Complete JavaScript Topics (Basics to Advanced)

# 1. Variables in JavaScript

📌 Description: Variables are containers for storing data values. In JavaScript, we use `var`, `let`, and `const`.

💡 Example:  
let name = "Ahsan";  
const age = 25;  
var isStudent = true;

🧠 Explanation: Use `let` for block-scoped variables, `const` for constants, and `var` for function-scoped variables (mostly avoided in modern JS).

# 2. Hoisting in JavaScript

📌 Description: Hoisting is JavaScript's default behavior of moving declarations to the top.

💡 Example:  
console.log(x); // undefined  
var x = 5;

🧠 Explanation: `var` declarations are hoisted to the top with an initial value of `undefined`. `let` and `const` are hoisted but not initialized.

# 3. Operators in JavaScript

📌 Description: Operators are used to perform operations on variables and values.

💡 Example:  
let sum = 10 + 5;  
let isEqual = (10 === "10");

🧠 Explanation: `+`, `-`, `\*`, `/` are arithmetic operators. `===` checks value and type.

# 4. Loops in JavaScript

📌 Description: Loops are used to run the same block of code repeatedly.

💡 Example:  
for(let i = 0; i < 5; i++){  
 console.log(i);  
}

🧠 Explanation: `for`, `while`, and `do...while` are common loops in JS.

# 5. Escape Characters

📌 Description: Escape characters are used to insert characters that are illegal in a string.

💡 Example:  
let quote = "He said, \"Hello\"";

🧠 Explanation: `\` is used to escape characters in strings.

# 6. Arrays and Their Methods

📌 Description: Arrays are used to store multiple values in a single variable.

💡 Example:  
let fruits = ["apple", "banana", "cherry"];  
fruits.push("orange");

🧠 Explanation: `push()`, `pop()`, `shift()`, `unshift()`, `map()`, `filter()`, `reduce()` are common methods.

# 7. Functions and Methods

📌 Description: Functions are blocks of code designed to perform tasks.

💡 Example:  
function greet(name){  
 return "Hello " + name;  
}

🧠 Explanation: Functions can be declared or expressed as variables. Methods are functions inside objects.

# 8. High Order Functions

📌 Description: Functions that take other functions as arguments or return a function.

💡 Example:  
let numbers = [1, 2, 3];  
let squared = numbers.map(num => num \* num);

🧠 Explanation: `map`, `filter`, `reduce` are examples of higher-order functions.

# 9. DOM in JavaScript

📌 Description: DOM represents the HTML structure as JavaScript objects.

💡 Example:  
document.getElementById("demo").innerText = "Hello!";

🧠 Explanation: DOM lets JavaScript interact with HTML elements.

# 10. Dynamic Manipulation with DOM

📌 Description: We can dynamically add, remove or change HTML elements using JavaScript.

💡 Example:  
let para = document.createElement("p");  
para.innerText = "New Paragraph";  
document.body.appendChild(para);

🧠 Explanation: We can use `createElement`, `appendChild`, `removeChild`, etc.

# 11. DOM Properties like getElementById

📌 Description: Methods to select elements in the DOM.

💡 Example:  
document.querySelector(".class");  
document.getElementsByTagName("div");

🧠 Explanation: Used to select elements to manipulate them.

# 12. Events and addEventListener

📌 Description: Events let us execute code when users interact with the web page.

💡 Example:  
button.addEventListener("click", () => {  
 alert("Clicked!");  
});

🧠 Explanation: `addEventListener` attaches event listeners like click, mouseover, etc.

# 13. Classes and Objects in JS

📌 Description: Classes are blueprints for creating objects.

💡 Example:  
class Person {  
 constructor(name) {  
 this.name = name;  
 }  
 greet() {  
 console.log("Hi " + this.name);  
 }  
}

🧠 Explanation: Use `class`, `constructor`, and `this` to define and use objects.

# 14. Inheritance in JS

📌 Description: Inheritance allows one class to inherit properties from another.

💡 Example:  
class Engineer extends Person {  
 work() {  
 console.log("Solving problems");  
 }  
}

🧠 Explanation: `extends` keyword is used to inherit from a base class.

# 15. Tic Tac Toe Game (Example)

📌 Description: A small project to apply JS knowledge: build the board, handle turns, and detect win.

💡 Example:  
// click to place X or O, check winning combinations

🧠 Explanation: Useful for applying DOM, logic, conditions, and functions.

# 16. Callback Functions

📌 Description: Functions passed into another function as arguments.

💡 Example:  
function greet(name, callback){  
 callback(name);  
}

🧠 Explanation: Used for async programming, event handling.

# 17. Synchronous vs Asynchronous

📌 Description: Synchronous: one task at a time. Asynchronous: tasks don’t block each other.

💡 Example:  
setTimeout(() => console.log("Hello"), 1000);  
console.log("Hi");

🧠 Explanation: `setTimeout`, Promises, async/await enable async code.

# 18. Callback Hell

📌 Description: Nested callbacks leading to unreadable code.

💡 Example:  
step1(() => {  
 step2(() => {  
 step3(() => {  
 console.log("Done");  
 });  
 });  
});

🧠 Explanation: Can be avoided using Promises and async/await.

# 19. Promises

📌 Description: Handle async operations with cleaner syntax.

💡 Example:  
let promise = new Promise((resolve, reject) => {  
 resolve("Success");  
});

🧠 Explanation: Use `.then()` and `.catch()` for chaining results.

# 20. async and await

📌 Description: Simplifies async code written using promises.

💡 Example:  
async function fetchData(){  
 let res = await fetch(url);  
 let data = await res.json();  
}

🧠 Explanation: `await` waits for the promise to resolve inside `async` functions.

# 21. Fetch APIs

📌 Description: Used to make HTTP requests.

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

🧠 Explanation: Used for GET, POST, PUT, DELETE requests to servers.