Tranches

Bases de Python

Pourquoi des tranches ?

Une **tranche** est une partie d'une **séquence**, c'est-à-dire d'une liste, d'une chaîne de caractères ou d'un tuple, que l'on extrait en une seule écriture. En anglais, on parle de **slicing** (ou « tranchage »). C'est l'un des outils les plus utilisés du langage.

On coupe une tranche comme une tranche de pain dans une baguette : on choisit où commencer, où s'arrêter, et éventuellement on ne prend qu'une rondelle sur deux.

Souvent, on n'a besoin que d'une partie d'une liste : par exemple, les trois premières notes d'un élève. Avec une boucle, on les copie une par une dans une nouvelle liste :

```
notes = [12, 15, 8, 17, 14, 9]
premieres = []
for i in range(3):
    premieres.append(notes[i])
print(premieres)
```

```
[12, 15, 8]
```

Une tranche fait le même travail en une seule ligne : entre crochets, on indique où commencer et où s'arrêter.

```
premieres = notes[0:3]
print(premieres)
```

```
[12, 15, 8]
```

Rappel : les indices

Une tranche se repère grâce aux indices des éléments. Les indices positifs commencent à 0, et les indices négatifs comptent à partir de la fin : −1 pour le dernier élément, −2 pour l'avant-dernier.

```
L = ['a', 'b', 'c', 'd', 'e', 'f']
```

	'a'	'b'	'c'	'd'	'e'	'f'
indice	0	1	2	3	4	5
indice négatif	−6	−5	−4	−3	−2	−1

Ainsi `L[0]` vaut `'a'`, `L[3]` vaut `'d'`, `L[-1]` vaut `'f'` et `L[-2]` vaut `'e'`. L'indice `-1` désigne toujours le dernier élément, quelle que soit la longueur de la liste.

Extraire une tranche

Pour extraire une tranche, on écrit après le nom de la séquence, entre crochets, deux indices séparés par deux-points : l'indice de **début**, puis l'indice de **fin**.

```
L[début:fin]
```

— `début` est l'indice du premier élément pris : il est **inclus** ;

— `fin` est l'indice où la tranche s'arrête : il est **exclu**, comme avec la fonction `range()`.

Le résultat est toujours une **nouvelle séquence** du même type.

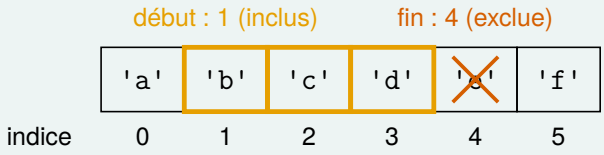
```
L = ['a', 'b', 'c', 'd', 'e', 'f']
print(L[1:4])
```

Question.
Quelles lettres contient la tranche `L[1:4]` ?

Réponse.

```
['b', 'c', 'd']
```

La tranche commence à l'indice `1`, qui est inclus : elle prend `'b'`, puis avance d'un indice à la fois, `'c'` à l'indice `2` et `'d'` à l'indice `3`. L'indice `4` est la fin : il est exclu, et la tranche s'arrête juste avant.



La fin exclue a un avantage : la tranche `L[i:j]` contient exactement `j - i` éléments. En mathématiques, elle correspond à l'intervalle $[i\ ;\ j[$: la borne de fin n'est **jamais incluse**.

Exemples de base

```
L = ['a', 'b', 'c', 'd', 'e', 'f']

print(L[1:4])    # indices 1, 2, 3
print(L[0:3])    # les 3 premiers
print(L[2:5])    # indices 2, 3, 4
```

```
['b', 'c', 'd']
['a', 'b', 'c']
['c', 'd', 'e']
```

Omettre le début ou la fin

On peut omettre un indice, ou les deux. Sans début, la tranche part du premier élément ; sans fin, elle va jusqu'au dernier élément, qui est alors inclus.

```
L = ['a', 'b', 'c', 'd', 'e', 'f']

print(L[:3])     # début omis : on part de l'indice 0
print(L[3:])     # fin omise : on va jusqu'à la fin
print(L[:])      # ni début ni fin : toute la liste
```

```
['a', 'b', 'c']
['d', 'e', 'f']
['a', 'b', 'c', 'd', 'e', 'f']
```

Les deux tranches `L[:3]` et `L[3:]` se complètent : la première prend les trois premiers éléments, la seconde tous les autres.

Règle générale :

— si `début` est omis, il vaut `0` (début de la séquence) ;

— si `fin` est omise, elle vaut `len(L)` (fin de la séquence) ;

— si `pas` est omis, il vaut `1` (le pas est présenté plus bas).

Copier une liste avec une tranche

Sans début ni fin, une tranche prend toute la liste : c'est une façon courante de **copier une liste**.

```
a = [1, 2, 3]
b = a[:]
b[0] = 99
print(a)
print(b)
```

Question.
La liste `a` est-elle modifiée ?

Réponse.

```
[1, 2, 3]
[99, 2, 3]
```

La liste `a` reste intacte, car une tranche est toujours une nouvelle liste : seule la copie `b` est modifiée. C'est une troisième façon de copier une liste, avec la méthode `copy()` et la fonction `list()` (voir le cours sur les listes).

Attention. comme `copy()`, la tranche `a[:]` ne copie que la liste principale. Pour une liste de listes, il faut la fonction `deepcopy()` du module `copy`.

Tranches et indices négatifs

Les indices négatifs fonctionnent aussi dans une tranche. Ils permettent de compter à partir de la fin, sans connaître la longueur de la liste.

```
L = ['a', 'b', 'c', 'd', 'e', 'f']
print(L[-3:])    # les 3 derniers
print(L[:-1])
```

Question.
Que donne la tranche `L[:-1]` ?

Réponse.

```
['d', 'e', 'f']
['a', 'b', 'c', 'd', 'e']
```

La tranche n'a pas de début : elle part du premier élément. Sa fin, `-1`, est exclue : elle s'arrête juste avant le dernier élément. On obtient donc toute la liste, sauf le dernier élément.

D'autres exemples :

```
L = ['a', 'b', 'c', 'd', 'e', 'f']
print(L[-4:-1])  # du 4e avant la fin jusqu'au dernier, exclu
print(L[1:-1])   # sans le premier ni le dernier
```

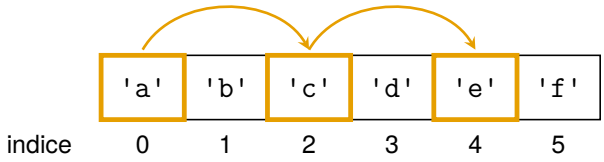
```
['c', 'd', 'e']
['b', 'c', 'd', 'e']
```

Sauter des éléments avec un pas

Une tranche peut aussi sauter des éléments. On ajoute alors un troisième nombre, après un second deux-points : le **pas**, c'est-à-dire l'écart entre deux indices pris. Sans pas, la tranche avance d'un indice à la fois.

```
L[début:fin:pas]
```

Avec un pas de 2, la tranche part de l'indice 0, puis saute à l'indice 2, puis à l'indice 4. L'indice suivant, 6, dépasse la fin de la liste : la tranche s'arrête là.



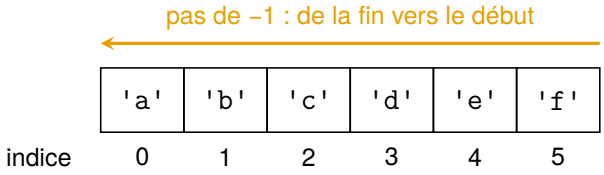

```
L = ['a', 'b', 'c', 'd', 'e', 'f']

print(L[::2])    # un sur deux, en partant du début
print(L[1::2])   # un sur deux, en partant de l'indice 1
print(L[::3])    # un sur trois
```

```
['a', 'c', 'e']
['b', 'd', 'f']
['a', 'd']
```

Pas négatif : lire à l'envers

Avec un pas négatif, la tranche avance dans l'autre sens, **de droite à gauche**. Avec un pas de `-1`, sans début ni fin, elle part du dernier élément et remonte jusqu'au premier : on obtient la séquence à l'envers.




```
L = ['a', 'b', 'c', 'd', 'e', 'f']

print(L[::-1])   # liste inversée
print(L[::-2])   # un sur deux, en sens inverse
```

```
['f', 'e', 'd', 'c', 'b', 'a']
['f', 'd', 'b']
```

```
print(L[4:1:-1])
```

Question.
Quelles lettres obtient-on avec `L[4:1:-1]` ?

Réponse.

```
['e', 'd', 'c']
```

La tranche part de l'indice `4`, puis descend d'un indice à la fois : elle prend `'e'`, puis `'d'`, puis `'c'`. L'indice `1` est la fin : comme toujours, il est exclu.

Valeurs par défaut avec un pas négatif :

— si `début` est omis, la tranche part du dernier élément ;

— si `fin` est omise, elle va jusqu'au premier élément inclus.

Élément ou tranche ?

Il ne faut pas confondre un indice seul et une tranche d'un seul élément.

```
L = ['a', 'b', 'c', 'd', 'e', 'f']

a = L[1]
b = L[1:2]
print(f"{type(a)} {a}")
print(f"{type(b)} {b}")
```

Question.
Les deux écritures `L[1]` et `L[1:2]` donnent-elles la même chose ?

Réponse.

```
<class 'str'> b
<class 'list'> ['b']
```

Pas du tout. Un indice seul donne l'**élément** lui-même : ici la chaîne de caractères `'b'`. Une tranche donne toujours une **nouvelle séquence**, ici une liste, même quand elle ne contient qu'un seul élément.

Les tranches d'une chaîne de caractères

Les tranches fonctionnent de la même façon sur les chaînes de caractères. Le résultat est alors une nouvelle chaîne.

```
mot = "informatique"

print(mot[:5])     # les 5 premiers caractères
print(mot[5:])     # à partir de l'indice 5
print(mot[::2])    # un caractère sur deux
print(mot[::-1])   # le mot à l'envers
```

```
infor
matique
ifraiu
euqitamrofni
```

Les tranches d'un tuple

Les tuples acceptent eux aussi les tranches, avec la même écriture. Le résultat est un nouveau tuple :

```
T = ('a', 'b', 'c', 'd', 'e')
print(T[1:4])
print(T[:3])
print(T[::2])
```

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

Une tranche donne donc toujours une séquence du même type : une liste pour une liste, une chaîne pour une chaîne, un tuple pour un tuple.

Remplacer une tranche d'une liste

Une liste peut être modifiée : on peut donc remplacer toute une tranche par d'autres éléments.

```
L = ['a', 'b', 'c', 'd', 'e', 'f']
L[2:4] = ['X', 'Y', 'Z']
print(L)
```

Question.
Combien d'éléments la liste contient-elle après ce remplacement ?

Réponse.

```
['a', 'b', 'X', 'Y', 'Z', 'e', 'f']
```

Sept. La tranche `L[2:4]` désigne deux éléments, `'c'` et `'d'`, remplacés par trois nouveaux : la liste passe de six à sept éléments, et les éléments suivants se décalent.

Attention. cela ne fonctionne que sur une liste. Une chaîne de caractères et un tuple ne se modifient pas.

```
mot = "python"
mot[0:1] = "P"
```

```
TypeError: 'str' object does not support item assignment
```

À retenir

1. Une tranche `L[début:fin]` extrait une partie d'une liste, d'une chaîne de caractères ou d'un tuple.
2. Le début est inclus, mais la fin est exclue : `L[1:4]` prend les indices 1, 2 et 3.
3. Sans début, la tranche part du premier élément ; sans fin, elle va jusqu'au dernier.
4. Une tranche sans début ni fin, `L[:]`, copie toute la liste.
5. Les indices négatifs comptent à partir de la fin : `L[-3:]` donne les trois derniers éléments, `L[:-1]` tous sauf le dernier.
6. Le pas fixe l'écart entre deux indices, `L[::2]`, et un pas négatif parcourt la séquence à l'envers, `L[::-1]`.
7. Un indice seul donne un élément ; une tranche donne toujours une nouvelle séquence du même type.
8. On peut remplacer une tranche d'une liste, mais pas celle d'une chaîne ou d'un tuple.

Résumé

Expression	Signification
`L[i:j]`	Éléments de l'indice `i` à `j-1`
`L[:j]`	Les `j` premiers éléments
`L[i:]`	De l'indice `i` à la fin (tous sauf les `i` premiers)
`L[:]`	Copie complète
`L[-k:]`	Les `k` derniers éléments
`L[::k]`	Un élément sur `k`
`L[::-1]`	Séquence inversée
`L[i:j:k]`	De `i` à `j-1`, par pas de `k`

Exercices

Exercice 1 Soit la liste `L = [10, 20, 30, 40, 50, 60, 70, 80]`. Sans exécuter le code, prédire le résultat de chaque expression, puis vérifier en Python :

```
L[2:5]
L[:4]
L[5:]
L[-3:]
L[1:7:2]
L[::-1]
```

Exercice 2 Soit la chaîne `phrase = "Numérique et Sciences Informatiques"`. En utilisant uniquement des tranches, extraire :

1. le mot « Numérique » ;
2. le mot « Informatiques » ;
3. les initiales « NSI ».

Exercice 3 Écrire une fonction `est_palindrome(mot)` qui renvoie `True` si le mot est un palindrome (se lit de la même façon dans les deux sens), en utilisant une tranche.

```
print(est_palindrome("kayak"))  # True
print(est_palindrome("python")) # False
```

Exercice 4 Écrire une fonction `rotation(L, k)` qui effectue une rotation de `k` positions vers la droite. Par exemple, `rotation([1, 2, 3, 4, 5], 2)` doit renvoyer `[4, 5, 1, 2, 3]`.

Indication : utiliser les tranches et la concaténation de listes.

Exercice 5 Soit la liste `L = [3, 8, 5, 12, 7, 4, 9, 16, 1, 6]`. Avec une tranche, extraire la sous-liste des éléments situés aux **positions impaires** (indices 1, 3, 5, 7, 9). Parmi ces éléments, lesquels sont pairs ?