Hoisting is something the JavaScript engine sets up **before** your code actually starts running — specifically during the **compile / creation phase**, not at "runtime" in the sense of executing your statements line by line.

Here's the sequence under the hood:

🛠️ The Two Phases of Execution

1️⃣ Creation Phase (a.k.a. compile time)

- The engine parses your code top to bottom, building the scope chain.
- For **function declarations**, it stores both the name **and** the full function body in memory.
- For **var** variables, it creates the binding and initializes it to `undefined`.
- For **let/const**, it creates the binding but keeps it in the **Temporal Dead Zone** (TDZ).
- This is when "hoisting" happens — before any statement is executed.

2️⃣ Execution Phase (runtime)

- Statements run in order, top to bottom.
- When the engine reaches an assignment or function expression, it updates the value in memory.
- This is when the TDZ ends for `let` / `const`.

🧩 Why People Get Confused

- You *see* hoisting effects at runtime (e.g., calling a function before its definition "works"), but the setup that makes that possible happened earlier, in the creation phase.
- The runtime is just taking advantage of the bindings that were already prepared.

Quick Visual Timeline

Code Copy

```
[ Creation Phase ]
  - Scope created
  - Declarations & bindings prepared
  - Functions fully stored
  - var set to undefined; let/const in TDZ

[ Execution Phase ]
```

- Run code top-to-bottom
- Assign values to vars, let/const
- Execute function calls