Episode 9 : Block Scope & Shadowing in JS

What is a **Block**?

- Block aka *compound statement* is used to group JS statements together into 1 group. We group them within {...}

```
{
    var a = 10;
    let b = 20;
    const c = 30;
    // Here let and const are hoisted in Block scope,
    // While, var is hoisted in Global scope.
}
```

- Block Scope and its accessibility example

```
{
    var a = 10;
    let b = 20;
    const c = 30;
}
console.log(a); // 10
console.log(b); // Uncaught ReferenceError: b is not defined
```

  - Reason?

    - In the BLOCK SCOPE; we get b and c inside it initialized as *undefined* as a part of hoisting (in a seperate memory space called **block**)
    - While, a is stored inside a GLOBAL scope.
    - Thus we say, *let* and *const* are BLOCK SCOPED. They are stored in a separate mem space which is reserved for this block. Also, they can't be accessed outside this block. But var a can be accessed anywhere as it is in global scope. Thus, we can't access them outside the Block.

What is **Shadowing**?

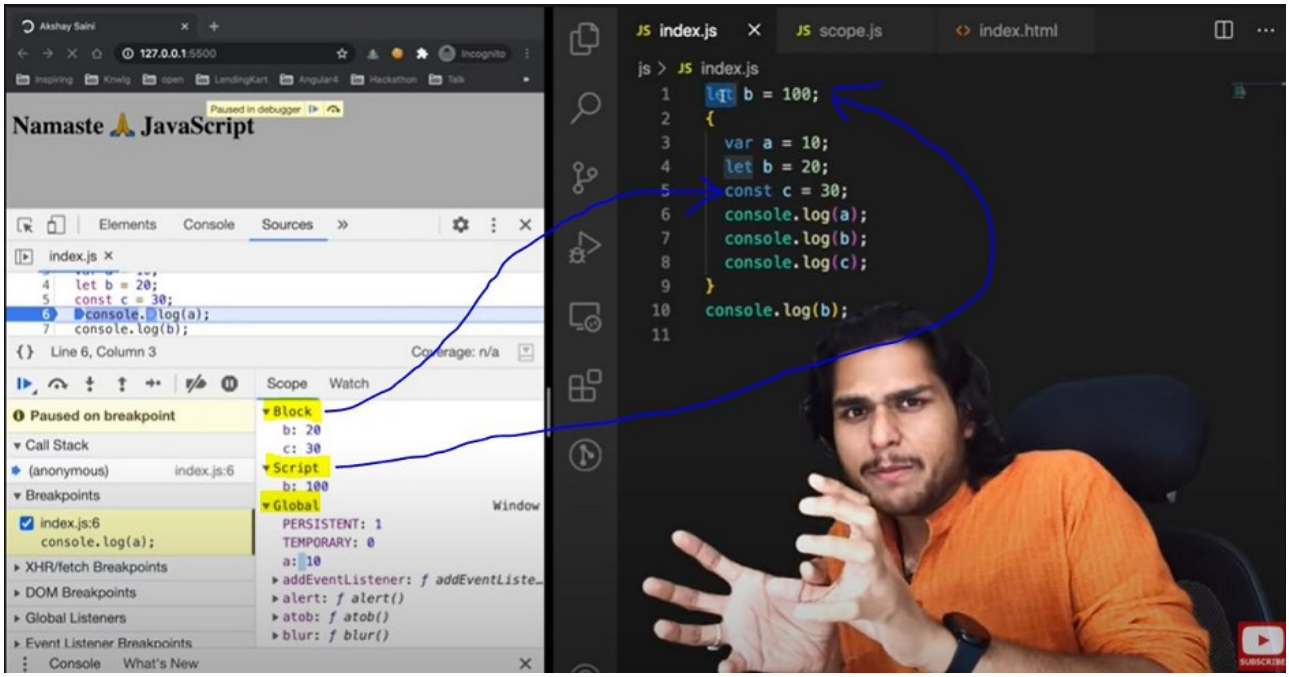
-

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

```
        console.log(c); // 30
    }
    console.log(a); // 10, instead of the 100 we were expecting. So
block "a" modified val of global "a" as well. In console, only b and c
are in block space. a initially is in global space(a = 100), and when
a = 10 line is run, a is not created in block space, but replaces 100
with 10 in global space itself.
```

- So, If one has same named variable outside the block, the variable inside the block *shadows* the outside variable. **This happens only for var**
- Let's observe the behaviour in case of let and const and understand it's reason.

```
let b = 100;
{
    var a = 10;
    let b = 20;
    const c = 30;
    console.log(b); // 20
}
console.log(b); // 100, Both b's are in separate spaces (one in
Block(20) and one in Script(another arbitrary mem space)(100)). Same
is also true for *const* declarations.
```



- Same logic is true even for **functions**

```
const c = 100;
function x() {
    const c = 10;
    console.log(c); // 10
}
```

```
x();
console.log(c); // 100
```

What is **Illegal Shadowing**?

- ```
  let a = 20;
  {
      var a = 20;
  }
  // Uncaught SyntaxError: Identifier 'a' has already been declared
  ```

  - We cannot shadow let with var. But it is **valid** to shadow a let using a let. However, we can shadow var with let.
  - All scope rules that work in function are same in arrow functions too.
  - Since var is function scoped, it is not a problem with the code below.

    ```
    let a = 20;
    function x() {
        var a = 20;
    }
    ```

Watch Live On Youtube below:
    ```