Interview Series

Part-1

Var vs Let vs Const

By Expresso Educator

The Story Starts with introducing our 3 characters:

① var : The old school hero, it came before any character is JS and is recognized in school by everyone.

② let : The class hero, new admission to school. everyone in the class knows him

③ Const: The introvert guy, but the most intelligent.

Before starting today game let's understand the rules

Scope Rules

Rule-1 : Doesn't matter where the var is he is recognized every where in school.

So, school here is a function.

e.g

```
1 function a () {
2         console.log(x);
3         If(x) {
4             var x = 10;
5         }
1 }
```

here x is declared with var keyword at line 4 But it accessible on 2,3,4,5

Rule 2 :- Doesn't matter where the let is he is recognized every where in class. Not outside the class. Here class is a block ({})

e.g

```
1. function a () {
2.     If (true) {
3.         let x = 2;
4.         let y = 3
5.         console.log(x);
6.     }
7. }
```

block

x is declared on line 3 but accessible on 3, 4, 5

Same rule applies for const as well

So, from rule number 1 and rule number 2 we can say var is function scoped and let is block scoped.

Redeclaration And Reinitialization Rules

Rule-3 :- we can redeclare var but not let and const; means

```
var a = 2
var a = 3
```

✓

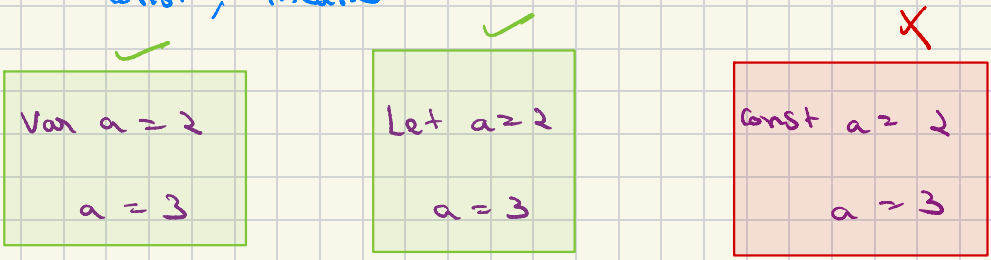
```
Let a = 2
Let a = 3
```

✗

Same for const

Redeclaration is not allowed for let and const

Rule-4: we can reinitialize let and var but not const, means



Hoisting and TDZ (temporal dead zone) Rules

Basic Rule

Every variable can hoist (will be declared on the top) of their respective scopes. Only var doesn't go to TDZ, any variable inside TDZ is not accesible before initialization.

Rule-5: variables declared with var, their declaration will go on the top inside the function they are declared and doesn't goto TDZ. e.g

```
function a(){
    console.log(x);
    var x = 2;
}
```

The above function is equal to:

```
function a () {
    var x;
    console.log (x);

    x = 2;
}
```

Declaration went up on the top in function.

Rule - 6:- Let and const will also be hoisted in the block scope but there is no meaning of them being hoisted as the declaration will go to TDZ. So, don't hoist.

So, Game is clear 3 players (let, var and const) with 6 Rules.

Let's Begin

Total Stages - 8

Stage 1 :-

```
1  function test() {
2    var x = 10;
3    if (true) {
4      var x = 20;
5      console.log(x); // ?
6    }
7    console.log(x); // ?
8  }
9  test();
```

→ Think and guess the output?

① var x = 10 (value of variable x is 10)

② If condition will pass due to always true.

③ var x = 20 → By Rule 1 scope of x is line 2 to 7 and By Rule 3 redeclaration is allowed as x is already declared in line 2. So, x is 20 now.

④ At line 5, output is 20.

⑤ x value is not changed on line 7, so output is again 20.

Output :-

20

20

Stage-2 :-

```
1. function example() {
2.   var x = 10;
3.   if (true) {
4.     let x = 20;
5.     console.log(x); // ?
6.   }
7.   console.log(x); // ?
8. }
9. example();
```

→ Think and guess the output?

① var x = 10, value of x is 10

② If condition will pass, always true

③ Let x = 20 → Rule-2, x will be accessible only on line 4 and 5 (block scoped)

④ line 5, will print x with let as it is in block scope (line 3 to 6) → 20

⑤ line 7, will print x with var as it is in function scope (line 1 to 8) → 10

output :-

20

10

Stage -3 :-

```
console.log(a);
let a = 5;
```

→ Think and guess the output?

Rule-6 a will hoist but will go to TDZ i.e not accessible before initialization.

So, output will be reference error : a is not accessible.

Stage -4 :-

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

→ Think and guess the output?

Rule-4 : const is not reinitializable.

So, output will be error.

Abhi toh bs sunvat h ,

y Stage aur baaki h

Stage-5 :-

```
const obj = { name: "Alice" };
obj.name = "Bob";
console.log(obj.name); // ?
```

→ Think and guess the output?

① Rule 4 stops us from reinitialization not mutation.

not allowed

```
const obj = { name: "Alice" }
obj = { name: "John" }
```

→ But this is allowed (this is mutation)

② So, output is "Bob"

Stage-6 :-

```
console.log(x);
var x = 10;

console.log(y);
let y = 20;
```

→ Think and guess the output?

① Rule-5 → declaration of var x will be hoisted.

Rule-6 → no benefit of hoisting let y.

(2) The above code is equal to:

```
var x;
console.log(x);
x = 10;
console.log(y);
let y = 20;
```

→ output is:

undefined
error

Stage-7:

```
for (var i = 0; i < 3; i++) {
  setTimeout(() => console.log(i), 100);
}
```

→ Think and guess the output?

Rule-1:- var is a function scoped and **Rule-6:-** hoist it's declaration on the top.

(1) The above code can be written as:

```
var i;
i = 0;
{
  If (i < 3) → true
    {
      setTimeout(() => console.log(i), 100);
    }
  i++; → i = 1
}
```

↑ will wait for 100 ms

{

If (i < 3) → true

{

setTimeout (( ) => console.log (i), 100),

↑ will wait for 100 ms

i++; } → i = 2

}

{

If (i < 3) → true

{

setTimeout (( ) => console.log (i), 100),

↑ will wait for 100 ms

i++; } → i = 3

}

{

If (i < 3) → false

{

setTimeout (( ) => console.log (i), 100),

i++; }

}

② After 100 ms all setTimeout will execute the callback and they will try to find value of i.

③ How many variable i we have? only one at the top with value now being 3.

④ So, all setTimeout callback's will print 3

output :-

3
3
3

Stage-8:

```
for (let j = 0; j < 3; j++) {
  setTimeout(() => console.log(j), 100);
}
```

→ Think and guess the output?

Rule-2 let is block ({}) scoped.

① The above code can be written as:

```
{
  let j = 0;
  if(j < 3) <= true
    {
      setTimeout ((()=> console.log(j), 100);
                  ↑ wait for 100 ms
    }
}
```

```
{
    let j = 1;
    if(j < 3)  <= true
    {
        setTimeout((()=> console.log(j), 100);
    }
}
{
    let j = 2;
    if(j < 3) <= true
    {
        setTimeout((()=> console.log(j), 100);
    }
}
{
    let j = 3; <= false
    if(j < 3)
    {
        setTimeout((()=> console.log(j), 100);
    }
}
```

wait for 100 ms

wait for 100 ms

② Every block have their own j, so after 100 ms, every setTimeout callback will print their own block j.

output:-

Keep learning

Keep Growing

Yours

Expresso Educator