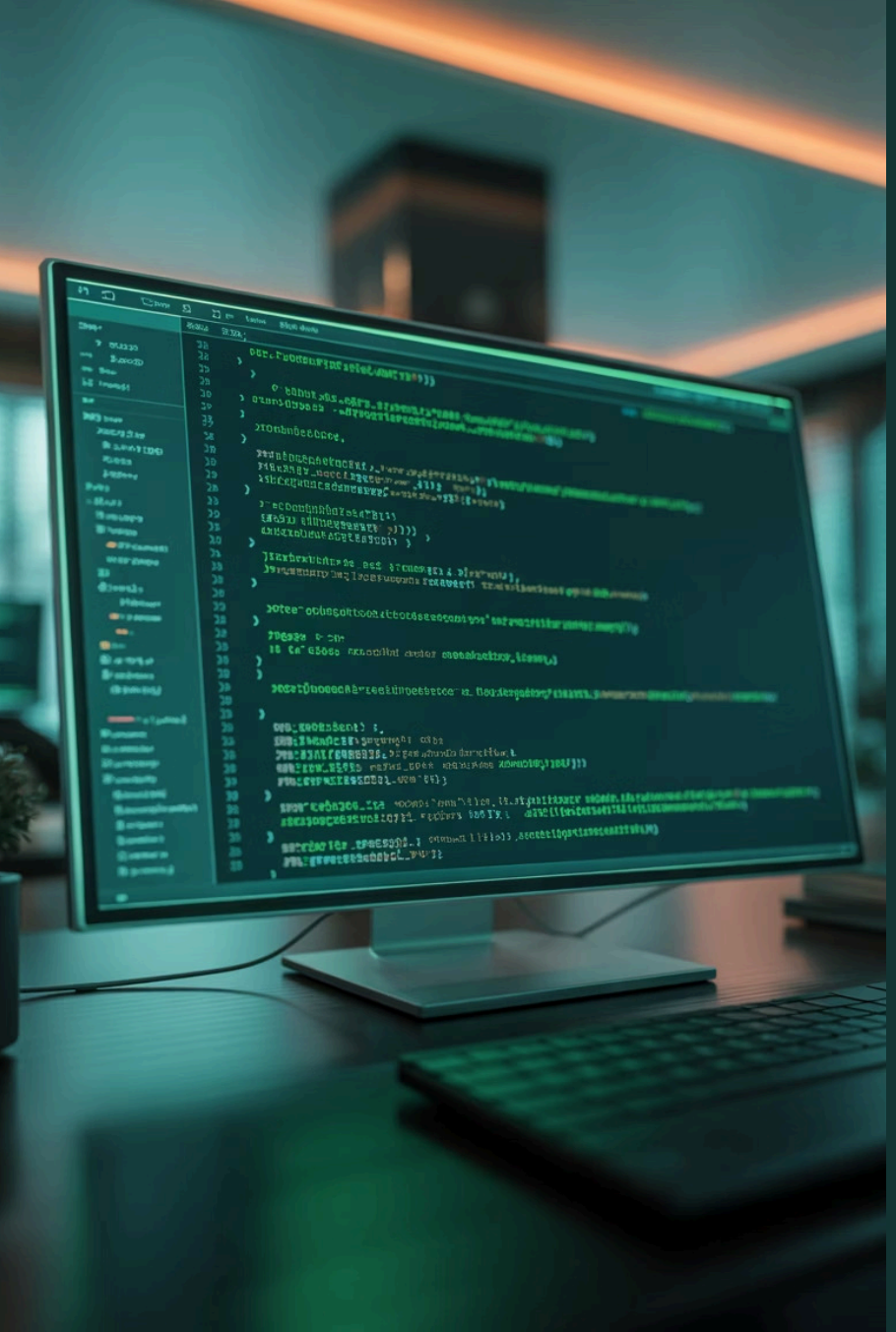

Les Sets en Python

Formation BTS SIO SLAM 2ème année - Structure de données avancée pour la gestion d'éléments uniques

Cours de David DONISA , Enseignant BTS SIO

Qu'est-ce qu'un Set ?

Définition

Un **set** est une collection non ordonnée d'éléments uniques en Python. C'est une structure de données puissante qui élimine automatiquement les doublons.

Contrairement aux listes, les sets ne conservent pas l'ordre d'insertion et n'acceptent qu'une seule occurrence de chaque valeur.

Caractéristiques principales

- Éléments **uniques** seulement
- Non ordonné (pas d'indexation)
- Modifiable (ajout/suppression)
- Performant pour les tests d'appartenance
- Idéal pour les opérations ensemblistes

Création d'un Set

À partir d'une liste

```
ma_liste = [1, 2, 2, 3, 3, 3]
mon_set = set(ma_liste)
print(mon_set)
# {1, 2, 3}
```

Transforme une liste en éliminant les doublons automatiquement

Set vide

```
mon_set_2 = set()
print(mon_set_2)
# set()
```

Utilisation du constructeur pour créer un ensemble vide

Avec valeurs initiales

```
mon_set_3 = {1, 4, 6}
print(mon_set_3)
# {1, 4, 6}
```

Syntaxe directe avec accolades pour initialiser des valeurs

Propriétés Fondamentales

Unicité des éléments

Chaque élément n'apparaît qu'**une seule fois** dans le set. Les tentatives d'ajout d'un élément déjà présent sont ignorées silencieusement.

```
mon_set = {1, 2, 3}
mon_set.add(2)
print(mon_set)  # {1, 2, 3}
```

Non-ordonnancement

Les éléments n'ont pas d'ordre défini. On ne peut pas accéder aux éléments par index comme avec les listes.

```
# mon_set[0] génère une erreur
# Pas d'accès par position
```

Mutabilité

Les sets peuvent être modifiés après leur création : ajout et suppression d'éléments sont possibles à tout moment.

Ajout et Suppression d'Éléments

Ajouter des éléments

Méthode add()

```
mon_set = set()
mon_set.add(3)
mon_set.add('a')
mon_set.add('a')  # Ignoré
print(mon_set)
# {3, 'a'}
```

Ajoute un élément unique au set. Les doublons sont automatiquement ignorés.

Méthode update()

```
mon_set_a = {1, 2, 3}
mon_set_b = {4, 5}
mon_set_a.update(mon_set_b)
# {1, 2, 3, 4, 5}
```

Fusionne plusieurs ensembles ensemble

Supprimer des éléments

Méthode remove()

```
mon_set.remove(25)
# KeyError si absent
```

Attention : génère une erreur si l'élément n'existe pas

Méthode discard()

```
mon_set.discard(25)
# Pas d'erreur
```

Recommandé : supprime l'élément sans erreur s'il est absent

Méthode pop()

```
element = mon_set.pop()
# Retire aléatoirement
```

Retire un élément arbitraire du set

Opérations Mathématiques sur les Sets

Union

```
set_a = {1, 2, 3, 4, 5, 6}
set_b = {5, 6, 7, 8, 9, 10}
set_a.union(set_b)
# {1,2,3,4,5,6,7,8,9,10}
```

Tous les éléments des deux sets

Différence symétrique

```
set_a.symmetric_difference(set_b)
# {1,2,3,4,7,8,9,10}
```

Éléments dans l'un ou l'autre, mais pas les deux

Intersection

```
set_a.intersection(set_b)
# {5, 6}
```

Éléments communs aux deux sets

Différence

```
set_a.difference(set_b)
# {1, 2, 3, 4}
```

Éléments dans set_a mais pas dans set_b

Méthodes de Comparaison

1

isdisjoint()

Teste s'il n'y a **aucun élément commun** entre deux sets

```
set_a = {1, 2, 3, 4, 5, 6}
set_b = {5, 6, 7, 8, 9, 10}
print(set_a.isdisjoint(set_b))
# False (5 et 6 en commun)

print({25,
50}.isdisjoint(set_b))
# True (aucun élément
commun)
```

2

issubset()

Vérifie si un set est un sous-ensemble d'un autre

```
print({1, 2, 3}.issubset({1, 2, 3,
4, 5, 6}))
# True
```

Tous les éléments du premier set sont dans le second

3

issuperset()

Teste si un set **contient** tous les éléments d'un autre

```
print({1, 2, 3, 4, 5,
6}.issuperset({1, 2, 3}))
# True
```

Le premier set inclut tous les éléments du second

Comparaison avec d'Autres Structures

Caractéristique	Set	Liste	Dictionnaire
Ordre	Non ordonné	Ordonné	Ordonné (Python 3.7+)
Doublons	❌ Non autorisés	✅ Autorisés	Clés uniques
Indexation	❌ Impossible	✅ Par position	✅ Par clé
Mutabilité	✅ Modifiable	✅ Modifiable	✅ Modifiable
Performance recherche	O(1) - Excellent	O(n) - Moyen	O(1) - Excellent
Opérations ensemblistes	✅ Natives	❌ Absentes	❌ Absentes

Cas d'Usage Pratiques

Élimination des doublons

```
emails = ['a@x.com', 'b@x.com',
'a@x.com']
emails_uniques = list(set(emails))
# ['a@x.com', 'b@x.com']
```

Nettoyage rapide de données

Tests d'appartenance rapides

```
admins = {'alice', 'bob', 'charlie'}
if user in admins:
    print("Accès autorisé")
```

Vérification O(1) ultra-performante

Comparaison de listes

```
clients_2023 = {'A', 'B', 'C'}
clients_2024 = {'B', 'C', 'D'}
nouveaux = clients_2024 -
clients_2023
# {'D'}
```

Analyse de données comparatives

Les sets sont particulièrement utiles pour gérer des **collections uniques**, effectuer des comparaisons mathématiques, et optimiser les performances de recherche dans vos programmes Python.

Récapitulatif et Bonnes Pratiques

1 **Utilisez les sets pour garantir l'unicité**

Quand vous devez éliminer les doublons automatiquement, le set est la structure idéale

2 **Privilégiez discard() à remove()**

Pour éviter les erreurs lors de la suppression d'éléments potentiellement absents

3 **Exploitez les opérations ensemblistes**

Union, intersection et différence sont natives et très performantes avec les sets

4 **Attention à l'absence d'ordre**

Si l'ordre est important, utilisez une liste. Les sets ne garantissent aucun ordonnancement

Point clé : Les sets offrent des performances exceptionnelles pour les tests d'appartenance (O(1)) et sont essentiels pour les opérations mathématiques sur des ensembles de données.