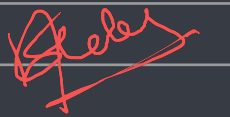
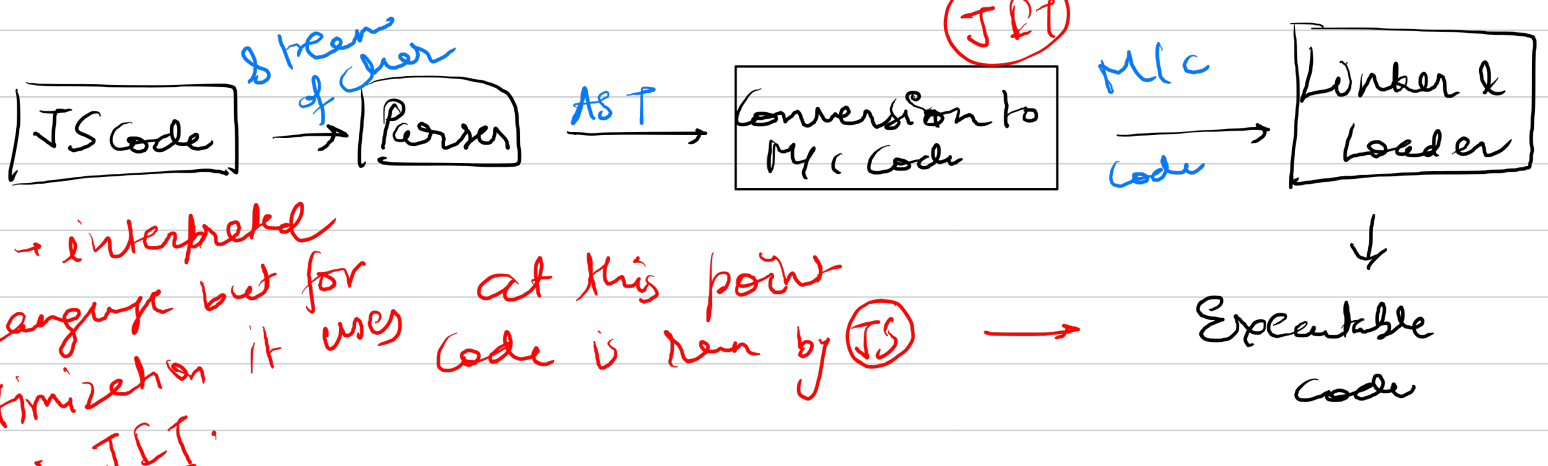
Javascript (Basics to Advanced)

Sandeep Kumar



How Javascript works :-

- Every Browser have JS Engine
  - Google chrome - V8
  - Mozilla firefox - Spider monkey
  - Internet explorer - Chakra



⭐ When code Runs (Execute)

Browser → window
Node JS → global

① It creates a global Execution context (default)

this

Call stack — associated with default window obj

- Every Execution context have two phase
- Whenever any functional call happens a new execution context will be created
- Stack (call stack) contain them execution context

EC

Memory	code

JS is a single-threaded & synchronous language

⊙ Memory creation phase :- Here for every (var) variable & function given to memory

var : undefined

fun : {--} (body of function)

(key - value pair)

⊙ Code (Run) phase → JS code executes.

⊛ Hoisted :→ Hoisted is a phenomenon in JS by which variables & functions are moved to the top of code into (JS). (in Memory Creation phase)

1. Variables - (var) - Hoisted (initially is undefined)
   (let & const) - Also hoisted but Remains in temporal deadzone → 'Can't Read before initialization' → Error

2. functions - function is hoisted with their full definition
   (simple fn)

3. function Expression - Hoisted as a variable (Initially undefined)

4. Arrow function - doesn't hoisted.

⊕ Scope chain :- It's a way to resolve the value of the variable in JS

- No local take global
- lexically defined
- Inner function can get Access to the parent function

Own Execution context + Global (parent) Execution context = Sharpness

functional scope – func () {...}

Block scope – {...}

Global scope – whole program

⑧ use strict Mode in JS :-

"use strict" :-

① No hoisting in JS
② var, let, const is neccesary
③ Identifier should not be special keyword
④ All property name in object unique
⑤ this is undefined in Regular function

⊛ Data types in JS

① primitive Data type :-
(passed as value)

- Number
- Boolean
- String
- undefined
- Symbol
- object

② Non-primitive Data type
(passed as Refrence)

- Arrays
- object
- functions

} ↑ means changing the other may effect the original one.

let a = 2
b = a ← passed as value

let a = [...]
b = a ← passed as refrence

⑨ functions is a first class citizen in JS

① ↳ Can be treated like other variables we have in JS
② Can be assigned to variable
③ Can be passed in a function
④ Can be return from a function.
⑤ We can pass the function in a property of object
⑥ JS is a functional programming language.

Higher order function → It takes function as an argument & return from an function.

Immediately Invoked function → self Executing Anonymous function

```
(()=>{...})();
```

```
(()=>{...})();
```

Grouping operator

Execution operator

due to function is a first class citizen

Still

⊛ Closures in JS :- Inner function has the access to it's outer function as well as the global variable.

(function + lexical environment)

① functions with lexical scope

```
function Exp(x){
    var a=7;
    func y(){
        Consol.log(a)
    }
    y
    return y
}
```

⇒ ⓐ var Z = Exp(2)

ⓑ Z()

Call, Apply, Bind :-

```
const obj1 = {
    name = "sandeep"
    class = 5
    printName : function () {
        console.log (this.name)
    }
}

const obj2 = {
    name = "Vishal";
    class = 8
}
```

Call → used to call the method of object which belong to another object

```
obj1.printName.call(obj2, ...)
```

Apply → Similiar to call but call method takes argument seprate it takes as an array

```
obj1.printName.apply(obj2, [...])
```

Bind :- It return a new function

```
fun = obj1.printName.bind(obj2)
fun()
```

ⓕ Rest & Spread operator :-

function define : Parameter (Params)

function call : Argument

Rest operator :- Infinite argument as an array operator.

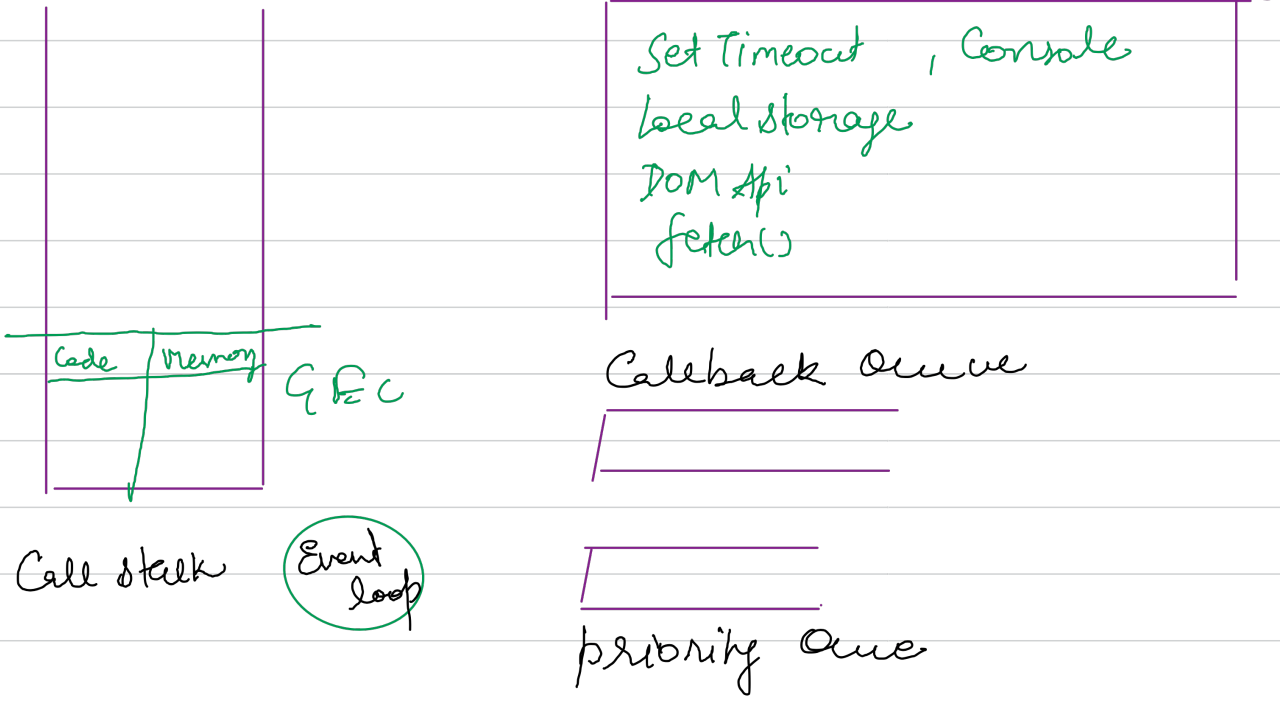
```
function Sum (...inputs) {
        input[0] + input[1]
}

Sum (1,2,3,4)
```

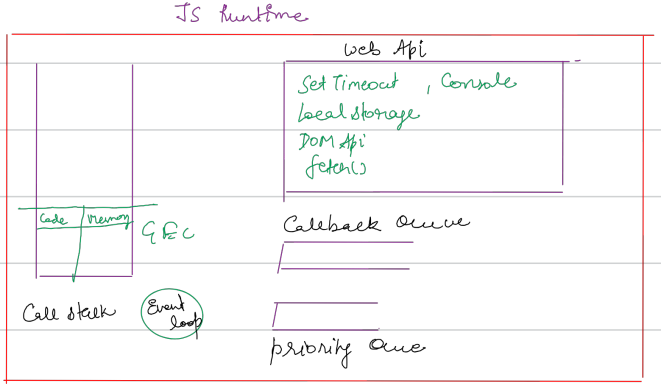
Spread operator :- to create a new array & object copy of

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

arr1 = [...arr] → arr, & arr1 have different object

JS Runtime





Event loop :→ It continuously monitors the Call stack. if Call stack is empty then it take the fn from Callback OR priority queue & pass to the JS Engine.

① Api Call (Promises, Mutation Observer) register inside the priority Queue

JS Engine (V8)

Code → Parsing → AST → Compilation → Optimized code ⇄ Execution

Garbage collector

(Heap | stack)

All String Methods

String Properties
constructor
length
prototype

String Methods
charAt()
charCodeAt()
concat()
fromCharCode()
indexOf()
lastIndexOf()
match()
replace()
search()
slice()
split()
substr()
substring()
toLowerCase()
toUpperCase()
valueOf()

Array Properties
constructor
length
prototype

Array Methods
concat()
indexOf()
join()
lastIndexOf()
pop()
push()
reverse()
shift()
slice()
sort()
splice()
toString()
unshift()
valueOf()

Number Properties
constructor
MAX_VALUE
MIN_VALUE
NEGATIVE_INFINITY
POSITIVE_INFINITY
prototype

Number Methods
toExponential(x)
toFixed(x)
toPrecision(x)
toString()
valueOf()

Boolean Properties
constructor
prototype

Boolean Methods
toString()
valueOf()

Math Properties
E
LN2
LN10
LOG2E
LOG10E
PI
SQRT1_2
SQRT2

Math Methods
abs(x)
acos(x)
asin(x)
atan(x)
atan2(y,x)
ceil(x)
cos(x)
exp(x)
floor(x)
log(x)
max(x,y,z,...,n)
min(x,y,z,...,n)
pow(x,y)
random()
round(x)
sin(x)
sqrt(x)
tan(x)

Date Properties
constructor
prototype

Date Methods
getDate()
getDay()
getFullYear()
getHours()
getMilliseconds()
getMinutes()
getMonth()
getSeconds()
getTime()
getTimezoneOffset()
getUTCDate()
getUTCDay()
getUTCFullYear()
getUTCHours()
getUTCMilliseconds()
getUTCMinutes()
getUTCMonth()
getUTCSeconds()
parse()

Global Objects
global
process
console
Class: Buffer
require()
require.resolve()
require.cache
require.extensions
__filename
__dirname
module
exports
setTimeout(cb, ms)
clearTimeout(t)
setInterval(cb, ms)
clearInterval(t)

console
console.log([data], [...])
console.info([data], [...])
console.error([data], [...])
console.warn([data], [...])
console.dir(obj)
console.time(label)
console.timeEnd(label)
console.trace(label)
console.assert(expression, [message])

⑧ for-of loop :-

array, string

①
```
for (const item of arr) {
    console.log(item)
}
```

②
```
const map = new Map()
map.set(1, a)
map.set(2, b)
```

⇒ insertion order is preserved

⇒ key can be anything unlike single object

```
for (const [key, val] of map) {
}
```

③ simple object is not iterable using for-of loop

⇓

use for-in loop

```
for (const key in obj) {
}
```

for-of loop → array, string, map

for-in loop → object

Object.keys

⊛ DOM Manipulations :-

① let button = document.getElementById(button)

document.getElementByClassName(button) → HTML Collection

① button.className

.getAttribute('id')

.setAttribute('id', 'new') — overwrite

.style = {

}

OR

.style.backgroundColor

.color

.padding

.margin

.innerHTML → With HTML

.innerText → visible text

.textContent → content (text)

② document.querySelector('#id')

└ #id, tag, .className, input[type=but

③ document.querySelectorAll('.class') → Returns | Node List |

getElementsByTagName / ClassName

↓

HTML Collection

querySelectorAll

↓

Node List → forEach

item, key, value

length

HTML Collection → No for loop

↓

Convert into Array first

Array.from(HTMLCollection)

Not an Array

⊛ Create a New Element in DOM

```
const parentNode = document.querySelector('.sam')
```

parent.children → HTML Collection gives all child Nodes)

⇒ parent.firstElementChild
- .lastElementChild
- .parentElement
- .NextElementSibling
- .childNodes

div =

⊛ document.createElement('div')

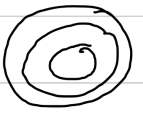
```
div.className = 'main div'
div.id = 'id'
div.setAttribute('title', 'set other')
div.innerText = 'some text'
```

document.body.appendChild(div)

Edit / Remove the child

```
document.querySelector('li:nth-child(2)')
```

Create → createElement(), appendChild()
Edit → replaceWith()
Remove → remove()

Events in JS → Event bubbling →  → child to parent
Event Capturing → parent to child

doc.getElementById('div').addEventListener('click', (e) => { }, false)

false → Event Bubbling } e.stopPropagation(), e.preventDefault()
True → Event Capturing }

default

Fetch –

fetch(URL)

(memory) ← Response

Web Browser / Environment

Promise State

Promise Result

W/w request

fetch return a body ReadableStream

⇓

await res.json()

fulfilled

reject

Async Code

⇓

Node ⇒ V8 + Libuv

Browser ⇒ V8 + webApi

⇓

Event loop & Queues

(Non blocking Code)

Callback functions — gives JS the power to run the Async code

⇒ `setTimeout( () => { }, 1000)`

`() => { }` — Callback fn

⇒ Using Callback will lead to two issues

→ Callback Hell — Promise Chaining

→ Inversion of control — promises

⭐ Promises :- to Run the Code Asynchronously

① Promise is an object which represent the eventual completion of failure of the async operation

② It is a container which will be filled later (after completion of async operation).

③ promises → resolved once & Immutable
- ① promise state → pending, fulfilled, rejected
- ② Promise Result → which will be filled later

⇒ We can control the callback function by using then & catch.

```
Promise ()                          new Promise ((resolve, reject) => {
  .then ((val) => {                   if (!validate) {
      return val                          reject ("error")
  })                                  }
  .then ((val) => {
                                      resolve (ordered)
  });                                 })
  .catch ((error) => {})
```

⭐ Promise API :- ① promise.all ② promise.race
② promise.allsettled ④ promise.any

for parallel Api calles

① Promise.all() → wait for all of them to finish. if any api gives error then it will throw error immediately.
(all or none)

② Promise.allSettled() ⇒ waits for all promise to settled if one gives error it still execute other & return a array of Result

[val1, err2, val2]

③ Promise.race( ) → gives the value of first settled promise (either success or fail)

④ promise.any( ) → gives the result of first successful promise. If all fails then return the result of aggregrate error.

⊛ Async & Await :- Syntatic sugar to handle the promises same as then & Catch by easy to write.

to create Async function async keyword used

Async fun → Always return the promise

async → await

⇒ then, Catch – JS engine will not wait for resolve promise it execute the next line of same fn

⇒ async, await – wait for fulfill the promise then only next line will run in async fn

↓

try & Catch

⭐ This Keyword :-

window → Browser
Global → Node

① In Global space this refers to the global object

② Inside the regular f^n this behaves differently
- strict mode - undefined
- non-strict mode - window

this substitution phenomenon → if the value of this is undefined or null then this will be replaced by global object

③ Value of this keyword depends on how function is called during runtime.

④ Inside DOM this refers to that element itself.

⑥ Arrow fn doesn't have this binding.

take global object

where it is present lexically (enclosing lexical context)