PAGE NO.:
DATE: / /

Advantages of closures ⇒

1. It is used in function currying.
2. It helps us in data hiding and encapsulation.

Data hiding :- Other functions will not be able to access other variable.

```
var counter = 0;
function incrementCounter() {
    counter++
}
```

(Anyone can access it)

Now with the help of closure, no other person can use it.

```
function outer(){
    var counter = 0;
    function increCounter(){
        counter++;
    }
}
console.log(counter);
```

Reference error: not defined.

Now, to get the value → we can simple return the increCounter function. and add console.log(counter) inside function.

```
var outer1 = outer();
```

outer1 → contains increCounter fun()

If we call again that will a different copy and counter value will be 1 only.

PAGE NO.:
DATE: / /

Function constructor in JS ⇒

```
function Counter(){
    var count = 0;
    this.incrementCounter = function() {
        count++;
        console.log(count);
    }
    this.decrementCounter = function(){
        count--;
        console.log(count);
    }
}

var counter1 = new Counter();

counter1.incrementCounter(); → 1
counter1.incrementCounter(); → 2
counter1.decrementCounter(); → 1
```

PAGE NO. :
DATE : / /

Disadvantages of Closures ⇒

* There could be a overconsumption of memory.

Garbage Collector :-

It is a program in browser or JS engine which clears the unutilized memory.

Like in c, cpp, memory allocate and deallocate done by developers but in JS, it is done by JS engine.

Relation between garbage collector and closures :-

```
function a(){
    var b = 10;
    return function c(){
        console.log(b);
    }
}
a()();
```

So, right now closures is making, so b can't be freeze c() has reference to b variable

but if closure will not make then b should clear after run the program.

But some smart engines do has some smart garbage collector like V8 like which variables has never used, V8 engine clear that variable.

```
function a() {
  var x=0; z=10;
  return function b() {
    console.log(x);
  }
}
var y = a();

y();
```

if we print console.log(z) in console, it will print error. (reference : x is not defined)

PAGE NO. :
DATE : / /

Anonymous function:-
A function which don't have any name.

~~Nested function~~
Anamyroy function used as values

Function statement =>

function a {

}

Function expression =>

var b = function () {

}

→ The difference between both is hoisting

PAGE NO.:
DATE: / /

function declaration ¶ →

function statement and declaration both are same thing.

Named function expressions

function with names.

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

```
var b = function xyz(){
    console.log("b called");
};
```

xyz() → error

↳ Imp. question.

Parameters and arguments ⇒

```
function a(n){          ↘ Parameter.
    console.log(n);
}
```

a(5) ↳ argument

First class functions ⇒

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

b(function{

    });
```

⇒ 1. Passing function as argument

The function to use values and passing as argument is called first class functions.

left → closing, arrow function

PAGE NO. :
DATE : / /

First class citizens →

First class functions are same as first class citizens.

[redacted]

Callback function

```
function x(y){

}

x(function y(){

});
```

Callback :- It gives the responsibility to different function, it will callback later, that's why we called these functions as callback.

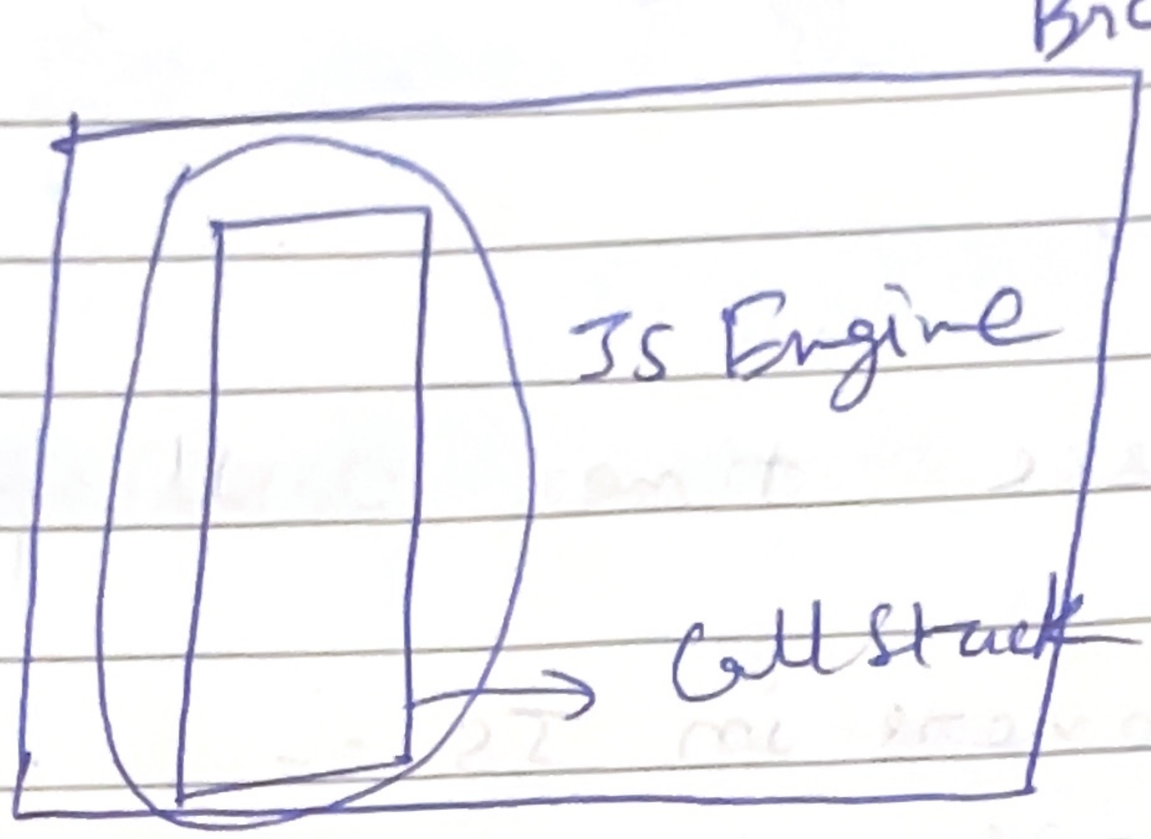
```
setTimeout(function() {
    console.log("timer");
},5000);
```

After 5 sec., timer will print.

Event listeners in JS :-
with closures

```
function attachEventListeners(){
    let count = 0;
    document.getElementById("clickMe")
    .addEventListener("click", function
    xyz(){
    console.log("Button clicked", ++count)
    })
}
attachEventListeners();
```

Event LOOP



Web APIs

window → global object

setTimeout()

DOM API

fetch()

LocalStorage

console

location

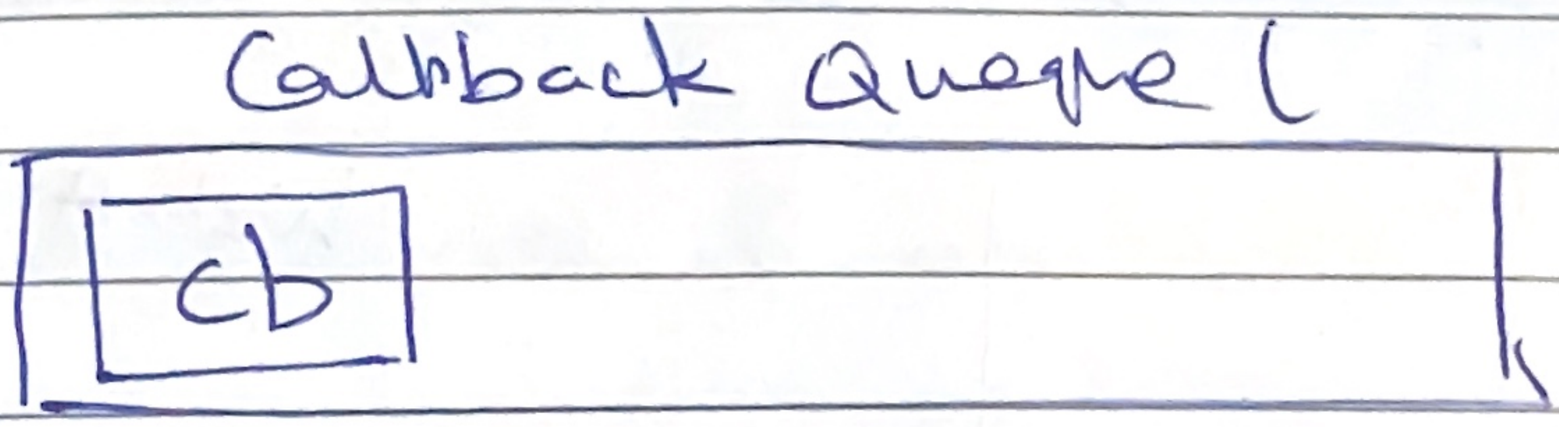
There Browser provides this.

These all web api's present in the window object.

```
console.log("start");
setTimeout(function cb(){
    console.log("callback");
},5000);

console.log("end");
```

When 5 sec. will expired, this callback function, ~~core~~ go to callback queque



Now event loop checks if there is something in the callback queque, it moves to the callback to the call stack.

PAGE NO.:
DATE: / /

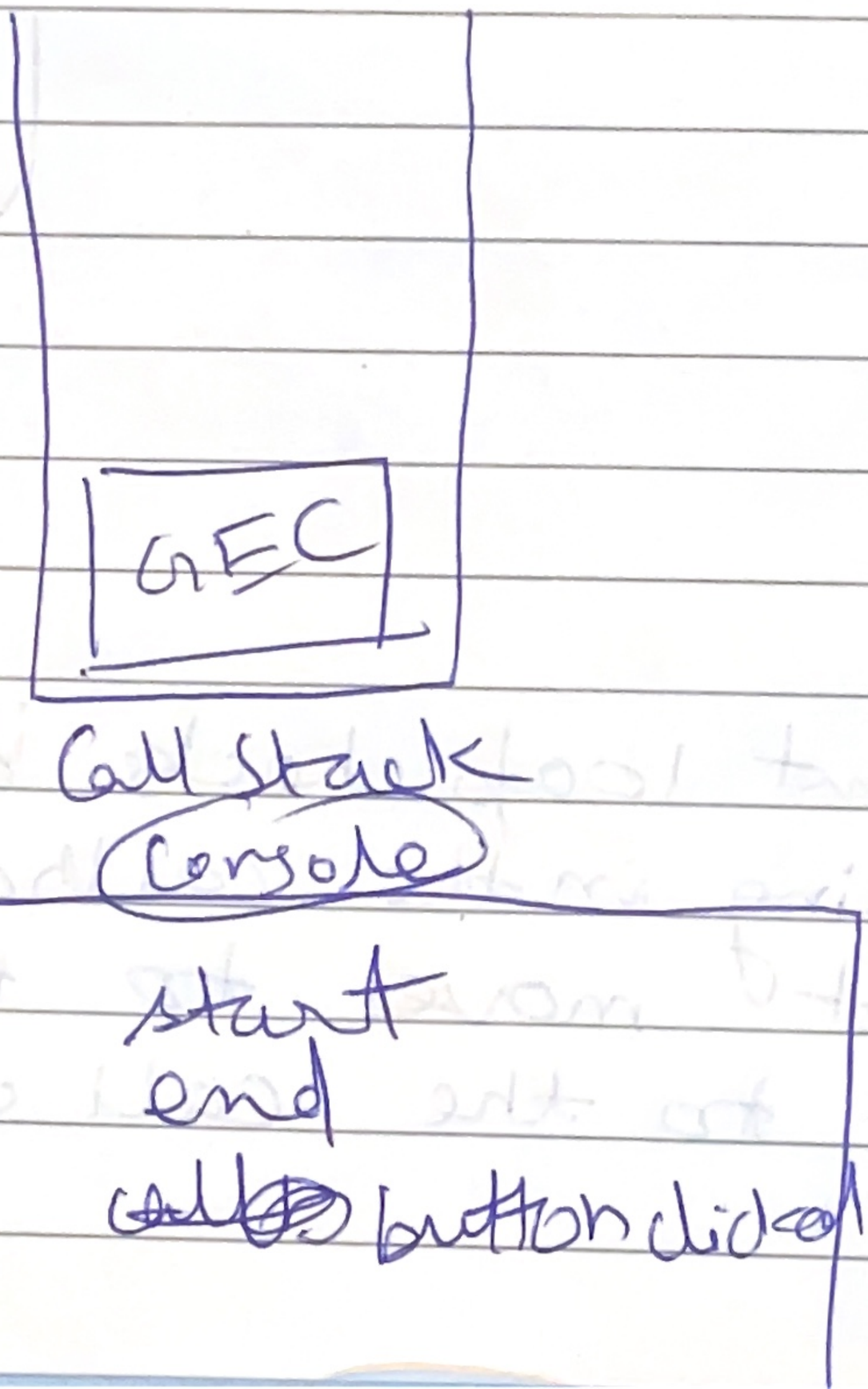
Some with Eventlistener =>

```
console.log("start");

document.getElementById("btn").
addEventListener("click", function cb()
        console.log("button clicked");
3);

console.log("end");
```

When this code will execute.



a callback is attached to click event, and when we click on to that button, it will go into the callback queque, and then event loop will check if call stack will be empty then it put the cb in callstack and execute.

How fetch (web api) works?

```
fetch("http://www.google.com");
```

↳ It returns a promise,

we have to pass a cb function which will execute once this promise is resolved.

```
console.log("start")
setTimeout(function cb(){
    console.log("Timeout");
},5000);
```

```
fetch("http://www.google.com")
.then(function cbF(){
        console.log("Google");
});

console.log("End");
```

① In callstack, GEC is created and it console.log("start")

② In WEB APIs environment cb function is attached to the timer

③ Save with fetch request when ~~promise~~ will be resolved callback attached to the fetch APIs

④ End will print

⑤ Meanwhile fetch response will wait in microtask queque

⑥ and callback fired

Console

start
end
CB Google
Timeout

will wait in callback queque

1. And eventloop is checking continously when QEC will popped out it will give first piriority to microtask queque then callback queque.
2. And puts then into callstack queque

What can be come inside microtask queque →

All cb which comes through promises goes inside microtask queque..

and mutation observer.

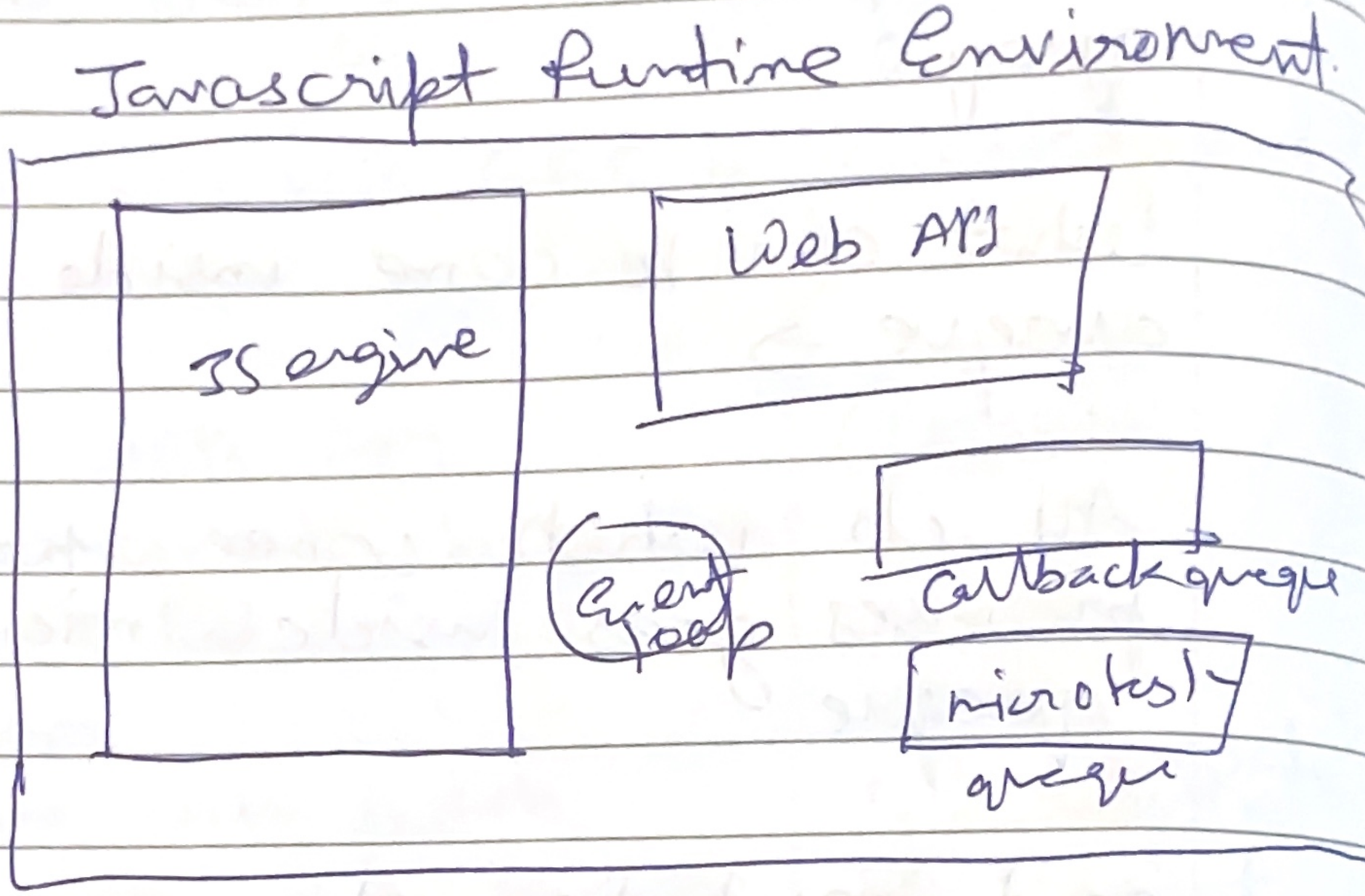
Starvation of callback queque :-

When callback queque doesn't get a chance to execute and go to callstack because of microtask queque.

PAGE NO. :
DATE : / /

JS Engine

To run code, we need js engine

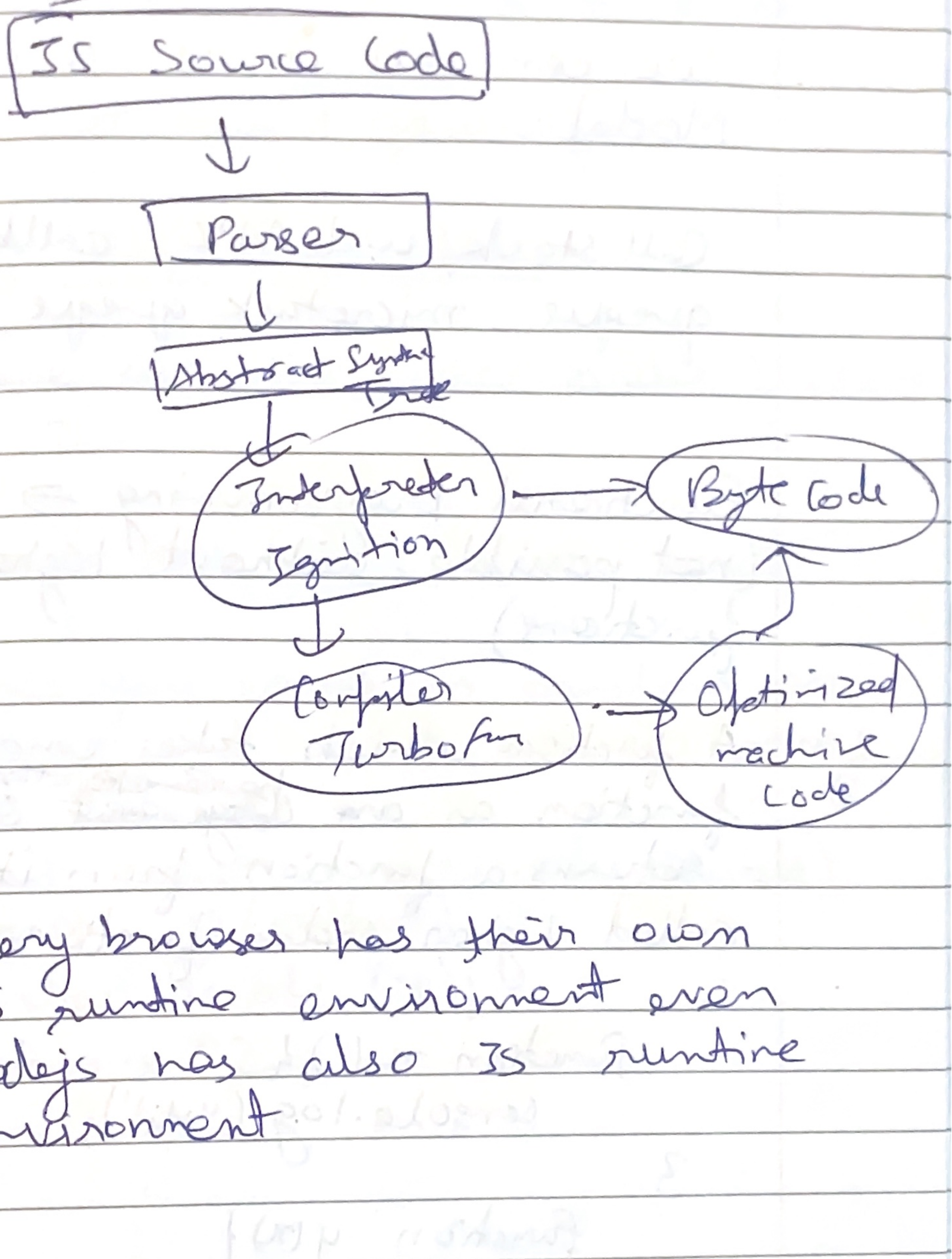


Just in time compilation.

(It uses both compiler and ~~interpreter~~ interpreter)

PAGE NO. :
DATE : / /

V8 JS engine



Every browser has their own JS runtime environment even nodejs has also JS runtime environment

arrow fun => const ... ;

PAGE NO.:
DATE: / /

SetTimeout =>

We can call it as a concurrency Model.

Call stack, web API, callback queque, microtask queque, eventLoop

Functional Programming =>
(not possible without higher order functions)

A function which takes another function as an ~~parameter~~ argument or returns a function from it is called higher order functions.

```
function x() {
    console.log("Hi");
}
function y(x) {
    x();
}
```

PAGE NO. :
DATE : / /

Eg :-

```
const radius = [3, 1, 2, 4];
```

"Do not repeat yourself"

Map, filter and reduce

These are the higher order functions in JS.

```
const arr = [5, 1, 3, 2, 6]
```

When you want to transform an array and wants a new array.

```
const output = arr.map(double);
console.log(output);
function double (x){
return x*2;
}
```

const arr = [5, 1, 3, 2, 6]

To filter out the value by putting some condition.

// filter odd

const output = arr.filter(isOdd)

function isOdd(x) {
  return x % 2 } ⇒ it will return true or false.

3

Reduce

const arr = [5, 1, 3, 2, 6]

To solve sum, max, min, we use reduce (to get the single value from this array)

PAGE NO. :
DATE : / /

Ex

```
const output = arr.reduce(function
                    (acc, curr){
    acc = acc + curr
3, 0)

return acc;
```

Issues with callback

① Callback hell ⇒

It is difficult to manage.

Inversion of control

② callbacks are dependent to each other, that's why it is very risky. We are giving control of our prog. to diff. function

To get rid of callback hell, we can use promises.

PAGE NO. :
DATE : / /

Promises

const cart = ["shoes", "pants", "kurta"];

Two apis =>
createOrder(cart); //orderId
proceedtopayment(orderId);

.then is a function available over promise object.

Promise result :- means data
Promise state :- "pending" or "fulfilled"

Promise is an object representing the eventual completion or failure of an asynchronous operation.

Normal and arrow function

Arrow function don't have their this context. They ~~bind~~ lexically bind this to the surrounding context, they inherit this from parent scope.

var name = "abhiraj"

let obj = {
    name : "rajput"
    arrowfunc : () => {
        console.log(this.name);
    },
    normalfunc : function(){
        console.log(this.name);
    }
}

obj.arrowfunc();
obj.normalfunc();

- Spread operation introduced in ES6

if we want to combine two array, it can do

const arr1 = [2,3]
const arr2 = [4,5]
const add = [...arr1, ...arr2]