Global Execution Context, Hoisting, and Temporal Dead Zone (TDZ)

🌀 1. Global Execution Context (GEC)

When JavaScript runs a program, it first creates a **Global Execution Context**, which is divided into two phases:

📌 Phases of GEC:

✅ 1. Memory Phase (Creation Phase)

- JavaScript scans the whole code.
- All `var` **variables** are hoisted and stored with initial value `undefined`.
- All **function declarations** are hoisted **with their full body**.
- `let` **and** `const` are also hoisted but stored in **Temporal Dead Zone**.

✅ 2. Code Phase (Execution Phase)

- Code runs **line-by-line**.
- Variables get assigned actual values.
- Functions are invoked.

🖼️ Visual Representation:

🪁 2. Hoisting

Hoisting means moving declarations to the top of their scope during the memory phase.

➕ What Gets Hoisted?

Type	Hoisted?	Initialized?
`var`	✅ Yes	`undefined`
`let` / `const`	✅ Yes	❌ No (TDZ until line)
Function	✅ Yes	✅ With full body
Function Expr	✅ Var only	❌ Undefined until assigned

🧪 Example:

```
console.log(x);  // undefined
var x = 10;
```

```
console.log(a);  // Error: Cannot access 'a' before initialization
let a = 20;
```

⚠️ 3. Temporal Dead Zone (TDZ)

TDZ is the time between the hoisting of a `let` or `const` variable and its actual declaration line.

> ❌ Accessing the variable in this zone causes an error.

❇️ Example:

```
console.log(y);  // ❌ ReferenceError
let y = 15;
```

🧾 Even though `let` is hoisted, you **cannot access it before the line it's declared** — this is **TDZ** in action.

🔄 4. Function Hoisting

✅ Function Declaration

```
sayHello();  // Works fine

function sayHello() {
  console.log("Hi");
}
```

❌ Function Expression with var

```
sayHi();  // ❌ TypeError: sayHi is not a function

var sayHi = function() {
  console.log("Hi");
};
```

🧪 5. Real Example with Execution Phases

🧾 Code:

```
console.log("Value of x is", x);
var x = 10;
```

🧠 Execution:

- Memory Phase:
  - `x → undefined`
- Code Phase:
  - Line 1: `console.log("Value of x is", x)` → prints "Value of x is undefined"
  - Line 2: `x = 10`

🔍 6. If We Use `let` Instead:

```
console.log("Value of x is", x);
let x = 10;
```

This throws:

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

Because `x` is in **TDZ**.

🔁 7. Summary Table

Feature	`var`	`let`	`const`
Hoisted	✅ Yes	✅ Yes	✅ Yes
Initial Value	`undefined`	❌ Not initialized	❌ Not initialized
TDZ Exists	❌ No	✅ Yes	✅ Yes
Reassignable	✅ Yes	✅ Yes	❌ No
Redeclarable	✅ Yes	❌ No	❌ No

📬 Interview Tip:

Don’t say just “`let` is not hoisted” — it's incorrect.

✅ Say:

"`let` is hoisted, but it stays in Temporal Dead Zone until its declaration is reached."

🎓 This shows your deep understanding and can help you crack tricky interview questions.

✍️ Revision Notes (Copy-Paste Friendly)

```
# JavaScript Execution Context and Hoisting

## Global Execution Context (GEC)
- Created when any JS file runs.
```

- Two phases:
  1. Memory Phase: all variables/functions are hoisted.
  2. Code Phase: code runs line-by-line.

Hoisting
- `var` is hoisted with value `undefined`.
- `let` and `const` are hoisted but not initialized (TDZ).
- Function declarations are hoisted fully.

Temporal Dead Zone (TDZ)
- Applies to `let` and `const`.
- Variable exists but cannot be accessed.
- Accessing during TDZ gives ReferenceError.

Function vs Function Expression
- Function declarations are hoisted with full body.
- Function expressions (using var) are hoisted as `undefined`.

Example:
```js
console.log(x);   // undefined
var x = 10;

console.log(y);   // ReferenceError
let y = 20;
```

Interview Tip

- Yes, `let` and `const` are hoisted.
- They throw error if accessed before declaration due to TDZ.