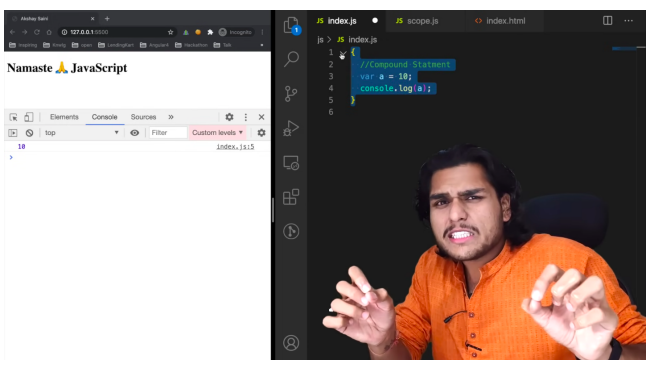
Block Scope and Shadowing in JS

Thursday, 17 August 2023 12:39 PM

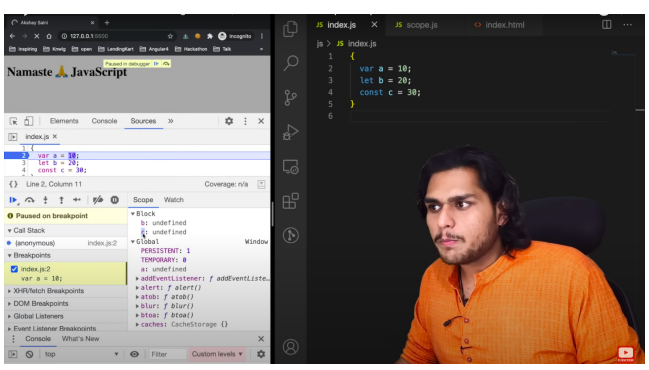
→ let and const variables are block-scoped.

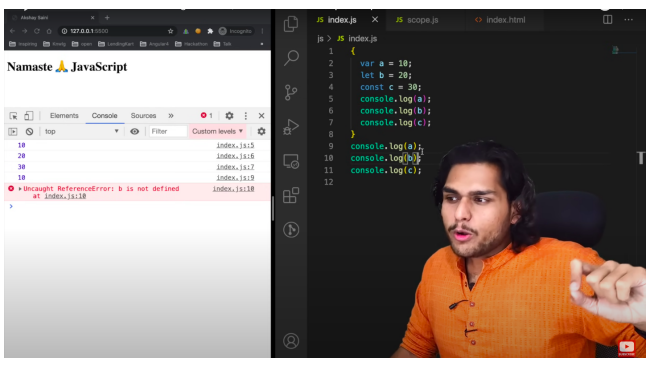
→ A block in JS in denoted by 2 curly braces. It combines multiple JS statements into a group. So, it is also called as a compound statement.



→ We group multiple statements together in a block so that we can use it where JS expects 1 statement. Eg; When we group statements while writing 'if' condition.

→ Block Scope : Block scope means whatever variables or f^s we can access inside a block.





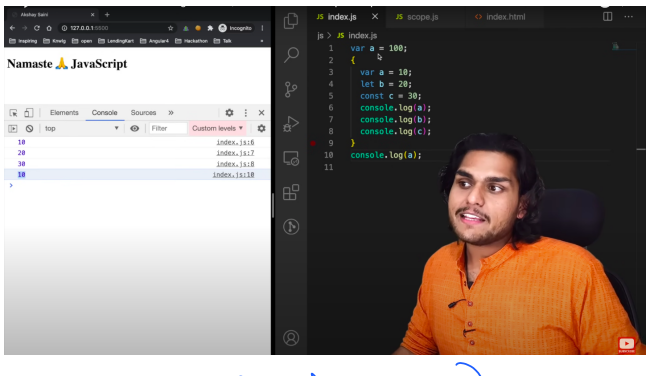
→ Now, we can see that var variables are global scoped and let & const variables are allocated memory in block scope. This block scope memory gets deleted as soon as the block of code ends.
We can also see in above that all variables are hoisted in JS i.e JS allocates memory to all variables in memory creation phase but let & const variables are in temporal deadzone.

→ Also according to the 2nd pic, we cannot access let & const variables outside the block in which they are defined. It gives reference error: b is not defined when we try to access 'b' outside the block because let & const variables are block scoped.

→ Shadowing :

In case of var. if we have a same named variable

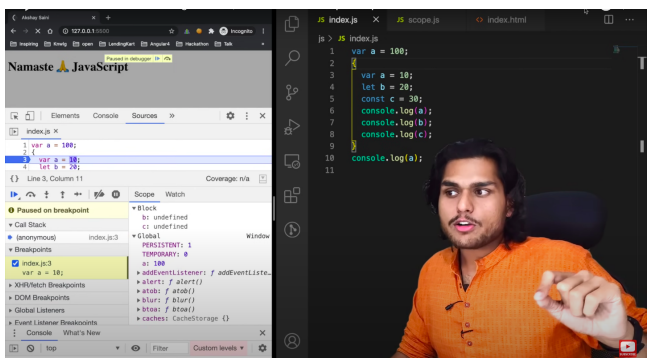
inside the block, then the same named variable inside the block shadows the global variable.

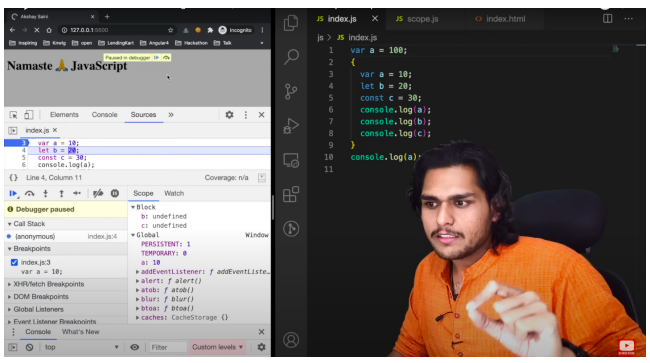


(demo of shadowing)

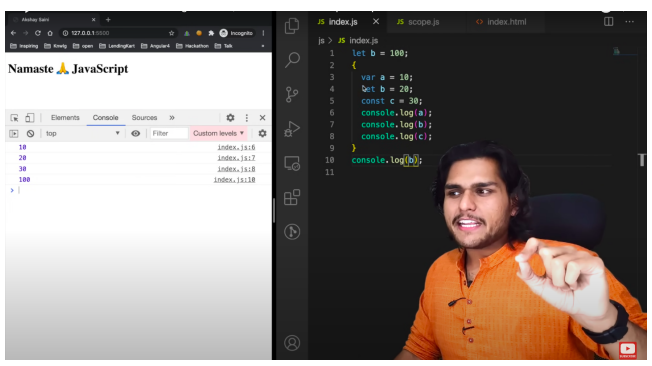
→ Also, if now we print the same named variable outside the block, then also 10 is printed because the previous variable was shadowed but it also modified the value of the variable in case of var because they both are pointing to the same memory location (in global object)

→ Browser demo of above explanation:

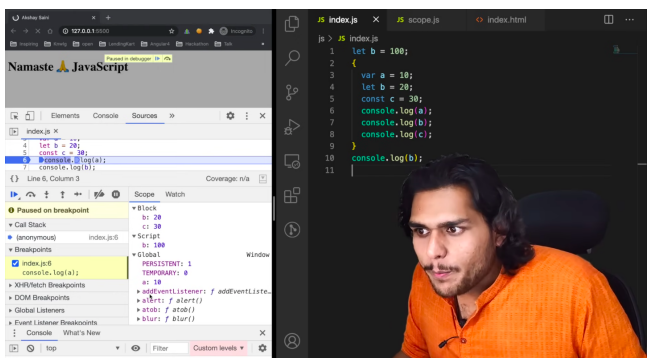




Note: But same is not the case with let & const variables.

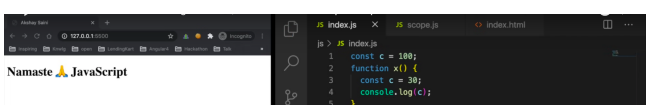


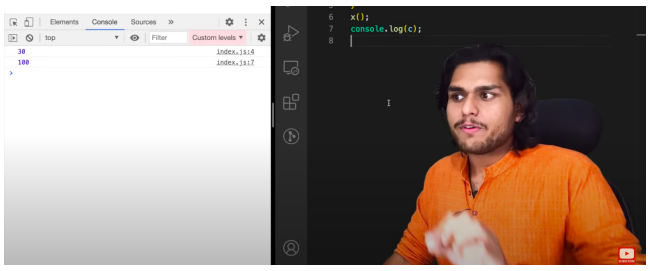
This is because let and const variables are block scoped.
In line 7, variable 'b' inside block scope shadows the 'b' outside block scope.
So, 20 is printed.



→ We can see that, another 'b' has been created in block scope, so when the block ends, block scope memory gets deleted. Thus, 100 is printed on line 10.

Since, b is a let variable, so it isn't stored inside global object, so value of b outside block scope isn't changed.
Similar things happen with const variables too.

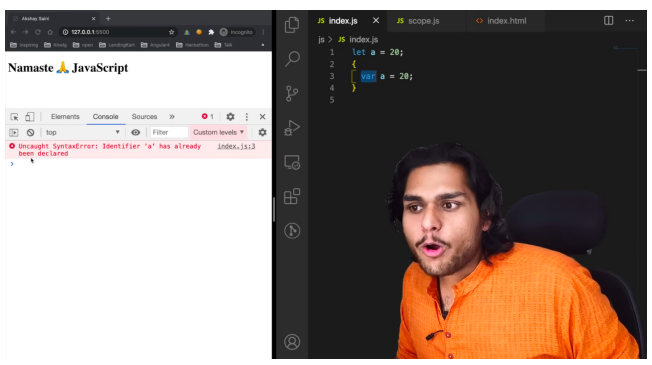




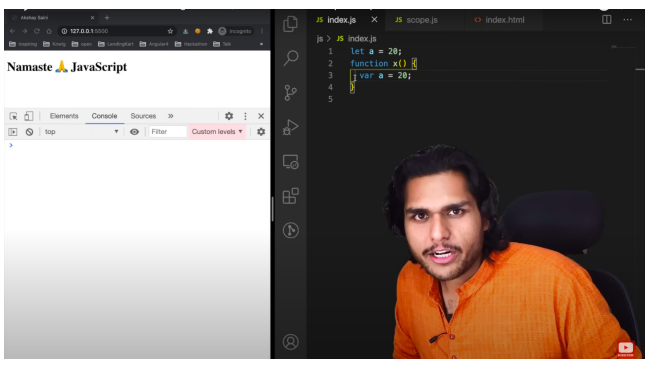
→ Shadowing also works with f° scope in the same way it does with block scope.

→ Illegal shadowing :

We cannot shadow a variable declared with let outside the block again by declaring it with var inside the block. This is illegal shadowing.

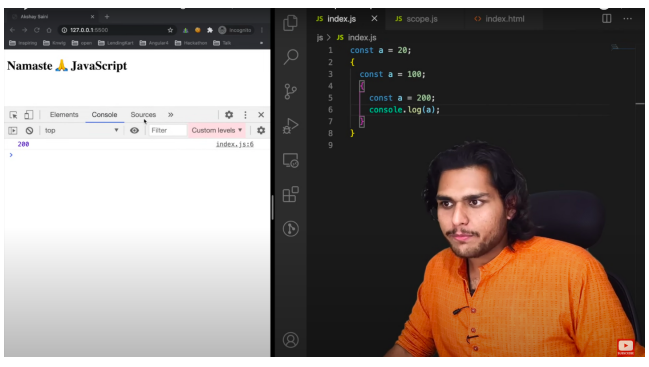


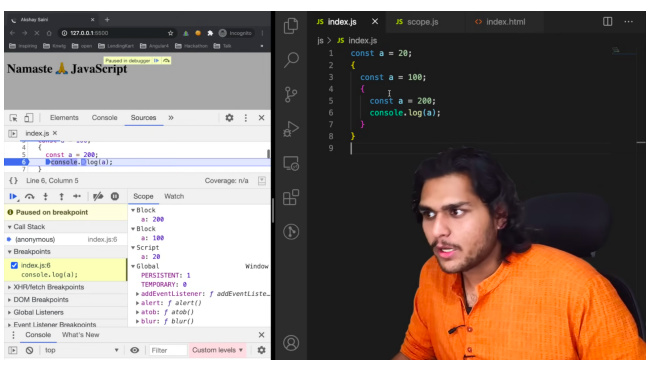
Note: However, a variable declared using let outside the block can be declared again using let inside the block. Also, a variable declared using var outside the block can be declared again using let inside the block.



→ This is also fine, as var variable is inside its boundaries, it is not interfering with the scope of global 'a'.

Note: Block scope also uses lexical scope.





So, this means that each block has access to the lexical environment of the previous scope.

Note: All the scope rules that work on functions, work the same way in the case of arrow functions too.