

Listes Python, chaînes de caractères

Cours de CPGE PSI, 16 septembre 2026

Adeline Pierrot

Rappels sur les listes en python

- Une liste se définit par la donnée explicite de ses éléments entre crochets [].

- Ses éléments peuvent être de n'importe quel type :

```
>>> L = [1, 0.5, [1,2,3], "toto"]
```

- La fonction `len( )` renvoie le nombre d'éléments d'une liste :

```
>>> len(L)
```

```
4
```

- On accède aux éléments par leur indice, entre crochets.

Le premier élément a pour indice 0.

Le dernier élément a pour indice `len(L)-1` (ou simplement -1) :

```
>>> L[0]; L[-1]; L[len(L)]
```

```
1
```

```
"toto"
```

```
IndexError: list index out of range
```

Opérations sur les listes

- Contrairement aux t-uplets ou aux chaînes de caractères, les listes sont des objets dont on peut modifier un élément :

```
>>> L = [1, 0.5, [1,2,3], "toto"]
>>> L[0] = "changé"; print(L)
["changé", 0.5, [1,2,3], "toto"]
```

- L'opération + concatène deux listes ; une nouvelle liste est créée

```
>>> L + [3.14, 'a']
[1, 0.5, [1,2,3], "toto", 3.14, 'a']
>>> 2 * L
[1, 0.5, [1,2,3], "toto", 1, 0.5, [1,2,3], "toto"]
```

- Pour ajouter un élément à une liste, utiliser append:

```
>>> L.append(42)
[1, 0.5, [1,2,3], "toto", 42]
```

Attention, L.append(x) est une instruction, pas une expression.

Ne pas écrire ~~L=L.append(x)~~ !

Création de liste

- On peut mettre directement les éléments dans la liste:

```
>>> l1 = [4,-1,10,9,7]
```

- On peut partir de la liste vide et lui ajouter des éléments un par un avec append:

```
>>> l1=[]
```

```
for i in range(n):  
    l1.append(0)
```

- On peut aussi initialiser la liste à 0 ainsi:

```
>>> l1=[0]*n
```

- On peut enfin créer une liste "par compréhension" (comme un ensemble mathématique):

```
>>> listeCarres = [ i**2 for i in range(10) ]
```

```
>>> print(listeCarres)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Sous-liste

- On peut extraire une sous-liste d'une liste par un slicing :

`L[i:j]` est la tranche de `i` (inclus) à `j` (exclu)

`L[i:j:k]` est la tranche de `i` à `j` par pas de `k`

Une copie est effectuée.

```
>>> L = [1, 0.5, 1+2j, [1,2,3], "toto"]
```

```
>>> L[1:-1:2]  
[0.5, [1,2,3]]
```

```
>>> L[-1::-1]  
["toto", [1,2,3], 1+2j, 0.5, 1]
```

Copie de listes

- Attention à l'affectation de listes :

```
>>> 12 = L
>>> 12[-1] = 0 # Modification du dernier élément de 12
>>> print(12)
[1, 0.5, [1,2,3], 0]
>>> print(L)
[1, 0.5, [1,2,3], 0] # Modifier 12 a aussi changé L
L'affectation 12 = L a juste créé un alias : un nouveau nom
référant à la même liste, dont une seule copie figure en mémoire.
```

- Pour recopier effectivement la liste, utiliser plutôt le slicing L[:] :

```
>>> 13 = L.copy()    ou 13 = L[:]
>>> 13[-1] = 0 # Modification du dernier élément de 13
>>> print(13)
[1, 0.5, [1,2,3], 0]
>>> print(L)
[1, 0.5, [1,2,3], "toto"] # Modifier 13 n'a pas changé L
```

Listes de listes

```
L = [ [8, 24, -2, 0] , [7, 3] , [-5, 67, -1] ]
```

```
L[0] vaut [8,24,-2,0]
```

```
L[0][1] vaut 24
```

```
L[1][0] vaut 7
```

```
len(L) vaut 3
```

```
len(L[0]) vaut 4
```

```
L[1].append(9)  ⇒  L vaut maintenant
```

```
[ [8, 24, -2, 0] , [7, 3, 9] , [-5, 67, -1] ]
```

Parcours d'une liste de listes (par exemple pour remettre tous les éléments à 0):

```
for i in range(len(L)):
    for j in range(len(L[i])):
        L[i][j] = 0
```

Les chaînes de caractères: type str

- Permettent de stocker une suite de caractères: un mot, une phrase...
- Notées entre guillemets doubles (ou simples): "Bonjour", 'Z'
- Se comportent essentiellement comme des listes de caractères

```
>>> chaine = "Bonjour"
```

```
>>> len(chaine)
```

```
7
```

```
>>> chaine[3]
```

```
j
```

```
>>> 2 * chaine
```

```
BonjourBonjour
```

Les chaînes de caractères ne sont pas des objets partiellement modifiables:

```
>>> chaine[0] = 'Z'
```

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

Types séquentiels

Les types `int`, `float`, `bool` sont des types scalaires.

Les types `list` (listes) et `str` (chaînes de caractère) sont des structures de données de type séquentiel.

Tous les objets de type séquentiel ont en commun les opérations suivantes :

Opération	Résultat
<code>s[i]</code>	élément d'indice <code>i</code> de <code>s</code>
<code>s[i:j]</code>	Tranche de <code>i</code> (inclus) à <code>j</code> (exclus)
<code>s[i:j:k]</code>	Tranche de <code>i</code> à <code>j</code> par pas de <code>k</code>
<code>len(s)</code>	Longueur de <code>s</code>
<code>s+t</code>	Concaténation de <code>s</code> et <code>t</code>
<code>s*n, n*s</code>	Concaténation de <code>n</code> copies de <code>s</code>

où `s` et `t` sont des objets séquentiels de même type, et `i`, `j`, `k`, `n` sont des entiers.