Introduction to Json:

JSON (JavaScript Object Notation) 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 primarily used to store and exchange data between a server and a web application, as well as for configuration files and APIs.

Key Points:

1. Format: JSON data consists of key-value pairs, similar to a dictionary or map in programming.
2. Structure:
    - Objects: Represented by curly braces {}, which contain key-value pairs.
    - Arrays: Represented by square brackets [], containing ordered values.
3. Data Types:
    - String: A sequence of characters in double quotes.
    - Number: Integer or floating-point numbers.
    - Boolean: true or false.
    - Array: A list of values enclosed in square brackets [].
    - Object: A collection of key-value pairs enclosed in curly braces {}.
    - Null: Represents an empty value.

```
{
  "name": "John",
  "age": 30,
  "isStudent": false,
  "courses": ["Math", "Science", "History"],
  "address": {
    "street": "123 Main St",
    "city": "Anytown"
  }
}
```

Advantages of JSON:

- Human-readable: The structure is simple and easy to understand.
- Lightweight: It has a minimal syntax, making it efficient for data transfer.
- Language-independent: JSON can be used in various programming languages (like JavaScript, Python, Java, etc.).

JSON Vs. XML

Feature	JSON (JavaScript Object Notation)	XML (Extensible Markup Language)
Syntax	Uses key-value pairs {}	Uses tags <tag></tag>
Readability	More human-readable	Verbose and complex
Data Size	Lightweight	Heavier due to opening/closing tags
Parsing Speed	Faster	Slower due to extensive parsing
Data Type Support	Supports strings, numbers, arrays, booleans, objects, and null	Only stores text (needs additional parsing for numbers, booleans, etc.)
Usage	Used in APIs, web services, and configuration files	Used in web services, configuration, and document storage
Extensibility	Fixed structure, limited flexibility	Highly extensible with custom tags

JSON Example

```
{
  "student": {
    "name": "Amit",
    "age": 22,
    "subjects": ["Math", "Physics", "Chemistry"],
    "isGraduated": false
  }
}
```

XML Example

```
<student>
   <name>Amit</name>
   <age>22</age>
   <subjects>
      <subject>Math</subject>
      <subject>Physics</subject>
      <subject>Chemistry</subject>
   </subjects>
   <isGraduated>false</isGraduated>
</student>
```

Important:

- JSON is preferred for web applications due to its lightweight structure.
- XML is useful for data storage and document processing.
- JSON is faster and easier to parse than XML.

JSON Object

A JSON object is a collection of key-value pairs enclosed within curly braces {}. Each key is a string, and the values can be of various data types such as strings, numbers, booleans, arrays, other objects, or null.

JSON Object

A JSON object is a collection of key-value pairs enclosed within curly braces {}. Each key is a string, and the values can be of various data types such as strings, numbers, booleans, arrays, other objects, or null.

```
{
  "name": "Rahul",
  "age": 25,
  "city": "Mumbai",
  "isStudent": false
}
```

```
{
  "name": "Rahul",
  "age": 25,
  "city": "Mumbai",
  "isStudent": false
}
```

"name" is a key with the value "Rahul" (string).
"age" is a key with the value 25 (number).
"city" is a key with the value "Mumbai" (string).
"isStudent" is a key with the value false (boolean).

JSON Array

A JSON array is an ordered list of values enclosed in square brackets []. The values inside an array can be strings, numbers, objects, booleans, arrays, or null.

```
{
 "students": [
  { "name": "Amit", "age": 21 },
  { "name": "Neha", "age": 22 },
  { "name": "Ravi", "age": 23 }
 ]
}
```

Here,

- The key "students" holds an array of three objects.
- Each object in the array represents a student with "name" and "age" properties.

JSON arrays are commonly used when dealing with multiple data entries, such as a list of users, products, or any structured data.

JSON Comments

JSON does not support comments officially because it is meant for data exchange, not for configuration or human-readable documentation.

However, you can use a workaround by adding a dummy key for comments.

Workaround for JSON Comments:

```
{
  "_comment": "This is a sample JSON with a
comment",
  "name": "Rahul",
  "age": 25,
  "city": "Mumbai"
}
```

Example: Convert PHP Array to JSON

```
<?php
// Creating a PHP array
$data = array(
    "name" => "Rahul",
    "age" => 25,
    "city" => "Mumbai"
);

// Converting PHP array to JSON
$jsonData = json_encode($data,
JSON_PRETTY_PRINT);

// Display JSON
echo $jsonData;
?>
```

OUTPUT


```
{
   "name": "Rahul",
   "age": 25,
   "city": "Mumbai"
}
```


What is Pretty Print in JSON?

By default, json_encode() in PHP returns JSON in a compact form (without spaces or new lines).

Pretty print makes the JSON output more readable by adding whitespace, indentation, and line breaks.

Example: Decode JSON to PHP Array

```
<?php
// JSON string
$jsonString = '{"name": "Rahul", "age": 25,
"city": "Mumbai"}';

// Convert JSON to PHP associative array
$dataArray = json_decode($jsonString,
true);

// Display PHP array
print_r($dataArray);
?>
```

OUTPUT

```
Array
(
    [name] => Rahul
    [age] => 25
    [city] => Mumbai
)
```

JSON USING PHP

```
<?php
// Creating an associative array
$data = array(
    "name" => "Rahul",
    "age" => 25,
    "city" => "Mumbai",
    "skills" => array("PHP", "JavaScript", "MySQL")
);

// Converting the array into JSON format
$jsonData = json_encode($data,
JSON_PRETTY_PRINT);

// Output the JSON
header('Content-Type: application/json');
echo $jsonData;
?>
```

JSON USING PHP - OUTPUT

```
{
   "name": "Rahul",
   "age": 25,
   "city": "Mumbai",
   "skills": [
      "PHP",
      "JavaScript",
      "MySQL"
   ]
}
```

- The json_encode() function is used to convert the PHP array into a JSON string.
- JSON_PRETTY_PRINT makes the JSON output more readable.
- header('Content-Type: application/json'); ensures that the response is treated as JSON.
- Finally, echo $jsonData; prints the JSON output.

JSON USING JAVA

```
import java.util.HashMap;
import java.util.Map;

public class Main {
    public static void main(String[] args) {
        // Creating a JSON-like structure using
HashMap
        Map<String, Object> data = new HashMap<>();
        data.put("name", "Rahul");
        data.put("age", 25);
        data.put("city", "Mumbai");

        // Creating a JSON-like Array manually
        String[] skills = {"Java", "Spring Boot",
"MySQL"};
```

JSON USING JAVA

```
// Building JSON String Manually
StringBuilder json = new StringBuilder("{\n");
json.append(" \"name\":
\"").append(data.get("name")).append("\",\n");
json.append(" \"age\":
").append(data.get("age")).append(",\n");
json.append(" \"city\":
\"").append(data.get("city")).append("\",\n");
json.append(" \"skills\": [");

for (int i = 0; i < skills.length; i++) {
json.append("\"").append(skills[i]).append("\"");
if (i < skills.length - 1) {
json.append(", ");
}
}

json.append("]\n}");

// Output JSON
System.out.println(json.toString());
}
}
```

JSON USING JAVA - OUTPUT

```
{
   "name": "Rahul",
   "age": 25,
   "city": "Mumbai",
   "skills": ["Java", "Spring Boot", "MySQL"]
}
```

JSON USING AJAX

index.html

```
<script>
    $(document).ready(function() {
      $("#fetchData").click(function() {
        $.ajax({
          url: "data.json", // JSON file or API endpoint
          type: "GET",
          dataType: "json",
          success: function(response) {
            let output = "<ul>";
            response.forEach(item => {
              output += `<li>${item.name} - ${item.age} years old</li>`;
            });
            output += "</ul>";
            $("#result").html(output);
          },
          error: function(xhr, status, error) {
            console.error("Error:", error);
          }
        });
      });
    });
  </script>
```

JSON USING AJAX

data.json

```
[
   { "name": "Amit", "age": 25 },
   { "name": "Priya", "age": 22 },
   { "name": "Raj", "age": 30 }
]
```