Complete JavaScript Topics Overview

# 1. Variables and Data Types

JavaScript variables can be declared using var, let, or const. Data types include string, number, boolean, object, array, null, undefined, and symbol.

Example:

let name = "John";  
const age = 30;  
var isStudent = false;

# 2. Functions and Scope

Functions are blocks of code designed to perform a task. Scope refers to the context where variables are accessible (global, local, block).

Example:

function greet(name) {  
 return "Hello, " + name;  
}  
console.log(greet("Alice"));

# 3. Objects and Prototypes

Objects are collections of key-value pairs. Every object in JavaScript has a prototype object from which it inherits methods and properties.

Example:

const person = {  
 name: "John",  
 greet() { console.log("Hello " + this.name); }  
};  
person.greet();

# 4. Classes and Inheritance

Classes are syntactic sugar over JavaScript's prototype-based inheritance.

Example:

class Person {  
 constructor(name) {  
 this.name = name;  
 }  
 greet() {  
 console.log("Hi " + this.name);  
 }  
}  
class Student extends Person {  
 study() {  
 console.log(this.name + " is studying");  
 }  
}  
const student = new Student("Ahsan");  
student.greet();  
student.study();

# 5. Callback and Callback Hell

A callback is a function passed into another function to be executed later. Callback hell happens when callbacks are nested deeply.

Example:

function getData(callback) {  
 setTimeout(() => {  
 console.log("Data received");  
 callback();  
 }, 1000);  
}  
getData(() => {  
 console.log("Callback executed");  
});

# 6. Promises

Promises are used to handle asynchronous operations and avoid callback hell.

Example:

const promise = new Promise((resolve, reject) => {  
 setTimeout(() => resolve("Success"), 1000);  
});  
promise.then(data => console.log(data));

# 7. Async/Await

Async/Await makes asynchronous code look and behave like synchronous code using promises under the hood.

Example:

async function fetchData() {  
 let result = await fetch('https://api.example.com');  
 let data = await result.json();  
 console.log(data);  
}  
fetchData();

# 8. Fetch API

Fetch API is used to make HTTP requests. It returns promises.

Example:

fetch('https://api.example.com')  
 .then(res => res.json())  
 .then(data => console.log(data));

# 9. RESTful APIs (GET, POST, PUT, DELETE)

RESTful APIs are APIs that follow REST architecture using HTTP methods like GET (retrieve), POST (create), PUT (update), DELETE (remove).

Example:

// GET  
fetch('/api/data')  
 .then(res => res.json())  
 .then(data => console.log(data));  
  
// POST  
fetch('/api/data', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ name: "Ahsan" })  
});

# 10. AJAX (Asynchronous JavaScript and XML)

AJAX is a technique to send and receive data asynchronously using JavaScript without refreshing the webpage.

Example:

const xhr = new XMLHttpRequest();  
xhr.open('GET', 'https://api.example.com');  
xhr.onload = function() {  
 console.log(JSON.parse(xhr.responseText));  
};  
xhr.send();

# 11. Event Loop, Microtasks & Macrotasks

JavaScript is single-threaded. The Event Loop handles async operations using a queue of tasks. Microtasks have higher priority than macrotasks.

Example:

console.log("Start");  
setTimeout(() => console.log("Timeout"), 0);  
Promise.resolve().then(() => console.log("Promise"));  
console.log("End");