***Everything in JavaScript happens inside an Execution Context***.

**Execution Context**

**Code | Thread of Execution**

**Memory | Variable Environment**

Here all the variable and function are stored as a key value pair.

Here whole code gets executed one line at a time.

***JavaScript is a synchronous single-threaded language.***

*As soon as we run JavaScript code in* ***browser****.* JS engine creates a

1. **Global Execution Context**
2. **Global Object (Window Object)**
3. **this variable**.

Even for empty JS file.

At Global level this will point to window object (***this === window # true***). And every variable gets attach to window object means global space.

var a = 10;

console.log(window.a); // 10

console.log(this.a); // 10

console.log(a); as it automatically assumes it referring to global space (window.a). // 10

This Execution Context is created into **two phases**.

1. **Memory Creation phase**

JS engine will skim through the whole JS code and store variable and function in key: value pair in Memory.

i.e., Memory is allocated to all variable (only var, not let & const) & functions even before code start executing.

*Before code execution Phase/ while memory creation phase*  
{

Variable: undefined, // **undefined** *allocated memory to the variable but value is not yet set so* **not defined** *not allocated memory to the variable.*

Function: { … } (stores the whole function code)

}

Where, var x = () => { }**[Arrow Function ]** or var y = function { }**[Function Expression]** are stored as undefined. i.e., {x: undefined, y: undefined}

1. **Code Execution phase**

*While Code Execution* line by line the values of variables get change and replaced with actual value.

Every time a new function is found a new **Local Execution Context** is created inside Global Execution context and Execution context order is maintained by **Call Stack**. As soon as return is observed the Local execution context get destroyed and popped out from call stack.

After whole code is executed Global execution context also get destroy.

***Call stack is used to maintains the order of execution of Execution Contexts.***

Call Stack | Execution Context stack | program stack | control stack | runtime stack | machine stack

***JavaScript is loosely typed language.***

It does not attach any specific type of data type to the variables.

var a; // undefined

a = 10; // number

a = ‘hi’; // string

a = false; // Boolean

a = undefined; // bad practice