**Hoisting**

Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

'use strict';

window.onload = () => {

    console.log(x);

    let x;

}

It returns:

Uncaught ReferenceError: Cannot access 'x' before initialization

So let and const keywords prevent us from using the variable before initialization.

If it is var:

'use strict';

window.onload = () => {

    console.log(x);

    var x;

}

Then it returns undefined. Cuz it is declared but not defined.

If it is a function:

'use strict';

window.onload = () => {

    console.log(x());

    var x=()=>{

        // function body

    };

}

Then it returns: Uncaught TypeError: x is not a function. Because it doesn’t know that x is a function. After Hoisting our code looks like this=>

'use strict';

window.onload = () => {

var x

    console.log(x());

    x=()=>{

        // function body

    };

}

“use strict” just makes sure that we declare our variable.

'use strict';

window.onload = () => {

    x = 3;

    console.log(x);

}

Here it gives an error that x is not defined if it weren’t for “use strict”

Then this would return 3

**Null vs Undefined**

In JavaScript, undefined means a variable has been declared but has not yet been assigned a value, such as:

 var testVar;

    alert(testVar); //shows undefined

    alert(typeof testVar); //shows undefined

null is an assignment value. It can be assigned to a variable as a representation of no value:

var testVar = null;

    alert(testVar); //shows null

    alert(typeof testVar); //shows object

From the preceding examples, it is clear that undefined and null are two distinct types: undefined is a type itself (undefined) while null is an object.

null === undefined // false

null == undefined //true

null === null // true

We have got a bookshelf with no books🡪if we take a book it is undefined

We haven’t got bookshelf 🡪if we take a book it is null

**IIFE (Immediately Invoked Function Expression)**

An **IIFE** (Immediately Invoked Function Expression) is a [JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript) [function](https://developer.mozilla.org/en-US/docs/Glossary/Function) that runs as soon as it is defined.

window.onload = () => {

    (( ) => {

        console.log("IIFE-1");

    })( );

    (function( ) {

        console.log("IIFE-2");

    })( );

We just create a function inside parenthesis and we also put parenthesis after the first parenthesis so that it is called once the page is loaded.

**Inheritance in ECMAScript 5 (ES5) call( ) method**

 const Animal = function(fname, age) {

        this.fname = fname;

        this.age = age;

        this.GetName = () => {

            console.log(this.fname);

        }

    }

    const Fox = function(fname, age) {

        Animal.call(this, fname, age);

    }

    const fox1 = new Fox("fox-1", 12)

    fox1.GetName();

we use call method to call animal constructor function in Fox. And We gotta pass “this” as well, the fox itself. Call is like a bind( ) method. The diffrence is that it calls that class immediately.

**How to equalize constructor functions(classes)**

In order for them to be equal we gotta equalize A’s prototype (base function’s) to B’s prototype!

 const Animal = function(fname, age) {

        this.fname = fname;

        this.age = age;

        this.GetName = () => {

            console.log(this.fname);

        }

    }

    const Fox = function(fname, age) {

        Animal.call(this, fname, age);

    }

    const fox1 = new Fox("fox-1", 12)

    Animal.prototype = Fox.prototype

    console.log(fox1 instanceof Animal);

the result is true.

**Function to generate a random number between a specific range**

function randomNumber(min, max) {

return Math.floor(Math.random() \* (max - min) + min);

}

For example 1-9 Math.floor((0-8)+1)

**Object.freeze(obj)**

We use this method to prevent changing the properties of the object

 const Animal = function(fname, age) {

        this.fname = fname;

        this.age = age;

    }

    const ani = new Animal("Vahid", 12)

    Object.freeze(ani)

    ani.fname = "jello";

Cannot assign to read only property 'fname' of object '#<Animal>'

A frozen object can no longer be changed, freezing an object prevents new properties from being added to it, existing properties from being removed

**Closure**

Closure means that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.

 function OuterFunction() {

        var outerVariable = 1;

        function InnerFunction() {

            alert(outerVariable);

        }

        InnerFunction();

    }