**1st. 25-8-23**

Topics for R & D;

* Global Execution Context
* Function Execution Context
* Hoisting
* Temporal Dead Zone
* Closures

**Execution Context**

Execution context (EC) is defined as **the environment in which the JavaScript code is executed**. The Execution Context contains the code that's currently running, and everything that aids in its execution.

“When Browser’s JS engine get the script file it will create **the environment in which the JavaScript code is executed** for the purpose of handling the transformation and execution of the code.”

During the Execution Context run-time, the specific code gets parsed by a parser, the variables and functions are stored in memory, executable byte-code gets generated, and the code gets executed.

**Global Execution Context:**

The GEC is the base/default Execution Context where all JavaScript code that is **not inside of a function** gets executed.

**Function Execution Context:**

Whenever a function is called, the JavaScript engine creates a different type of Execution Context known as a Function Execution Context (FEC) within the GEC to evaluate and execute the code within that function.

**EVAL Execution Context:**

**Temporal Dead Zone**

**Defination:**

The Temporal Dead Zone is the period of time between the hoisting of the variable and its actual declaration, during which the variable cannot be accessed.

**Start and End:**

Temporal Dead Zone starts when the code execution enters the block which contains the let or const declaration and continues until the declaration has executed.

**Why:**

This is to prevent developers from accidentally using variables before they are declared, which can lead to hard-to-debug errors.

**Solution:**

To avoid the Temporal Dead Zone, it's important to always declare your variables before using them and to use shadowing when necessary.

[**https://stackdiary.com/tutorials/temporal-dead-zone-in-javascript/**](https://stackdiary.com/tutorials/temporal-dead-zone-in-javascript/)

**Difference between let, var and constant Temporal Dead Zone**

The main difference between the temporal dead zone of a var, let, and const variable is when their TDZ ends.

let (or const) variable’s TDZ ends when JavaScript fully initializes it with the value specified during its declaration.

However, a var variable’s TDZ ends immediately after its hoisting—not when the variable gets fully initialized with the value specified during its declaration.

<https://www.freecodecamp.org/news/javascript-temporal-dead-zone-and-hoisting-explained/>

**Hoisting:**

**Hoisting** refers to JavaScript giving higher precedence to the declaration of variables, classes, imports and functions during a program’s execution.

Hoisting makes the computer process declarations before any other code.

Hoisting does not mean JavaScript rearranges or moves code above one another.

**Precedence of Declaration, initialization and invocation:**

Declaration > Initialization > Invocation

In other words, the program’s [declarations](https://www.codesweetly.com/declaration-initialization-invocation-in-programming#what-exactly-does-declaration-mean) got higher precedence over [initializations](https://www.codesweetly.com/declaration-initialization-invocation-in-programming#what-does-initialization-mean), [invocations](https://www.codesweetly.com/declaration-initialization-invocation-in-programming#what-does-invocation-mean-in-programming), and other code.

## Lexical Environment

In [JavaScript](https://www.frontendmag.com/insights/is-typescript-faster-than-javascript/), a lexical environment is a data structure that stores all variables and function declarations. It allows the interpreter to recognize which variables or functions are accessible in different scopes within your program.

A lexical environment consists of two components: an environment record and a reference to the outer environment.

* **An environment record** is an object that stores the variables and functions that are declared within the current lexical environment.
* **The reference to the outer environment** is a pointer to the parent lexical environment, allowing the interpreter to access variables and functions defined in parent scopes.

A new lexical environment is formed every time a function or block is executed. The environment record of this lexical environment will contain all the variables and functions declared within that particular function or block.

<https://www.frontendmag.com/tutorials/lexical-environment-in-javascript/>

## Closures

## When a function is defined in another function, the inner function has access to the code defined in that of the outer function, and that of its parents. This behavior is called ****lexical scoping****.

## <https://www.freecodecamp.org/news/execution-context-how-javascript-works-behind-the-scenes/>

In JavaScript, a [closure](https://www.frontendmag.com/insights/advantages-of-closures-in-javascript/) is a feature that allows a function to not only access variables and functions within its own lexical scope, but also to maintain that access even if the function is called outside of its original environment. A closure is created by defining a function inside of another function.

A closure holds on to its connection to its lexical environment, allowing it to access the variables and functions that existed at its creation. Consequently, closures are capable of “retaining” their condition, even when they are called in a different situation.

<https://www.frontendmag.com/tutorials/lexical-environment-in-javascript/>

**For Quiz:**

[**https://www.codeguage.com/courses/js/functions-closures-quiz**](https://www.codeguage.com/courses/js/functions-closures-quiz)

[**https://quiz.typeofnan.dev/closure-and-hoisting/**](https://quiz.typeofnan.dev/closure-and-hoisting/)

**Lexical scope:**

Function can access variable from outside due to its lexical scope.

Function outside the world

**Closure:**

Function inside the function

**Best explanation for the execution context and its phases**

[**https://www.freecodecamp.org/news/execution-context-how-javascript-works-behind-the-scenes/**](https://www.freecodecamp.org/news/execution-context-how-javascript-works-behind-the-scenes/)