JSON and Asynchronous JS

Introduction to JSON

JSON stands for JavaScript Object Notation and is a way of representing information in a format that is easy for computers to understand and share. Think of it as a digital list of information, like a grocery list, person biodata, etc., where each item is a key-value pair.
For example, consider the following JSON data that represents a person:

```json
{
  "name": "Barbarik",
  "age": 22,
  "address": {
    "street": "Ramnagar Road",
    "city": "Varanasi",
    "state": "Uttar Pradesh"
  },
  "hobbies": [
    "reading",
    "coding",
    "traveling"
  ]
}
```

Here, the key-value pairs represent information about a person: their name, age, address, and hobbies. The address is another set of key-value pairs, and hobbies is a list of strings.

JSON is a great way to store and share data between different computer systems. It's easy to read and write, and is used in many websites and apps to transfer information between the server and the client, or between different programs.

In short, JSON is a simple and flexible way to represent information in a format that is easy for computers to understand and share.

JSON Data Types

In JSON, there are several data types that can be used to represent different kinds of information. Here are the most common data types in JSON and simple examples of each:

- String: A sequence of characters, represented by quotes. For example: "hello"
- Number: A numerical value, either an integer or a decimal. For example: 42 or 3.14
- Object: A collection of key-value pairs, represented by curly braces {}. For example:

```
{
  "name": "John Doe",
  "age": 32
}
```

- Array: An ordered list of values, represented by square brackets []. For example:

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

- Boolean: A value that is either true or false. For example: `true` or `false`

- Null: A special value that represents a missing or unknown value. It is represented by the keyword null. For example: `null`

These data types can be combined in different ways to represent more complex information in JSON. For example, an array of objects can be used to represent a list of people and their information:

```
[
    {    "name": "John Doe",    "age": 32  },
    {    "name": "Jane Doe",    "age": 30  }
]
```

In short, these data types provide a way to represent different kinds of information in JSON, allowing us to store and share complex data structures between computer systems.

JSON Parsing and Converting JSON to String

JSON parsing is the process of taking a string of JSON data and converting it into a more usable data structure, such as an object or array. This is often done in order to extract information from a JSON string and use it in a program.

For example, consider the following JSON string that represents a person:

```
"{
  "name": "Barbarik",
  "age": 22,
  "address": {
    "street": "Ramnagar Road",
    "city": "Varanasi",
    "state": "Uttar Pradesh"
  },
  "hobbies": [
    "reading",
    "coding",
    "traveling"
  ]
}"
```

To parse this string, we would use a JSON parsing function, such as `JSON.parse()` in JavaScript. This function takes the JSON string as an argument and returns a JavaScript object that we can work with:

```
const jsonString = "{
  "name": "Barbarik",
  "age": 22,
  "address": {
    "street": "Ramnagar Road",
    "city": "Varanasi",
    "state": "Uttar Pradesh"
  },
  "hobbies": [
    "reading",
    "coding",
    "traveling"
  ]
}"
 const person = JSON.parse(jsonString);
 console.log(person.name); // "Barbarik"
```

Conversely, converting a JavaScript object to a JSON string is called stringifying. This is often done in order to send the data over the network or to store it in a file. To stringify a JavaScript object, we can use the `JSON.stringify()` function:

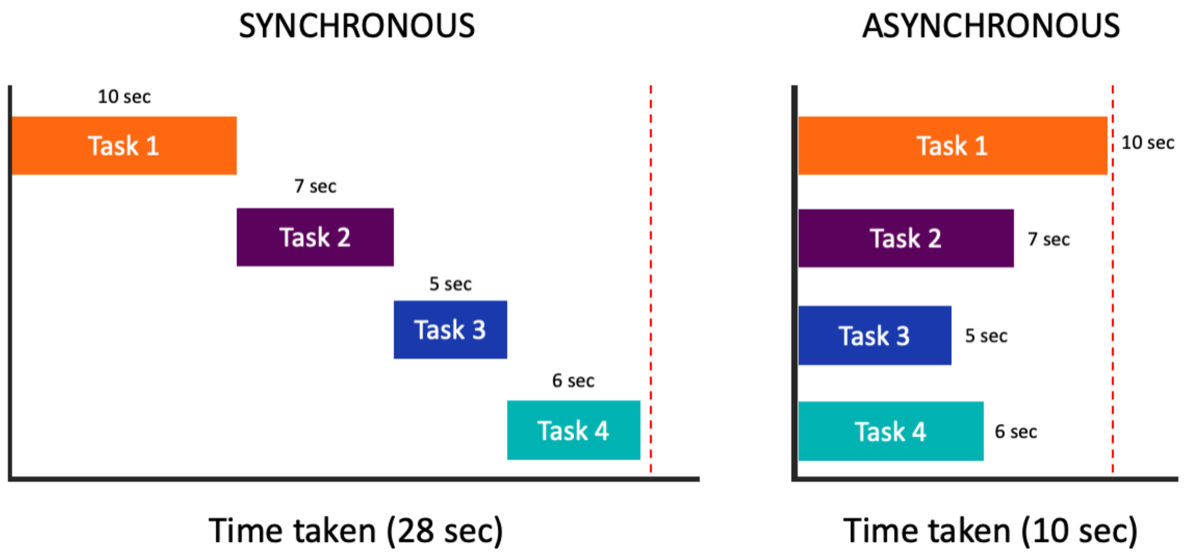
```
const person = {
  "name": "Barbarik",
  "age": 22,
  "address": {
    "street": "Ramnagar Road",
    "city": "Varanasi",
    "state": "Uttar Pradesh"
  },
  "hobbies": [
    "reading",
    "coding",
    "traveling"
  ]
}
```

```
const jsonString = JSON.stringify(person);
console.log(jsonString);
```

In short, parsing and stringifying JSON allows us to easily convert between JSON data and more usable data structures, making it easier to work with JSON data in our programs.

Asynchronous JS

Asynchronous programming is a way of writing code that allows multiple tasks to run at the same time, rather than waiting for one task to finish before starting another. It can make your code run faster and more efficiently.



Think of it like having multiple people work on different tasks simultaneously. For example, you could have one person bake a cake and another person make tea at the same time, instead of having one person do one task first and then the other.

Here's a simple example in JavaScript:
Note that the `async` keyword is used to define asynchronous functions.

```
async function makeCakeAndTea() {
  bakeCake();
  makeTea();
  console.log("Cake is baking and tea is brewing!");
}


async function bakeCake() {
  // baking cake code
}

async function makeTea() {
  // making tea code
}
```

In this example, the `makeCakeAndTea` function starts both `bakeCake` and `makeTea` asynchronously, allowing them to run at the same time. The `console.log` statement is executed right away, letting us know that both tasks are running simultaneously.

setTimeout() function

`setTimeout()` is a JavaScript function that allows you to run a piece of code after a specified amount of time.
Think of it like a timer: you can set a time interval, and after that time interval is up, the code inside `setTimeout()` will run.

Here's a simple example:

```
console.log("Starting timer...");
setTimeout(function() {
  console.log("Time's up!");
}, 5000); // 5000 milliseconds = 5 seconds
```

In the function the first argument is the function to be run, and the second argument is the time in milliseconds after which the function is to be run
In this example, the console.log statement logs "Starting timer..." immediately. Then, the `setTimeout()` function starts a timer for 5 seconds (5000 milliseconds). After 5 seconds, the code inside `setTimeout()` will run and log "Time's up!".

setInterval() Function

The setInterval() function in JavaScript is similar to the setTimeout() function, but it runs repeatedly at specified intervals.
Think of it like an alarm clock that goes off at a certain time every day.

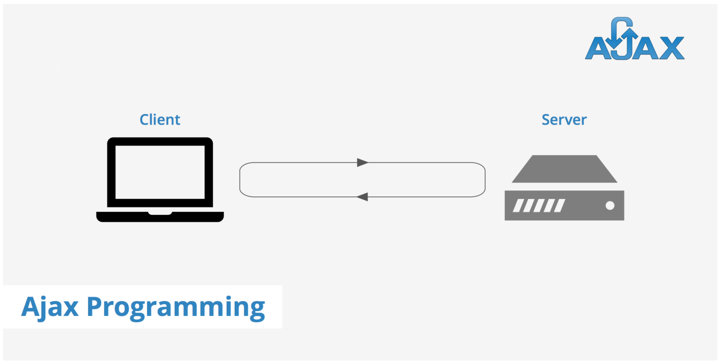
Here's a simple example:

```
let count = 0;
console.log("Starting interval...");
setInterval(function() {
  count++;
  console.log(`Interval has run ${count} times.`);
}, 5000); // 5000 milliseconds = 5 seconds
```

In this example, the console.log statement logs "`Starting interval...`" immediately. Then, the `setInterval()` function starts running the code inside it every 5 seconds (`5000` milliseconds). The count variable keeps track of how many times the code has run, and it is logged every time the interval runs. This process repeats until the program is stopped.

AJAX

AJAX (Asynchronous JavaScript and XML) is a technique in JavaScript that allows you to make asynchronous requests to a server to retrieve data or send data without having to reload the entire page.



AJAX allows web pages to be updated asynchronously by exchanging data with the web server in the background. This essentially ensures that only a part of the web page is loaded without reloading the entire page.
AJAX uses a combination of `XMLHttpRequest` object along with JavaScript and DOM.

Get, Post, Put, Patch and Delete methods

When you use a web application, you might want to fetch some data or send some data to a server. There are different ways to do this, and each one is represented by a different type of request method:

- GET method: Imagine you want to fetch a book from the library. You go to the library, tell the librarian the title of the book you want, and they give it to you. This is like making a GET request to a server to fetch some data. You're asking the server for some data, and it's sending it back to you.
- POST method: Imagine you want to add a new book to the library. You go to the library, give them the book and the information about it, and they add it to their collection. This is like making a POST request to a server to add some data. You're sending data to the server, and it's adding it to its collection.
- PUT method: Imagine you want to update a book in the library. You go to the library, give them the book and the updated information, and they replace the old book with the new

information. This is like making a PUT request to a server to update some data. You're sending data to the server, and it's replacing the old data with the new data.

- PATCH method: Imagine you want to make a small change to a book in the library. You go to the library, tell them the title of the book and what change you want to make, and they make the change. This is like making a PATCH request to a server to update some data. You're sending data to the server, and it's updating only a specific part of the data.
- DELETE method: Imagine you want to remove a book from the library. You go to the library, tell them the title of the book, and they remove it from their collection. This is like making a DELETE request to a server to remove some data. You're asking the server to remove some data, and it's removing it.

Each of these request methods has a specific purpose and is used in a different way. In a web application, you might use GET to fetch some data, POST to add some data, PUT or PATCH to update some data, and DELETE to remove some data.

XMLHttpRequest

The `XMLHttpRequest` object in JavaScript is like a tool that lets you communicate with a server without refreshing the page. Imagine you want to fetch a book from the library, but instead of physically going to the library, you use a phone to video-call the librarian and ask for the book. The librarian brings the book to you over the phone, and you can read it without having to leave your seat. This is like making an `XMLHttpRequest` in JavaScript. *You're making a request to the server for some data, and it's sending the data back to you without refreshing the page*.

This tool is really useful when you want to fetch some data or send some data to a server without refreshing the page. For example, you might use it to fetch data from a server and display it on a page, or to send data to a server and process it without refreshing the page. You can use this tool to make different types of requests, like `GET`, `POST`, `PUT`, `PATCH`, and `DELETE`.

Here's a simple example:

```
var xhr = new XMLHttpRequest();
xhr.open("GET", "https://api.example.com/data", true);
xhr.onreadystatechange = function() {
  if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
    console.log(xhr.responseText);
  }
};
xhr.send();
```

In this code, we first create a new `XMLHttpRequest` object. Then we use its open method to make a call to a server at `https://api.example.com/data` and ask for some data.

Every time the state of the call changes, the `onreadystatechange` function is called. This function checks if the call is done (by checking `xhr.readyState`) and if everything is OK (by checking `xhr.status`). If everything is OK, it logs the response text received from the server.

XMLHttpRequest can also be used to send data to the server, like this:

```
var xhr = new XMLHttpRequest();
xhr.open("POST", "https://api.example.com/data", true);
xhr.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
xhr.onreadystatechange = function() {
  if (xhr.readyState === XMLHttpRequest.DONE && xhr.status === 200) {
    console.log(xhr.responseText);
  }
};
xhr.send(JSON.stringify({ name: "John", age: 30 }));
```

In this example, we first create a new XMLHttpRequest object and use its open method to make a call to a server at `https://api.example.com/data`. Then we use the `setRequestHeader` method to tell the server what kind of data we're sending.

Every time the state of the call changes, the onreadystatechange function is called. This function checks if the call is done (by checking `xhr.readyState`) and if everything is OK (by checking `xhr.status`). If everything is OK, it logs the response text received from the server. The data we're sending to the server is contained in a JSON object :

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

and is sent using the send method.