# JSON


JSON
- Reading a JSON File
  - Example:
- Writing a JSON File
  - Example:
- json.load vs json.loads
  - Parsing JSON from a String (json.loads)
- Key Points to Remember


Working with JSON (JavaScript Object Notation) files in Python is straightforward and efficient thanks to the `json` module, which is part of the Python standard library. JSON is a lightweight data interchange format that's easy for humans to read and write, and easy for machines to parse and generate. It's widely used for data transmission in web applications.

## Reading a JSON File

To read data from a JSON file, you use the `json.load()` function. This function parses the JSON content of a file and returns a Python object (usually a dictionary or a list, depending on the JSON structure).

### Example:

Suppose you have a JSON file named `example.json` with the following content:

```
{
  "name": "Rajath Kumar",
  "age": 31,
  "is_employee": true,
  "email": "rajathkuamrks@gmail.com",
  "skills": ["Embedded/RTOS", "IoT", "Python", "ML/DL"]
}

```

You can read this file as follows in Python:

```
import json

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

print(data)

```

# Writing a JSON File

To write data to a JSON file, you use the `json.dump()` function. This function takes a Python object and writes it to a file in JSON format.

### Example:

Let's create a new JSON file named `new_example.json` and write some data into it:

```python
import json

data = {
    "name": "Elon Musk",
    "age": 45,
    "is_employee": False,
    "skills": ["HTML", "CSS", "JavaScript"]
}

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

```

This will create a file `new_example.json` with the following content:

```json
{
    "name": "Elon Musk",
    "age": 45,
    "is_employee": false,
    "skills": [
        "HTML",
        "CSS",
        "JavaScript"
    ]
}
```

In the `json.dump()` function, the `indent` parameter is optional and specifies the number of spaces to use for pretty-printing the JSON output. Without it, the JSON data would be written in a compact form without any whitespace (which is more efficient for storage and transmission but harder for humans to read).

## json.load vs json.loads

1. **`json.load`**: This function is used to read JSON data from a file-like object (like one obtained from a `open()` call). It parses the JSON content of a file and returns a Python object.
2. **`json.loads`**: This function is used to parse a JSON string. It takes a JSON string as input and converts it to a Python object. The 's' in `loads` stands for 'string'.

### Parsing JSON from a String (json.loads)

Sometimes, you might receive JSON data as a string, which is common when working with web APIs.

Example JSON String,

```
json_string = '{"name": "Rajath Kumar", "age": 31, "cities_visited": ["Paris", "London", "Dubai"]}'
```

Parsing the String

```
json_string = '{"name": "Rajath Kumar", "age": 31, "cities_visited": ["Paris", "London", "Dubai"]}'

import json

parsed_data = json.loads(json_string)

print(parsed_data)

# =================

data = json.dumps(json_string, indent=4)
print(data)

#"{\"name\": \"Rajath Kumar\", \"age\": 31, \"cities_visited\": [\"Paris\", \"London\", \"Dubai\"]}"
```

## Key Points to Remember

- `json.load` and `json.loads` are for reading JSON data into Python objects. The difference lies in the source of the JSON (file vs. string).
- `json.dump` and `json.dumps` are for writing Python objects to JSON format. The difference is whether you're writing to a file or creating a string.
- JSON only supports a specific set of data types: objects (dictionaries in Python), arrays (lists in Python), strings, numbers, booleans (`true`/`false` in JSON, `True`/`False` in Python), and `null` (`None` in Python).
- Handling errors and exceptions (like `FileNotFoundError` or `json.JSONDecodeError`) is important for robust code, especially when dealing with external files or data sources.