**How JavaScript works behind the Scenes**

**How our code is translated into machine code?**

Java Script Engine

Code runs

Conversion to Machine Code

Parser

Our Code

Machine Code

Abstract Syntax Tree

JavaScript Engine is a program which executes the java script codes. They are many other engines available.

Our code is parsed by a parser which basically reads our code line by line and checks the syntax of our code that we gave it is correct. In case of incorrect syntax it throws an error and stops the execution (It knows the rules of JS and how it has to written in order to be valid). And if everything is correct parser produces a data structure known as Abstract Syntax tree which is then translated into machine code so this code is no longer a JavaScript code but a set of instruction that can be executed by the computer processor. And actually then our code runs performs its work.

**Execution Contexts:**

* All the JS needs an environment to run and this environment are called execution contexts.
* A box or a wrapper which stores variables and in which a piece of our code is evaluated and executed
* The default execution context is the global execution context (is for variable or function that are not inside of any functions). It is associated with global object which in the case of browser is the window object. Everything we declare in global object gets automatically attach to the window object in the browser

Example:

var name = ‘John’; (is stored in the global execution context) and the function declaration is also consider in global execution context.

* But when we call any function a new execution context is created on top of global context forming a stack called execution stack. For every new function, new execution context is created.
* For the duration of that function the new execution context becomes act of context for that function and after the function execution completes the context is removed from the top of the stack.

Execution Context Object has three properties:

1. Variable Object (VO)
2. Scope Chain
3. “This” variable

When a function is called and the new execution context is created on top and this happens in two phases:

1. **Creation Phase**: The creation of variable object, scope chain and value of ‘this’ variable is determine and set

2. **Execution Phase**: The code of function that generated the current execution context is run line by line and all the variables are defined.

If it’s a global context, then global code is executed.

**The Variable Object:**

* It contains the function argument, inner variable declaration and function declarations.
* The arg\* Object is created containing all the arguments that were passed into function. The code is scanned for function declaration: for each function a property is created in the variable Object pointing to the function. The function declaration is stored in vo even before its execution.
* The code is scanned for variable declaration: for each variable a property is created in the variable Object and set to undefined.
* Functions and variables are hoisting in JS which means they are available before the execution phase actually starts.

**The Scope Chain:**

* It contains current variable objects as well as variable objects of all its parents.
* Scoping basically answers: where can we access a certain variable
* In JS each new function creates a scope which is the space/environment in which the variables that it defines are accessible. Only way to create a scope in JS is function
* Lexically scoping: a function that is lexically within another function gets access to scope of the outer function (also called parent function) and also get access to its variables and functions that it(outer function) defines.

Based on Example of Scope Chain:

**Execution Stack** **Scope Chain**

EC third() Global Scope

EC second() first()

EC first() second()

Global (EC)Execution Context third()

(Order in which functions (Order in which functions are written in are called) code which are lexically)

**The “this” Keyword:**

Regular function call: the” **this”** keyword points at the global object (the window object, in the browser)

Method call: the this variable points to the object that is calling the method

The “this” keyword is not assigned a value until a function where it is defined is actually called

**Object and Properties:**

In object we define key-values pairs which means each value has a name called key

In object we can group variables together that belong together, that have no particular order.

Object\_name.property = value;

**Object and Methods:**

Object can contain methods and we can directly add a new property to an object with a value returned by a function already in the object.

**Loops and Iterations:**

Loops are used to perform repetitive task for a number of times.

**For loop:**

The for loop loops through a block of code a number of times

for (initial value; condition; increment){ //code }

Initial value: sets a variable before the loop starts

Condition: defines the condition for executing the code block.

Increment: counter update after each iteration

**While loop:**

The while loop loops through a block of code as long as a specified condition is true

If you forget to increase the variable used in the condition, the loop will never end.

while (condition) { //code }

**Do-while loop:**

This loop will execute the code block once, even if the condition is false and then it will repeat the loop as long as the condition is true.

do { //code } while (condition);