Arr. prototype.sum() { }

prototype

Inheritance in Javascript in

Imp Interview

① Javascript is ↑ which language Synchronous language.
It execute the code line by line.
It is single threaded and
It execute the code from top to bottom.

task are performed one at time
It the code goes line by line.

② To ~~tell me~~ about JS.
① object-oriented language
② JS is single threaded, Synchronous language.
③ JS is scripting language
④ JS is dynamic browserent dom.
⑤ world most popular language

③ JS engine.
responsibility of render ~~in~~ JS on webpage
example ~~browser~~ engine.
Chrome → V8
m.e → chacra
~~Safari~~
mozilla firefox → spidr monkey
Apple → safari

Second smallest even prime

④ How JS code is run on browser

~~what is execution context~~
execution contex
because.

① what is diff ebetween var, let, const

var, let, const are keywords.

var = global scope.
let = block scope.
const = block scope.

var ⇒ can do declaration
can reinitalised again.
~~let~~
let = can be reinitalised again
can't be declared.
const = can't be reinitalised.
can't be ~~declared~~ declared.

e.g.

```
var a = 10;
console.log(a)
var a = 20;
console.log(a)
```

```
var a = 10;          — declared + initialize
console.log(a)
a = 20               — reinitialised
console.log(a)
```

```
let   a= 10;
  console.log(a);
let   a=20;
console.log(a);
```

error → can't declared again

already declared one variable not declared again (let).

```
let   a=10
  console.log(a);
a=20
console.log(a);
```

10
20

```
const   a=10;
  console.log(a)
const a= 20;
  console.log(a);
```

redeclared again error because

Const can't be reinitialized and redeclared.

```
const a=10;
   console.log(a)
a=20                    — reinitialized
 console.log(a)
```

output
10
error

Here we do are not declared again but we reinitialized it so but we can't redeclared and reinitialized Const.

⑤ Hoisting.

It is behaviour of JS.

~~We can access the variable~~ to move function variables to the top of their scope.

It allows us to use variables before it declaration.

let and const can't be hoisted if will give you reference error.

In JS initialization is not hoisted.

```
var a=10
function add(){
    console.log(a)
    var a=20;
}
add();
```

There are two types.

① variable hoisting

② function hoisting

eg

①
```
console.log(a)
var a=10;
```

② hoisting with function

```
a();
function a(){
    console.log("a")
}
```

e.g.

```
a();

function a(){
    var b=4
    console.log(b)
}
```

(6) Temporaral deadzone

```
console.log (a)
to var a=10;        } undefined

var a
console.log(a)      } undefined.
a=10.

let a;
console.log(a)      } undefined.
a =10;
```

If we tried to access the variables using let and const before their initialization. then it gives us reference because they are in block scope, unless and until they declared

```
a=10
console.log(a)      } to
var a;
```

Page No.
Date

output 10.

value assign first
↓
a=10; ← ~~declaration~~
console.log(a)
let a; ← ~~initialization~~ declaration

if we use var instead of let and const the output will be 10.

reference error.

give this example.

a is in temporal dead zone until and unless it declared that area is called temporal dead zone.

⑥ execution context

① execution context our JS code is being executed /executing and ~~there is JS engine who help us to~~ and every browser is having JS engine which help us to execute JS code. and ② I say something about ~~JS~~ JS engine

① JS engine uses a callstack and also create a special environment to handle the execution of JS code.

② ~~In~~ execution context global ~~function~~

- There are two kind of context
① global ~~cont~~ execution context
② function —11—

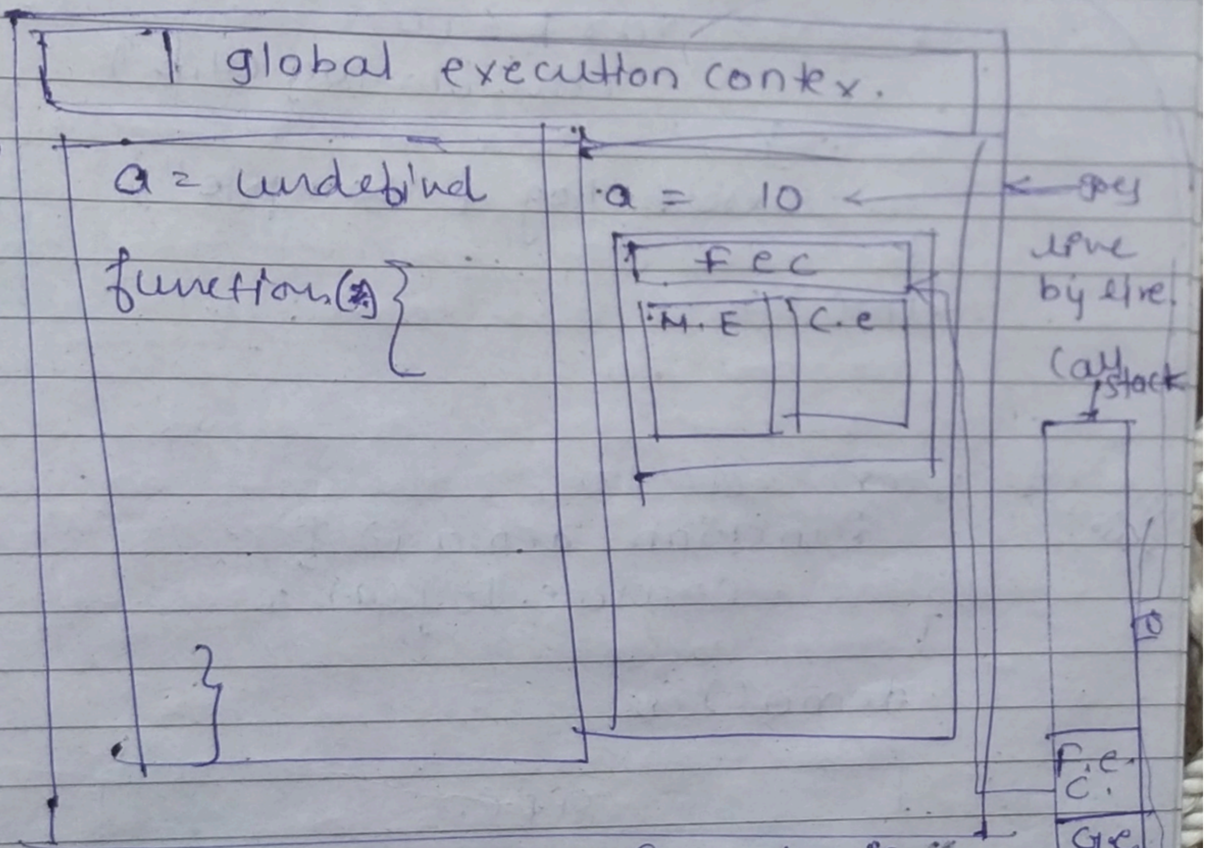
- There are two phase in ~~ex~~ running behind execation context.
  (1) creating phase
  (2) execution phase

- memorry alloction * variable, function declartn ~~allocation~~ is in creating phase

- ~~code run in ECO is~~ we can execute is code line by line in execution ~~code~~ phase.

→ can explain with the help of diagram.

```
var a = 10
function abc() {
  let b = 20
  c = a+b;
  console.log(c)
}
abc()
```



(5) callstack is mechanism to keep track of its place that call multiple funn. what funcn is currently being run and what funn are called from a funn that define

(1) In callstack it includes both f.e.c g.ec first in last out

(2) All process running inside callstack

execution context
- global e.c
  - memory
  - code context execution
- function e.c
  - M.E
  - C.E

④ It store f.e.c and g.e.c in lifo manner last in first out

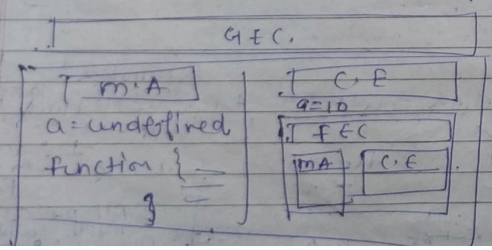
~~First f.e.c - pop out~~

When our callstack empty
① No code is written.
② our code will run sucessfully

```
function abc () {
  var/let a = 10;
      function sample () {
        var b = 10
        console.log (a,b)
      }
    console.log ("sample")
}
abc ();
```

e.g. Give this

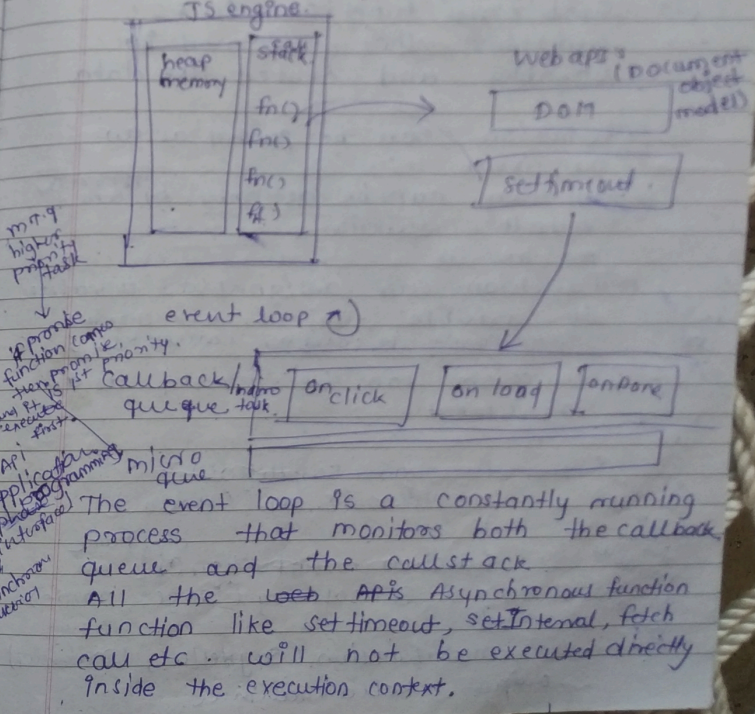
```
var a = 10;
function demo () {
  console.log (a);
}
demo();
```



Day -02.

① What is event loop and callback stack
② what is meant by first class function.
③ what is call back hell.
④ explain call(), apply() and bind() method.
⑤ what is pure function.
⑥ what are closure.
⑦ what is callback

① What is event loop and callstack?



The event loop is a constantly running process that monitors both the callback queue and the callstack. All the ~~web APIs~~ Asynchronous function like setTimeout, setInterval, fetch call etc. will not be executed directly inside the execution context.

firstly they will be moved to callback queque and then the moment callstack gets empty these functions will be pushed to call stack from callback queue by event loop.

This moment check by event loop

If the call stack is not empty, the event loop waits until it is empty and places empty.

② first class function in JS.
→ functions that are stored in variables and treated like data

① function can be passed as an argument
② function can be returned from a another function
③ function can assigned as a value to variable, object or array then can't be called as callback()

① function can be passed as an argument.

We are store one fun inside of one variable and treated that variable as function

e.g. function sayHello() {
return console.log('Hello')
}
function mainfuction(arg, name) {
console.log(arg(), name)
}
mainfunction(sayHello, 'Mahima')
output :- Hello mahima

e.g. const add = function() {
console.log("Hello");
}
add();

① e.g.
function sayHello() {
return ('Hello')
}
function mainfunction() {
return
console.log(
main func

function sayHello() {
return ('Hello');
}
function mainfunction(arg, name) {
console.log(arg, name);
}
mainfunction(sayHello(), 'Mahima');

output :- Hello mahima.

② function can be returned by an another function

function add() {
return function() {
console.log('added');
}
}
add()()

because we return the function so this syntax

③ can be assigned as a value to a variable, object or array.

① variable.
const abc = function sayHello() { }

② for object.

```
const newObj = {

    getName : function abc() {

        }

};
```

③ for array.

```
let arr = [];
arr.push(function abc() { });
```

③ what is callback hell.

→ callback Hell is a pattern with multiple nested callbacks which makes code hard to read and debug when dealing with asynchronous logic. The callback Hell look like a pyramid structure.

④ ~~callstack~~ :-

hard to manage, dealing with ~~formation~~ is ~~the~~ asynchronous logic

callstack

Can we Synch. and Asynch both?

→ yes we can

to execute Asy funct or event loop.

event loop : contantly running process that monitors both the call back queue and call stack.

~~AH th~~

①

① diff. setTimeout() and promise().

callback queue. several

② callback queue consist async fun from web API. e.g. setTimeout, setInterval, onclick, fetch.

- web API consist of all synch. fun

* web API present in every browser. ~~and~~ all asynch fun and our DOM part already present inside ~~in~~ of web API.

* ~~event~~

③ pure function :- ① It always returns the same result if the same argument is passed. ② It doesn't ~~It does not depends~~ depend upon any state or data change during its program execution.

e.g.:

```
const add = (a) => {
    return a+5;
}
console.log(add(3))
```

outputs- 8

impure fun°.

```
var b=5;
const add = (a) => {
    return b*a+5;
}
console.log(add(3))
```

impure because o/p depend upon b which is declared in global scope.

④ callback function
any function that passed as an argument in another function is called as callback function.

the functn is import in another functn

why we need callback() fun°
instead of we JS is an event driven language instead of waiting response JS will keep execute code while listing the other code.

e.g.

```
function sum (a,b) {
    console.log (a+b);
}
function dom (num1, num2, call) {
    call (num1, num2)
}
dom (2,5, sum))
```

e.g.

```
function callBack (name) {
    console.log ('Hello' + name);
}
function outerfunction (callBack) {

function greet (callback) {
    console.log ('Hello');
    callback();
}
function sayGoodBye () {
    console.log ('GoodBye!');
}
greet (sayGoodBye);
```

why we call callback() inside of outerfunction

If P1 P2 not store the context of sayGood by fun°.

* call(), apply(), bind().

These are normal method and or normal approches to call the function.
and it has some advantages.

* How we show the object

```
let obj = {
    name : "malima",
    age : 22;
}
function display (){
    console.log(obj)
}
display();
```

```
let obj = {
    name : "malima"
    age : 22
}
function display(){
    console.log(obj.name + " " +
        "is" + " " + obj.age +
        "year old");
}
```

```
let obj = {
    name : "malima";
    age:22;
}
function display (a) {
    console.log (a.name + " " + "is a" +
        a.age + " yearsold")
}
display (obj).
```

Advantages

① By using this approches we don't have to defined the object as a parameter and we don't have to use the object name to access the individual values.

e.g.

```
let obj = {
    name : "malima";
    age : 22;
}.
function display () {
    console.log( this.name + " " + "is" + " " +
        this.age + "yearsold")
}
display.call (obj).
```

this is keyword to refer to an object call is fun used to call object.

② In case of apply each and every argument which is req. to be passed to the fun^n. will be passed inside of array

③ but in case of bind it will return a new fun^n and this new function can use anywhere, any time, any where.

* closure

If variable and function is present in the same scope, then the functn is capable of accesing the variable.

closure is comb^n of fun^n & and lexical environment

lexical environment means nothing just surrounding state.

we can access outer function ~~var~~ scope from inner fun^n.

syntax:-

```
function outerfunction() {
    const x = "hello"
    function innerfunction() {
        console.log(x);
    }
    return innerfunction;
}
```

while comparing Null and undefine == return true in === it return false

Strickly comparision operator

comparision operator

① what is the difference between "==" and "===" ?

→ ① Both are comparison operator.
② and they return boolean values always.
③ The difference between both operators
① == is used to compare values
② === is used to compare both value and data type.
④ console.log(x == y) :- It return true because it only check values not data types.
⑤ console.log(x === Y) :- It returns true because you can see that values are same but x is of string type and y is of Number type.

② what is spread operator ?
The JS spread operator (...) allows us to destructure the non-primitive data types like Array and object to access elements individually.
e.g.

```
const arr1 = [1,2,3];
const arr2 = [4,5,6];
const combined = [...arr1, ...arr2]
console.log(combined);
```

output :-
[1,2,3,4,5,6]

③ what is the setTimeout ?
→ ① The setTimeout() is the asynchronous function. ② The setTimeout() is a

method inside of window object, it calls the specified function or evaluate a JS expression provided as a string after a given time period for only once.

we all use alarms or reminders several times, the setTimeout() method also has the same purpose in web application.

We use this to delay some kind of execution.

④ What is the purpose of async/await keyword?

→

① An async function can be declared with the async keyword. and await keyword permitted within it.
② It avoid the need of promise chaine.

The use of await posses async function until our promise returns a result.
async fun^n is fun^n declared with async function.
Basically async keyword declared with asynchr. funct^n and we have to await keyword inside our async function. we do and the await keyword posses the async fun^n until the promise returns a result. we can use async and await keyword to avoid the promise chaining and to write more cleaner code.

Callback, hell, async await, Promise, promise chain, IIFE, closure, fun^n, execution context.

⑤ promise → It has three stage.

① Pending
② Resolved (If operation successfully)
③ Reject (If error occur)

simple code for promise.

①
```
new Promise ((resolve, reject) => {
    setTimeout (() => {
        console.log ("promise run")
    }, 2000)
});
```

②
```
By using .then and .catch

const promise = new Promise ((resolve,
                reject) => {

    setTimeout (() => {
        resolve("promis run succesfully");
    }, 2000);

promise.then((promisedata) => {
    console.log (promisedata)
});

promise.catch ((promiseerr) => {
    console.log (promiseerr)
});
```

our promise set two parameter resolve, reject.
if our promise resolve it goes to .then()
if our promise reject it goes to .catch()

what are promise and why do we need them.
Promis is introduce to manage multiple asynchron function specifically in callback hell, it and!
① promise are used to handle asynchronous JS. operation in JS.
② They are easy to manage when dealing with asynchronous operation
③ promise are the ideal choice for handling asynchronous operation in the simplest manner.
④ It is has 3 stages.
① pending := either fulfilled nor rejected.
② Resolve = operation completly fulfilled.
③ reject = operation failed

⑤ Advantages of promises.
① Improves code Reliability.
② Better handling asynchronous operations
③ Better Flow of control asynchronous logic.
④ To mange the code in more efficient way. ⑤ for better error handling.

⑥ what is DOM?
→ ① DOM is a document object model
② DOM is the data representation of the objects that comprise the structure and content of document on the web.
③ DOM is the programming interface of the web document.
④ It represents the page so that program can change document structure, style and content.
⑤ with the help of DOM programming language can interact with the page.

promise chain :- the process of executing multiple asyn. function one after another using promise. we have to use multiple .then methods one after another to perform operation task

⑦ diffference between undefined vs not defined. Vs Nan.
→
undefined :- Is assigns undefined to any variable that has been declared but not initialized.

Null :- It is one JS's primitive data type and is treated as Falsy for boolean operations.

Not defined :-
A not defined is a variable that is not declared inside the code at a given point of time with declaration keyword like var, let, or const.

⑧ what are the constructor function in JS?
→ ① Constructor is nothing but the object is created using the new keyword.
② we have multiple constructor like
① Function constructor For function.
② Object constructor for object.
③ Array constructor ~~function~~ for arrays.

example :-

```
function User (Fn, Ln) {
    this.Fname = fn;
    this.Lname = Ln;
}
var obj1 = User new User ("malina", "pawar")
var obj = obj.age = 22
var obj1 = new User ("nonu", "pawar");
```

⑨ explain prototypes :-

→ ① Every object in JS has built-in-property, which is called its prototype.

② prototype itself is an object, so that prototype will have its own prototype called as prototype chain.

⑩ map(), reduce(), filter().

→

diff :-

map() , return new array doesn't change main array

filter() ,

reduce() , return single one output.

⑥ map , returns new array. does not change the length of array. it returns an array of same length as the original array

① let arr = [1,2,3,4,5]

filter :- return subset array

⑦ reduce : reduce a array to single value by applying a function to each element in existing array.

map always return new array. it doesn't return new array it just updates main array

③ let arr = [1,2,3,4,5]
let arr2 = arr.find((num) => {
return num%2 == 0;
});
console.log(arr2)

④ let arr = [1,2,3,4,5]
let arr2 = arr.sum((num) => {
return num%2 == 0;
});
console.log(arr2)

both

⑤ let arr = [1,2,3,4,5]
let arr2 = arr.even((num) => {
return num%2 == 0;
});
console.log(arr2)

e.g. let arr = [1,2,3,4,5]

① let arr2 = arr.map((num) => {
return num*2
});
console.log(arr2);

② let arr = [1,2,3,4,5]
let arr2 = arr.filter((num) => {
return num%2 == 0;
});
console.log(arr2)

③ reduce().

```
let arr = [1,2,3,4,5];
let arr2 = arr.((num) => {
        return num %2 ===0;
});
console.log(arr2)
```

```
let arr = [1,2,3,4,5];
let arr2 = arr.reduce((accumulator, currentVal
        => {
            return accumulator + currentVal
        });
console.log(arr2);
```

accumulator and currentvalue are fixed parameter.

① accumulator is privious value.

② current value is current value.

start index [0] ← accumulator is indicate index.

start index [1] ← current value is indicate element of array at particular index.