1. A) **Execution context** is a way of describing how the internal processes of javascript execution work. It tells us how in the backend the flow of JS execution is carried out. The execution context tells which code segment has access to the functions, variables, and objects used in the code. It is of three types:
2. Global Execution context: All the code except the code segments inside a function is present inside Global execution context.
3. Function execution context: All the code that is inside a function is present in the functional execution context.
4. Eval function execution context: all the code that is inside an eval function is there in the eval function execution context.

B) **Call Stack** is a stack with LIFO structure which is used to store the execution context which was made during the execution of code. It pushes the execution context of that time (starting with global execution stack) and then on to each and every functional execution stack as it comes one after the other in the JavaScript code.

C) **Event loop** in javascript pushes the events in callstack which are queued up in the message queue and which are ready for execution.

2. **Call back** functions are those functions which are passed as an argument in other functions. the aim of using callback function is to wait until the processing of 1st function before executing 2nd function. They are mainly used in Asynchronous execution of javascript.

When there are nested callback functions it is known as **callback hell**. Every nested callback takes the result of its parent callback. So if there is an error in one, there will be an error in all the callbacks.

The way of avoiding this callback hell is using promises and async/await.

3. **Promises** are the objects with take a function as arguments and are used to handle the asynchronous nature of the code. They represents the completion or failure of a callback function.

Syntax :

let promise = new Promise(function(resolve, reject) {

// logic

});

A promise is called using .then() function which has 2 arguments ( result or error) and renders text in both cases. If I want only succesfull or only error then we can also use only 1 argument.

.catch() can also be used for handling errors.

**Async/await** : async functions are just a better of making promises. When we write async keyword befor function declaration , that function automatically returns a promise that further contains other values.

Await keyword can be used only inside an async function and it waits until the promise is settles then it executes.

Eg:

async function func() {

let promise = new Promise((resolve, reject) => {

setTimeout(() => resolve("done"), 2000)

});

let result = await promise;

alert(result);

}

func();