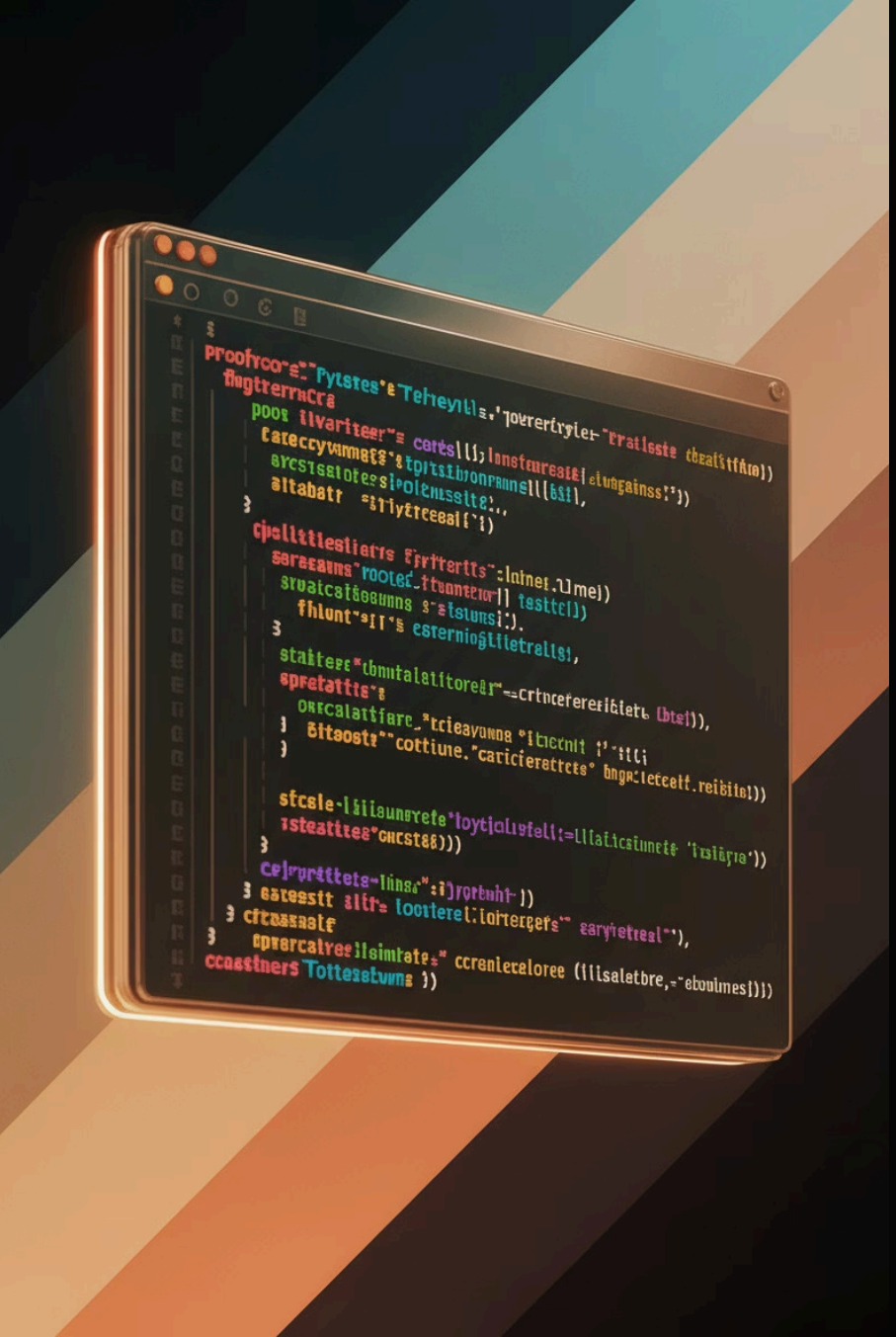


Les Dictionnaires et Fichiers JSON en Python

Maîtrisez les structures de données clés et la manipulation de fichiers JSON pour vos projets SLAM

Cours de David DONISA , Enseigant BTS SIO

Introduction aux Dictionnaires Python

Qu'est-ce qu'un dictionnaire ?

Un dictionnaire est une structure de données Python qui stocke des paires **clé-valeur**. C'est l'équivalent d'un objet JavaScript ou d'une HashMap en Java. Chaque clé est unique et permet d'accéder rapidement à sa valeur associée.

Les dictionnaires sont **mutables**, **non ordonnés** (avant Python 3.7) et extrêmement performants pour la recherche de données. Ils constituent un outil fondamental pour tout développeur Python.

Caractéristiques essentielles

- Clés uniques (pas de doublons)
- Clés immuables (int, str, tuple)
- Valeurs de tout type
- Accès en temps constant O(1)
- Syntaxe avec accolades { }

Créer et Manipuler des Dictionnaires

01

Création

Utilisez les accolades `{ }` pour créer un dictionnaire vide ou avec des données initiales

02

Accès aux valeurs

Utilisez la notation entre crochets `dict[clé]` pour récupérer une valeur

03

Ajout d'éléments

Assignez une valeur à une nouvelle clé : `dict['nouvelle_clé'] = valeur`

04

Modification

Réaffectez simplement une valeur à une clé existante pour la modifier

```
mon_dict = {}
mon_dict_a = {'test': 'Texte de test', 1: 25}
print(mon_dict_a[1]) # Affiche: 25
mon_dict_a['blabla'] = 'Nouvelle valeur'
```

Méthodes Essentielles des Dictionnaires

.keys()

Retourne une vue itérable de toutes les clés du dictionnaire. Parfait pour parcourir uniquement les clés dans une boucle `for`.

```
for k in mon_dict.keys():
    print(k)
```

.values()

Retourne une vue itérable de toutes les valeurs. Utilisez-la quand seules les valeurs vous intéressent, sans les clés.

```
print(mon_dict.values())
```

.items()

Retourne des tuples (clé, valeur) pour chaque paire. La méthode la plus utilisée pour itérer sur un dictionnaire complet.

```
for key, value in dict.items():
 print(f"{key}: {value}")
```

Supprimer et Fusionner des Éléments

Méthodes de suppression

Python offre deux méthodes principales pour supprimer des éléments d'un dictionnaire :

L'instruction `del` supprime directement une clé. Attention : elle lève une exception `KeyError` si la clé n'existe pas.

```
del mon_dict_a['test']
```

La méthode `.pop()` supprime et retourne la valeur associée à la clé. Vous pouvez fournir une valeur par défaut pour éviter les erreurs.

```
valeur = mon_dict_a.pop(1)
# Avec défaut:
val = dict.pop('clé', None)
```

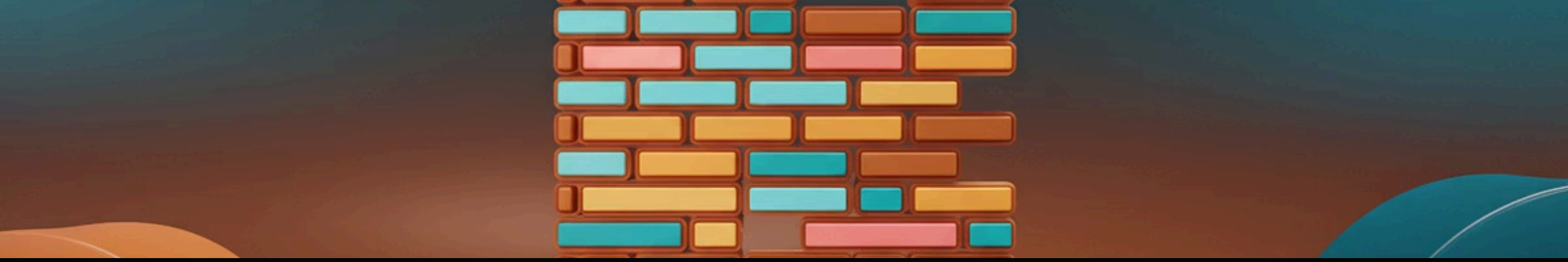
Fusion de dictionnaires

La méthode `.update()` fusionne deux dictionnaires. Les valeurs du dictionnaire source écrasent celles du dictionnaire cible en cas de clés identiques.

```
mon_dict_a.update({2: "Valeur"})
```

⚠️ Attention aux exceptions

Toujours vérifier l'existence d'une clé avec `if 'clé' in dict:` avant d'utiliser `del` ou préférer `.pop()` avec une valeur par défaut.



Introduction au Format JSON

Qu'est-ce que JSON ?

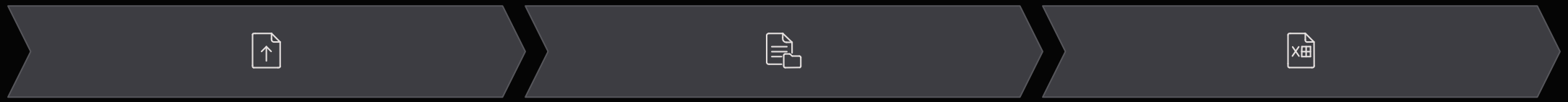
JSON (JavaScript Object Notation) est un format de données textuelles léger et lisible. Il est devenu le standard pour l'échange de données entre applications web, API REST et systèmes distribués.

JSON structure les données avec des paires clé-valeur, similaires aux dictionnaires Python. Cette correspondance naturelle facilite grandement la manipulation de données JSON en Python.

Pourquoi utiliser JSON ?

- Format universel et interopérable
- Lisible par les humains et machines
- Léger et performant
- Support natif dans Python
- Idéal pour les API et configurations

Le Module JSON en Python



Import du module

Commencez par importer le module standard `json` qui contient toutes les fonctions nécessaires

```
import json
```

Lecture de fichiers

Utilisez `.load()` pour charger un fichier JSON directement en dictionnaire Python

```
with open('file.json', 'r') as f:
    data = json.load(f)
```

Écriture de fichiers

Utilisez `.dump()` pour sauvegarder un dictionnaire dans un fichier JSON

```
with open('file.json', 'w') as f:
    json.dump(mon_dict, f, indent=4)
```

Le paramètre `indent=4` rend le fichier JSON formaté et lisible. C'est une bonne pratique pour la maintenance et le débogage.

Manipuler JSON : Fichiers et Chaînes

Méthodes pour les fichiers

json.load()

Lit un fichier JSON et retourne un dictionnaire Python

json.dump()

Écrit un dictionnaire Python dans un fichier JSON

```
file = open('file.json', 'r')
data = json.load(file)
file.close()
```

Méthodes pour les chaînes

json.dumps()

Convertit un dictionnaire en chaîne JSON (le 's' = string)

json.loads()

Parse une chaîne JSON et retourne un dictionnaire

```
json_str = json.dumps(mon_dict, indent=4)
data = json.loads(json_str)
print(type(data)) #
```

Exemple Pratique Complet

Voici un exemple complet qui illustre la vérification d'existence de fichier, la lecture/écriture JSON, et la manipulation des données :

```
import os, json

file_path = './file.json'
mon_dict = {'people': ['Albert', 'Martin', 'Louis'],
            'mes_chiens': [1, 2, 4, 5]}

if os.path.exists(file_path):
    # Le fichier existe : on le lit
    with open(file_path, 'r') as file:
        data = json.load(file)
        print(data)
        print(data['people'])  # Accès aux données
else:
    # Le fichier n'existe pas : on le crée
    with open(file_path, 'w') as file:
        json.dump(mon_dict, file, indent=4)
        print("Fichier créé avec succès")

# Conversion dict → JSON string
json_str = json.dumps(mon_dict, indent=4)

# Conversion JSON string → dict
nouveau_dict = json.loads(json_str)
```

💡 Bonne pratique

Utilisez toujours `with open()` pour gérer automatiquement la fermeture des fichiers et éviter les fuites de ressources.

Points Clés à Retenir

🔑 Dictionnaires

- Structure clé-valeur native Python
- Méthodes essentielles : `.keys()`, `.values()`, `.items()`
- Suppression : `del` ou `.pop()`
- Fusion : `.update()`

📄 Format JSON

- Standard universel d'échange de données
- Correspondance naturelle avec les dictionnaires Python
- Léger, lisible et performant
- Module `json` intégré à Python

🔧 Manipulation JSON

- `load()` / `dump()` : fichiers
- `loads()` / `dumps()` : chaînes
- Paramètre `indent` pour la lisibilité
- Gestion des erreurs avec `os.path.exists()`

Vous maîtrisez maintenant les fondamentaux des dictionnaires et de JSON en Python ! Ces compétences sont essentielles pour vos projets SLAM : API REST, configuration d'applications, manipulation de données structurées. Pratiquez régulièrement avec des exercices concrets pour consolider ces acquis.