📄 Session Notes – 16-09-2025

1. Classic JavaScript Variables (var) – Creation & Behavior

- In older versions of JavaScript, variables were declared using `var`.
- **Characteristics of `var`:**
  - Function-scoped (accessible inside the function where declared).
  - Can be **re-declared** within the same scope.
  - Can be **updated/re-assigned**.
  - Hoisted to the top of their scope but initialized with `undefined`.

👉 Example:

```js
console.log(a); // undefined (due to hoisting)
var a = 10;
console.log(a); // 10
```

⚠️ Problem: Can cause bugs in larger codebases because of its **global leakage** and **redeclaration allowance**.

2. Debugging Code in Browser & Understanding Scopes

- **How to Debug:**
  1. Open **Developer Tools** → Console / Sources tab.
  2. Add `debugger;` keyword in your code to pause execution.
  3. Step through line by line to analyze variable values.
- **Scopes in Browser DevTools:**
  - **Global Scope** → Variables accessible everywhere (declared with `var` at global level).
  - **Script Scope** → Variables restricted to the current script file/module.
  - **Local/Block Scope** → Variables declared with `let` or `const` inside `{ }` are not accessible outside.

👉 Example with Debugger:

```js
function testScope() {
  var x = 10;
  let y = 20;
  const z = 30;
  debugger; // pause here
  console.log(x, y, z);
}
testScope();
```

When paused → check "Scopes" section in DevTools to see where variables live.

---

3. Modern JavaScript Variables (let & const) – Creation & Behavior

- Introduced in **ES6 (2015)** to overcome issues with `var`.
- **let:**
  - Block-scoped (exists only within `{ }`).
  - Can be updated/reassigned but not redeclared in the same scope.
- **const:**
  - Block-scoped.
  - Must be initialized during declaration.
  - Cannot be reassigned (but objects/arrays can have their internal values modified).

👉 Example:

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

const country = "India";
country = "USA"; // ❌ Error: Assignment to constant variable
```

---

4. let vs const – Key Difference Intro

- **let** → use when you expect the value to change.
- **const** → use when the value should remain constant throughout.
- Both are block-scoped and safer than `var`.

👉 Example:

```js
let score = 50;
score = 80; // works fine

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

---

5. Errors Observed in Variable Usage

During debugging, the following common errors were noted:

- **ReferenceError:** Accessing a variable before declaration (in `let` or `const`).
- **SyntaxError:** Redeclaring a variable with the same name in the same scope using `let` or `const`.
- **TypeError:** Trying to reassign a `const` variable.

👉 Example Errors:

```js
console.log(x); // ReferenceError
let x = 5;

const y = 10;
y = 20; // TypeError
```

✅ Conclusion

- Use **`let`** for values that can change.
- Use **`const`** for constants and objects/arrays that should not be reassigned.
- Avoid **`var`** in modern development (except in legacy code).
- Always use browser DevTools for debugging and understanding **scope behavior**.

📌 Assignment

- Explore **browser debugging** in detail:
  - Add `debugger;` in different functions and loops.
  - Watch variables change step by step.
- Explore **different scopes**:

    - Script, Global, Block scope.
- Try scenarios like redeclaration, reassignment, and hoisting for `var`, `let`, and `const`.
