🟢 What is JavaScript?

JavaScript (JS) is a programming language used to make websites interactive. It's essential for adding dynamic behavior to web pages like:

✅ Updating content without reloading the page
✅ Animating elements
✅ Handling user input (like forms, buttons)

🧱 Basic Syntax

Let's cover some key elements:

1. **Variables** – Store data values.

```
let name = "John"; // Can be reassigned

const age = 25; // Cannot be reassigned

var city = "New York"; // Old way (avoid using it)
```

2. **Data Types** – JS has different types of data:

```
let str = "Hello"; // String

let num = 42; // Number

let isOnline = true; // Boolean

let fruits = ["apple", "banana"]; // Array

let person = { name: "Alice", age: 30 }; // Object
```

3. **Functions** – Reusable blocks of code:

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


greet("John"); // Output: Hello, John
```

4. **Conditions** – Perform actions based on conditions:

```
let age = 18;

if (age >= 18) {
  console.log("Adult");
} else {
  console.log("Minor");
}
```

5. **Loops** – Repeat actions:

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

🟠 6. Arrays

An **array** stores multiple values in one variable.

```
let colors = ["red", "green", "blue"];
console.log(colors[0]); // Output: red

// Add element
colors.push("yellow");

// Loop through an array
colors.forEach((color) => console.log(color));
```

🟠 7. Objects

An **object** stores data as key-value pairs.

```
let person = {
  name: "Alice",
  age: 30,
  city: "New York"
};

// Access values
console.log(person.name); // Output: Alice

// Add new property
person.country = "USA";
```

🟠 8. Arrow Functions

Shorter syntax for functions.

```
// Regular function
function add(a, b) {
  return a + b;
}

// Arrow function
const add = (a, b) => a + b;

console.log(add(3, 5)); // Output: 8
```

🟠 9. Template Literals

Use backticks (`) to embed expressions directly in strings.

```
let name = "John";

console.log(`Hello, ${name}!`); // Output: Hello, John!
```

🟠 10. Destructuring

Extract values from arrays or objects easily.

```
// Array destructuring

const [first, second] = ["apple", "banana"];

console.log(first); // Output: apple


// Object destructuring

const { name, age } = person;

console.log(name); // Output: Alice
```

🟠 11. Spread and Rest Operators

- **Spread (...)** – Expands arrays/objects into individual elements.
- **Rest (...)** – Collects remaining elements into an array.

```
// Spread

const arr = [1, 2, 3];

const newArr = [...arr, 4, 5];

console.log(newArr); // Output: [1, 2, 3, 4, 5]


// Rest

const sum = (...nums) => nums.reduce((a, b) => a + b, 0);

console.log(sum(1, 2, 3)); // Output: 6
```

let's level up! 😎

🟢 12. Promises

A **Promise** handles asynchronous operations (like fetching data) and returns a result when the operation is done.

- resolve → Success
- reject → Failure

✅ **Example:**

```
const fetchData = () => {
  return new Promise((resolve, reject) => {
    let success = true; // Try changing to false to see rejection
    setTimeout(() => {
      if (success) {
        resolve("Data fetched!");
      } else {
        reject("Failed to fetch data");
      }
    }, 1000);
  });
};

fetchData()
  .then((result) => console.log(result)) // Output: Data fetched!
  .catch((error) => console.log(error)); // Output: Failed to fetch data
```

🟢 13. Async/Await

Cleaner way to handle Promises using async and await.
✅ async makes a function return a Promise.
✅ await makes JS wait until the Promise resolves.

✅ **Example:**

```
const fetchData = async () => {
  try {
    let response = await fetch("https://jsonplaceholder.typicode.com/posts/1");
    let data = await response.json();
    console.log(data);
  } catch (error) {
    console.log("Error:", error);
  }
};

fetchData();
```

🟢 14. Callbacks

A **callback** is a function passed as an argument to another function.

✅ **Example:**

```
function greet(name, callback) {
  console.log("Hello " + name);
  callback();
}

function askQuestion() {
  console.log("How are you?");
}

greet("Alice", askQuestion);
// Output:
// Hello Alice
// How are you?
```

🟢 **15. setTimeout and setInterval**

- setTimeout → Runs a function after a delay.
- setInterval → Runs a function repeatedly at fixed intervals.

✅ **Example:**

```
// After 2 seconds
setTimeout(() => {
  console.log("Executed after 2 seconds");
}, 2000);

// Every 1 second
const interval = setInterval(() => {
  console.log("Repeating every second");
}, 1000);

// Stop after 5 seconds
setTimeout(() => clearInterval(interval), 5000);
```

Let’s get into **DOM manipulation**! 😎

🟢 16. DOM (Document Object Model)

The DOM is a tree-like structure that represents the elements of a webpage.
👉 You can use JavaScript to modify the DOM to change or add content dynamically.

✅ Selecting Elements

You can select elements using:

```
// By ID
const heading = document.getElementById("main-heading");

// By class
const items = document.getElementsByClassName("list-item");

// By tag name
const paragraphs = document.getElementsByTagName("p");

// Modern way (querySelector)
const firstItem = document.querySelector(".list-item");
const allItems = document.querySelectorAll(".list-item");
```

✅ Modifying Content

You can modify text, HTML, and attributes:

```
const heading = document.getElementById("main-heading");

heading.textContent = "New Heading"; // Changes text only
heading.innerHTML = "<em>New Heading</em>"; // Changes HTML content
heading.style.color = "red"; // Changes CSS
heading.setAttribute("id", "new-id"); // Changes attributes
```

✅ Creating and Adding Elements

You can create new elements and add them to the DOM:

```
const newItem = document.createElement("li");
newItem.textContent = "New List Item";
```

```
// Append to existing list
const list = document.querySelector("ul");
list.appendChild(newItem);
```

✅ **Removing Elements**

You can remove elements like this:

```
const item = document.querySelector(".list-item");
item.remove();
```

✅ **Event Listeners**

Attach an event (like a click) to an element:

```
const button = document.querySelector("button");

button.addEventListener("click", () => {
  alert("Button clicked!");
});
```

🌟 **Mini Project – Try This:**

Create a simple HTML file and add this JS to make an interactive to-do list:

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>To-Do List</title>
</head>
<body>
  <h1>To-Do List</h1>
  <input type="text" id="task-input" placeholder="Add a new task" />
  <button id="add-btn">Add Task</button>
  <ul id="task-list"></ul>
```

```
  <script src="script.js"></script>
</body>
</html>
```

JS (script.js):

```
const input = document.getElementById("task-input");
const button = document.getElementById("add-btn");
const list = document.getElementById("task-list");

button.addEventListener("click", () => {
  if (input.value.trim()) {
    const task = document.createElement("li");
    task.textContent = input.value;

    // Add delete button
    const deleteBtn = document.createElement("button");
    deleteBtn.textContent = " ❌ ";
    deleteBtn.addEventListener("click", () => task.remove());

    task.appendChild(deleteBtn);
    list.appendChild(task);

    input.value = ""; // Clear input
  }
});
```
