string* en anglais) est une **séquence de caractères**. En Python, on peut créer une chaîne avec :

- des **guillemets simples** `'...'`
- des **guillemets doubles** `"..."`

```
texte1 = 'Bonjour'
texte2 = "Python"
```

2. Opérations de Base sur les Chaînes

2.1. Affichage

```
print("Bonjour tout le monde")
```

2.2. Concaténation (assemblage)

```
prenom = "Ali"
nom = "Traore"
print(prenom + " " + nom)  # Résultat : Ali Traore
```

2.3. Répétition

```
mot = "Hey! "
print(mot * 3)  # Résultat : Hey! Hey! Hey!
```

3. Accès aux Caractères et Slicing

3.1. Accès par indice

```
chaine = "Python"
print(chaine[0])  # Résultat : 'P'
print(chaine[2])  # Résultat : 't'
```

3.2. Parcours d'une chaîne

```
for lettre in "Python":
    print(lettre)
```

3.3. Tranches (Slicing)

```
texte = "Programmation"
print(texte[0:4])  # 'Prog' (de l'indice 0 à 3)
print(texte[:6])   # 'Progra' (du début à l'indice 5)
print(texte[6:])   # 'mmation' (de l'indice 6 à la fin)
print(texte[::-1]) # Inversion de la chaîne
```

4. Fonctions Utiles sur les Chaînes

4.1. Longueur d'une chaîne

```
chaine = "Python"
print(len(chaine))  # Résultat : 6
```

4.2. Conversion de casse

```
phrase = "Bonjour"
print(phrase.upper())  # 'BONJOUR'
print(phrase.lower())  # 'bonjour'
print(phrase.capitalize())  # 'Bonjour'
```

4.3. Suppression des espaces

```
texte = "  Python  "
print(texte.strip())  # 'Python' (supprime les espaces)
```

4.4. Remplacement de caractères

```
chaine = "Bonjour"
print(chaine.replace("o", "0"))  # 'B0nj0ur'
```

4.5. Recherche dans une chaîne

```
texte = "Je programme en Python"
print("Python" in texte)  # True
print(texte.find("Python"))  # Retourne l'indice de départ : 16
print(texte.find("Java"))    # Retourne -1 si non trouvé
```

5. Découper une Chaîne (Split et Join)

5.1. Découper une phrase en mots

```
phrase = "Python est simple et puissant"
mots = phrase.split()  # ['Python', 'est', 'simple', 'et', 'puissant']
```

5.2. Fusionner une liste en chaîne

```
mots = ['Python', 'est', 'génial']
phrase = " ".join(mots)
print(phrase)  # 'Python est génial'
```

6. Les méthodes booléennes

```
chaine = "Python123"

print(chaine.isalpha())   # False (car il y a des chiffres)
print(chaine.isdigit())   # False (car il y a des lettres)
print(chaine.isalnum())   # True  (lettres et chiffres uniquement)
print(chaine.startswith("Py"))  # True
print(chaine.endswith("123"))   # True
```

7. Exemple Pratique : analyse de texte

Soit la fonction suivante :

```
def analyse_texte(phrase):
    print("Longueur de la phrase :", len(phrase))
    print("En majuscules :", phrase.upper())
    print("Nombre de mots :", len(phrase.split()))
    print("Phrase inversée :", phrase[::-1])
    if "Python" in phrase:
        print("Le mot 'Python' est présent.")
    else:
        print("Le mot 'Python' est absent.")

# Test
analyse_texte("Python est un langage puissant")
```

Opérations clés sur les chaînes :

Opération	Syntaxe
Longueur de la chaîne	`len(chaine)`
Accès par indice	`chaine[i]`
Tranche (slicing)	`chaine[start:end:pas]`
Concaténation	`chaine1 + chaine2`
Répétition	`chaine * n`
Majuscules/Minuscules	`chaine.upper()`
Remplacement	`chaine.replace(x, y)`
Suppression d'espaces	`chaine.strip()`
Recherche	`"mot" in chaine`
Découper en mots	`chaine.split()`
Fusionner une liste	`" ".join(liste)`

Exercice

Exercice 1 :

Écrire une fonction `mots_longs(phrase, taille)` qui prend une phrase et retourne la liste des mots dont la longueur est **supérieure ou égale** à `taille`.

Exemple :

```
mots_longs("Python est un langage très puissant", 5)
# Résultat attendu : ['Python', 'langage', 'puissant']
```

Exercice 2 :

Écrire une fonction `frequence_mots(phrase)` qui transforme une phrase en une **liste de mots** et retourne un **dictionnaire** contenant la fréquence d'apparition de chaque mot.

Exemple :

```
frequence_mots("Python est facile et Python est puissant")
# Résultat attendu : {'Python': 2, 'est': 2, 'facile': 1, 'et': 1, 'puissant': 1}
```

Exercice 3

Écrire une fonction `mot_plus_long(phrase)` qui retourne le mot le plus long dans une phrase et sa longueur.

Exemple :

```
mot_plus_long("La programmation en Python est fascinante")
# Résultat attendu : ('programmation', 13)
```

Exercice 4

Écrire une fonction `inverser_mots(phrase)` qui retourne une nouvelle phrase où **l'ordre des mots est inversé**.

Exemple :

```
inverser_mots("Je suis en train d'apprendre Python")
# Résultat attendu : "Python d'apprendre train en suis Je"
```

Exercice 5 : Nettoyer une phrase

Écrire une fonction `nettoyer_phrase(phrase)` qui :

- Enlève **tous les caractères spéciaux et les chiffres**
- Transforme la phrase en **minuscules**
- Retourne une **liste des mots propres**.

Exemple :

```
nettoyer_phrase("Python3 est !super, et très #rapide 2024")
# Résultat attendu : ['python', 'est', 'super', 'et', 'très', 'rapide']
```