Execution of JavaScript code

Execution context is the concept for describing the internal working of a code. In JavaScript, the environment that enables the JavaScript code to get executed is what we call JavaScript Execution Context.

It is the execution context that decides which code section has access to the functions, variables, and objects used in the code. During the execution context, the specific code gets parsed line by line then the variables and functions are stored in the memory.

An execution context is similar to a container that stores variables, and the code gets evaluated and then executed. Thus, it is the execution context that provides an environment for the specific code to get executed.

Their are different types of execution context in js :

- Global Execution Context
- Functional Execution Context
- Eval Execution Context

Global Execution Context -

Global Execution Context is also called the base/default execution. Any JavaScript code which does not reside in any function will be present in the global execution context. The reason behind its name 'default execution context' where the code begins its execution when the file first loads in the web browser. GEC performs the two following tasks:

- Firstly, it creates a global object where it is for Node.js and Window object for the browsers.

- Secondly, reference the Windows object to 'this' keyword.

- Create a memory heap in order to store variables and function references.

- Then it stores all the functions declarations in the memory heap area and the variables in the GEC with initial values as 'undefined'.

Functional Execution Context -

Functional Execution Code is that type of context which is created by the JavaScript engine when any function call is found. Every function has its own execution context, and thus unlike GEC, the FEC can be more than one.

Also, FEC can access the entire code of the GEC, but it is not possible for GEC to access all the code of the FEC. During the GEC code execution, a function call is initiated, and when found by the JS engine, it creates a new FEC for that specific function.

Eval Execution Context -

Any JS code that gets executed within the eval function creates and holds its own execution context.

In order to understand working process of execution context following is an example of code.

```
let x = 'Vitthal Patil';
function one() {
 console.log('It is the first function');
function two() {
 console.log('It is the second function');
}
two();
}
one();
console.log('It is my way');
```

Flow of working :

1. Firstly all the code is loaded into the browser.
2. After it the js engine pushes the GEC at top of execution stack.
3. As soon as js engine encounters the first engine call, it setup a new FEC for it.
4. Then we can see that it is the invocation of the second function within the first function.
5. When the second function is completed, the execution function is popped out of the stack

6. At last, when the execution of the entire code gets completed, the JS engine removes the GEC from the current stack

.

This is the steps of execution of execution stack