JSON

1. JSON is text, written with 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.
2. Python supports JSON through a built-in package called JSON.

*Import JSON*

1. The text in JSON is done through quoted-string which contains the value in key-value mapping within { }.
2. It is similar to the dictionary in Python
3. JSON shows an API similar to users of Standard Library marshal and pickle modules

Parse JSON - Convert from JSON to Python

If you have a JSON string, you can parse it by using the json.loads() method

*import json  
  
# some JSON:  
x =  '{ "name":"John", "age":30, "city":"New York"}'  
  
# parse x:  
y = json.loads(x)  
  
# the result is a Python dictionary:  
print(y["age"])*

## Convert from Python to JSON

If you have a Python object, you can convert it into a JSON string by using the json.dumps() method.

*import json  
  
# a Python object (dict):  
x = {  
  "name": "John",  
  "age": 30,  
  "city": "New York"  
}  
  
# convert into JSON:  
y = json.dumps(x)  
  
# the result is a JSON string:  
print(y)*

You can convert Python objects of the following types, into JSON strings:

* dict
* list
* tuple
* string
* int
* float
* True
* False
* None

import json  
  
print(json.dumps({"name": "John", "age": 30}))  
print(json.dumps(["apple", "bananas"]))  
print(json.dumps(("apple", "bananas")))  
print(json.dumps("hello"))  
print(json.dumps(42))  
print(json.dumps(31.76))  
print(json.dumps(True))  
print(json.dumps(False))  
print(json.dumps(None))

When you convert from Python to JSON, Python objects are converted into the JSON (JavaScript) equivalent:

|  |  |
| --- | --- |
| **Python** | **JSON** |
| dict | Object |
| list | Array |
| tuple | Array |
| str | String |
| int | Number |
| float | Number |
| True | true |
| False | false |
| None | null |

Convert a Python object containing all the legal data types:

import json  
  
x = {  
  "name": "John",  
  "age": 30,  
  "married": True,  
  "divorced": False,  
  "children": ("Ann","Billy"),  
  "pets": None,  
  "cars": [  
    {"model": "BMW 230", "mpg": 27.5},  
    {"model": "Ford Edge", "mpg": 24.1}  
  ]  
}  
  
print(json.dumps(x))

## Format the Result(parameters in dumps)

The example above prints a JSON string, but it is not very easy to read, with no indentations and line breaks.

The json.dumps() method has parameters to make it easier to read the result:

**Use the indent parameter to define the numbers of indents:**

json.dumps(x, indent=4)

**You can also define the separators,** default value is (", ", ": "), which means using a comma and a space to separate each object, and a colon and a space to separate keys from values:

Use the separators parameter to change the default separator:

json.dumps(x, indent=4, separators=(". ", " = "))

## Order the Result

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

*#formating the string*

*import json*

*x = {*

*"name": "John",*

*"age": 30,*

*"married": True,*

*"divorced": False,*

*"children": ("Ann","Billy"),*

*"pets": None,*

*"cars": [*

*{"model": "BMW 230", "mpg": 27.5},*

*{"model": "Ford Edge", "mpg": 24.1}*

*]*

*}*

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

### **Serializing JSON:**

1. The process of encoding JSON is usually called **serialization**.
2. This term refers to the transformation of data into a series of bytes (hence serial) to be stored or transmitted across a network.
3. 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
4. so it makes it easy to write data to files. See the following table given below.

*#serialization(python to json=> dump)*

*var = {*

*"Subjects": {*

*"Maths":85,*

*"Physics":90*

*}*

*}*

*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:**

1. Deserialization is the opposite of Serialization, i.e. conversion of JSON objects into their respective Python objects.
2. The load() method is used for it.
3. 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(),
4. which is usually used to load from a string, otherwise, the root object is in a list or dict.

*#Deserialization(json to python= load)*

*import json*

*fp=open("Sample1.json", "r")*

*data = json.load(fp)*

*print(data)*

***Pickling***

*Pickling is a process of converting a class object into a byte stream so that it can be stored into a file. This is also called as object serialization.*

*We use pickle module to perform pickling and unpickling.*

*dump( ) – This function is used to perform the pickling. It returns the pickled representation of the object as a bytes object, instead of writing it to a file.*

*This method belongs to pickle module.*

*Syntax:-*

*import pickle*

*pickle.dump(object, file)*

**Unpickling**

*Unpickling is a process whereby byte stream is converted back into a class object. It is inverse operation of pickling. This is also called as de-serialization.*

*Pickling and unpickling should be done using binary files since they support byte streams.*

*We use pickle module to perform pickling and unpickling.*

*Warning: The pickle module is not secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.*

*load( ) – This function is used to read an pickled object from a binary file and returns it into object. This method belongs to pickle module.*

*Syntax:-*

*import pickle*

*pickle.load(file)*

***Why do we need Pickling and Unpickling***

*When we store some structured data in the file and want to perform calculation that time we need pickling and unpickling.*

