Date: / /

Execution Context and Callstack in Javascript

Everything happens in javascript happens inside execution context

Each execution context has 2 components:

- Memory Component (Variable Environment)
- Code Component (Thread of Execution)

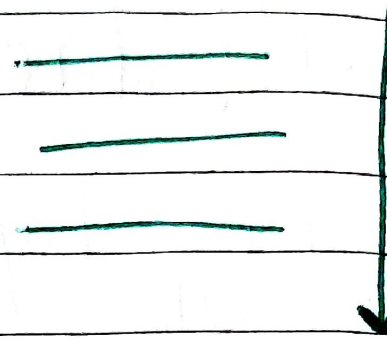
Each execution context has 2 phases also

- Memory Creation Phase
- Execution Phase

* In Memory Creation Phase, variables and function are declared and stored as key-value pairs in the memory Component

* In Execution Phase, code is executed line by line in code component

Date: / /

Memory Component	Code Component
key : value x : 5 fn : {..} It is variable Environment	 Code is executed line by line Thread of Execution

There are 2 types of Execution Context

1. Global Execution Context → gets created whenever program starts running

2. Function Execution Context → gets created whenever any function is invoked

Date: / /

Global Execution Context

This is first Execution context created wherever we run the code.

There can only be one GEC

* In memory creation phase of GEC (GEC is Global Execution Context here) inside memory component, the Javascript Engin will:

1. Create a global object
2. Create a this object
3. Set up memory space for variables and functions
4. Assign variables a default value of undefined and function declaration stored as key-value pairs.

Note → In the browser the global object would be window and In Node.js environment, the global object would be global.

Date: / /

Eg →

```
Var num = 10;
Function square (num) {
  var result = num * num;
  return result;
}
var ans = square (num);
```

Memory Creation phase of GEC for above example

Global Execution Context

Phase : Creation

window : global object
this : window
num : undefined
square : fn()
ans : undefined

Date: __/__/__

Points to be noticed here

1. window object is created

2. this which is GEC to window object (browser environment) created.

3. num and ans assigned a default value of undefined.

4. Any function declaration (square) placed into memory.

In execution phase of GEC in memory component, the JS Engine starts executing code line by line

In this phase following changes take place. → Earlier num : undefined
↓
Now num : 10

Earlier ans : undefined
↓
Now ans : 100
(after invoking square function)

Date: / /

Function Execution Context (FEC)

Execution context is created only 2 times throughout lifecycle of program.

Once, when JS starts executing code that is global Execution Context

And another when we invoke a function that is function execution context.

Similar to Global Execution Context the following thing happens in FEC with an Exception.

In memory creation phase of FEC, JS will

* create an argument object instead of global object.

* create this object whose value depends on what context is a function.

Date: __/__/__

★ Set up memory space for variables and functions

★ Assign the variables a default value of undefined and place any fn code as key-value pairs.

Note - Arrow functions don't get their own arguments and this object.

They use closest surrounding function's arguement and this object

Memory ~~Execution~~ creation Phase

Square Execution Context

Phase: Creation

arguements : {0: 10, length: 1}
this : window
result : undefined
num : 10

Date: / /

* this object is created whose value depends on which context the function was called (window object)
* arguement object get created
* num is assigned the value that is been passed as parameter to square fn that is 10
* result will contain undefined.

In Execution phase

result: 100

which is calculate after executing line 3.

Callstack

Callstact maintains order of execution of execution contexts.

Javascript can only execute one command at a time in a order.

Date: / /

The order of Execution is maintained by call stack.

So →

* When program starts running, the global execution context is created and pushed onto stack

* When square fn invoked, a function execution context is created and pushed onto stack

* Once sqare fn is done executing, It returns result and control of program where fn was originally invoked

* After this, sqr function popped off from stack

* Once last line of code finished executing, the global execution context popped off from stack.

The call stack becomes empty.