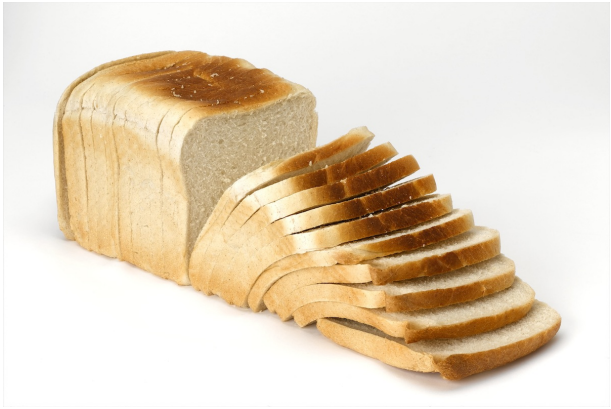
Les tranches en PYTHON

Le terme de *tranche* (*slice* en anglais) est associé à l'idée de *découpage*.



En PYTHON, une tranche permet le découpage de structures de données séquentielles, typiquement des chaînes de caractères, des listes ou des tuples. Ce sont des expressions qui permettent d'extraire des éléments d'une séquence *en une ligne de code*. Leur intérêt réside essentiellement dans la concision et la souplesse de leur syntaxe.

Tranches avec deux indices

Soit `S` une séquence, par exemple une chaîne de caractères ou une liste, une expression de la forme `S[4:16]` est une tranche de cette séquence. Cette syntaxe utilise deux indices, ici les indices 4 (indice de début) et 16 (indice de fin).

PYTHON

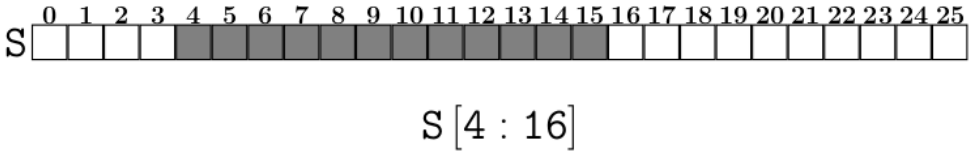
```
In [1]: alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

In [2]: alpha[4:16]
Out[2]: 'EFGHIJKLMNOP'
```

Ici, `alpha[4:16]` est une tranche et a pour valeur la chaîne extraite de la chaîne `alpha` dont les éléments sont situés :

- à droite de l'élément d'indice 4 ;
- *strictement* à gauche de l'élément d'indice 16, autrement dit jusqu'à l'indice 15 inclus.

Ce qui donne graphiquement :



✔

Si `i` et `j` sont des indices positifs, la syntaxe `S[i:j]` a pour valeur la séquence, de même nature que `S`, formée des éléments `S[k]` pour $k \in [\![i; j[\![$. Noter l'intervalle entier *semi-fermé* : le terme d'indice de droite n'est jamais inclus dans la tranche obtenue.

PYTHON

```
In [3]: L = [65, 31, 9, 32, 81, 82, 46, 12]

In [4]: L[2:6]
Out[4]: [9, 32, 81, 82]
```

Une tranche peut être vide :

PYTHON

```
In [5]: alpha[10:8]
Out[5]: ''
```

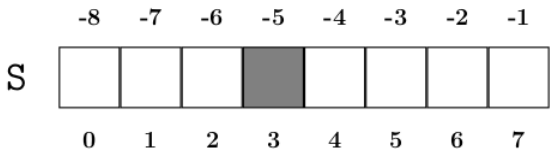
Les tranches s'appliquent essentiellement à des chaînes, des listes et des tuples. Étant construits à partir d'indices entiers, ils n'ont pas de sens pour des structures de données non ordonnées comme les dictionnaires ou les ensembles.

On ne met pas d'espaces autour du deux-points, sauf dans certains cas, voir :

https://sup3.prepa-carnot.fr/pep8

Indices négatifs

Un indice strictement négatif dans une séquence permet d'accéder à la séquence à partir de la fin. Par exemple, `S[-5]` désigne le 5e élément de `S` à partir de la fin. C'est comme si on écrivait `S[len(S) - 5]` mais avec `len(S)` implicite.



$$S[-5] = S[3]$$

PYTHON

```
In [6]: L[-1]  # Le dernier élément
Out[6]: 12

In [7]: L[-2]  # L'avant-dernier élément
Out[7]: 46

In [8]: L[-5]  # Le 5ème élément avant la fin
Out[8]: 32
```

Les indices d'une tranche peuvent être des entiers négatifs, la signification étant la même que pour des indices positifs :

PYTHON

```
In [9]: alpha[5:-3]
Out[9]: 'FGHIJKLMNOPQRSTUVW'

In [10]: alpha[-5:24]
Out[10]: 'VWX'

In [11]: alpha[-5:-1]
Out[11]: 'VWXY'
```

Omission d'indices

Quand une tranche se réfère à une des deux extrémités de la séquence, un raccourci syntaxique permet d'omettre l'indice de début ou de fin. Par exemple, la tranche `S[:j]` est synonyme de `S[0:j]` (que l'indice soit positif ou non). Il est même possible d'ignorer les deux indices. La tranche `S[:]` est alors défini comme `S[0:len(S)]`.

PYTHON

```
In [12]: alpha[:5]  # Les 5 premiers
Out[12]: 'ABCDE'

In [13]: alpha[-5:]  # Les 5 derniers
Out[13]: 'VWXYZ'

In [14]: alpha[5:]  # Tous sauf les 5 premiers
Out[14]: 'FGHIJKLMNOPQRSTUVWXYZ'

In [15]: alpha[:-5]  # Tous sauf les 5 derniers
Out[15]: 'ABCDEFGHIJKLMNOPQRSTU'

In [16]: alpha[:]
Out[16]: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

Tranches et copies

PYTHON

```
In [17]: M = L[5:]

In [18]: M[0] = -1

In [19]: M
Out[19]: [-1, 46, 12]

In [20]: L  # N'est pas modifiée
Out[20]: [65, 31, 9, 32, 81, 82, 46, 12]
```

⚠ Une tranche renvoie une *nouvelle* liste formée des éléments désignés.

✔ La tranche `L[:]` permet donc d'obtenir une copie d'une liste `L`, tout comme `list(L)`.

Tranches étendus

Comme pour `range`, la syntaxe des tranches admet un 3e paramètre : un *pas* (*step* en anglais).

PYTHON
```
In [21]: alpha[4:23:3]
Out[21]: 'EHKNQTW'
```

On obtient la chaîne de caractères formée des éléments de la chaîne `alpha` d'indices allant de 3 en 3 entre l'indice 4 inclus et l'indice 23 exclu.

$S[4:23:3]$

Le pas d'une tranche ne peut pas être nul, mais il peut être négatif. On compte alors de $-k$ en $-k$, *en arrière*, à partir de la position de départ. Par exemple :

PYTHON
```
In [22]: alpha[23:4:-3]
Out[22]: 'XUROLIF'
```

Comme pour les tranches avec deux indices, les indices peuvent être négatifs et/ou absents.

PYTHON
```
In [23]: alpha[::-5]   # De 5 en 5 à partir de la fin
Out[23]: 'ZUPKFA'
```

Un cas particulier très pratique est l'inversion de séquence avec une tranche de pas valant -1.

✔ `S[::-1]` renvoie la séquence complète inversée.

Application : On peut écrire très simplement une fonction de détection de palindrome (un mot qui ne change pas qu'on le lise de gauche à droite ou de droite à gauche) :

PYTHON
```
def est_palindrome(mot):
    """Vérifie si un mot est un palindrome."""
    return mot == mot[::-1]

In [24]: est_palindrome("semâmes")
Out[24]: True
```

Tableau des idiomes sur les tranches ♥

Action	Syntaxe	Valeur (`s = "ABCDEFGHIJKL"`)
Extraction	`s[2:7]`	`CDEFG`
Les 4 premiers	`s[:4]`	`ABCD`
Les 4 derniers	`s[-4:]`	`IJKL`
Tous sauf les 4 premiers	`s[4:]`	`EFGHIJKL`
Tous sauf les 4 derniers	`s[:-4]`	`ABCDEFGH`
Partitionner	`s[:3], s[3:7], s[7:]`	`('ABC', 'DEFG', 'HIJKL')`
De 3 en 3	`s[::3]`	`ADGJ`
De 3 en 3 à partir de la fin	`s[::-3]`	`LIFC`
Les indices pairs	`s[::2]`	`ACEGIK`
Les indices impairs	`s[1::2]`	`BDFHJL`
Copie superficielle	`s[:]`	`ABCDEFGHIJKL`
Copie à l'envers	`s[::-1]`	`LKJIHGFEDCBA`

Pour aller plus loin

Ce cours est adapté du *Tutoriel complet sur les slices en Python* de Pascal ORTIZ.

https://zestedesavoir.com/tutoriels/582/les-slices-en-python/

Les plus avancés pourront lire la deuxième partie : *Slices en écriture*.