Day 16: Working with JSON in Python

JSON (**JavaScript Object Notation**) is a **lightweight** data format used for storing and exchanging data. It is widely used in **web APIs, configuration files, data storage, and data exchange between frontend and backend systems.**

Python provides the built-in **json** module to handle **JSON reading, writing, serialization, and deserialization.**

Key Topics Covered Today

1. **What is JSON?**
2. **Reading and Writing JSON Files**
    - Writing JSON to a file
    - Reading JSON from a file
3. **JSON Serialization and Deserialization**
    - Converting Python objects to JSON (Serialization)
    - Converting JSON to Python objects (Deserialization)
4. **Working with Complex JSON Data**
5. **Handling Errors in JSON Processing**

---

1. What is JSON?

JSON is a **text-based** format that represents structured data using **key-value pairs**. It is similar to a **Python dictionary** but is language-independent.

JSON Syntax Rules

✅ Data is stored as **key-value pairs** (like a Python dictionary)
✅ **Keys** must be **strings**
✅ **Values** can be strings, numbers, booleans, arrays (lists), or nested JSON objects

Example of JSON Data

```
{
 "name": "John Doe",
 "age": 30,
 "isEmployed": true,
 "skills": ["Python", "Django", "Machine Learning"],
 "address": {
  "city": "New York",
  "country": "USA"
 }
}
```

JSON vs Python Dictionary

JSON Type	Python Equivalent
Object {}	Dictionary dict
Array []	List list
String "text"	String str
Number 123	Integer int / Float float
Boolean true / false	Boolean True / False
Null null	None None

2. Reading and Writing JSON Files

Writing JSON to a File

To save a Python dictionary as a **JSON file**, use json.dump().

Example: Writing JSON to a File

```
import json


# Python dictionary
data = {
    "name": "Alice",
    "age": 25,
    "skills": ["Python", "Data Science"],
    "address": {"city": "San Francisco", "country": "USA"}
}


# Writing to a JSON file
with open("data.json", "w") as file:
    json.dump(data, file, indent=4)


print("JSON file written successfully!")
```

Explanation:

- **json.dump(data, file)** → Converts Python dictionary data into JSON format and writes it to data.json.

- **indent=4** → Formats JSON with indentation for readability.

Reading JSON from a File

To load JSON data from a file into a Python object, use json.load().

Example: Reading JSON from a File

```
import json


# Reading from a JSON file
with open("data.json", "r") as file:
    data = json.load(file)


print("Data loaded from JSON file:")
print(data)
```

Explanation:

- **json.load(file)** → Reads the JSON file and converts it into a Python dictionary.
- The output is a normal Python dictionary that can be accessed like data["name"].

---

3 JSON Serialization and Deserialization

What is Serialization?

Serialization is the process of **converting Python objects into JSON format** so they can be stored or transmitted.

What is Deserialization?

Deserialization is the process of **converting JSON data back into Python objects**.

JSON Serialization (Python → JSON String)

Use json.dumps() to convert a Python object into a **JSON string**.

Example: Converting Python Object to JSON String

```
import json

# Python object
person = {
    "name": "Bob",
    "age": 28,
    "isMarried": False,
    "pets": None
}

# Convert Python dictionary to JSON string
json_string = json.dumps(person, indent=4)

print(json_string)
```

Output:

```
{
    "name": "Bob",
    "age": 28,
    "isMarried": false,
```

```
  "pets": null
}
```

Explanation:

- **json.dumps(person)** → Converts Python dictionary person to a JSON **formatted string**.
- **Boolean & None Conversion:**
  - False → false
  - None → null

JSON Deserialization (JSON String → Python Object)

Use json.loads() to convert a **JSON string** into a Python dictionary.

Example: Converting JSON String to Python Object

```
import json

# JSON string
json_data = '{"name": "Eve", "age": 24, "city": "Los Angeles"}'

# Convert JSON string to Python dictionary
python_dict = json.loads(json_data)

print(python_dict)
print(python_dict["name"])  # Accessing values like a dictionary
```

Output:

```
{'name': 'Eve', 'age': 24, 'city': 'Los Angeles'}
Eve
```

---

4 Working with Complex JSON Data (Nested JSON Parsing)

JSON can contain **nested objects** and **lists**. Let's see how to parse them.

Example: Parsing Nested JSON

```
import json

# JSON with nested structure
json_data = '''
{
   "company": "TechCorp",
   "employees": [
      {"name": "Alice", "role": "Developer"},
      {"name": "Bob", "role": "Designer"},
      {"name": "Charlie", "role": "Manager"}
   ]
}
'''

# Convert JSON string to Python dictionary
data = json.loads(json_data)
```

```
# Extracting nested data
for emp in data["employees"]:
    print(f"Name: {emp['name']}, Role: {emp['role']}")
```

Output:

```
Name: Alice, Role: Developer
Name: Bob, Role: Designer
Name: Charlie, Role: Manager
```

Explanation:

- **Nested JSON Objects (employees)** are parsed as **lists of dictionaries**.
- We can loop through **data["employees"]** to extract employee details.

---

5 Handling Errors in JSON Processing

JSON operations can fail due to various reasons, such as **invalid JSON format** or **file not found**. We should always handle exceptions.

Example: Handling JSON Errors

```
import json

invalid_json = '{ "name": "John", age: 30 }'  # Invalid JSON (age key is not in quotes)

try:
    data = json.loads(invalid_json)
except json.JSONDecodeError as e:
    print("Error decoding JSON:", e)
```

Output:

Error decoding JSON: Expecting property name enclosed in double quotes

Best Practices for Handling JSON Errors

✅ Use try-except when working with JSON.
✅ Ensure **proper formatting** (keys should be strings, use double quotes).

Summary of Key Learnings

✅ **Reading & Writing JSON Files:** json.dump(), json.load()
✅ **JSON Serialization:** json.dumps() (Python → JSON)
✅ **JSON Deserialization:** json.loads() (JSON → Python)
✅ **Handling Nested JSON Data**
✅ **Error Handling in JSON Processing**

Mastering JSON in Python is **essential** for working with APIs, data storage, and web applications!