Function Declarations vs Function Expressions — Hoisting Rules

1️⃣ Function Declaration

Javascript Copy

```
sayHi(); // ✅ Works — full function is hoisted

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

- **Hoisted completely**: both the name and the function body are available from the start of the scope.
- Can be called **anywhere in its scope** — even before its definition line.
- Works fine for recursion, since the function name is in scope inside its own body.

2️⃣ Function Expression (var)

Javascript Copy

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

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

- **Hoisting still happens** for the **variable** name (`sayHi`), because `var` is hoisted.
- BUT the value (the function) is **not** assigned until that line executes.
- So before the assignment, it's `undefined`.
- Hoisting here is **variable hoisting**, not **function hoisting**.

3️⃣ Function Expression (let / const)

Javascript Copy

```
sayHi(); // ❌ ReferenceError (Temporal Dead Zone)

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

- The binding is hoisted, but stays in the **Temporal Dead Zone (TDZ)** until the declaration line runs.
- Accessing it before that throws `ReferenceError`.

🧠 Recursion Note

- **Function declarations**: The name is bound in its entire scope → recursion works.
- **Named function expressions**: The name is only visible **inside** the function body:

Javascript Copy

```
const factorial = function fact(n) {
  return n <= 1 ? 1 : n * fact(n - 1); // recursion works via 'fact'
};
```

- **Anonymous function expressions**: No internal name; recursive calls require the variable name.