JavaScript Execution Context

- How JS Works Behind The Scenes

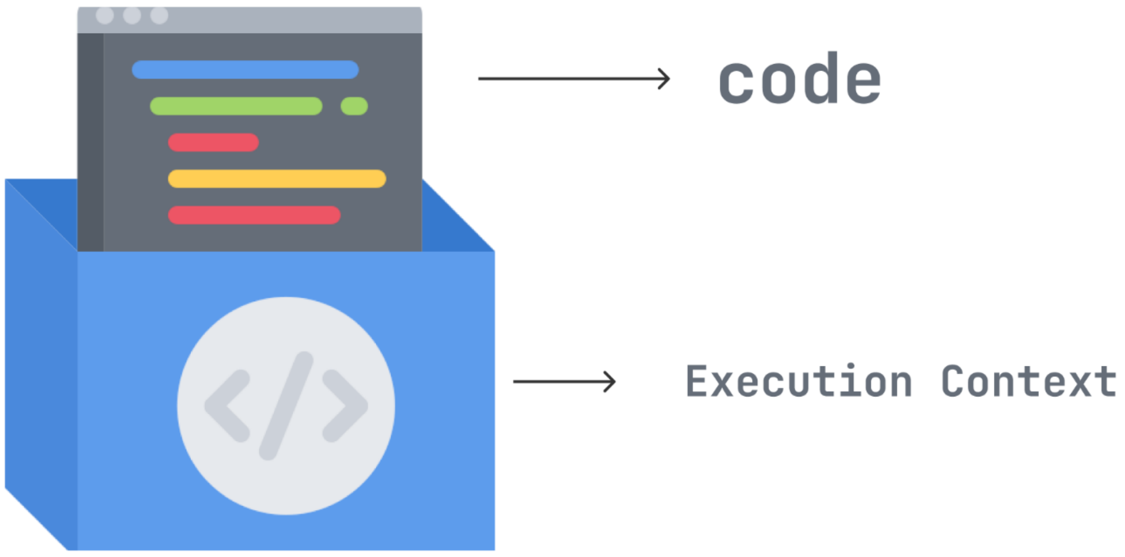
All JavaScript code needs to be hosted and run in some kind of environment. In most cases, that environment would be a web browser.

For any piece of JavaScript code to be executed in a web browser, a lot of processes take place behind the scenes. In this article, we'll take a look at everything that happens behind the scenes for JavaScript code to run in a web browser.

The browser's JavaScript engine then creates a special environment to handle the transformation and execution of this JavaScript code. This environment is known as the `Execution Context`.

The Execution Context contains the code that's currently running, and everything that aids in its execution.

During the Execution Context run-time, the specific code gets parsed by a parser, the variables and functions are stored in memory, executable byte-code gets generated, and the code gets executed.



Imagine your code is placed inside a box whenever you write JS code. That box is the Execution context, a conceptual environment created by JS for code evaluation and execution.

There are three types of Execution Context in JS.

The Global Execution Context (GEC): This is the primary or base Execution Context. It is the highest level of abstraction for an execution context in JS. It has two main functions: (1) Create a global object, and (2) Attach the `this` value to the global object.

The Function Execution Context (FEC): An execution context is created for every function invocation. It is not created when the function is declared, only when called or invoked. When a function is called, it is executed through the execution stack (which we'll discuss below).

The `eval()` Execution Context: Due to the malicious nature of eval(), it is rarely used in JS so we won't discuss it. However, an execution context is created whenever it is used.

To keep track of all the execution contexts, including the global execution context and function execution contexts, the JavaScript engine uses the call stack

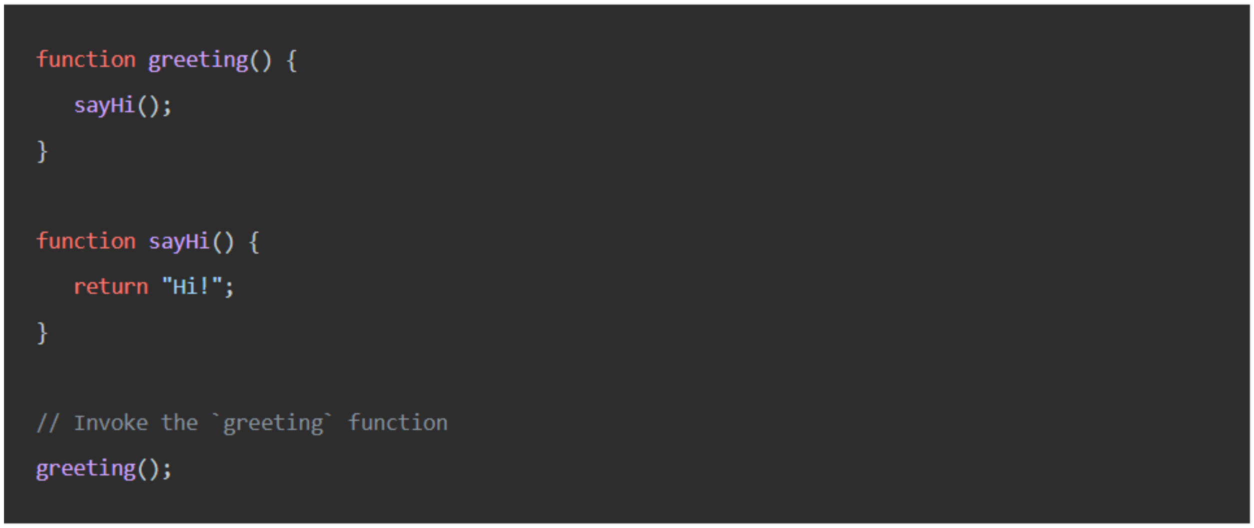
The Call (Execution) Stack

A stack is a data structure that follows the Last in First Out (LIFO) principle. However, the execution stack is a stack that keeps track of all the execution context created during code execution.

STEP 1: The global execution context is initially added(pushed) on the execution stack by default, in the form of the global() object.

STEP 2: A Function execution context is added to the stack when a function is invoked or called.

STEP 3: The Invoked function is executed and removed (popped) from the stack, along with its execution context.



```
function greeting() {
   sayHi();
}

function sayHi() {
   return "Hi!";
}

// Invoke the `greeting` function
greeting();
```

Example:

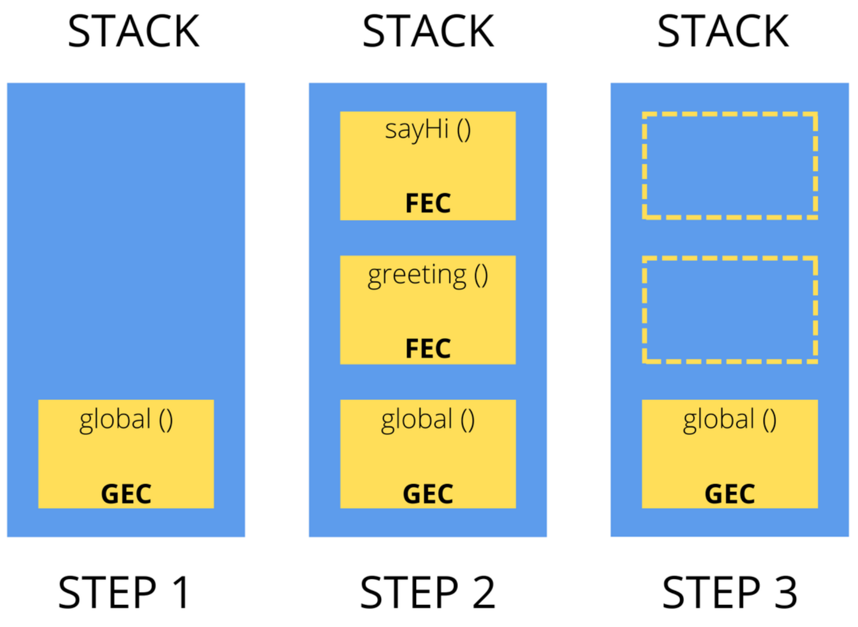
STEP 1: The GEC is created and pushed on the execution stack as the `global()` object.

STEP 2: The `greeting()` function is invoked and pushed on the stack.

STEP 2: The `sayHi()` function is invoked and pushed on the stack.

STEP 3: The `sayHi()` function is popped off the stack.

STEP 3: The `greetings()` function is popped off the stack.



Now let's understand how an actual JS code goes through the two stages of the Execution Context.

This is how the JS engine will execute the preceding code in the two stages of the Execution Context.

Creation stage

Here our code is being scanned for variable and function declarations. In the creation stage, the JS Engine creates a Global Execution Context to Execute the global code, which should look like this.

When the add(40, 50) function is called, a new Function Execution Context is created to execute the function code. The Function Execution Execution Context will look like this in the creation stage.

Execution Stage

Here values are assigned to variables.

Global Code: In the Execution Stage, when variable assignments are done. This is what the Global Execution Context will look like.

Function Code: After the Global Execution Context has gone through the execution stage, assignments to variables are done. This also includes variables inside every function declaration in the Global Execution Context. The Function Execution Context will look like this during the execution stage

Conclusion:

The Execution Context is an abstract environment created by JS to execute code.

The Execution Stack is used to execute functions and keep track of all the Execution Context created.

The Execution context is created in two stages, the creation and the execution stage.

The Lexical Environment is a core part of the creation stage.

Variable assignments are done in the execution stage.

Hoisting is being able to access a variable before it is declared.