Python JSON Concept:

In this tutorial, we'll see how we can create, manipulate, and parse JSON in Python using the standard a **json** module. The built-in Python json module provides us with methods and classes that are used to parse and manipulate JSON in Python.

What is JSON:

JSON (an acronym for JavaScript Object Notation) is a data-interchange format and is most commonly used for client-server communication

Example:

```
{"name": "jane doe", "salary": 9000, "email": "JaneDoe@pynative.com"}
```

A JSON is an unordered collection of key and value pairs, resembling Python's native dictionary.

- Keys are unique Strings that cannot be null.
- Values can be anything from a String, Boolean, Number, list, or even null.
- A JSONO can be represented by a String enclosed within curly braces with keys and values separated by a colon, and pairs separated by a comma

Why do we use JSON?

- Whenever the client needs information, it calls the server using a URI, and the server returns data to the client in the form of JSON. Later we can use this data in our application as per our requirement.
- Also, when the client application wants to store the data on the server. It can POST that data in the form of JSON.
- It is used primarily to transmit data between a server and web application, serving as an alternative to XML.

JSON is most commonly used for client-server communication because:

- It is human readable.
- It's both easy to read/write and
- JSON is language-independent.

Why JSON is Better than XML ??

- XML is a much more difficult to Parse the data than JSON.
- JSON doesn't use the Tags . JSON is Shorter and Use Arrays.
- It is very Fast to read and write the data
- For AJAXs applications , JSON is Faster and easier than XML.

Example 1:

json_data = { "fname" : "Srinivas" , "lname" : "Rao" , "age" : 27 }

Example 2:

json_data = { "names" : ["Ramu" , "Ravi" , "Raju"], "data" : { "name" : "Sri" , "location" : "HYD"} }

Note 1: In JSON structure , Curly braces hold the objects and Square brackets hlod the Arrays.

Note 2: Each key and value should be in a Double quotes if it is strings.

MIME value:

- **MIME** stand for Multipurpose Internet Mail Extention.
- By uing MIME type attribute we will represents what type of data we want to return as response when we are sending the request.
- The official Internet media type for JSON is **application/json**.

Syntax: return HttpResponse(resp , content_type = 'application/json')

Example 1: Write a program to define an Employees object with an Array of 3 employees by uing JSON structure ????

Using JSON:

```
{
    "employees" : [
            { "fname" : "Srinivas" , "lname" : "Rao" } ,
            { "fname" :  "Virat", "lname" : "Kohli" } ,
            { "fname" : "Rohit" , "lname" : "Sharma" } ,
     ]
}
```

Example 2: Write a program to define an Employees object with an Array of 3 employees by uing XML structure ????

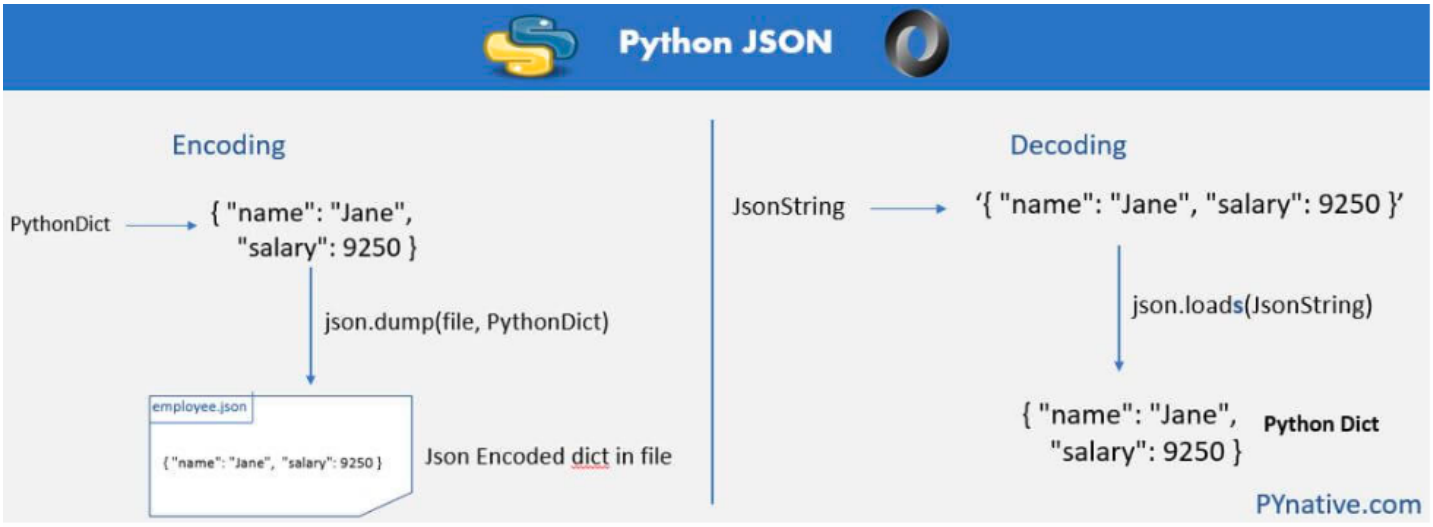
```
<employees>
     <employee>
```

```
        <fname> Srinivas </fname>
        <lname> Rao </lname>
    </employee>
     <employee>
        <fname>Rohit </fname>
        <lname>Sharma </lname>
    </employee>
    <employee>
        <fname> Virat</fname>
        <lname>Kohli </lname>
    </employee>
</employees>
```

Python json Module:

Python comes with a built-in module called **json** for working with JSON data. You only need to add `import json` at the start of your file and you are ready to use it.

Encoding and Decoding Structure:



Python JSON Tutorial

Mapping between JSON and Python entities while Encoding

Now let's see how to convert all Python primitive types such as a `dict`, `list`, `set`, `tuple`, `str`, numbers into JSON formatted data. Please refer to the following table to know the mapping between JSON and Python data types.

Python	JSON
dict	object
list, tuple	array
str	string
int, float, int & float-derived Enums	number
True	true
False	false
None	null

Mapping between JSON and Python data types

Example:

 # Import the json module.

```
import  json
```

Parse JSON - Convert from JSON to Python:

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

Example: Convert from JSON to Python:

```
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"])
```

Output:

30

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.

Example: Convert from Python to JSON:

```
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)
```

Output: {"name" : "John" , "age" : 30 , "city" : "New York"}

Example: Convert Python objects into JSON strings, and print the values.

```
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))
```

Output:

```
{ "name" :  "John" ,  "age" :  30}
["apple", "bananas"]
["apple", "bananas"]
"hello"
42
31.76
true
false
null
```

Example: 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))
```

Output: {"name": "John", "age": 30, "married": true, "divorced": false, "children": ["Ann","Billy"], "pets": null, "cars": [{"model": "BMW 230", "mpg": 27.5}, {"model": "Ford Edge", "mpg": 24.1}]}

Format the Result:

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:

Example:

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

For example, **json.dumps(x, indent=4)**

Code:

```
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}
```

```
  ]
}
# use four indents to make it easier to read the result:
print(json.dumps(x, indent=4))
```

Output:

```
{
   "name" :  "John",
   "age" :  30,
   "married" :  true,
   "divorced" :  false,
   "children" : [
      "Ann",
      "Billy"
   ],
   "pets": null,
   "cars": [
      {
         "model": "BMW 230",
         "mpg": 27.5
      },
      {
         "model": "Ford Edge",
         "mpg": 24.1
      }
   ]
}
```

Order the Result:

The json.dumps() method has parameters to order the keys in the result:

Example:

Use the sort_keys parameter to specify if the result should be sorted or not:

For example, json.dumps(x, indent=4, sort_keys=True)

Example:

```
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}
  ]
}
# sort the result alphabetically by keys:
print(json.dumps(x, indent=4, sort_keys=True))
```

Output:

```
{
   "age": 30,
   "cars": [
      {
         "model": "BMW 230",
         "mpg": 27.5
      },
      {
         "model": "Ford Edge",
         "mpg": 24.1
      }
   ],
   "children": [
      "Ann",
      "Billy"
   ],
   "divorced": false,
   "married": true,
   "name": "John",
   "pets": null
}
```

Note : 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:

Example:

Use the separators parameter to change the default separator:
For example**, json.dumps(x, indent=4, separators=(". ", " = "))**

Read JSON data from a file and convert it into dict using json.load()

- Using a json.load() method, we can read JSON data from text, JSON, or binary file.
- The json.load() method returns data in the form of a Python dictionary.
- Later we use this dictionary to access and manipulate data in our application or system.

Now, let's see the example. For this example, I am reading the "developer.json" file present on my hard drive.

This file contains the following JSON data.

developer.js

```
{
   "name": "jane doe",
   "salary": 9000,
   "skills": [
      "Raspberry pi",
      "Machine Learning",
      "Web Development"
   ],
   "email": "JaneDoe@pynative.com",
   "projects": [
      "Python Data Mining",
      "Python Data Science"
   ]
}
```

developer.py

```
import json
print("Started Reading JSON file")

with open("developer.json", "r") as read_file:
   print("Converting JSON encoded data into Python dictionary")
   developer = json.load(read_file)
   print("Decoded JSON Data From File")
```

```
    for key, value in developer.items():
        print(key, ":", value)
    print("Done reading json file")
```

Output:

```
Started Reading JSON file
Converting JSON encoded data into Python dictionary
Decoded JSON Data From File
name : jane doe
salary : 9000
skills : ['Raspberry pi', 'Machine Learning', 'Web Development']
email : JaneDoe@pynative.com
projects : ['Python Data Mining', 'Python Data Science']
Done reading json file
```

Access JSON data directly using key name:

Use the following code If you want to access the JSON key directly instead of iterating the entire JSON from a file

developer.py

```
import json
print("Started Reading JSON file")
with open("developer.json", "r") as read_file:
    print("Converting JSON encoded data into Python dictionary")
    developer = json.load(read_file)
    print("Decoding JSON Data From File")
    print("Printing JSON values using key")
    print(developer["name"])
    print(developer["salary"])
    print(developer["skills"])
    print(developer["email"])
```

```
print("Done reading json file")
```

Output:

```
Started Reading JSON file
Converting JSON encoded data into Python dictionary
Decoding JSON Data From File
Printing JSON values using key
jane doe
9000
['Raspberry pi', 'Machine Learning', 'Web Development']
JaneDoe@pynative.com
Done reading json file
```

Note : You can read the JSON data from text, json, or a binary file using the same way mentioned above.

Convert JSON String to Python dictionary using json.loads():

- Sometimes we receive JSON response in string format.
- So to use it in our application, we need to convert JSON string into a Python dictionary.
- Using the json.loads() method, we can deserialize native String, byte, or bytearray instance containing a JSON document to a Python dictionary.
- We can refer to the conversion table mentioned at the start of an article.

Example:

```
import json
developerJsonString = """{
   "name": "jane doe",
   "salary": 9000,
   "skills": [
      "Raspberry pi",
      "Machine Learning",
      "Web Development"
```

```
    ],
    "email": "JaneDoe@pynative.com",
    "projects": [
        "Python Data Mining",
        "Python Data Science"
    ]
}
"""
print("Started converting JSON string document to Python dictionary")
developerDict = json.loads(developerJsonString)
print("Printing key and value")
print(developerDict["name"])
print(developerDict["salary"])
print(developerDict["skills"])
print(developerDict["email"])
print(developerDict["projects"])
print("Done converting JSON string document to a dictionary")
```

Output:

```
Started converting JSON string document to Python dictionary
Printing key and value
jane doe
9000
['Raspberry pi', 'Machine Learning', 'Web Development']
JaneDoe@pynative.com
['Python Data Mining', 'Python Data Science']
Done converting JSON string document to a dictionary
```

Parse and Retrieve nested JSON array key-values:

Let’s assume that you’ve got a JSON response that looks like this:

```
developerInfo = """{
   "id": 23,
   "name": "jane doe",
   "salary": 9000,
   "email": "JaneDoe@pynative.com",
   "experience": {"python":5, "data Science":2},
   "projectinfo": [{"id":100, "name":"Data Mining"}]
}
"""
```

For example, You want to retrieve the project name from the developer info JSON array to get to know on which project he/she is working.

Let’s see now how to read nested JSON array key-values.

In this example, we are using a developer info JSON array, which has project info and experience as nested JSON data.

Code:

```
import json
print("Started reading nested JSON array")
developerDict = json.loads(developerInfo)
print("Project name: ", developerDict["projectinfo"][0]["name"])
print("Experience: ", developerDict["experience"]["python"])
print("Done reading nested JSON Array")
```

Output:

```
Started reading nested JSON array
Project name:  Data Mining
Experience:  5
Done reading nested JSON Array
```