Name : Mayur Purushvani

How Our code will execute ?

JavaScript Engine :

Java script engine is that where our code is executed!

It is an program where our code is executed.

In java script engine, there are mainly 3 processes which is :

- Parser
- Conversion to machine code
- Code Runs

Parser :

The parser is the read our code line by line and checks our syntaxes.

If everything is correct then it passed to the conversion part

If there are some syntax error, then it will not execute further, and give error to user.

It is an abstract syntax tree.

Conversion to machine code :

This part is convert our code to the machine code

When machine code is ready to execute, then finally code passes to the Run code section.

Code Runs :

When everything is fine, then this section execute our code and generate the output.

Execution Context :

A box, a containers, or a wrapper which stores variables and in which a piece of code is evaluated and executed.

Now, every execution context block have a global execution context box. Which is as follow:

Global Execution Context Box :

Global execution is that, where some piece of code is executing in queue.

Code that is **not inside any function.**

Associated with the **Global Objects.**

In the browser, That's the window object.

e.g. : lastname === window.lastname //true

Now, How it will execute our code, let's take a example :

```
<html>
<head>
   <title>Execution Example</title>
</head>
<script type="text/javascript">

var name = "mayur";

function first()
{
   var a = "hello";
   second();
   console.log(a + name);
}
function second()
{
   var b = "hey";
   third();
   console.log(b + name);
}
function third()
{
   var c = "Hi";
   console.log(c + name);
}
first();
</script>
</html>
```

In above example, There are three functions named : first(), second(), and third().

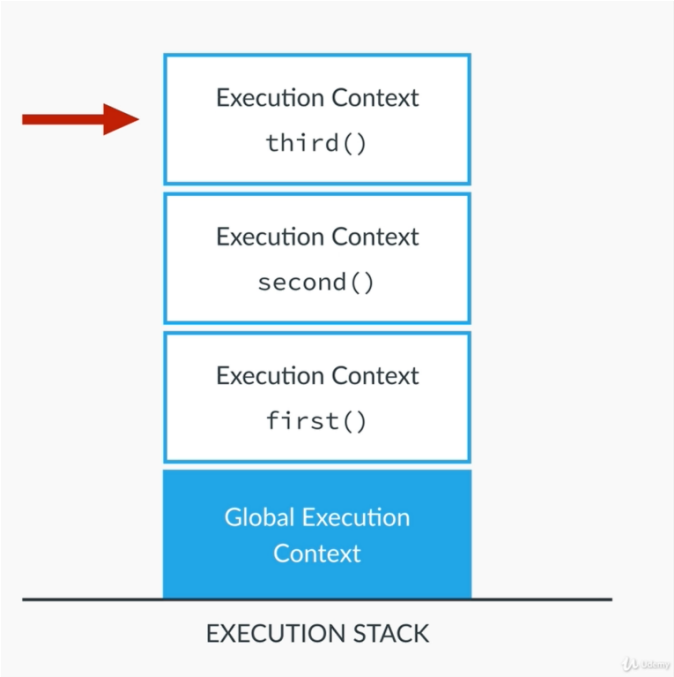
I've called the second function from 1st, and third function from 2nd.

Now, what is happen, At last I've execute only first().

So, One Global execution is start, then it declare first() execution, then second() and then third().

And as follow, When execution is completed, the pop() is executing, like first pop the third(), then second(), and then first().

Below is the image that will help you to understand whole execution process.



Execution Context Objects :

- Variable Objects (VO)
- Scope chain
- "This" variable

Variable Objects (VO) :

It contains variable arguments.

Scope Chain :

It contains the current variable objects

This variable :

It contains current variable's value.

Hoisting :

The hoisting is based on the main two concepts :

- Function declaration and function expression
- Variables

✓ Fucntions :

In function declaration and function expression, there are some cases which you must follow like below example:

```
        <script>


    //Fucntion Declaration
    calculateAge(1999) //Before
    function calculateAge(year)
    {
       console.log(2020 - year);
    }
     calculateAge(1999) //After


    //Fucntion Expression
    calculateAge1(1999) //Before
    var calculateAge1 = function(year)
    {
       console.log(year);
    }
    calculateAge1(1999) //After

  </script>
```

In above example, You can see I've created two functions, and I'll execute that before declare an function and after declare an function. In Function declaration, It'll executed and output is shown but in function expression, we can not execute before the function expression. That is the main difference between function declaration and function expression.

The output of above code is :

21

21

ERROR //We can not Execute function before function expression.

21

- ✓ Variables :

  In variable, we can declare an global as well as local variables. Let's take an example.

```
//Variables
console.log(age);  //Before
var age = 22;
function foo()
{
   console.log(age);  //Before local
   var age = 65;
   console.log(age);  //After local
}
foo();
console.log(age);  //After
```

In variables, We can see that, We can write the variable before it declared. But the output of that variable is **undefined.** When we simply write console.log(age) and we don't have any variable declared in code, So, it generates an error.

We can also see the difference between local variables and global variables.

The output of the following code is :

Undefined

Undefined

65

22

Scoping :

- Scoping is that where we can access a certain variables.
- Each new function creates it's new scope.
- Lexical scoping : A function that is lexically within another function gets access to the scope of the outer function.

Let's take an example :

```
        <script>

    var a = 'Hello!'; //GLOBAL SCOPE
    first();

    function first()  //LOCAL SCOPE IN first() and second()
    {
        var b = 'Hi!';
        second();

        function second() //LOCAL SCOPE IN second()
        {
            var c = 'Hey!';
            console.log(a+b+c);
        }
    }
</script>
```

In above example, we can see that the global scope variable 'a', which is declare globally. And we can access it in local functions as well.

And in first() function, we can see that we can not access the variables of second().

And in second() function, we can access whole variables declared in code, because second() is a local function.

Output is : Hello!Hi!Hey!

Now we can try to declare third() function outside the local functions. And try to access the variables of local functions :

```
<script>
    var a = 'Hello!';
    first();

    function first()
    {
        var b = 'Hi!';
        second();

        function second()
        {
            var c = 'Hey!';
            console.log(a+b+c);
            third();
        }
    }
    function third()
    {
        var d = 'mayur';
        console.log(a+b+c+d); //This will not work.
         console.log(a+d); //This will work.
    }
</script>
```

We can not access the local variables from globally. So this code generates an error.

Since the third() can not in the scope of second(), we can not access the variables of second() function.

Output : Error

“This” keyword :

- This keyword is a Regular function call and points at the global objects.
- Method call : This variable points to the object that is calling the method.
- This keyword is not assigned a value until a function where it is defined is actually called.’

```
<script>

  //console.log(this);

  calculateAge(1999);

  function calculateAge(year)
  {
     console.log(2020 - year);
     console.log(this);
  }
 </script>
```

In above example, this represent nothing. Because this keyword is inline the local scope.

```
<script>
var mayur = {
   name : 'mayur',
   birthdate : 1999 ,
   calculateAge : function()
   {
      console.log(this);
      console.log(2020 - this.birthdate);

     /* function innerFunction()
      {
         console.log(this);
      }
      innerFunction(); */
   }
}
mayur.calculateAge();

var mike = {
   name : 'mike',
   birthdate : 1998
};

mike.calculateAge = mayur.calculateAge;
mike.calculateAge();


</script>
```

In above example, I've created a some functions. The var mayor is the expression which have one sub function named ' calculateAge()'. And inside that I put the log detailes.

So, this keyword represents the whole object. Like name:'mayur', birthdate:1999, calculateAge:22

And after that, I've created one more function outside the function 'mayur'.

And just assign the calculateAge() method to the mike's function.

And now mike also can access the calculateAge() method. But this represents the current variable values. You can see the difference between this two values.

Output :

{Like name:'mayur', birthdate:1999, calculateAge:21}

21

{Like name:'mike', birthdate:1998, calculateAge:22}

22

Objects and properties :

Example :

```
<script>
    var mayur  = {
        firstName : 'mayur',
        lastName : 'purushvani',
        birthYear : 1999,
        family : ['dad','mom','sister'],
        isMarried : false
    };
    console.log(mayur);
    console.log(mayur.firstName);
    mayur.birthYear = 1998;
    mayur.isMarried = true;
    console.log(mayur);

    var abcd = new Object();
    abcd.firstName = 'abcd';
    abcd.isMarried = false;
    console.log(abcd);
</script>
```

Objects and Methods :

Example :

```
<script>
            var mayur  = {
        firstName : 'mayur',
        lastName : 'purushvani',
        birthYear : 1999,
        family : ['dad','mom','sister'],
        isMarried : false,
        calcAge : function()
        {
            this.age = 2020 - this.birthYear
        }
    };
    mayur.calcAge();
    console.log(mayur);

</script>
```

Loops and iteration :

Example :

```
<script>
    //1
    for (var j = 0; j<10; j++)
    {
        console.log(j);
    }
    //2
    var mayur = ['mayur','vasudev','purushvani'];
    for (var i = 0;i<mayur.length;i++)
    {
        console.log(mayur[i]);
    }
    //3
    var mp = ['mayur','vasudev','purushvani'];
    var i = 0;
    while ( i < mp.length)
    {
        console.log(mayur[i]);
        i++;
    }
    //4 CONTINUE
    var mvp = ['mayur','vasudev',1999,'purushvani',true];
    for (var i = 0; i <  mvp.length; i ++)
    {
        if(typeof mvp[i] !== 'string') continue;
        console.log(mvp[i]);
    }
    //5 BREAK
    for (var i = 0; i <  mvp.length; i ++)
    {
        if(typeof mvp[i] !== 'string') break;
        console.log(mvp[i]);
    }
    //6 BACKWARD LOOP
    for(var i = mvp.length - 1; i>=0; i–)
    {
        console.log(mvp[i]);
    }
  </script>
```

History of JavaScript :

1996 : changed from livescript to javascript.

1997 : ES1 became a first version

2009 : EC5 was released with lots of new features

2015 : ES6 Biggest update to the language ever.

2016/2017/2018/2019 : Release ES2016/ES2017/ES2018/ES2019