DATE

(8). Scopes :-
Scope defines the visibility and accessibility of a variable.

↳ We have two types of scope :
1. Global scope →
(i). variable declared in global scope can be accessed anywhere in the program.

(ii). It has the highest accessibility.

2. Local scope :-
(i). variable declared in Local scope can be accessed in that block only. We can't access a variable from outside.

(ii). JS engine create Local scope for functions and blocks.
↳ Function Local scope →
(a). Local scope created for function is known as function scope.

(b). variable declared in function scope can't be accessed from outside.

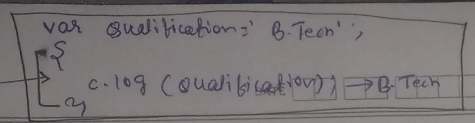
(c). variable declared with var, let and const all are locally scope inside a function.

↳ Blocks Local scope →
(a). Local scope created for block is known as block scope.

(b). variable declared with let and const

PAGE

Local scope.



inside a block are locally scoped.

(c). But variable declared with var are accessible from outside of block.

GIEC :-

(1). Everything we write inside javascript it will happen inside an execution context.

(2). When we give js code to js engine, it creates a global execution context, where js code get executed.

(3). In GIEC, javascript runs two phase →

(i). Variable phase :-

(a). In variable phase, js engine allocate memory to the variables.

(b). To allocate memory javascript engine search for a variable declaration statement.

↳ In var case

(c). The time when memory block is created javascript engine stores undefined in it.

Note :- for let and const declaration javascript engine will not store undefined in its variable phase.

(ii). Execution phase :-

(a) In this phase, all the instructions

DATE

get executed in the order (top to bottom) they are available in global execution context

↳ GIEC (global execution context)

① Variable phase
↳ memory creation phase
↳ variable declaration statement.

② Execution phase
↳ variable initialization
↳ console statement
↳ function call statement.

eg: (1). Questions on scope :- ↳ using GIEC :-

```
① var a = 10;
② let b = 20;
③ const c = 30;
④ {
⑤   var a = 100;
⑥   let b = 200;
⑦   const c = 300;
⑧   c.log(a);
⑨   c.log(b);
⑩   c.log(c);
⑪ }
⑫ c.log(a);
⑬ c.log(b);
⑭ c.log(c);
```

GIEC → memory creation

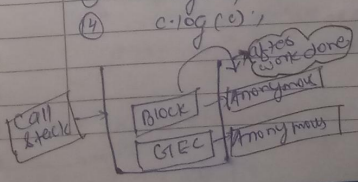
V.P	E.P
a undefined → 10 → 100	① a=10
	② b=20
script	③ c=30
scope	④ {
b 20	⑪ }
c 30	⑫ c.log(a) → 100
	⑬ c.log(b) → 20
	⑭ c.log(c) → 30

global scope

Block/Local Execution Context

V.P	E.P	garbage collected
a	a=100;	
b	b=200;	
c	c=300;	
b 200	c.log(a) → 100	
c 300	c.log(b) → 200	
	c.log(c) → 300	

local scope and in the script scope not here.



DATE

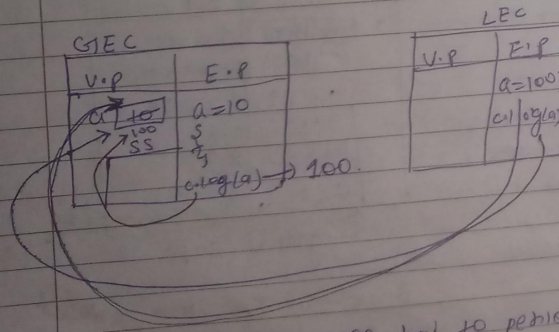
Note:- Local Scope execution context me jab `var a=100;` likha tha tab ye ayega a declare nahi hoga jab `var a=10` pehle hi GTEC me declared hai.

(2) Jab LEC `c.log(a)` or `a=100;` kr raha tha tab LEC pehle apne V.E me dekhega ki koi 'a' naam ka variable hai ya nahi. agr usko nahi milega to wo GTEC me jaake dhundega.

(3).

```
let a=10;
{
    let a=20;
    c.log(a); ⟹ 20
}
c.log(a); ⟹ 10
```

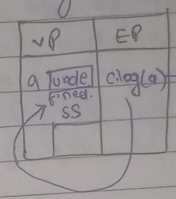
```
let a=10;
{
    a=100;
    c.log(a); → 100
}
c.log(a); → 100
```



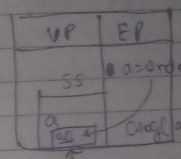
- Scope and or a=100 hai to pehle ye dekhega isi local me ki koi 'a' declare hai ya nahi fir ye global me check karega.

DATE

(4). `var a;`
`c.log(a);` → undefined



`let a;`
`console.log(a);` → undefined



↳ JS engine execution ke const phase me let se declare variable me undefined initialize kar deta hai.

Hoisting in javascript:-

- The ability of javascript engine to access a variable before its declaration statement.
- variable declared with var, let and const all can be hoisted.

Note:- • var are hoisted with with a default initialization
• in let and const how hoisting happen →

```
console.log(number);
let number=10;
// or const number=10;
```

↳ ye ReferenceError: Can't access 'number' before initialization.

```
console.log(number2);
let number=10;
// or const number=10;
```

↳ ReferenceError: number2 is not defined.

DATE

↳ ek error aarena ki aap variable ko access nahi kar sakte, ~~declare~~ declare se initialization se pehle ps. yahan pr javascript ko ye pta hai ki 'numbers' variable aage initialize hua hai and 'numbers' variable ke baare me pta hai usko. or ye isiliye pta hai kyunki 'numbers' hoisted hua hai, top of global scope ps.

↳ ps doosra 'numbers2' variable ke baare me javascript ko nahi pta hai or ye sahi v hai kyunki hamne usko aage v nahi declare kiya hai or nahi initialize.

↳ To ye hoisting behaviour hai var, let/const me. var hoisted hota hai ek default value ke saath pr let/const bina kisi default value ke hoisted hota hai, isiliye wo error throw karte hai ki can't access before initialization and var 'undefined' return karte hai. (undefined)

eg:
1. var gameName = 'GTA 5';
2. C.log(gameName) -> GTA5

Top
1. C.log(superHero); -> undefined (var ke case me)
V.P | E.P
2. var superHero = 'yogi ji';
3. C.log(superHero); -> yogi ji
To
Bottom

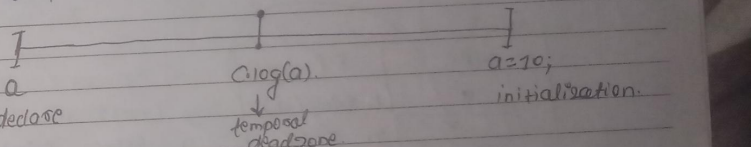
DATE

GEC

V.P	E.P
superHero: undefined → yogi ji	1. C.log(superHero); → undefined 2. superHero = 'yogi ji'; 3. C.log(superHero); → yogi ji

⇒ Temporal dead zone :- (error name).
It is a time frame b/w variable declaration and variable initialization. In this time frame, we can not access the variable

variable declared with let and const belongs to temporal dead zone.



eg: (i) for let →
1. let favTeam;
2. C.log(favTeam);

GEC

V.P	E.P
	1. favTeam
SS	2. C.log(favTeam); → undefined
favTeam: undefined	

let wale variable me E.P variable me store nahi hai.

DATE

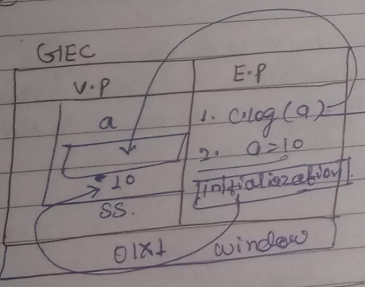
Hoisting in let & const

There's some time period during execution where let/const variables are hoisted but not accessible; it's called the TDZ.

eg- ①
```
c.log(number);
let number=10;
c.log(number)
```

number variable is in a temporal dead zone where Javascript knows of its existence (b/c its declaration is hoisted) but it's not accessible (as it doesn't have an initialization.

② ①. `c.log(a);`
②. `let a=10;`



→ Error: can't access 'a' before initialization (TDZ).

don't know why sir had written this.

→ we can't redeclare a variable with same name in same scope.

eg- function sayHello{
```
var a=10;
let b=20;
const c=30;
}
```
→ locally accessible.

```
var a=10;
{
var a=100;
var b=200
}
```
```
var a=10;
var a=100;
c.log(a) → 100
```


DATE

more differences b/w let const var :-

	var	let	const
(4).	variable declared with var goes to global scope.	variable declared with let is block scoped.	variable declared with const is block scoped.
(5).	we can re-declare variable with same name within same scope.	we can't.	we can't.
(6).	we can re-initialize the value of a variable. eg→ `var course='html'; course='css';`	Here also we can re-initialize	Here we can't re-initialize.
(7).	we can declare variable without initialization	Here also. eg→ `let age;`	Here can't. eg→ `const a;` → ✗
(8).	JS engine will store undefined in its variable phase.	Here JS engine store undefined in its execution phase after reading declaration statement.	same like let.
(9).	variable declared with var can be hoisted.	It is also can be hoisted.	can be hoisted.

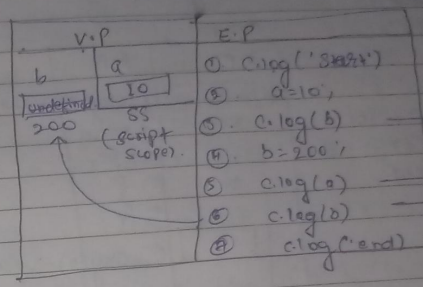
(10).	var does not belongs to TDZ.	variable declared with let in hoisting belongs to TDZ.	Same like let.
(11).	variable declared inside a block will go to global scope.	variable declared with let does not belongs to global scope, We can't access them with the help of window variable.	same like let.

eg→
```
var a = 5;
{
  var a = 100;
  var b = 200;
}
```
c.log(a) → 100
c.log(b) → 200

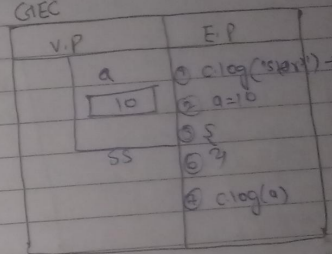
(12).	variable declared inside a function will not go to global scope. It will be accessible inside function only.	variable declared inside a function will be accessible inside a function only.	Same like var and let.

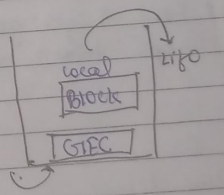
⇔ (1). Questions on Scope:- (using GIEC)

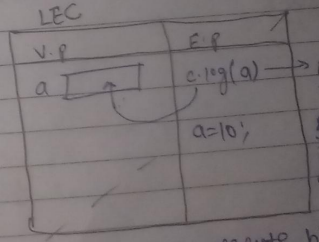
```
1. c.log('start')
2. let a=10;
3. c.log(b);
4. var b = 200;
5. c.log(a)
6. c.log(b)
7. c.log('end');
```

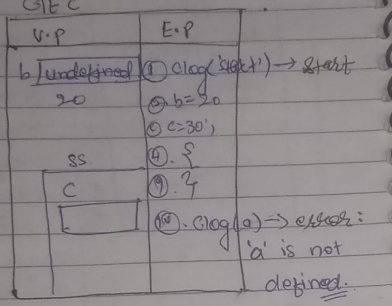

```
1. c.log('start')
2.   let a = 10;
3. {
4.     c.log(a);
5.     let a=10;
6. }
7.   c.log(a)
8.   c.log(b)
9.   c.log(end)
```

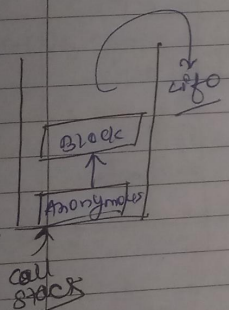


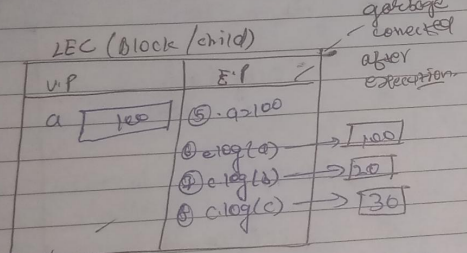




(3).
(1). console.log('start');
(2). var b = 20;
(3). const c = 30;
(4). {
(5). let a = 100;
(6). console.log(a);
(7). console.log(b);
(8). console.log(c);
(9). }
(10). console.log(a)
(11). console.log(b);
(12). console.log(end);







ye para Local hi Let time script scope nahi hoga. global me script scope hota hai.

★ Predefined functions :-

math :- (object)

(1). math.floor() →
ye decimal ko hata deta hai number se

eg:- let a = 11

math.floor(a/2); → 5

workings:- 2)11(5.5 ⇒ 5.5 → .point wale ko hata dega.
10
10
10
0

5.6=6 dis nahi karega roundoff nahi karega sidha hata dega point wale value ko.

(2). math.ceil() →
math.floor ka opposite hai ye. ye v roundoff nahi karta hai.

eg:- math.ceil(0.60) → 1
" " (5.5) → 6
" " (3.4) → 4
" " (5.2 → 6
" " (-5.1) → -5

Floor = 5
5.1
ceil = 6

DATE

(3). Math.round() →
ye roundoff karata hai decimal number ko.

eg:- Math.round(5.5) → 6

Math.round(5.3) → 5

(4). Math.random() →
ye 0 se 1 ke bich random number generate karta hai.

eg:- c.log(Math.random()) → 0.2371518...

(5). Math.max() →
eg:-
c.log(Math.max(5,4,3,9,8)) → 9

(6). Math.min() →
eg:- c.log(Math.min(1,5,8,9)) → 1

(7). Math.pow() →

eg:- c.log(Math.pow(10,2)) → 100
↳ 10^2

(8). Math.sqrt() → Squareroot
eg:- let a = 16;
let output = Math.sqrt(a);
$\sqrt{16} = 4$

DATE

c.log(output) → 4

(9). Math.cbrt() → (cuberoot)
eg:- let a=125;
c.log(Math.cbrt(a)) → 5

(2). Date():-
console.log(Date()); → Apr 11 24 03:48:52 Today date and time

Date.now() → 1712037566506 (1st January 1970 se aaj tak ka time in milliseconds)

↳ function/object

#. Its subfunctions :

```
const date = new Date();
  ram
  Shyam
  (Any name)
```

const → keyword
date → variable
new → keyword
Date() → Constructor function

↳ Is se Date() function aaj ka date, day name, year ye sab return karega and us date me store hoga and fir usspar ham alag alag subfunction use karenge specific cheezo ko get karne ke liye.
eg:- day ka day (Sun, Mon, Tue etc) chahiye to "get day" use karke get kar lenge. aise hi or r subfunctions use karenge isspar.

console.log(ram); → Wed Apr 03 2024 10:20:18 GMT+0530 (IST)
or
2024-04-11T10:12:00.000Z → in Mone

(1). getDate() →
. Is se aaj ka date milega.
const date = new Date();
eg: c.log(date.getDate()) → 11

(2). getDay() →
. ye aaj ka din return karta hai. 0 to 6 me o/p deta hai.

0	1	2	3	4	5	6
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday

. To output days ke naam me lene ke liye ek array bana denge, days ka array.

eg: const date = new Date();

let days = ['Sun', 'mon', 'tue', 'wed', 'thur', 'Fri', 'sat'];
0 1 2 3 4 5 6

c.log(days(date.getDay())); → c.log(days(4)); → thur

(3). getTime() →
. Is se 1st January 1970 se aaj tak ka time milliseconds me.
const date = new Date();
eg: c.log(date.getTime()); → 17120....

(4) getHours() →
eg: const date1 = new Date();
c.log(date1.getHours()) → 10

(5) getMinutes() →
eg: c.log(date1.getMinutes()) → 39

(6) getSeconds() →
eg: c.log(date1.getSeconds()) → 0

(7) getMilliseconds() →
eg: c.log(date1.getMilliseconds()) → 457

(8) getMonth() →
. ye v get day ki tarah number me o/p deta hai.
. ye 0 - 11 tak o/p deta hai jisme 0 → January and 11 → December.
. number ko array ki help se month name me badlenge using array like get day().

eg: const mydate = new Date();
const months = ['Jan', 'Feb', 'mar', 'apr', 'may', 'June', 'July', 'Aug', 'September', 'oct', 'nov', 'dec'];
c.log(months(mydate.getMonths())); → getMonth() return 3
c.log(months(3)); → apr

(9) getFullYear() →
eg: const date = new Date();
c.log(date.getFullYear()); → 2024

DATE

↳ make own date and apply subfunctions of date on that date :-

① const covidHoliday = new Date('03-25-2020');
↳ ab aaj ka date nahi, ye wala date jaayega covidHoliday variable me.

console.log(covidHoliday);
↓
2020-03-25T00:00:00.000Z

② const covidHoliday = new Date('03-25-2020 06:00:00');

↳ console.log(covidHoliday); → Wed Mar 25 2020 06:00:00 GMT(IST)

(date) → const date = covidHoliday.getDate();

const monthArray = ['Jan', 'Feb', ---- , 'Dec'];

(month) ⇒ const month = monthArray[covidHoliday.getMonth()];

(year) → const years = covidHoliday.getFullYear();

(seconds) → const sec = covidHoliday.getSeconds() > 9 ? : '0';

(hours) → const hour = covidHoliday.getHours();

(min) ⇒ const min = covidHoliday.getMinutes() > 9 ? : '0';

↳ console.log(date + "/" + month + "/" + year); 25/Mar/2020

↳ " " (date + " " + month + " " + year); 25 Mar 2020

↳ console.log('Holiday starts from' + hour + ':' + min + ':' + sec);

↳ Holiday starts from 6:00:00

(10). Date.now() →

1st January 1970] milliseconds

3rd April 2024

PAGE

DATE

(10). Functions :- (Object)

↳ Functions in javascript →

(i) Function is an object.

(ii) Function is a block of instructions. which is used to perform some specific task.

(iii) A function get executed only when it is called.

(iv) main advantage of function is we can achieve code re-usability.

(v) To call a function we need function reference/ function name and '()' parenthesis.

(vi) variable written/declared inside function have local scope.

(vii). Functions can be hoisted.

(viii). Function does not belong to tdz.

↳ parameters →

(i). The variables declared in the function definition is known as parameters.

eg:- { function sum(a,b){}

↳ Here variable a, and b are parameters.

(ii). parameters are used to hold the values passed by calling statement

(iii). The parameters have local scope. It can be used only inside a function.

PAGE

DATE

↳ Arguments →

(i). value passed in the function call statement is known as arguments.

(ii). An argument can be literal {10, 20, etc}, variable {a, b, sum, etc} or expression/function which gives a result.

↳ Return keyword →

(i). It is the keyword which is used to transfer the control to the calling statement.

(ii). Return keyword will stop the execution of a function.

Types of functions in javascript :-

(1). Function declaration/Function statement.
(2). Function as expression/Function expression.
(3). Iife (Immediate invoke function expression) / anonymous function)
(4). Arrow function.
(5). HOF (Higher order function).
(6). ~~cbf~~ (call back function).

(1). Function Declaration Statement :-

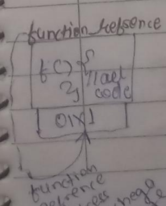
Syntax:- → [keyword]

function function reference (para-1, para-2, para-3=0)
or function name ↓ [default value]

{

// Block of code or instructions

}



[function_reference (Argument-1, Argument-2);]

↳ For call.

PAGE

DATE

eg-① function SayHbd (fname, lname, age=1) {

console.log('Happy birthday' + fname + " " + lname + " " + age);

return 'abra ka dabra...'

}

const fname = prompt ('Enter your first name');

" lname = prompt ('Enter your last name');

" age = + prompt ('enter your age');

let response = SayHbd (fname, lname, age);

C.log (response); → pehle fname, last name, age print hoga fir return wala abra ka dabra...

② function puneetMemes (name) {

return ('bekar hai ${name} bhai me to tut gaya', 'ab tu dekh ${name} ab tu dekh beta')

}

C.log (puneetMemes ('puneet'));

↳ ab tu dekh puneet ab tu dekh beta.

Sirf pehla doosra statement return hua return se.

↳ dono statement ko return karane ke liye, function puneetMemes {

const meme1 = 'bekar hai ${name} bhai me to tut gaya';

function Salmon() → Salmon ek variable hai jo function keyword se declare hai

const meme2 = 'ab tu dekh ${name} ab tu dekh beta ab tu dekh ${name}';

return (meme1 + meme2);

}

C.log (puneetMemes ('golu')); ↳ bekar h golu bhai me to tut gaya. ab tu dekh golu ab tu dekh beta ab tu dekh golu.

Iss baar done meme return hoga

Hoisting in first type of function (function declaration statement) :-

① C.log (SalmonLines());

② function SalmonLines () {

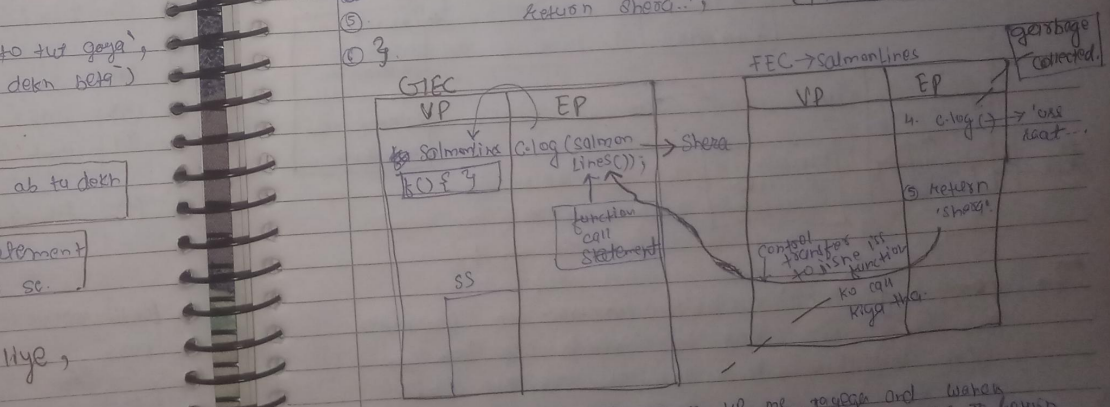
③ C. log ('uss raat gaadi driver

④ Chala raha tha');

⑤ return 'Shera..';

⑥ }

FCF/LEC → SalmonLines → Anonymous (LIFO)



• SalmonLines() console me se VP me jayega and waha se pura function leke aayega and wo execute hote hi return karega wo C.log me aayega and print ho jayega.

DATE

- Jo function clog up se layega uska ek alag local scope banega and waham execute hoga.

② ① C.log (vimalMeme ('kuldeep'));

③ function vimalMeme (name)

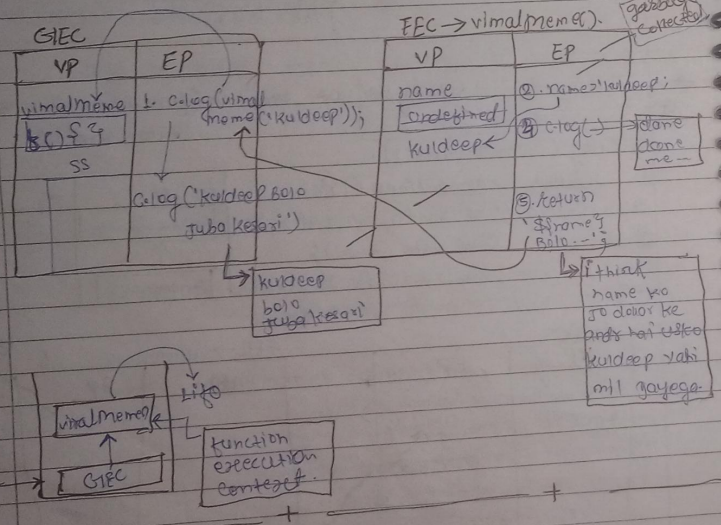
④ {

⑤ C.log(' name name me kesar ka swaad');

⑥ return ` ${name} Bolo juba kesari';

⑦ }

Function hai scope or Local scope.



Note → name → jo 'name' variable function ke parameter me hai' wo var ki' jaisa act karega.
- aise variable ko undefined variable phase me hi mil jaata hai'.

DATE

(2). Function as expression / function expression / first class function :-

(i). Function as expression ko hamesha function definition and declaration ke baad hi console karenge.

eg:-
```
let a = function() {};
C.log(a);
```

(ii). Function as expression can't be hoisted.

(iii). Function as expression does not belongs to temporal dead zone.

(function aaya hi' nahi' hai' variable phase me kahin kho gaya hai' value tab wo TDZ me nahi' jayega (pad agr value aajata variable ke andr and fr access nahi hota tab ham keh sakte hi' wo TDZ me hai').

(iv). let se function banayenge to wo baad me uske andr doosra value initialize ho sakte hai' isiliye const se karenge.

[redacted]

(v). dis function ke declaration / definition me pehle ki' variable aa gaye (let, const, var) tab wo function function as a expression kehlaata hai'.

Syntax:-

```
const rajpalMeme = function ()
keyword  variable  {
        c.log('Aey bhagwan! kya zulm hai');
    }
```

function call statement → rajpalMeme(); → parenthesis
(function key / function-name)

rajpalmeme: f() { }

c.log(rajpalmeme); → f(){}

eg ①.
```
const baburaoMeme = function () {
    console.log('Kya gunda banega re tu...');
}
baburaoMeme();
```
→ Kya gunda banega re tu.

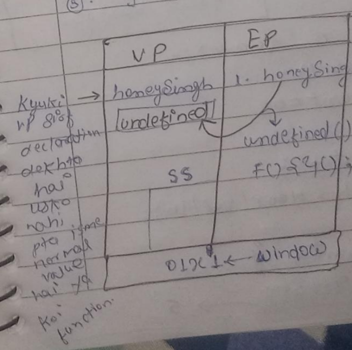
②.
```
const totlaSethMeme = function () {
    return `me bhi totla mera baap bhi totla mera pura khandan totla`;
}
const returnValue = totlaSethMeme();
c.log(returnValue);
```
→ me bhi totla mera baap bhi totla mera pura khandan totla hai

③.
```
const udayMeme = function (name) {
    return `Kya karu me ${name} isko maar v to nahi sakta pyar jo karta hai mujhse...`;
}
c.log(udayMeme('golu'));
```
→ Kya karu me...

④.
```
let myGF = function () {
    c.log('GF name is Chombi');
};
myGF = 30;
c.log(myGF);
```
→ 30

→ **Hoisting in function expression :-** ① var case →

①. honeySingh() {agr yahape c.log(honeySingh) = o/p = undefined}
②. var honeySingh = fuction ()
③. {
④. c.log('Alexa play Haye mera dil');
⑤. }



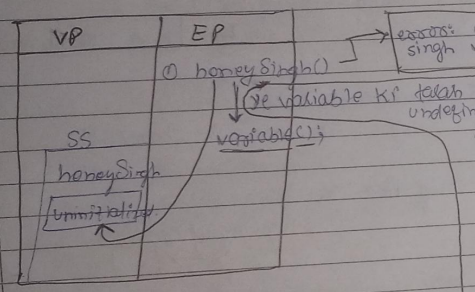
→ honeySingh is not a valid function.

DATE

(2) let case :-

agr same program me function ko let se declare hote,

eg:- let honeysing = function() {
console.log('Alexa play Kaye mera dil');
}



error: Can't access honey singh variable before initialization.

Ye variable ki value abhi undefined() in var case. undefined VP se leke aaya tha EP honeysingh ki jagah pr. and fir function ki value treat kiya tha undefined() ko. Isliye V pehle honeysingh ki jagah VP se value leke aayega fir unde aage () lgate hi a function treat karega.

8. # WAJSP to find count of a numbers:-

```
const findCount = function(val) {
    let count = 0;
    while(val !== 0) {
        count++;
        val = math.floor(val/10);
    }
    return count; }
console.log(findCount(12345)); → 5
```


DATE

(3). IIFE (Immediate invoke function expression) / anonymous function) :-

(i). When a function is called, as soon as its object is created is known as immediate invoke function.

(ii). The function is not visible outside the scope.

(iii). Immediate invoke function executes only once.

(iv). Local se global me value bhejne ke liye :-

eg: let userName;
(function() {
let name = prompt("enter your name");
username = name;
})()
console.log(username);

Syntax:

function reference

```
function() {
    let result = confirm("Do you want to share your location");
    console.log(result);
})().
```

DATE

eg:-
```
const finalOutput = (function(){
    let result = confirm('do you want to share your location');
    return result;
})()
console.log(finalOutput);
```
→ true

(H). Arrow function :-

(i). It is introduced in es6.

(ii) main advantage of arrow function is, it reduces the syntax.

(iii) If we have only single parameter then there is no neccessary to use paranthesis for single parameter.

(iv). If function have single statment then curly braces is not used.

(v). Arrow function does not have 'this' keyword.

↳ Types of return in arrow function :-

(1). Implicit return :-

. If there is only one statement we need not to use curly braces. Js engine will return that statment automatically.

DATE

(2). Explicit return :-

. If block is created (curly braces used) we need to use return keyword to return a value otherwise js engine will return undefined.

Syntax :-

(1). Explicit return → parameters

```
const sum = (a,b) => {
    return (a+b);
}
```

2. Implicit return →

(i). const sum = (a,b) => a+b;
↳ this will automatically return.

(ii) const user = name => console.log(`userName is: ${name}`);

eg:- (i). const product = (a,b) => {
    console.log(a*b);
}
console.log(product(2,5));
↳ kyuki hamne koi v return nahi kiya isliye automatically JS engine undefined return karega.

(ii). const product = (a,b) => {
    console.log(a*b);
    return `product of ${a} and ${b} is: a*b;`
}
console.log(product(2,5));
↳ product of 2 and 5 is :10

(iii). const user = name => name;

c.log (user('Amit'));

↳ Amit | return Implicitly

Arrow function syntax without parameters:-

let myFunction = () => {
    c.log ("Hello");
    c.log ('Developers');
}

Functional programming :-

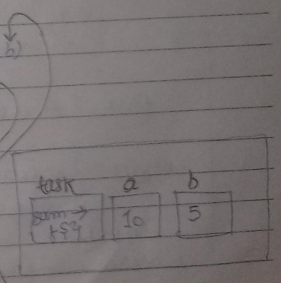
(i). It is a programming technique, where we pass a function along with a value to another function.

(ii). In this approach we generate generic function. Here function task is not predefined, it perform multiple task.

(5). HOF (Higher order function) →
The function which accepts another function as a parameter or return a function is known as higher order function.

(6). CBF (Call back function) →
The function which is passed to another function or the function which is returned by another function is known as call back function.

eg.
// HOF
function Calculator (task, a, b) {
    return task (a, b);
}
// CBF
function Sum (m, n) {
    return (m+n);
}
let result = Calculate (Sum, 10, 5);
c.log (result);



DATE

- calcuate function task(a,b); return karega jo ki sum(a,b) hoga.

ye result ke andr store ho jayega and fir jab colog(result) karenge to sum(a,b) result me hoga to ye bhi ho jayega yahi se and answers result leke aayega and result ki jagah replace karega and is print kar dega.

```
// HOF
function first (task){
    c.log ('first function called');
    task()
}
// CBF
function second (){
    c.log ('second function is called.')
}
first (second);
```

output:-
first function called
second function called.

```
//HOF
function company (candidate, name, age)
{
    c.log ('company name is TCS');
    c.log ('Hiring for SDE');
    candidate (name, age);
}
```


DATE

```
// CBF
function employee (name, age)
{
    c.log ('name is :' + name);
    c.log ('age is: ' + age);
}
company (employee, 'pritam', 23);
```

output:-
company name is TCS
Hiring for SDE
Name is: pritam
age is : 23

```
// HOF
function company (candidate, locality, name, age, state, city)
{
    c.log ('company name is TCS');
    c.log ('Hiring for SDE');
    candidate (locality, name, age, state, city)
}
// CBF
function employee (task, name, age, state, city)
{
    c.log ('name is :' + name);
    c.log ('age is: ' + age);
    task (state, city);
}
// CBF function locality (state, city) {
    c.log ('state name is ' + state);
    c.log ('city name is ' + city);
}
company (employee, locality, 'pritam', 23, 'UP', 'Noida');
```

Need to understand

DATE

⑤. functional programming using arrow function :-

```
//HOF
cont calculator = (task, a, b) => task(a, b);

c.log(calculator((a, b) => a+b, 5, 5));
                  └──CBF──┘
```

task = (a, b) => a+b

a = 5
b = 5

⑥.

```
const sum = (p, q) => p+q;
»     sub = (p, q) => p-q;
»     div = (p, q) => p/q;
»     prod = (p, q) => p*q;
»     sqrs = (p, q) => p**q;

const generic = (task, a, b) => task(a, b);

c.log(generic(sqrs, 5, 2));
```

⑦.

```
const sum = (p, q) => c.log(`sum is ${p+q}`);
»     sub = (p, q) => c.log(p-q);
»     div = (p, q) => c.log(p/q);
»     prod = (p, q) => c.log(p**q);

//HOF
const generic = (task, a, b) => {
    switch(task)
        case 'sum':
            return sum(a, b);
        case 'sub':
            return sub(a, b);
```


DATE

```
        case 'div':
            return div(a, b);

        case 'prod':
            return prod(a, b);

        case 'pow':
            return pow(a, b);

        default:
            return 'you are entering wrong task
            pls enter task: sum, sub, div, prod,
            sqrs.
    }
}

//CBf
const result = prompt('enter your task: sum, sub, div,
                       prod, pow');

const a = +prompt('Enter first Number');
const b = +prompt('Enter second value');
c.log(generic(result, a, b));
```

⑧.

```
//HOF
function generic(task, userName){
    task(userName);
}

//CBf generic(function(username){
    c.log('user Name is: ' + userName);
}, 'mahendra singh dhoni');

//CBf generic((email) => c.log('user email is: ' + email), 'xxx@gmail.com');
```

DATE

Nested function :-

(i). Function inside function is known as Nested function.

(ii). The outer function is known as parent function.
The inner function is known as child function.

(iii). Inner function is local to the outer function, it means we can't access inner function from outside.

(iv). To access inner function, outside outer function, the outer function should return the reference of inner function.

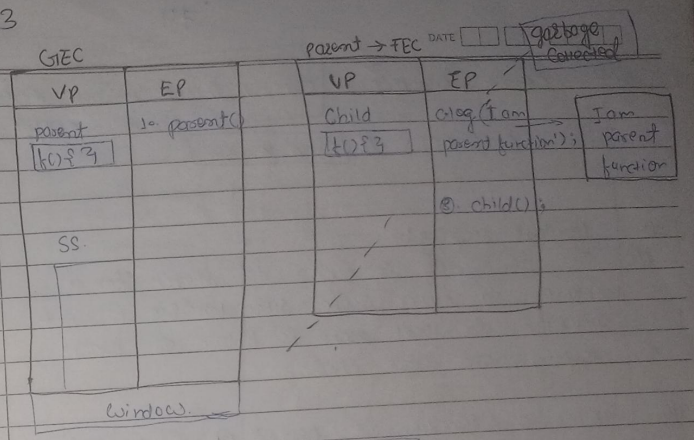
eg:-

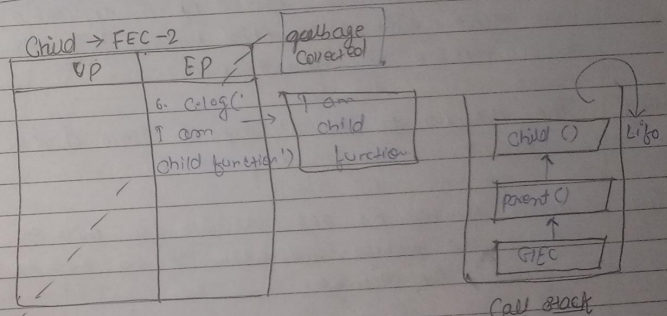
```
function parent ( )
{
    c.log (' I am parent function...');

    function child()
    {
        c.log (' I am child function...');
    }
    child() → Calling child function
}
parent(). → Calling parent function.
```

output:-

```
I am parent function
I am child function...
```


DATE






```
const message = 'Lexical scoping';

function outer() {
    c.log ('outside parent function...');
    function inner () {
        c.log ('Inner child function');
        c.log (message);
    }
    return inner;
}
```

(inner())

```
const output = outer(); // inner function stored inside output
output(); // Child function.
```

lexical scoping:-

(i). The ability of js engine to search for a variable in outer scope when variable is not available in local scope is known as lexical scoping or scope chaining.

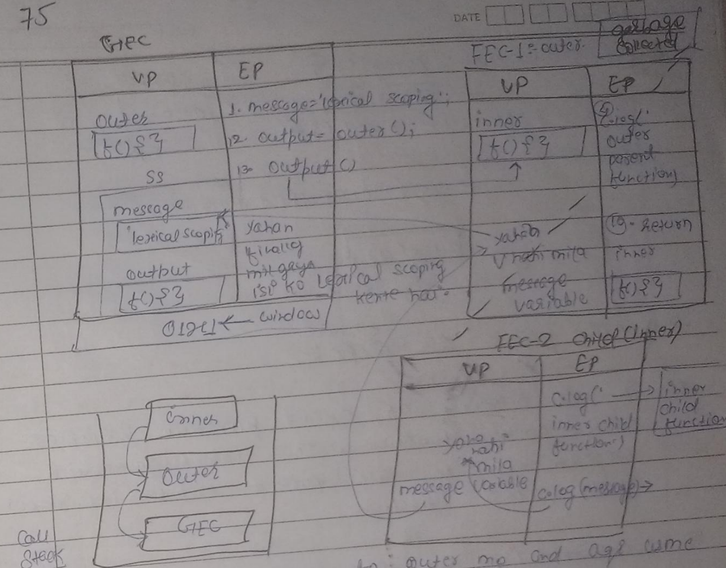
(ii). When a function is nested it is applicable at that time when child function tries to access variable in parent function.

(iii). The direction of lexical scope is from child to parent.

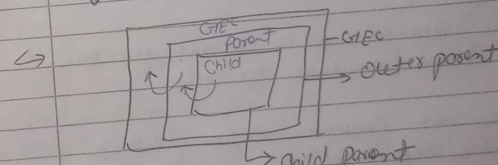
(iv). Inner function me agg variable na mile to outer me dekhegg and wahan v na mile to global me dekhegg.

eg.

```
① const message = 'lexical scoping';
② function outer ()
③ {
④   c.log ('outer parent function...');
⑤   function inner ()
⑥   {
⑦     c.log ('Inner child function');
⑧     c.log (message);
⑨   }
⑩   return inner;
⑪ }
⑫ const output = outer();
⑬ output();
```

• inner me na mile to outer me and agar wahan v na ho tab GEC me dekhenge.



closure: Jab parent function ke andr koi variable ho and wo child function me use ho raha ho and parent function garbage collected ho jayega tab wo variable ko ek alag memory de dega taki parent function jab destroy ho jaaye tab bhi child function uss variable ko use kar sake. usi memory place ko closure kehte hai.

DATE

eg:- function parent() {
let a=10;
function child() {
c.log('inner child function');
}
c.log(a);
}

closure
a=10

More points on closure :-

1. It is binding of parent function variables with child function, this enables javascript progeammers to use parent members (variables) inside child function.
2. closure is an object. It have the state of parent function.
3. closure is created when there is one function inside another function.
4. closure helps to achieve scope chaining or lexical scoping.
5. Child function will have the reefference of closure.
6. closure reserves the state of parent function even after the execution of parent function (garbage collected of parent function) completed.

DATE

7. Every time a parent function is called a new closure is created.
8. Closure has access to variable in three scopes,
   (i) variable declared in that scope
   (ii) variable declared in parent function scope.
   (iii) variable declared in global name scope.

↳ Disadvantage of closure :-

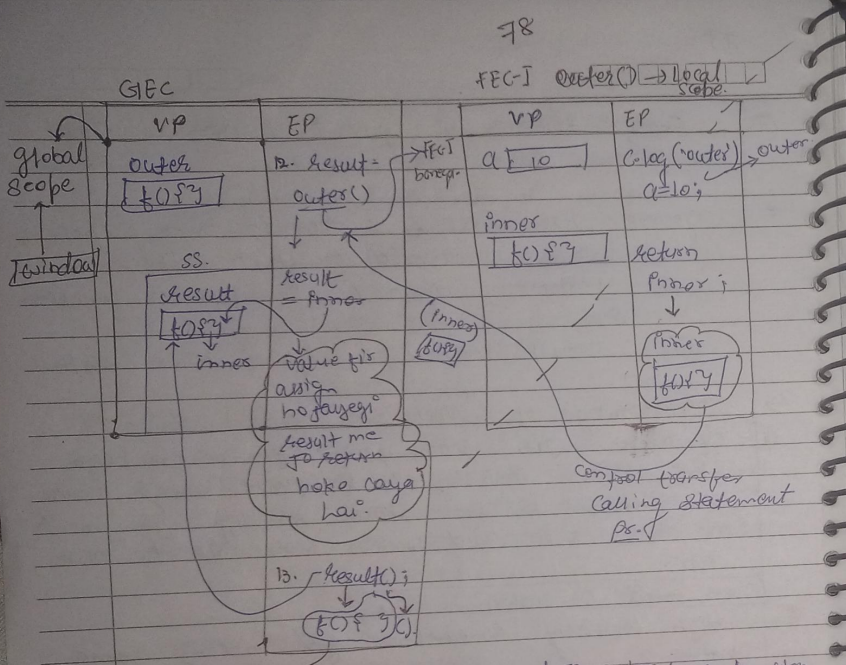
1. By me → Jitni baar parent function call hoga utni baar, memory block create karega closure un variables ko store karne ke liye jo parent me hai and child function me call ho rahe hai.

Some point in english by Sir

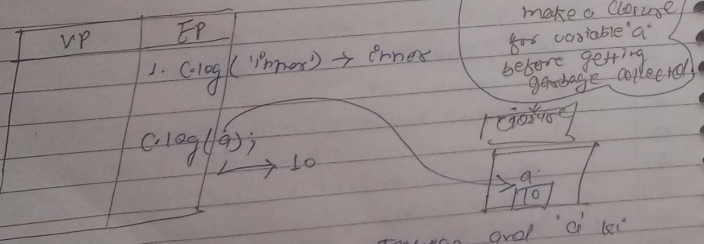
2. High memory consumption for holding same values.

↳ closure understanding using GIFC :-

```
1  function outer()
2  {
3      c.log('outer');
4      let a = 10;
5      function inner()
6      {
7          c.log('inner');
8          c.log(a);
9      }
10     return inner;
11 }
12 let result = outer(); (13) result();
```



Ab iske liye ek GEC banega called FEC-2 for inner function.
Yahan par inner function jo result me hai wo call hoga and ek local execution context banega iska.



inner function ka 'a' → fir closure me jayega and 'a' ki value leke aayega.

DATE

(11). Arrays in JS :-

① Array is an object
② It is non-primitive type of data.
③ It is block of memory which is used to store multiple type of value (any type of value) in some memory block.

④. Array size is dynamic (size is not fixed like java), it means we can store 'n' no. of elements and JS engine will handle memory usage automatically.

⑤ Values stored inside array are reffered as array elements.

⑥. Array elements are arranged in a sequence that is represented by integer number called as index.
Array index starts from zero to array size - 1. (index = length - 1)

⑦. To access array element,
array object reference [index]
or
array name [index number].

⑧ we get undefined if ham array ke last index se jyada wale element ko access kiya to.
eg:- let a = [1,2]; C.log(a[2]); → undefined
① ②

⑨ array elements are separated by comma.

index = length-1

length = 4

Syntax :-

array element

```
const movies=['Golmaal', 'Dhamal', 'Phir hera pheri',
'PK'];
```

index of [0] — 'Golmaal'; array element [1] — 'Dhamal'; [2] — 'Phir hera pheri'; array element [3] — 'PK'

→ for print :-

Array reference [index of array element]
or
array name

```
c.log (movies[3]);  → PK
```

(10) Array is heterogenous (sare elements zaruri nahi ek hi type ke literals or ek hi type ke data hoge).

eg:-

```
let arr=['Chombu', 24, 'noida', false, null];
```

string, number, string, Boolean, object

(4) Fourth point eg:-
array size is dynamic,

```
let courses = ['web', 'Java', 'SQL'];
```

→ baad me isi array me ham add kar sakte hai.

```
courses = ['web', 'Java', 'data science', 'python'];
```


Ways to create an array:-

(1). Empty Array:-

```
const empty_array =[];

c.log (empty_array);  → Array[]
```

(2). Array with literals :-

length=3

```
const users =['Ram', 'Mohan', 'Manan'];

c.log (users);  → ▾ Array (3) ['Ram', 'mohan', 'manan']
                    prototype: Array[]
```

↳ ye array ko help karta hai.

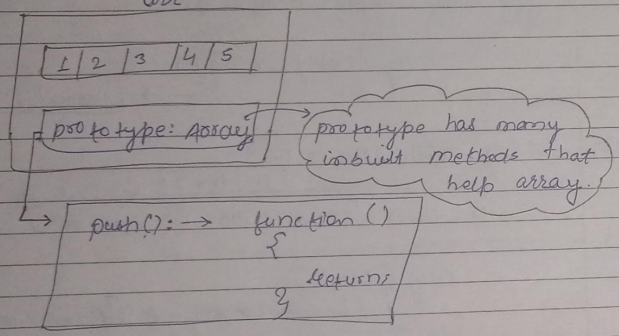
(3). By using Array Constructors :-

```
① const my_array = new Array (____);
                               values

c.log (my_array);  → Array []
                     Empty array

② const second_arr = new Array(1,2,3,4,5);

c.log (second_arr);  → dropdown list
                       ▸ Array (5) [1,2,3,4,5]
                       prototype: Array []
```

DATE

Array inbuilt methods :-

let arr = [1,2,3,4,5];



↳ These are the following type of inbuilt methods on array :-

(1). push() → add element in given array at last position.

eg: let arr = [10,20,30,40] ← original array
length = 4

const output = arr.push(50,60,70);

console.log(arr); → [10,20,30,40,50,60,70]
length = 7

DATE

. It return, length of array after pushing element or new array that considers new elements. (~~Arr~~ element push hone ke baad jo array milega usko length return karta hai).

console.log(output); → 7

2. pop() →

(i) It deletes element from last index of array

(ii) Return deleted element

eg: let vegItems = ['paneer chilly', 'Aloo paratha', 'Raita', 'Gulab jamun']; ← last index element

console.log(vegItems.pop()); → Gulab jamun (deleted element returned)

console.log(vegItems); → ['paneer chilly', 'Aloo paratha', 'Raita'] ← updated array

3. unshift(value) →

(i) Insert element at first index of array

(ii) returns array length

eg: let arr = [30,40,50]

DATE

```
c.log(arr.unshift(10,20));
                  └→ [5] → length of updated array
c.log(arr);
   └→ [10,20,30,40,50]
```

4. shift() →

(i) delete element from first index of array.
(ii) return deleted element.

eg: let arr1 = ['dA', 'OS', 'MERN', 'java'];

```
let notAVl = [];
for (let i = 0; i<2; i++){
    notAVl.push(arr1.shift());
}
c.log(notAVl);
   └→ ['dA', 'OS']
```

(5). Splice (a, b, c) →

(i) perform insertion, deletion and updation in original array.

(ii) It will modify the original array.

(iii) return array of deleted elements.

PAGE

DATE

a → starting index
b → no of elements to be deleted.
c → inserting value / update values.

eg:

① let arr = [10, 20, 50, 60, 70]
(insert?) ↓ ↓ ↓ ↓ ↓
0 1 2 3 4
(these value to be added)

```
const output = arr.splice(2, 0, 30, 40);
                          └→ 0 elements delete
   └→ start from second index
```

```
c.log(arr):
   └→ [10,20,30,40,50,60,70]
```

c.log(output):
→ Array[] (kyuki abhi kuj v delete nahi hua hai to return v nahi hoga kuj v.)

② let arr = [10,20,100,90,50,60,70]
(delete)

```
const output = arr.splice(2,2,30,40);
```

```
c.log(arr):
   └→ [10,20,30,40,50,60,70]
```

```
c.log(output):
   └→ [100,90]
```

PAGE

DATE

deletion :-

let movies = ['All the best', ~~'Avengers'~~, 'Hera pheri', 'welcome', 'No problem', 'Ready']
(0) (1) (2) (3) (4) (5)

const result = movies.splice(1,1);

c.log(movies);
→ ['All the best', 'Hera pheri', 'welcome', 'No problem', 'Ready']

c.log(result);
→ ['Avengers']

updation →

let movies = ['All the best', ~~'Avengers'~~, 'Hera pheri', 'welcome', 'No problem', ~~'Ready'~~]
(0) (1) (2) (3) (4) (5)

const rest = movies.splice(1, 5, 'Hera pheri', 'welcome', 'No problem']

c.log(movies)
→ ['All the best', 'Hera pheri', 'welcome', 'No problem']

c.log(res);
→ ['Avengers', 'Ready']

DATE

(6). Slice() → Slice(a, b)

(1). used to copy array elements.

(2). It will not modify the original array.

(3). returns array, of copied elements.

a → starting index
b → last index → last index excluded (b-1 index-1)

eg: (1) let arr = [1, 2, 3, 4, 5]
(0) (1) (2) (3) (4)

let arr2 = arr.slice(1, 4)
→ last index excluded

c.log(arr2);
→ [2, 3, 4]

eg: (2). let movies = ['RRR', 'Avengers', 'kgf', 'Jaat-x', 'Hanuman']

let myFav = movies.slice(0)
→ start starting index
→ last index not give so all element from starting element copied

c.log(myFav);
→ ['RRR', 'Avengers', 'kgf', 'Jaat-x', 'Hanuman']

myFav.splice(1, 3, 'kgf');

c.log(myFav);
→ ['RRR', 'kgf', 'Hanuman']

88

(7). indexOf() → [indexOf(a,b)]

1. use to get index of array element.

2. (i) If element's is available → it returns, element index number.

(ii). if not available → returns, [-1]

↳ [indexOf(a,b)]

a → value to be searched. (Jis element ke index ko search karna hai).
b → search starting index.
(Jahan se search start karni hai).

eg. ①

```
           (1)  (2)  (3)  (4)  (5)
let arr = [ 10, 20, 30, 40, 50 ]

const res = arr. indexOf (30);

c.log (res) ;   → [2]

c.log( arr. indexOf (50))   → [4]

c.log( arr. indexOf (100))   → [-1]
```

89

②

```
              (0)      (1)    (2)     (3)       (4)       (5)
let users = [ 'Raman', 'sid', 'joe', 'Adam', 'Kuldeep', 'Amit',
              'sid' ]
              ↳ (6)

c.log (users. indexOf ('sid', 2));
```

sid ko find karo isk index number

↳ searching sid ke index number ki index 2 se start karo.

→ [6]

```
c.log (users. indexOf (" "));   → [-1]
```

(8). includes () → [includes (a,b)]

a → search value
b → starting index (default = 0)

1. Check element is available or not.

2. if available then return → [true]
else return → [false]

eg.

```
let arr = [ 10, 20, 30, 40, 50 ]

c.log ( arr. include (20)) ;   → [true], [means 20 hai iss array me]

c.log ( arr. include( 20,2))   → [false] [index 2 ke baad 20 nahi mila]
```

DATE

(9). reverse() →

① use to reverse the array.

② It will modify the original array.

eg: let arr=[10,20,30,40,50];

c.log(arr.reverse()); → [50,40,30,20,10]

(10). sort(callback) →

① It will modify original array.

② Syntax:

sort(function(a,b){return (a-b)})

→ descending (b-a) ho jayega.

① 30-20 +ve = shuffle
② 2=2 0 = No shuffle
③ 20-30 -ve = no shuffle

③ Syntax:

sort((a,b) ⇒ a-b)

eg: ① let arr = [1,3,2,5,4]

arr.sort((a-b) ⇒ a-b)

1-3 = -2 (-ve) means NO shuffle

DATE

c.log(arr): → [1,2,3,4,5]

↳ to make descending order make (b-a).

eg: ②

let arr = [1,5,2,4,3];

arr.sort((a,b) ⇒ b-a)

c.log(arr): → [5,4,3,2,1]

(11). forEach (callback) →

① It is a higher order function.

② use to iterate over array elements and index.

③ doesn't return anything, so js engine implicitly return undefined.

syntax:

const res = arrayReference.forEach((currentVal, indexOfCurrentVal, Array) ⇒ {})

c.log(res): → undefined

DATE

currentVal = array ke element ko get karega ek ek karke.

index = jo element currentval me aayega uska index number.

eg:- (1) let numbers = [1,2,3,4];

numbers.forEach(val ⇒ val*10); → because we pass function as a parameter

↳ call back function.

C.log(numbers); → [1,2,3,4,5] kyuki forEach return nahi karta hai kuch bhi isliye purana array wise hi aayega.

↳ Kuchh return karne ke liye,

let numbers = [1,2,3,4,5];

let convertedNumber = [];

numbers.forEach(val ⇒ {
convertedNumbers.push(val*10);
});

C.log(convertedNumbers); → [10,20,30,40,50];

Note:- Diff b/w foreach and map →

for-each ↳ ye kuchh v return nahi karta hai (chahe return keyword likhde tab v)

map ↳ ye return karta hai.

DATE

(12) Map:- (map(callback)).

(1) It is HOF
(2) use to iterate over array
(3) It will not modify original array
(4) Return new array.
(5) value return by callback function will be inserted in new array, if it doesn't return anything undefined will be stored.

eg:- (i) // output :- [10,20,30,40,50]

let arr = [1,2,3,4,5];

let result = arr.map((val, indexe, array) ⇒ {
return val*10;
});
1*10
2*10

C.log(result); → [10,20,30,40,50]

(ii) let numbers = [10,20,30,40,50]

let result = numbers.map((val, indexe, array) ⇒ {
return val/10;
});

C.log(result); → [1,2,3,4,5]

(iii) let arr = [10,20,30,40,50];
let sum = 0;

```
arr.map((val, index, array) => {
        sum = sum + val;
});
   c.log (sum);  →  150
```

(14). let arr = [100, 200, 500, 300, 400]

```
let biggest = 0;

arr.map ((val, index, array) => {
     biggest = val > biggest ? val : biggest;
});

c.log (biggest);  →  500
```

// also inbuilt array methods.

(13). flat() →

(a) flat method creates a new array by flattening a nested array up to specified depth.

means, jitna depth ham bataoge utne nested array ko normal array banayega means unke [ ] brackets ko hatake unke original array ko part bana dega. yes unke nested array ke elements ko original array ke element jaise kar dega.

eg. let numbers = [1, 2, [3, 4, [5, 6, [7, 8]]]]


```
let newArray = numbers.flat(2);
```

↳ 2 depth → means 0 index se shuru karke jo pehle do nested array honge unko original array me combine kar dega unke [ ] bracket hatake

```
c.log ( newArray);
```

↓ output

[1, 2, 3, 4, 5, 6, [7, 8]]

(I) flat() parameters →

(a) flat method takes a single parameter,

depth → . default value 1.
. ye batata hai ki kitne deep ok kitne nested array ko original me combine kar dega.

(II) flat() return value →

(a) Returns new array (flatted array) with sub-array elements concatenated into it.

simple se ek naya array return karta hai nesting ko specified depth tak khatam karne ke baad.

Note (I) does not change original array
(II) remove empty slots in array.

DATE

eg:- ① let arr = [10,20,30,[40,50,[60,70],[80,90],100]];

let output = arr.flat(2);

c.log(output)
↳ [10,20,30,40,50,60,70,(2)[...],100]
depth=2

or

let output2 = arr.flat(3);

c.log(output2);
↳ [10,20,30,40,50,60,70,80,90,100]
depth=3.

⑥ // flat and map use at same time.

let arr = [10,20,30,[40,50,[60,70],[80,90],100]];

let output = arr.flat(3).map(cval => math.floor(val/10));

c.log(output);
↳ [1,2,3,4,5,6,7,8,9,10]

⑭ flatmap() → pehle flat and map alag alag use karte the ab method se done kaam hoga.

① It first maps each element of an array using a mapping function, then flattens it into a new array.
↳ to flat (सामान्य array में convert करना).

DATE

not modify original array

till now = 1 slice (1-9) for each map flat flatmap.

eg:- let numbers = [1,2,3,4,5]

// sabhi elements ko pehle square karo fir flatten karo.
const resultingArray = numbers.flatMap((val) => val**2);

c.log(resultingArray); ↳ [1,4,9,16,25]

↳ Syntax:
arr.flatmap(callback(currentValue), thisArg)

⑫ flatmap() parameters →
↳ takes two parameter.

① callback → callback function to initially execute on each array element. It takes in,
(i) current value.

② thisArg → (optional) - value to use as this when executing callback.

③ flatmap() return value →

④ returns new array

Note: ① does not modify original array
② flatmap() = array.map().flat(); (both are same).

eg:- ② let arr = [1,2,[3,4],5]

const output = arr.flatMap((val) => val**2);

c.log(output);
↳ [1,4,NaN,25]

PAGE

Return baar chalta hai and har baar value return karta hai acc chatgpt

DATE

(15) reduce (callback) →

① HOF

② use to iterate and conclude result to a single value.

③ not modify original array.

④ returns single value.

⑤ if we does not pass initial value of accumulator first element of array will be stored automatically.

eg:- ① let arr = [1,2,3,4,5]

let sum = arr.reduce ((accumator, val, index, array) => {

accumulator = accumulator + val;

return accumulator;

}, 0);

Initial value of accumulator.

c.log (sum);

→ 15

② let arr = [360, 180, 500, 190, 480]

let biggest = arr.reduce ((acc, val) => {

if (val > acc) {

acc = val;

}

return acc;

}, 0);

c.log (biggest);

→ 500

(2-9) (slice) → not modify

(forEach, map, find, findIndex, reduce, filter) → not modify original

DATE

(16) filter() → filter (callback) (same syntax [forEach, map, filter, reduce + accumulator])

① HOF

② Iterate over array

③ not modify original array

④ returns new array.

⑤ Here, element will be inserted in new array only when callback function returns true.

eg:- ① // Question →

const numbers = [100, 200, 500, 900, 600, 300, 400, 700, 800, 1000]

// output = [600, 700, 800, 900, 100];

const output = numbers.filter (val => {

if (val > 500) {

return val;

}).sort ((a-b) => a-b);

c.log (output);

→ [600, 700, 800, 900, 1000]

or

// sorting baad me karo.

output.sort ((a-b) => a-b);

c.log (output);

→ [600, 700, 800, 900, 1000]

100 let Array=[1,2,3,4,5] Array.isArray (literal);

ye array reference or array name nahi hai. ye isArray ke saath use karte hi hai

ye array koi array ka naam nahi hai.

(17). isArray (literal) → Array.isArray (literal) →

1. use to check given literal is array or not.

2. if it is array → It will return true. otherwise, return false.

eg. ①

```
C.log (Array.isArray ({}));
        ⟶ false

C.log (Array.isArray (10));
        ⟶ false

c.log (Array.isArray ([10,20,30]));
        ⟶ true
```

imp ② // nesting removal without flat() method.

```
let arr = [1,2,3,4,5,[6,7],8,[9,10]];
   or   let arr = [1,2,3,4,5,[6,7,8,[8,[9,10]]]]

let output=[];

function flatArray (arr) {
    arr.map (val => {
        if (!Array.isArray (val)) {
            output.push (val);
        } else {
            flatArray (val)
        }
    });
}

flatArray (arr);
C.log ( output);
   ⟶ [1,2,3,4,5,6,7,8,9,10]
```

101

Array.from (literal)

yaha v array, array ko reference ya array ka naam nahi hai jisko use kar rahe hai

(18) from (literal) → Array.from (literal).

1. use to convert literable literals (like object or string) to array

2. if literal is literable → it returns new array of elements

3. if literal is not literable → It returns empty array

eg. ① // convert string to array

```
const str = "hello";

const arr = Array.from (str);

C.log (arr);
   ⟶ ["h", "e", "l", "l", "o"]
```

②

```
let arr = 10;
let output = Array.from (arr);
C.log (output);
   ⟶ Array(); 
```

literal is not literable so it return empty array

③ // convert object to array

```
Const user = {
    username: 'chulbul pandey',
    dialog: 'abc',
    actor: 'salman bhai'
}
```

comma

102

Case 1: Const result = Array.from(user);

↳ c.log(result);
⟶ Array() → empty array, kyuki object par from lagane ke time Object ke kuchh predefined cheeze v use karni padti hai.

Case 2: Const result = Array.from(object.entries(user));

↳ c.log(result);
↳ [['username', 'chulbul pandey'],
['dialog', 'abc'],
['actor', 'salman bhai']
]

Or,

Array(3) [(2) {...}, (2) {...}, (2) {...}]

- 0: Array ["username", "chulbul pandey"]
- 1: Array ["dialog", "abc"]
- 2: Array ["actor", "salman bhai"]
- <prototype>: Array[].

(4). // Array.from(ref, mapfunction) — literal or object

```
let arr = [1,2,3,4,5]
let output = Array.from(arr, val => val*10)
c.log(output);
   ⟶ [10,20,30,40,50]
```

103

```
let a = [1,2,3]
let b = [4,5]
output = a.concat(b);
```

** concat se pehle koi array ka reference dena must hai for dot lagake concat use karne ke liye.

(19). **concat() :**

[array_reference.concat(any value or array any (kitna v) no. of time)]

(1) concat() method returns a new array by merging two or more values / arrays.

eg. (1)

```
let arr1 = [1,2,3]
let arr2 = [4,5]

let output = []
output = output.concat(arr1, arr2);

c.log(output);
   ⟶ [1,2,3,4,5]
```

And

```
output = output.concat(arr2, arr1);
c.log(output);
   ⟶ [4,5,1,2,3]
```

(2) // syntax:

arr.concat(value1, value2, ... valueN)

arr = array_reference or array name

(3) concat() parameters → concat() takes in an arbitrary (मनमाना) no. of arrays and/or values as arguments.

(4). concat() return → returns newly created array after merging all arrays/values passed in the argument.

DATE

(2). let courses = ['web', 'java', 'SQL']

let new1 = ['Adv java', 'DSA']
let new2 = ['python', 'Django']

courses = courses.concat(new1, "Communication", new2);

console.log(courses);

['web', 'java', 'SQL', 'Adv java', 'DSA', 'Communication', 'python', 'Django']

(20). join() →

let meme = ['Kanishq', 'Bhai', 'vo', 'tujhe', 'hi', 'dekh', 'rahi', 'hai'];

let output = meme.join(" ");

ek space aayega har word ke bad.

console.log(output);

Kanishq bhai vo tujhe hi dekh rahi hai

(1) join() method returns a new string by concatenating all of the elements in an array, separated by a specified separator.

(2) // Syntax :

DATE

arr.join(separator)

arr = array reference or array-name

(3) join() parameters :-

join method takes in,

↳ separator (optional) - A string to separate each pair of adjacent elements of the array.

By default ',' if it's comma

(4) join() Return value :-

Returns a string with all the array elements joined by separator.

Note (i) join() not modify original array.
(ii) Elements like undefined, null or empty array have an empty string.

PAGE

DATE

(12) Objects in Javascript :-

An object is a block of memory which has state (variable), behaviour (method's) and where we can store heterogeneous data.

An object is a collection of key-value paires that can contain various data types such as numbers, strings, arrays, function and other objects.

ek object me bahut saare key : value pair ho sakte hai. and wo separated hote hai (,) comma se.

Hum object ke value ko access kar sakte hai,
① (.) dot operator and/or ② [] square bracket se.

```
const obj = {
    name : "mahtab",
    Age : 20
};
```

```
c.log (obj.name);
```
→ mahtab

```
c.log (Obj["name"]);
```
→ error

↳ sqr bracket me name key ko double quotes me dena hota hai agr key ke value me string value ho.

So, `c.log(obj["name"]);` → mahtab

PAGE

107

let obj={} → do not make Singleton
let obj = new Object(); → make singleton.
Otherwise both are same.

DATE

Object Key (property):

① Object ke jo key hote hai wo js engine unko automatically string me convert kar dete hai.

eg:
let obj = {
"name" ← actually name : "mahtab",
"Age" ← Age : 20
}

② Agr key ka name numbers me hai, js engine unko string me convert karke ascending order me arrange kar dega.

eg:
let Object = {
"0" ← 0 : "mahtab",
"1" ← 1 : 20,
"3" ← 3 : "bihar",
"2" ← 2 : golu@gmail.com,
}

arrange in ascending order

String me conversion.

③ Space wale key name ko double quotes me likhna hoga.

eg:
User name : golu ✗
"User name" : golu ✓

④ Agr hum computed ya user defined property bana chahte hai as a key tab hame square brackets and variable name use karna padega.

PAGE

ye eg next page pr thoda naya with more details

Predefined property

DATE 108

let mysymbol = Symbol("key");

let obj = {
name: "golu",
[mysymbol] ← mysymbol : "India"
}

c.log(obj.mysymbol) → India (string type)
c.log(obj[mysymbol]); → India (Symbol)

⑤ Agr key ka naam, kisi variable name se mila hai to jo key value hold karte ho. uss case me variable name ko do baar likhne se achha hai ek baar hi likhe.

eg:
let phone = 12345;

let obj = {
phone, // instead of phone: phone
name: "golu"
}

detailed Syntax:

let user = {
user1D : 11223 → comma (separator)
Key value
States (variables)
userName: "Ram",
mob : 9050 40153

behaviour (methods)
getDetail : function(){
Key
c.log("key" + this.username);
value
}

PAGE

DATE

③ let obj = {
0: "Ram",
1: "Shyam",
3: "golu",
2: "Chambu"
}

c.log(obj);

↳
"0"	: Ram
"1"	: Shyam
"2"	: Chambu
"3"	: golu

② // Key and value take from user.

let key = prompt("Enter key name");
let value = prompt("Enter value");

let object = {
user defined Key → [key]: value
}

c.log(object)

↳ object{
name: "golu"
}

③ // no. of key and value take from user.

let num = +prompt("enter how many pair you want");

DATE

let object = {}

for(let i=0; i<numbers; i++){
let key = prompt("enter [i+1] key");
let value = prompt(" ");
obj[key] = value;
}

c.log(obj);

↳ Additional concepts
use cases of (.) dot operator and [] Jaa bracket
// Jyada dot hi use karte hai pr kuchh jagah sirf [] bi
eg: // easy to use (.) more instead of [].

① both can apply
const obj = {
name: "Hitesh",
age: 18,
location: "Jaipur",
email: "hitesh@google.com",
isLoggedIn: false,
lastLoginDays: ["monday", "Saturday"]
}

① c.log(obj.email); ✓
② c.log(obj["email"]); ✗
" (obj["email"]); ✓
↳ output

② only [] can apply

let obj = {
"full name": "H Choudhary"
}
c.log(obj."full name") → ✗
c.log(obj["full name"]) → ✓

Interview question only [] applicable

③ // Symbol as a key.

```
const mySym = Symbol("key1")

const obj = {
    mySym : "mykey1"
}

console.log(obj.mySym);
console.log(typeof obj.mySym);
```

→ mykey1 (String type)

OR

```
const obj = {
    [mySym] : "mykey1"
}

console.log(obj) → [Symbol(key1)] : 'mykey1'
console.log(obj[mySym])
```

→ mykey1 (Symbol)

↳ more general points →

④ // to change value of any key.

```
const obj = {
    email : abc@gpt.com
}

obj.email = abc@microsoft.com
console.log(obj.email);
```

→ abc@microsoft.com

DATE

② // Object freeze, isko karne ke baad koi v changes nahi ho payege objects me

```
const obj = {
    email : abc@tcs.com
}

obj.email = abc@wipro.com;

Object.freeze(obj);

obj.email = abc@accenture.com;

console.log(obj.email);
```

→ abc@wipro.com

Last change before Object freeze applied only.

⑤ // adding function as a key and calling it.

```
const obj = { };

obj.greeting = function () {
    console.log("Hello jis user");
}

console.log(obj.greeting);
```

→ [undefined] greeting ek function hai (abhi baaki object scope se pta chalega).

```
console.log(obj.greeting());
```

→ Hello jis user

DATE

Ways to create an objects :-

(1). By using {} curly braces and Literals.

(i) Empty Object →

```
const object = {}
```

(ii) Object with literals →

```
const obj = {
    userName: 'Chomba',
    city : noida
}
```

(2). By using object constructor and new keyword :-

(i) Empty object →

```
const obj = new Object({})
```

(const = keyword, obj = ref)

```
o.log( obj)
```
→ [{}]

(ii) Object with literals →

```
const obj= new Object({
    name: "golu",
    age : 20
})
```

```
o.log( object);
```
→ [{ values }]

DATE

const obj = new Object()
function call

function Object() { } → already defined in javascript

working of object constructor

(1) ⇒ prototype :

```
const obj = {}
```

[10x1]

prototype : object

```
const arr = []
```

[0x1]

prototype : array

map reduce filter etc

Obj/array ke andr values hoti hai and ek prototype v hota hai jo object/array ko help karte hai uske andr ke values pr kaam karke like array me reverse uske values ko traverse kar data hai.

(3). By using new keyword and Constructor function :- [khud se banayenge]

constructor

It is used to load and initialize all the non-static members inside an object.

function

Block of code that perform some specific task

⇒ A constructor function always written in upper cammel case.

eg:

CreateMango
MyName
pehla letter capital of each word.

new keyword → ye ek empty object create kar deta hai.
this → ?

DATE 115

// Normal function
function createObject()
{
}

// Constructor function.
function CreateFunction()
{
}

Qo. ① function CreateHuman(hname, age, gender, stateName)
{
this.hname = hname;
this.age = age;
this.gender = gender;
this.stateName = stateName;
}

const person1 = new CreateHuman("Abhay", 22, "male", "UP");

new keyword empty object creation.

Sabhi value return hoke yahan aajayegi.

hname: "Abhay"
age: 22
gender: "male"
stateName: "UP"

② function CreateCar (carName, price, color)
{
this.carName = carName;
this.price = price;
this.color = color;
}

DATE

const toyota = new CreateCar ('Fortuner', 55000, 'Black');
const Hyundai = new CreateCar ('Verna', 2000, 'Black');

clog(toyota); → {-3}
clog(Hyundai); → {-3}

⑧ // function as a parameter in Constructor function.

function CreateCar (carName, price, color, getDetails){
this.carName = carName;
this.price = price;
this.color = color;
this.getDetails = getDetails;
}

const hyundai = new CreateCar ('verna', 20000, 'Black',
function () {
clog('car Name is: ' + this.carName)
)) ('price is: ' + this.price)
)) ('color is: ' + this.color)
});

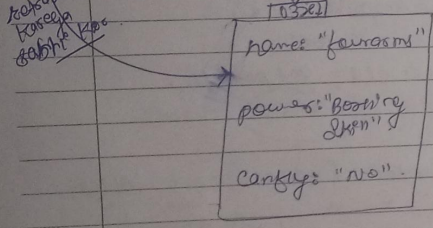
Hyundai.getDetails
↳ car name is: verna
price is: 20000
color is: Black

clog(Hyundai);
↳ {values}

(4). By using class :-

① class Omnitrix {
constructor (name, power, canfly)
{
this. name = name;
this. power = power;
this. canfly = canfly;
}
}

const alien1 = new Omnitrix("Four arms", "Beating Skin", "No");



② // function as a parameter passed.
class Omnitrix {
constructor (name, power, canfly, getDetails)
{
this. name = name;
this. power = power;
this. canfly = canfly;
this. getDetails;
}
}
const alien2 = new Omnitrix ("ultimate way Big", "strength", 'yes', function() {

c.log ('alien name is :' + this.name)
c.log (' power is :' + this.power)
c.log (' canfly :' + this.canfly)
})

c.log (Aliens);
→ {---}

Har baar naye object (like alien1) ke liye function create and pass karna hoga har baar jaise aliens me kiya hai upr. Iske khatam karne ke liye jis jis baar baar na likhna pade ham constructor ke alawa ek or function bana yenge.

class Omnitrix {
constructor (name, power, canfly) {
this. name = name
this. power = power
this. canfly = canfly
}
getDetails () {
c.log ('Alien name is :' + this.name)
c.log (' power is :' + this.power)
c.log (' can fly :' + this.canfly)
}
}

Const Alien1 = new Omnitrix ('Alien-x', 'infinity', 'yes')
Alien1.getDetails();
→ Alien name is: Alien x
power is: infinity
canfly: yes

Additional points

Object Method :-

functions jo object ke andar hote hai.

eg:-

```
const user = {
    name: "golu",
    age : 23,
    getDetails: function() {
        console.log("Name is:" + this.name)
        console.log("Age is:" + this.age)
        return 'user details showing on your screen'
    }
}
```

state (variables)

behaviour (methods)

user.name

user.age

* dot (.) →

console.log(user.getDetails) → f() {}

console.log(user.getDetails()) →

getDetails = method Name, () = parenthesis

method call.

* square brackets [] →

console.log(user["getDetails"]) → function code f() {}

console.log(user["getDetails"]())

user = Obj_ref, "getDetails" = Method Name, () = parenthesis

Add key value pairs in objects :-

eg:-

```
const student = {
    name: 'Akash',
    roll : 21
}
```

(1st Key & value pair)

(2nd Key & value pair)

0121
name: 'Akash'
roll: 21
country: 'India'

* dot (.) →

student.country = "India"

obj ref | Key Name | value

* using square bracket [] →

```
const actor = {
    name: "SSR"
}
actor["State"] = "Bihar"
```

Update a key value pairs in objects :-

```
const user = {
    name : 'chombu',
    city : 'pune'
}
```

0121	user
name	'chombu'
city	'pune' → 'Rohini'
State	'Delhi'

update → user.city = "Rohini"

Add → user["State"] = "Delhi"

↳ Delete a key value pair :-

```
const employee = {
        name : 'Akash'
        'employee salary' : 40000
    }
```

* using dot →

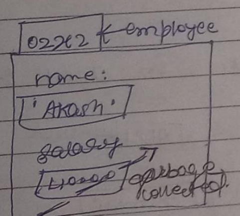
delete employee.salary

(delete = keyword, employee = obj ref, salary = key)

* using sqr bracket →

delete employee["employee salary"]

(employee = obj ref, "employee salary" = key)



↳ Check any key (property) is present or not :-

```
const employee = {
        name: 'Golu',
        age: 20
    }
```

c.log ("age" in employee) → true, otherwise false

("age" = key, in = keyword, employee = obj_ref)

→ Shallow copy simply jaise variable me doosra variable assign karte hai waisa hota hai.

2→ Shallow and Deep copy in object :-

① Swallow copy :-

```
let sweet1 = {
        name: "Ras malai"
        price: 40
    }
```

[shallow copy in sweet2 of sweet1]

let sweet2 = sweet1

c.log (sweet1) → Object {name: , price: }

c.log (sweet2) → object {name: 'Ras..' , price: }

• Update in sweet2 →

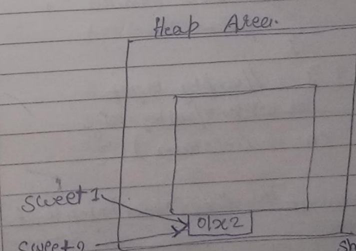
sweet2.name = 'milk cake'

c.log (sweet2) → object {name: 'milk cake', price: 40}

c.log (sweet1) → object {name: 'milk cake', price: 40}

↳ sweet2 ke name me change kiya to sweet1 ke name me v change ho gaya this is cause of shallow copy of object.

Heap Area.



↳ sweet1 ko sweet2 me shallow copy karne pr sweet2 v same memory address ko point karne lagta hai change hota hai.

DATE

② Deep copy for an object :-

Deep copy basically ek process hai jisme ham for-in loop ki help se ek ek key and value ko nikal ke doosre object me add karne ka process hai.

Is se original wala waise hi rahega but ek uska deep copy ban jayega jisme changes karne se uske original object me ko change nahi aayega.

Object :- for-in	Array :- for-of

→ syntax of for-in loop → (ek ek key variable name me jaayega each iteration me)

```
for( variable_name in ice cream (object_ref))
{

}
```

eg:

```
let iceCream ={
        flavour :" vanilla ",
        price: 60
    }

let harsh ={ }   //empty object.
                 1st key=flavor / 2nd key = price
//loop→  for ( element in iceCream ) {
    harsh[element]= icecream [element]
                    iceCream [ "flavour" ]
    harsh ["flavor"] =   "vanilla"
              add
```

PAGE

DATE

Process :- ① iceCream [element] → isme element ki jagah key hai usko insert karenge.

so, iceCream [" flavor"] → ye syntax simply keh raha hai, icecream naam ke andar jao and flavor key ko dhundo and uski value leke aao.

so, "vanilla" → to right side resolve hoke ye value aayegi.

process ② harsh [element] → yahan v element ki jagah jo key hai jo for-in loop se object (iceCream) se aayi wo insert hoga.

so, harsh["flavor"] → iska matlab v same wahi ki harsh object me jaao and flavor naam ke key ko dhundo. pr wahan nahi mile to ye key ke saamne agr koi value hai to inko as a key value pair ki tarah add kardo.

so, harsh ["flavor"] = "vanilla" → ye statement simply key: value pair ko add karne ka syntax hai and wahi hoga.

Result → Isi tarah for-in loop ek ek key laayega and ye poora process karega internally.

PAGE

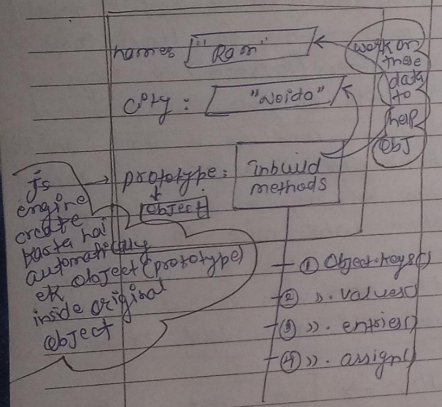
DATE

object inbuild methods :-

↳ comparison ek aise hi boss dekh rahe hai object me and array me kaise or kahan se inbuild methods aata hai.

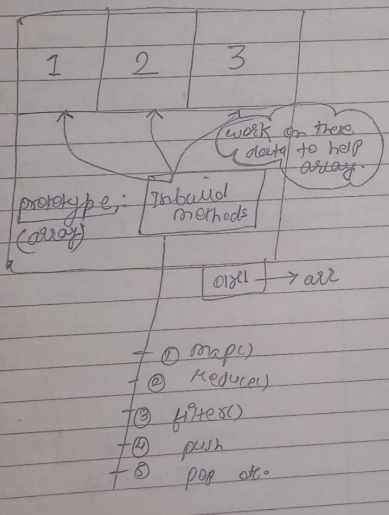
object

```
let obj={
    name:'Ram',
    city:'noida'
}
```



Array (also an object)

```
let arr=[1,2,3]
```



↳ Types of object inbuild methods :-

```
const developer={
    name:'Ramesh',
    loggin: true
}
```

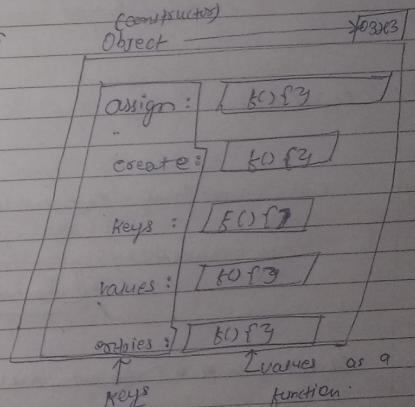

DATE

function Object()
↳ first letter of a function name is capital, ye constructor function me hota hai.

↳ object{ }
▼ prototype: object{...}
▼ constructor

name: 'Ramesh'
loggin: true
prototype: constructor: function Object
{ } means
key: keys() { key: function() { }
values()
assign()
entries()
freeze()
}

↳ Object() {
variable or func-key or obj-key
}



→ so, Object.keys() means Object constructor me jao and wahan se keys naam ka key leke aao.
Ab keys ek function ko dhoke kar kaafi hai keys ke aage parenthesis lagega. Same jaise object ke ando method as a value ko access karte the.

Simple meaning of Object.keys by me → obj.name() Jhamare define kiya hai us object me jao and name naam ke key ke value ko lekar aao

Object.keys() Js engine ne jo define kiya hai us object me jao and keys naam ke key ke value ko lekar aao.

127

(1). Object.keys (obj-ref) :- Returns an array of keys.
constructor property/function method

eg: let obj = {
name: 'Kamal',
isWorking: false
}

const output = Object.keys(obj);

c.log(output); → ["name", "isWorking"]

(2). Object.values (obj- ref) :- Returns an array of values.

eg: const valuesArray = Object.values(obj);

c.log(valuesArray); → ["kamal", false]

(3). Object.entries (object- ref) :- Returns an array of key, value array.

eg: const result = Object.entries(user)

c.log(result); → [["email", "abc@gmail.com"], ["password", 123]]

let user = {
email: "abc@gmail.com",
password: 123
}

PAGE

modify target object

128

DATE

(4). Object.assign (obj) :- It concatenate multiple object key:value pair into one targeted object.

Object.assign(, , , ... ,)
↓ Targeted object; object ref-1; object ref-2; object ref-n.

eg:

const company = {
"company name": "ED info Tech".
}

const requirements = {
post: "front-end",
salary: 20,000
}

const candidate = {
"candidate name": "golu",
skills: ["html", "css", "javascript"]
}

Target object

const result = Object.assign(company, requirements, candidate);

c.log(result); → Object {"company name": "ED info tech", post: "front-", salary: 20,000, "candidate name": "golu", skills: (4) [...]}

c.log(company)) → same like result

PAGE

→ par ham nahi chahte ki target object modify ho to ek better way hai assign karne ka :-

```
// create one empty object first
let output = {}
```

(ye v method se ho jayega pr abhi, {} bina naam wale object se implement kar liya.)

// rest of all those three objects also present from previous example.

```
const selection = Object.assign({}, company, requirement, candidate)
```

↑ targated object — obj-1st, obj-2, obj-3

c.log (selection);
↳ Object{"Company name": "ED infotech", "post": "front-end devo loper", "salary": 20,000, "candidate name": "golu", skills: (4) [...] }

c.log (company);
↳ Object{ "Company name": "ED infotech" }

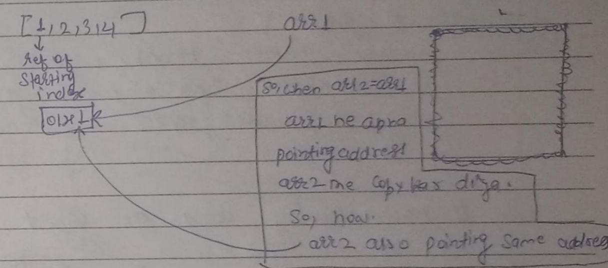
Shallow and deep copy in array :-

① Shallow copy:

① let arr1 = [1,2,3,4]
② let arr2 = arr1;
③ c.log (arr2);
↳ [1,2,3,4]

Hindi version → arr2 me 5 add kiya to arr1 me v automatically ho gaya.
→ element added by adding element in arr2 also in arr1

④. arr2.push (5);
⑤. c.log (arr1); → [1,2,3,4,5] → This happened cause of shallow copy of array
⑥. c.log (arr2); → [1,2,3,4,5]



② Deep copy of array :- (for-of loop)

① let movies = ["PK", "All the best", "welcome"]
② myFav = [];
③ for (element of movies) — movies me se har iteration me ek ek element ko leke jayega for-of loop and element naam ke variable me store karega further operation karne ke liye.
④ {
⑤ myfav.push (element);
⑥ }

⑧ c.log (movies);
↳ ["PK", "All the best", "welcome"] — ye nahi change hua due to deep copy

(isko movies ko console karne se pahle likha hai aisa samajhna)
⑦ myfav.push (" phir hera pheri", " Golmal-3")

⑨ c.log (myfav);
↳ ['PK', 'All the best', 'wel come', 'phir hera pheri', 'Golmaal-3']
↳ changes sirf dusre array me hua.

Imp for react

DATE

Object destructuring :- process of value ko get karne ka bina key name se without object reference and (.) dot or sqr bracket ke.

```
const obj = {
    name: "Kanishq",
    state: "Noida",
    pincode: 201301
}
```

// destructuring inside an object

```
const { name, state, pincode, city } = obj
```

201301 undefined "Noida" "Kanishq"

```
clog(name) → "Kanishq"
clog(state) → "Noida"
clog(pincode) → 201301
clog(city) → undefined.
```

- points to remember process of destructuring of an object :-

1. ek variable jene ka hai like const, let etc.
2. jo separator (grouping symbol) use kiya hai wahi likha hai.

eg:- let obj = {
}

↑ yahan {} ye wala grouping symbol (punctuator) use hua hai to yeh variable name ke baad hoga.

PAGE

DATE

(3) Grouping symbol ke andr keys ke naam same order me and har letter same likha hai, kyuki compare hoga and age match nahi° kiya to undefined aayega.

eg:- (1) const {name, state} = give undefined while calling (state)
↑ state hona chahiye

(2) const a = {
name : 'Ram',
age : 20
};

const {name, age} = a

aapos me compare honge ye and ye same order me xani, 1st name and second age hona chahiye. and spelling v same hona chahiye.

(4) fir , (=) assignment operator deke uske baad object reference or object naam dena hai jis v object ko hum de-structure karke chahte hai:

eg:- taking example from above example
Const {name, age} = a
↳ object_ref or object name.

→ ye saare process follow karke hum kisi object ko de-structure kar sakte hai tabhi fir's sirf keys ke naam se unke values ko get kar paaye.

PAGE

Imp for React

Array de-structuring:-

```
let movies = ['fast-x', 'Transformers', 'Avengers',
              'Terminator']
```

```
// Array destructure.
const [a, b, c, d, e] = movies
```

- ① 'fast-x'
- ② Transformers
- ③ 'Avengers'
- ④ Terminator
- undefined

```
// agr sirf do value leni ho to:-
const [a,,,d] = movies
```

- ① fast-x
- Terminator ④

```
c.log(a) → 'fast-x'
c.log(b) → 'Transformers'
  " (c) → 'Avengers'
  " (d) → 'Terminator'
  " (e) → undefined
```

• Rules to remember while de-structuring arrays:-

1. take a variable (same like object de-structuring).
2. use same grouping symbol (same like object-destructuring point)
3. Inside grouping symbol take same no. of any variable (unlike object jaruri nahi same naam ho). pr agr no. of variable jyada hue to unme undefined store kardega js engine.

eg:-

```
let a = [1,2,3,4]
const [a,b,c,d,e,f] = a
       1 2 3 4 undefined undefined
```

4. same like object, (=) assignment baad array-ref or array-name.

JSON:-

1. Stands for javascript object notation.
2. used to transfer data b/w applications through api's.
3. JSON keys must be strings enclosed in double quotes.
4. It support six data types: object, array, string, number, boolean and null.
5. Support nested structures, allow objects and arrays to be nested within each other.

(Inbuilt method part)

↳ JSON methods:-

1. JSON.stringify(value) (object reference):- javascript object or value ko JSON string me convert karta hai and JSON return karta hai.

• It does not support: function properties, symbolic keys and values and properties that store undefined.

→ Simple object me key ko ham bina double quotes me likh sakte hai pr jab unko hi ham double quotes ke andr likhe to wo JSON object kehlayega.

Normal object

```
let a = {
  name: "Ram"
}
```

JSON object

```
let a = {
  "name": "Ram"
}
```

DATE

↳ Stringify normal object ko json object me convert karta hai.

eg: // normal object
const user = {
name: "Ram",
age: 23,
isWorking: false,
State: {
stateName: 'Rohini',
city: 'Rohini',
pincode: 201301
}
}

// json object
① // JSON.stringify (object_refrence)

const output = JSON.stringify(user);
c.log(output);

↳ {"name": "Ram", "age": 20, "isWorking": false, "State": {"stateName": "Rohini", "city": "Rohini", "pincode": 201301}}

Sabhi key ko double quotes se enclosed kar diya.

② JSON.parse (object_refrence) (value)

Ye JSON object ko normal object me convert karta hai. return plain javascript method.

DATE

eg: // output taking from previous example.

const output1 = JSON.parse(output);

c.log(output1);

↳ { ... }

Same object jo shuru mein normal object, ~~abhi~~ jisme keys double quote se enclosed nahi hai.

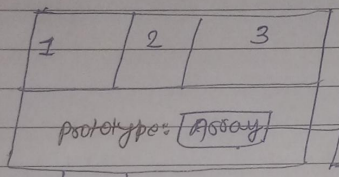
DATE

prototype :-

Kisi array me, object me ya function me jenginе ek prototype bana deta hai jo help karta hai unko.

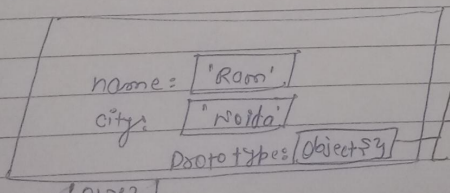
① In case of array,

```
const arr = [1,2,3,4]
```



② In case of object,

```
const obj = {
    name: 'Ram',
    city: 'Noida'
}
```



③ In case of functions,

```
function Something(){
}
```

PAGE

Something

This

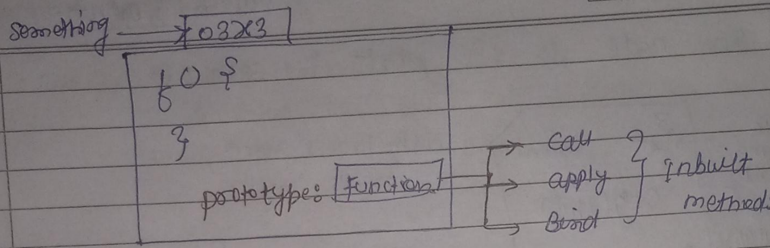
ye

(1) case-1,

Explana

imp

DATE



This keyword :-

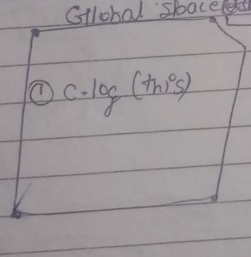
ye current object ko refer/point karta hai.

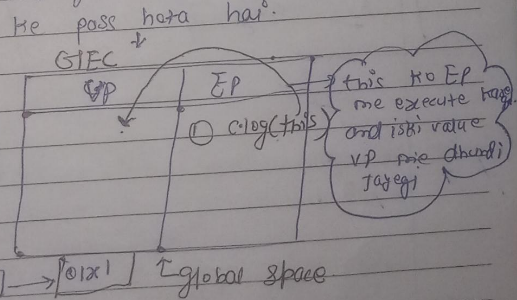
(1). case-1, when we write this in global space / or globally

c.log (this) → Ab ye "window" object ko point karega. and node js me {} curly braces ko.

Explanation : ① in browser, (window ko print karega). :-
Global space ~~global object window object hota~~

↳ and global space ko reference window ke pass hota hai.





Imp another view ⇒ window = {
c.log (this)
}
↖ point current object

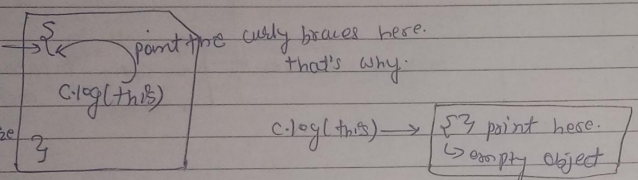
↓

Simpli baat hai, window ke andr object c.log(this) likha hua hai. tha this current object window hoga.

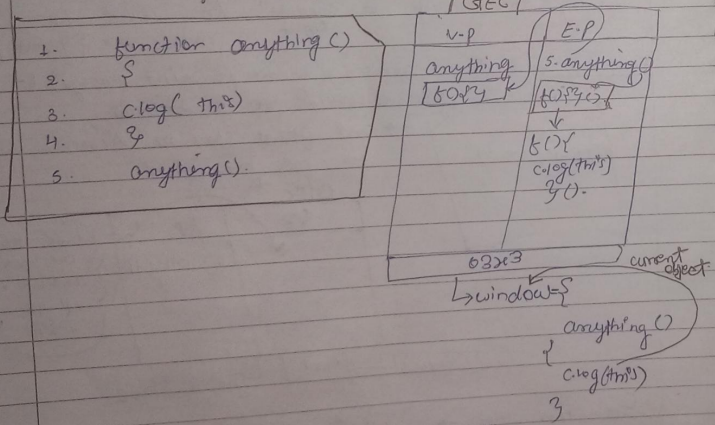
139

browser → window object print
nodejs → empty object {} print

(1) In node js {} print in case 1 :-



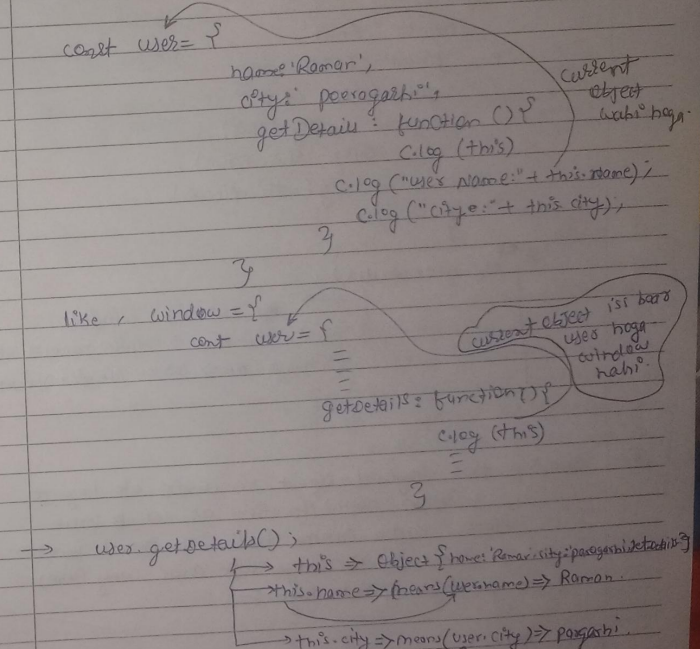
(2) Case-2, globally declare function ke andr this keyword console kiya to kya print karega :-



→ conclusion :- {} print refers window object because it is its current object.

→ Note:- poora ka poora global space samjho window object hai.

(3) Case-3, Ek object ke andr, ek key hai jiske andr function as a value hai and us function ke andr this ko console kar rahe hai to kya



PAGE

DATE

④ Case 4, this keyword arrow function ke andr hamesha window object ko hi point/refer karega :-

```
let obj = {
    name: 'Amit',
    pincode: 1234
    getDetails: () => {
        console.log(this);
        console.log(this.name);
        console.log(this.pincode);
    }
}
```

obj.getDetails();

→ this ⇒ window

→ this.name ⇒ (window.name) ⇒ undefined

→ this.pincode ⇒ (window.pincode) ⇒ undefined

⑤ Case-5, this keyword object ke andr ek key jo ~~function~~ nested function as a value store karke rakhe hai, us nested function ka jo inner function hoga uske andr console.log(this) karne par kya print hoga :-

```
let movies = {
    name: 'Gadman',
    ticket: 70,

    bookTicket: function () {
        console.log(this);
```

DATE

```
function inner(){
    clog(this)
}
inner();
```

```
window={
    movie =f
    getDetails: function(){
        clog(this)
        function inner(){
            clog(this)
        }
    }
}
```

current object print: {frame: 'Golmal', ticket: 70, bookTicket: bookTicket()}

ye V jaayega apne current parent ko dhundhne par isko ek parent function milega, isko koi current object nahi milega tab ye global me jaayega and global ka to window hai hi to window ko refer karne lagega.

Conclusion:- parent ke andr ka this → curr object and nested function (inner or child function) ka this → window (sabse bade) ke pass jaayega.
ko print/refer karega.

DATE

Function inbuilt methods 8-

- call, Apply and Bind methods are used to store object refrence inside "this" keyword of a function.

- when function's 'this' have refrence of objects tab ham uss object ke states(variable) and behaviour (methods) ko access kar sakte hai.

↳ Need (To ek problem aati hai object ko de-structure karte time) 8-

1st need.

```
let obj = {
    name : 'Ram',
    city : 'Noida',
    getDetails : function () {
        c.log(this.city)
    }
}

// de structure object
const { name, city, getDetails } = obj;

c.log(name) → Ram

// problem arise when,
getDetails();
    ↳ undefined !?
```

→ ① pehli cheej jab getDetails destructure hogi to wo uss object se bahar aajayega and global me aa jayega and tab uske andr ka this, current object jo ki ab window hai uske refers karne lagega and isliye this.city = undefined dega.

DATE

eg:-

```
let obj = {
    name: 'abc',
    getDetails: f() {
        c.log(this.name)
    }
}
```

• now (abhi), destructure karne se pehle

```
// de-structuring

const { name, getDetails } = obj

• After de-structuring kaisa dikhega exactly →

let obj = {
    name: 'abc'
}

getDetails = f() {
    c.log(this.name)
}
```

yahan pr getDetails bahar nikal gaya hai.

And ham jante hai ki agr function global me ho to wo, this window object ko point/refers karta hai.

Note:- To pehli to iss problem ko solve karta hai method/ function inbuilt methods.

• Jab de-structure kar dete hai, ham tab bhi chahte hai ki jo getDetails hai wo uss object ke refers kare bhale hi we ab bahar nikal chuka hai to aise case me call, apply and bind use karenge iss cheej ke achieve karne ke liye.

PAGE

DATE

② Second need :-

- Koi function kisi object ke bahar hai shuru se hi and hum chahte hai ki uska this → kisi v particular

Object ko refer karne lage tab bhi hum ye use karte hai.

eg: let obj = {
  name : 'Ram'
}

function show_details () {
  c.log (this. name)
}

output should come = Ram.

⇒ Types and implentation of function inbuilt methods :-

① call () :-

- call method ek object reference as a first argument and uske baad 'n' no. of ames arguments leta hai.
- And baaki normal argument hote hai.
- ye function ko immediately call kar deta hai.

DATE

→ Syntax :-

function_ref . call (object_ref , a1 , a2 ... , an)
or
function_name object_name

eg: ① (need and) const obj = {
  name: 'Raman',
  city: 'Noida'
}

function show_details (pincode, login_count, state) {
  c.log (this)
  c.log ("User's name :" + this.name);
  " (this. city);
  " (pincode);
  " (login_count);
  " (state)
}

show_details . call (obj, 201301, 4, "up");

this → object { name: Raman, city: Noida }
this.name → Raman
this.city → Noida
→ 201301
→ 4
→ UP
} these are normal argument

need first

② const obj = {
name: 'Raman',
city: 'Noida'
getDetails: function() {
c.log(this);
c.log(this.name);
c.log(this.city);
}
}

const { name, city, getDetails } = obj;
// after de-structing
getDetails();

this → window
this.name → means (window.name) → Undefined
this.city → means (window.city) → undefined

// after de-structuring but applying call.

getDetails.call(obj);

this → object { name: 'Raman', city: 'Noida', getDetails: function() { } }
this.name → (obj.name) → Raman
this.city → (obj.city) → Noida

2. apply() :-

ye sirf do arguments ko support karta hai, jisme 1st argument object ref hota hai and 2nd ek array of arguments hota hai.

ye v call() ki tarah function ko immediately call kar deta hai.

↳ Syntax :

function ref . apply (object ref, [a1, a2, ..., an])
or function name or object name (always array)

eg: // same call() wali object and showDetails function (2nd need) hoga.

showDetails.apply(obj ref, [201301, 4, "UP"])
pincode, city, state

this → object { }
this.name → obj.name → 'Raman'
this.city → obj.city → Noida
→ 201301
→ 4
→ UP

DATE

3. bind() :-

- ye ek object reference as a first argument leta hai and uske baad 'n' of arguments (same like call() method).
- ye function ko immediately call nahi karta balkhe call() and apply() method.
- ye ek function return karta hai "Bound" fir hum iske aage () paranthesis lagake isko call karenge.

eg:-

```
const obj = {
    name: 'Ram',
    email: 'Ram@gmail.com'
}

function printDetails(comp, salary, city){
    console.log(this);
    console.log(this.name);
    console.log(this.email);
    console.log(comp);
    console.log(salary);
    console.log(city);
}

printDetails.bind(user, "Testyantra", 4, "Hyderabad")
```

bind → return a method called → Bound method

// Bound method ko store karaenge and uske aage paranthesis lgake usko call karenge.

const

this

this.

this or

// con

function

}

functio

setName

}

DATE

const result = printDetails.bind(user, "Test Yantra", 4, "Hyderabaad");

result();

this → object { ... }
this.name → obj.name → Ram
this.email → obj.email → Ram@gmail.com
→ Test Yantra
→ 4
→ Hyderabaad

this and call() used in this example;
// constructor function case

function setName(name) {
console.log("set Name function called");
this.name = name;
}

function CreateObject(name, age, city) {
console.log("Inside constructor function");
setName.call(this, name);
this.age = age;
this.city = city;
}

(employee) ye object ka reference store karke rakha hai, and iske ref se pass kiya setName me for jab wahan likhega.
this.name = name
tab iska matlab hoga,
employee.name = name

const employee = new CreateObject("Raman", 24, "Delhi");
console.log(employee) → Object { name: "Raman", age: 24, city: "Delhi" }

Rest and Spread :-

1. Rest parameter :-

- Rest ek parameter hota hai jo multiple or "n" no. of arguments ko accept karta hai and unko ek array me store karega. and uss array ka naam wahi hoga jo Rest parameter ka naam hoga.

- Kisi variable ko Rest variable/parameter banane ke liye hame uss variable ke aage '...' three dot lagane hote hai.

  eg:- ① let ...variable_name;
  ② ...a;
  ③ ...roll_no's

- Isko ham uss function me use kar sakte hai, jab hame pta nahi ho ki kitne arguments aane wale hai.

eg:- ①.

```
function roll_numbers(...roll){        → roll=[a1, a2, a3, ... an]
                                               0   1   2        n-1
    ...roll.map(val, index, array){
        c.log(val);
    }
}

roll_Numbers(1,2,3,4);
   ↳ 1
     2
     3
     4
   output
```


②.

```
function ShopingCart(... Item){
    ↳ ["cold drink", "Ice cream", "cake", "Ros malai"]
          0             1           2        3

    c.log(Item[0]);
      ↳ cold drink
    c.log(Item[2]);
      ↳ cake
}

shoppingCart("cold drink", "Ice-cream", "Cake", "Ros-malai");
```

2. Spread parameter :-

- ye elements ko unpack karta hai.

// use case.

- Isko ham as an argument bhejte hai function me.

eg. ①

```
let arr = [1,2,3,4,5]

function Sum(a, b, ...c){        → Rest parameter
    let sum = a+b+c[0]+c[1]+c[2];
}

Sum(...arr);
  ↳ Spread parameter ← [1,2,3,4,5]
                          Isko unpack kardega
                          1,2,3,4,5
                          Isko ek ek a, b ko bhejega and bacha hua sab ...c ko.
```

Prototype → this keyword → function inbuilt method → Rest & Spread = 4

DATE 153

② const employee = {

skill: ['FE', 'BE'];
salary: '5LPA';
}

const candidate = {
name: 'Ram'
city: 'Noida'
... employee → as a spread parameter
}

skill and salary key jaisa include ho jayega candidate me.

clog (candidate);
→ object{ name: 'Ram', city: 'Noida', skill: [---], salary: '5LPA'}

③ let arr = [1,2,3,4,5] → [1,2,3,4,5]
function findSum(...a) {
const sum = a.reduce((acc, val) => {
acc = acc + val; return acc;
}, 0);
return sum;
}
clog(findSum(...arr)); // 15

Iske pass ye value aayegi jinko hme ek array me store karaa liya.

spread parameter