The difference between Zero, null and undefined.

- **Zero :** This is the numerical value zero. It is considered falsy in JavaScript, which means it evaluates to false in a boolean context. For example:

Null: This is a special value that represents the absence of a value. It is also considered falsy in JavaScript. Null is an object . It can be assigned to a variable as a representation of no value. **JavaScript never sets a value to null**. That must be done programmatically.

Undefined: means a variable has been declared but has not yet been assigned a value. undefined is a type by itself (undefined). Unassigned variables are initialized by JavaScript with a default value of undefined.

```
let s;
console.log(s);
let x=null;
console.log(x) ;


//Data type
console.log(typeof undefined);
console.log(typeof null );


//Undefined and null are falsy:
console.log(!!undefined);
console.log(!!null);

// Comparing undefined and null
console.log(undefined == null) //it compares only the values.and both 0 so reture true

console.log(undefined === null) //it compare both type and value
```

```
undefined                                   search.js:2
null                                        search.js:4
undefined                                   search.js:8
object                                      search.js:9
false                                       search.js:13
false                                       search.js:14
true                                        search.js:17
false                                       search.js:19
>
```

The difference between var , let and const.

Scope:

1)var: *Global scoped* or function scoped. The scope of the *var* keyword is the global or function scope. It means variables defined outside the function can be accessed globally, and variables defined inside a particular function can be accessed within the function. If the user tries to access it outside the function, it will display the error.

```
//global scope
var number = 50

function print() {
  var square = number * number
  console.log(square)
}
console.log(number) // 50
print() // 2500

//local scope
function print() {
    var number = 50
    var square = number * number
    console.log(square)
  }

  print() // 2500

  console.log(number)
  // ReferenceError: number is not definedD
```

2)let : The scope of a *let* variable is only block scoped. It can't be accessible outside the particular block ({block}).

```
let a = 10;
function f() {
    if (true) {
        let b = 9


        console.log(b); //9
    }


    console.log(b); // It gives error as it
    // defined  outside if  block
}
f()


console.log(a) //10 global scope
```

3)const: block scope When users declare a *const* variable, they need to initialize it, otherwise, it returns an error

```
// const
const b=8;
console.log(b); //8
```

2) redeclare and reassign variables

1) **Var: The user** can re-declare the variable using *var* and the user can update the *var* variable.

```
var a = 10;
var a = 8;
a = 7;
console.log(a) //7
```

If users use the var variable before the declaration, it initializes with the *undefined* value. The output is shown in the console.

```
var a
console.log(a)  //undefined
```

2)let: Users cannot re-declare it display `Uncaught Syntax Error: Identifier 'a' has already been declared`, the variable defined with the *let* keyword but can updata it.

```
1
2   let a = 10;
3
4   a = 7;
5   console.log(a)  //7
6
```

3)const : if We are changing the value of the const variable so that it returns an error. The output is shown in the console.

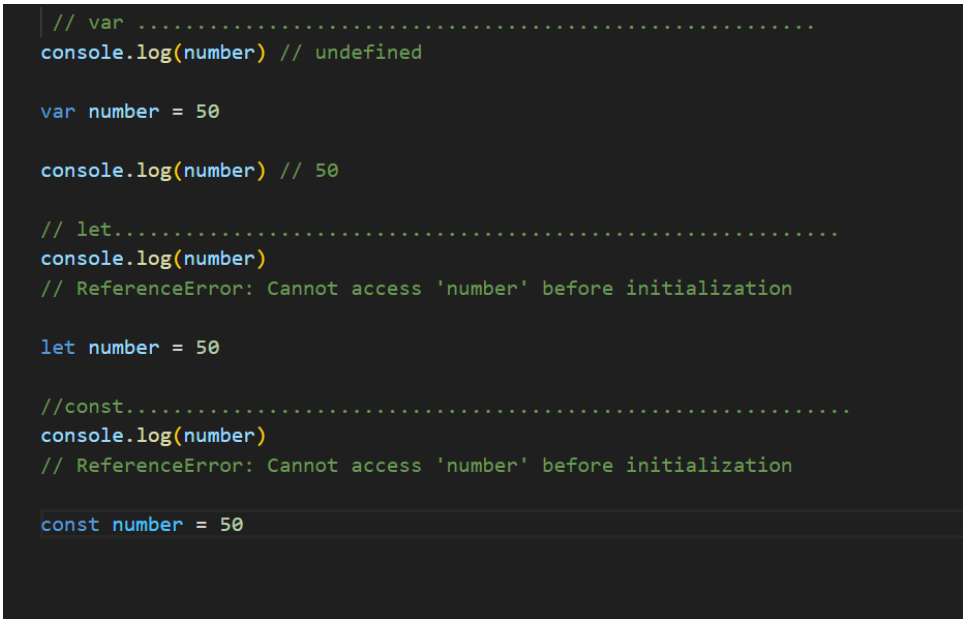
```
const a = 10;
function f() {
    a = 9
    console.log(a)
}
f();//TypeError:Assignment to constant variable.
```

3)Hoisting:

Var : hoisting done, with initializing as ‘default’ value

Let : Hoisting is done, but not initialized (this is the reason for the error when we access the let variable before declaration/initialization

Const : Hoisting is done, but not initialized (this is the reason for the error when we access the const variable before declaration/initialization.



```
// var ..........................................................
console.log(number) // undefined

var number = 50

console.log(number) // 50

// let..........................................................
console.log(number)
// ReferenceError: Cannot access 'number' before initialization

let number = 50

//const.........................................................
console.log(number)
// ReferenceError: Cannot access 'number' before initialization

const number = 50
```