✨ LECTURE 17 – HOW JAVASCRIPT CODE WORKS & HOISTING

🔷 JAVASCRIPT NATURE (BASIC DNA)

👉 Synchronous

- JavaScript **ek time pe ek hi kaam** karta hai
- Jab tak ek line complete nahi hoti, next line execute nahi hoti

🧠 Example (Real Life):

Pehle chai banegi ☕ → tabhi biscuit khaya jaayega 🍪

👉 Single-Threaded

- JavaScript ke paas **sirf ek main thread** hota hai
- Code **line by line** execute hota hai

📌 Matlab:

JS multitasking nahi karta, balki **queue + execution rules** follow karta hai

---

🔷 EXECUTION CONTEXT (EC)

👉 What is Execution Context?

Jab bhi JS code run hota hai, JS engine sabse pehle **Execution Context** banata hai.

🟫 Execution Context =
➡ Memory + Code Execution ka **container**

🔷 EXECUTION CONTEXT PHASES

🟡 1️⃣ Memory Creation Phase (Hoisting Phase)

Is phase me:

- Variables & functions ke liye **memory allocate** hoti hai
- Code execute **nahi** hota

🔶 **Variable Rules**

- **var** → `undefined` assign hota hai
- **let / const** → memory milti hai, value nahi (TDZ)
- **function declaration** → poora function memory me store

🟢 2️⃣ **Code Execution Phase**

- Ab code **line by line** execute hota hai
- Values assign hoti hain
- Functions call hote hain

🔷 HOISTING (DEEP CONCEPT)

👉 What is Hoisting?

Hoisting ka matlab:

> JS **pehle memory allocate karta hai**, phir code execute karta hai

⚠️ **Code upar move nahi hota**, sirf **declaration memory me chali jaati hai**

🔷 HOISTING CASES

✅ Case 1️⃣: var

```
console.log(x);

var x = 10;
```

📤 Output:

```
undefined
```

🧠 Reason:

- Memory phase me `var x = undefined`
- Value baad me assign hui

❌ Case 2️⃣: let / const

```
console.log(y);

let y = 20;
```

📤 Output:

```
ReferenceError: Cannot access 'y' before initialization
```

🧠 Reason:

- Hoist hota hai
- Par **TDZ me hota hai**

🔷 TEMPORAL DEAD ZONE (TDZ)

👉 Definition

TDZ wo time hota hai:

> Jab variable **exist karta hai**,
> par **use access karna illegal hota hai**

📌 Sirf **let & const** ke saath hota hai

❌ TDZ Example

```
console.log(a);

let a = 10;
```

📤 Output:

```
ReferenceError
```

💡 **Best Practice**

Hamesha variables ko scope ke top pe declare karo

🔷 FUNCTION & HOISTING

✅ Function Declaration

```
greet();


function greet() {
  console.log("Hello from greet");
}
```

📤 Output:

```
Hello from greet
```

🧠 Reason:

- Poora function memory phase me store ho jaata hai

❌ Function Expression (let)

```
meet();
let meet = function() {
  console.log("meet");
};
```

📤 Output:

```
ReferenceError
```

🧠 Reason:

- let → TDZ issue

❌ Function Expression (var)

```
meet();

var meet = function() {

  console.log("meet");

};
```

📤 Output:

```
TypeError: meet is not a function
```

🧠 Reason:

- var meet = undefined
- undefined() call ho raha hai

🔷 REAL EXECUTION CONTEXT EXAMPLE

```
let a = 10;

let b = 20;


function add(num1, num2) {

  let result = num1 + num2;

  return result;

}


var ans = add(a, b);

console.log(ans);
```

📤 Output:

```
30
```

🧠 Memory Phase

- a → uninitialized (TDZ)
- b → uninitialized (TDZ)
- add → function stored
- ans → undefined

🧠 Execution Phase

- a = 10
- b = 20
- add(a,b) → 30
- ans = 30

🔷 QUICK SCENARIOS (EXAM / INTERVIEW)

✅ Scenario 1

```
greet();

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

✓ Works

❌ Scenario 2

```
meet();

let meet = () => {};
```

❌ TDZ Error

❌ Scenario 3

```
meet();

var meet = () => {};
```

❌ TypeError

🔁 FINAL SUMMARY – EK NAZAR ME

- ✅ JS = **Synchronous + Single-Threaded**
- ✅ Execution Context = Memory Phase + Code Phase
- ✅ Hoisting = memory allocation before execution
- ✅ var → undefined
- ✅ let / const → TDZ
- ✅ Function Declaration → fully hoisted
- ❌ Function Expression → depends on var / let

MASTER INTERVIEW LINE

Hoisting doesn't move code upward — JavaScript just prepares memory first.