JavaScript Hoisting

(Memory Creation vs Execution Phase Explained Deeply)

Subtitle: Understand how JavaScript moves declarations to memory before execution and why some code works before it appears.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

🧠 Introduction

Hoisting is one of the most confusing JavaScript behaviors for beginners because it looks like:

“Code is running before it is written.”

That is not what actually happens.

JavaScript does not move code physically. Instead, it separates execution into **two phases** and prepares memory before running anything.

Once you understand this, many “random bugs” stop feeling random.

⚙️ Step 1: JavaScript Execution Model (Foundation Layer)

Every JavaScript program runs in two main phases:

1. Memory Creation Phase (Setup Phase)

- Variables are allocated in memory
- Functions are stored in memory
- Nothing is executed yet

2. Execution Phase

- Code runs line by line
- Values are assigned
- Functions are called

🧠 Mental Model:

PHASE 1: MEMORY

- variables reserved

- functions stored

PHASE 2: EXECUTION

- code runs line by line

- values assigned

⚙️ Step 2: What is Hoisting?

Hoisting means:

JavaScript moves declarations to the top of their scope during the memory phase.

Important clarification:

- Only **declarations** are hoisted
- Not assignments

🧱 Step 3: Variable Hoisting (let, const, var)

1. var Hoisting (Old Behavior)

console.log(a);

var a = 10;

Internally behaves like:

var a; // hoisted

console.log(a); // undefined

a = 10;

Key Output:

- undefined (not error)

⚠️ Problem with var:

- accessible before initialization
- leads to unpredictable bugs

2. let & const Hoisting (Modern Behavior)

```
console.log(a);


let a = 10;
```

Result:

❌ Reference Error

Why?

Because they exist in:

Temporal Dead Zone (TDZ)

⛔ Step 4: Temporal Dead Zone (TDZ)

TDZ = time between:

- variable creation (memory phase)
- variable initialization (execution phase)

```
console.log(a); // error

let a = 10;
```

During TDZ:

- variable exists
- but cannot be accessed

⚙️ Step 5: Function Hoisting

Function Declaration (Fully Hoisted)

```
greet();


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

Why it works:

Function is fully stored in memory phase.

Function Expression (Not Fully Hoisted)

```
greet();


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

Result:

❌ Error

Because:

- variable exists
- but function assignment happens later

🔄 Step 6: Memory vs Execution Breakdown

Example:

```
var x = 5;


function test() {
  console.log("running");
}


test();
```

Memory Phase:

x = undefined

test = function stored

Execution Phase:

x = 5

test() runs

🌍 Step 7: Real World Mental Model

Think of JavaScript like a restaurant:

Memory Phase:

- chef prepares ingredients
- menu items defined
- nothing is cooked yet

Execution Phase:

- orders start coming
- food is prepared step by step

Hoisting = menu already prepared before customers arrive

⚠️ Step 8: Common Mistakes

❌ 1. Assuming all variables behave same

- var ≠ let/const behavior

❌ 2. Ignoring TDZ

Leads to runtime crashes

❌ 3. Thinking code physically moves

It doesn't — only memory setup changes

❌ 4. Mixing function types

Declaration vs expression confusion causes errors

🧪 Step 9: Mini Exercises

Exercise 1

Predict output:

```
console.log(a);

var a = 10;
```

Exercise 2

Predict output:

```
console.log(a);

let a = 10;
```

Exercise 3

Test function declaration vs expression hoisting

🧠 Step 10: Mini Quiz

1. What happens in memory creation phase?
2. Why does var return undefined instead of error?
3. What is TDZ?
4. Why are function declarations hoisted fully?

🚀 Step 11: Thinking Upgrade

If you understand hoisting:

- JavaScript stops feeling random
- errors become predictable
- debugging becomes logical

👉 You start thinking like an engine, not a user.

📌 Step 12: Summary

- JS runs in memory + execution phases
- hoisting = declarations moved to memory phase
- var = hoisted with undefined
- let/const = TDZ (not accessible early)
- function declarations = fully hoisted
- function expressions = partially hoisted