When a script file is scanned by the JavaScript engine, it creates an Execution Context, which manages the transformation and execution of the code. During runtime, the source code is parsed, and memory is allocated for variables and functions. The code is then generated and executed.

There are two types of execution contexts: global and function. The global execution context represents the global scope and is created when a JavaScript script starts running. On the other hand, a function execution context is created whenever a function is called and represents the local scope of that function.

In creation phase the memory is allocated to

Variables and functions ,



Let's consider the example:

var n = 5;

function square(n) {

var ans = n \* n;

return ans;

}

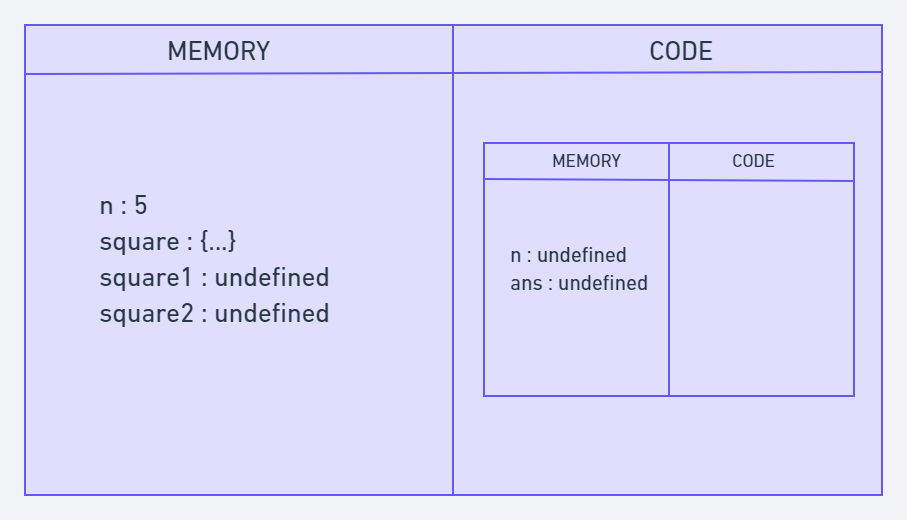
var square1 = square(n);

var square2 = square(8);

console.log(square1);

console.log(square2);

In the creation phase, memory is allocated for variables and functions. During the execution phase, the code is executed line by line. The value 5 is assigned to the variable n when encountered. The square function is invoked, creating a new function execution context. The calculation is performed, and the result is stored in the ans variable. The function returns the value, and the function execution context is destroyed.

Execution phase:  


The returned values from square() are assigned to square1 and square2. Once the code execution is completed, the global context is destroyed.

The Call Stack is used by the JavaScript engine to keep track of all the contexts, including the global and function contexts. It follows the Last-In-First-Out (LIFO) principle. The global context is initially pushed onto the stack. When a function is invoked, its context is created and pushed to the top of the stack. After the execution of a function is finished, its context is removed from the stack, returning to its parent context.