Javascript

Data, Variables and Hoisting

Data

In JavaScript, data represents values used and manipulated by a program. These can be numbers, text, boolean values, etc.

- marks obtained
- age of a person
- roll number of a student
- bank account number

Variables

variables are containers that you can store values in .

There are four ways to declare variables :

```
var a = 5;
let b = 6;          (2015)
const c = 7;        (2015)
d = 8;
```

```
const x;        Error

const y=5;
y=6;            Error
```

var vs let

Feature	var	let
Scope	Function scoped	block scoped
Hoisting	Hoisted with undefined	Hoisted but not initialized
Redeclaration	Allowed in the same scope	Not allowed in the same scope
Global Object Binding	Yes (Window.varName)	No
	Legacy	Recommended

Scope

```
function f1(){
    if (true) {
        var x = 5;
    }
    console.log(x);
}
```

prints 5

```
function f1(){
    if (true) {
        let x = 5;
    }
    console.log(x);
}
```

Error

Hoisting

Hoisting is JavaScript's behaviour of moving declarations to the top of their scope during the compilation phase before the code actually runs

Only declarations are hoisted not initializations

```
console.log(a);
var a = 5;
```

→

```
var a;
console.log(a);
a = 5;
```

let and const are hoisted, but not initialised, so they cannot be used until the line they are declared.
They stay in temporal dead zone (TDZ).

```
console.log(a)  →  Error
let a=5;
    or
const a=5;
```

Function declarations are fully hoisted

```
f1();

function f1() {
    console.log("Hello MySirG");
}
```

Entire function is moved to the top of the scope

Function expressions are not hoisted like declarations

```
f1();          ⟶ Error

var f1 = function() {
        console.log("Hello MySirG");
}
```

Hoisting

```
console.log(a);
var a = 5;
```

→ prints undefined

```
console.log(b);
let b = 5;
```

→ Error as value unavailable

Redeclaration

```
var x = 10;
var x = 20;
```

No Error

```
let a = 10;
let a = 20;
```

Syntax Error