Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

How JavaScript Run?

Behind the Scenes...

Created By
Gagan Saini


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

How JS Run?

In JavaScript, when your code is **executed**, it **doesn't** just run in **isolation**.

It operates within a **structured environment** known as the **execution context**.

This context provides a framework for managing **variables, functions**, and the scope of your code.


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

Global

GEC
Global Execution Context

This is the **first context created** when your JavaScript code starts running, like the main stage in a play.

It has **built-in objects** and **functions** (e.g., console, window) and **provides the foundation** for all other contexts.


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

Functional

FEC
Function Execution Context

Whenever you **call a function** in JS, a **new FEC** is created.

Think of it as setting up a **smaller stage** specifically for that function's execution.

Each **FEC** has its **own set of variables** and **function** arguments, separate from the GEC and other FECs.

This **isolation** helps **prevent** naming conflicts and unexpected behavior.


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

Execution Context

MEMORY	CODE
Variable: undefined Function: {.....}	Executes each line by line of source code from top to bottom in JS file.

Breakdown

Execution Context works in **2 phases**.

In the first phase it declare the **variables** & **functions** and **assign them memory**.

The **values** in variables are **still not assigned** so variable initially hold **undefined**.

And **function** stores their **complete body**.

First phase named as **Creation Phase**.

In the second phase **interpreter starts executing** JavaScript from the **top to bottom**.

And in this phase a **variable value will stored** in their corresponding variable.

And if function **call occurs** then function will be **called**.

Now, you might be wondering is that **everything** happening in **same context**?


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

No, JS start executing it’s code in **GEC** and as know as an function is called a **new execution context** (**FEC**) is created to run the function.

It will isolate the function variables from GEC & **executes the function code** (that’s why you can use same name variables inside & outside of a function).

And when you return from the function it’s **FEC** will be **removed from call stack**.


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

GEC

MEMORY	CODE
Variable: undefined Function: {.....}	

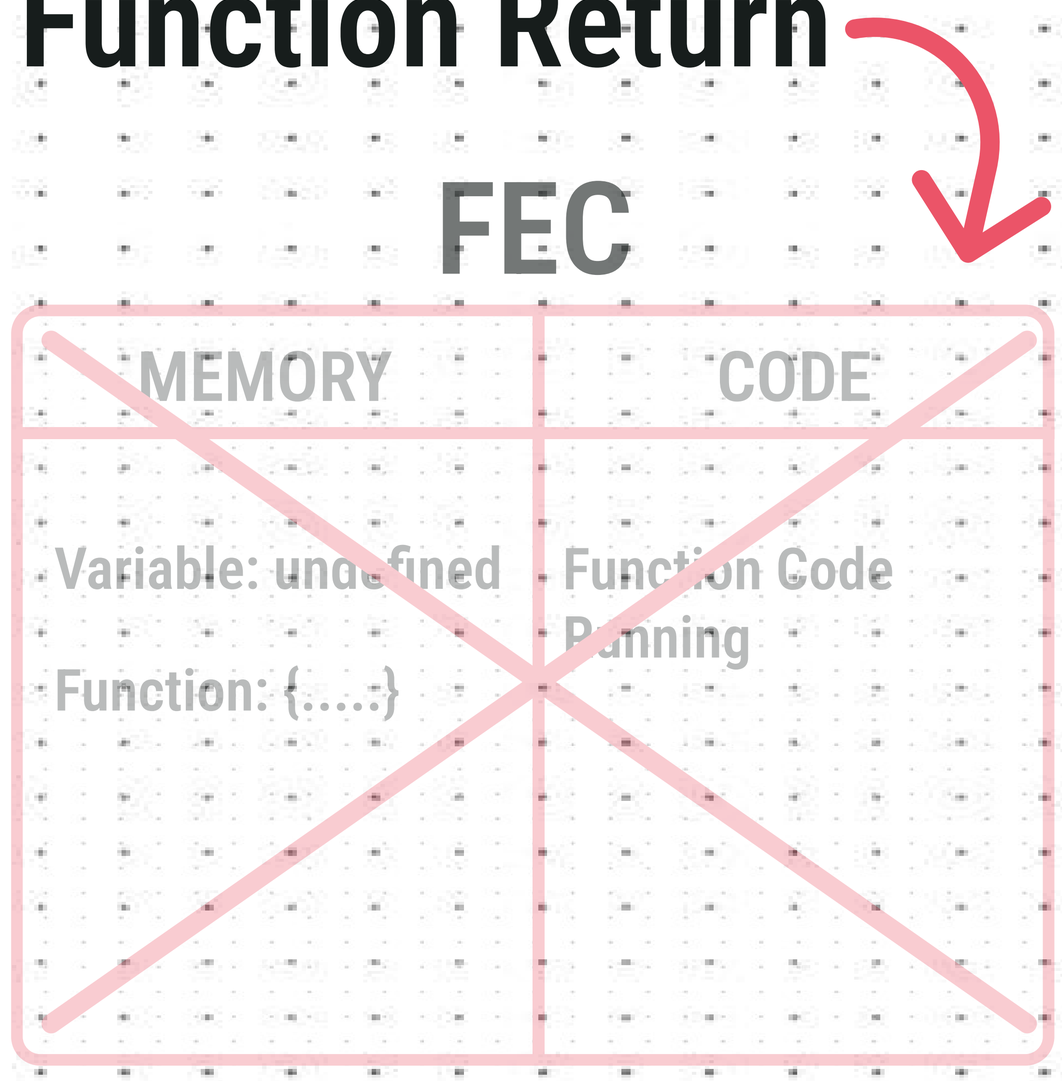
Function call

FEC

MEMORY	CODE
Variable: undefined Function: {.....}	Function Code Running

Function call **FEC created.**

GEC

MEMORY	CODE
Variable: undefined Function: {.....}	Function Return 

Returned from function **FEC removed**.


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

I used a term **Call Stack** in last paragraph.

So what does it mean?

Well our lovely **JavaScript** use Call Stack to track the function calls in program.

Whenever a new **function** is called it **got added in call stack**.

When it return then it **got removed** from call stack.

Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

The call stack works on **LIFO** (Last In First Out) principle.

It means the **first** called function is present at **bottom** of it and **last** called function is at **top**.


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

And yeah **Call Stack** can definitely store lot of function calls.

But **not infinite calls**, Call stack also has his **limit to store** after that it gives error.

callstack.js

```
const call = () => {
    console.log("Infinite");
    call();
}

call();
```


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

If you try to **run the code** given in previous snippet then you end up with this error.

“The Maximum Size of call stack exceeded”

```
▶Uncaught RangeError: Maximum call stack size exceeded
    at call (<anonymous>:2:5)
    at call (<anonymous>:3:5)
    at call (<anonymous>:3:5)
    at call (<anonymous>:3:5)
    at call (<anonymous>:3:5)
    at call (<anonymous>:3:5)
    at call (<anonymous>:3:5)
    at call (<anonymous>:3:5)
    at call (<anonymous>:3:5)
    at call (<anonymous>:3:5)
    at call (<anonymous>:3:5)
```

More in part 2...

Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

If found it useful then, Like & Share 💙

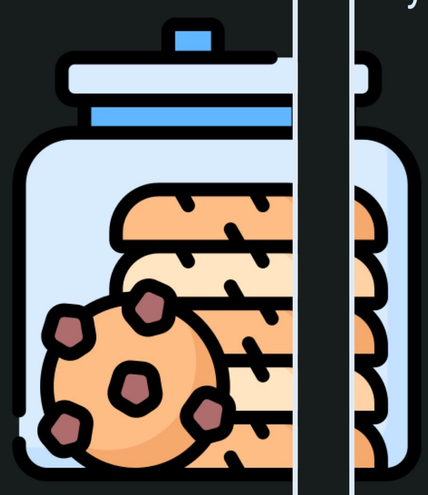
Learn What is Cookies?

Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

What is it?

How they are used to store data?



Gagan Saini
@gagan-saini-gs

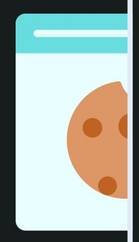
Github: Gagan-Saini-GS

COOKIES

They are small pieces of data websites store on your computer

Imagine tiny notes a website leaves on your browser.

These notes contain information like your login details, preferences, or items you added to your shopping cart.



Gagan Saini
@gagan-saini-gs

Websites use cookies and personalize your

They're like little reminders browser, saving you time a

Be mindful of user privacy

Always get consent before information in cookies.

Created By
Gagan Saini

Thanks for reading
Comment Feedback Below