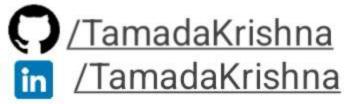
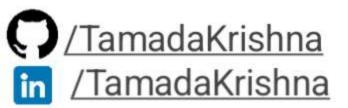
Hoisting in JavaScript



Hoisting

- Hoisting in JavaScript refers to the behavior where variable and function declarations are moved to the top of their containing scope during the compilation phase, before the code is executed.
- This allows you to use functions and variables before they are actually declared in the code. However, only the declarations are hoisted, not the initializations.



Variable Hoisting

 Variable hoisting in JavaScript involves the behavior where variable declarations (using var) are moved to the top of their enclosing function or global scope during the compilation phase.

Using var:

```
Hoisting.js

console.log(a); // undefined, because `a` is hoisted but not yet assigned
var a = 3;
console.log(a); // 3, after assignment
```



Using let and const:

- Variables declared with let and const are also hoisted, but they are not initialized.
- Accessing them before their declaration results in a ReferenceError due to the temporal dead zone (TDZ).

```
Hoisting.js

console.log(b); // ReferenceError: Cannot access 'b' before initialization
let b = 5;
```



Function Hoisting

Function Declarations:

- When you declare a function using a function declaration, the entire function (both its name and body) is hoisted to the top of its scope.
- This means you can call the function before the point where it is actually defined in the code.

```
Hoisting.js

greet(); // Works fine because the function is hoisted

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

Function Expressions:

- Function expressions are not hoisted in the same way.
- If you assign a function to a variable (whether using var, let, or const), only the variable declaration is hoisted, not the function assignment.

```
Hoisting.js

console.log(fn); // undefined, because the assignment is not hoisted

var fn = function() {
   console.log('Hello from function expression');
};

fn(); // Works fine, because `fn` is now a function
```

