JavaScript Code Execution

Last Updated : 16 Nov, 2021

JavaScript is a *synchronous* (Moves to the next line only when the execution of the current line is completed) and *single-threaded* (Executes one command at a time in a specific order one after another serially) language. To know behind the scene of how JavaScript code gets executed internally, we have to know something called **Execution Context** and its role in the execution of JavaScript code.

Execution Context: Everything in JavaScript is wrapped inside Execution Context, which is an abstract concept (can be treated as a container) that holds the whole information about the environment within which the current JavaScript code is being executed.

Now, an Execution Context has two components and JavaScript code gets executed in two phases.

- **Memory Allocation Phase:** In this phase, all the functions and variables of the JavaScript code get stored as a key-value pair inside the memory component of the execution context. In the case of a function, JavaScript copied the whole function into the memory block but in the case of variables, it assigns *undefined* as a placeholder.
- **Code Execution Phase:** In this phase, the JavaScript code is executed one line at a time inside the Code Component (also known as the Thread of execution) of Execution Context.

Let's see the whole process through an example.

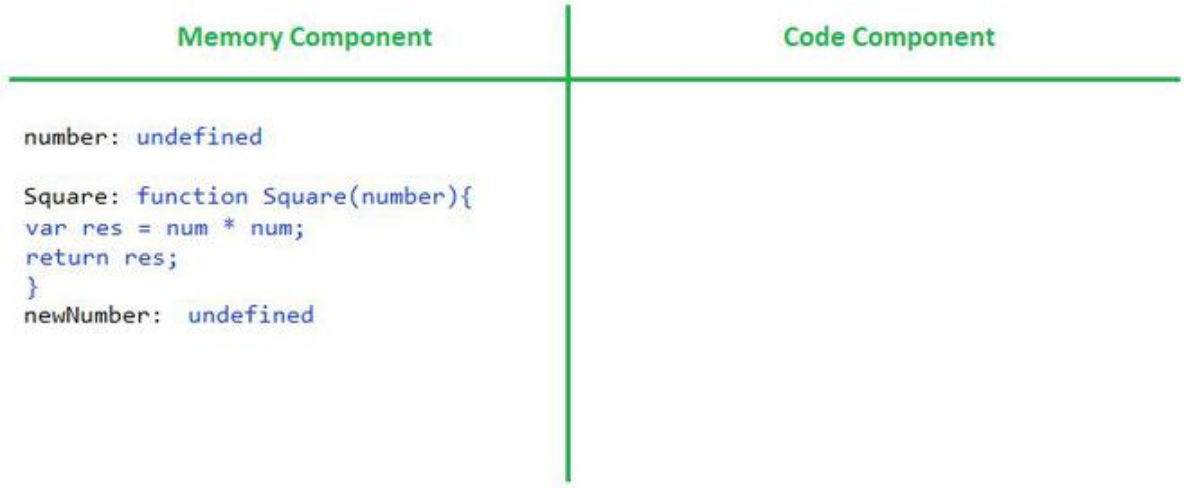
Javascript

```
var number = 2;
function Square (n) {
    var res = n * n;
    return res;
```

```
}
var newNumber = Square(3);
```

In the above JavaScript code, there are two variables named *number* and *newNumber* and one function named *Square* which is returning the square of the number. So when we run this program, Global Execution Context is created.

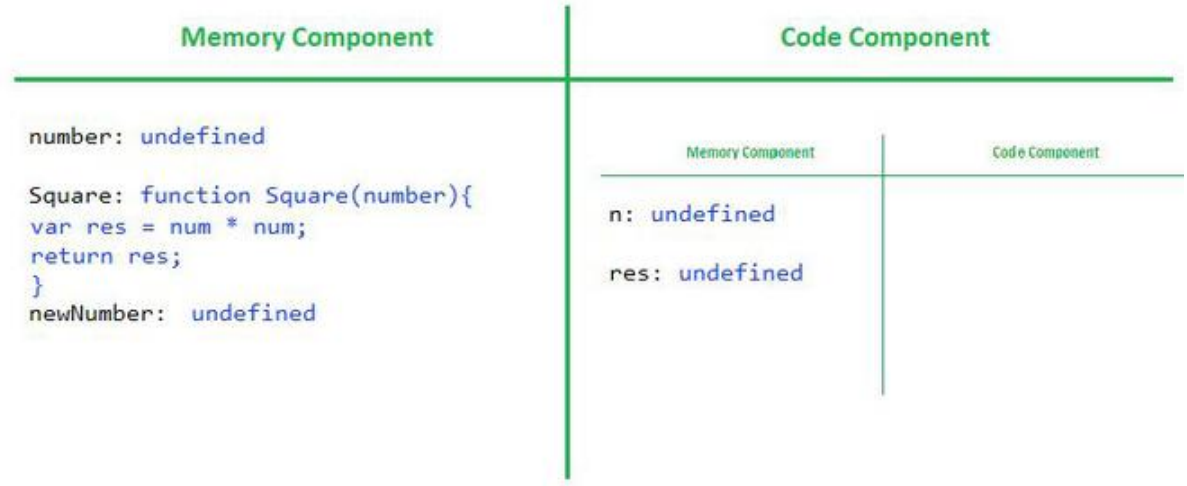
So, in the Memory Allocation phase, the memory will be allocated for these variables and functions like this.



Global Execution Context

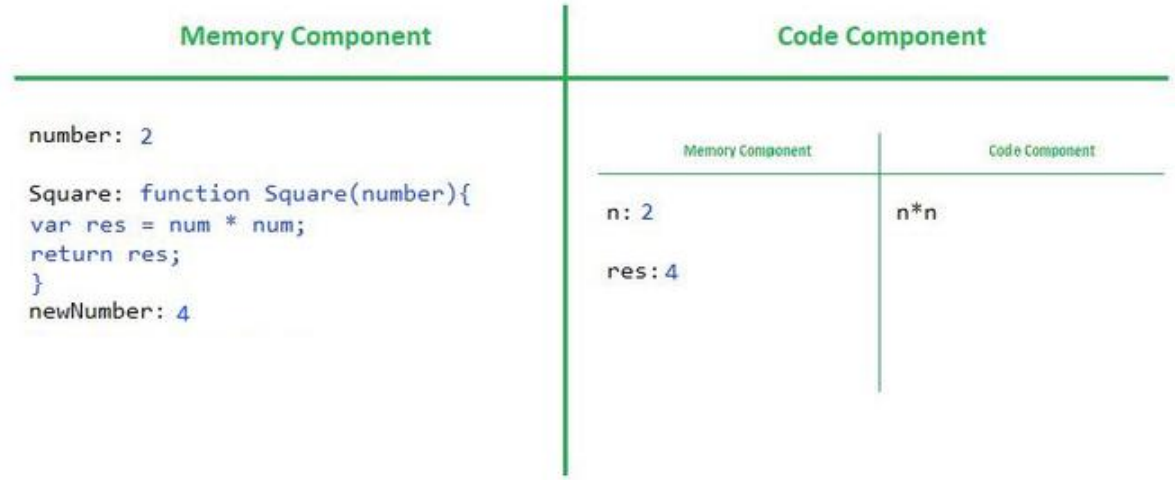
In the Code Execution Phase, JavaScript being a single thread language again runs through the code line by line and updates the values of function and variables which are stored in the Memory Allocation Phase in the Memory Component.

So in the code execution phase, whenever a new function is called, a new Execution Context is created. So, every time a function is invoked in the Code Component, a new Execution Context is created inside the previous global execution context.



Global Execution Context

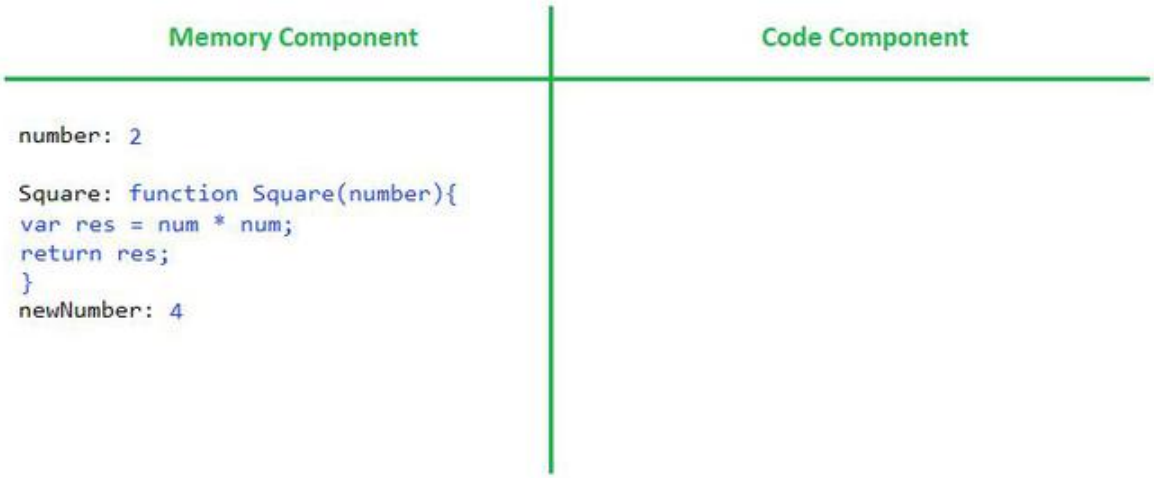
So again, before the memory allocation is completed in the Memory Component of the new Execution Context. Then, in the Code Execution Phase of the newly created Execution Context, the global Execution Context will look like the following.



Global Execution Context

As we can see, the values are assigned in the memory component after executing the code line by line, i.e. *number: 2, res: 4, newNumber: 4.*

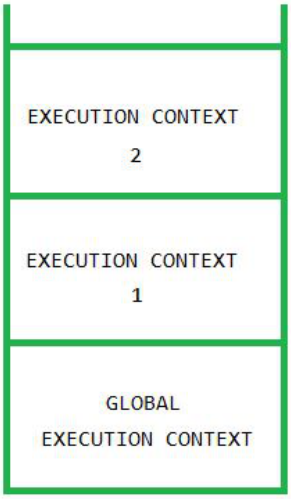
After the *return* statement of the invoked function, the returned value is assigned in place of undefined in the memory allocation of the previous execution context. After returning the value, the new execution context (temporary) gets completely deleted. Whenever the execution encounters the return statement, It gives the control back to the execution context where the function was invoked.



Global Execution Context

After executing the first function call when we call the function again, JavaScript creates again another temporary context where the same procedure repeats accordingly (memory execution and code execution). In the end, the global execution context gets deleted just like child execution contexts. The whole execution context for the instance of that function will be deleted

Call Stack: When a program starts execution JavaScript pushes the whole program as global context into a stack which is known as **Call Stack** and continues execution. Whenever JavaScript executes a new context and just follows the same process and pushes to the stack. When the context finishes, JavaScript just pops the top of the stack accordingly.



Call Stack

When JavaScript completes the execution of the entire code, the Global Execution Context gets deleted and popped out from the Call Stack making the Call stack empty.

JavaScript Code Execution

Next Article

Convert a Number to a String in JavaScript

Similar Reads

What is An Event Loop in JavaScript?

The event loop is an important concept in JavaScript that enables asynchronous programming by handling tasks efficiently. Since JavaScrip...

15+ min read

How JavaScript works and code is executed behind the scene ?

JavaScript is an interesting language in the world and its working procedure quite be different from other languages. JavaScript is...

15+ min read