JS object

→ group of key: value pair of key values pairs

let emp = { id : 01, Name : "Lalitha", Sal : 45000 } — properties

→ group of properties as are entity

→ duplicates keys are not allowed.

indexing concept is not applicable

↳ object is not iterable object

JS array

→ group of element values an are entity

→ let id = [101, 102, 103, 104]

iterable object — array

iterable

for()

do()

while()

for (-) let ids = [101, 102, 205]

C	R	U	L
let emp = { id : 101; name : "lali"; Sal : 45000	clg (emp); object . properties; emp . id — 101; emp . name — 45000; emp . Sal; clg (emp . email) — undefined	emp . Sal = 55000; emp . x = "wayand	

Read

let = ids [101, 102, 103, 104]

0 1 2 3

Read

clg (ids)

clg (ids [0])

clg [ids [3]]

ids [3] = 1104

100

let emp = {

id : 101

name : "Lalitha"

email : "la@com

}

// how to read object and object prop

console . log (emp)

// read all properties

console . log (emp . id)

console . log (emp . name)

(emp . email)

do { }

do { }

let emp = {

id = 101

}

console.log (emp)

emp.name = "Lalitha";

emp.sal = "45000";

console.log(emp)

id: 101

Company name : "Taacs

delet emp (id)

clg.(emp)

[id: 101, name: "",]

if let enames = ["Rah", "Sonia", "priya"]

[0]	1	2	[3]

i = 0

in = i

cond = i <= 3

Inc = i++

for (let i=0; i<=3, i=i+1)

clg

for (ename of enames) {

clg ('ename)

}

let

~~for of object~~ — withoct of insuction of condition

for of

let numbers = [10, 9, 8, 4, 6, 54, 3 21]

for (x of number) {

consol. log ('x);

}

write values

namy

different

type of print.

for [of]

Ex:-

x

employ

employ

let var = ..

while(i=0 ; i<= var.leng-1)

{

console. log (var.leng[i])

i++;

}

Function is a reasuble:-

* what is a function?

group of statement specify specific function purpose

{

① function → key values

② in working

wish (③) { ④ - block

code is reasubility

clg ("hi")

clog ("hi")

declaration

function are methoden

both are equal

declaration switch: ON: off

code is resusability

parameters.

function add ( ) {

clg ("add")

}

add()

add()

function (a, b) {

temply values.

receiving the argument

cay (a+b)

}

add (10, 20)

parameters.

passing a argument

add (2000, 3000)

add ("10, "Rai")

add (10);

not a number

function wisch ( ) {

console.log ("hi")

conde ( )("GA")

} cons ( ) ("hi")

wisch()

guest first function is note there.

{

}

d [i]

let i = [ ]

let ename = ['rahul', 'gandhi']

delete ename [0]

(ename)

del i (i0)

(on, log (i)

pu

① function add (a, b) {
consol. log (a+b)
}
add (10, 20)
add (10, " Rahua")

---

② function cal_ Age (year) {
console. log (2022 - year)
}
cal - Age (1983)

---

let user = {
id: 101,
Name: " ",
color: "blue"
Salary: 15000,
Location: banglor"
}

for(~~let~~ in ~~Name~~)
if (fal) { }

user
solution
cons. cons( ) ; i<= user.location length)
~~Lets to~~ for(i=0 ; i++)
console. log console. log (location (
[i]

user
for(let i=0; i<=
i++)
company:
console
(compa
Consolelog (

153

1×1×1 = 1

5×5×5 = 125

3×3×3 = 27

practice

153

Initialize use = for 153 153

let sum = 0;

const number = prompt ("Enter a three-digit positive integer:");

// create a temp variable

let temp = number;

while (temp > 0) {

condition // finding the ones digit

let reminder = temp % 10;

10) 153 (15.3
150
30
30
0

sum

sum += reminder * reminder

15.3

// removing last digit from the number

3 5 3/2

1

15/10 2

} // check the condition

if (sum == number) {

1.5

}. Console.log (${number} is amst");

else {

Console.log (${number} is an ans)

2/12/2022

Javascript class

groupofeddembussing logica sector the value

① without

```
function add( ) {
    clg("addition")
}
add
```

② parameter

```
function add (a, b){
    clg (a+b)
}
add (10 20)
```

argument

add (1,2)

③

```
function add (a, b, c = 100)
    clg (a+b+c)
}
add (10, 20, 30)
add (1,2) ; 103
add (1, 1, 1)  3
```

default values

```
function add (a, b) { }
    return a+b;
}
Let r1 = add (10, 20).
    clg(r1)  300
Let r2 = add (100, 200)
    clg (r2) out 300
    clg (add (1, 2))
```

return

30

inworking function

~~add (a, b)~~

```
function wish (message) {
    return "Hello:" + message
}
let r1 = wish ("Good morning")
let r2 = wish (Good-afternoon)
con.log(wish ('Nigh');
```

function

```
con.log ( add (10, 20)
clog. (add ('lalli') )
```

return
↓
capture the value

```
function Cal - age (name) {
    let no_of year = 2022 - year / cons.log (no.of_year)
    return "Employ. Name" : " + Name + " age" : + 2022-year
}
let r1 = cal- age ("Noram", 1983)
con.log (r1)
let r2 = cal - age ("deep:", 1998)
con. log ( r2 )
```

A function is a there are exec that is import / export (doubly pickles) Modularity

Java script - function Road map (Learning

function => innerfunction / neastion function => anonymous function / nameless function => fatarow / Arrowfun

assync program

Callback fun
↓
ansyc await
↓
closer and promise

higher order component.

return statement data

r/b come out from note going execute.

```
Function geta data ( ) {
    clg ("one")
    clg .log ("two")
    return "three
}
con.le. (four )
Let r1 = getdata
```

Arrow:-
Function

```
let contacts = [ "a", "b", "c", 'd' ]
for ( contact of contac )
x
    clg ( contact )
}
```

string

```
let contact = {
    id: 101
    name: " "
}
clg(contact.id)
clg(contact.name)
```

object

```
let contacts = [ { id: 101, name: 'Rahul' }, { id: 202, name:
for ( contact of contacts )
    clg ( contact.id ) {
}
Console.log(contacts[0].name
```

key values → object pairs

(singular)

Markarrow:-

id

JSON
"id": 101
"name":

```
for (let i = 0; i <= user.length -1 ; i++)
    Console.log(users[i].name)
```

```
{
    "name":
    "price"
    "qty":
}
```

ECMA Script — Orgnization

↓

Who started JS

what is Method ?

a = [1, 2, 3, 4]

Con.a[a] = " 4 "

a[3] = 4

a.name =

let a = [10, 20, 30] 40 last

push(40) → Add last

unshift(25) → Del last Del

Con.log(push) → unshift add first

Conlog Con(2)

10 20 30 40

25 10, 20, 30, 40 Del first

0 pop

shift

Delet a[0]

Con(0)

4/12/22

JS array — iterable object
- for
- while
- do
- for of

JS object — Non iterable

for in

${ }
- string Letter
- ~ backtick

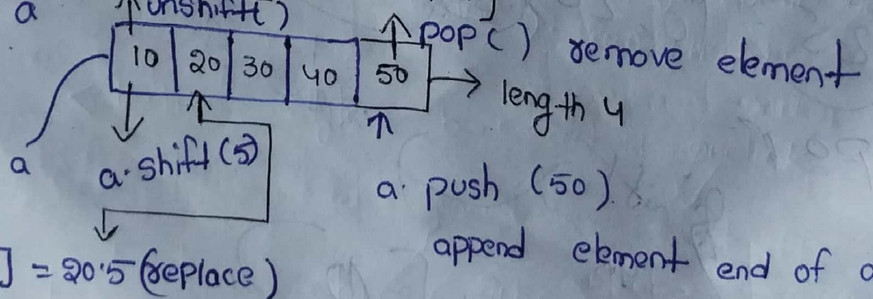
```
Let ename = 'Rahul';
clg ("employe Name:", ename);
clg ( ~ employe Name: ${ ename} ~)
```

```
for (person of persons) {
console.log ("person Name: $ (person.first_name} -
        Email Id : $ {person.Email})
```

array is one class → inbuild array

array is stored in value → index value

let a = [10, 20, 30, 40]



a[1] = 20.5 (replace)

fill (4, 2)

10	20	30	40

10	20	4	4

```
let index_value = a.index
        value ["Rahu"]
co(index_value)
```

let a = [10, 20, 30, 40]

let new_length = a.push (50)

console.log (new_length (a.push))

slice is particular position

Let ename = [10, 20, 30, 40, 60, 70]

con.log (ename . slice (2, 4))

type = '/da

tex = . tex

JS event

- Mouse,
  - onclick
  - ondblick
  - on mouseover
  - on mouse move
  - on mouse
- form
  - onchang
  - on Input
  - on Key pres
  - on blur
- Bo

index.html

data.JS

```
<   >
<head>
  <Script src =          type =          >  <   >
</head>

<body>
  <button onclick = " display () " > test </button>
```

function display() {
}

onclick → event

display() → event handle.

```
<body>
let user = {
    id : 101 , Name : " Rahul "
for (prop in user) {
    console . log ( user [ prop ] )
}
```

→ object but not use

101

Rahul

console. log (user. prop)

id

Name

copy array

constant array1 = [1, 2, 3, 4, 5, 6]

const copyof array1 = array1.S

{tag} a update.

Document object Method

↓

<P id = 'abc' > | </P>

Need to write JS, update <P>

document . get element pn Id ('abc'). innerHTmL = "Hi

property

index.html

<html>

<head>

<script src = "/one.JS type = 'text /Javascript>

</head>

<body>

<p id = 'abc' > </P>

</ document.getelementId ('abc')

document.getelementId('abc'). InnerHTmL = new Date().toLocaleTimeString()

< button style = 'b-c' id 'id:abc' onclick = "——"

event

event holde

function change (abc)() {

document.getelement Id (abc) . style = abc

}

function wish() {

return

}

let x = wish()

dg(x)

Ap

Event

Mouse | from | Brows

```
<html>
<head>
</head>
<body>

<button style = "background - color : red;" >Gm</ /button>
        onclick = " change color ()"
        id = " btn '
< script src = '. /one.Js ' type = 'text/ Javascript ></ /sr>
```

<u>one . Js</u>

```
function color change ( ) {
  document. get element Id ( btn ' ) . style. bc = 'Green'
```

function change

<button stoap>

<button>

events

dom

value

User name [Rabu]

```
form change case ( ) {
let name = document . get
Id('uName') . value =
```

$\boxed{\text{name . to uppercase}}$

```
<html>
<head>
<lable>
<label> Blur ex </label>
<input type = 'text' id = 'uName' />
onblur = 'change case'
<script src = '/from data . Js typ = text/J>
document . get ement Id ('uName') . value = name to uppercase(  </sr>
```

we have write a only anything / write Javascript

document. getelmentbyId ('abc'). Inner html =

new Id " Hello Good "

document. getelement ById ('abc'). Inner HTmL = new Date( ).
toLocale time string ()

7/12/22

Let employ

```
<>
<head>
</head> <script>  <'> trig ha executed
                  browes time of
<bod onload = 'display()'>
      event        data
<table border = '1' >

    <table>

      <thead >  <th> Id </th>

                <th> Name </th>

                <th> salary </th>
</thead>  <th> Company </th>

<tbody id = 'abc' >

    <tr>
        <td> 101 </td>

        <td> lovame </td>

        <td> Salary </td>
    </tr>

</tbod>
```

data.JS

using about backtick

```
let employes = [ { id:101, name : Rahul , Salary: }
                 {                                  }
                 {                                  }

- function display ( ) {

    let row = " ";
    for (employ of employes) {

      rows =
      row + '<tr> <tr >${user.id} </tr>
      </tr>'
      document.get element Id ('abc').innerHTML = row
```

```
let rows = " "
let i=0;
while (i<=user.length-1) {
row = row + <tr>
<td >${user[i].id} </td >
<td>${user[i].gender.substring[0,1]}<
<td>${user[i].email
```

email

(user[i].email.indexof ('@')+1)

pagenation boost record. Javascript

dumy data mack

application
react
Node JS

required. (:/ data)

blance work pagenation.

8/12/22

inner function program.

```
function outer () {
    let a=10;
    let b=20;
    clg (a+b);
    function inner () {
        clg ("tody is - ");
    }
    inner();
}
outer();
```

→ closure

→ fat arrow/arrow fn
↓
Call back
↓
promise
↓
async wait
+
Closure

HoC

Asynchronus
↓
programing

fat arrow function:- alternative native way to write function

fat arrow is we have write code short line

```
function add () {
    clg(addition)
}
```

① ② ③ ④

```
add()
add()
```

```
let a = 10;
let add = function () {
    clg ('Addition')
}
add()
add();
```

is holding value 10

variable holding 10.

() → { } Java Lamda

() ⇒ { }

() ⇒ { }

```
let add =( ) => {
    clg( addition )
}
add();
```

```
let add = (a,b) => {
    con(a+b)
}
add(10, 20)
```

```
function add( ){
    colg (a+b)
}
add (10,20);
```

→
```
Function add (a,b) {
    ~~clg~~(a+b+100)
}
add (10,20)
```

→
```
let add = (a,b) => {
    return a+b+10
}
let r1 = add (10,20)
clg(r1)
```

Fat arrow

→
```
let add = (a,b) => a+b+100;
let r1 = add (10,20);
clg (r1)
```

→
```
Function add (a){
    return a+100
}
~~add (10)~~
Let r1 = add(10)
(log (r1);
```

→ f/r
```
let message = msg => "Hello" + msg;
clg (message ("Gm"))
```

funtic
```
function mess ( msg ) {
    return "Hellow" + msg
}
Consol log ("Gm" )
```

1. consise / short - code
implicit Return
This - return = { }
overriding conflicts &
Var declar

→ function over writing

→ fatarrow:- alterative way to fatarrow

~~Var wish()~~ c

```
Var wish = ( ) => {
    clg('Gm')
}
wish( )

Var wish = ( ) => {
    clg('GN')
}
wish();
```

out:- Gm
GN

```
function wish ( ) {
    conlg('GM')
}
wish( )

function wish( ) {
    Conlg(GN)
}
wish( )
```

out: GN
GN

9/12/2022

OOP :-

To solve realword problem?

- using object
  ↓
  Methodlgy
  ↓

class is a object
↓ template | plan | desing → to construct object

```
class account {
    // is templet / plan / desing
}
```

new account ( )
new account ( )
new account ( )

→ to construct object → memory location allowed

running in the node JS

Var	→ P
function	→ M
class	→ P+M
:	
object	

empty object

```
class account {

}
let a1 = new account();
let a2 = new account();
```

cdg(a1)
clog(a2)

```
→ class account {
    MPn-Balance = 500;
    acc-Name = "Rahual"
}
let r1 = new account();
    r2 = new account();

    cong(r1)
    cong(r2)   cong(r1. min-Balance);
    cong(r1. account-Name);
```

account { min-Bal : 500, account-Name : 'Rahul' }

"
"
500;
Rahual;

```
class account {
    Min- Bal= 500;
    open - account(); {
        console.log ("deposited amount
    }

    deposit - amount () {
    }

    get - Bal( ){
        display Bal
    }

    get - st ()
        No cash come tomorrow
    }

    widthdraw - amount() {
        not possible pay Gst
    }
}
let a1 = new account();
let a2 = new account();
elg (a1. min- Balon); 500
    clog ( a1. oppen account);
```

```
class employe {
    e_Id : 101
    e- Name: " Rahul"
    e- Sal : "4000"
}
let r1 = new employ ()
let r2 = new employ()
let r3 = new employ()
cog(r1)        cog(r1.e-Id)
cog(r2)
cog(r3)
```

12/12/22

OOPS :-

```
class Account
    min-bl : 500;
    acc-bl : 0;
    deposit-Amount(Amount){
        This.ac-Bal = this.acc-Bal + amount
                                     5000
    }
    get-Ball ( ){
        return this.acc-Bal - this.min-bal
    }
}
```

pure-function | impure-function

this - keyword / pointer to current obj

{ min-Bal 500 acc-B 0 }

{ minBal 500 account 0 }

class to refer used this

```
let a1 = new account ();
Let a2 = new account ();
    a1.deposit-amount(5000);
    a1 deposit-amount (4500)
    a2 deposit-amount (50000)
```

```
let a1 = new account()
let a2 = new account()
console.log(a1)
console.log(a2)
console.log("*** after Deposit ***
```

```
let a1 = new account();
    a2 = " " ();
console.log (a1)
console.log (a2)
```

```
class Account
    min-bl : 500;
    acc-bl : 0;
    account-Name : " ";
```

class → var-pro, fun method

m+p.

object

```
class account {
    account-Id : 101;
    account-Name : "Rahul"
    deposit-account( ){
        consol.log(
}
Let L1 = new account( );
```

constructed methode special object method

```
class account {
    acc-Name;
    acc-Id;
    acc-Bl;
    construct (a, b) {
      This. acc-Name = name
      This. acc-Id = Id
      This. acc-Bl = amount
    }
}
```

executed to automatical at the time object creation

we can't construct method execut esplicity

```
Let a1 = new Account (101, 'Rahu', 5000)
Let a2 = new Account (102, 'Priya', 500)
console.log (a1)
console.log (a2)
```

```
class employee {
    Construct (id) {
      This. emp-Id = id
      console. log ("No Rain")
      This. emp-Name = Name
    }
    get_details () { }
}
```

this point current working

construction is automatically used

```
Let e1 = new employe();   101, "Rahul"
    e2 = new employe();   102, "dilip"
    e3 = new employe();   undefined
    e4 = new employ ();   undefined
    e5 = new employe();   undefined.
```

→ note write

class
object
function
oops

14/12/22

Inference :-

adding the new feature (a extning)

code reausing the existing function.

reausability

→ code is reausability → main

Interitance

→ Inheriting properties from one class to another class
→ Inheriting properties from parent class to child class
→ Reausing existing class functionality
and adding extending new feature

```
class parent {
    asset = '10CR' parent
}

class child extends parent {
    qualities = 'Good'
}
```

new features

keyword.

```
let G = new child();
dg(G);
```

{ asset: "10CR" qualities }

objectis is a memory location.

Ex:-

Main-lence the relastion is extends

```
class A {
    m1() {
        console.log("class A - m1 method")
    m2() {
        console.log("class A - m2 method")
    }
}
class B extends A {
}
let a1 = new B()

a1.m1()
a1.m2()
```

Ex:-
overriding inheritence.
method
child

React

promp immutable

compunicat.object

State—Javascript—
objec → imutable.
compositin our own Data
class

```
class A{
    m1 (){
        clg ('A' - m1())
    }
}
let a1 = new a( );

class B extends A {
    m1 (){
        clg ('B- m1()')
    }
    m2 (){
        clg ('3 - m2 ()')
    }
}
let a1 = new B( )
    a1. m1( ) // (B- m1)
    a2. m2( ) // (B-m2)
```

Ex:-

```
class A {
    constructor ( ) {
        cons.log ("A- const - special")
        super( )
    }
    console
}

class B extends ( ){
    constructor ( ){
        super
        cons.log ( )
}
let a1 = new B( )
```

invoworking the object values supper class

parent clas.

super( ). get_data

```
class A{
    m1 (){
        clg ('A' - m1())
    }
}
let a1 = new a( );

class B extends A {
    m1 (){
        clg ('B- m1()')
    }
    m2 (){
        clg ('3 - m2()')
    }
}

let a1 = new B()
    a1. m1() // (B- m1)
    a2. m2() // (B-m2)
```

super ( ). get_debit

Ex:-

```
class A {
    constructor ( ) {
        cons.log ("A- const - special")
        super()
    }
    console
}

class B extends ( ){
    constructor ( ){
        super
        cons.log ( )
    }
let a1 - new B ( )
```

object values

invoworking the supper class

parent clas

```
class A {
    get_details() {
        console("Gm")
    }
}

class B extds A {
    Con-to () {
        supper();
    super.get_details()
    }
    get_Reddy () {
        supper.get_details()
    }
}
new B( ).get_Ready( )
```

inworking the parent class members

what is ECMA?

ECMA computer mani associat

startand scripting programing langauge..

python software

orack

syn

psf — Java — ECMA

p.1 — SPL

es 5

p.2 — 5 — es 6

p.3 — 6 — es 7

es 8

internal algorithm

chrome Google — v8 — es

m.f m — v8 — es

c mw — v8 — es

saf Atsyle — v8 — es

→ every body understand the web browser HTML

Scanned with OKEN Scanner

17/12/22

→ ES6 →

Tranpiler → Babel presenting Javascript invironment → Laborery

↓

upp
dow } grading programing language

ES6 → Comparability

```
let a=10
let b = 20;
Export { a, b }
```

ES6 new feautre

Compile Interpreter

high level language understand

Low level language.

protefule
gitub hub. io
how

→ Let, Const,

→ var, class

→ Fat arrow function alternative. syntax

→ array supporting method

[ ]. Map( )
[ ]. foreach( )
[ ]. reduce ( )
[ ] filter

ES6 → global scope
→ The declaration is possible
one function as arguments

→ ....
spread rest

→ object destructring

→ ${ } — string Literal

→ Asynchronous programing using JS

1. Call back
2. promise
3. async & await

→ closure

→ set

Global
var a;
const var a=10;
const var a=20;
modefy var a = Rabul

class

let a;
let a=10;

→ function/bo scope
→ Re-declarat is note possible

* construct

To instalization object values

array → iterable
object is a note iterable

→ let enames = [" , " , " , "]

for (var i=0; i<= enames.length, i=i++] {
clog.(enames)
}

while

Ex

→ let enames = [" ", " ", " "]
// create new array based on existing //
let new — enames = [ ];

upper grading

for(ename of enames) {
new - enames.push ("Gm");
new - enames.push (ename. toupper case( ))
}
c.log (enames)
clg (new - enames);

GM GM GM

return is something

Ex:- Map( )

Map is

let enams = [" ", " ", " "]

enames. map ((enames) => {
clg (ename)
});

function add ( ){
}
add (10, 10)

passing parameters

→ enames. for each ((ename) => {
calg (ename)
})

Ex:- let enames = [" " " " "]

execut function

enames . map ( ) => {

iterative array

clg ('Gm')

});

→ iterate array, execute your function provided argument

enames. map ((enames) => {

clg (ename)

});

Ex: enames . for each ((ename) => {

return undefined

clg(ename)

})

→ Map is iterative array execut the function
return value is a new array

→ for each is iterative array execut the function
return value is a undefined

Data source
let employees = [ ]

inside function
function display_data () {

execute

employ . map ((employee) => {

let rows = " ";

Iterative array

rows = rows + ~ <tr>

<td> ${ employee.Id } </td>

~ </tr>

}

document . get element Id ('abc') . innerHTL = rows;

Ex: Let [1, 2, 3, 4, 5].push(6);

passing the array

Ltet a = [1, 2, 3, 4, 5].map((x, index) => {

return x+1;

})

clog(a)

[2, 3, 4, 5, 6].

["bilip", "mano", "vasu"].map((name) -> { return name.tolocaluppercase

Ex: let car = [{}.]

let new-car = [ ]

for (car of cars)

if (car.color === "white") {

new-car.push(car)

}

}

console.log(car)

Ex- car-> is object filter

let new-cars = cars.filter((car) => {

return car.color === "white.

})

→ let new-car = car.filter((car) => {

return car.price === 50,00,000.

})

console.log(new-car)

Triple dot

spread

[ ] [ ]

rest

function - param

let Marks = [35, 36, 37, 38, 39, 40]

```
let a = [ ]
for ( mark of marks ) {
    a.push ( mark )
}
```

1) copy

Let marks = [35, 36, 37, 38, 40]

let a = [... marks]

applicable array and object

2 extending array element → extract & Copy → spread

let a = [10, 20, 30, 40]

let b = [... a, 50, 60, 70, 80]

let a = [10, 20, 30, 40] ; let b = [50, 60, 70, 80]

3. let c = [... a , ... b]

let marks = [35, 36, 37

let a = [10, 20, 30]
b = [40, 50, 60]

Let c = [... a, ... b]

/* for (ele of a) {

c. push (ele)

}

for (ele of b) {

c. push (ele)

}*/

console. log (c)

→ let

user = { uid : 101;
name : 'Rahul'
email : ' @ .com

}

let details = {

email : ' rahu@ '.

sal : 45000
loc : Bangl '

details }

Let user details = { ... user , ... details }

let a = {
uid : 101,
name : 'Rahu',
email : " Rahu@ gmail.com" }

user = { ... a , email : " Rahu@ gmail.com" }
console. log (user)

let details = { email: 'rau@ com" , sal : 45000 , loc : Nayada" }

uid : 101 ,
name : 'Rahul'
email : 'ran
sal : 45000;
loc : 'wayada'

Callbacks

→ A Callback is a function passed as an argument to another function → this technique allows a function to call another function

→ A call back function can run after another function has finished.

→ My calculator Ex.

```
function mydisplayer ( some) {
    console. log ("some")
    document. getelementById ("demo"). Innerhtml
        = some;
}
function mycalculator (num1, num2){
    let sum = num1 + num2
    return sum;
}
let result = mycalculator (5,5)
mydisplayer(result)
```

output :

parent

```
class A {
    constructor (x, y) {
        This.x = x;
        This.y = y;
    }
}
```

→ constructor :-

super Calling in parent

invorking parent Call

child

```
class B extends A {
    constructor (a, b, c) {
        super (a, c)
        this.b = b;
    }
}
```

b1

```
let b1 = new B (10, 20, 30)
```

parent

```
class Account {
    constructor (id, name) {
        This.account_Id = id;
        This.account_Name = name;
    }
}
```

Saving

master account

Current account

child

```
Class saving_Account extends Accounts {
    constructor (id, Name, amount) {
        super (id, name);
        this.amount = amount;
    }
}
```

```
new saving_Account (101, "Lalitha", 50000)
```

```
class Saving_Account extends amount( )
    constructor (id, name, amount){
        super( id, name)
    }   this.amount = amount

}

let s1 = new Saving - account (101, "name", 5000)

console.log(s1)

class
```

Ex:

```
class A {
    m1() {
        con.log (m1)
    }
```

c

dirved call

```
class B extends A {
    constructor (a, b)
        super()
    }
    m1() {
        // super.m1()
        console.log ("class-B: m1 ()")
    }
    m2() {
        // super.m2( )
        console.log ("class-B: m2()")
    }
}
Let b = new b (10, 20)
```

Ex:

```
class A {
    constructor (c) {
        this.c = c
    }
}
class B extend A {
    constructor (a, b, c) {
        super(c);
        this.a = a;
        this.b = b;
    }
}
let b1 = new B (10, 20, 30)
```