Chapitre 4 : Les types et structures de données en Python

Par Robert DIASSÉ

Dans ce chapitre, nous allons approfondir les **structures de données** que Python met à disposition pour gérer et manipuler efficacement des collections d'éléments. Les types de base (`int`, `float`, `str`, `bool`, `complex`) ont déjà été abordés dans les chapitres précédents, et nous les rappellerons brièvement avant de nous concentrer sur les structures plus avancées.

Plan du chapitre

---

1. Rappel des types de base

En Python, les types fondamentaux incluent :

- `int` **: Entiers**
- `float` **: Nombres à virgule flottante**
- `complex` **: Nombres complexes**
- `str` **: Chaînes de caractères**
- `bool` **: Booléens (`True` ou `False`)**

---

2. Introduction aux structures de données

Les structures de données permettent de **stocker, organiser et manipuler des collections** d'éléments. En Python, les principales sont :

- **Liste (`list`)** : Collection **ordonnée** et **modifiable**.

- **Tuple (`tuple`)** : Collection **ordonnée** mais **immuable**.
- **Ensemble (`set`)** : Collection **non ordonnée** et **sans doublons**.
- **Dictionnaire (`dict`)** : Collection de **paires clé-valeur**.

---

3. Les listes (`list`) et leurs fonctions

Une liste est une collection **ordonnée, modifiable, et hétérogène**. Elle est très utilisée en Python.

3.1 Caractéristiques principales

- Création : Utilisez des crochets `[]`.
- Modifiable : On peut ajouter, supprimer ou modifier des éléments.

3.2 Fonctions usuelles pour manipuler les listes

Fonction	Description	Exemple
`append(x)`	Ajoute un élément à la fin de la liste	`fruits.append("orange")`
`insert(i, x)`	Insère un élément à une position donnée	`fruits.insert(1, "kiwi")`
`remove(x)`	Supprime la première occurrence de l'élément `x`	`fruits.remove("kiwi")`
`pop(i)`	Supprime et retourne l'élément à l'indice `i`	`element = fruits.pop(2)`
`sort()`	Trie les éléments dans l'ordre croissant	`fruits.sort()`
`reverse()`	Inverse l'ordre des éléments	`fruits.reverse()`
`index(x)`	Retourne l'indice de la première occurrence de `x`	`indice = fruits.index("pomme")`
`count(x)`	Compte le nombre d'occurrences de `x`	`nb = fruits.count("pomme")`
`clear()`	Supprime tous les éléments de la liste	`fruits.clear()`
`copy()`	Retourne une copie superficielle de la liste	`nouvelle_liste = fruits.copy()`

3.3 Exemple

```
fruits = ["pomme", "banane", "cerise"]
fruits.append("orange")  # Ajout
fruits.insert(1, "kiwi")  # Insertion
fruits.remove("cerise")  # Suppression
fruits.sort()  # Tri
print(fruits)  # ['banane', 'kiwi', 'orange', 'pomme']
```

---

4. Les tuples (`tuple`) et leurs fonctions

Un tuple est une collection **ordonnée mais immuable**.

4.1 Caractéristiques principales

- Création : Utilisez des parenthèses `()`.
- Les tuples sont **plus rapides** et idéaux pour les données constantes.

4.2 Fonctions usuelles pour manipuler les tuples

Fonction	Description	Exemple
`index(x)`	Retourne l'indice de la première occurrence de `x`	`tup.index("rouge")`
`count(x)`	Compte le nombre d'occurrences de `x`	`tup.count(3)`

4.3 Exemple

```
couleurs = ("rouge", "vert", "bleu", "rouge")
print(couleurs.index("vert"))  # Affiche 1
print(couleurs.count("rouge"))  # Affiche 2
```

---

5. Les ensembles (`set`) et leurs fonctions

Un ensemble est une collection **non ordonnée** et **sans doublons**.

5.1 Caractéristiques principales

- Création : Utilisez des accolades `{}` ou la fonction `set()`.
- Utile pour les opérations ensemblistes.

5.2 Fonctions usuelles pour manipuler les ensembles

Fonction	Description	Exemple
`add(x)`	Ajoute un élément à l'ensemble	`s.add(5)`
`remove(x)`	Supprime l'élément `x` (erreur si absent)	`s.remove(3)`
`discard(x)`	Supprime l'élément `x` (sans erreur si absent)	`s.discard(10)`
`pop()`	Supprime et retourne un élément aléatoire	`element = s.pop()`
`clear()`	Vide l'ensemble	`s.clear()`
`union(s2)`	Retourne l'union avec un autre ensemble	`s.union(s2)`
`intersection(s2)`	Retourne l'intersection avec un autre ensemble	`s.intersection(s2)`
`difference(s2)`	Retourne la différence avec un autre ensemble	`s.difference(s2)`

5.3 Exemple

```
nombres = {1, 2, 3}
nombres.add(4)
nombres.remove(2)
autres = {3, 4, 5}
union = nombres.union(autres)
print(union)  # {1, 3, 4, 5}
```

---

6. Les dictionnaires (`dict`) et leurs fonctions

Un dictionnaire est une collection de **paires clé-valeur**.

6.1 Caractéristiques principales

- Création : Utilisez `{}` avec des paires clé-valeur.
- Clés uniques, valeurs modifiables.

6.2 Fonctions usuelles pour manipuler les dictionnaires

Fonction	Description	Exemple
`keys()`	Retourne une vue des clés	`d.keys()`
`values()`	Retourne une vue des valeurs	`d.values()`
`items()`	Retourne une vue des paires clé-valeur	`d.items()`
`get(key, def)`	Retourne la valeur associée à `key` ou `def` si absent	`d.get("nom", "inconnu")`
`pop(key)`	Supprime et retourne la valeur associée à `key`	`val = d.pop("nom")`
`update(d2)`	Met à jour le dictionnaire avec un autre dictionnaire	`d.update(d2)`

6.3 Exemple

```
etudiant = {"nom": "Alice", "age": 22}
etudiant["classe"] = "Mathématiques"
print(etudiant.get("nom", "Inconnu"))  # Affiche 'Alice'
```

7. Conversions entre types et structures de données

Python fournit des fonctions intégrées pour **convertir des structures de données**. Ces conversions sont utiles lorsque les données doivent être manipulées différemment en fonction des besoins.

7.1 Conversions possibles

Type source	Type cible	Méthode utilisée	Exemple

Type source	Type cible	Méthode utilisée	Exemple
`list`	`tuple`	`tuple(liste)`	`t = tuple([1, 2, 3])`
`tuple`	`list`	`list(tuple)`	`l = list((1, 2, 3))`
`list` ou `tuple`	`set`	`set(liste)`	`s = set([1, 2, 3])`
`set`	`list`	`list(set)`	`l = list({1, 2, 3})`
`dict`	`list`	`list(dictionnaire.keys())`	`l = list({"nom": "Alice", "age": 22}.keys())`
`list` ou `tuple`	`str`	`str(sequence)`	`s = str([1, 2, 3])`
`str`	`list` ou `tuple`	`list(chaine)` ou `tuple(chaine)`	`l = list("abc") # ['a', 'b', 'c']`

7.2 Exemple pratique de conversions

Supposons que nous avons une liste et que nous souhaitons obtenir différentes structures de données à partir de celle-ci.

```
# Liste initiale
nombres = [1, 2, 3, 4, 5]

# Conversion en tuple
tup = tuple(nombres)

# Conversion en ensemble
ens = set(nombres)

# Conversion en chaîne de caractères
chaine = str(nombres)

# Résultats
print("Tuple :", tup)       # (1, 2, 3, 4, 5)
print("Ensemble :", ens)    # {1, 2, 3, 4, 5}
print("Chaîne :", chaine)   # "[1, 2, 3, 4, 5]"
```

---

8. Exercices

Pour consolider vos connaissances, voici quelques exercices :

Exercice 1 : Manipulation de listes

1. Créez une liste de 5 fruits.

2. Ajoutez un fruit à la fin.
3. Insérez un fruit au début.
4. Supprimez un fruit par son nom.
5. Inversez l'ordre des fruits.
6. Affichez la liste finale.

Exercice 2 : Opérations sur les ensembles

1. Créez deux ensembles : `ens1 = {1, 2, 3, 4}` et `ens2 = {3, 4, 5, 6}`.
2. Trouvez l'union, l'intersection et la différence de ces deux ensembles.
3. Supprimez un élément de `ens1`.
4. Ajoutez un nouvel élément à `ens2`.

Exercice 3 : Création et manipulation de dictionnaires

1. Créez un dictionnaire représentant un étudiant avec les clés : `nom`, `age`, `classe`.
2. Ajoutez une clé `note` avec une valeur.
3. Modifiez la valeur de `classe`.
4. Supprimez la clé `note`.
5. Affichez toutes les clés, les valeurs et les paires clé-valeur.

Bonnes pratiques

Pour clore le chapitre, voici quelques **bonnes pratiques** pour manipuler les structures de données efficacement :

- **Évitez les modifications répétées sur un tuple.** Les tuples étant immuables, chaque modification nécessite de recréer un objet.
- **Utilisez des ensembles pour les recherches rapides.** Ils offrent une complexité moyenne en O(1) pour les ajouts, suppressions et recherches.
- **Optimisez les dictionnaires.** Ils sont idéaux pour associer des clés uniques à des valeurs, mais les clés doivent être immuables (par exemple, chaînes ou tuples).
