JavaScript Execution Context

Summary: in this tutorial, you will learn about the JavaScript execution context to deeply understand how JavaScript code gets executed.

Introduction to the JavaScript execution context

Let's start with the following example:

```
let x = 10;

function timesTen(a){
    return a * 10;
}

let y = timesTen(x);

console.log(y); // 100
```

In this example:

- First, declare the `x` variable and initialize its value with `10`.
- Second, declare the `timesTen()` function that accepts an argument and returns a value that is the result of the multiplication of the argument with `10`.
- Third, call the `timesTen()` function with the argument as the value of the `x` variable and store result in the variable `y`.
- Finally, output the variable `y` to the Console.

Behind the scenes, JavaScript does many things. in this tutorial, you will focus on execution contexts.

When the JavaScript engine executes the JavaScript code, it creates execution contexts.

Each execution context has two phases: the creation phase and the execution phase.

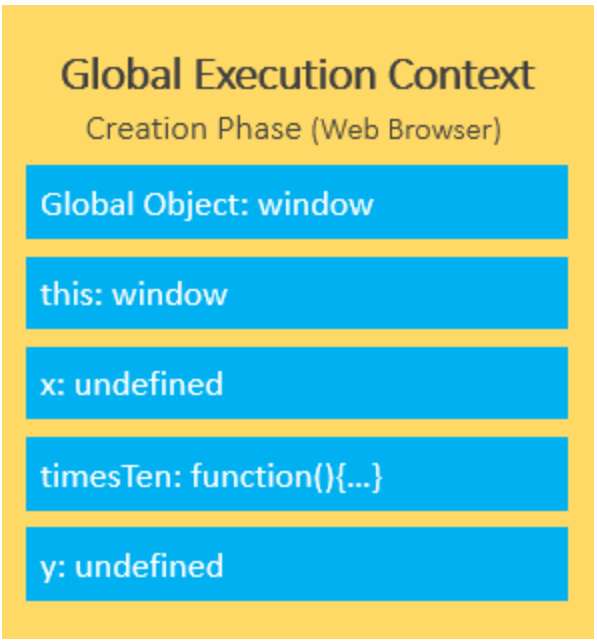
The creation phase

When the JavaScript engine executes a script for the first time, it creates the global execution context. During this phase, the JavaScript engine performs the following tasks:

- Create the global object i.e., `window` in the web browser or `global` in Node.js.
- Create the `this` object and bind it to the global object.
- Set up a memory heap for storing variables and function references.
- Store the function declarations in the memory heap and variables within the global execution context with the initial values as `undefined`.

When the JavaScript engine executes the code example above, it does the following in the creation phase:

- First, store the variables `x` and `y` and function declaration `timesTen()` in the global execution context.
- Second, initialize the variables `x` and `y` to `undefined`.



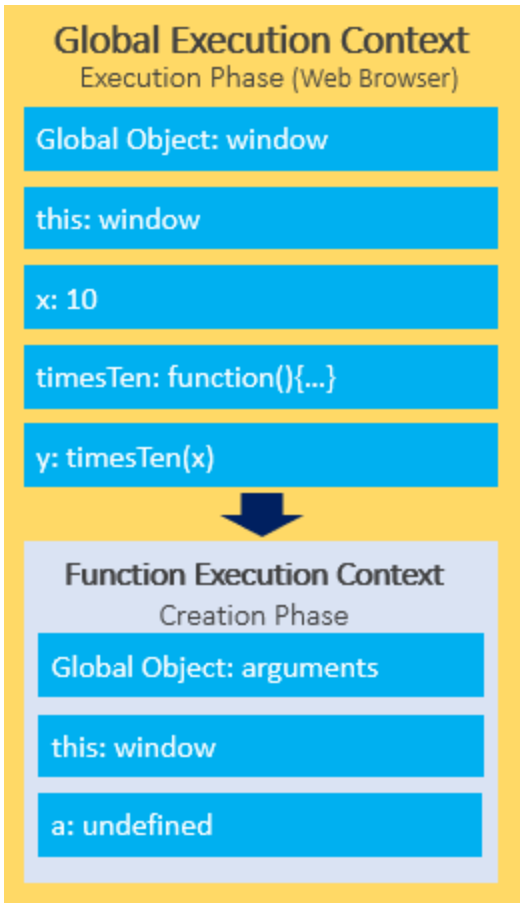
After the creation phase, the global execution context moves to the execution phase.

The execution phase

During the execution phase, the JavaScript engine executes the code line by line, assigns the values to variables, and executes the function calls.

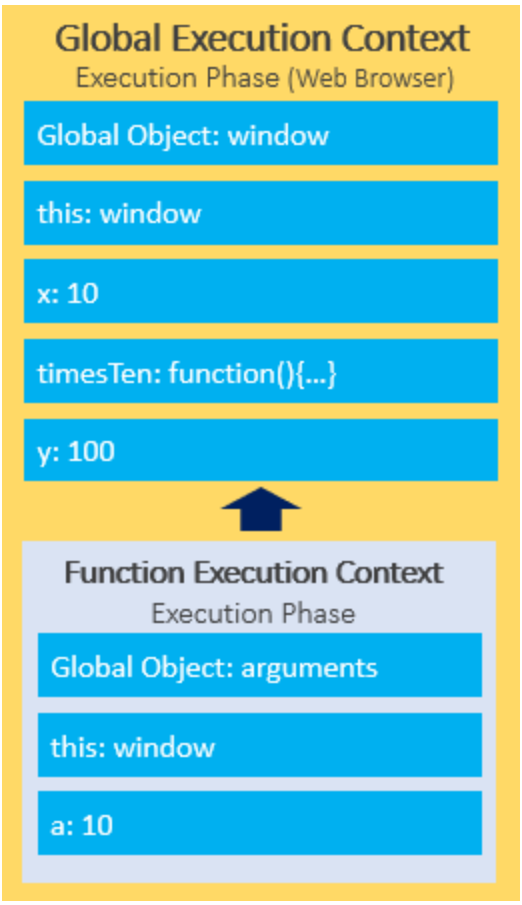
For each function call, the JavaScript engine creates a new **function execution context**.

The function execution context is similar to the global execution context. But instead of creating the global object, the JavaScript engine creates the `arguments` object that is a reference to all the parameters of the function:



In our example, the function execution context creates the `arguments` object that references all parameters passed into the function, sets `this` value to the global object, and initializes the `a` parameter to `undefined`.

During the execution phase of the function execution context, the JavaScript engine assigns `10` to the parameter `a` and returns the result (`100`) to the global execution context:



To keep track of all the execution contexts, including the global execution context and function execution contexts, the JavaScript engine uses the call stack, which you will learn in the next tutorial.

In this tutorial, you have learned about the JavaScript execution contexts, including the global execution context and function execution contexts.