JavaScript

JavaScript Basics — Detailed Intro & Variables (with Memory Explanation)

1) JavaScript Introduction (Detailed)

What is JavaScript?

- **JavaScript (JS)** is a high-level, interpreted programming language primarily used to make web pages interactive and dynamic.
- It works alongside **HTML** (structure) and **CSS** (style).
- JS is **lightweight, cross-platform, and versatile**.

Why JavaScript?

- Runs directly in web browsers without extra setup.
- Supported by all major browsers (Chrome, Firefox, Safari, Edge).
- Can also run on servers (Node.js), making it a **full-stack language**.

Features of JavaScript

1. **Dynamic typing:** No need to define the type of a variable. Types are determined at runtime.
2. **Event-driven:** Reacts to user actions like clicks, keypresses, form submissions.
3. **Prototype-based OOP:** Objects inherit directly from other objects.
4. **Asynchronous capabilities:** Supports callbacks, promises, async/await for non-blocking operations.
5. **Platform independent:** Write once, run anywhere (where JS engines exist).

Where JavaScript is used?

- **Frontend development:** DOM manipulation, animations, form validation, SPAs (Single Page Apps).
- **Backend development:** With Node.js to build APIs and servers.
- **Mobile apps:** Frameworks like React Native, Ionic.
- **Desktop apps:** Electron (VS Code is built with it).
- **Game development:** 2D/3D games using libraries like Phaser, Babylon.js.
- **Machine learning & AI:** TensorFlow.js, Brain.js.
- **IoT devices:** Controlling hardware with JS.

```
// Example: Change text on a webpage
function changeText() {
  document.getElementById("demo").innerHTML = "Hello JavaScript!";
}
```

---

2) JavaScript Variables (Detailed)

What are Variables?

- Variables are containers used to **store data**.
- Think of them as **named memory locations**.

Types of Variables (Declarations)

1. **var** (Old, avoid in modern code)
   - Function-scoped.
   - Hoisted and initialized as `undefined`.
   - Can be re-declared.
2. **let** (Modern, preferred for reassignable variables)
   - Block-scoped.
   - Hoisted but in the **Temporal Dead Zone (TDZ)** until declared.

- Cannot be re-declared in the same scope.

3. **const** (Use when value never changes)
   - Block-scoped.
   - Hoisted but in TDZ.
   - Must be initialized at declaration.
   - Cannot be reassigned.

```
var x = 10; // function-scoped
let y = 20; // block-scoped
const z = 30; // block-scoped, fixed value
```

Scope of Variables

- **Global Scope:** Accessible anywhere in the program.
- **Function Scope:** Created with `var`, accessible inside the function.
- **Block Scope:** Created with `let` or `const`, only inside `{}` block.

```
if (true) {
  var a = 1;   // available outside block
  let b = 2;   // only inside block
  const c = 3; // only inside block
}
console.log(a); // 1
console.log(b); // Error
console.log(c); // Error
```

Hoisting

- All variable declarations are moved to the top of their scope by the JS engine.
- `var` → hoisted & initialized as `undefined`.
- `let` / `const` → hoisted but stay in **TDZ** until the actual declaration line.

```
console.log(a); // undefined
var a = 5;

console.log(b); // ReferenceError
let b = 10;
```

Memory Allocation of Variables

- JavaScript manages memory in two main areas:

1. **Stack Memory (Primitive Values):**
   - Stores simple, fixed-size values like numbers, strings, booleans, `null`, `undefined`, `symbol`, `bigint`.
   - Stored directly in memory.

```
let num = 100; // stored directly in stack
let str = "hello"; // stored directly in stack
```

2. **Heap Memory (Reference Types):**
   - Stores complex or dynamic data like objects, arrays, functions.
   - The variable stores only a **reference (pointer)** in stack, actual value is in heap.

```
let arr = [1, 2, 3];
let obj = {name: "Swati"};
// arr and obj references are in stack, but data is stored in heap.
```

- **Copy behavior:**
  - Primitives are copied **by value**.
  - Objects/arrays/functions are copied **by reference**.

```
let x = 10;
let y = x;
```

```
y = 20;
console.log(x); // 10 (independent copy)

let obj1 = {name: "JS"};
let obj2 = obj1;
obj2.name = "JavaScript";
console.log(obj1.name); // "JavaScript" (same reference)
```

Naming Rules

- Can include letters, numbers, `_`, `$`.
- Must not start with a number.
- Reserved keywords (e.g., `let`, `class`) can't be used.
- Case-sensitive (`Name` ≠ `name`).

Reassignment Examples

```
let age = 25;
age = 26; //  allowed

const pi = 3.14;
// pi = 3.14159;  Error
```

Dynamic Typing

- Variables can hold values of any type, and the type can change.

```
let data = 42;    // number
data = "hello";   // now string
```

Best Practices

- Use **const** by default.
- Use **let** when reassignment is needed.

- Avoid **var** in modern projects.
- Use meaningful variable names (e.g., `userAge` not `x`).
