**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Javascript:-**

* Javascript is a scripting language.
* Javascript is implemented by Brendan in 1995, at Netscape.
* It was initially named Livescript.
* Latest version ES14 released in June 2023
* ES -> ECMA Script
* ECMA -> European Computer Manufacturers association
* Use of Javascript
* - Scripting
* - Making Page as dynamic.
* - Client side validation.
* - Modern UI framework applications
* - AngularJS
* - ReactJS
* - VeuJS
* - NodeJS
* - ...
* - etc
* Javascript programs(scripts) can be executed by either web browsers or Node Engines.
* Javascript is Object Based Scripting language.
* Javascript supports predefined objects
* Eg Object
* window,
* document,
* console, etc
* Javascript supports Primitives and nonprimitives
* primitives
* - string
* - number
* - boolean
* - undefined
* - null
* - symbol ???
* - bigInt
* nonprimitives
* - object
* - list
* - arrays
* - class ???
* - Set
* - Map
* - JSON

===============================================

Environmental Setup

===============================================

VS Code

Website :- https://code.visualstudio.com/#alt-downloads

NodeJS

Website :- https://nodejs.org/en/download/

Note:- Javascript programs must have extension '.js'

===============================================

Execution of Javascript programs

===============================================

demo.js

console.log("Hello world")

1. Command prompt

>node demo.js

Note:- 'document' object not supported here

2. HTML

index.html

...

<script src = "demo.js">

...

- open this file in web browser

- inspect console

3. HTML with script

...

<script>

console.log("Hello World")

</script>

...

- open this file in web browser

- inspect console

===============================================

Variables:-

===============================================

- Variables are used to store data.

- by using variables we can store any kind of data,

Eg number, sting, date, audio, video, file, etc.

- Variables can be declared using 'var', 'let', 'const'

- let and const introduced in ES5.

Strings:-

- Strings can be declared with '', "", ``

- `` introduced in ES6.

- `` is used to define multiline strings

var wish = 'Good morning'

console.log(wish)

var sub = `Javascript`

console.log(sub)

var myWish = `Welcome to ${sub}`

console.log(myWish)

Numbers

- Javascript supports decimal, double, bigInt, hexadecimal, octal and binary

let decimal = 100

console.log(decimal)

let double = 12.34

console.log(double)

let hexadecimal = 0x12ab

console.log(hexadecimal)

let octal = 0o127

console.log(octal)

let bin = 0b11

console.log(bin)

//Output will be in base 10 form

//all alphabates are case insensitive

let bigInt = 12345739282634929n

console.log(bigInt)

console.log(typeof(decimal))

console.log(typeof(octal))

console.log(typeof(bigInt))

let largest = 1.7 \* 10\*\*308

console.log(largest)

let anotherMax = Number.MAX\_VALUE //is it really maximum number ?

console.log(anotherMax)

anotherMax += 1

console.log(anotherMax)

console.log(typeof(anotherMax))

Boolean

var flag = true

console.log(flag)

var flag = false

console.log(flag)

//var vs let

//Eg01

for(var i = 0; i < 10; i++){}

console.log(i)

for(let j = 0; j < 10; j++){}

console.log(j) //ReferenceError: j is not defined

//Eg02

var data = 100

var data = 200

console.log(data)

let num = 100

//let num = 200 //SyntaxError: Identifier 'num' has already been declared

console.log(num)

//Eg03

var data = 100

{

var data = 200

}

console.log(data)

let num = 100

{

let num = 200

}

console.log(num)

//global polluting issue:- local variables affecting global variables.

//Eg04

console.log(data)

var data = 100

console.log(num) //ReferenceError: Cannot access 'num' before initialization

let num = 200

//Eg 05 var vs const

const data = {

value : 100

}

console.log(data.value)

data.value = 200

console.log(data.value)

//data = {} //TypeError: Assignment to constant variable.

===============================================

Control statements:-

===============================================

- Changing flow of program

Types

i) Decision control

ii)Looping control

iii)Case control

iv)Jump control

===============================================

Operators

===============================================

i) Arithmetic Operators -> eg +, -, \*, /, %, ++, --, \*\*

ii)Relational Operators -> eg ==, !=, >, <, >=, <=,

iii)Logical Operators -> eg &&, ||, !

iv)Bitwise Operators -> eg &, |, ^, ~, <<, >>

v) Assignment Operators -> eg =, +=, -=, \*=, /=,...,<<=,>>=,...

Strictly equal to -> ===

i)Arithmetic Operators:-

Consider A = 25, B = 7, Calculate C

C = A + B it adds A & B, C = 32

C = A - B it substracts B from A C = 18

C = A \* B it multiplies A and B C = 175

C = A / B it devides A by B C = 3.5714285714285716

C = A % B it stores reminder of division of A by B, C = 4

A++ it increments A by 1 A = 26

B-- it decrements B by 1 B = 6

3 \*\* 4 = 81

ii)Relational Operators:-

Consider A = 5, B = 10, C = 5

d = '5'

A == B False

A == C True

A == d True

A === d False

A != B True

A > C False

A >= C True

B >= A True

C < B True

B < C False

iii)Logical Operators

Consider A = 1, B = 0

(A && B) -> 0

(A || B) -> 1

!(A && B) ->true

iv)Bitwise Operators

Truth Table

a b a&b a|b a^b ~a ~b

0 0 0 0 0 1(-1) 1(-1)

0 1 0 1 1 1(-1) 0(-2)

1 0 0 1 1 0(-2) 1(-1)

1 1 1 1 0 0(-2) 0(-2)

2's Complement form = 1's Complement form + 1

1's Complement form = replace 0's with 1's and vice versa

Dec bin 1’s 2’s dec

10 1010 0101 0110 -10

11 1011 0100 0101 -11

12 1100 0011 0100 -12

13 1101 0010 0011 -13

14 1110 0001 0010 -14

15 1111 0000 0001 -15

Shifting

<< left shift -> pad 0's from right

>> right shift -> bit discard from right

Consider C = 4, bin form 0100

bin shift dec

C << 1 0100 01000 8

C >> 1 0100 010 2

5<<2

7<<2

120>>2

5<<2 0101

7<<2 111

120>>2 1111000

5<<2 0101 10100

7<<2 111 11100

120>>2 1111000 11110

5<<2 0101 10100 20

7<<2 111 11100 28

120>>2 1111000 11110 30

v) Assignment Operators

Consider A = 5

Expression Result

C = A 5

C += A C = C + A 10

C -= A C = C - A 5

C \*= A C = C \* A 25

C /= A C = C / A 5

C <<= 2 C = C << 2 20

===============================================

Function:-

===============================================

- a particular logic is called as a function.

- functions are used to reuse the logic.

- There are following type of functions

i) Named functions

ii)Anonymous functions

iii)Constructor functions

- we can pass various type of parameters to functions

i) Rest parameters

ii)Optional parameters

iii)Default parameters

i) Named functions

- the function with name is called as named functions

- Syntax

//defining a function

function function\_name(parameters)

{

//business logic

}

//calling a function

function\_name(parameters)

//Eg01

Create a function

@fun\_one

fun\_one return "Hello...!"

function fun\_one() {

return 'Hello...!'

}

console.log(fun\_one)

console.log(fun\_one())

//Eg02

fun\_one return fun\_two definition

function fun\_one() {

return fun\_two

}

function fun\_two() {

return 'Good morning'

}

console.log(fun\_one) //?

console.log(fun\_one()) //?

console.log(fun\_one()()) //?

ii)Anonymous functions

- the function without name is called as anonymous functions

- these are also called as arrow functions or callback functions

- we can represent arrow functions by '=>' symbol.

- arrow functions introduced in ES6.

- arrow functions are more secure than name functions.

- arrow functions utilize heap memory effectively.

- Syntax

//defining anonymous function

let variable\_name = function(arguments){

//business logic

}

//defining arrow function

let variable\_name = () =>{

//business logic

}

//calling a function

variable\_name(arguments)

//Eg01

create arrow function using following variables

@fun\_one

fun\_one return "I am from arrow function...!"

//anonymous function

let fun\_one = function () {

return 'I am from anonymous function'

}

console.log(fun\_one())

//arrow function

let fun\_one = () =>{

return ' I am from arrow function'

}

console.log(fun\_one())

//Eg02

one arrow function return another arrow function

inner arrow function return 'Have a nice day...!'

let fun\_one = () =>{

return () =>{

return 'Have a nice day...!'

}

}

console.log(fun\_one)

console.log(fun\_one())

console.log(fun\_one()())

===============================================

Constructor Functions:-

===============================================

- Constructor function is used to create a class like structure.

- In constructor function everything(variables and functions)

start with 'this' keyword.

- We can create the object to the constructor using the 'new' keyword.

//Eg01

function class\_one(){

this.wish = `Welcome to counstruct function`

this.myWish = () =>{

return this.wish

}

}

let obj = new class\_one()

console.log(obj.wish)

console.log(obj.myWish())

//Eg02

function class\_one(arg1, arg2, arg3) {

this.sub\_one = arg1

this.sub\_two = arg2

this.sub\_three = arg3

}

let obj = new class\_one(1, 2, 3)

console.log(obj.sub\_one, obj.sub\_two, obj.sub\_three)

let obj1 = new class\_one('Maths', 'SS', 'Tamil')

console.log(obj1.sub\_one, obj1.sub\_two, obj1.sub\_three)

===============================================

prototype

===============================================

- prototype is a property which adds members

dynamically to constructor functions

function class\_one(){}

class\_one.prototype.sub\_one = `Javascript`

class\_one.prototype.sub\_two = `ReactJS`

class\_one.prototype.sub\_three = `Node.JS`

let obj = new class\_one()

console.log(obj.sub\_one, obj.sub\_two, obj.sub\_three)

===============================================

Prototype Chaining:-

===============================================

- Nesting one constructor prototype to another constructor

is called as prototype chaining.

- Concept of inheritance is achieved with prototype chaining

//Eg01 Single inheritance

function class\_one(){}

class\_one.prototype.fun\_one = () =>{

return 'I am from fun\_one'

}

function class\_two(){}

class\_two.prototype = Object.create(class\_one.prototype)

class\_two.prototype.fun\_two = () =>{

return 'I am from fun\_two'

}

let obj = new class\_two()

console.log(obj.fun\_one(), obj.fun\_two())

//Eg02 Multilevel inheritance

function class\_one(){}

class\_one.prototype.fun\_one = function(){

return "I am from Function one"

}

function class\_two(){}

class\_two.prototype = Object.create(class\_one.prototype)

class\_two.prototype.fun\_two = function(){

return "I am from Function two"

}

function class\_three(){}

class\_three.prototype = Object.create(class\_two.prototype)

class\_three.prototype.fun\_three = function(){

return "I am from Function three"

}

let obj = new class\_three()

console.log(obj.fun\_one())

console.log(obj.fun\_two())

console.log(obj.fun\_three())

//Eg03 Heirarchical inheritance

function class\_one(){}

class\_one.prototype.fun\_one = function(){

return "I am from Function one"

}

function class\_two(){}

class\_two.prototype = Object.create(class\_one.prototype)

class\_two.prototype.fun\_two = function(){

return "I am from Function two"

}

function class\_three(){}

class\_three.prototype = Object.create(class\_one.prototype)

class\_three.prototype.fun\_three = function(){

return "I am from Function three"

}

let obj1 = new class\_two()

let obj2 = new class\_three()

console.log(obj1.fun\_one())

console.log(obj1.fun\_two())

console.log(obj2.fun\_one())

console.log(obj2.fun\_three())

//Function Overriding

function class\_one() { }

class\_one.prototype.dbFun = () => {

return "MySql data soon"

}

function class\_two() { }

class\_two.prototype = Object.create(class\_one.prototype)

class\_two.prototype.dbFun = () => {

return "MongoDB data soon"

}

let obj = new class\_two()

console.log(obj.dbFun())

===============================================

Passing parameters to functions

===============================================

1. Default Parameters

- It allows named parameters to be initialised with

default values if no value or undefined is passed.

function fun\_one(arg1 = 10, arg2 = "Fullstack", arg3 = "Angular") {

console.log(arg1, arg2, arg3)

}

fun\_one() //10 FullStack Angular

fun\_one(25) //25 Fullstack Angular

fun\_one(undefined, "MERN") //10 MERN Angular

fun\_one(undefined, 'undefined', undefined) //10 undefined Angular

fun\_one(null, null, null) //null null null

fun\_one(null, undefined, "MongoDB") //null Fullstack MongoDB

fun\_one(undefined, undefined, undefined) //10 FullStack Angular

2. Optional Parameters

- While calling a function no need to pass all arguments.

- We can keep few arguments as Optional.

- Optional Parameters introduced in ES6.

function fun\_one(arg1, arg2, arg3) {

console.log(arg1, arg2, arg3)

}

fun\_one() //undefined undefined undefined

fun\_one("Hello\_1") //Hello\_1 undefined undefined

fun\_one(null, "Hello\_2") //null Hello\_2 undefined

fun\_one(10, 15.7, "MEAN") //10 15.7 MEAN

fun\_one(null, null, null) //null null null

3. Rest Parameters

- It is an improved way to handle function parameters.

- It allows us to represent indefinite number of arguments

as an array.

- Rest parameters represented by '...' ( ... is called as spread operator)

function fun\_one(...arg) {

console.log(arg)

}

fun\_one("Angular") //[ 'Angular' ]

fun\_one() //[]

fun\_one(`Angular`, `Fullstack`) //[ 'Angular', 'Fullstack' ]

fun\_one(undefined, null) //[ undefined, null ]

function fun\_one(arg1, arg2 = "Hello\_2", ...arg3) {

console.log(arg1, arg2, arg3)

}

fun\_one("Hello\_1") //Hello\_1 Hello\_2 []

fun\_one("Hello\_1", undefined, "Hello\_3") //Hello\_1 Hello\_2 [ 'Hello\_3' ]

fun\_one("Hello\_1", "Hello\_2", "Hello\_3", "Hello\_4") //Hello\_1 Hello\_2 [ 'Hello\_3', 'Hello\_4' ]

fun\_one(undefined, undefined, undefined) //undefined Hello\_2 [ undefined ]

fun\_one(undefined, undefined, [1, 2, 3], [4, 5, 6]) //undefined Hello\_2 [ [ 1, 2, 3 ], [ 4, 5, 6 ] ]

fun\_one(undefined, undefined, [1, 2, 3], 4, 5, 6) //undefined Hello\_2 [ [ 1, 2, 3 ], 4, 5, 6 ]

fun\_one("Hello\_1", [1, 2, 3], 4, 5, 6) //Hello\_1 [ 1, 2, 3 ] [ 4, 5, 6 ]

//function fun\_one(...arg1, ...arg2){} //SyntaxError: Rest parameter must be last formal parameter

//function fun\_one(...arg1, arg2){} //SyntaxError: Rest parameter must be last formal parameter

===============================================

Inputs and outputs

===============================================

prompt

let var\_one = document.getElementById(id).value

alert

document.getElementById(id).innerHTML = res

window.open(<filename.html>)

Assignment:-

Design a simple calculator

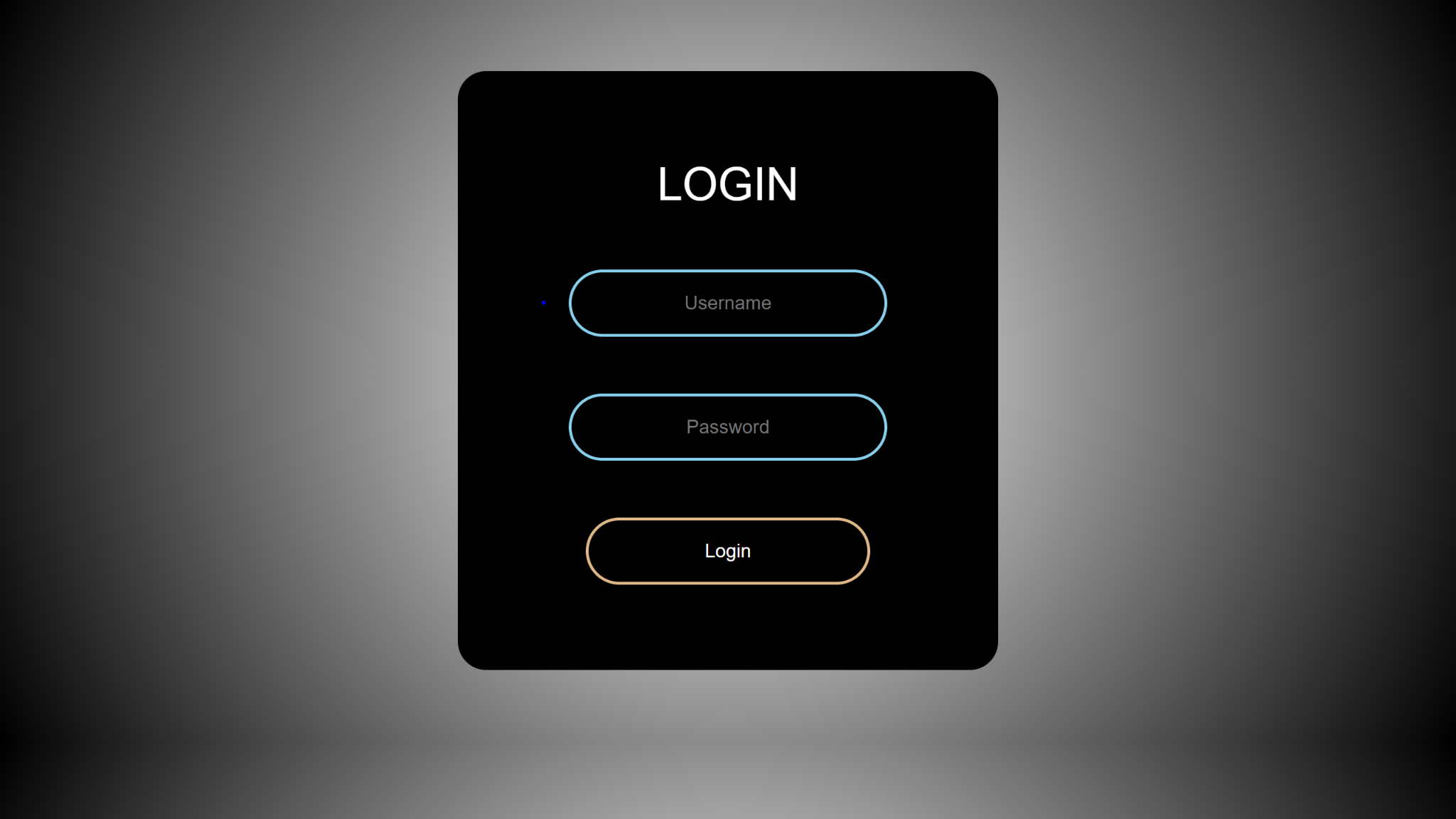
- accept numbers in input

- display output

- operations + - x /

- Apply appropriate styling

Design like this



Callbacks

* Passing one function as argument to another function is called as callback

//Eg01

function fun\_one(arg){

console.log(arg())

}

fun\_one(()=>{

return 'Hello...!'

})

//Eg02

function fun\_one(arg1, arg2, arg3){

console.log(arg1(), arg2(), arg3)

}

fun\_one(()=>{

return 123

},()=>{

return 'Javascript'

},()=>{

return 'ReactJS'

})

//Eg03

function add(num, callback) {

return callback((num + 5), false)

}

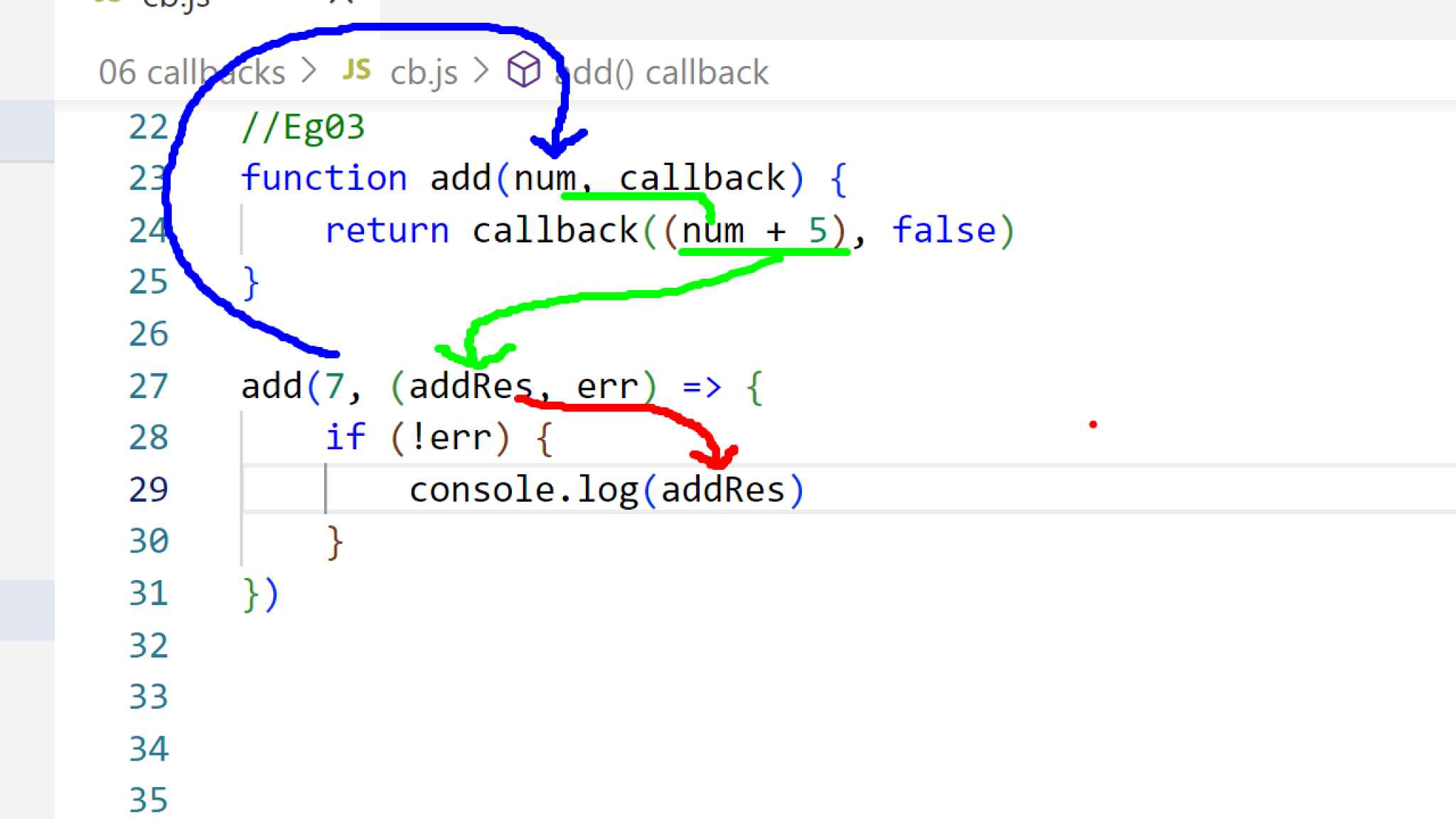
add(7, (addRes, err) => {

if (!err) {

console.log(addRes)

}

})



//Eg01

function fun\_one(arg){

console.log(arg())

}

fun\_one(()=>{

return 'Hello...!'

})

//Eg02

function fun\_one(arg1, arg2, arg3){

console.log(arg1(), arg2(), arg3)

}

fun\_one(()=>{

return 123

},()=>{

return 'Javascript'

},()=>{

return 'ReactJS'

})

//Eg03

function add(num, callback) {

return callback((num + 5), false)

}

add(7, (addRes, err) => {

if (!err) {

console.log(addRes)

}

})

//Eg04

function add(num, callback) {

return callback((num + 5), false)

}

function sub(num, callback) {

return callback((num - 3), false)

}

function mul(num, callback) {

return callback((num \* 2), false)

}

function div(num, callback) {

return callback((num / 2), false)

}

add(5, (addRes, err) => {

if (!err) {

sub(addRes, (subRes, err) => {

if (!err) {

mul(subRes, (mulRes, err) => {

if (!err) {

div(mulRes, (divRes, err) => {

if (!err) {

console.log(divRes)

}

}) //div

}

}) //mul

}

}) //sub

}

}) //add