Python JSON

Python JSON JavaScript Object Notation is a format for structuring data. It is mainly used for storing and transferring data between the browser and the server. Python too supports JSON with a built-in package called JSON. This package provides all the necessary tools for working with JSON Objects including parsing, serializing, deserializing, and many more.

```
import json

employee = '{"id":"09", "name": "Nitin", "department":"Finance"}'

print("This is JSON", type(employee))

print("\nNow convert from JSON to Python")

employee_dict = json.loads(employee)

print("Converted to Python", type(employee_dict))

print(employee_dict)
```

Convert from Python object to JSON

```
import json

employee_dict = {'id': '09', 'name': 'Nitin', 'department': 'Finance'}

print("This is Python", type(employee_dict))

print("\nNow Convert from Python to JSON")

json_object = json.dumps(employee_dict, indent=4)

print("Converted to JSON", type(json_object))

print(json_object)
```

JSON is JavaScript Object Notation. It means that a script (executable) file which is made of text in a programming language, is used to store and transfer the data. Python supports JSON through a built-in package called JSON. Python3

```
import json

a ={"name":"John","age":31,"Salary":25000}

b = json.dumps(a)

print(b)
```

```
# Python program showing that

# json support different primitive

# types

import json

print(json.dumps(['Welcome', "to", "DishaComputerInstitute"]))

print(json.dumps(("Welcome", "to", "DishaComputerInstitute")))

print(json.dumps("Hi"))

print(json.dumps(123))

print(json.dumps(23.572))

print(json.dumps(True))

print(json.dumps(False))

print(json.dumps(None))
```

Serializing JSON:

The process of encoding JSON is usually called **serialization**. This term refers to the transformation of data into a series of bytes (hence serial) to be stored or transmitted across a network. To handle the data flow in a file, the JSON library in Python uses dump() function to convert the Python objects into their respective JSON object, so it makes it easy to write data to files. See the following table given below.

Python object	JSON object
dict	object
list, tuple	array
str	string
int, long, float	numbers
True	true
False	false
None	null

Example: Serialization

Consider the given example of a Python object.

```
var = {
        "Subjects": {"Maths":85,"Physics":90}
    }
```

Using Python's context manager, create a file named Sample.json and open it with write mode.

```
with open("Sample.json", "w") as p:
        json.dump(var, p)
```

Here, the dump() takes two arguments first, the data object to be serialized, and second the object to which it will be written(Byte format).

Deserializing JSON:

Deserialization is the opposite of Serialization, i.e. conversion of JSON objects into their respective Python objects. The load() method is used for it. If you have used JSON data from another program or obtained it as a string format of JSON, then it can easily be deserialized with load(), which is usually used to load from a string, otherwise, the root object is in a list or dict.

EX-

```
with open("Sample.json", "r") as read_it:
        data = json.load(read_it)
```

Example: Deserialization

```
json_var ="""
{
        "Country": {"name": "INDIA","Languages_spoken": [{"names": ["Hindi", "English", "Bengali","Telugu"]}]}
}
"""
var = json.loads(json_var)
```

Append to JSON file using Python

Functions Used:

- **json.loads():** json.loads() function is present in python built-in 'json' module. This function is used to parse the JSON string.

  Syntax: *json.loads(json_string)*
  Parameter: *It takes JSON string as the parameter.*
  Return type: *It returns the python dictionary object.*

- **json.dumps():** json.dumps() function is present in python built-in 'json' module. This function is used to convert Python object into JSON string.

  Syntax: *json.dumps(object)*
  Parameter: *It takes Python Object as the parameter.*
  Return type: *It returns the JSON string.*

- **update():** This method updates the dictionary with elements from another dictionary object or from an iterable key/value pair.

  Syntax: *dict.update([other])*
  Parameters: *Takes another dictionary or an iterable key/value pair.*
  Return type: *Returns None.*

```
# Python program to update

# JSON

import json:

x = '{ "organization":"DishaComputerInstitute","city":"Pune","country":"India"}'

y = {"pin":110096}

z = json.loads(x)

z.update(y)

print(json.dumps(z))
```

```
# Python program to update

# JSON

import json

def write_json(new_data, filename='data.json'):

    with open(filename,'r+') as file:

            file_data = json.load(file)
```

```
            file_data["emp_details"].append(new_data)

            file.seek(0)

            json.dump(file_data, file, indent = 4)

y = {"emp_name":"Nikhil","email": "nikhil@geeksforgeeks.org","job_profile": "Full Time"}

write_json(y)
```

Serializing JSON data in Python

Serialization is the process of encoding the from naive data type to JSON format. The Python module json converts a Python dictionary object into JSON object, and list and tuple are converted into JSON array, and int and float converted as JSON number, None converted as JSON null.

json.dump()

json.dump() method can be used for writing to JSON file. Write data to a file-like object in json format.

Syntax: json.dump(dict, file_pointer)
Parameters:
dictionary – name of dictionary which should be converted to JSON object.
file pointer – pointer of the file opened in write or append mode.

Converting python object and writing into json file.

```
import json

data = {"user": {"name": "satyam kumar","age": 21,"Place": "Patna","Blood group": "O+"}}

with open( "datafile.json" , "w" ) as write:

        json.dump( data , write )
```

json.dumps()

json.dumps() method can convert a Python object into a JSON string.
Syntax: json.dumps(dict)
Parameters:
dictionary – name of dictionary which should be converted to JSON object.

```
  import json
   data = {"user": {"name": "satyam kumar","age": 21,"Place": "Patna","Blood group":
"O+"}}
  res = json.dumps( data )
   print( res )
```

Deserialize JSON to Object in Python

Let us see how to deserialize a JSON document into a Python object. Deserialization is the process of decoding the data that is in JSON format into native data type. In Python, deserialization decodes JSON data into a dictionary(data type in python).
We will be using these methods of the **json** module to perform this task :

- **loads()** : to deserialize a JSON document to a Python object.
- **load()** : to deserialize a JSON formatted stream (which supports reading from a file) to a Python object.

Example 1 : Using the loads() function.

```
# importing the module
import json
data = '{"Name" : "Vanshika", "Gender" : "Female"}'
print("Datatype before deserialization : "+ str(type(data)))
data = json.loads(data)
print("Datatype after deserialization : "+ str(type(data)))
```

```
# importing the module

import json

data = open('file.json',)

print("Datatype before deserialization : "+ str(type(data)))

data = json.load(data)

print("Datatype after deserialization : "+ str(type(data)))
```