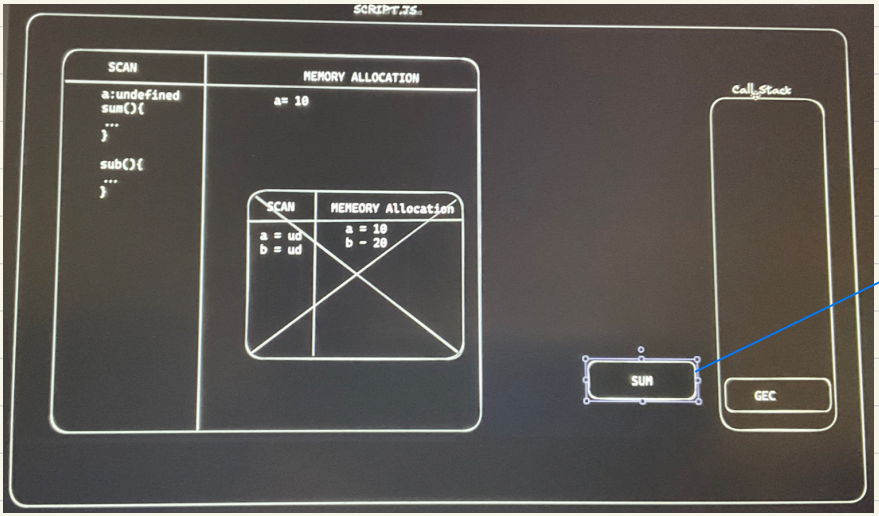
* All the code in JS goes either under Callstack or under Scope

* Whenever we run a JS code a Global Execution Context is created in Callstack (GEC)

* Stack : a pile



(GEC) is a part of the Callstack where the memory allocation takes place

* Hoisting is declaration of functions, variables or classes to the top of their scope, prior to execution of the code.

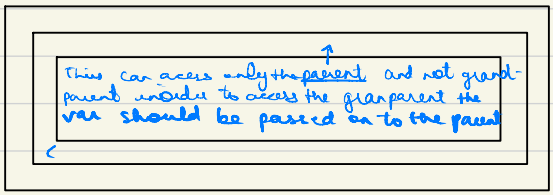
eg: sum (2, 2)

```
Function sum (a, b) {
    return a + b;
}
```

#Closure

```
function sum() {
    var a = 10;
    function sub() {
        console.log(a);
    }
    sub();
}
sum();
```

(also applicable with let)
Subfunction is having closure of a or lexical environment of sum function



* lexical means it has the access to the parent

eg:
```
function sum() {
    var a = 10;
    function sub(){
        console.log(a);
        function mult(){
            console.log(a);
        }
    }
```

here it does not have access to grand child so undefined

inorder to access you can write:
```
function sum() {
    var a = 10;
    function sub(x){
        console.log(a);
        function mult(){
            console.log(a);
        }
    }
    sub(a);
}
```

now subchild has access of grand-parent.

Home Work

Behaviour of : var sum = function () {

}

let sub = () => {

}

#HOF: Functions that take other functions as parameters or return a function

Callback: a function which can be passed as parameter to other function

```
function A(b){ HOF
}
function b(){ Callback
}
```

* setTimeout (() => {
    console.log ("hello");
  }, 3000);

  setInterval (() => {
    console.log ("Refresh");
  }, 1000);

- setTimeout is HOF
- arrow function is callback
- setTimeout has higher Precedence than setInterval.

Functional Programming

So that we dont have to write regular loop and function.

* foreach

```
const arr = ["Shiva", "Anurag", "Yuvrah"];
arr.forEach((under) => {console.log(under);});
```

Map, Filter, Reduce

①
```
const number = [1,2,3,4,5,6,7,8,9,16];
number.map((ind) => {
  return ind * ind;
});
```

output : [1, 2, 9, 16, 25, 36, 49, 64, 81, 256]

if we do not put curly brackets we dont have to write return but simply ind * ind;

② Filter

```
const country = ["Finland", "Scotland", "India"];
const count = country.filter((under) => {
                under.included("land");
              });
console.log(count);
```

② Reduce

```
const array1 = [1, 2, 3, 4];
const initialValue = 0;
const sumWithInitial = array1.reduce(
(accumulator, currentValue) => accumulator + current
 value, initial value
);
console.log(sumWithInitial)
```

output = 10