Python JSON

JSON (**J**ava**S**cript **O**bject **N**otation) is a popular data format used for representing structured data. It's common to transmit and receive data between a server and web application in JSON format.

In Python, JSON exists as a string. For example:

```
p = '{"name": "Bob", "languages": ["Python", "Java"]}'
```

It's also common to store a JSON object in a file.

Import json Module

To work with JSON (string, or file containing JSON object), you can use Python's `json` module. You need to import the module before you can use it.

```
import json
```

Parse JSON in Python

The `json` module makes it easy to parse JSON strings and files containing JSON object.

Example 1: Python JSONÂ to dict

You can parse a JSON string using `json.loads()` method. The method returns a dictionary.

```
import json

person = '{"name": "Bob", "languages": ["English", "French"]}'
person_dict = json.loads(person)

# Output: {'name': 'Bob', 'languages': ['English', 'French']}
print( person_dict)

# Output: ['English', 'French']
print(person_dict['languages'])
```

Here, *person* is a JSON string, and *person_dict* is a dictionary.

Example 2: Python read JSON file

You can use `json.load()` method to read a file containing JSON object.

Suppose, you have a file named `person.json` which contains a JSON object.

```
{"name": "Bob",
"languages": ["English", "French"]
}
```

Here's how you can parse this file:

```
import json

with open('path_to_file/person.json', 'r') as f:
  data = json.load(f)

# Output: {'name': 'Bob', 'languages': ['English', 'French']}
print(data)
```

Here, we have used the `open()` function to read the json file. Then, the file is parsed using `json.load()` method which gives us a dictionary named *data*.

If you do not know how to read and write files in Python, we recommend you to check Python File I/O.

Python Convert to JSON string

You can convert a dictionary to JSON string using `json.dumps()` method.

Example 3: Convert dict to JSON

```
import json

person_dict = {'name': 'Bob',
'age': 12,
'children': None
}
person_json = json.dumps(person_dict)

# Output: {"name": "Bob", "age": 12, "children": null}
print(person_json)
```

Here's a table showing Python objects and their equivalent conversion to JSON.

Python	JSON Equivalent
`dict`	object
`list`, `tuple`	array
`str`	string
`int`, `float`, `int`	number
`True`	true
`False`	false
`None`	null

Writing JSON to a file

To write JSON to a file in Python, we can use `json.dump()` method.

Example 4: Writing JSON to a file

```
import json

person_dict = {"name": "Bob",
"languages": ["English", "French"],
"married": True,
"age": 32
}

with open('person.txt', 'w') as json_file:
  json.dump(person_dict, json_file)
```

In the above program, we have opened a file named `person.txt` in writing mode using `'w'`. If the file doesn't already exist, it will be created. Then, `json.dump()` transforms `person_dict` to a JSON string which will be saved in the `person.txt` file.

When you run the program, the `person.txt` file will be created. The file has following text inside it.

```
{"name": "Bob", "languages": ["English", "French"], "married": true, "age": 32}
```

Python pretty print JSON

To analyze and debug JSON data, we may need to print it in a more readable format. This can be done by passing additional parameters `indent` and `sort_keys` to `json.dumps()` and `json.dump()` method.

Example 5: Python pretty print JSON

```
import json

person_string = '{"name": "Bob", "languages": "English", "numbers": [2, 1.6, null]}'

# Getting dictionary
person_dict = json.loads(person_string)

# Pretty Printing JSON string back
print(json.dumps(person_dict, indent = 4, sort_keys=True))
```

When you run the program, the output will be:

```
{
    "languages": "English",
    "name": "Bob",
    "numbers": [
        2,
        1.6,
        null
    ]
}
```

In the above program, we have used `4` spaces for indentation. And, the keys are sorted in ascending order.

By the way, the default value of *indent* is `None`. And, the default value of *sort_keys* is `False`.

Recommended Readings:

- Python JSON to CSV and vice-versa
- Python XML to JSON and vice-versa
- Python simplejson

Table of Contents