**Chapitre 3 – Les Types de Conteneurs**

Ce chapitre m’a permis de découvrir les **différents types de conteneurs** en Python, essentiels pour structurer et manipuler les données. J’y ai appris à distinguer les listes, tuples, dictionnaires et ensembles, ainsi que leur fonctionnement.

**3.1 Les listes**

Les **listes** sont les conteneurs les plus utilisés. Elles peuvent contenir n’importe quel type d’objet (nombres, chaînes, même d’autres listes). Par exemple :

python

CopierModifier

L = ['a', 20.0, 5]

M = [3, ['a', -3.0, 5]]

L’**indexation commence à 0**, et on peut accéder à des éléments imbriqués comme M[1][2].

J’ai découvert le **slicing** (découpage) :

python

CopierModifier

L[1:5] # éléments entre les index 1 et 4

L[:3] # les 3 premiers éléments

L[-2:] # les deux derniers

On peut aussi spécifier un **pas (stride)** :

python

CopierModifier

L[::2] # un élément sur deux

L[::-1] # liste inversée

Il faut être attentif aux **index négatifs** et à l’utilisation de variables dans les tranches. Cela peut générer des résultats surprenants si on ne fait pas attention.

**3.1.2 Modification de listes**

J’ai vu comment **insérer ou supprimer** des éléments :

python

CopierModifier

L[2:3] = [] # suppression

L[1:1] = [1000, 5] # insertion

Et aussi comment **concaténer** ou **répéter** une liste :

python

CopierModifier

[1, 2] + [3, 4] # [1, 2, 3, 4]

[0] \* 4 # [0, 0, 0, 0]

**3.1.3 Appartenance à une liste**

On peut tester si un élément est présent :

python

CopierModifier

'a' in L

4 not in L

**3.1.4 Méthodes des listes**

Certaines méthodes modifient la liste directement :

python

CopierModifier

append, extend, insert, remove, sort, reverse, pop

D’autres renvoient une nouvelle valeur :

python

CopierModifier

count, copy

⚠️ Attention aux affectations comme newL = L.sort() qui retournent None !

**3.1.5 Fusion de listes – zip**

zip permet de **fusionner deux listes** en une liste de couples :

python

CopierModifier

zip(['a', 'b'], [1, 2]) # [('a', 1), ('b', 2)]

Le résultat est tronqué à la taille de la plus petite liste.

**3.1.6 Compréhensions de listes**

Syntaxe très puissante pour créer des listes :

python

CopierModifier

[x\*2 for x in L]

[x for x in L if x > 2]

On peut même faire des **boucles imbriquées** pour aplatir une matrice.

**3.2 Notion de tableau (array)**

Avec NumPy, on peut créer des **tableaux numériques homogènes** :

python

CopierModifier

v = array([1., 2., 3.])

M = array([[1, 2], [3, 4]])

Les **opérations sont élément par élément**, et on utilise @ ou dot pour les produits matriciels.

Les tableaux sont **moins flexibles que les listes** : pas de append, pas de changement de taille par slicing, mais ils sont beaucoup plus efficaces pour les calculs scientifiques.

**3.3 Les tuples**

Les **tuples** ressemblent aux listes, mais sont **immuables** :

python

CopierModifier

t = (1, 2, 3)

Ils sont pratiques pour **regrouper des valeurs** ou **retourner plusieurs résultats**.

On peut **dépaqueter** un tuple :

python

CopierModifier

a, b = (1, 2)

a, b = b, a # échange

**3.4 Les dictionnaires**

Les **dictionnaires** stockent des paires clé/valeur :

python

CopierModifier

d = {'mass': 5.7, 'name': 'wheel'}

Ils sont très utiles pour **organiser des données non ordonnées**.

On peut **boucler** sur les clés, les valeurs ou les items :

python

CopierModifier

for k in d.keys(), d.values(), d.items()

**3.5 Les ensembles (sets)**

Les **sets** sont comme les ensembles mathématiques : **pas de doublons**, **non ordonnés** :

python

CopierModifier

A = {1, 2, 3}

B = {3, 4}

A.union(B) # {1, 2, 3, 4}

A.intersection(B) # {3}

Il existe aussi issubset, issuperset, difference, etc.

Un set vide s’écrit set(), car {} crée un dictionnaire.

**3.6 Conversion entre conteneurs**

Il est fréquent de **convertir** entre types :

* Liste → tuple : tuple(L)
* Tuple → liste : list(t)
* Liste → set : set(L)
* Dictionnaire → liste de valeurs : d.values()

**3.7 Vérification du type**

On peut vérifier le type d’un objet :

python

CopierModifier

type(x), isinstance(x, list)

Il vaut mieux utiliser isinstance pour ne pas rater des sous-types (ex : bool est un sous-type de int).

**Ce que je retiens**

Ce chapitre m’a donné une **vision claire et structurée des conteneurs**. J’ai compris que chaque type (liste, tuple, dict, set) a ses particularités : certains sont ordonnés, d’autres non ; certains sont modifiables, d’autres immuables. Cette compréhension m’est indispensable pour bien organiser mes données et coder efficacement.