Structures de données en Python

Pierre Jaumier

Bienvenue – Structures de données

Structures de données en Python

- Objectif : Apprendre à utiliser les structures de données essentielles
- Pierre Jaumier

> Aujourd'hui, on va découvrir : - Les listes, tuples, dictionnaires et ensembles - Comment les manipuler efficacement - Quand choisir l'une plutôt que l'autre

Pourquoi les structures de données ?

> "Les structures de données sont les briques de base de tout programme."

Elles permettent de : - Organiser des informations - Faciliter leur manipulation - Gagner en performance

Exemples : - Liste de courses → `list` - Coordonnées GPS → `tuple` - Profil utilisateur → `dict` - Catégories d'objets → `set`

Liste – Création et accès

```
# Création d'une liste
fruits = ['pomme', 'banane', 'orange']

# Accès par index
print(fruits[0])   # 'pomme'
print(fruits[-1])  # 'orange'

# Modification
fruits[1] = 'kiwi'
```

Le premier élément est à l'index `0`.

Liste – Opérations courantes

```
# Ajout
fruits.append('mangue')       # ajout à la fin
fruits.insert(1, 'citron')    # insertion à la position 1

# Suppression
fruits.remove('citron')       # suppression par valeur
del fruits[0]                 # suppression par index

# Tri
fruits.sort()
```

Liste – Slicing

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

# Syntaxe : liste[start:end:step]
print(liste[1:3])     # ['b', 'c']
print(liste[:3])      # ['a', 'b', 'c']
print(liste[2:])      # ['c', 'd', 'e']
print(liste[::-1])    # ['e', 'd', 'c', 'b', 'a'] (inversion)
```

Les List Comprehensions en Python

> En Python, les **list comprehensions** ou création de listes à la volée sont une façon concise de créer des listes.

Elles permettent de transformer ou filtrer des données **en une seule ligne**.

Avant (avec boucle `for`) :

```
carres = []
for x in range(5):
    carres.append(x**2)
```

Après (avec list comprehension) :

```
carres = [x**2 for x in range(5)]
```

Syntaxe basique

Structure :

```
[nouvelle_expression for élément in itérable]
```

Exemples :

```
# Carrés des nombres de 0 à 4
[x**2 for x in range(5)]  #  [0, 1, 4, 9, 16]

# Longueurs des mots d'une liste
[len(mot) for mot in ['chat', 'chien', 'oiseau']]  #  [4, 5, 6]
```

Ajouter une condition (`if`)

Filtrage avec `if` :

```
[nouvelle_expression for élément in itérable if condition]
```

Exemple :

```
# Nombres pairs entre 0 et 9
[x for x in range(10) if x % 2 == 0]  #  [0, 2, 4, 6, 8]
```

Exercices rapides

Exercice 1 : Créez une liste des lettres majuscules du mot `'python'`.

Exercice 2 : Filtrez les mots dont la longueur est supérieure à 3 dans `['un', 'deux', 'trois', 'quatre']`.

Exercice 3 : Créez une liste avec `'pair'` ou `'impair'` pour chaque nombre de 0 à 4.

Correction des exercices

Exercice 1 :

```
[lettre.upper() for lettre in 'python']  #  ['P', 'Y', 'T', 'H', 'O', 'N']
```

Exercice 2 :

```
[mot for mot in ['un', 'deux', 'trois', 'quatre'] if len(mot) > 3]
#  ['deux', 'trois', 'quatre']
```

Exercice 3 :

```
['pair' if x % 2 == 0 else 'impair' for x in range(5)]
#  ['pair', 'impair', 'pair', 'impair', 'pair']
```

Bonnes pratiques

Utilisez les list comprehensions : - Quand le code est court et clair - Pour transformer ou filtrer des données

Évitez-les quand : - Il y a trop de conditions imbriquées - La logique est complexe

Lisibilité avant concision !

Tuple – Création et utilisation

```
# Création
point = (10, 20)

# Accès
x, y = point
print(x)  # 10

# Utilisation commune : retour multiple de fonction
def divmod(a, b):
    return a // b, a % b

resultat = divmod(10, 3)  # (3, 1)
```

Tuple vs Liste

Caractéristique	Liste	Tuple
Modifiable ?	Oui	Non
Performance	Moins rapide	Plus rapide
Sécurité	Moins sûr	Plus sûr
Utilisation	Données variables	Données fixes

Exemple pratique :

```
# Tuple utilisé comme clé dans un dictionnaire
positions = {('Paris', 'Lyon'): 450}
```

Dictionnaire – Création et accès

```
# Création
personne = {'nom': 'Alice', 'âge': 25}

# Accès
print(personne['nom'])  # 'Alice'

# Ajout/modification
personne['ville'] = 'Paris'  # ajout
personne['âge'] = 30         # modification
```

Dictionnaire – Méthodes utiles

```
# .keys(), .values(), .items()
for cle in personne.keys():
    print(cle)

for valeur in personne.values():
    print(valeur)

for cle, valeur in personne.items():
    print(f'{cle}: {valeur}')
```

```python
# Vérifier existence
if 'ville' in personne:
    print('La ville est connue')
```

Dict comprehensions

> En Python, les **dict comprehensions** permettent de créer des dictionnaires **de façon élégante et rapide**, comme les list comprehensions mais avec une structure adaptée.

Avant (avec boucle `for`) :

Structure :

```python
{nouvelle_clé: nouvelle_valeur for élément in itérable}
```

Exemples :

```python
# Carrés des nombres de 0 à 4
{x: x**2 for x in range(5)}
#   {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

# Dictionnaire à partir d'une liste de mots
{mot: len(mot) for mot in ['chat', 'chien', 'oiseau']}
#   {'chat': 4, 'chien': 5, 'oiseau': 6}
```

Ensembles – Création et unicité

```python
# Création
set1 = {'pomme', 'banane', 'orange'}
set2 = set(['pomme', 'poire', 'pomme'])

print(set2)  # {'poire', 'pomme'} (doublons supprimés)
```

Utile pour : - Éliminer les doublons - Rechercher rapidement si un élément existe

Ensembles – Opérations mathématiques

```python
set1 = {'pomme', 'banane', 'orange'}
set2 = {'banane', 'kiwi'}

# Union
print(set1 | set2)  # {'pomme', 'banane', 'orange', 'kiwi'}

# Intersection
print(set1 & set2)  # {'banane'}

# Différence
print(set1 - set2)  # {'pomme', 'orange'}

# Appartenance
print('pomme' in set1)  # True
```

Comparaison des structures

Structure	Mutable	Ordonné	Indexable	Unicité
Liste	Oui	Oui	Oui	Non
Tuple	Non	Oui	Oui	Non
Dictionnaire	Oui	Oui (3.7+)	Non (par clé)	Non (clé unique)
Ensemble	Oui	Non	Non	Oui

Choisissez selon vos besoins !

Exercices guidés

Exercice 1 : Liste - Créez une liste de 5 films préférés - Triez-la par ordre alphabétique - Affichez le 2e film

Exercice 2 : Dictionnaire - Créez un dictionnaire avec votre nom, âge et ville - Affichez chaque information sur une ligne séparée

Exercice 3 : Ensemble - Comparez deux listes de fruits - Trouvez les fruits communs aux deux listes

Conclusion & Questions

Ce qu'on a appris aujourd'hui :

- Manipuler des listes, tuples, dictionnaires et ensembles
- Comprendre leurs différences et cas d'usage
- Choisir la bonne structure selon le besoin

Des questions ?

Prochaines étapes

- Introduction aux fonctions
- Manipulation de fichiers
- Introduction aux modules standard