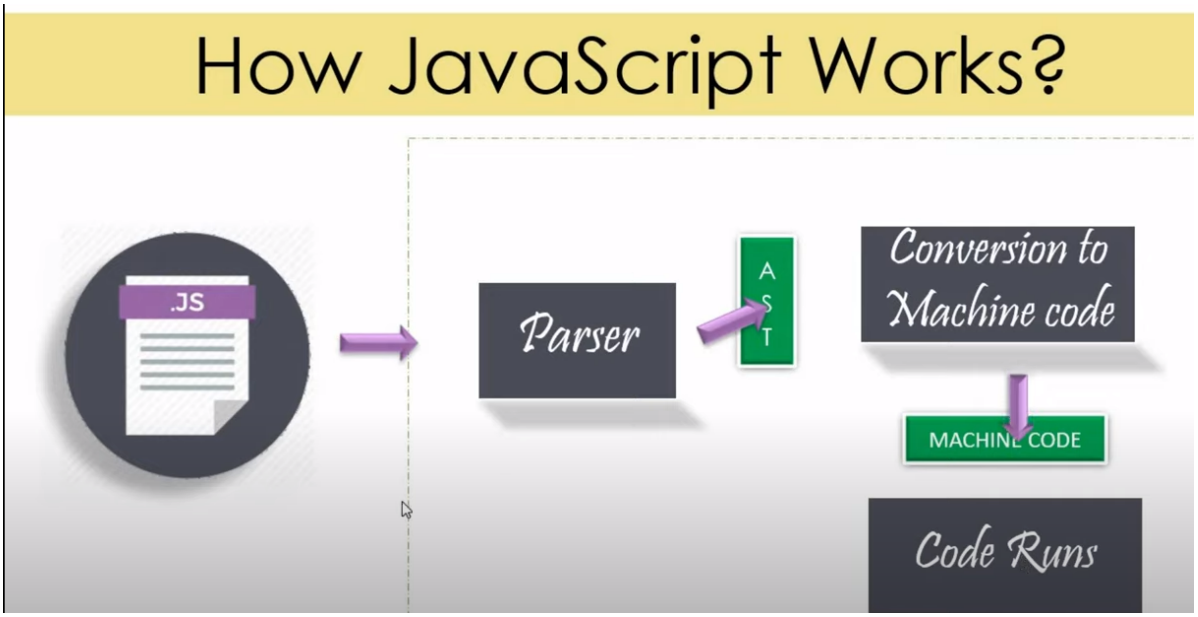
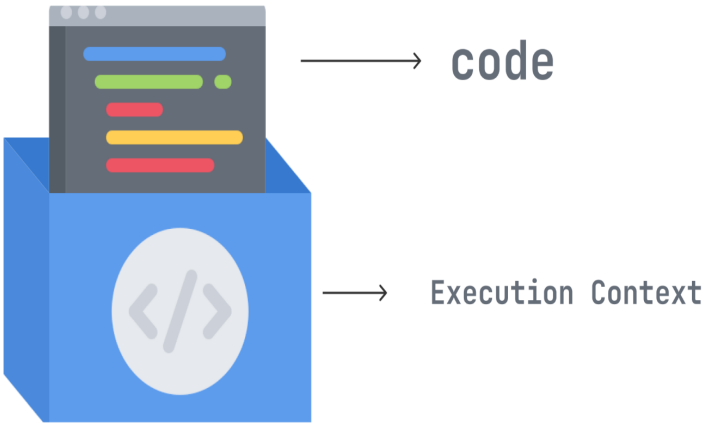
Chapter 1

Reference:[Reference for Execution Context]

HOW JAVASCRIPT WORK
JavaScript is an interpreted language, therefore it is parsed line by line through the parser.



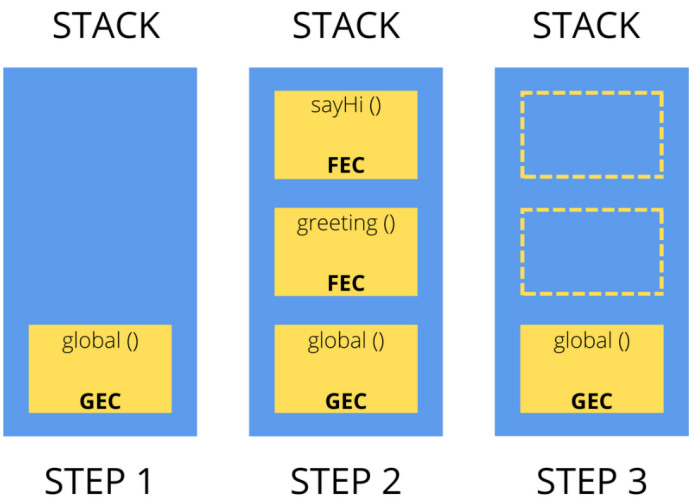
Execution context: a conceptual environment created by JS for code evaluation and execution



Execution stack

Call (Execution) Stack is a stack that keeps track of all the execution context created during code execution. Follows Last in First Out (LIFO) principle.
Example:

```
function greeting() {
    sayHi();
}
function sayHi() {
    return "Hi!";
}
// Invoke the `greeting`
function
greeting();
```



The *this* Binding

The *this* binding component is where its value is set.

- ➔ The value of *this* is set to the global object (i.e., window object in browsers) in the Global Execution Context
- ➔ In the Function Execution Context, the *this* value depends on how the function is called. If the function is a method in an object, this value of the function is set to that object when called by the object reference. Otherwise, it is set to the global object.

Variable Environment

The Variable Environment Is a type of Lexical Environment used specifically by the var VariableStatement within an Execution Context. This means that the Environment Record of the VariableEnvironment only records bindings created by the var VariableStatement.

- ➢ Whereas the Environment Record of the **LexicalEnvironment** records both variable (let and const) bindings and function declarations.

How code is executed

1. **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.
   When a function is called, a new Function Execution Context is created to execute the function code.
2. **Execution Stage**
   Here values are assigned to variables.
   Global Code: In the Execution Stage, when variable assignments are done.

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 reason behind hoisting

The code is scanned for variable and function declarations during the creation stage. The function declaration is stored completely in memory, while variables are initially set to undefined(for var declaration) or remain uninitialized (for let and const declarations).

This is why variables defined with var can be accessed before they are declared, though you will get a value of undefined. However, this is not the case for variables defined with let and const, which will throw a reference error when accessed before being declared. This is known as *hoisting* in JS, which is simply being able to access a variable before it is declared.