10 video

Block scope & shadowing in JS

Block is defined by {...}. Everything written inside these bracket comes in a block.

It is a compound statement which starts with '{' and end with '}'. It is used to - group multiple items in a single unit.

These group of statement can be use where JS expects a single statement.

→ Each block acts separate block scope / execution context where variable / function defined inside a block will be hoisted. outside of this block ~~will be~~ no element will be accessible which is declaration inside this block.

→ let and const are block scoped because they have separate block in which they are hoisted.

```
var a = 10;
{
  var b = 11;
  console.log(a);
  console.log(b);
}
console.log(a);
console.log(b);
```

o/p

10
11
10

ReferenceError: b is defined in this scope.

Shadowing

```
var a = 100;
let b = 100;
{ var a = 10;
  let b = 20;
  const c = 30;
  console.log(a);
  // // (b);
  // // (c);
}
console.log(a);
```

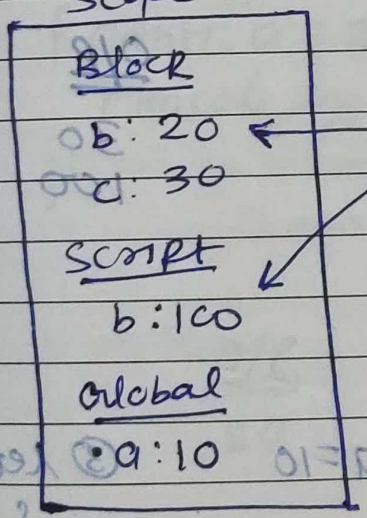
O/P

10
20
30
10

* If there exists a same named variable outside the block then this variable shadows the value of outside variable.

a is 10] This is called shadowing

As a is changed to 10 because they were referring to same memory location.



- Both b indicate different memory location.
- Similar thing happens with const also.

```
let a = 100;
{ var a = 10;
  let b = 20;
  const c = 30;
  console.log(a);
  console.log(b);
  console.log(c);
}
console.log(a);
```

both associated to global scope

↓

point same memory location.

O/P

10
20
30
10

Scope → when code is executed upto ③ ↓

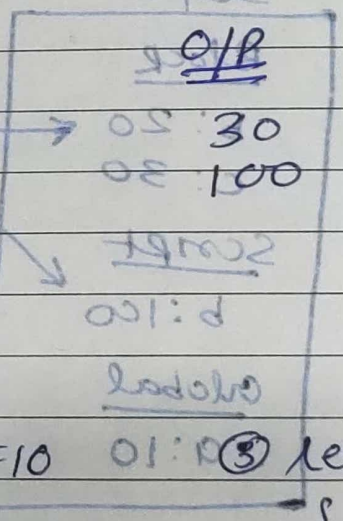
→ Block
b: undefined
c: undefined
→ Global
a: 10

value of global will be modified to 10. When block is executed then again value of a will not change to 100.

— line ⑤ & ⑱ both* are refering to same memory location.

Function Shadowing: also work in similar fashion

```
const c = 100;
function x() {
    const c = 30;
    console.log(c);
}
x();
console.log(c);
```



* ① let a = 10
{ let a = 100; }

② var a = 10
{ var a = 100; }

③ let a = 10
{ var a = 100; }

④ var a = 10
{ let a = 100; }

①, ②, ④ → Shadowing (works fine)

③ → Shadowing is not possible - Illegal shadowing

```
let a = 10;
function x() {
  var a = 100;
}
```

✓ [Not illegal shadowing]

⇒ Similar Behaviour with const variable also

Block scope also follows lexical scope.

↓

follows scope chain pattern.

```
const a = 20;
{
  const a = 100;
  {
    const a = 200;
    console.log(a);
  }
}
```

← debugged

Scope
Block
a: 200
Block
a: 100
script
a: 20

O/P

20

```
const a = 20;
{
  const a = 100;
  {
    console.log(a);
  }
}
```

O/P

100

⇒ All scope concept work exactly in same fashion for arrow function also.