Java script Notes

dfsd

Dddfd dfsd

* 1
* 1.1
* “Everything in javascript is happens inside an execution Context”

**\*What is an Execution Context?**

Simply put, an execution context is an abstract concept of an environment where the Javascript code is evaluated and executed. Whenever any code is run in JavaScript, it’s run inside an execution context.

**\*Types of Execution Context**

There are three types of execution context in JavaScript.

**Global Execution Context** — This is the default or base execution context. The code that is not inside any function is in the global execution context. It performs two things: it creates a global object which is a window object (in the case of browsers) and sets the value of this to equal to the global object. There can only be one global execution context in a program.

**Functional Execution Context** — Every time a function is invoked, a brand new execution context is created for that function. Each function has its own execution context, but it’s created when the function is invoked or called. There can be any number of function execution contexts. Whenever a new execution context is created.

**Eval Function Execution Context** — Code executed inside an eval function also gets its own execution context, but as eval isn’t usually used by JavaScript developers.

**\* Execution context stack**

**Also Know as**

**>run stack**

**>calling stack**

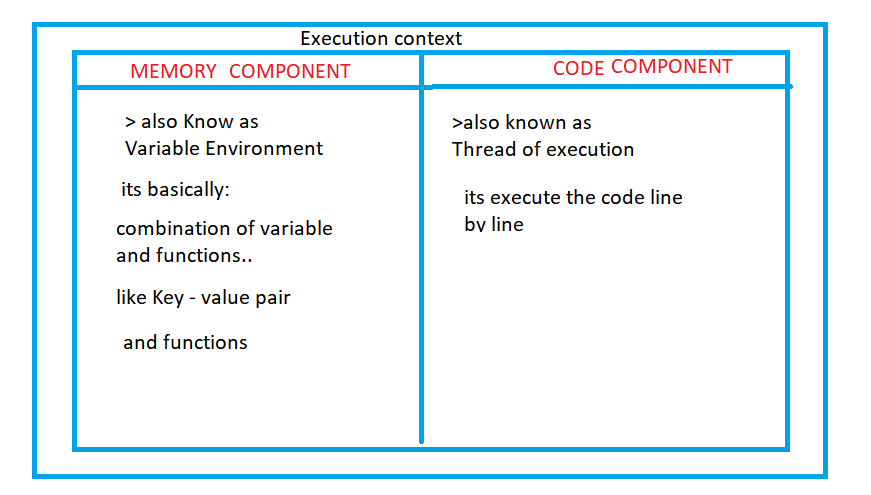
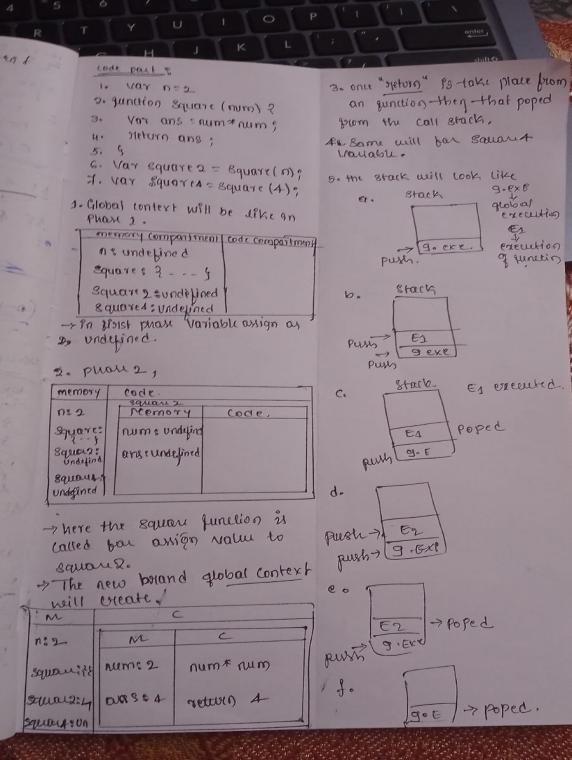
**>program stack**

**>control stack**

**>machine stack**

Execution stack, also known as “calling stack” in other programming languages, is a stack with a LIFO (Last in, First out) structure, which is used to store all the execution context created during the code execution.

When the JavaScript engine first encounters your script, it creates a global execution context and pushes it to the current execution stack. Whenever the engine finds a function invocation, it creates a new execution context for that function and pushes it to the top of the stack.

The engine executes the function whose execution context is at the top of the stack. When this function completes, its execution stack is popped off from the stack, and the control reaches to the context below it in the current stack. 

# **Execution Phase:**

* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10