JSON Fundamentals

JSON or JavaScript Object Notation is a lightweight text-based **open standard** designed for human-readable data interchange. The JSON format was originally specified by Douglas Crockford, and is described in RFC 4627. The official Internet media type for JSON is application/json. The JSON filename extension is .json. Conventions used by JSON are known to programs, which include C, C++, Java, Python, Perl, etc.

Uses of JSON

- It is used while writing JavaScript based applications that includes browser extensions and websites.
- It is commonly used for transmitting data(Sending and receiving bytes of data) in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa) using HTTP protocols
- JSON format is used for serializing and transmitting structured data over network connection.
- It is primarily used to transmit data between a server and web applications even though it is sometimes difficult to debug JSON.
- Web services and APIs use JSON format to provide public data.
- It can be used with modern programming languages.

Characteristics of JSON

- JSON is easy to read and write.
- It is a lightweight text-based interchange format.
- JSON is language independent.
- JSON is purely a string with a specified data format — it contains only properties, no methods.
- JSON requires double quotes to be used around strings and property names. Single quotes are not valid other than surrounding the entire JSON string.
- Even a single misplaced comma or colon can cause a JSON file to go wrong, and not work. You should be careful to validate any data you are attempting to use (although computer-generated JSON is less likely to include errors, as long as the generator program is working correctly). You can validate JSON using an application like JSONLint.
- JSON can actually take the form of any data type that is valid for inclusion inside JSON, not just arrays or objects. So for example, a single string or number would be valid JSON.

Simple Example in JSON

The following example shows how to use JSON to store information related to books based on their topic and edition.

```
{
  “book”: [
    {
      “id”:”01”,
      “language”: “Java”,
      “edition”: “third”,
      “author”: “Herbert Schildt”
    },
 {
      “id”:”07”,
      “language”: “C++”,
      “edition”: “second”,
      “author”: “E.Balagurusamy”
    }
  ]
}
```

Another Example:

```
{"employees":[
                {"name":"Sonoo", "email":"sonoojaiswal1987@gmail.com"},
                {"name":"Rahul", "email":"rahul32@gmail.com"},
                {"name":"John", "email":"john32bob@gmail.com"}
               ]
}
```

Let's have a quick look at the basic syntax of JSON. JSON syntax is basically considered as a subset of JavaScript syntax; it includes the following −

- Data is represented in name/value pairs.

- Curly braces hold objects and each name is followed by ':'(colon), the name/value pairs are separated by , (comma).
- Square brackets hold arrays and values are separated by ,(comma).

JSON supports the following two data structures −

- **Collection of name/value pairs** − This Data Structure is supported by different programming languages.
- **Ordered list of values** − It includes array, list, vector or sequence etc.

JSON format supports the following data types −

Sr.No.	Type & Description
1	**Number** double- precision floating-point format in JavaScript
2	**String** double-quoted Unicode with backslash escaping
3	**Boolean** true or false
4	**Array** an ordered sequence of values
5	**Value** it can be a string, a number, true or false, null etc
6	**Object** an unordered collection of key:value pairs
7	**Whitespace** can be used between any pair of tokens
8	**null**

	empty

Number

- It is a double precision floating-point format in JavaScript and it depends on implementation.
- Octal and hexadecimal formats are not used.
- No NaN or Infinity is used in Number.

The following table shows the number types −

Sr.No.	Type & Description
1	**Integer** Digits 1-9, 0 and positive or negative
2	**Fraction** Fractions like .3, .9
3	**Exponent** Exponent like e, e+, e-, E, E+, E-

JSON Objects

In JSON, objects refer to dictionaries, which are enclosed in curly brackets, i.e., { }. These objects are written in key/value pairs, where the key has to be a string and values have to be a valid JSON data type such as string, number, object, Boolean or null. Here the key and values are separated by a colon, and a comma separates each key/value pair.

For example:

```
{
"name" : "Jack",
 "employeeid" : 001,
"present" : false,
“emails”:[ravi@gmail.com,”nallan@gmail.com”]
“Address”: [{
```

```
                "Apt no":"A213",
                "Line 1":  "Western plaza",
               "City":"Hyderabad",
              "State":"Telangana",
             "Pincode":500005
  },
  {
                "Apt no":"A215",
                "Line 1":  "my home jewel",
               "City":"Banglore",
              "State":"Karnataka",
             "Pincode":500008
  }
  ]

  }
```

Arrays

Arrays are the lists that are represented by the square brackets, and the values have commas in between them. They can contain mix data types, i.e., a single array can have strings, Boolean, numbers.

For example:

Example 1: [1, 2, 7.8, 5, 9, 10];

Example 2: ["red", "yellow", "green"];

Example 3: [8, "hello", null, true];

In the above, example 1 is an array of numbers, example 2 is an array of strings, and example 3 is an array of mix data types.

JSON Arrays

In JSON, arrays can be understood as a list of objects, which are mainly enclosed in square brackets [ ]. An array value can be a string, number, object, array, boolean or null.

Example 1:

```
“pizzas”: [
{
"PizzaName" : "Country Feast",
"Base" : "Cheese burst",
"Toppings" : ["Jalepenos", "Black Olives", "Extra cheese", "Sausages", "Cherry to
matoes"],
"Spicy" : "yes",
"Veg" : "yes"
},
{
"PizzaName" : "Veggie Paradise",
"Base" : "Thin crust",
"Toppings" : ["Jalepenos", "Black Olives",  "Onions", "Cherry"],
"Spicy" : true,
"Veg" : "yes"
}
]
```

In the above example, the object "Pizza" is an array. It contains five objects, i.e., PizzaName, Base, Toppings, Spicy, and Veg.

Nesting

Nesting involves keeping the arrays and objects inside of each other. We can put the arrays inside objects, objects inside arrays, arrays inside arrays, etc. We can say that json file is a big object with lots of objects and arrays inside.

Example:

```
{
  "song" :
          {
              "title" :  "Hey Dude";
               "artist": "The Beatles";
                "musicians": ["John Lennon", "Paul McCratney", "Ringo Starr"];
           }
}
```

In the above code, the song starts with a curly bracket. Therefore, a song is an object. It contains three key-value pairs wherein title, artist and musicians are the keys.'

Another Example:

A JSON object can have another object also. Let's see a simple example of JSON object having another object.

```
{
    "firstName": "Sonoo",
    "lastName": "Jaiswal",
    "age": 27,
    "address" : {
       "streetAddress": "Plot-6, Mohan Nagar",
       "city": "Ghaziabad",
```

```
            "state": "UP",
            "postalCode": "201007"
        }
    }
```

JSON Comments

JSON doesn't support comments. It is not a standard.

But you can do some tricks such as adding extra attribute for comment in JSON object, for example:

Here, "comments" attribute can be treated as comment.

```
{
    "employee": {
        "name":       "Bob",
        "salary":      56000,
        "comments":    "He is a nice man"
    }
}
```

Accessing JSON using python

Python has a very good support for JSON. The python json module will help us parse json string in to python objects making it easy to process data in python. The json module converts json string in to python dictionaries and lists. The json package can also convert python objects in to JSON strings.

Converting JSON string to python object

JSON data is frequently stored in strings. This is a common scenario when working with APIs. The JSON data would be stored in string variables before it can be parsed. As a result, the most common task related to JSON is to parse the JSON string into the Python dictionary. The JSON module can take care of this task easily.

The first step would be importing the Python json module. This module contains two important functions – **loads (load string)** and **load (load bytes)**

Sample Json:

```
{
    "name": "United States",
    "population": 331002651,

}
```

```
# JSON string
country = '{"name": "United States", "population": 331002651}'

print(type(country))

c_dict=json.loads(country)

print(type(c_dict)). --Dictionary
```

Access the json just how you access a dictionary in python

```
print(c_dict[“name”])
```

If the input string is a list, it will convert to python list

```
countries = '["United States", "Canada"]'
counties_list= json.loads(countries)

print(type(counties_list))

# OUTPUT:  <class 'list'>
```

Converting a JSON file to a python object

```
import json

with open('/Users/ravi/Downloads/countries.json') as f:
  data = json.load(f)

print(type(data))
```

Converting a python object JSON object

```
import json

languages = ["English","French"]
country = {
    "name": "Canada",
    "population": 37742154,
    "languages": languages,
    "president": None,
}

country_string = json.dumps(country)
print(country_string)
```

Converting a python object JSON file

```
import json

# Tuple is encoded to JSON array.
languages = ("English", "French")
# Dictionary is encoded to JSON object.
country = {
    "name": "Canada",
```

```
    "population": 37742154,
    "languages": languages,
    "president": None,
}

with open('countries_exported.json', 'w') as f:
    json.dump(country, f)
```