**Madiha Aimon Tappal**

[**madihaaimon@gmail.com**](mailto:madihaaimon@gmail.com)

**Data Engineering Batch – 1**

**Day – 9 Assignment**

**Python**

**JSON**

JSON, which stands for JavaScript Object Notation, is a lightweight data interchange format. It is easy for humans to read and write, and it is easy for machines to parse and generate. JSON is often used to transmit data between a server and a web application, as an alternative to XML (eXtensible Markup Language).

Key features of JSON include:

1. **Data Structure:** JSON is built on two structures:
   * **Objects:** Unordered collections of key-value pairs. In various programming languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
   * **Arrays:** Ordered lists of values. In most languages, this is realized as an array, vector, list, or sequence.
2. **Easy to Read and Write:** JSON data is represented in a human-readable format, making it easy for both humans and machines to understand.
3. **Lightweight:** JSON is a simple and lightweight data interchange format compared to XML. It uses minimal punctuation and does not have as many features as XML, which can make it faster and easier to parse.
4. **Language Independence:** JSON is language-independent, meaning it can be used with any programming language. There are libraries and parsers available for most programming languages to work with JSON data.
5. **Widely Supported:** JSON is supported by a wide range of programming languages and has become a standard data format in web development.

**Here's a simple example of JSON:**

{

"name": "John Doe",

"age": 30,

"city": "New York",

"isStudent": false,

"hobbies": ["reading", "coding", "traveling"],

"address": {

"street": "123 Main St",

"zipCode": "10001"

}

}

In this example:

* The data is represented as key-value pairs.
* Arrays are denoted by square brackets (**[]**).
* Objects are denoted by curly braces (**{}**).

To use JSON in a programming context, you can:

1. **Serialize:** Convert a data structure (like an object or array) into a JSON-formatted string. This is useful for sending data from a client to a server or for storing data.
2. **Deserialize:** Convert a JSON-formatted string into a data structure. This is useful for receiving and processing data from a server.

Most programming languages have built-in or third-party libraries to handle JSON serialization and deserialization. For example, in JavaScript, you can use **JSON.stringify()** to serialize an object to a JSON string and **JSON.parse()** to deserialize a JSON string back into an object.

**Features of JSON:**

* **Easy to understand:** JSON is easy to read and write.
* **Format:** It is a text-based interchange format. It can store any kind of data in an array of video, audio, and image anything that you required.
* **Support:** It is light-weighted and supported by almost every language and OS. It has a wide range of support for the browsers approx each browser supported by JSON.
* **Dependency:** It is an Independent language that is text-based. It is much faster compared to other text-based structured data.

**JSON Syntax Rules:**Data is in name/value pairs and they are separated by commas. It uses curly brackets to hold the objects and square brackets to hold the arrays.

**Example:**

* **JavaScript**

|  |
| --- |
| {      "Courses": [          {              "Name" : "Java Foundation",              "Created by" : "Hexa",              "Content" : [ "Java Core", "JSP",                          "Servlets", "Collections" ]          },            {              "Name" : "Data Structures",              "also known as" : "Interview Preparation Course",              "Topics" : [ "Trees", "Graphs", "Maps" ]          }      ]  } |

* The top-level structure is an object with a key **"Courses"**, and its corresponding value is an array denoted by square brackets **[]**.
* Inside the array are two objects, each representing a course.
  + **Java Foundation Course:**
    - **"Name"**: "Java Foundation"
    - **"Created by"**: "Hexa"
    - **"Content"**: An array listing the content of the course, including "Java Core," "JSP," "Servlets," and "Collections."
  + **Data Structures Course:**
    - **"Name"**: "Data Structures"
    - **"also known as"**: "Interview Preparation Course"
    - **"Topics"**: An array listing the topics covered in the course, including "Trees," "Graphs," and "Maps."

This structure allows you to represent information about multiple courses, each with its own set of properties and content.

**Advantages of JSON:**

1. JSON stores all the data in an array so data transfer makes easier. That’s why JSON is the best for sharing data of any size even audio, video, etc.
2. Its syntax is very easy to use. Its syntax is very small and light-weighted that’s the reason that it executes and response in a faster way.
3. JSON has a wide range for the browser support compatibility with the operating systems, it doesn’t require much effort to make it all browser compatible.
4. On the server-side parsing the most important part that developers want, if the parsing will be fast on the server side then the user can get the fast response, so in this case JSON server-side parsing is the strong point compare tot others.

**Disadvantages of JSON:**

* The main disadvantage for JSON is that there is no error handling in JSON, if there was a slight mistake in the JSON script then you will not get the structured data.
* JSON becomes quite dangerous when you used it with some unauthorized browsers. Like JSON service return a JSON file wrapped in a function call that has to be executed by the browsers if the browsers are unauthorized then your data can be hacked.
* JSON has limited supported tools that we can use during JSON development.

JSON stands for 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. To use this feature, we import the Python JSON package into Python script. The text in JSON is done through quoted-string which contains a value in key-value mapping within { }. It is similar to the dictionary in Python.

**Function Used**

**json.load():** json.load() function is present in Python built-in ‘JSON’ module. This function is used to parse the JSON string.

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

**JSON (JavaScript Object Notation):**

**JSON** is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It is often used to transmit data between a server and a web application. JSON data is represented as key-value pairs and supports arrays and nested structures.

**Python json Module:**

Python provides a built-in module called **json** for working with JSON data. The **json** module includes two methods relevant to our discussion:

1. **json.loads() - Load String:**
   * The **json.loads()** function is used to parse a JSON-formatted string and convert it into a Python object, usually a dictionary.
   * The name **loads** stands for "load string."

Example:

import json

json\_string = '{"key": "value", "number": 42}'

python\_dict = json.loads(json\_string)

1. **json.load() - Load from File:**

* If your JSON data is in a file rather than a string, you can use **json.load()** to load the JSON content from a file into a Python object.

Example:

import json

with open('data.json', 'r') as file:

python\_dict = json.load(file)

### Converting JSON String to Dictionary:

1. **Import the json module:**

import json

1. **JSON String:**

* Have a JSON-formatted string that you want to convert to a Python dictionary.

Example:

json\_string = '{"key": "value", "number": 42}'

1. **Use json.loads():**

* The **json.loads()** function parses the JSON string and returns a Python object (usually a dictionary).

Example:

python\_dict = json.loads(json\_string)

1. **Access the Dictionary:**

* Now, **python\_dict** is a standard Python dictionary containing the data from the JSON string.

Example:

print(python\_dict)

|  |
| --- |
| # Import JSON module  **import** json    # Define JSON string  jsonString **=** '{ "id": 121, "name": "Naveen", "course": "MERN Stack"}'    # Convert JSON String to Python  student\_details **=** json.loads(jsonString)    # Print Dictionary  **print**(student\_details)    # Print values using keys  print(student\_details['name'])  print(student\_details['course']) |

**Output**

{'id': 121, 'name': 'Naveen', 'course': 'MERN Stack'}

Naveen

MERN Stack

**Convert JSON File to Python Object**

**Python json Module:**

Python's built-in **json** module provides functionality for encoding and decoding JSON data. Two main functions are relevant for reading JSON data from a file:

1. **json.load() - Load from File:**
   * The **json.load()** function reads the content of a JSON file and parses it into a Python object.
   * This function is suitable when you have a JSON file and want to load its content into a Python object.
   * Example:

import json

with open('data.json', 'r') as file:

python\_object = json.load(file)

1. **json.loads() - Load String:**

* The **json.loads()** function is used to parse a JSON-formatted string and convert it into a Python object.
* Example:

import json

json\_string = '{"key": "value", "number": 42}'

python\_object = json.loads(json\_string

### Steps to Convert JSON File to Python Object:

* + 1. **Import the json Module**
    2. **Specify the File Path**
    3. **Open the File**
    4. **Use json.load()**
    5. **Access the Python Object**

Below is the JSON file that we will convert to Python dictionary using **json.load()** method.

In the below code, firstly we open the “data.json” file using file handling in Python and then convert the file to Python object using the json.load() method we have also print the type of data after conversion and print the dictionary.

* Python3

|  |
| --- |
| # Python program to demonstrate  # Conversion of JSON data to  # dictionary    # importing the module  **import** json    # Opening JSON file  with open('data.json') as json\_file:      data **=** json.load(json\_file)        # Print the type of data variable  **print**("Type:", type(data))        # Print the data of dictionary  **print**("\nPeople1:", data['people1'])      print("\nPeople2:", data['people2']) |
|  |

**Convert Nested JSON Object to Dictionary**

In this example, we will convert the nested JSON into a Python dictionary. For JSON data we will use the same JSON file used in the above example.

* Python3

|  |
| --- |
| # importing the module  **import** json    # Opening JSON file  with open('data.json') as json\_file:      data **=** json.load(json\_file)        # for reading nested data [0] represents      # the index value of the list  **print**(data['people1'][0])        # for printing the key-value pair of      # nested dictionary for loop can be used      print("\nPrinting nested dictionary as a key-value pair\n")  **for** i **in** data['people1']:  **print**("Name:", i['name'])          print("Website:", i['website'])          print("From:", i['from'])          print() |

**Convert JSON String to Dictionary in Python**

In this example, we will convert the json string into Python dictionary using json.loads() method. Firstly, we will import JSON module. Create a json string and store it in a variable ‘json\_string’ after that we will convert the json string into dictionary by passing ‘json\_string’ into json.loads() as argument and store the converted dictionary in ‘json\_dict’. Finally, print the Python dictionary.

* Python3

|  |
| --- |
| **import** json    # JSON string  json\_string **=** '{"Name": "Suezen", "age": 23, "Course": "DSA"}'    # Convert JSON string to dictionary  json\_dict **=** json.loads(json\_string)    print(json\_dict) |

**Output**

{'Name': 'Suezen', 'age': 23, 'Course': 'DSA'}

JSON is a lightweight data format for data interchange that can be easily read and written by humans, and easily parsed and generated by machines. It is a complete language-independent text format. To work with JSON data, Python has a built-in package called JSON.

**Example of JSON String**

s = '{"id":01, "name": "Emily", "language": ["C++", "Python"]}'

The syntax of JSON is considered a subset of the syntax of JavaScript including the following:

·        **Name/Value pairs:** Represents Data, the name is followed by a colon(**:**), and the Name/Value pairs are separated by a comma(**,**).

·        **Curly braces:** Holds objects.

·        **Square brackets:** Hold arrays with values separated by a comma (**,**).

Keys/Name must be strings with double quotes and values must be data types amongst the following:

·        String

·        Number

·        Object (JSON object)

·        array

·        Boolean

·        Null

**Example of JSON file:**

 {  
    "employee": [  
   {  
      "id": "01",  
      "name": "Amit",  
          "department": "Sales"  
   },  
   {  
      "id": "04",  
      "name": "sunil",  
          "department": "HR"  
   }  
    ]  
 }

**Python Parse JSON String**

In the below code, we are going to convert JSON to a Python object. To parse JSON string Python firstly we import the JSON module. We have a JSON string stored in a variable **’employee’** and we convert this JSON string to a Python object using **json.loads()** method of JSON module in Python. After that, we print the name of an employee using the key ‘name’.

* Python3

|  |
| --- |
| # Python program to convert JSON to Python  **import** json    # JSON string  employee **=**'{"id":"09", "name": "Nitin", "department":"Finance"}'    # Convert string to Python dict  employee\_dict **=** json.loads(employee)  print(employee\_dict)    **print**(employee\_dict['name']) |

**Output**

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

Nitin

# Read, Write and Parse JSON using Python

**Python read JSON file**

Let’s suppose we have a JSON file that looks like this.

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.

* Python3

|  |
| --- |
| **import** json    # Opening JSON file  f **=** open('data.json',)    # returns JSON object as  # a dictionary  data **=** json.load(f)    # Iterating through the json  # list  **for** i **in** data['emp\_details']:      print(i)    # Closing file  f.close() |

**Convert Python Dict to JSON**

In the below code, we are converting a Python dictionary to a JSON object using json.dumps() method of JSON module in Python. We first import the JSON module and then make a small dictionary with some key-value pairs and then passed it into json.dumps() method with ‘indent=4’ to convert this Python dictionary into a JSON object. As we have given the value of indent to 4 there are four whitespaces before each data as seen in the output.

* Python3

|  |
| --- |
| # Python program to convert  # Python to JSON  **import** json    # Data to be written  dictionary **=** {    "id": "04",    "name": "sunil",    "department": "HR"  }    # Serializing json  json\_object **=** json.dumps(dictionary, indent **=** 4)  print(json\_object) |

**Output**

{

"id": "04",

"name": "sunil",

"department": "HR"

}

The following types of Python objects can be converted into JSON strings:

·        dict

·        list

·        tuple

·        string

·        int

·        float

·        True

·        False

·        None

Python objects and their equivalent conversion to JSON:

|  |  |
| --- | --- |
| **Python** | **JSON Equivalent** |
| dict | object |
| list, tuple | array |
| str | string |
| int, float | number |
| True | true |
| False | false |
| None | null |

**Writing JSON to a file in Python**

We can write JSON to file using json.dump() function of JSON module and file handling in Python. In the below program, we have opened a file named sample.json in writing mode using **‘w’**. The file will be created if it does not exist. Json.dump() will transform the Python dictionary to a JSON string and it will be saved in the file sample.json.

* Python3

|  |
| --- |
| # Python program to write JSON  # to a file  **import** json    # Data to be written  dictionary **=**{      "name" : "sathiyajith",      "rollno" : 56,      "cgpa" : 8.6,      "phonenumber" : "9976770500"  }    with open("sample.json", "w") as outfile:      json.dump(dictionary, outfile) |

**Python Pretty Print JSON**

When we convert a string to JSON the data is in a less readable format. To make it more readable we can use pretty printing by passing additional arguments in json.dumps() function such as **indent** and **sort\_keys** as used in the below code.

* Python3

|  |
| --- |
| # Python program to convert JSON to Python  **import** json    # JSON string  employee **=**'{"id":"09", "name": "Nitin", "department":"Finance"}'    # Convert string to Python dict  employee\_dict **=** json.loads(employee)    # Pretty Printing JSON string back  **print**(json.dumps(employee\_dict, indent **=** 4, sort\_keys**=** True)) |

**Output**

{

"department": "Finance",

"id": "09",

"name": "Nitin"

}

**An example of a simple JSON file:**

*A simple JSON representation*

As you can see in the example, a single key-value pair is separated by a colon (:) whereas each key-value pairs are separated by a comma (,). Here, “name”, “profile”, “age”, and “location” are the key fields while the corresponding values are “*Amit Pathak*“, “*Software Engineer*“, “24”, “London, UK” respectively.

A nested JSON is a structure where the value for one or more fields can be an another JSON format. For example, follow the below example that we are going to use to convert to CSV format.

**An example of a nested JSON file:**

*A nested JSON example*

In the above example, the key field “*article*” has a value which is another JSON format. JSON supports multiple nests to create complex JSON files if required.