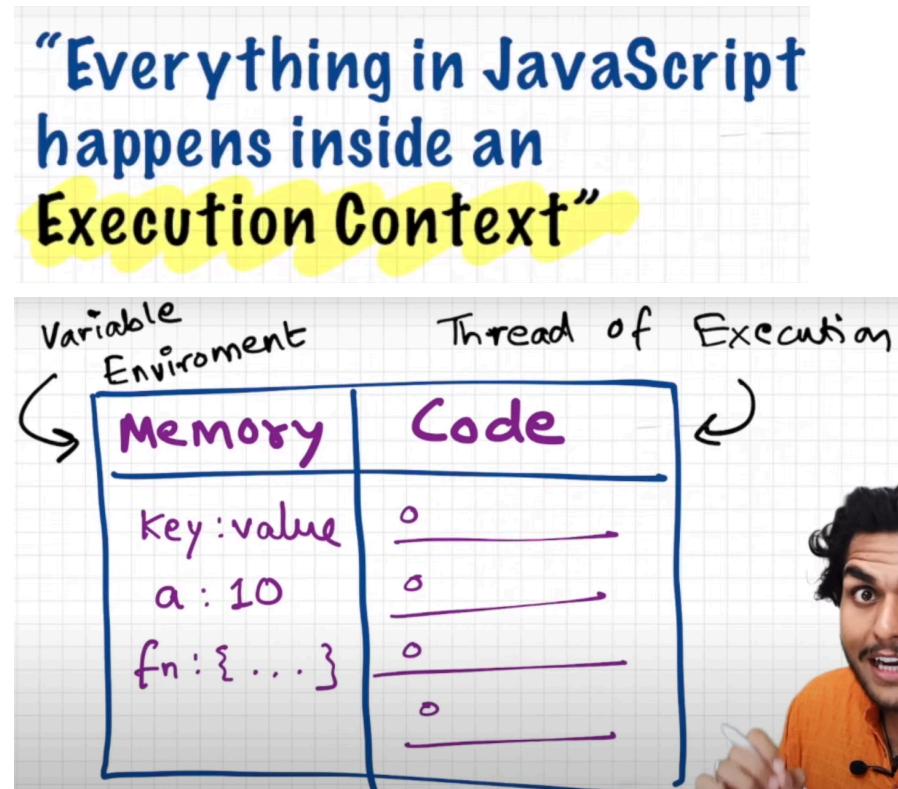


Season 1

Execution Context



Memory component is where variables and functions are saved as key value pairs and code component is the place where whole javascript code is executed

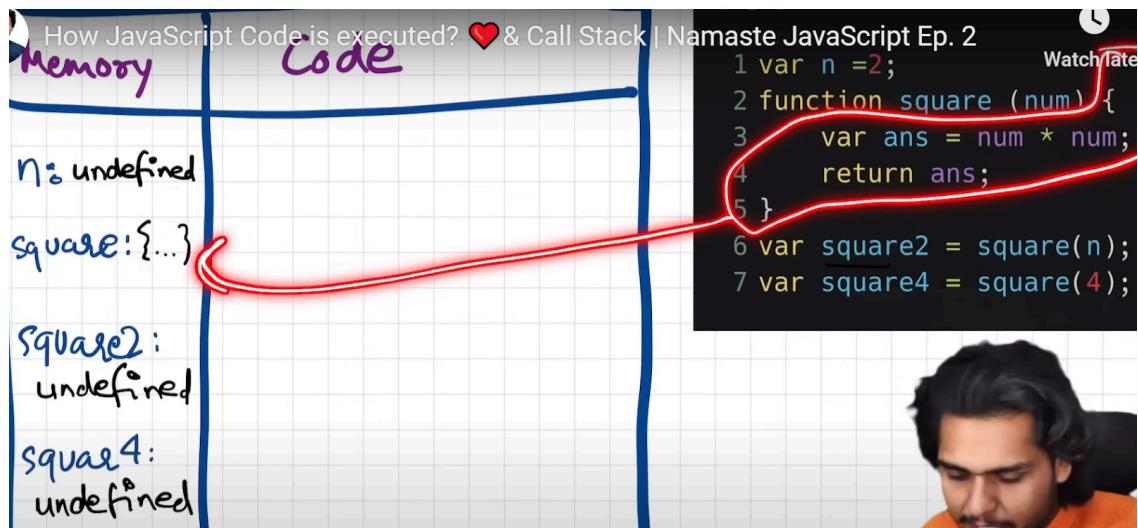
"JavaScript is a synchronous single-threaded language"

Single threaded - javascript can execute single command at a time

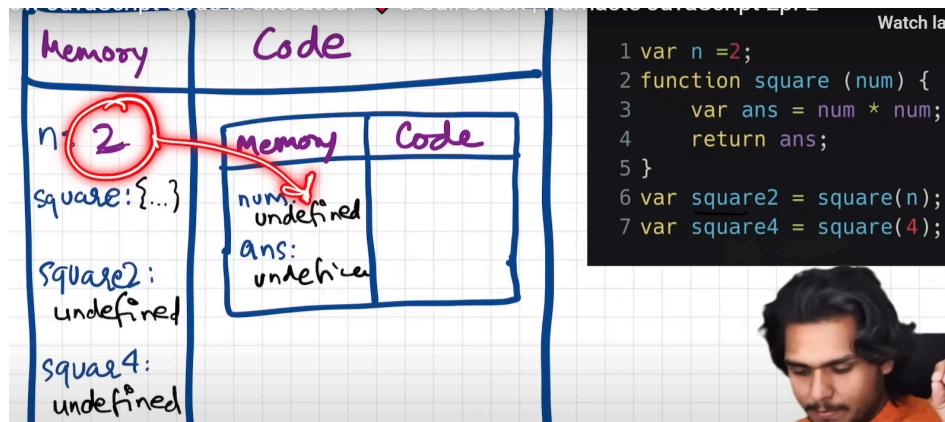
Synchronous - js can execute code only one command in a specific order. It will go to next line when current line is finished execution

Phase 1: Javascript engine goes through the entire code and allotted memory for variables and functions. For variables special value 'undefined' will get stored. For functions entire code will be stored and undefined value for the variables inside function

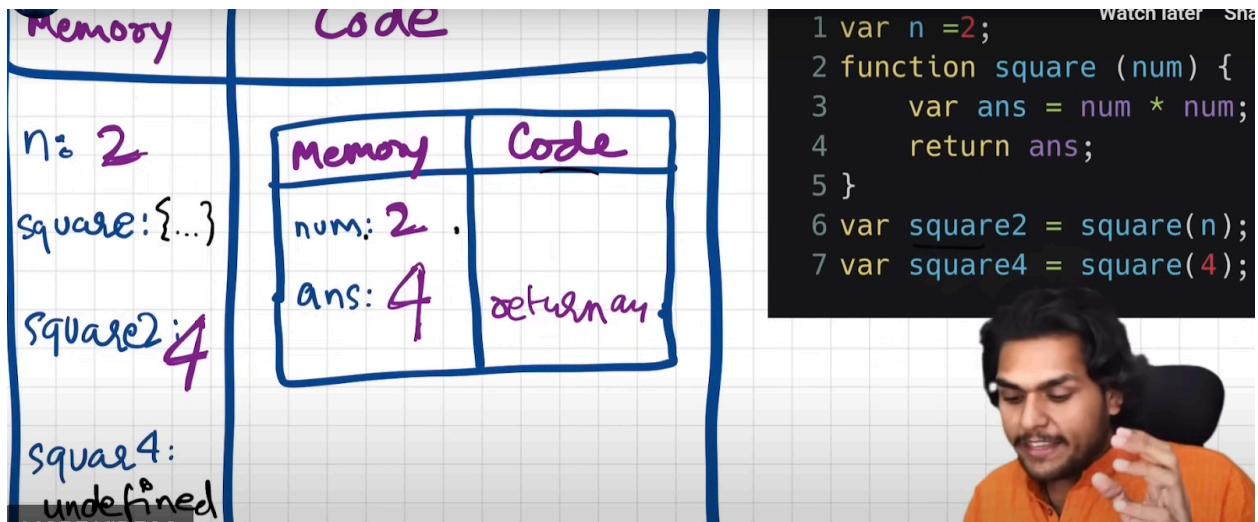
Phase 2 : execute the code line by line. In this step n = 2 function invocation function name () - square(). For functions new execution context will be created



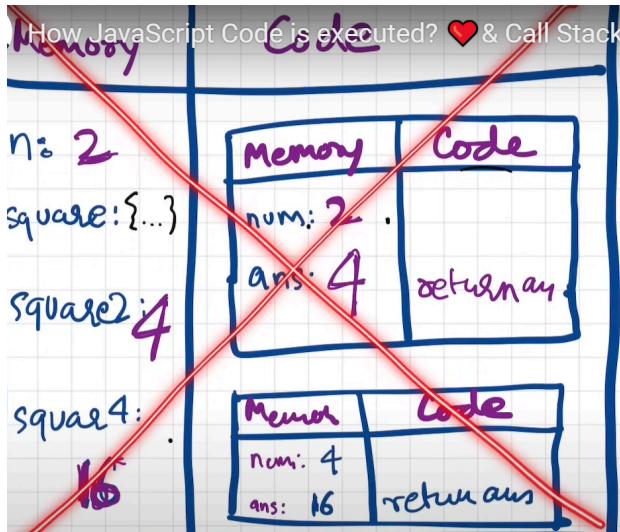
For functions, new memory context will be created



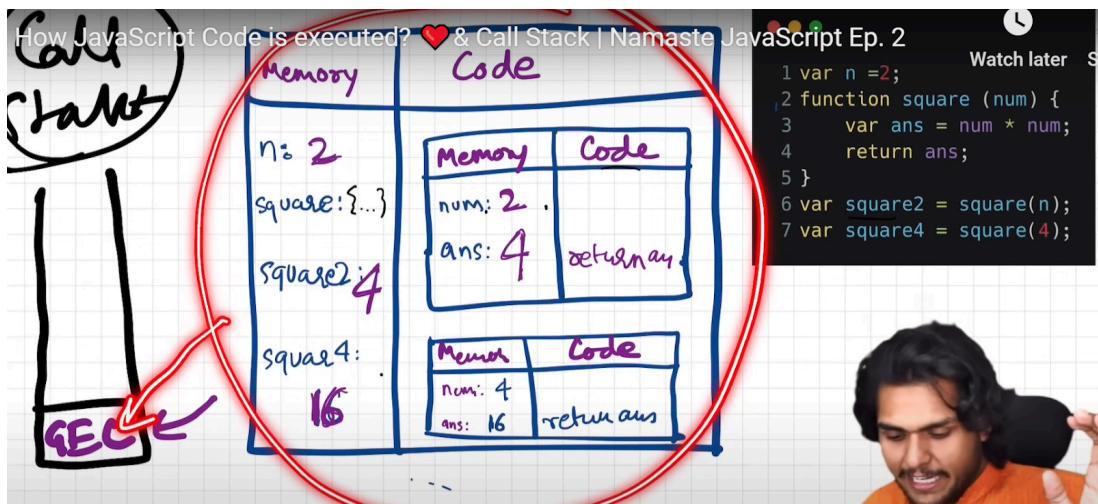
After the function execution once it completed the execution context will be deleted automatically



Once the entire code gets executed, Entire global context will be deleted



Call Stack



In call stack, GEC global execution context will get created as a first step. Then E1 execution context for function `square(n)` will go next to the call stack. After it E1 got executed it will popped out from stack and the control moved to GEC where it left for example to line# 6 here



entire execution context creation and deletion is managed by callstack in javascript

“Call stack maintains the order of execution of execution contexts”

ow JavaScript Code is executed? ❤ & Call Stack | Namaste JavaScript

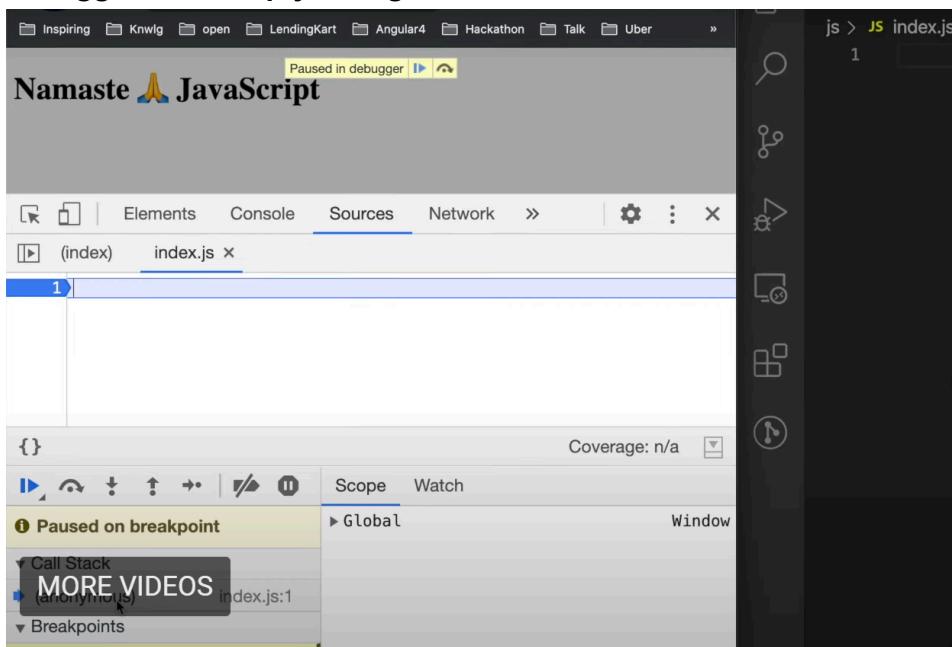
- Call Stack
- Execution Context Stack
- Program Stack
- Control Stack
- Runtime Stack
- Machine Stack

Shortest Javascript Program

Shortest Javascript program is just an empty file



Debugger in an empty file - global execution context is created



Javascript engine will create a window. You can see it in console

We didn't put window inside our code. JS engine creates on its own



A screenshot of a browser's developer tools console. The tabs at the top are Elements, Console, Sources, and Network. The Console tab is selected. Below the tabs, there are icons for back, forward, and search, followed by the text "top". A "Filter" input field is present. To the right, there is a "Custom levels" dropdown and other settings. The console output shows:

```
> window
<  ▶ Window {window: Window, self: Window, document: document, name:
  "", location: Location, ...}
```

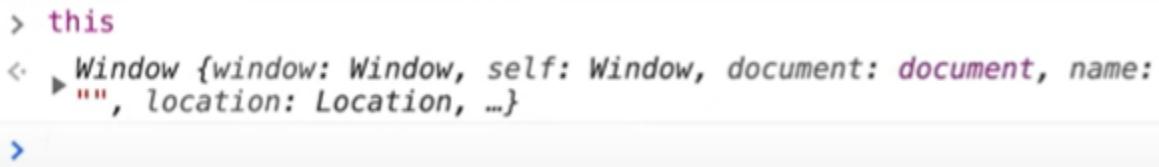
The variables and functions are created by JS engine and put in the global execution context



A screenshot of a browser's developer tools console. The "window" object is expanded to show its properties. The properties listed are:

- alert: function alert()
- atob: function atob()
- blur: function blur()
- btoa: function btoa()
- caches: CacheStorage {}
- cancelAnimationFrame: function cancelAnimationFrame()
- cancelIdleCallback: function cancelIdleCallback()
- captureEvents: function captureEvents()
- chrome: {loadTimes: function loadTimes(), os: function os()}

JS engine will also create 'this' keyword



A screenshot of a browser's developer tools console. The "this" keyword is typed, and the dropdown menu shows the "window" object as a suggestion.

```
> this
<  ▶ Window {window: Window, self: Window, document: document, name:
  "", location: Location, ...}
```