le Slicing (Tranches) en Python

Enseignant : GHANEM

Définition

- Le **slicing** (tranches) est une technique qui permet d'extraire une partie d'une séquence (liste, chaîne, tuple, etc.) en spécifiant un début, une fin et un pas.

Syntaxe de base

- séquence[début:fin:pas]

séquence[début:fin:pas]

Paramètre	Signification	Valeur par défaut
début	Index de départ	0
fin	Index de fin (exclusif)	len(sequence)
pas	Pas/incrément	1

Exemples

1. Slicing basique

```
liste = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(liste[2:5])      # [2, 3, 4] (index 2 à 4)
print(liste[:4])       # [0, 1, 2, 3] (du début à index 3)
print(liste[6:])       # [6, 7, 8, 9] (de index 6 à la fin)
print(liste[:])        # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] (copie complète)
```

On veut inverser les éléments de la liste :

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

Donner le code

```
liste[::-1]
```

- Exercice :

  liste = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

On veut afficher la liste des nombres impairs en ordre décroissant:

[9, 7, 5, 3, 1]

Donner le code

liste[9:0:-2]

- **2. Slicing avec pas**

```
liste = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(liste[::2])      # [0, 2, 4, 6, 8] (un élément sur deux)
print(liste[1::2])     # [1, 3, 5, 7, 9] (éléments impairs)
print(liste[2:8:3])    # [2, 5] (de index 2 à 7, pas de 3)
```

- **3. Slicing négatif**

- Les **index négatifs** comptent **à partir de la fin** de la séquence :

```
mon_tuple =       ('a’,  'b’,   'c’,   'd’,   'e')
# Index positifs:    0     1     2      3      4
# Index négatifs:  -5   -4    -3     -2    -1
```

- **3. Slicing négatif**

Signification des index négatifs

- -1 = dernier élément
- -2 = avant-dernier élément
- -3 = troisième en partant de la fin
- etc.

- **3. Slicing négatif**

```
mon_tuple = ('a', 'b', 'c', 'd', 'e')

print(mon_tuple[-1])  # 'e' (dernier)
print(mon_tuple[-2])  # 'd' (avant-dernier)
print(mon_tuple[-3])  # 'c' (troisième depuis la fin)
print(mon_tuple[-4])  # 'b'
print(mon_tuple[-5])  # 'a'
```

- **3. Slicing négatif**

```
mon_tuple = ('a', 'b', 'c', 'd', 'e’)
    # Index:    -5   -4   -3   -2   -1

print(mon_tuple[-2:])  # ('d', 'e’)
```

- **Explication :**

-2: signifie : "**commence à l'index -2** et **va jusqu'à la fin**"

Index -2 = 'd'

Jusqu'à la fin = 'd' et 'e'

- **3. Slicing négatif**

Exemples :

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

print(liste[-1:])    # [50] (dernier élément)
print(liste[-3:])    # [30, 40, 50] (3 derniers)
print(liste[:-2])    # [10, 20, 30] (tout sauf les 2 derniers)
print(liste[-4:-1])  # [20, 30, 40] (index -4 à -2, -1 exclus)
```

- **3. Slicing négatif**

Un exemple Pour ne jamais oublier le principe:

```
sequence = ['A',    'B',      'C',   'D',   'E']
      # Index:   0       1         2      3      4
  # Négatifs: -5     -4       -3     -2     -1
```

- print(sequence[1:4])   # ['B', 'C', 'D']
- print(sequence[-4:-1]) # ['B', 'C', 'D'] (Même résultat !)

- **3. Slicing négatif**

EXRCICE

```
liste = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(liste[-3:])      # ??????
print(liste[:-3])      # ??????
print(liste[-5:-2])    # ?????
```

- **3. Slicing négatif**

Résultats

```
liste = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(liste[-3:])      # [7, 8, 9] (3 derniers éléments)
print(liste[:-3])      # [0, 1, 2, 3, 4, 5, 6] (tout sauf les 3 derniers)
print(liste[-5:-2])    # [5, 6, 7] (de l'avant-dernier au 5ème en partant de la fin)
```

- **4. Slicing avec pas négatif (inversion)**

```
liste = [0, 1, 2, 3, 4, 5]

print(liste[::-1])     # [5, 4, 3, 2, 1, 0] (inverse complet)
print(liste[4:1:-1])   # [4, 3, 2] (de index 4 à 2 en descendant)
print(liste[5:0:-2])   # [5, 3, 1] (de la fin au début, pas de -2)
```

Applications pratiques

Avec les chaînes de caractères

```
texte = "Bonjour le monde"

print(texte[0:7])      # "Bonjour"
print(texte[8:10])     # "le"
print(texte[::-1])     # "ednom el ruojnoB"
print(texte[::2])      # "Bnorl od"
```

- **Avec les tuples**

```
mon_tuple = ('a', 'b', 'c', 'd', 'e')

print(mon_tuple[1:4])  # ('b', 'c', 'd')
print(mon_tuple[-2:])  # ('d', 'e')
```

Cas d'utilisation avancés

```
# Extraire une sous-liste
jours = ["lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche"]
semaine_travail = jours[0:5]  # ["lundi", "mardi", "mercredi", "jeudi", "vendredi"]
```

```
# Inverser une chaîne
palindrome = "radar"
print(palindrome == palindrome[::-1])  # True
```

DÉFI PROGRAMMATION

- Y a-t-il parmi nous des étudiants compétents en langage C, capables d'écrire rapidement un code pour une idée simple ?
- **C'est parti !** Nous lançons le chronomètre :
- Un étudiant écrit un programme en **Python** sur papier
- Un autre écrit le même programme en **langage C** sur papier
- **Objectif :**
- Les deux codes doivent afficher les multiples de 3 compris entre 0 et 30.

```
a = list(range(31))
b= a[::3]
b
```

- Version avec l’utilisation de print (l’ autre version ne marche que en mode console)

```
a = list(range(31))
b= a[::3]
print(b)
```

Nous aborderons la signification et l'utilisation de la fonction range() dans les prochaines séances de cours.

Une autre utilisation de [:]

pour une « Copie superficielle »

- Faisons tout d'abord une copie classique pour comparer :

```
originale = [[1, 2], [3, 4]]
copie = originale  # copie et originale seront exactement la même chose

print("Avant modification:")
print(originale)
print(copie)
# Modifions la liste principale (ajout d'un élément)
copie.append([5, 6])
print("\nAprès modification de la liste principale:")
print(originale)
print(copie)
copie[0][0] = 999
print("\nAprès modification d'une sous-liste:")
print(originale)
print(copie)
```

Copie superficielle

```
originale = [[1, 2], [3, 4]]
copie = originale[:] # Copie superficielle
```

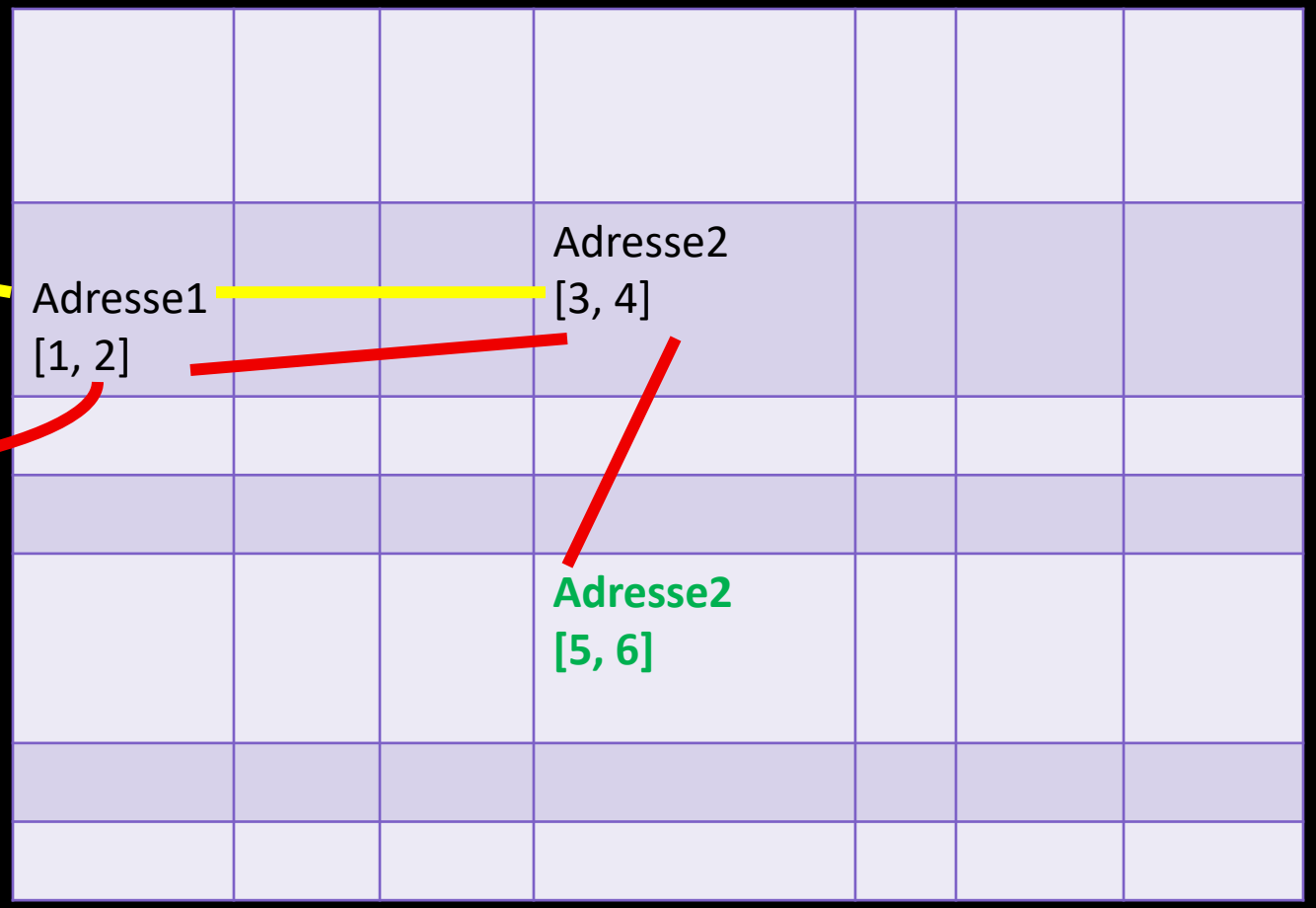
```
originale = [[1, 2], [3, 4]]
copie = originale[:]  # Copie superficielle

print("Avant modification:")
print(originale)  # [[1, 2], [3, 4]]
print(copie)      # [[1, 2], [3, 4]]

# Modifions la liste principale (ajout d'un élément)
copie.append([5, 6])
print("\nAprès modification de la liste principale:")
print(originale)  # [[1, 2], [3, 4]] ← inchangée
print(copie)      # [[1, 2], [3, 4], [5, 6]] changée

# Maintenant modifions une SOUS-liste
copie[0][0] = 999
print("\nAprès modification d'une sous-liste:")
print(originale)  # [[999, 2], [3, 4]] ← MODIFIÉE !
print(copie)      # [[999, 2], [3, 4], [5, 6]]
```

- **Pourquoi ça?**

En mémoire :

originale → [adresse1, adresse2]

copie → [adresse1, adresse2]



- **copie.append([5, 6])**