**JavaScript Hoisting**

Hoisting is nothing but, Mechanism of giving/allocating memory to variable in the creation

**Hoisting** in JavaScript is a behavior in which a function or a variable can be used before declaration.

For example:

var age=23

console.log(age)

----------------

console.log(age)

var age=23

-----------------

ex2:

var age=23

var age=23

What will happen?

Will it be hoisted once or twice?

----------------------------------------------------------------

Partially hoisted:

When the value is not given in creation phase is called partially hoisted.

Variables are partially hoisted

lets see what happens with function:

1. functions are fully hoisted

function sayHello(){

console.log("hello")

}

sayHello()

2)

sayHello()

function sayHello(){

console.log("hello")

}

=>it should be undefined (like variable, but it shows defined)

=>in js visualizer check

=>it will not show undefined....

=>functions also hoisted..

=>functions are completely hoisted

=>memory allocated and value given in creation phase

Function expression example:

1

var sayHello=function(){le

console.log("hello")

}

say Hello()

2

say Hello()

var sayHello=function(){

console.log("hello")

}

---

let example:

console.log(name)

let name="hello"

=>let and const also hoisted but not initialised

What is temporal deadzone in JavaScript?

A temporal dead zone (TDZ) is the block where a variable is not accessible until the moment the computer initializes it with a value.

======================================================

// using test before declaring

console.log(test); // undefined

var test;

The above program works and the output will be undefined. The above program behaves as

// using test before declaring

var test;

console.log(test); // undefined

Since the variable test is only declared and has no value, undefined value is assigned to it.

**Note**: In hoisting, though it seems that the declaration has moved up in the program, the actual thing that happens is that the function and variable declarations are added to memory during the compile phase.

**Variable Hoisting**

In terms of variables and constants, keyword var is hoisted and let and const does not allow hoisting.

For example,

// program to display value

a = 5;

console.log(a);

var a; // 5

In the above example, variable a is used before declaring it. And the program works and displays the output 5. The program behaves as:

// program to display value

var a;

a = 5;

console.log(a); // 5

However in JavaScript, initializations are not hoisted. For example,

// program to display value

console.log(a);

var a = 5;

**Output**

undefined

The above program behaves as:

var a;

console.log(a);

a = 5;

Only the declaration is moved to the memory in the compile phase. Hence, the value of variable a is undefined because a is printed without initializing it.

Also, when the variable is used inside the function, the variable is hoisted only to the top of the function. For example,

// program to display value

var a = 4;

function greet() {

b = 'hello';

console.log(b); // hello

var b;

}

greet(); // hello

console.log(b);

**Output**

hello

Uncaught ReferenceError: b is not defined

In the above example, variable b is hoisted to the top of the function greet and becomes a local variable. Hence b is only accessible inside the function. b does not become a global variable.

**Note**: In hoisting, the variable declaration is only accessible to the immediate scope.

If a variable is used with the let keyword, that variable is not hoisted. For example,

// program to display value

a = 5;

console.log(a);

let a; // error

**Output**

Uncaught ReferenceError: Cannot access 'a' before initialization

While using let, the variable must be declared first.

**Function Hoisting**

A function can be called before declaring it. For example,

// program to print the text

greet();

function greet() {

console.log('Hi, there.');

}

**Output**

Hi, there

In the above program, the function greet is called before declaring it and the program shows the output. This is due to hoisting.

However, when a function is used as an **expression**, an error occurs because only declarations are hoisted. For example;

// program to print the text

greet();

let greet = function() {

console.log('Hi, there.');

}

**Output**

Uncaught ReferenceError: greet is not defined

If var was used in the above program, the error would be:

Uncaught TypeError: greet is not a function

Hoisting can cause undesirable outcomes in your program. And it is best to declare variables and functions first before using them and avoid hoisting.

In the case of variables, it is better to use let than var.