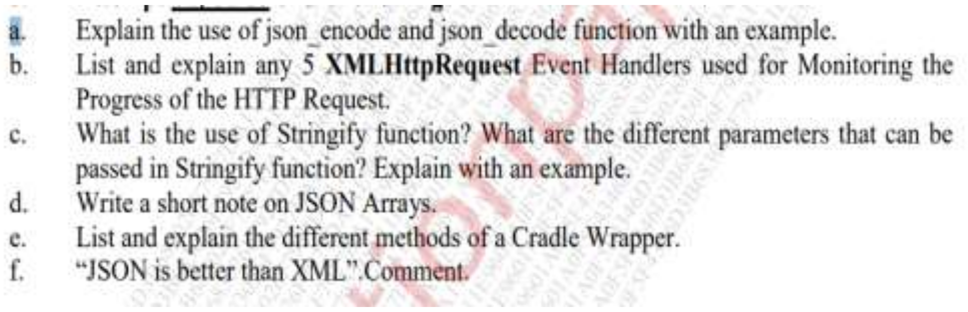
UNIT -V

JSON

JSON: Introduction, JSON Grammar, JSON Values, JSON Tokens, Syntax, JSON vs XML, Data Types, Objects, Arrays, Creating JSON, JSON Object, Parsing JSON, Persisting JSON, Data Interchange, JSON PHP, JSON HTML, JSONP

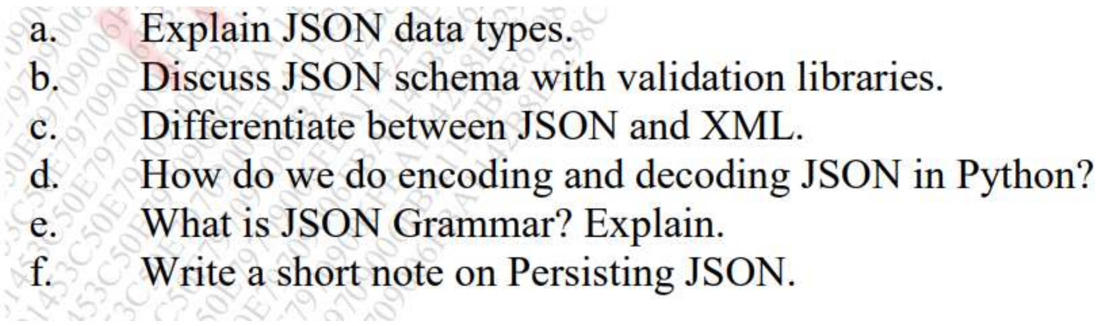
PYQ : (Previous Year Mumbai University Question)

Nov – 18



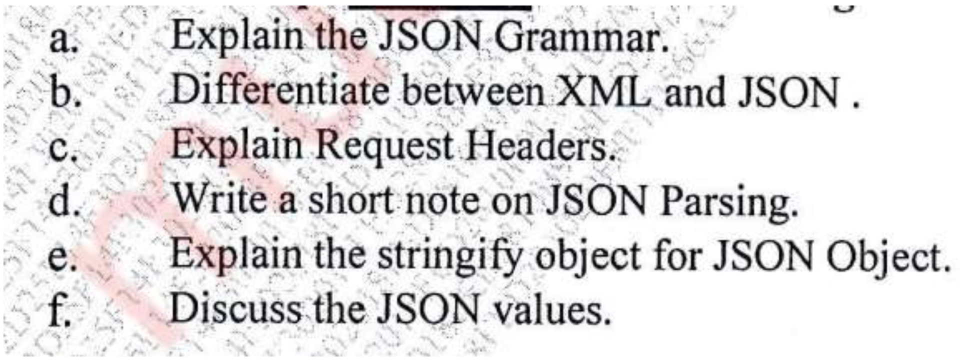
a. Explain the use of json_encode and json_decode function with an example.
b. List and explain any 5 **XMLHttpRequest** Event Handlers used for Monitoring the Progress of the HTTP Request.
c. What is the use of Stringify function? What are the different parameters that can be passed in Stringify function? Explain with an example.
d. Write a short note on JSON Arrays.
e. List and explain the different methods of a Cradle Wrapper.
f. "JSON is better than XML".Comment.

Apr -19



a. Explain JSON data types.
b. Discuss JSON schema with validation libraries.
c. Differentiate between JSON and XML.
d. How do we do encoding and decoding JSON in Python?
e. What is JSON Grammar? Explain.
f. Write a short note on Persisting JSON.

Nov – 19



a. Explain the JSON Grammar.
b. Differentiate between XML and JSON .
c. Explain Request Headers.
d. Write a short note on JSON Parsing.
e. Explain the stringify object for JSON Object.
f. Discuss the JSON values.

Nov – 22

a. What is the use of stringify method? Explain with syntax.
b. Explain the six members of the web storage Interface.
c. Explain the structure of Hypertext Transfer Protocol (HTTP) – Request.
d. Explain the JSON Grammar.
e. Give an overview about JavaScript Object Notation (JSON). Also explain about JSON tokens.
f. Explain about JSON parsing with syntax.

Dec – 23

a. List and explain the various data types available in JSON.
b. What are the differences between JSON and XML? Provide examples.
c. What is the XMLHttpRequest object? List and explain the Request Methods associated with it.
d. Write a note on Stringify() method.
e. Explain JSON HTML with an example.
f. Write a short note on JSON Arrays.

Nov – 24

a. List and explain the various data types available in JSON.
b. What are the differences between JSON and XML? Provide examples.
c. What is the XMLHttpRequest object? List and explain the Request Methods associated with it.
d. Write a note on Stringify() method.
e. Explain JSON HTML with an example.
f. Write a short note on JSON Arrays.

Introduction:

JSON :

### JSON Introduction (for 5 Marks - TYBSC Mumbai University)

1. What is JSON?
- JSON stands for JavaScript Object Notation.
- It is a lightweight data format used for storing and exchanging data between a server and a client (like web applications).
- It is a human-readable format, making it easy for people to understand.

2. Structure of JSON:
- Key-Value Pairs: JSON consists of data in the form of key-value pairs.

Example:

```
json
{
  "name": "John",
  "age": 30
}
```

- "name" is the key, and "John" is the value.

- Data Types Supported in JSON:
  - String (e.g., "hello")
  - Number (e.g., 45)
  - Boolean (e.g., true or false)
  - Array (e.g., ["apple", "banana"])
  - Object (e.g., {"name": "John", "age": 30})

3. Why is JSON Popular?
- Easy to Understand: JSON is easy for humans to read and write.
- Language Independent: It is not tied to any specific programming language. Most modern languages support JSON (Java, Python, JavaScript, etc.).
- Lightweight: It is a compact format, making it easy to transmit over the web.
- Widely Used in APIs: JSON is used in web services and APIs to exchange data between different systems.

4. JSON vs. XML:
- JSON is often compared to XML (another data format).
- JSON is simpler and easier to use than XML.
- JSON is more compact and faster to process than XML.

5. Real-World Usage:
- Web Development: JSON is commonly used in web applications to send data between clients and servers (for example, when fetching data from a website).
- APIs: When interacting with web services (like Google Maps, Facebook APIs), JSON is the format used to exchange information.

By using JSON, developers can efficiently transfer structured data between different systems in a readable and easy-to-use format, making it an essential tool for modern programming.

JSON Grammer :

### JSON Grammar (for 5 Marks - TYBSC IT Mumbai University)

1. Basic Structure of JSON
- JSON is a way to represent data using key-value pairs.
- The entire JSON structure consists of two key elements:
  - Objects (Curly braces {})
  - Arrays (Square brackets [])

2. JSON Object Syntax
- A JSON Object is written inside curly braces {}.
- It contains one or more key-value pairs, separated by commas ,.

  Example:
```
json
{
  "key1": "value1",
  "key2": "value2"
}
```

  - Each key is a string (in double quotes " ").
  - Each value can be a string, number, boolean, object, array, or null.

3. JSON Array Syntax
- A JSON Array is written inside square brackets [].
- It contains a list of values, separated by commas ,.

  Example:
```
json
["value1", "value2", "value3"]
```

  - Arrays can contain multiple data types, including objects and arrays inside them.

4. Data Types in JSON

JSON supports the following data types:
- String: A sequence of characters enclosed in double quotes.
  Example: "hello"
- Number: Numeric values (without quotes).
  Example: 123
- Boolean: Logical values true or false.
  Example: true
- Array: A collection of values enclosed in square brackets.

Example: ["apple", "banana", "cherry"]
- Object: Key-value pairs enclosed in curly braces.
  Example: {"name": "John", "age": 30}
- Null: Represents the absence of a value.
  Example: null

5. JSON Rules (Grammar Guidelines)
- Keys:
  - Keys must be strings in double quotes (").
  - No special characters or spaces allowed in keys.

- Values:
  - Values can be strings, numbers, booleans, objects, arrays, or null.

- Commas:
  - Use commas , to separate different key-value pairs or items in an array.

- No Trailing Commas:
  - Do not add a comma after the last key-value pair or array item.

- Whitespace:
  - JSON can include spaces and line breaks to improve readability, but these are ignored when parsing the data.

### Example:
json
```
{
  "name": "Alice",
  "age": 25,
  "isStudent": true,
  "courses": ["Math", "Science", "English"],
  "address": {
    "city": "Mumbai",
    "zipCode": 400001
  },
  "graduated": null
}
```

This example includes all core JSON elements: strings, numbers, booleans, arrays, objects, and null values.

By understanding JSON's grammar, developers can effectively format data, making it easy to share between applications in a structured and readable way.

JSON values :

### JSON Values (for 5 Marks - TYBSC IT Mumbai University)

In JSON (JavaScript Object Notation), values are the fundamental part of the key-value pairs that make up the data structure. These values can be of various types, each serving a different purpose in representing data. Below is a detailed explanation of the types of values supported by JSON:

---

1. Strings
- A string is a sequence of characters enclosed in double quotes " ".
- Strings are used to represent text data.

  Example:
  json
  "name": "John"

  - Here, the value of the key "name" is the string "John".

---

2. Numbers
- JSON supports both integer and floating-point numbers.
- Numbers are written without quotes.

  Example:
  json
  "age": 25,
  "price": 99.99

  - The value of the key "age" is the number 25, and the value of the key "price" is the floating-point number 99.99.

---

3. Booleans

- Boolean values are either true or false.
- They represent logical values (truth values) and are written without quotes.

Example:
```json
"isStudent": true
```

- The value of the key "isStudent" is true.

---

4. Arrays
- An array is a list of values enclosed in square brackets [ ].
- Arrays can hold multiple values, and these values can be of any JSON-supported data type (strings, numbers, booleans, objects, or other arrays).

Example:
```json
"subjects": ["Math", "Science", "English"]
```

- The value of the key "subjects" is an array containing three string values: "Math", "Science", and "English".

---

5. Objects
- A JSON object is a collection of key-value pairs enclosed in curly braces { }.
- The value of a key can itself be another JSON object, creating a nested structure.

Example:
```json
"address": {
  "city": "Mumbai",
  "zipCode": 400001
}
```

- The value of the key "address" is another object containing two key-value pairs: "city" and "zipCode".

---

6. Null
- The null value represents the absence of a value.
- It is useful when you want to indicate that a key exists but currently has no value assigned to it.

  Example:
  json
```
 "graduated": null
```

  - The value of the key "graduated" is null, meaning no value is assigned.

### Example:
json
```
{
  "name": "Alice",
  "age": 25,
  "isStudent": true,
  "subjects": ["Math", "Science", "English"],
  "address": {
    "city": "Mumbai",
    "zipCode": 400001
  },
  "graduated": null
}
```

- "name" is a string.
- "age" is a number.
- "isStudent" is a boolean.
- "subjects" is an array.
- "address" is an object.
- "graduated" is null.

### Summary of JSON Values:
- String: Text data enclosed in double quotes.
- Number: Integer or floating-point number, without quotes.
- Boolean: True or false, without quotes.

- Array: A list of values enclosed in square brackets.
- Object: A nested structure of key-value pairs enclosed in curly braces.
- Null: Represents an empty or non-existent value.

Understanding these JSON values helps in structuring data in a simple, readable, and efficient way, making it easier for communication between systems.

JSON Tokens :

### JSON Tokens (for 5 Marks - TYBSC IT Mumbai University)

In JSON (JavaScript Object Notation), tokens are the smallest elements that are used to construct the entire JSON data structure. These tokens define how data is organized and represented. Below is a detailed explanation of the different types of tokens in JSON:

---

1. Curly Braces { }
- Curly braces are used to define a JSON object.
- A JSON object is a collection of key-value pairs enclosed in {}.

Example:
```
json
{
  "name": "John",
  "age": 30
}
```

- Here, {} defines an object containing two key-value pairs.

---

2. Square Brackets [ ]
- Square brackets are used to define a JSON array.
- A JSON array is a list of values, which can be of any data type (strings, numbers, booleans, objects, or even other arrays).

Example:

```json
["apple", "banana", "cherry"]
```

- Here, [] defines an array containing three string values.

---

3. Colons :
- A colon : is used to separate a key from its value within a key-value pair.
- It always appears between the key and the value inside an object.

  Example:
```json
"name": "Alice"
```

- Here, the colon separates the key "name" from its value "Alice".

---

4. Commas ,
- A comma , is used to separate multiple key-value pairs in a JSON object or multiple values in a JSON array.

  Example (In an object):
```json
{
  "name": "Alice",
  "age": 25
}
```

- Here, the comma separates the key-value pairs "name": "Alice" and "age": 25.

  Example (In an array):
```json
["apple", "banana", "cherry"]
```

- Here, the comma separates the values "apple", "banana", and "cherry".

---

5. Double Quotes "

- Double quotes are used to define strings in JSON.
- Keys in JSON must always be strings, and string values are enclosed in double quotes as well.

Example:
json
```
"city": "Mumbai"
```

- Both the key "city" and the value "Mumbai" are enclosed in double quotes.

---

6. Literal Values (Keywords)
- True, False, and Null are special tokens used to represent boolean and null values.
  - True and False: Represent boolean values, used for logical conditions.
  - Null: Represents the absence of a value or an empty value.

Example:
json
```
{
  "isStudent": true,
  "graduated": null
}
```

- true is a boolean value, and null represents the absence of a value.

### Example:
json
```
{
 "name": "Alice",
 "age": 25,
 "isStudent": true,
 "subjects": ["Math", "Science"],
 "address": {
   "city": "Mumbai",
   "zipCode": 400001
 },
 "graduated": null
```

}

In this example:
- Curly braces {} define the main JSON object and the nested "address" object.
- Square brackets [] define the "subjects" array.
- Colons : separate keys from values.
- Commas , separate key-value pairs and array items.
- Double quotes " define strings (both keys and string values).
- True, False, and Null are used for boolean and null values.

### Summary of JSON Tokens:
1. Curly Braces {}: Used to define objects.
2. Square Brackets []: Used to define arrays.
3. Colon :: Separates keys from values.
4. Comma ,: Separates key-value pairs or values in an array.
5. Double Quotes ": Encloses strings.
6. True, False, Null: Special tokens for boolean and null values.

By understanding JSON tokens, one can accurately construct and parse JSON data, which is essential for data exchange in web development and APIs.

JSON Syntax :

### JSON Syntax (for 5 Marks - TYBSC IT Mumbai University)

JSON (JavaScript Object Notation) has a simple and easy-to-understand syntax used to represent structured data. Below is a detailed explanation of JSON syntax in simple points:

---

1. Data is Represented as Key-Value Pairs
- JSON data is written in the form of key-value pairs.
- Each key must be a string (enclosed in double quotes "), and each value can be a string, number, object, array, boolean, or null.

Example:
json
"name": "John"

---

2. JSON Objects
- A JSON object is a collection of key-value pairs enclosed in curly braces {}.
- Multiple key-value pairs inside an object are separated by commas ,.

Example:
json
```
{
  "name": "John",
  "age": 30
}
```

- Here, the object contains two key-value pairs: "name": "John" and "age": 30.

---

3. JSON Arrays
- A JSON array is a collection of values enclosed in square brackets [].
- The values inside an array can be of any data type (strings, numbers, objects, arrays, booleans, or null), and are separated by commas ,.

Example:
json
```
["apple", "banana", "cherry"]
```

- This array contains three string values.

Array with different data types:
json
```
[25, "Math", true, null]
```

---

4. Strings in JSON
- Strings in JSON must be enclosed in double quotes " ".
- A string can represent text, such as names or messages.

Example:
json
```
"city": "Mumbai"
```

- "city" is the key, and "Mumbai" is the string value.

---

5. Numbers in JSON
- Numbers in JSON are written without quotes.
- They can be integers or floating-point numbers.

Example:
json
```
"age": 25,
"price": 99.99
```

---

6. Boolean Values
- JSON supports two boolean values: true and false (without quotes).

Example:
json
```
"isStudent": true
```

---

7. Null Values
- The null value represents an empty or non-existent value in JSON (without quotes).

Example:
json
```
"graduated": null
```

---

8. Nested Objects and Arrays
- JSON objects and arrays can be nested inside other objects and arrays.
- This allows for complex data structures.

Example (Nested Object):

```json
{
  "name": "John",
  "address": {
    "city": "Mumbai",
    "zipCode": 400001
  }
}
```

- Here, "address" is an object inside the main object.

Example (Array Inside Object):

```json
{
  "name": "Alice",
  "subjects": ["Math", "Science"]
}
```

- Here, "subjects" is an array inside the main object.

---

9. No Trailing Commas
- In JSON, there should be no trailing commas after the last key-value pair or the last item in an array.

Correct:

```json
{
  "name": "John",
  "age": 30
}
```

Incorrect (with trailing comma):

```json
{
  "name": "John",
  "age": 30,
}
```

---

10. Whitespace is Ignored
- JSON allows whitespace (spaces, tabs, or newlines) to make the data more readable. However, the whitespace is ignored when the data is processed.

Example:

```json
{
  "name": "John",
  "age": 30
}
```

- The spaces and line breaks improve readability but are ignored by JSON parsers.

### Example of Well-Formatted JSON:

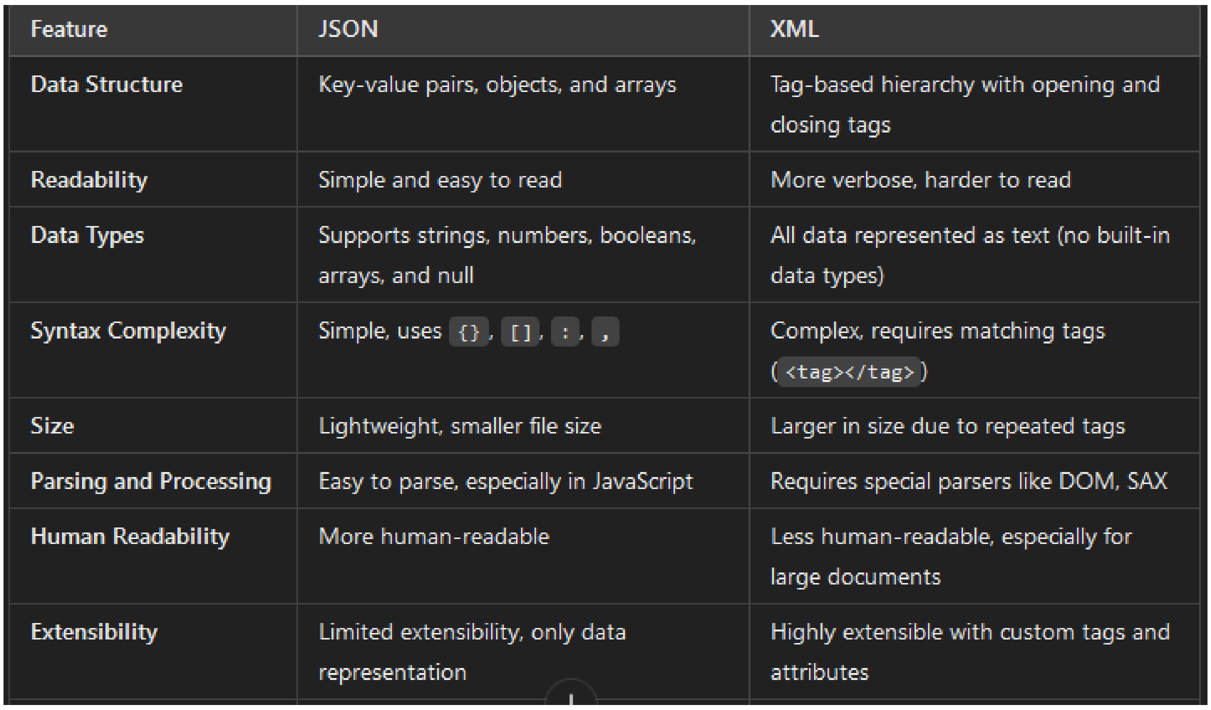
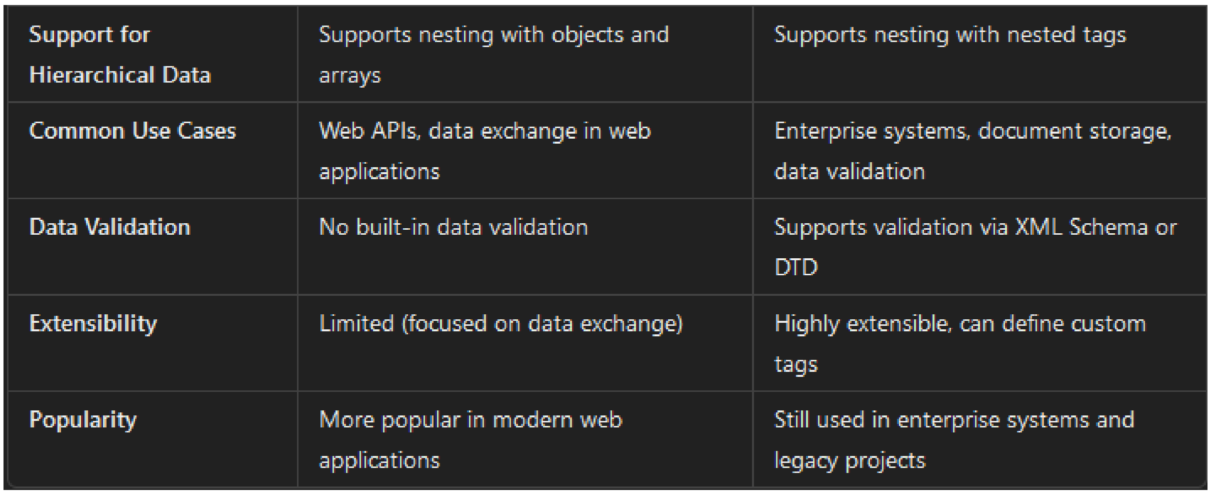
```json
{
  "name": "Alice",
  "age": 25,
  "isStudent": true,
  "subjects": ["Math", "Science", "English"],
  "address": {
    "city": "Mumbai",
    "zipCode": 400001
  },
  "graduated": null
}
```

This JSON structure includes:
- Strings ("Alice", "Mumbai")
- Numbers (25, 400001)
- Booleans (true)
- Arrays (["Math", "Science", "English"])
- Nested objects ("address": { "city": "Mumbai", "zipCode": 400001 })
- Null values (null)

### Summary of JSON Syntax:
1. Key-Value Pairs: Data is represented as key-value pairs.
2. Objects: Enclosed in curly braces {}, key-value pairs are separated by commas.
3. Arrays: Enclosed in square brackets [], values are separated by commas.
4. Strings: Enclosed in double quotes " (for both keys and string values).
5. Numbers: Written without quotes.
6. Booleans: true and false, without quotes.
7. Null: Represents empty values, written as null.
8. Nesting: Objects and arrays can be nested inside each other.
9. No Trailing Commas: No commas after the last key-value pair or array element.
10. Whitespace: Can be used for readability but is ignored during parsing.

By following these simple rules, you can create valid and properly structured JSON data, which is essential for data exchange between systems and web applications.

JSON Vs XML :

Feature	JSON	XML
Data Structure	Key-value pairs, objects, and arrays	Tag-based hierarchy with opening and closing tags
Readability	Simple and easy to read	More verbose, harder to read
Data Types	Supports strings, numbers, booleans, arrays, and null	All data represented as text (no built-in data types)
Syntax Complexity	Simple, uses `{}`, `[]`, `:`, `,`	Complex, requires matching tags (`<tag></tag>`)
Size	Lightweight, smaller file size	Larger in size due to repeated tags
Parsing and Processing	Easy to parse, especially in JavaScript	Requires special parsers like DOM, SAX
Human Readability	More human-readable	Less human-readable, especially for large documents
Extensibility	Limited extensibility, only data representation	Highly extensible with custom tags and attributes
Support for Hierarchical Data	Supports nesting with objects and arrays	Supports nesting with nested tags
Common Use Cases	Web APIs, data exchange in web applications	Enterprise systems, document storage, data validation
Data Validation	No built-in data validation	Supports validation via XML Schema or DTD
Extensibility	Limited (focused on data exchange)	Highly extensible, can define custom tags
Popularity	More popular in modern web applications	Still used in enterprise systems and legacy projects

----

JSON Data Type :

### JSON Data Types (for 5 Marks - TYBSC IT Mumbai University)

JSON (JavaScript Object Notation) supports a limited set of data types that make it easy to represent simple data structures. Below is a detailed explanation of JSON data types in simple points:

### 1. String

- A sequence of characters enclosed in double quotes "".
- Can include letters, numbers, and special characters.
- Used to represent textual data.

Example:
json
```
{
  "name": "Alice"
}
```

### 2. Number
- Represents both integers and floating-point (decimal) numbers.
- No quotes are used around numbers.
- Can represent positive and negative values.

Example:
json
```
{
  "age": 25,
  "height": 5.6
}
```

### 3. Boolean
- Represents true or false values.
- Used for conditions or to represent a binary state.

Example:
json
```
{
  "isStudent": true,
  "hasGraduated": false
}
```

### 4. Array
- An ordered list of values.
- Enclosed in square brackets [].
- Can hold multiple types of data like strings, numbers, objects, or other arrays.

Example:
json
```
{
  "subjects": ["Math", "Science", "English"]
}
```

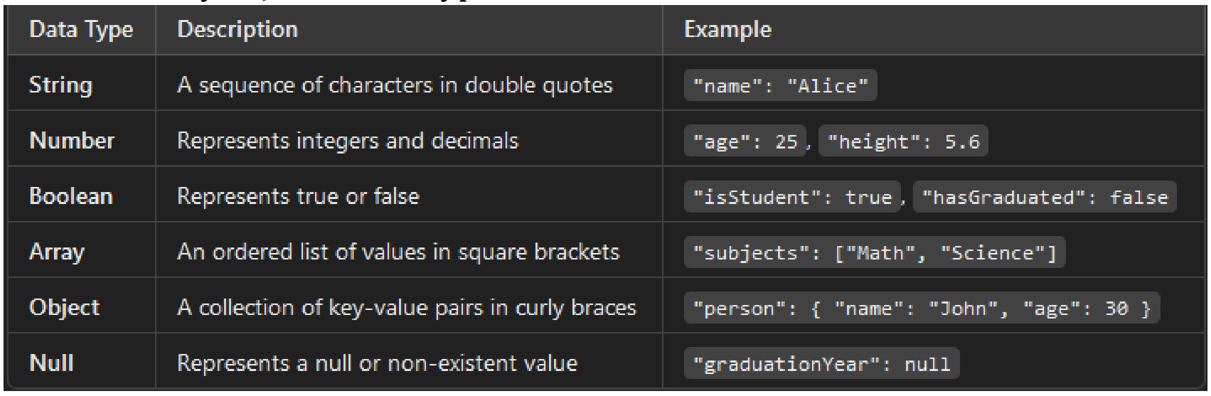
### 5. Object
- Represents a collection of key-value pairs (like a dictionary).
- Enclosed in curly braces {}.
- Each key is a string, and its value can be any JSON data type (string, number, array, object, etc.).

Example:
json
```
{
  "person": {
    "name": "John",
    "age": 30
  }
}
```

### 6. Null
- Represents an empty or non-existent value.
- Used to indicate the absence of data.

Example:
json
```
{
  "graduationYear": null
}
```

### Summary of JSON Data Types:



Data Type	Description	Example
String	A sequence of characters in double quotes	`"name": "Alice"`
Number	Represents integers and decimals	`"age": 25`, `"height": 5.6`
Boolean	Represents true or false	`"isStudent": true`, `"hasGraduated": false`
Array	An ordered list of values in square brackets	`"subjects": ["Math", "Science"]`
Object	A collection of key-value pairs in curly braces	`"person": { "name": "John", "age": 30 }`
Null	Represents a null or non-existent value	`"graduationYear": null`

### Conclusion:
- JSON data types include String, Number, Boolean, Array, Object, and Null.
- These data types allow for simple representation of data that is easy to parse and read across systems. JSON's simplicity is one reason it is widely used in web development and APIs.

JSON Object :

### JSON Objects (for 5 Marks - TYBSC IT Mumbai University)

A JSON object is one of the key data structures in JSON (JavaScript Object Notation). It is used to represent data as a collection of key-value pairs. Below is a detailed explanation of JSON objects in simple points:

### 1. Structure of a JSON Object
- A JSON object is enclosed in curly braces {}.
- Inside the object, data is stored as key-value pairs.
- Each key is a string (in double quotes ""), and each value can be any valid JSON data type (string, number, boolean, array, another object, or null).

Example:
json
{

```
  "name": "John",
  "age": 30,
  "isStudent": false
}
```

### 2. Key-Value Pairs
- Each key is followed by a colon :, and the corresponding value is placed after the colon.
- Keys must be unique and should always be enclosed in double quotes.
- The value can be of any data type, such as:
  - String
  - Number
  - Boolean
  - Array
  - Another JSON Object
  - Null

 Example:
 json
```
{
  "city": "Mumbai",
  "population": 20000000,
  "isCoastal": true
}
```

### 3. Multiple Key-Value Pairs
- Multiple key-value pairs inside an object are separated by commas ,.
- Objects can store as many key-value pairs as needed, making them highly flexible for representing complex data.

 Example:
 json
```
{
  "name": "Alice",
  "age": 25,
  "subjects": ["Math", "Science"],
  "address": {
    "city": "New York",
    "zip": "10001"
  }
}
```

### 4. Nesting of JSON Objects
- JSON objects can be nested inside other objects.
- This nesting allows JSON to represent complex and hierarchical data structures.

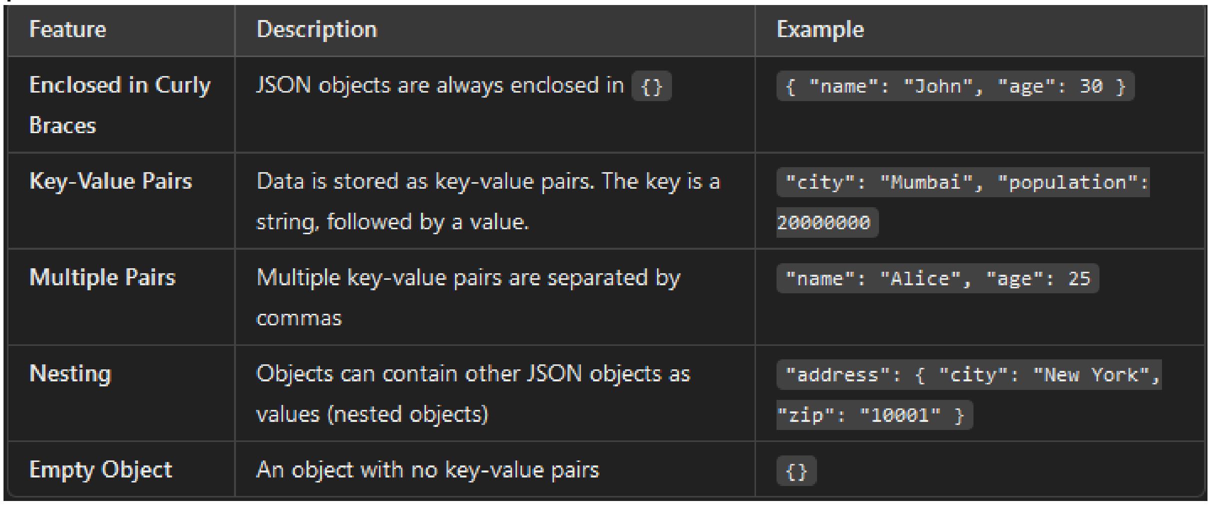
 Example:
 json
```
 {
   "person": {
     "name": "Bob",
     "age": 40,
     "address": {
       "street": "123 Main St",
       "city": "Los Angeles"
     }
   }
 }
```

### 5. Empty JSON Object
- A JSON object can also be empty, containing no key-value pairs, which is represented by just two curly braces {}.

 Example:
 json
```
 {
   "data": {}
 }
```

### Summary of JSON Objects:

| Feature | Description | Example
|



Feature	Description	Example
Enclosed in Curly Braces	JSON objects are always enclosed in `{}`	`{ "name": "John", "age": 30 }`
Key-Value Pairs	Data is stored as key-value pairs. The key is a string, followed by a value.	`"city": "Mumbai", "population": 20000000`
Multiple Pairs	Multiple key-value pairs are separated by commas	`"name": "Alice", "age": 25`
Nesting	Objects can contain other JSON objects as values (nested objects)	`"address": { "city": "New York", "zip": "10001" }`
Empty Object	An object with no key-value pairs	`{}`

### Conclusion:
- JSON objects are the fundamental way to represent structured data in JSON format.
- They consist of key-value pairs, which can be used to represent simple or complex data structures.
- JSON objects allow easy nesting and flexibility, making them ideal for representing data in web applications and APIs.

JSON Array :

### JSON Arrays (for 5 Marks - TYBSC IT Mumbai University)

A JSON array is a way to represent a collection of values in a specific order. Below is a detailed explanation of JSON arrays in simple points:

### 1. Structure of a JSON Array
- JSON arrays are enclosed in square brackets [].
- Inside the array, values are listed separated by commas ,.
- Arrays can hold multiple values of any valid JSON data type:
  - Strings
  - Numbers

- Booleans
- Objects
- Other arrays (nested arrays)
- Null

Example:
json
```
{
  "fruits": ["Apple", "Banana", "Orange"]
}
```

### 2. Elements of an Array
- Each value in the array is called an element.
- JSON arrays can store elements of different types (strings, numbers, booleans, etc.) in the same array.

Example:
json
```
{
  "data": ["Alice", 25, true, null]
}
```

### 3. Order Matters
- JSON arrays maintain the order of elements. The order in which the elements appear in the array is the order in which they are processed.

Example:
json
```
{
  "numbers": [1, 2, 3, 4, 5]
}
```

In this example, 1 is the first element, 2 is the second, and so on.

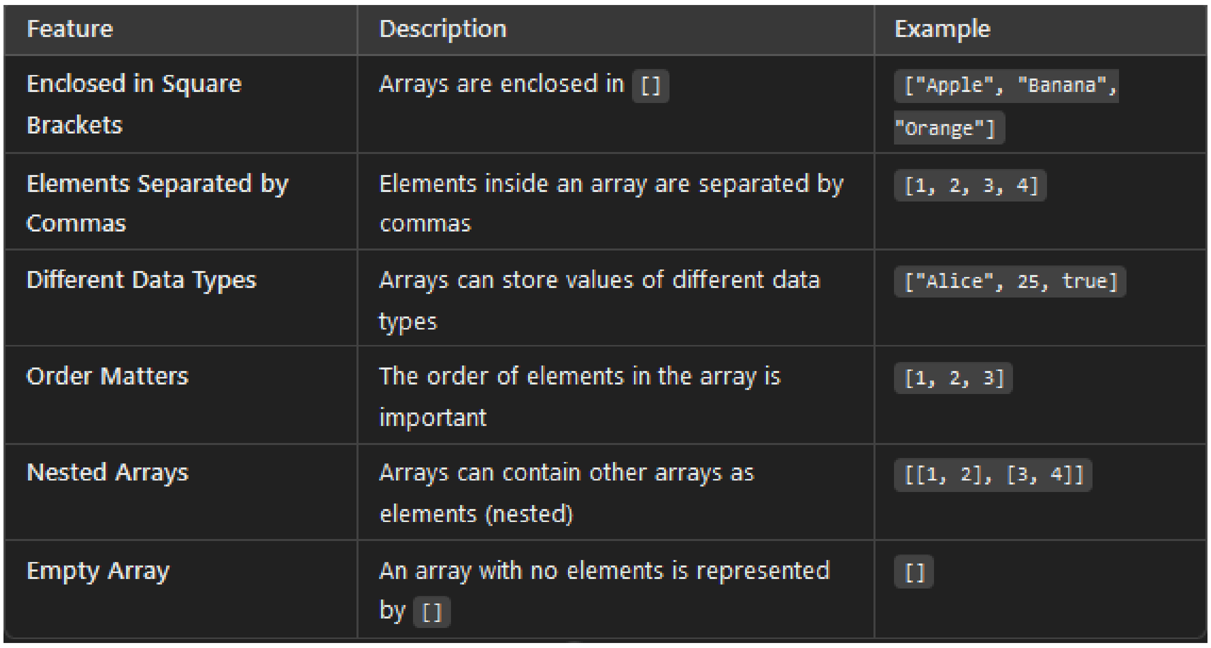
### 4. Nested Arrays
- JSON arrays can contain other arrays as elements, creating nested arrays. This allows for more complex structures like multidimensional arrays.

Example:
json
```
{
  "matrix": [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
  ]
}
```

### 5. Empty Array
- A JSON array can also be empty, containing no elements. This is represented by just two square brackets [].

Example:
json
```
{
  "emptyArray": []
}
```

### Summary of JSON Arrays:



Feature	Description	Example
Enclosed in Square Brackets	Arrays are enclosed in `[]`	`["Apple", "Banana", "Orange"]`
Elements Separated by Commas	Elements inside an array are separated by commas	`[1, 2, 3, 4]`
Different Data Types	Arrays can store values of different data types	`["Alice", 25, true]`
Order Matters	The order of elements in the array is important	`[1, 2, 3]`
Nested Arrays	Arrays can contain other arrays as elements (nested)	`[[1, 2], [3, 4]]`
Empty Array	An array with no elements is represented by `[]`	`[]`

### Conclusion:
- JSON arrays are a way to store ordered lists of values.
- Arrays can contain multiple data types and can even be nested inside each other.
- They are widely used in web applications and APIs for representing lists or collections of data in a simple, human-readable format.

Creating JSON :
### Creating JSON (for 5 Marks - TYBSC IT Mumbai University)

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. Here is a detailed explanation of creating JSON in simple points:

### 1. Understand the Structure of JSON
- JSON consists of objects and arrays.
- Objects are collections of key-value pairs enclosed in curly braces {}.
- Arrays are ordered lists of values enclosed in square brackets [].

### 2. Creating a JSON Object
- To create a JSON object, start with { and end with }.
- Inside, define key-value pairs using the format "key": value.
- Each key must be a string (enclosed in double quotes) and must be unique within the object.
- Values can be strings, numbers, arrays, objects, booleans, or null.

Example:
```
json
{
  "name": "John",
  "age": 30,
  "isStudent": false
}
```

### 3. Creating a JSON Array
- To create a JSON array, start with [and end with].
- Inside, list the values separated by commas.
- Values can be of any JSON data type, including other arrays or objects.

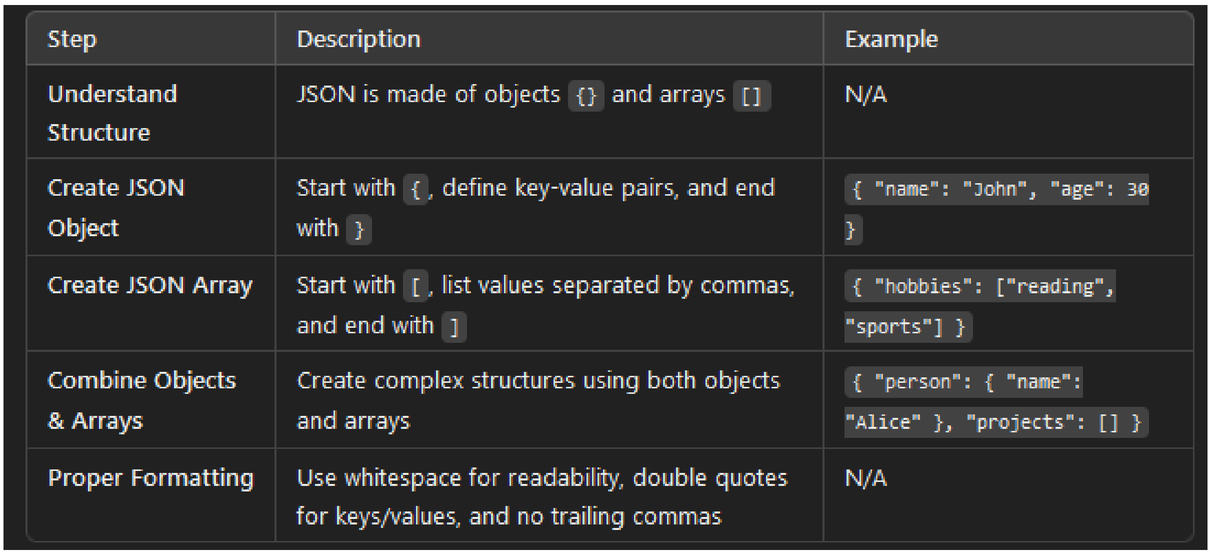
Example:
```
json
{
  "hobbies": ["reading", "sports", "music"]
}
```

### 4. Combining Objects and Arrays
- You can combine objects and arrays to create complex data structures.
- An object can contain arrays, and an array can contain objects.

Example:
```
json
{
  "person": {
    "name": "Alice",
    "age": 25,
    "skills": ["Java", "Python", "JavaScript"]
  },
  "projects": [
    {
      "title": "Project 1",
      "duration": "3 months"
    },
    {
      "title": "Project 2",
      "duration": "6 months"
    }
  ]
}
```

### 5. Using Proper Formatting
- Whitespace: Use spaces or new lines to make JSON readable. They are ignored by parsers but help in readability.
- Quotes: Always use double quotes for keys and string values.
- No Trailing Commas: Do not add a comma after the last key-value pair in an object or the last element in an array.

### Summary of Creating JSON:



Step	Description	Example
Understand Structure	JSON is made of objects `{}` and arrays `[]`	N/A
Create JSON Object	Start with `{`, define key-value pairs, and end with `}`	`{ "name": "John", "age": 30 }`
Create JSON Array	Start with `[`, list values separated by commas, and end with `]`	`{ "hobbies": ["reading", "sports"] }`
Combine Objects & Arrays	Create complex structures using both objects and arrays	`{ "person": { "name": "Alice" }, "projects": [] }`
Proper Formatting	Use whitespace for readability, double quotes for keys/values, and no trailing commas	N/A

### Conclusion:
- Creating JSON involves understanding its structure and syntax.
- By combining objects and arrays, you can represent complex data in a simple format.
- JSON is widely used in web applications and APIs due to its simplicity and ease of use.

Parsing JSON :

### Parsing JSON (for 5 Marks - TYBSC IT Mumbai University)

Parsing JSON (JavaScript Object Notation) is the process of converting a JSON string into a data structure that can be easily used in programming languages. This guide explains how to parse JSON in simple points.

### 1. What is JSON Parsing?
- JSON parsing involves converting a JSON-formatted string into a JavaScript object or another data structure in programming languages.
- It allows you to access and manipulate the data stored in the JSON format.

### 2. How to Parse JSON in JavaScript
- In JavaScript, the global method JSON.parse() is used to parse a JSON string.
- It takes a JSON string as an argument and converts it into a JavaScript object.

Example:
javascript
```
const jsonString = '{"name": "John", "age": 30}';
const jsonObject = JSON.parse(jsonString);
console.log(jsonObject.name); // Output: John
```

### 3. Handling Errors
- It's essential to handle errors while parsing JSON, as invalid JSON will throw an error.
- You can use a try...catch block to manage errors effectively.

Example:
javascript
```
const jsonString = '{"name": "John", "age": 30'; // Invalid JSON
try {
    const jsonObject = JSON.parse(jsonString);
} catch (error) {
    console.log("Error parsing JSON:", error.message); // Output: Error parsing JSON: Unexpected end of JSON input
}
```

### 4. Accessing Parsed Data
- Once the JSON string is parsed, you can access the data using dot notation or bracket notation.
- This allows you to read or modify values easily.

Example:
javascript
```
const jsonString = '{"name": "Alice", "hobbies": ["reading", "sports"]}';
const jsonObject = JSON.parse(jsonString);
console.log(jsonObject.hobbies[0]); // Output: reading
jsonObject.age = 25; // Adding a new property
```

### 5. Parsing JSON in Other Languages
- Different programming languages have their own libraries or built-in functions for parsing JSON. Here are a few examples:
  - Python: Use json.loads()
  - Java: Use libraries like Jackson or Gson
  - PHP: Use json_decode()

Example in Python:

python
```
import json
json_string = '{"name": "John", "age": 30}'
json_object = json.loads(json_string)
print(json_object['name'])  # Output: John
```

### Summary of Parsing JSON:

Step	Description	Example
What is JSON Parsing?	Converting JSON strings into data structures for easy manipulation	N/A
Parse in JavaScript	Use JSON.parse() to convert a JSON string to an object	const obj = JSON.parse('{"name": "John"}');
Handling Errors	Use try...catch to manage parsing errors	try { JSON.parse(invalidJSON); } catch (error) {}
Accessing Data	Use dot or bracket notation to read or modify parsed data	console.log(obj.name);
Other Languages	Different languages have specific functions for parsing JSON	Python: json.loads(), PHP: json_decode()

### Conclusion:
- Parsing JSON is a crucial skill in programming, allowing for effective data manipulation.
- Using JSON.parse() in JavaScript and similar functions in other languages makes it straightforward to work with JSON data.
- Proper error handling ensures robust applications when dealing with JSON.

Persisting JSON :

### Persisting JSON (for 5 Marks - TYBSC IT Mumbai University)

Persisting JSON refers to the process of storing JSON data in a way that allows it to be retrieved and used later. This is important for applications that need to save data between sessions. Below is a detailed explanation of persisting JSON in simple points.

### 1. **What is JSON Persistence?**
- JSON persistence means saving JSON data to a storage medium, such as a file or a database, so that the data can be retrieved later.
- This is often used in web applications to store user settings, application state, or any data that needs to be saved for future use.

### 2. **Common Methods for Persisting JSON**
There are several common methods to persist JSON data:

- **File Storage**: Saving JSON data to a file on the server or local filesystem.
- **Database Storage**: Storing JSON data in a database (SQL or NoSQL).
- **Local Storage**: In web applications, using the browser's local storage to save JSON data on the client side.

### 3. **Persisting JSON to a File (JavaScript Example)**
- In Node.js, you can use the `fs` (file system) module to write JSON data to a file.

  Example:
  ```javascript
  const fs = require('fs');

  const jsonObject = {
    name: "John",
    age: 30,
    hobbies: ["reading", "sports"]
  };

  // Convert JSON object to string
  const jsonString = JSON.stringify(jsonObject);
  ```

```
 // Write JSON string to a file
 fs.writeFile('data.json', jsonString, (err) => {
   if (err) {
     console.log('Error writing to file', err);
   } else {
     console.log('JSON data saved successfully!');
   }
 });
```

### 4. **Persisting JSON in a Database**
- Many databases can store JSON data directly. For example, NoSQL databases like MongoDB are designed for JSON-like documents.

Example (MongoDB):
```javascript
 const MongoClient = require('mongodb').MongoClient;

 const url = 'mongodb://localhost:27017';
 const dbName = 'mydatabase';
 const client = new MongoClient(url);

 const jsonObject = { name: "Alice", age: 25 };

 client.connect(err => {
   const db = client.db(dbName);
   const collection = db.collection('users');

   // Insert JSON object into the collection
   collection.insertOne(jsonObject, (err, result) => {
     if (err) {
       console.log('Error inserting data', err);
     } else {
       console.log('JSON data persisted in database');
     }
     client.close();
   });
 });
```

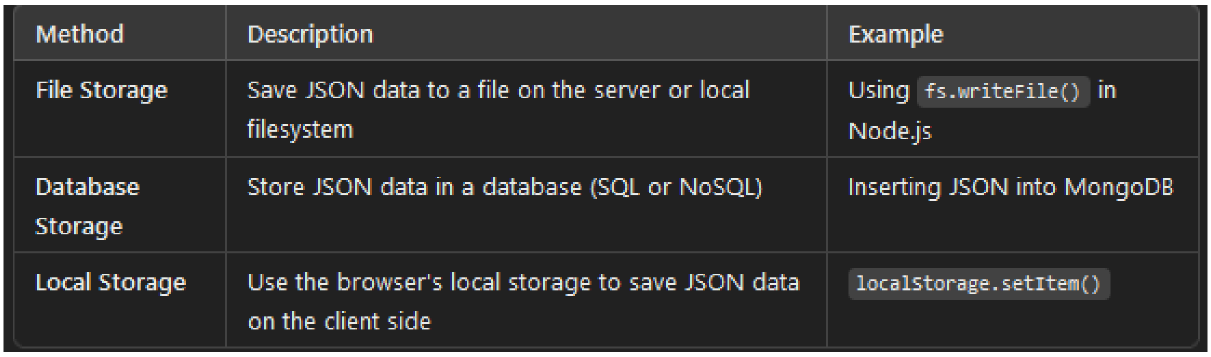
### 5. **Using Local Storage in Web Applications**
- In web applications, JSON data can be persisted in the browser's local storage. This is a simple way to store data without a server.

Example:
```javascript
const jsonObject = {
  name: "John",
  age: 30
};

// Convert JSON object to string and save to local storage
localStorage.setItem('userData', JSON.stringify(jsonObject));

// Retrieve JSON data from local storage
const retrievedData = JSON.parse(localStorage.getItem('userData'));
console.log(retrievedData.name); // Output: John
```

### Summary of Persisting JSON:



Method	Description	Example
File Storage	Save JSON data to a file on the server or local filesystem	Using `fs.writeFile()` in Node.js
Database Storage	Store JSON data in a database (SQL or NoSQL)	Inserting JSON into MongoDB
Local Storage	Use the browser's local storage to save JSON data on the client side	`localStorage.setItem()`

### Conclusion:
- Persisting JSON is essential for storing data that needs to be retained across sessions.
- Whether using file storage, databases, or local storage, there are various methods to effectively save JSON data.

- Understanding how to persist JSON enables developers to create more dynamic and stateful applications.

Data Interchange :

Sure! Here's a simplified breakdown:

### Main Ideas

1. Learning JSON: You've been learning how to work with JSON (a way to structure data) in your applications.

2. JSON as Data Exchange: JSON can be sent over the Internet, allowing your app to communicate with databases that you control, which is more secure than local storage options like cookies.

3. Sharing and Using Data: You can not only store your own JSON data but also access data shared by others, such as public APIs from social media platforms like Twitter, Facebook, and Instagram.

4. Benefits of JSON: Because JSON is easy to use and widely accepted, it's the preferred format for many social APIs.

5. Upcoming Topics: You'll learn how to send, receive, and save JSON data in your application and how to use data from APIs like Twitter.

6. Understanding Communication: Before diving into those topics, it's important to understand how browsers communicate with servers to request and receive resources.

- **Hypertext Transfer Protocol :**

    1. **What is HTTP?**: HTTP stands for Hypertext Transfer Protocol. It's the main way we interact with the Internet.
    2. **How It Works**: HTTP enables communication between a client (like a web browser) and a server. When you use a browser (like Chrome or Firefox), it sends a request to a server for a specific resource (like a webpage or an image).

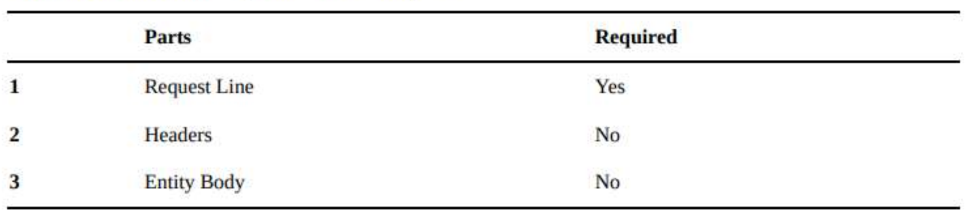
3. **Request and Response**: A request must be made to get any resource, and only a web server can send back a response. This means you need to ask for something before you can receive it.
4. **Initiation**: The process starts when a client (browser or server) initiates the request for a resource.

- **HTTP-Request**

1. **Purpose of a Request**: An HTTP request tells the server exactly what resource you need. This helps the server understand and provide the correct response.
2. **Restaurant** Analogy: Think of making a request like ordering food at a restaurant. When you tell the waiter what you want, you specify how you'd like it cooked or served. In this analogy:
3. **HTTP Protocol**: The waiter
4. **Your Order**: The HTTP request
5. **Food Served:** The HTTP response

6. **Components of a Request:** An HTTP request has three main parts that help specify what resource you're asking for. These parts help the server know exactly what you need.

Table 8-1. Structure of the HTTP Request



	Parts	Required
1	Request Line	Yes
2	Headers	No
3	Entity Body	No

- **Request Line** :

1. **What is the Request Line?**: The request line is the most important part of an HTTP request. It tells the server what you want.
2. **Three Parts of the Request Line**:
   - **Method**: This indicates what action you want to perform on the resource. Common methods include:
     - **GET**: Used to retrieve data (like loading a webpage).
     - **POST**: Used to send data to the server (like submitting a form).
   - **Request-URI**: This is the specific address of the resource you want (like a URL for a CSS file or an image).
   - **HTTP-Version**: This specifies the version of HTTP being used (usually HTTP/1.1).
3. **Safety of Methods**:
   - **GET** is considered "safe" because it doesn't change anything on the server.

- **POST** is "unsafe" because it can change data on the server.

4. **Example of a Request Line**:
   - GET http://example.com/image.jpg HTTP/1.1
   - POST http://example.com/form.php HTTP/1.1

- **Headers :**

1. **General Headers**:
   - These provide basic information about the response. They can include:
     - **Cache-Control**: Instructions on how to cache the response.
     - **Connection**: Whether the connection should stay open.
     - **Date**: The date and time of the response.
     - Others include **Pragma**, **Trailer**, **Transfer-Encoding**, **Upgrade**, **Via**, and **Warning**.
2. **Response Headers**:
   - These give details about the server and the requested resource. They inform the client about:
     - Accepted HTTP methods (like GET or POST).
     - Whether authorization is needed.
     - Whether the request should be retried later.
   - Examples include **Accept-Ranges**, **Age**, **ETag**, **Location**, **Server**, and **WWW-Authenticate**.
3. **Entity Headers**:
   - These provide information about the actual data being sent. They describe the content type and how it should be read.
   - The **Content-Type** header is key, indicating the type of data (like text, image, etc.).
   - Other entity headers include **Content-Length**, **Content-Encoding**, **Last-Modified**, and **Expires**.

- **Entity Body :**

 What is the Entity Body?: The entity body is the actual data sent back by the server. While headers give information about the response, the entity body contains the content itself.

 Response Example: When you access a URL (like http://json.sandboxed.guru/chapter8/headers.php), the server responds with:

- A **status line** showing the HTTP version and status code (e.g., 200, which means "OK").

- Various headers that provide details about the response.

   MIME Type: The last header typically indicates the MIME type, which tells the browser how to handle the data. For example, if the data is HTML, the header will say Content-Type: text/html.

   Viewing the Data: The actual content returned (like HTML markup) can be seen in the response tab of your browser.

   Developer Insight: This technical information is mainly known by server-side developers, as they configure how requests and responses work. Most front-end developers don't need to worry about these details since the browser handles them.

   Next Steps: Soon, you'll learn how to create your own HTTP requests using JavaScript to send and receive JSON data.

- **XMLHttpRequest Interface :**

A Deeper Dive into XHR Event Handlers

Understanding XHR Event Handlers

XHR event handlers are crucial for managing the lifecycle of an asynchronous HTTP request. They allow you to respond to different stages of the request, from its initiation to its completion.

Key Event Handlers:

1. **onloadstart:**
   - Fired when the request starts.
   - Useful for initializing progress indicators or starting timers.
2. **onprogress:**
   - Fired periodically during the request, especially for large responses.
   - Can be used to update progress bars or provide feedback to the user.
3. **onload:**
   - Fired when the request completes successfully.
   - Access the response data using xhr.responseText or xhr.responseXML.
4. **onerror:**

- Fired when an error occurs during the request (e.g., network issues, server errors).
- Handle the error gracefully, provide feedback to the user, or retry the request.

5. **ontimeout:**

- Fired if the request takes longer than the specified timeout.
- Display a timeout message or allow the user to retry.

6. **onabort:**

- Fired if the request is manually aborted (e.g., using xhr.abort()).
- Clean up any resources or notify the user.

7. **onreadystatechange:**

- Fired whenever the readyState property of the XHR object changes.
- readyState values:
  - 0: UNSENT (request not initialized)
  - 1: OPENED (request has been opened)
  - 2: HEADERS_RECEIVED (HTTP headers received)
  - 3: LOADING (response body is being received)
  - 4: DONE (request is complete)

Example:

JavaScript

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://api.example.com/data');

xhr.onloadstart = function() {
  console.log('Request started');
};

xhr.onprogress = function(event) {
  var progress = (event.loaded / event.total) * 100;
  console.log('Progress:', progress + '%');
};

xhr.onload = function() {
  if (xhr.status === 200) {
    var data = JSON.parse(xhr.responseText);
    console.log('Data received:', data);
  } else {
    console.error('Request failed: ' + xhr.status);
  }
};

xhr.onerror = function() {
  console.error('Request error');
};

xhr.ontimeout = function() {
  console.error('Request timed out');
};

xhr.send();
```

Key Considerations:

- **Error handling:** Implement robust error handling to provide a better user experience.
- **Timeout:** Set a reasonable timeout to prevent requests from hanging indefinitely.
- **Progress updates:** Use the onprogress event to provide feedback to the user for long-running requests.

- **Asynchronous nature:** Remember that XHR requests are asynchronous, so the response may not be immediate.

By effectively using XHR event handlers, you can create more responsive and user-friendly web applications.

JSON PHP :

### What is JSON?

JSON (JavaScript Object Notation) is a lightweight data format that is easy for humans to read and write, and easy for machines to parse and generate. It's commonly used to send data between a server and a web application.

### Using JSON in PHP

PHP can easily create and manipulate JSON data. We can encode data in JSON format to send it to a client or decode JSON data received from a client.

### Step-by-Step Example

Let’s create a simple example where PHP generates JSON data and sends it to the client.

#### Step 1: Create a PHP File

Create a file named data.php. This file will generate JSON data.

php

```
<?php
// Step 1: Create an associative array
$data = array(
   "name" => "John Doe",
   "age" => 30,
   "city" => "New York"
);

// Step 2: Convert the PHP array to JSON format
$json_data = json_encode($data);

// Step 3: Set the content type to application/json
header('Content-Type: application/json');

// Step 4: Output the JSON data
echo $json_data;
?>
```

### Explanation of the PHP Code

1. Create an Associative Array:
   - We create a PHP array with some data about a person.

2. Convert to JSON:
   - We use json_encode() to convert the PHP array into a JSON string.

3. Set Content-Type:

  - We set the HTTP header to application/json so the client knows we are sending JSON data.

4. Output JSON:
   - Finally, we use echo to send the JSON string to the client.

#### Step 2: Create an HTML File to Fetch JSON

Next, create an HTML file named index.html to fetch and display the JSON data.

```
html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>JSON with PHP Example</title>
</head>
<body>
    <h1>Data from PHP</h1>
    <div id="result"></div>

    <script>
        // Step 1: Fetch the JSON data from the PHP file
        fetch('data.php')
            .then(response => response.json()) // Step 2: Parse the JSON data
            .then(data => {
                // Step 3: Display the data
                document.getElementById('result').innerText =
JSON.stringify(data);
            })
            .catch(error => console.error('Error:', error)); // Step 4: Handle any
errors
    </script>
</body>
</html>
```

### Explanation of the HTML Code

1. Fetch the JSON Data:
   - We use the fetch function to request the JSON data from data.php.

2. Parse JSON Data:
   - The response is converted from JSON format using response.json().

3. Display the Data:
   - We display the received data in a <div> with the ID result.

4. Error Handling:
   - If there's an error in fetching the data, it will be logged to the console.

### Putting It All Together

1. Create data.php:
   - This file generates and serves JSON data.

2. Create index.html:
   - This file fetches the JSON data from data.php and displays it.

### Example Folder Structure

```
/your-project-folder
    ├── data.php
    └── index.html
```

### How to Test

1. Make sure you have a local server (like XAMPP or WAMP) running.
2. Place both files in the server's root directory (like htdocs for XAMPP).
3. Open index.html in your web browser.

You should see the JSON data displayed on the page.

### Summary

- PHP Array: Create a PHP array.
- Convert to JSON: Use json_encode() to convert it.
- Set Header: Set the content type to application/json.
- Fetch in HTML: Use fetch to get the JSON data and display it.

JSON HTML :

Sure! Let’s explore how to use JSON with HTML step by step, using simple words and a straightforward example.

### What is JSON?

JSON (JavaScript Object Notation) is a format for structuring data that is easy for both humans and machines to read and write. It's often used to send data between a server and a web application.

### Using JSON in HTML

You can use JSON directly in your HTML file with JavaScript to display or manipulate data on a webpage.

### Step-by-Step Example

Let's create a simple example where we have some JSON data embedded in an HTML file, and we’ll use JavaScript to display it on the page.

#### Step 1: Create an HTML File

Create a file named index.html and add the following code:

html
```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <title>JSON and HTML Example</title>
</head>
<body>
   <h1>My Favorite Fruits</h1>
   <div id="result"></div>

   <script>
```

```
    // Step 1: Define JSON data as a JavaScript object
    const jsonData = [
      { "name": "Apple", "color": "Red" },
      { "name": "Banana", "color": "Yellow" },
      { "name": "Grapes", "color": "Purple" }
    ];

    // Step 2: Get the result div
    const resultDiv = document.getElementById('result');

    // Step 3: Loop through the JSON data and create HTML content
    jsonData.forEach(fruit => {
      const fruitElement = document.createElement('p');
      fruitElement.innerText = Name: ${fruit.name}, Color: ${fruit.color};
      resultDiv.appendChild(fruitElement);
    });
  </script>
</body>
</html>
```

### Explanation of the Code

1. HTML Structure:
   - We set up a basic HTML document with a heading and a div to display our results.

2. Define JSON Data:
   - Inside a <script> tag, we define a JavaScript array called jsonData that contains some fruit objects, each with a name and color.

3. Select the Result Div:
   - We select the <div> with the ID result where we will display our fruit data.

4. Loop Through the JSON Data:
   - We use the forEach method to loop through each fruit in the jsonData array.
   - For each fruit, we create a new paragraph element (<p>), set its text to show the fruit's name and color, and append it to the resultDiv.

### Step 2: Open the HTML File

1. Save the index.html file.
2. Open it in your web browser.

### Result

You should see the following displayed on the webpage:

```
My Favorite Fruits
Name: Apple, Color: Red
Name: Banana, Color: Yellow
Name: Grapes, Color: Purple
```

### Summary

- JSON Data: We defined some JSON-like data as a JavaScript array.
- Display Data: We used JavaScript to create HTML elements for each item in the JSON data and displayed them on the page.

This simple example shows how you can use JSON with HTML to dynamically display data. You can expand on this by fetching JSON data from a server or using more complex data structures.

JSONP :

Sure! Let's break down JSONP in simple words and go through a straightforward example step by step.

### What is JSONP?

JSONP (JSON with Padding) is a method used to request data from a server in a different domain than the one the web page is on. This is necessary because of the same-origin policy, which restricts how a document or script can interact with resources from another origin. JSONP allows you to bypass this restriction by using <script> tags.

### How JSONP Works

1. Callback Function: The server returns data wrapped in a JavaScript function call (the callback).
2. Script Tag: The client creates a <script> tag to request the data from the server.
3. Execution: When the script loads, it executes the callback function, receiving the data.

### Step-by-Step Example

Let's create a simple example where a server responds with JSONP data.

#### Step 1: Create a JSONP Server

First, create a PHP file named jsonp.php that simulates a server response.

php
```php
<?php
// Step 1: Get the callback function name from the query parameter
$callback = isset($_GET['callback']) ? $_GET['callback'] : 'callback';

// Step 2: Create a sample data array
$data = array(
   "name" => "John Doe",
   "age" => 30,
   "city" => "New York"
);

// Step 3: Encode the data as JSON
$json_data = json_encode($data);

// Step 4: Wrap the JSON data in the callback function
$response = $callback . '(' . $json_data . ');';

// Step 5: Output the response
header('Content-Type: application/javascript');
echo $response;
?>
```

### Explanation of the JSONP Server Code

1. Get Callback:
   - We retrieve the callback parameter from the URL query string. This is the name of the JavaScript function that will handle the response.

2. Create Sample Data:
   - We create an associative array with some sample data.

3. Encode as JSON:
   - We use json_encode() to convert the PHP array into a JSON string.

4. Wrap in Callback:
   - We create a response that calls the callback function with the JSON data.

5. Set Header:
   - We set the content type to application/javascript since we're sending JavaScript code.

#### Step 2: Create an HTML File to Fetch JSONP

Next, create an HTML file named index.html to make the JSONP request.

html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>JSONP Example</title>
</head>
<body>
    <h1>JSONP Example</h1>
    <div id="result"></div>

    <script>
        // Step 1: Define the callback function
        function handleResponse(data) {
            // Step 2: Display the received data
            document.getElementById('result').innerText =
                Name: ${data.name}, Age: ${data.age}, City: ${data.city};
        }
        // Step 3: Create a script tag to request JSONP data
        const script = document.createElement('script');
        script.src = 'jsonp.php?callback=handleResponse'; // Pass the
callback function name
        document.body.appendChild(script); // Add the script tag to the body
    </script>
</body>
</html>
```

### Explanation of the HTML Code

1. Define Callback Function:
   - We create a JavaScript function called handleResponse that will receive the data.

2. Display Data:
   - This function updates the <div> with the ID result to show the received data.

3. Request JSONP:
   - We create a <script> tag and set its src to jsonp.php, including the callback function name as a query parameter.

4. Append Script:
   - Finally, we append the <script> tag to the document body. When the script loads, it executes the callback function with the data.

### How to Test

1. Make sure you have a local server (like XAMPP or WAMP) running.
2. Place both files (jsonp.php and index.html) in the server's root directory (like htdocs for XAMPP).
3. Open index.html in your web browser.

### Result

You should see the following displayed on the page:

```
Name: John Doe, Age: 30, City: New York
```

### Summary

- Callback Function: The server wraps the JSON data in a function call.
- Script Tag: The client creates a script tag to request the data.
- Execution: When the script loads, it calls the function with the data.

That's it! You now have a simple example of using JSONP to retrieve data from a server.