Why do we use JSON?

The **JSON** format **is** often **used** for serializing and transmitting structured data over a network connection. It **is used** primarily to transmit data between a server and web application, serving as an alternative to XML.

When transmitting data or storing them in a file, the data are required to be byte strings, but complex objects are seldom in this format. Serialization can convert these complex objects into byte strings for such use. After the byte strings are transmitted, the receiver will have to recover the original object from the byte string. This is known as deserialization.It is mainly used between server and a web application

Say, you have an object

{foo: [1, 4, 7, 10], bar: "baz"}

serializing into JSON will convert it into a string:

'{"foo":[1,4,7,10],"bar":"baz"}'

which can be stored or sent through wire to anywhere. The receiver can then deserialize this string to get back the original object. {foo: [1, 4, 7, 10], bar: "baz"}.

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:

1. 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.

1. import json

### Example 1: Python JSON to dict

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

1. import json
2. person = '{"name": "Bob", "languages": ["English", "Fench"]}'
3. person\_dict = json.loads(person)
4. # Output: {'name': 'Bob', 'languages': ['English', 'Fench']}
5. print( person\_dict)
6. # Output: ['English', 'French']
7. 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.

1. {"name": "Bob",
2. "languages": ["English", "Fench"]
3. }

Here's how you can parse this file:

1. import json
2. with open('path\_to\_file/person.json') as f:
3. data = json.load(f)
4. # Output: {'name': 'Bob', 'languages': ['English', 'Fench']}
5. 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](https://www.programiz.com/python-programming/file-operation).

## Python Convert to JSON string

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

### Example 3: Convert dict to JSON

1. import json
2. person\_dict = {'name': 'Bob',
3. 'age': 12,
4. 'children': None
5. }
6. person\_json = json.dumps(person\_dict)
7. # Output: {"name": "Bob", "age": 12, "children": null}
8. 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

1. import json
2. person\_dict = {"name": "Bob",
3. "languages": ["English", "Fench"],
4. "married": True,
5. "age": 32
6. }
7. with open('person.txt', 'w') as json\_file:
8. 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.

1. {"name": "Bob", "languages": ["English", "Fench"], "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\_keysto json.dumps() and json.dump() method.

### Example 5: Python pretty print JSON

1. import json
2. person\_string = '{"name": "Bob", "languages": "English", "numbers": [2, 1.6, null]}'
3. # Getting dictionary
4. person\_dict = json.loads(person\_string)
5. # Pretty Printing JSON string back
6. 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.

#convert dict to JSON

import json

persond = {'name': 'Bob','age': 12,'children': None}

personj = json.dumps(persond)#convert a dictionary to JSON string using json.dumps() method.

# Output: {"name": "Bob", "age": 12, "children": null}

print(personj)

#Python printing JSON filterimport json

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

# Getting dictionary

persondict = json.loads(personstring)

# Pretty Printing JSON string back

print(json.dumps(persondict, indent = 4, sort\_keys=True))