Hoisting

-> Hoisting in a javascript is a mechanism where variables and function declarations are moved to the top of their containing scope (either global or function scope) before the execution of the code.

Ex:
```
getName();
console.log(x);
console.log(getName);

var x=7;
function getName()
    {
    console.log("Javascript");
    }
```

O/p:
```
Javascript
undefined.
getName()
    {console.log("Javascript");
    }
```

→ Whenever a JS Program is run, a global execution Context is created, which has 2 phases

1) Memory Creation
2) Code Creation

→ Before the code creation, memory is allocated to all the variables and functions, so we can use them even before they are executed.

→ Variables are initialized to undefined when they declared and function definition, whole code is stored as it is in memory creation phases.

Ex: console.log(x); // ReferenceError - x is not defined.
Now in execution context, 'x' will not be present and also no value been assigned, so shows error..
Means 'x' is no way initialized in program, and you are trying to access the 'x'

→ 'undefined' means variables has been declared, but value is not assigned, whereas 'not defined' means variable is not declared.

Ex: var getName = () => {
    console.log("JS")
}

⇒ Arrow functions are treated like variables.

1) using let & const

console.log(myFun); // ReferenceError: cannot access 'myFun' before initialization.

const myFun = () =>
{ console.log("Hello");
}

2) using var:

console.log(myFun); // output: undefined

var myFun = () =>
{
console.log("Hello");
}

→ Variable declarations are scanned and are made undefined

→ function declaration are scanned and are made available.

Hoisting for let and Const

→ variables declared with let and const are hoisted, but they are not initialized. This means they cannot be accessed before their declaration, leading to a Reference Error.

→ They exist in a temporal dead zone, [It is a period in JS where variables declared with let and const cannot be accessed before their declaration.

```
console.log(a);
let a = 10;
console.log(5)
const b = 20
```

// Reference Error: cannot access 'b' before initialization.

Function Expressions

Function expressions are not hoisted in the same way as function declaration.

⇒ only the variable declaration is hoisted, but the function itself it not available until the code reaches the assignment

Ex: `console.log(myFnc); //o/p :- undefined`

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

```
console.log(myFnc); //ReferenceError: cannot access
                    'myFnc' before initialization
```

```
let myFnc = function ()
    {
    console.log ("Hello");
    }
```

```
let a=10;
Console.log(a);
var b=100;
```

→ window object is same as global object

In console

```
window.b
  =100
window.a
  =undefined
```

[Because variables using let & const are stored in different memory location]

→ let & const variables are not attached to global object

⇒ At global, this = window

In Console

```
this.b
  =100
this.a
  =undefined
```

```
let a=10
let a=10
```

→ we get SyntaxError: 'a' has already been declared

```
let a=10
var a=10
```

⇒ we get same SyntaxError.

var a=10;

var a=10;

→ we won't get any error.

const b;

b=10;

→ we get syntax Error: Missing initializer in const declaration.

→ Means we have to initialize the value in same line

Errors

→ Syntax Error is similar to compile Error

→ Reference Error falls under run time error.

→ Syntax Error – violation of JS syntax

→ type Error → while trying to re-initialize const variable.

→ Reference Error - while trying to access variable which is not there in global memory.