**HOISTING**

Hoisting is a phenomenon in JavaScript where you will be able to access variable and functions even before you have initialized or assigned value to it.

1. **Hoisting With VAR**

* The declaration part of a **var** variable is hoisted to the top of its scope during the compilation phase.
* This means that the variable is recognized throughout the entire scope, even if it's accessed before its actual declaration in the code.

console.log(x); // Outputs: undefined

var x = 5;

console.log(x); // Outputs: 5

* When a variable is hoisted, it is automatically initialized with undefined until an actual value is assigned.
* The variable **y** is hoisted with an implicit initialization, and the first **console.log(y)** outputs **undefined**.

console.log(y); // Outputs: undefined

var y;

* **Function Declaration with Var**
  + Function declarations using **var** are hoisted, allowing you to call the function before its declaration in the code.

greet(); // Outputs: Hello!

function greet() {

  console.log("Hello!");

}

* + **var** declarations are function-scoped or globally scoped.
  + **var** variables can be redeclared within the same scope without causing an error, which can lead to unexpected behaviour.
* **Function Expression with Var**

try {

  nonHoistedFunction(); // TypeError: nonHoistedFunction is not a function

} catch (error) {

  console.error(error.message);

}

var nonHoistedFunction = function () {

  console.log("I am not hoisted!"); };

* + Function expressions, where a function is assigned to a variable, behave differently.
  + The variable declaration (myFunctionExpression) is hoisted to the top, but the function assignment (function () {...}) is not hoisted.
  + Attempting to call the function before its assignment results in undefined.
  + You can call the function after the assignment, and it works as expected.
* **Arrow Function with Var:**

var myArrowFunction = () => {

  return "Hello, world!";

};

console.log(myArrowFunction());

* Arrow functions, when assigned to variables, follow the same behaviour as function expressions.
* The variable declaration (myArrowFunction) is hoisted, but the function assignment (() => {...}) is not.
* Attempting to access the variable before its assignment results in an error.
* You can call the function after the assignment, and it works as expected.

1. **Hoisting With Let & Const**

* **let** and **const** declarations are hoisted. But it’s different from var.

console.log(a); // ReferenceError: Cannot access 'a' before initialization

console.log(b); // prints undefined as expected

let a = 10;

console.log(a); // 10

var b = 15;

console.log(window.a); // undefined

console.log(window.b); // 15

* It looks like let isn't hoisted, but it is, let's understand.
  + Both a and b are actually initialized as *undefined* in hoisting stage.
  + But var b is inside the storage space of GLOBAL, and a is in a separate memory object called script, where it can be accessed only after assigning some value to it first i.e. one can access 'a' only if it is assigned. Thus, it throws error.
* **Temporal Dead-Zone:**
  + Time since when the let variable was hoisted until it is initialized some value.
  + So any line till before "let a = 10" is the TDZ for a
  + Since a is not accessible on global, its not accessible in window/this also. window.b or this.b -> 15; But window.a or this.a ->undefined, just like window.x->undefined (x isn't declared anywhere.
* **Reference Error** are thrown when variables are in temporal dead zone.
* **Syntax Error** doesn't even let us run single line of code.
* **Let is a stricter version of var. Now, const is even stricter than let.**

let a1;

a1 = 10;

console.log(a1) // 10. Note declaration and assigning of a is in different lines.

const b1;

b1 = 10;

console.log(b1); // SyntaxError: Missing initializer in const declaration.(This type of declaration won't work with const. const b = 10 only will work)

const b3 = 100;

b3 = 1000; //this gives us TypeError: Assignment to constant variable.

* **Types of Error: Syntax, Reference, and Type.**
  + **Uncaught Reference Error: x is not defined at ...**
    - This Error signifies that x has never been in the scope of the program. This literally means that x was never defined/declared and is being tried to be accessed.
  + **Uncaught Reference Error**: cannot access 'a' before initialization
    - This Error signifies that 'a' cannot be accessed because it is declared as 'let' and since it is not assigned a value, it is its Temporal Dead Zone. Thus, this error occurs.
  + **Uncaught Syntax Error**: Identifier 'a' has already been declared
    - This Error signifies that we are redeclaring a variable that is 'let' declared. No execution will take place.
  + **Uncaught SyntaxError**: Missing initializer in const declaration
    - This Error signifies that we haven't initialized or assigned value to a const declaration.
  + **Uncaught TypeError**: Assignment to constant variable
    - This Error signifies that we are reassigning to a const variable.