Namaste JavaScript

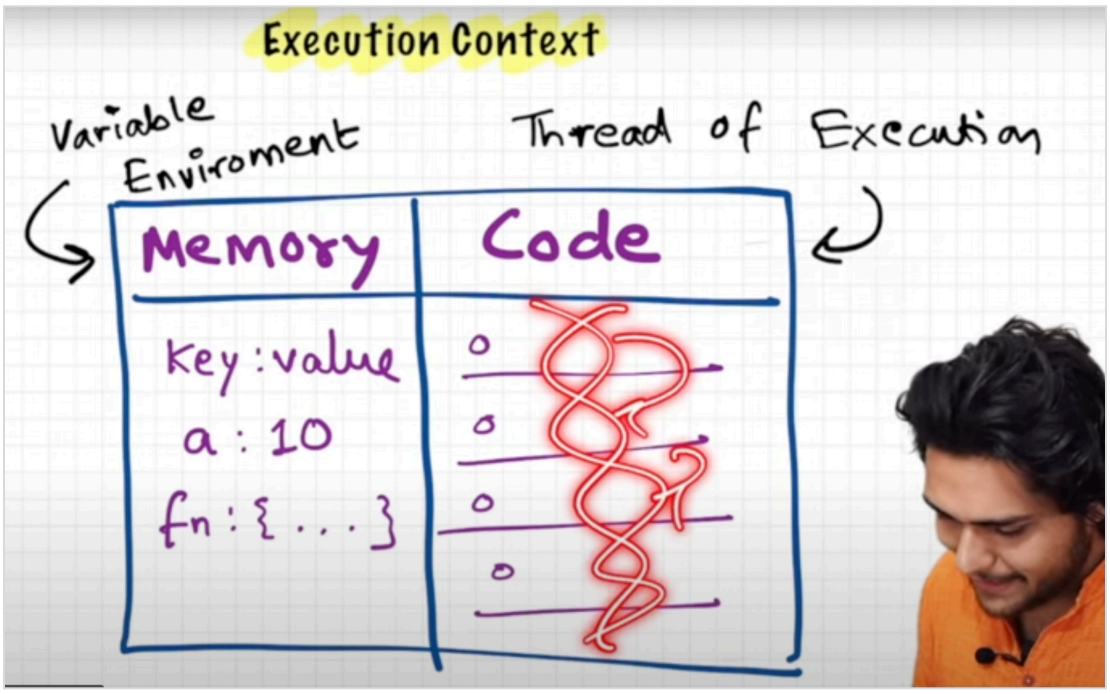
By Akshay Saini during Nov 2024

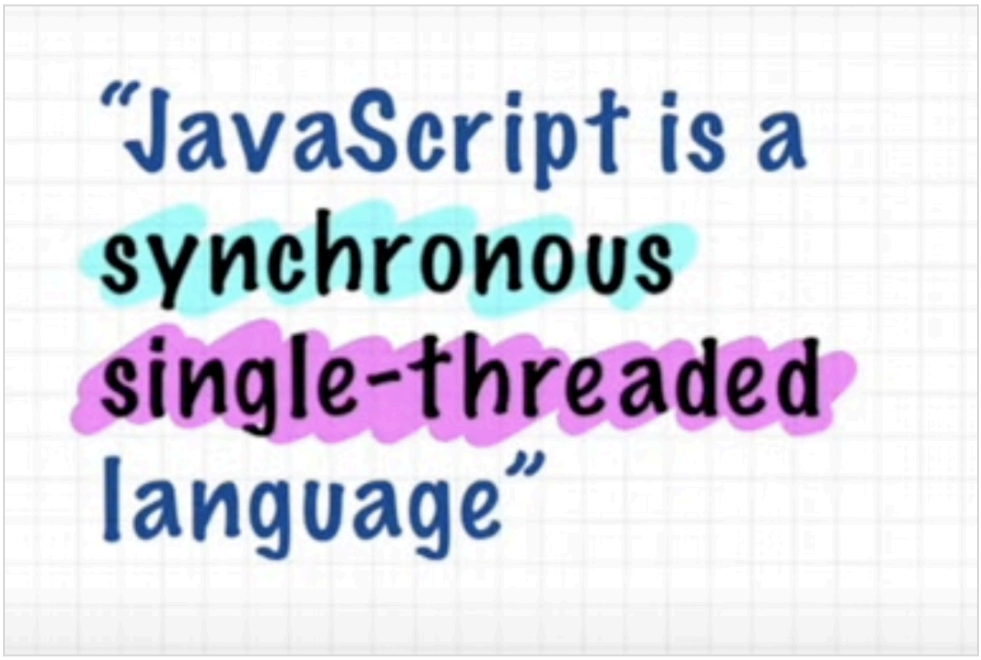
- How JavaScript Works 🔥& Execution Context
  - **Everything in Javascript, happens inside an Execution Context**

  - Execution Context is like big box or container
  - Execution Context has 2 major components

    - **Memory**, where all JS variables, funtions stored as key, value pair and used to call as **Variable Environment**

    - **Code**, all code executed in code component, executed line by line, one line at a time, calling as **Thread of Execution**





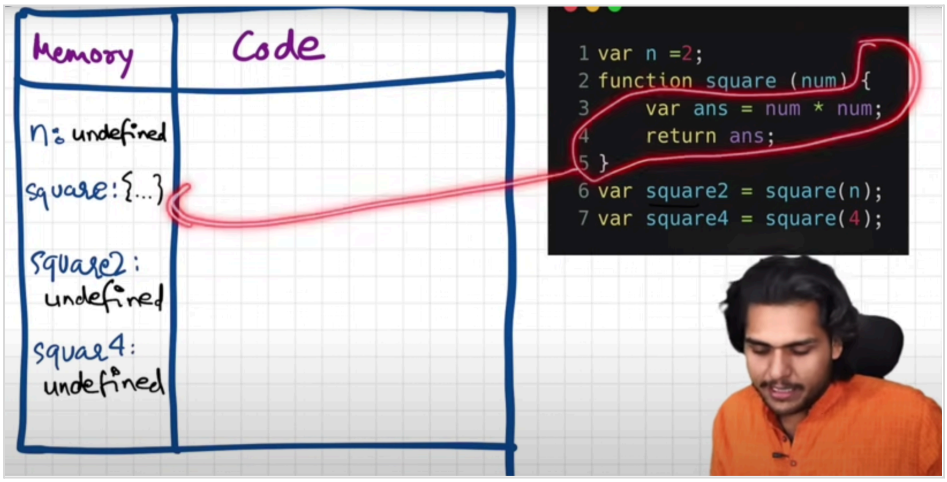
- JS is **Single Threaded**, since JS executed one command at a time only once with **Synchronously** (orderly from top one by one)

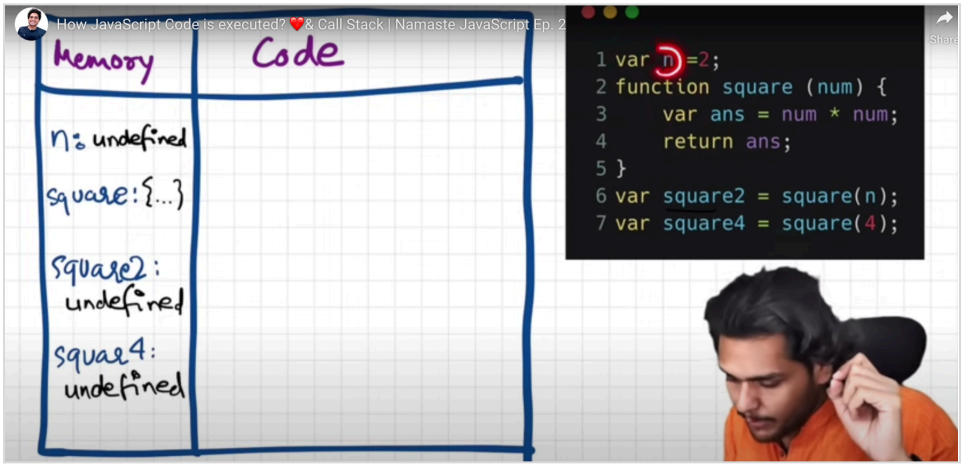
- How JavaScript Code is executed? ❤️& Call Stack
  - What happens when you run the JS code ? - **Global Execution Context will create with 2 phases**
    - Phase 1 - **Memory creation phase**, will allocate memory for all JS variables & functions
    - Phase 2 - **Code execution phase**, all computation & variable initialization will take care

  - **What happens when run below the piece of JS code ?**

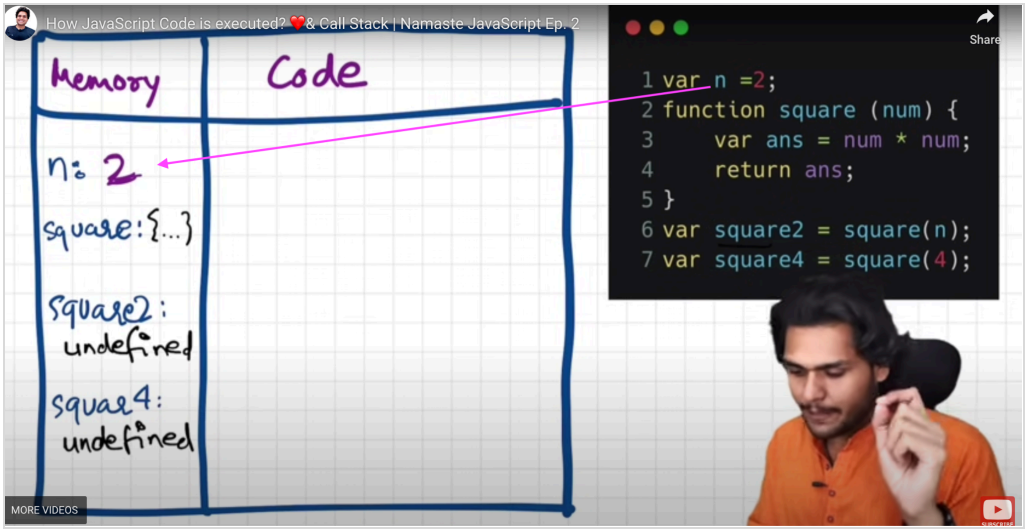
```
var n = 2;

function square(num) {
  var ans = num * num;
  return ans;
}

var square2 = square(n);
var square4 = square(4);

console.log("square2", square2);
console.log("square4", square4);
```

- **Initially Global Execution Context** will create and start allocating memory for all JS variables & functions
- During Memory creation phase 1
    - Initially for all variables assign with **undefined, undefined is special keyword which initially during memory creation phase assign to JS variable**
    - For functions, takes value as a whole function





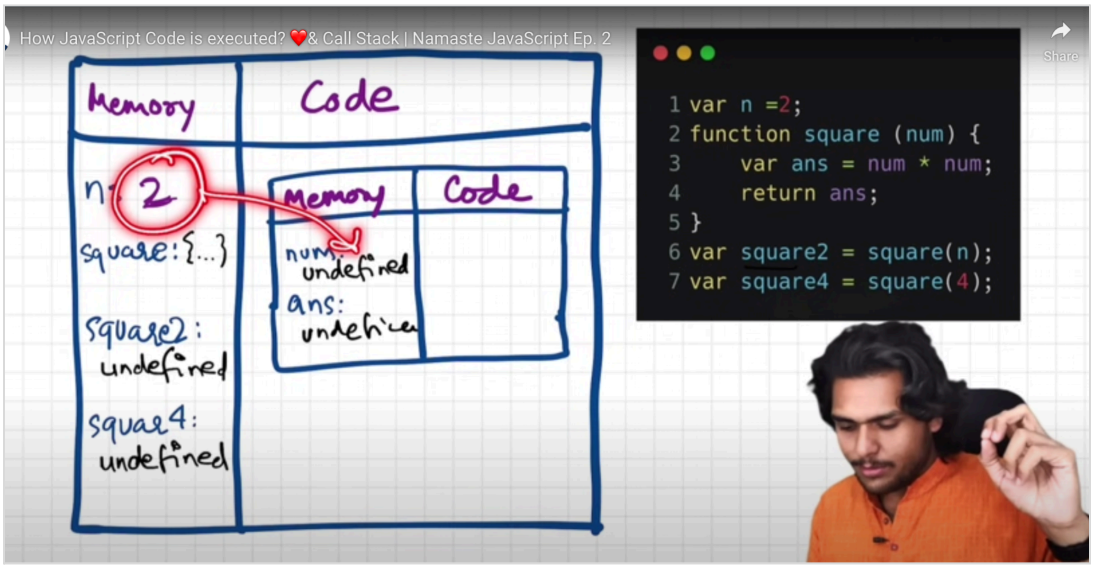
- During Code execution phase 2, actual calculation will take care
    - When control encounters, For variables, start initialize the value in memory by replacing the undefined

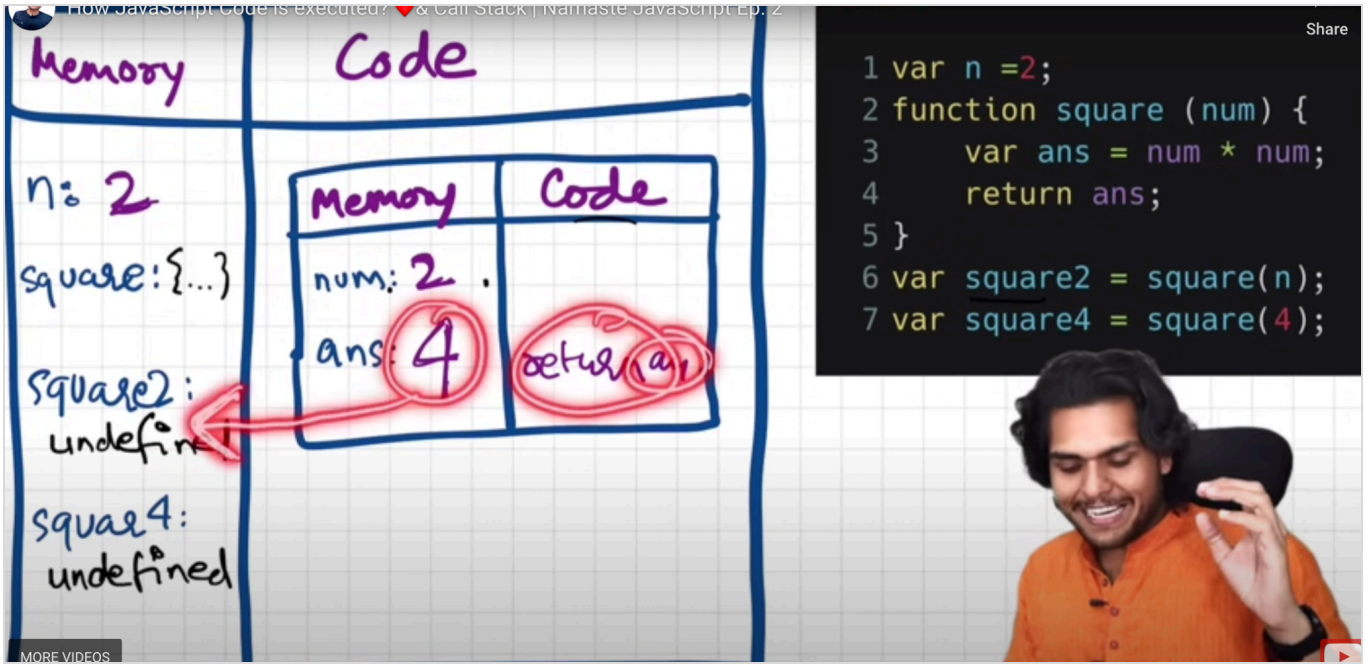


- Line #6, have a function invocation / calling / executed

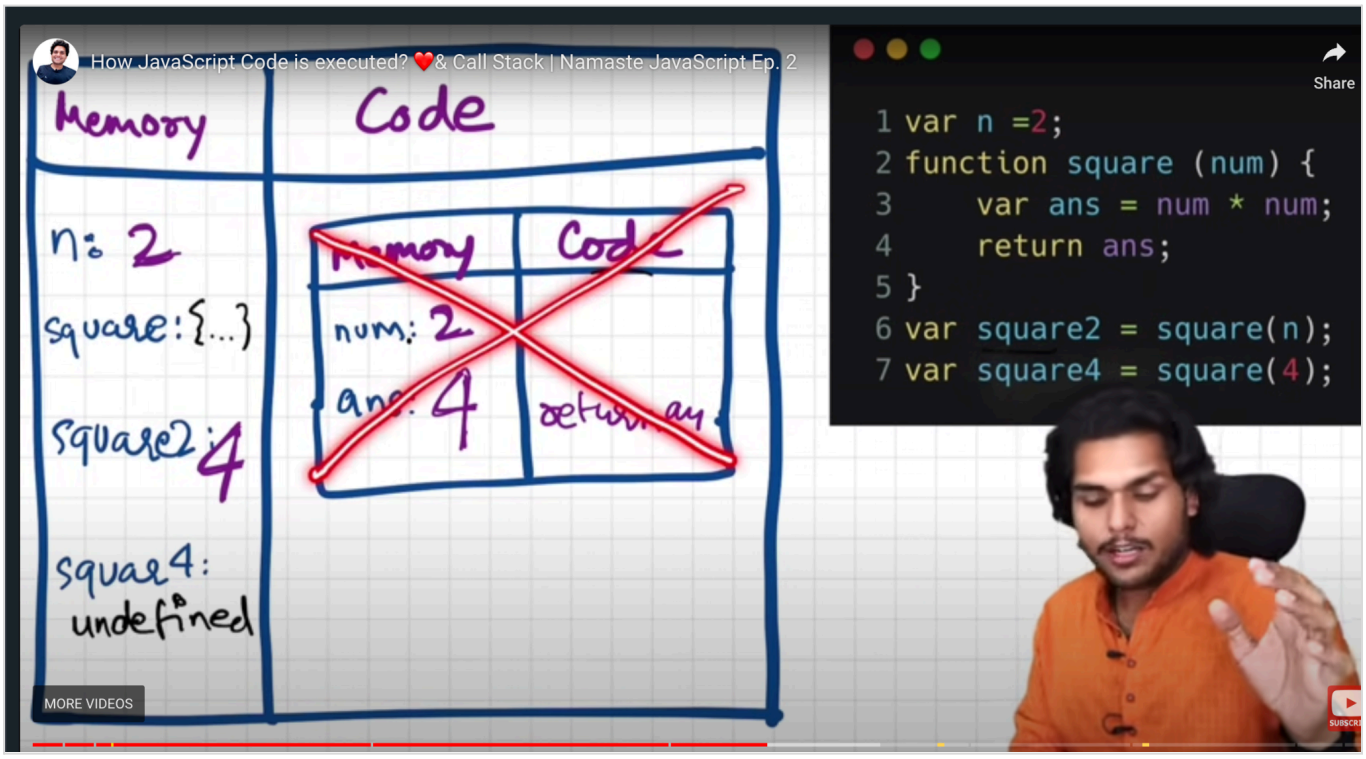
- Functions are heart of Javascript, act as mini program & behave differently compare to other programming language

- Line #6, When invoke / call the function, **brand new Execution Context will create within global and follow the same Memory creation phase & Code execution phase**

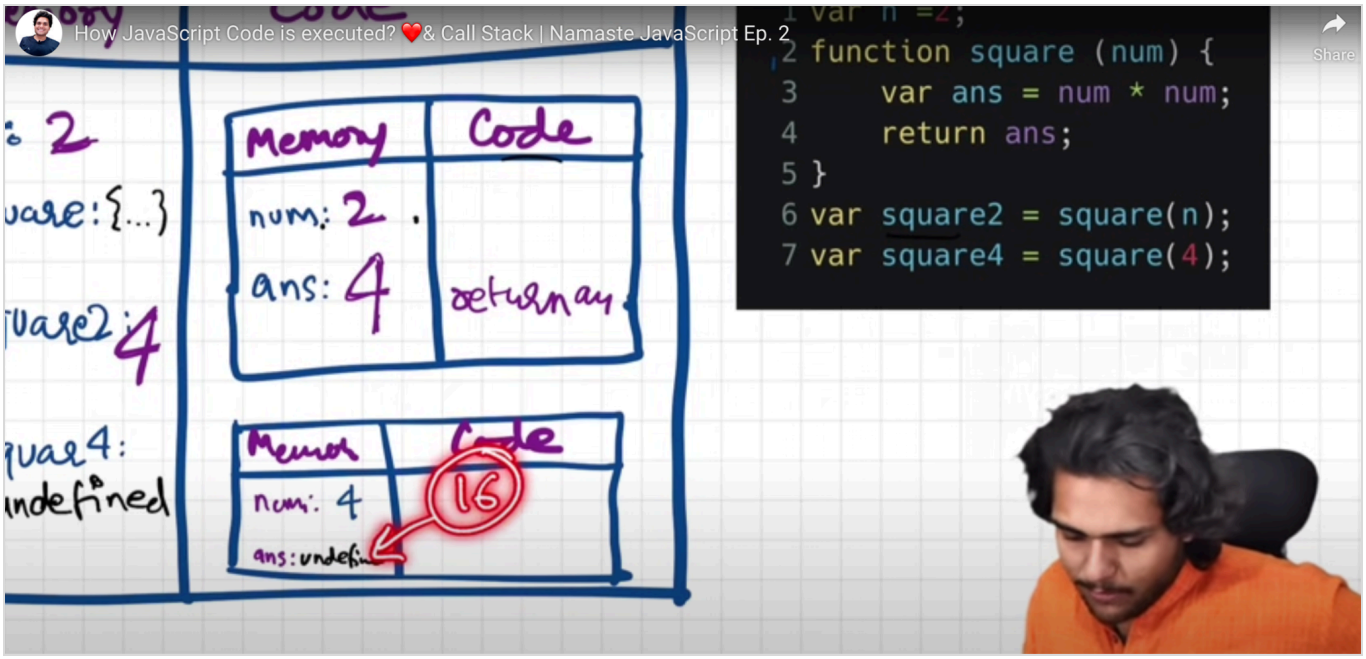


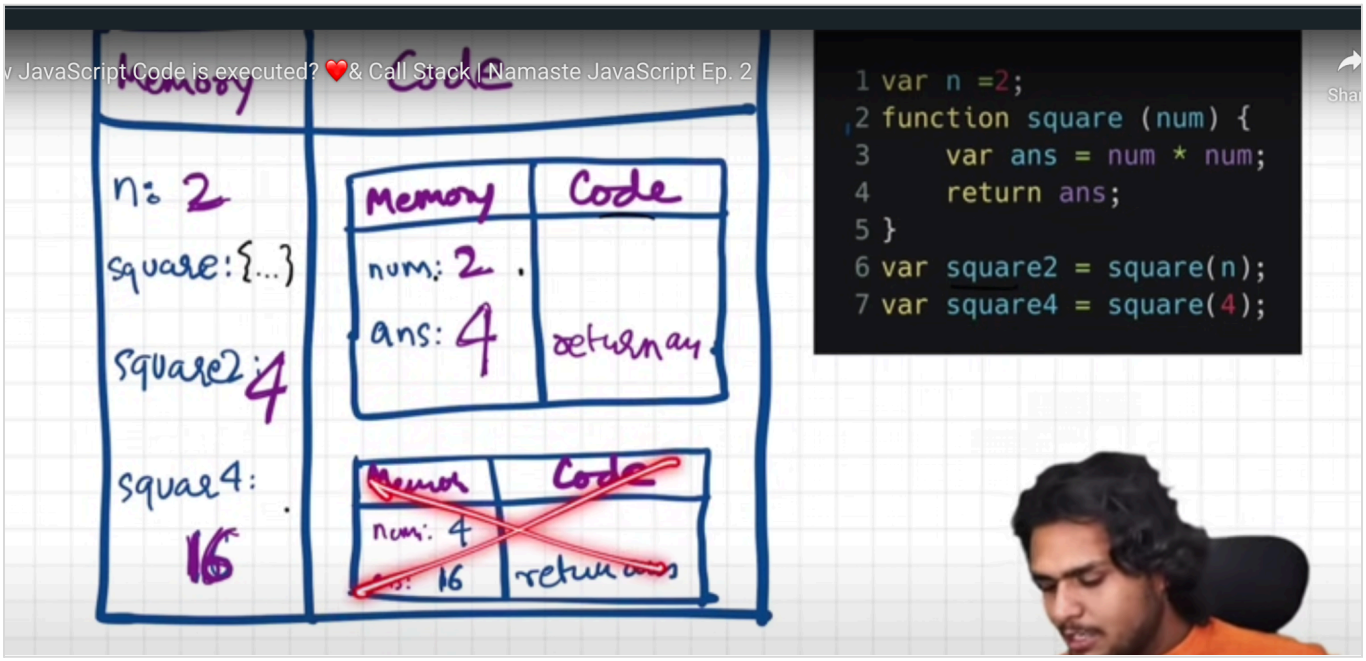


- **As soon as function completed its execution, its execution context will be removed / deleted & give the control to from where the function is called**

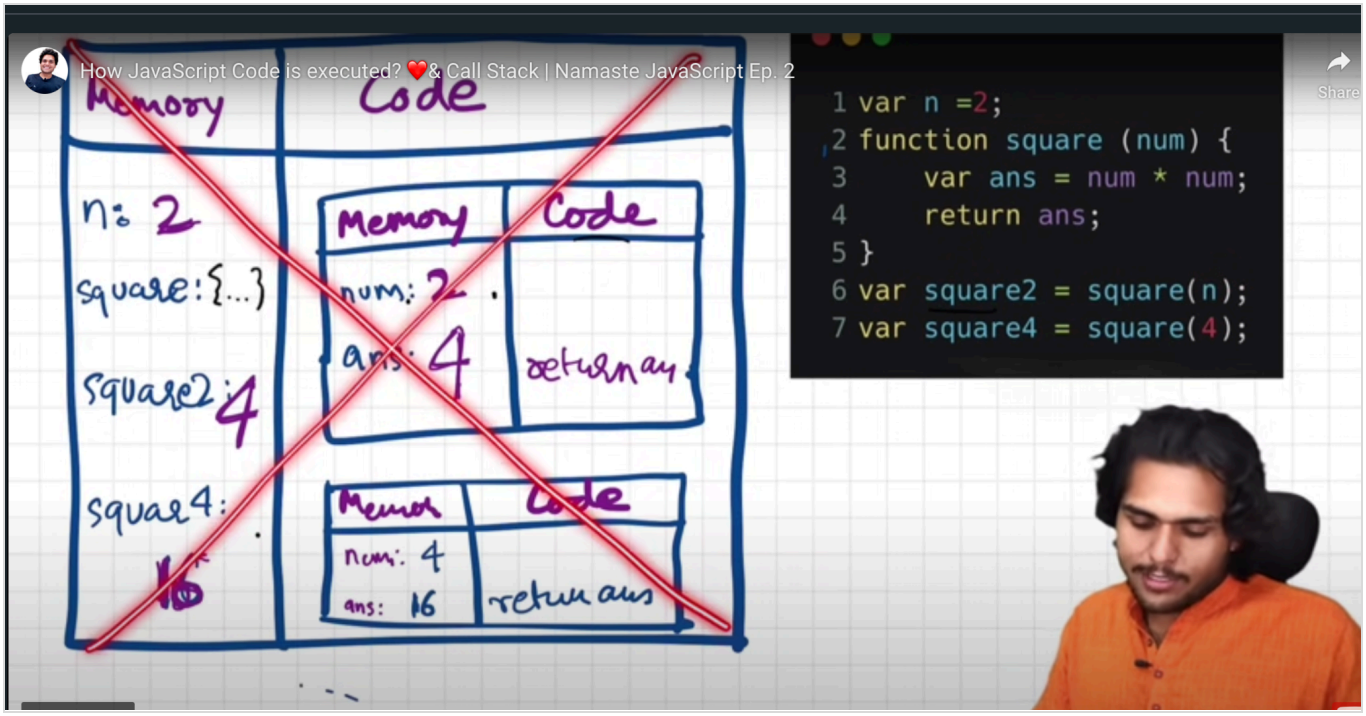


- Similarly in line#7 also will executed with brand new execution context & follow the same process

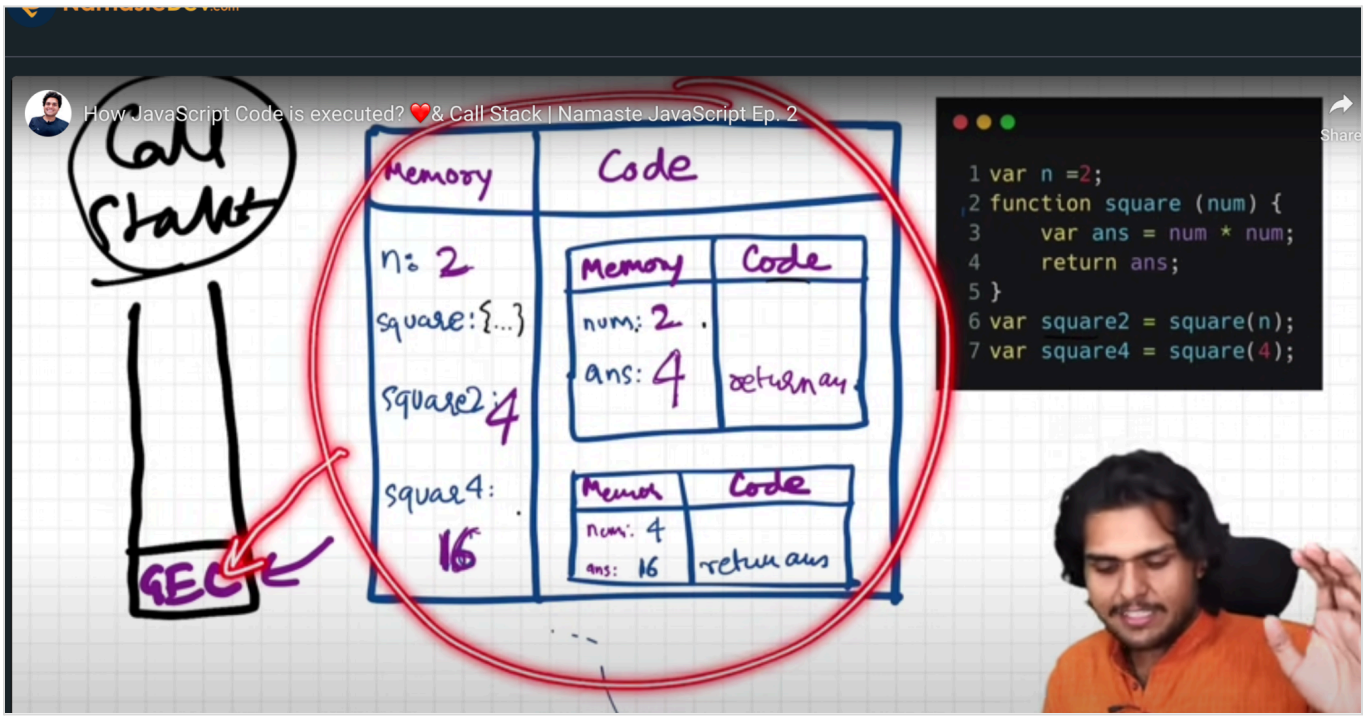




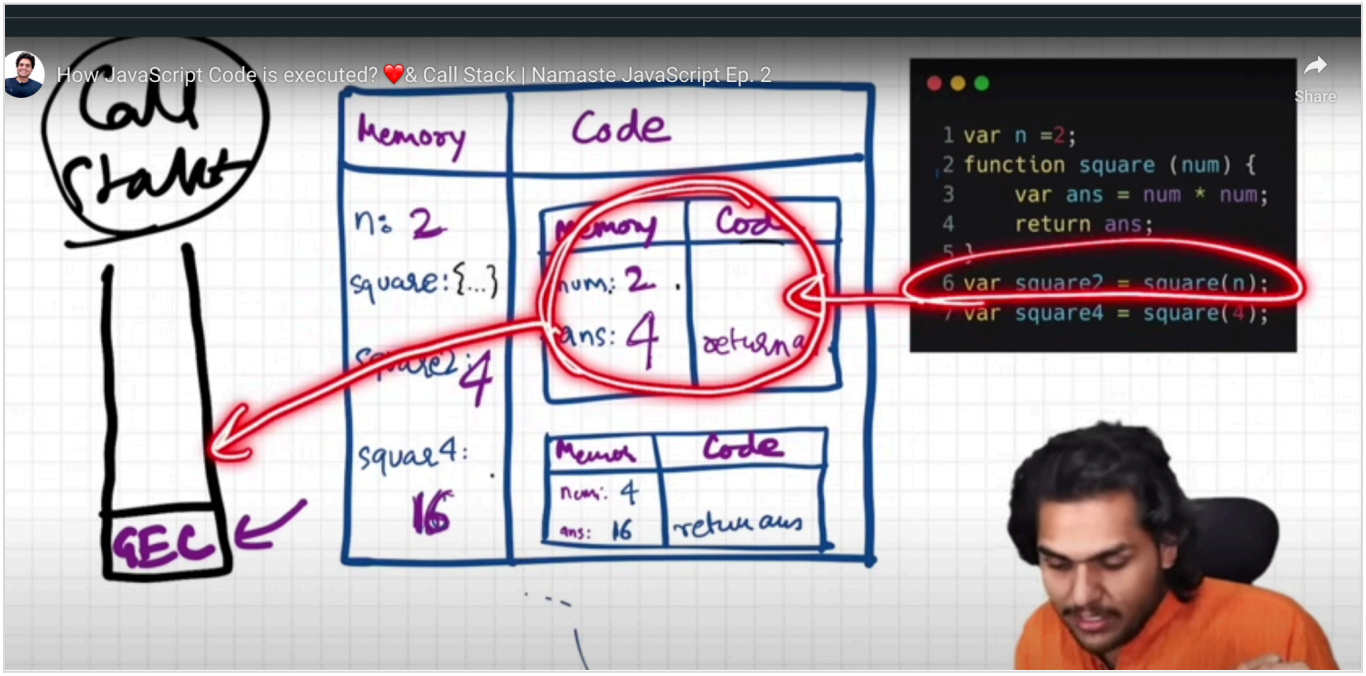
- **When whole JS code done with work / executed completely, global execution context will be removed**

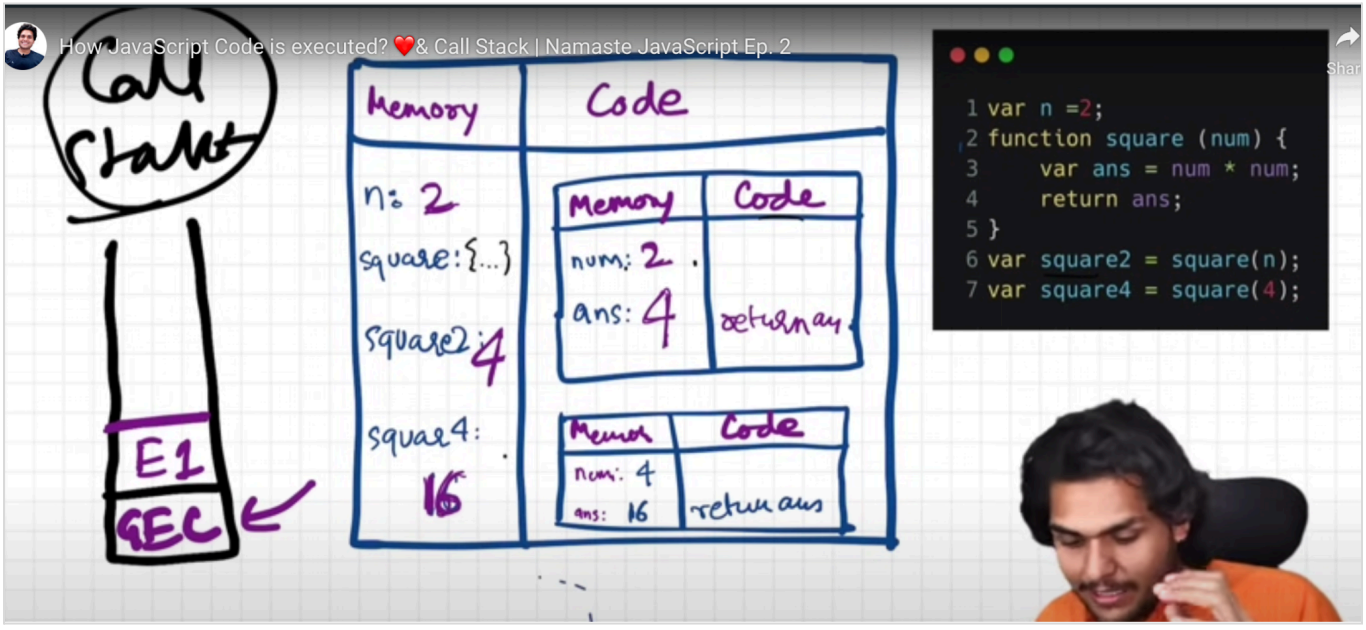


- **How does JS manage creation and deletion of Execution context when ever JS functions called ?**
  - **Manage with Call Stack, with data structure stack behavior**



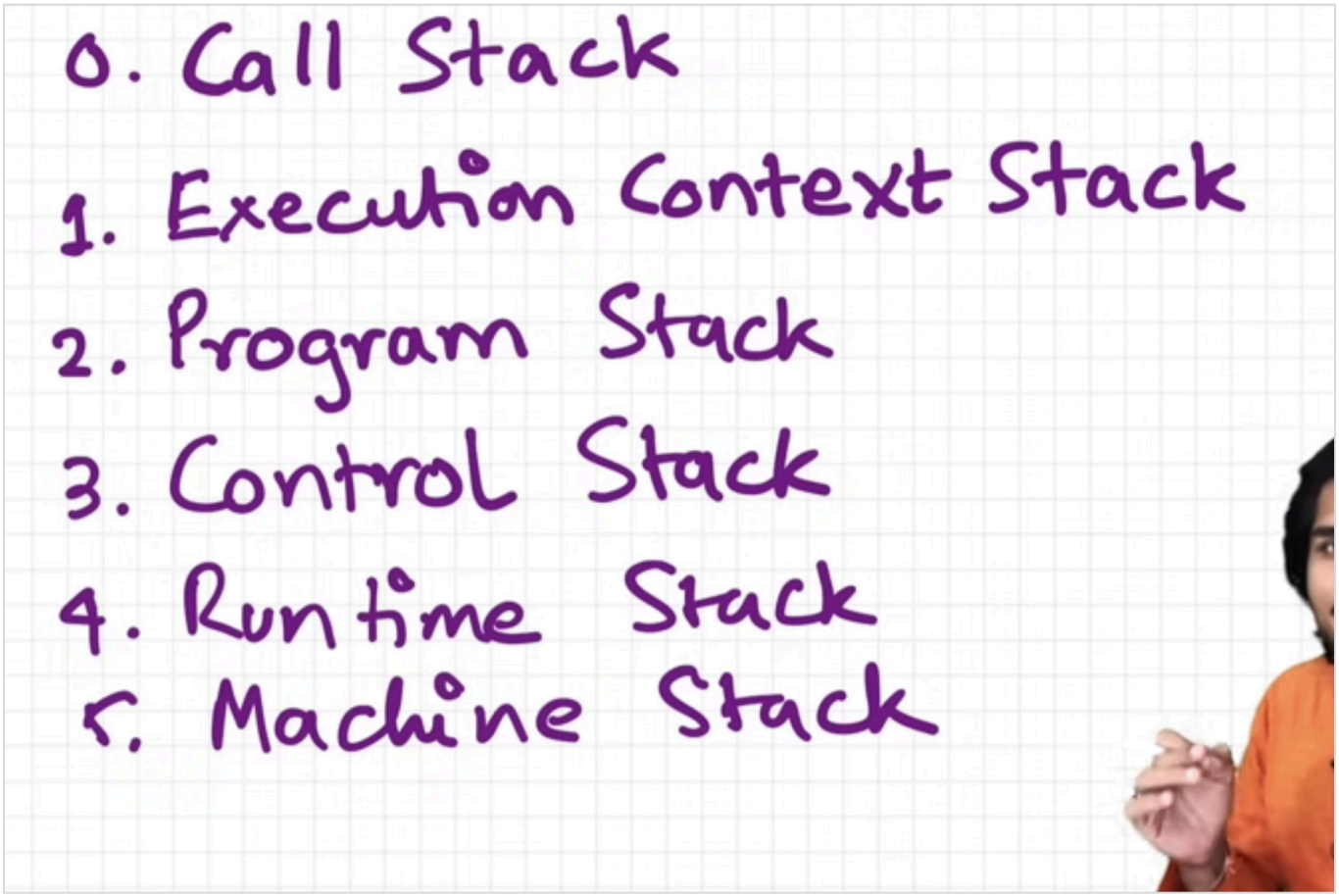
- **Always Global Execution Context will be there at bottom of Call Stack**
- **Whenever new function is called, branch new Execution Context will pushed into Call Stack**



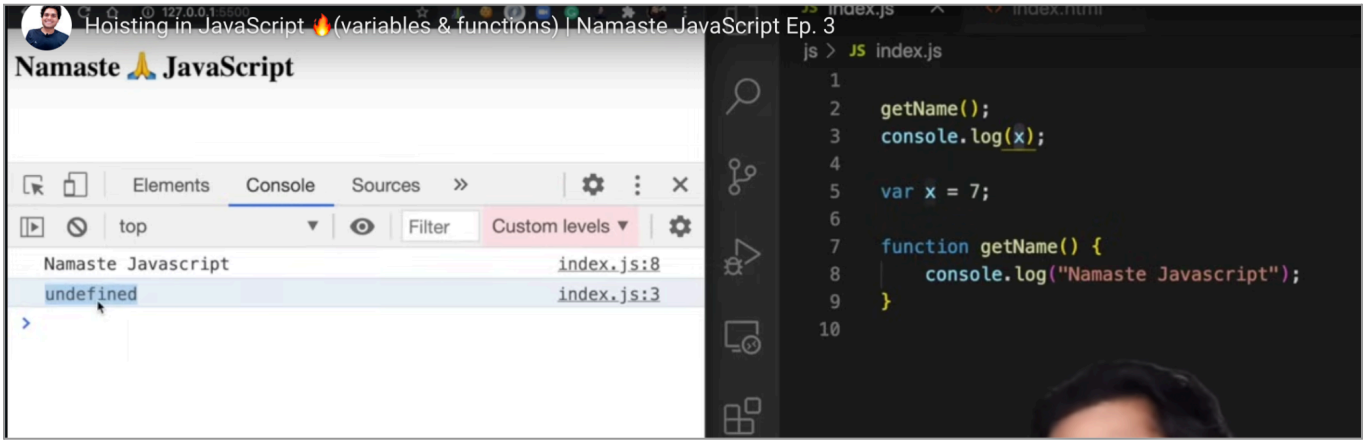


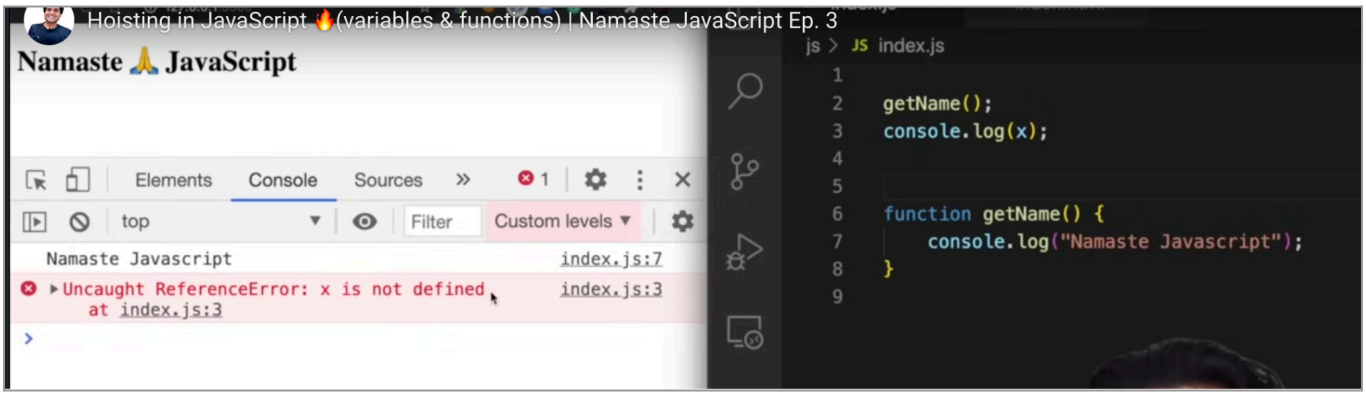
- **As soon as done with E1 pop from stack, control back to where it called in global execution context**
  - Similarly will execute with line #7 as well
- **As complete JS code execution done, Global EC also will removed & Call Stack becomes empty**

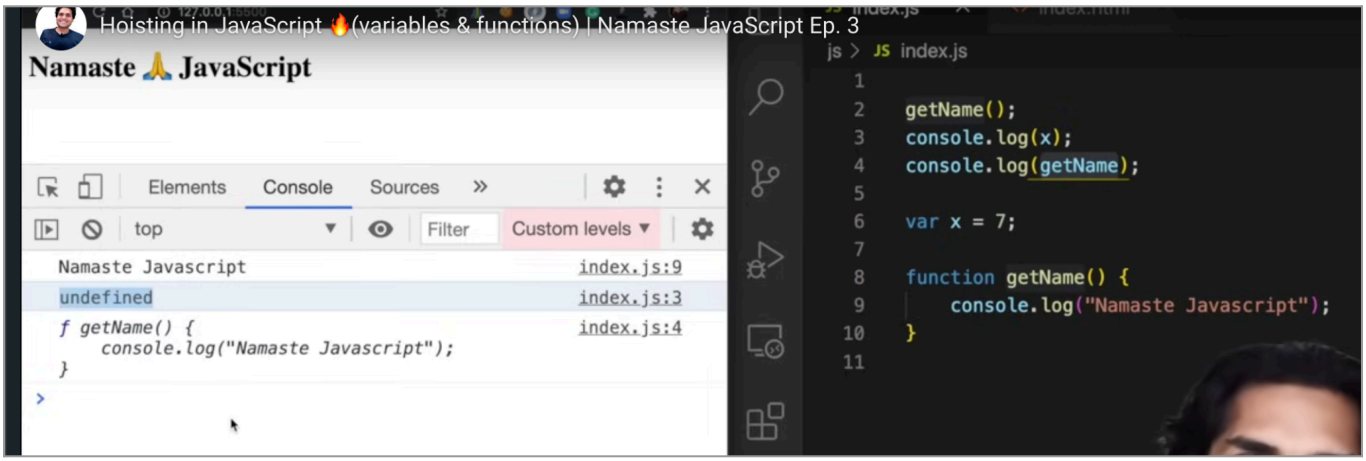
- Call Stack calling with different names



- Hoisting in JavaScript 🔥(variables & functions)
  - Answer in Memory creation phase & code execution phase
    - Magic of JS hoisting



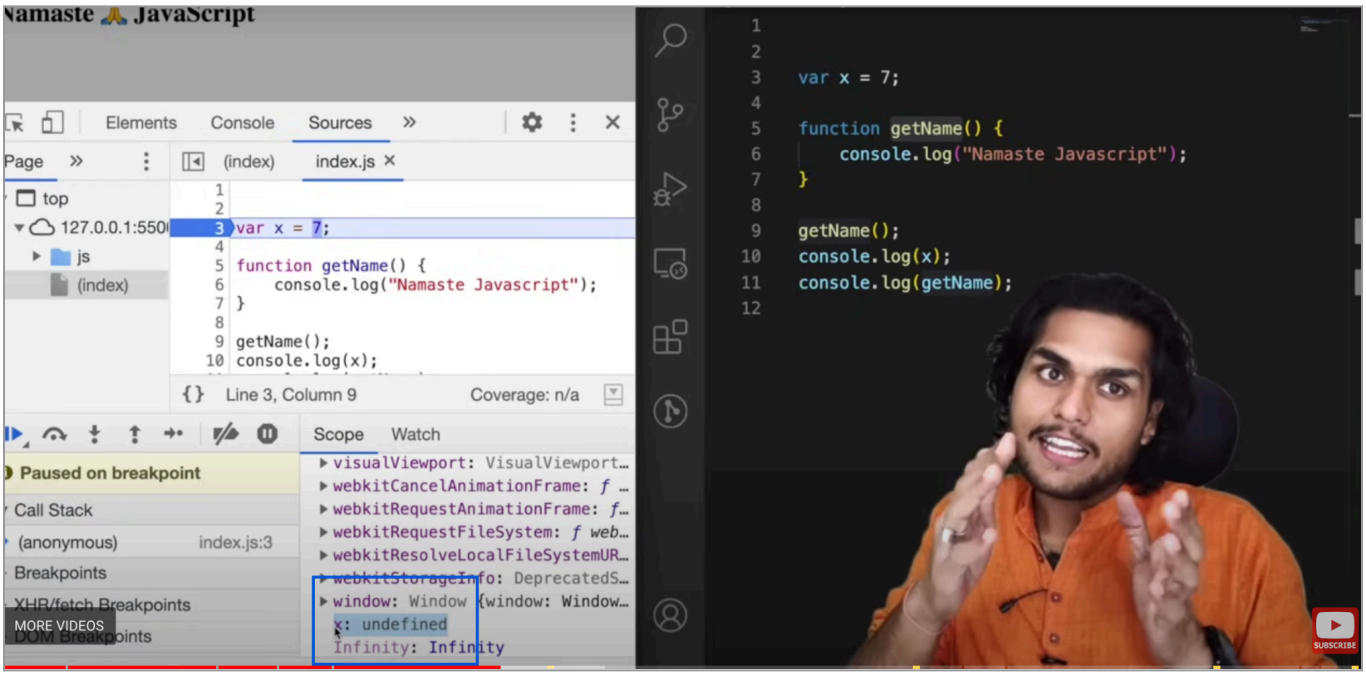


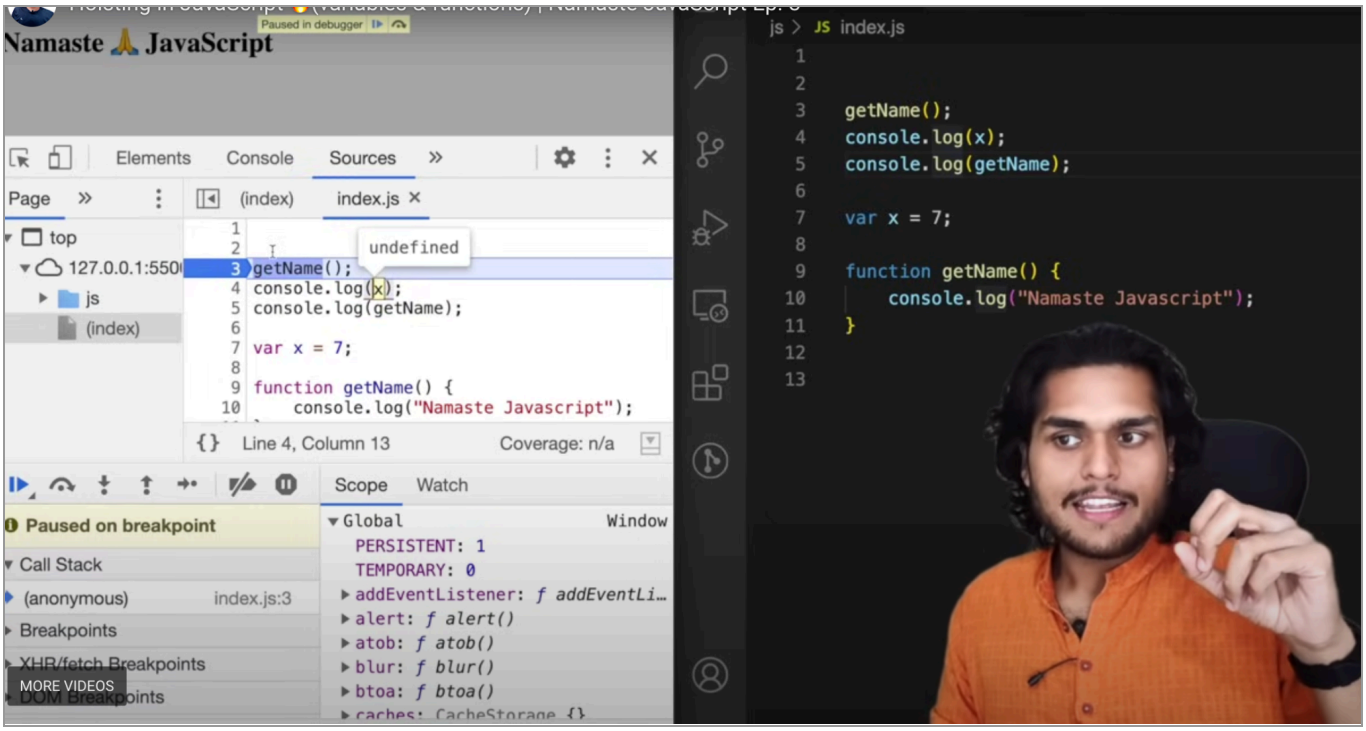


- Is it **undefined & not defined** both are same ? And: NO
  Both are different

- **JS Hoisting is behavoiur where access all JS variables & function even before initialized it, without any error**

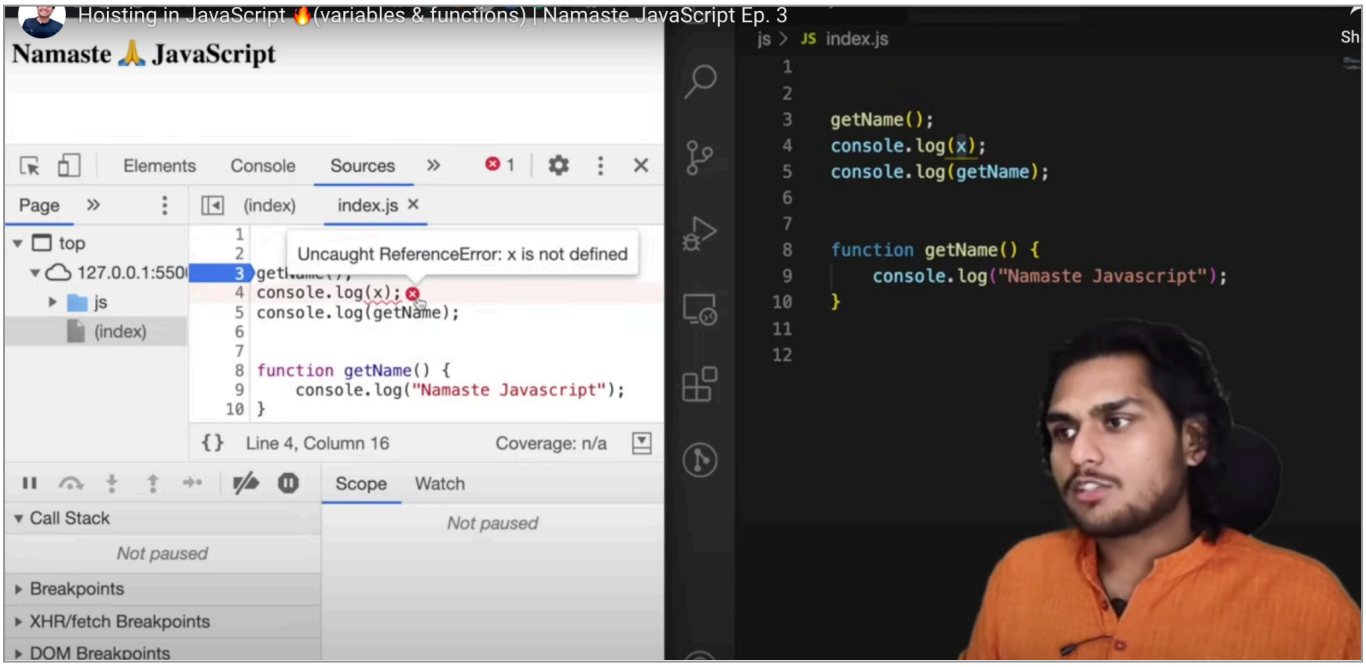
- Even before code starts executing, memory is allocating to all JS variables & function during memory creation phase



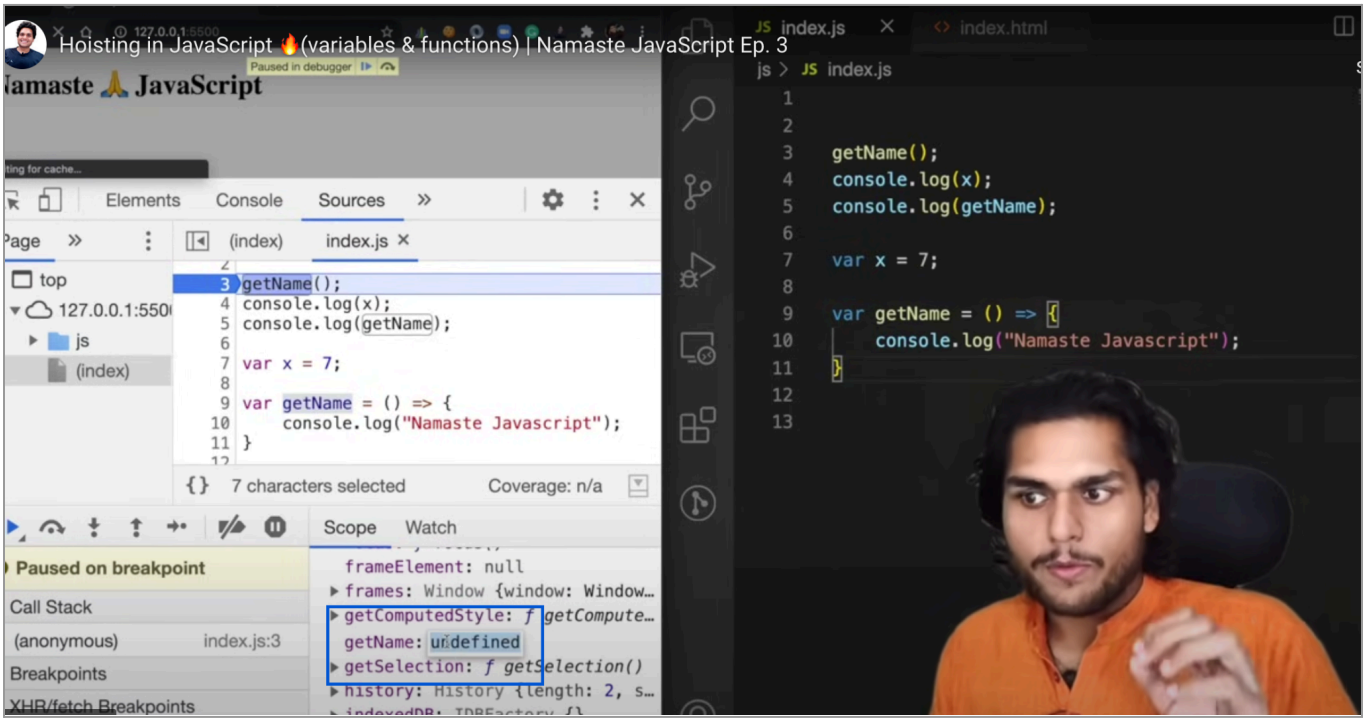


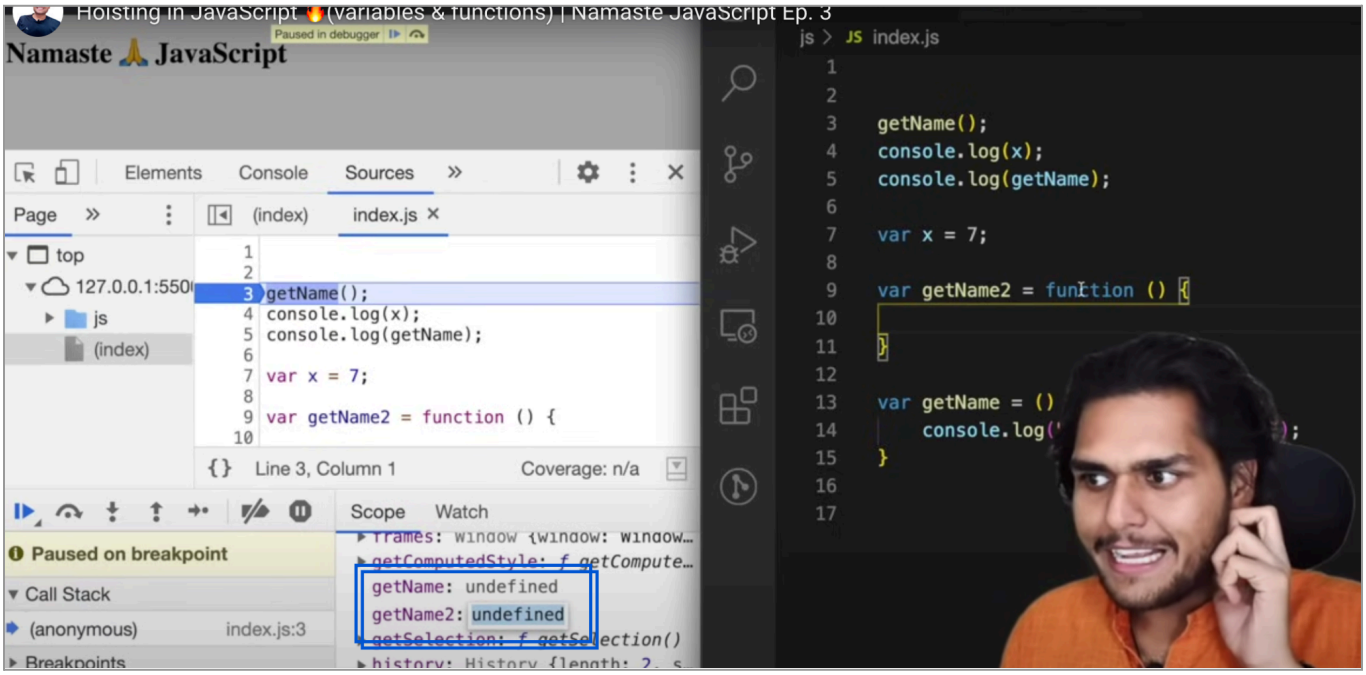
- **Not defined**

When memory is not allocated for variable x, since not defined, will get error

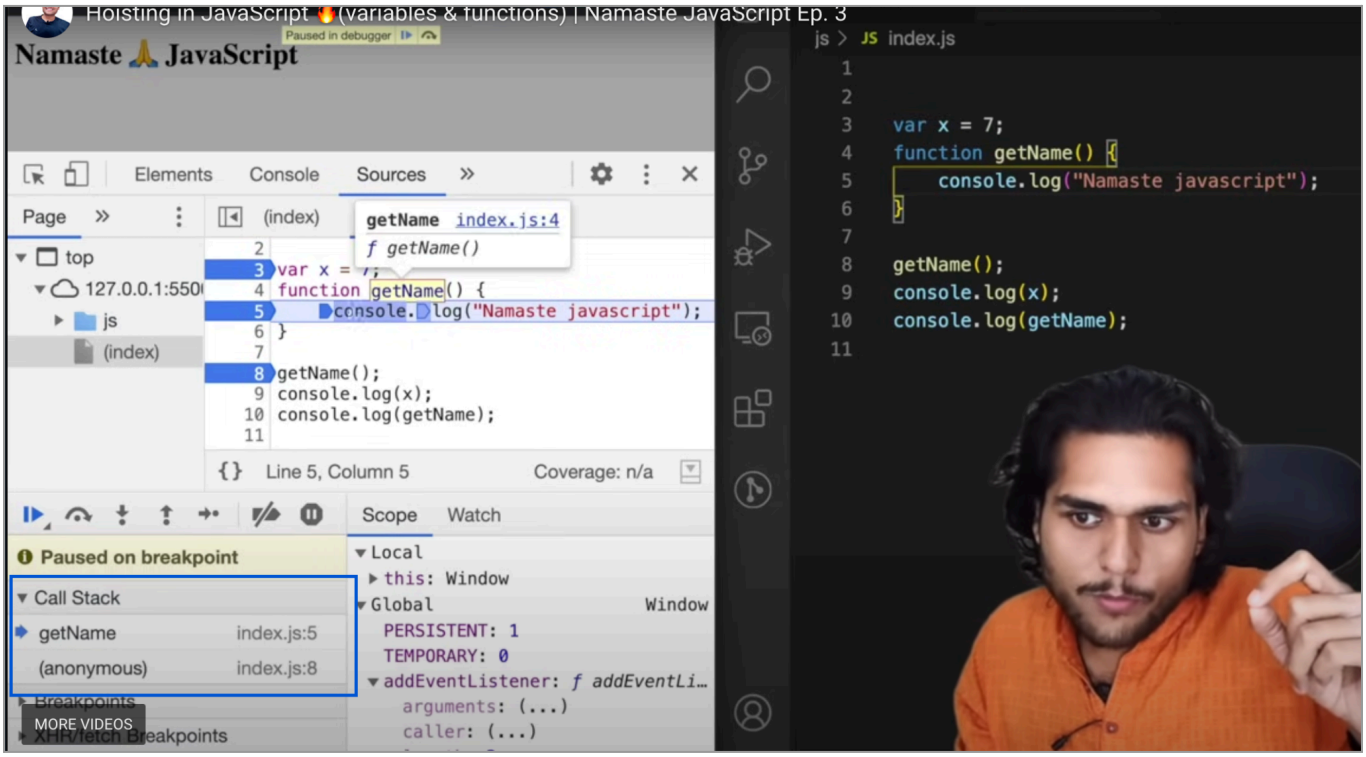


- If function defined with fat arrow function or variable access

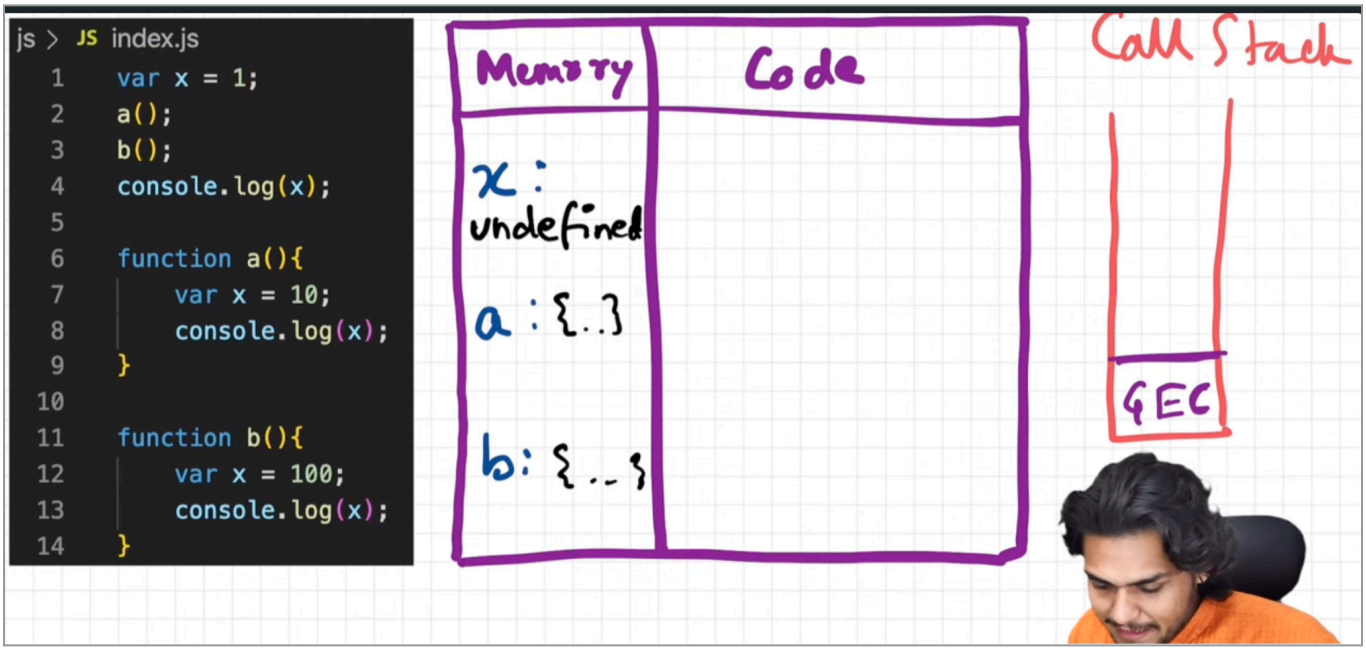


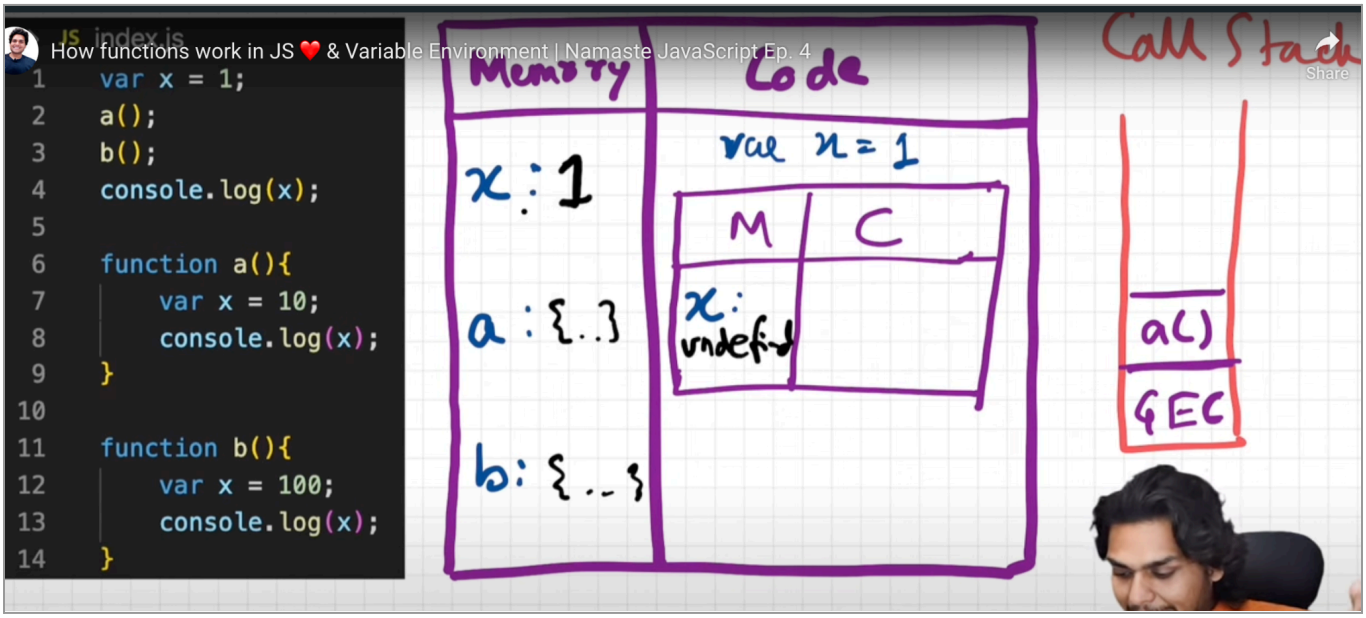


- **Call Stack in browser - maintains the order of Execution of Execution contexts**



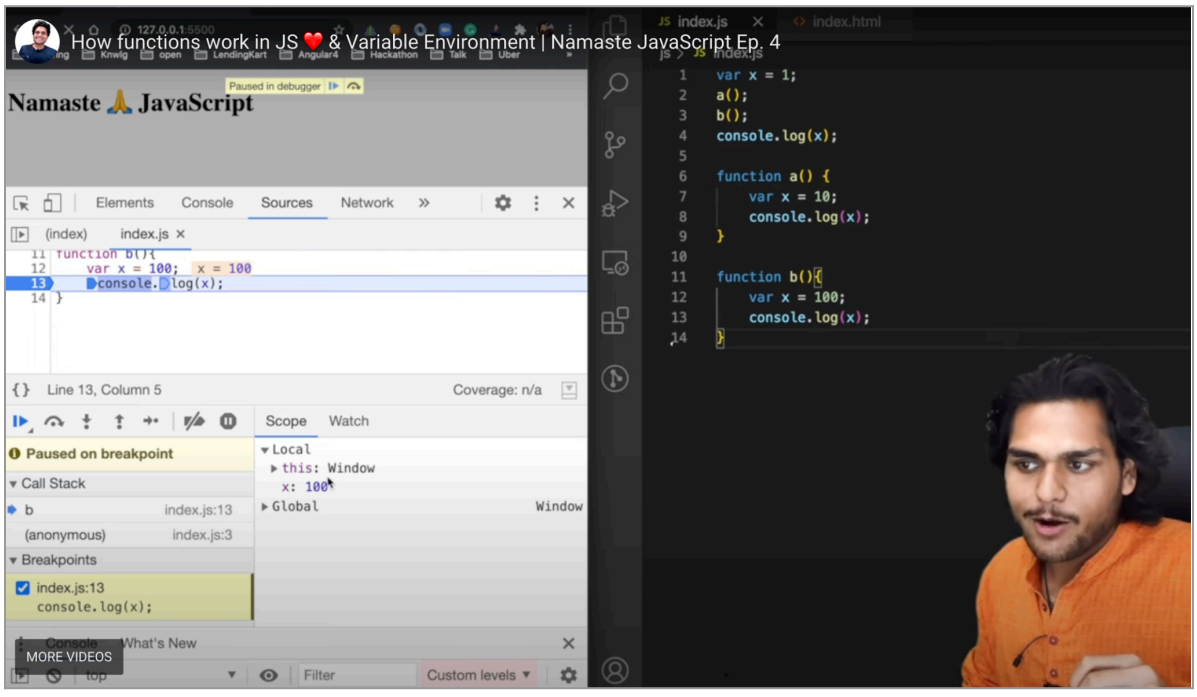
- How functions work in JS ❤️ & Variable Environment
  - Functions are heart of javascript, act as mini program
  - Global Execution context & functions





- While executing Each function has its own / separate memory space, which is independent

- By executing JS code, will understand
  - Call stack, where is the controls with line number
  - Separate memory space for function (local, global memory)
  - Memory creation & Code execution phase for Execution contexts

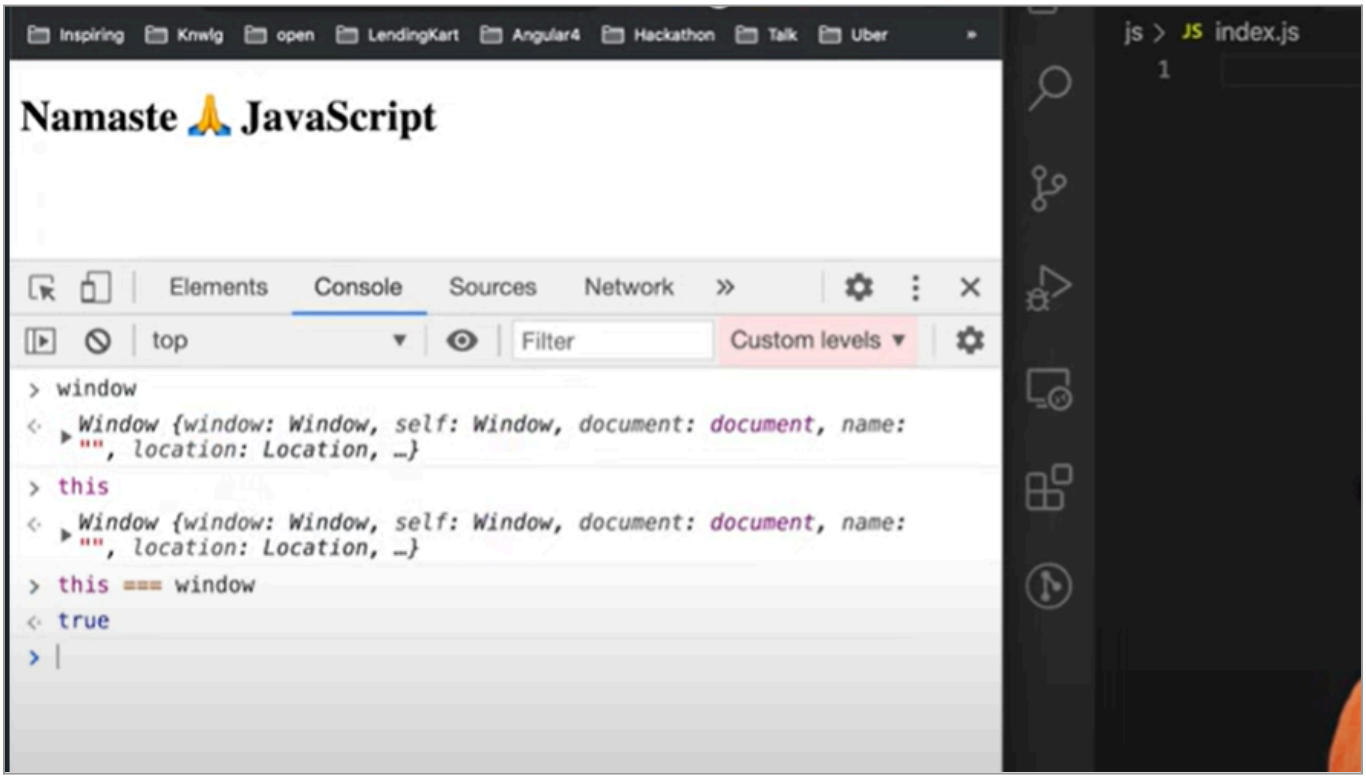


- SHORTEST JS Program 🔥window & this keyword
  - Shortest JS program is empty JS file
  - What happens when run Empty JS file in browser
    - Still Global Execution will create & global spaces
  - JS engine will create global window object

  - window is global object which created along with Global Execution Context when you run any JS code

  - All JS engines have responsibility to creates the window global object

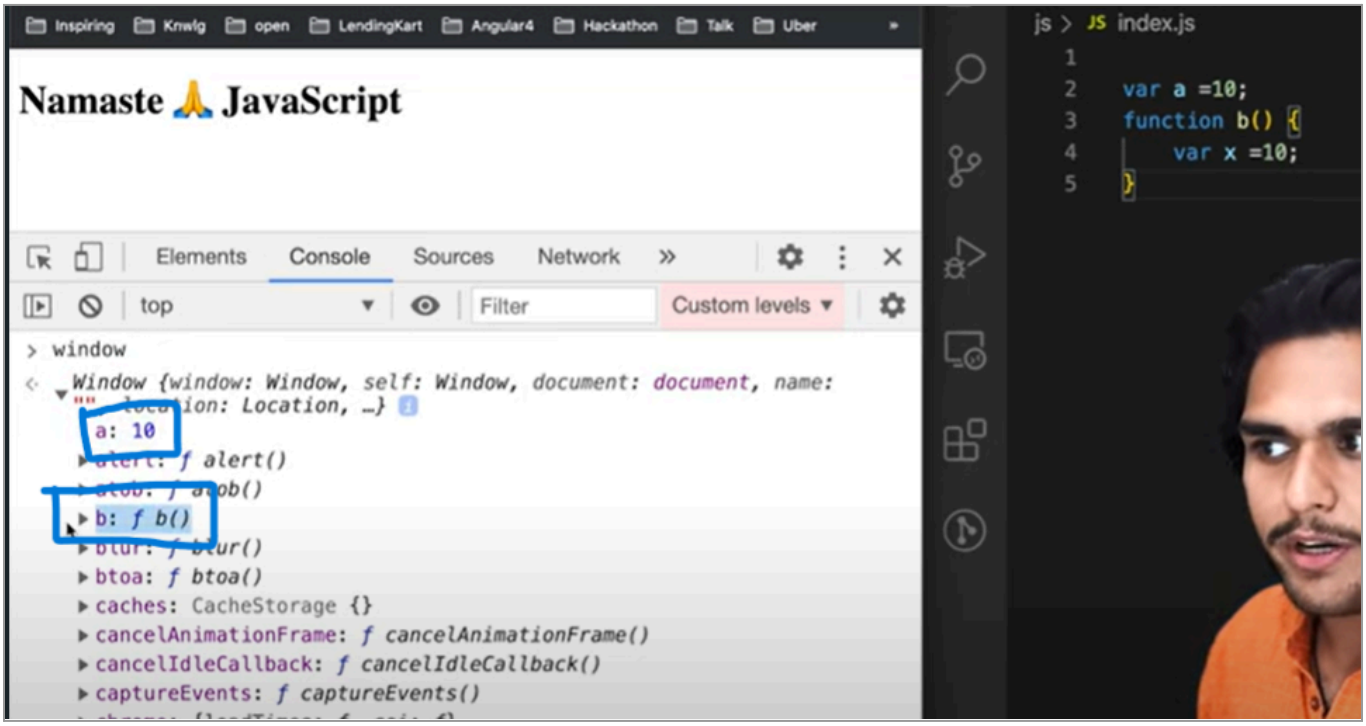
  - In global level, window is equals to this
    window === this ⇒ returns true

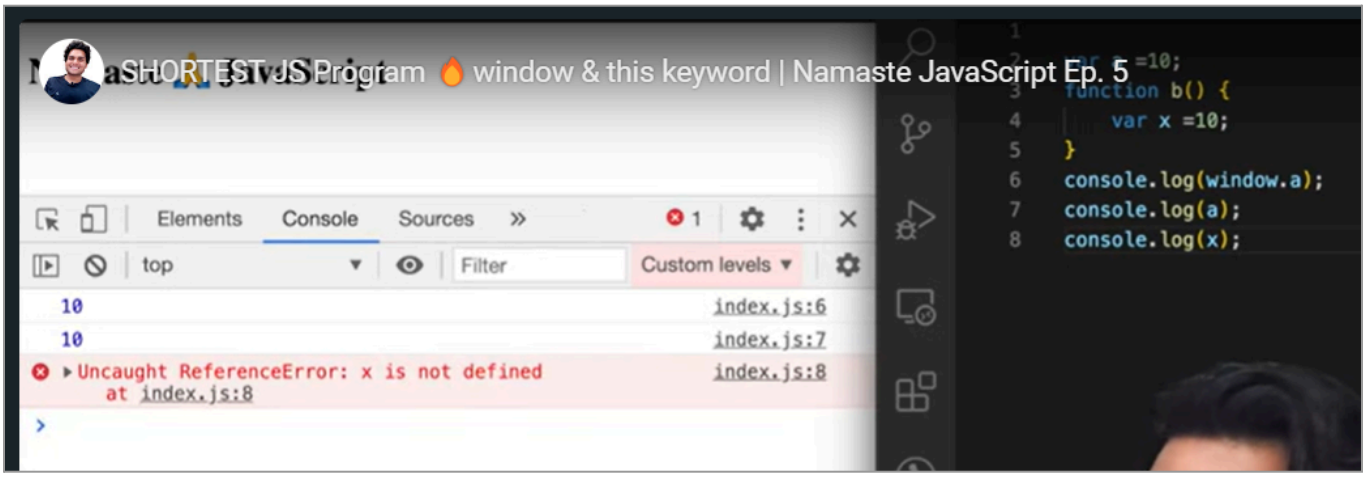


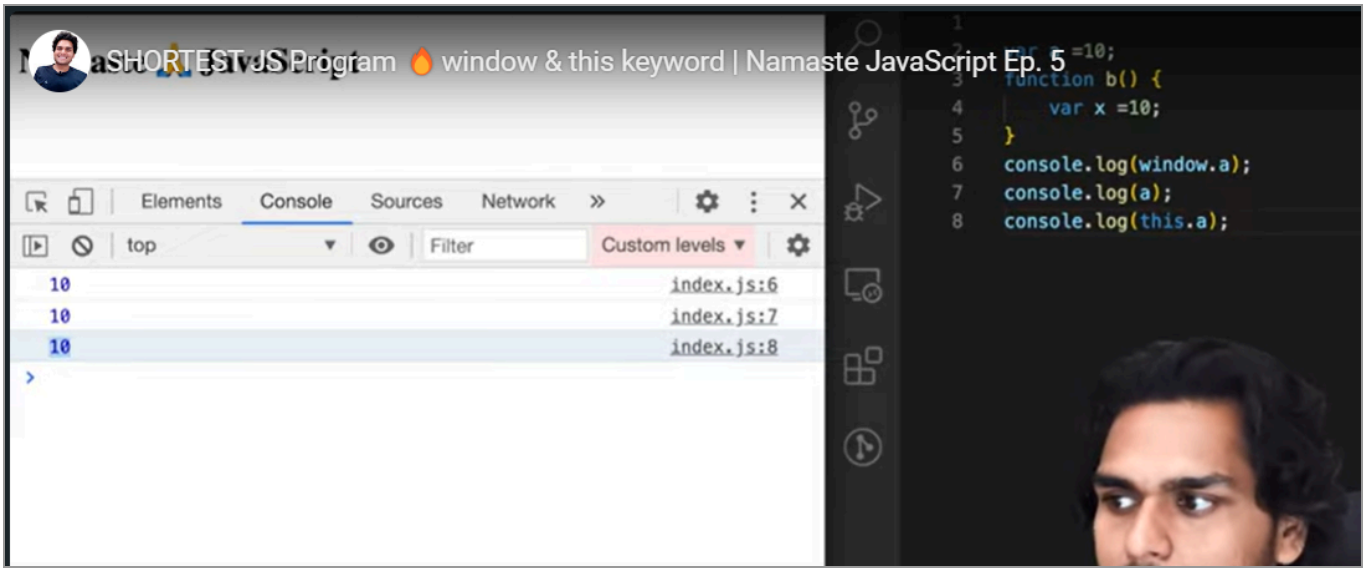
  - Whenever **global execution context created**, along with **window & this will create** and even **for function execution context this will create**

  - Global Space
    - Any code write in JS which is not inside the functions

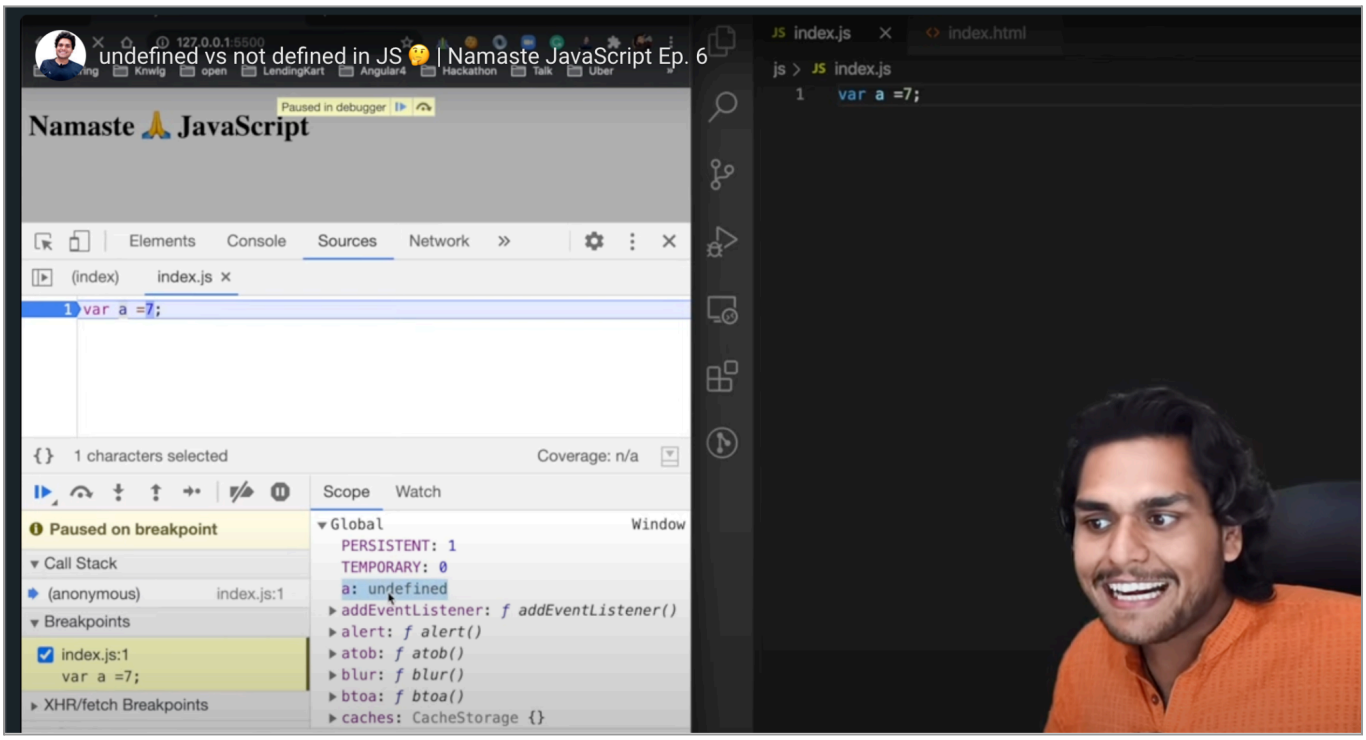
- All variables & function which is in Global spaces, are attached to global window / this object





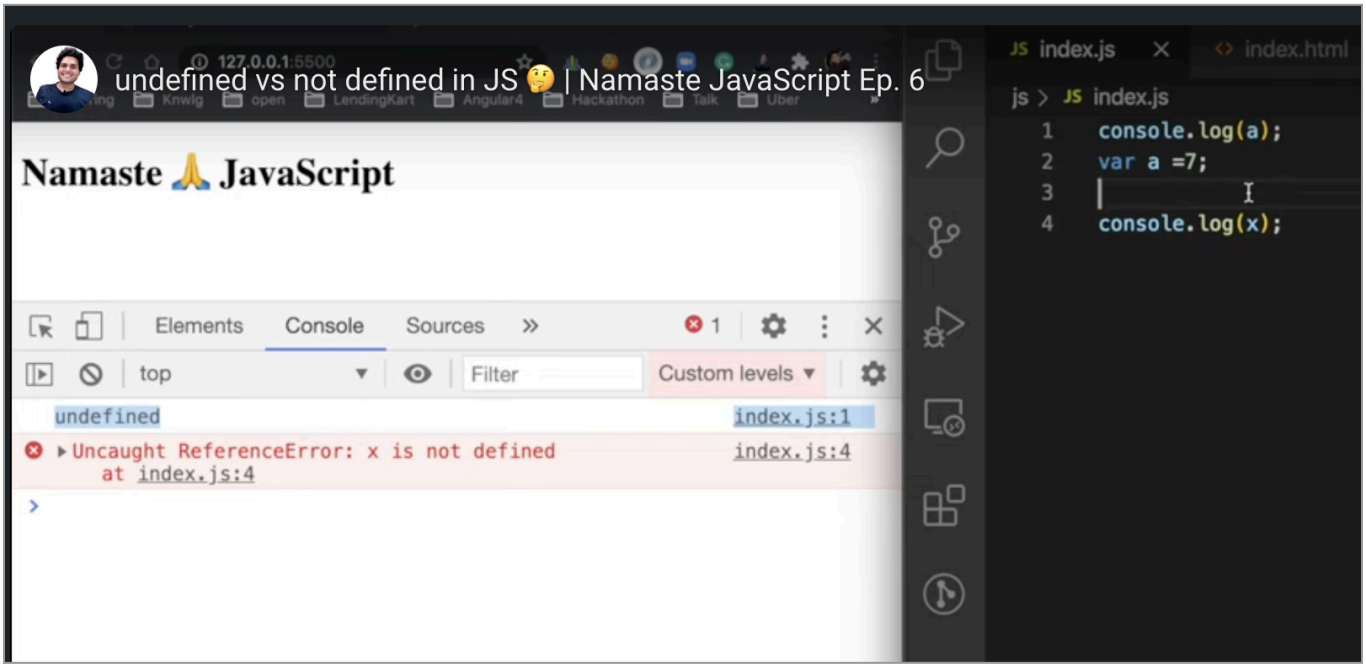


- undefined vs not defined in JS 🤔
  - Undefined is very special keyword in JS language
  - Undefined comes into picture during memory allocation phase, before starts executing the code

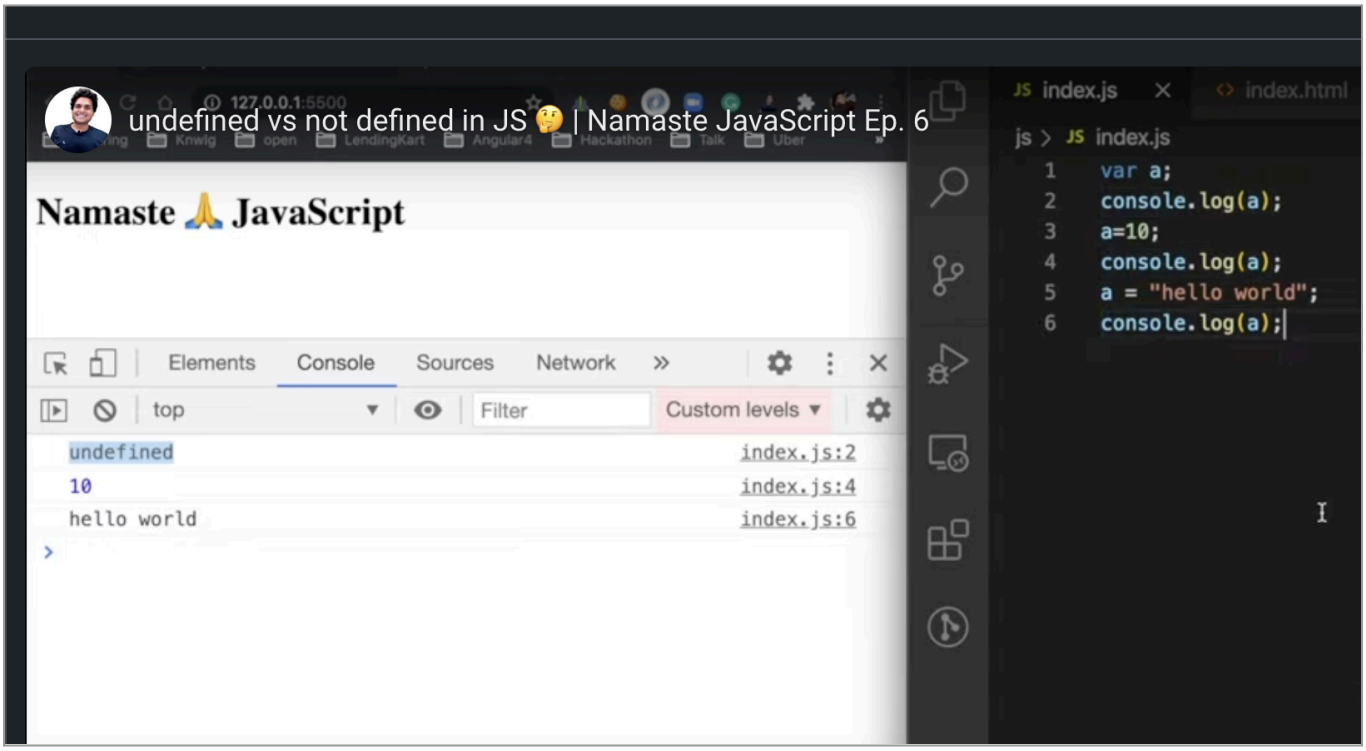


- **undefined is like placeholder placed in the memory** during JS variables memory allocation, until assign some values to JS variables

- “undefined” is completely different from “not defined”
  “not defined” will get which is not allocated memory



- JS is loosely typed / weakly typed language, since JS does not attach its variable to any specific data types
  JS is very flexible to changes its data types for variable, calling dynamically typed language



- Never do the mistake by assigning “undefined” to variable, not a good practice
  Like
  var a = undefined

- The Scope Chain, 🔥Scope & Lexical Environment
  - Sdfasdf
  - Dsfasd
  - Dsfas

- sdASF
- FDSsf
- Sfaf
- S
- Ff
- FDSAF