JS

Last Week :-

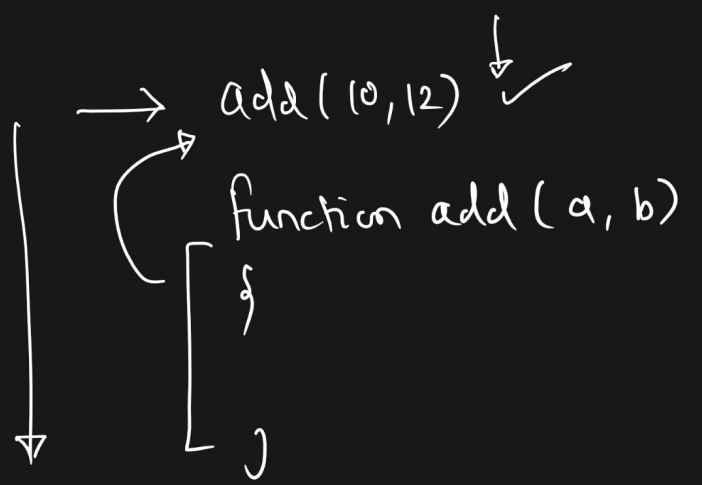
1) Variable Declr :- var, let, const
2) Hoisting , - variable / fun^n
3) execution context → memory allocation phase
   execution phase
4) call stack
5) data types :- primitive type
   - Number
   - String
   - boolean
   - bigint
   - null
   - undefined
   - symbol

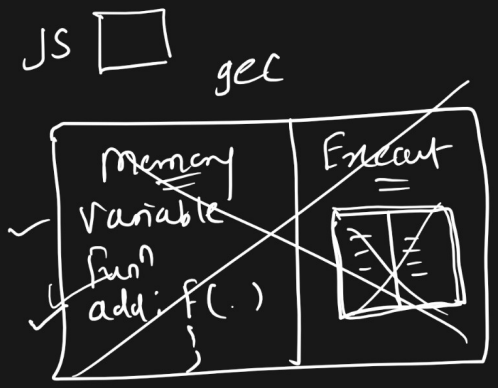
   Non primitive data type ⟹ object
6) operators
7) git hub → use, how to use

8) Functions :-

Type to define :- 1) Fun^n decl^r — } calling fun^n statement to call the fun^n
2) Fun^n Expression — }
3) Anonymous fun^n —
4) IIFE →
5) arrow fun^n

9) Fun^n Hoisting ⟹ We call the fun^n before it's defn →





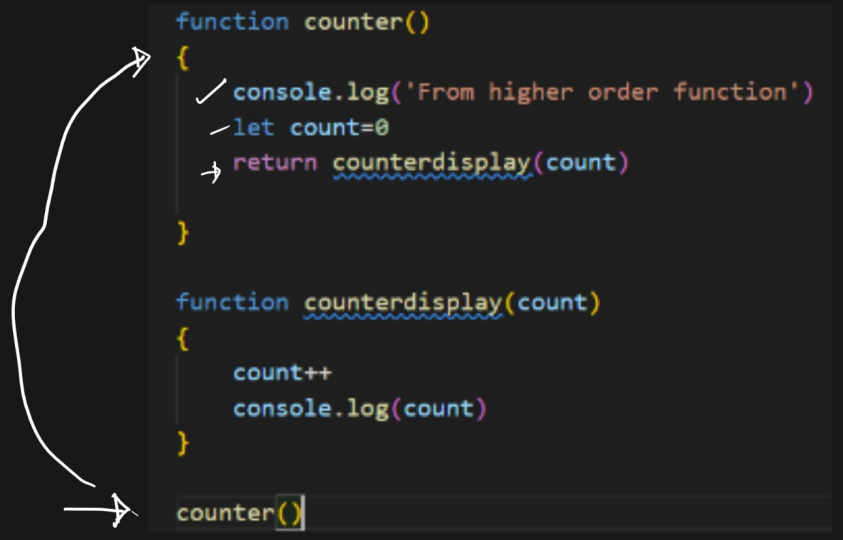
Types of Function (Refer → Folder 23/g → higher order fun? & call back fun? sub folder)

Higher Order Fun? & Call Back Function

Higher Order Fun? :-
Whichever the fun? accept the fun? as a parameter or any fun? return as a fun?
that fun? we called as higher order fun?

Call back fun? :-
Whichever the fun? we passed as a argument to any fun? then that fun? we called as call back fun?.

o/p → From high order fun?
counter()



```
function counter()
{
    console.log('From higher order function')
    let count=0
    return counterdisplay(count)

}

function counterdisplay(count)
{
    count++
    console.log(count)
}

counter()
```

```
function counter() {
    clg('From higher order fun');
    return counterdisplay
}
```

counter()
↓ call

✓ o/p → From higher order fun

```
function counterdisplay() {
    clg('From counter display');
}
counter()
```

✓ console.log(counter())

(return value display fun → counterdisplay)

```
let result = counter()
```

result ⇒ fun counterdisplay

c.log(result)

```
let result = function counterdisplay() {
}
c.log(result)
```

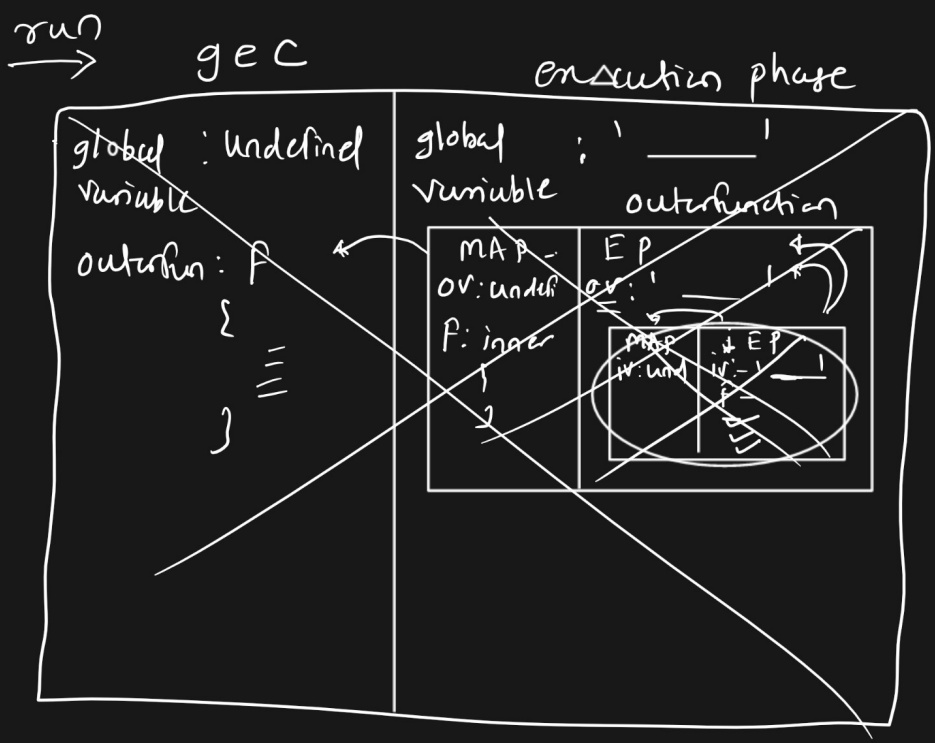
Type of scope

1) global scope - var variable & fun^
2) block scope → { } → bt variable
3) local scope → fun^ { _ } → local scope
4) script scope → let, const →
5) lexical scope

```
let globalvariable='I am script / global variable variable'

function outerfunction()
{
    let outervariable='I am outer variable'
    console.log('From outer function');
    console.log(outervariable)
    function innerfunction()
    {
        let innervariable='I am inner type of variable'
        console.log('from inner function');
        console.log(innervariable)
        console.log(outervariable)
          console.log(globalvariable)
    }
    innerfunction()
}

outerfunction()
```



Lexical Scope :- When a fun^n is created inside the another fun^n, the inner fun^n can access variables from the outer fun^n scope (and even further out to the global scope)

Closure :- when a fun^n is created inside the another fun^n and allowing to inner fun^n to access variables from the outer fun^n scope, even after the outer fun^n has finished their execution.