Java Scipt → CRUD.

Extension ⇒ .js.

DataTyps:

- ↳ String
- ↳ number
- ↳ boolean
- ↳ null ⟶ Empty Value (Eg a = null.)
- ↳ undefined → declare a variable but not assign the value to the variable. (a =)

Declaration:

↳ Var, let, const.

for print: (output).

console.log(a, "a")

Camel case:

abishekRaja.

Comment:

//.

/* */.

Var:

Outside function globally act.

inside function only act.

declare same name 2 times reassign the value

Re declare

Re-assign.

global scoped or function scope

Eg var.

var a = 10.
console.log(a)

let:

Const:

Esb => To know

Esy: Breakpoints

let
block scoped.
outside block it can't be accessed.

Can't redeclare
can declare same name in diff blocks.

equals

*) = → assi
*) == → equal or not
*) === → check also its datatype

Conditional Statements

① if
else
else if

② switch.

Const
block scoped.
can't update

Object: Eg: let obj = { . . . }

String methods.

Functions: ( ): (overcurdle).

let x = function ( ) { (function expression)

let fn = ( ) => { } => (Arrow function)

Date

new Date(). → Today's data.

---

for: (Looping)

foreach
map
filter
reduce
} → Same Syntax.

find
find Indexing.

for in – sconst

String Methods.
Array Methods.

Eg

```
.foreach(function (a){
   }
```

for in

```
⇒ for (let x in ...){ }
```

for of

```
⇒ for (let x of ...){ }.
```

[1, [:],], [2 :]

R co

~~Git~~

Variables:

Var : we can declare both inside and outside function (ie global & local).

Pro

```
Var a = 10;
var a = 25;
function abi()
{
 Var a = 20;
  Console.log(a)    //20
}abi()
Console.log(a)     //10.
```

let / const :

Pro

```
let a = 10;
function abi()
{
let a = 20;
console.log(a) //20
}
abi()
Console.log(a) //10
```

For in, For each, For of, For:

for in → for object

Others → array.

map → Returns a new array.

filter → true or false.

Reduce → a.reduce ((x,y) =>)

first value, second value.

find/indexof ⇒ Using unique id. (Returns Index)

find ⇒ Returns Object: Only one object

a = ["Abi", "Ammu", "papa"]

for(x in a) one a[x]

Eg. (Map) [{}, {}, {}]

arrgobj = {[ ]}

let n = arrgobj.map ((obj) => {
    let obj1 = {
        name : obj.name
    }
    return obj1
})

[01]

arrgob = {[ ]}

let n = arrgobj.map (obj =>
    obj.name
).

let a = ["Abishek", "Raja", "good", "boy"]
let b = ""
let x;
for (x in a):
    b += a

b b
let - let + a
b = b + a

: Abishek - Raj - good boy

Eg → filter.

```
let n = arrofobj.filter((obj) => {
    if (obj.age == 22){
        return obj;
    }
});
```

(or)

```
let n = arrofobj.filter(obj =>
    obj.age == 22
);
```

Eg: find.

```
let a = arrofobj.find((obj) => {
    if (obj.rollNumber == "001"){
        return obj;
    }
});
```

Reduce :

```
const n = [1,2,3,4];
const sum = no reduce( (result, item) => {
    return result + item;
}, 0);
console.log(sum, "sum");
```

```
if (result.age) {
    return result.age + item.age;
} else {
    return result + item.age
}
});
```

(or)

```
const n2 = [1,2,3,4];
con  s = ar. redu (creul, item) => {
    return result + item.age;
}, 0);
```

Eg: Add Index.

```
function f(a,b)
{
  let d = a+b
  console.log(d)
}
f(4,5);
```

```
fun f(a,b)
{
  return a+b
}
let c = f(4,5)
console.log(c)
```

```
a = ["abi", "Shiva", "Raja"];
for (let i=0; i<a.length; i++){
  console.log(a[i])
}
```

for, for in, for of, for each

for :

```
a = ["Meena", "Abi", "Raja"]

for (let i=0; i<a.length; i++)
{
  console.log(a[i])
}
```

for in :

```
a = ["Meena", "Abi", "Raja"]
let x;
for (x in a){
Console.log(a[x])
}
```

for of :

```
a = ["Meena", "Mani", "Sullo"]
for (x of a)
{
console.log (x)
}
```

↳ object

for Each

```
a = ["Meena", "Abi", "Yamu", "Rex"]
a.forEach (f1)
function f1 (value) {
  console.log (value)
}.
```

DOM. → (Combine Html, CSS, JS).

Document Object Model.

↳ Can't view.

↳ Tree Structure.

Eg: Html

```
<html> <body>
<p id = "demo"> Abi
</p>.
```

dom.js.

```
let a = document.
    getElementById("demo").
console.log("a", a)
```

getElementById

.Innerhtml.

textcontent

getElementByClassName

:getElementByTagName.

.attribute

.style.property

.Innertext

~~Create Element~~.

:querySelectorAll ( ).

~~Innertext~~.

Syntax

document . getElemById

(" "). innerhtml

getElemby ClassName.

d[0]. innertext ( )

CreateElement CreateTextNode removechild.	innerhtml Tags with content	textcontent Every text inside Html.	~~I~~nner text Shows only display contents.

let c = 'Hello'
let e = document. Create Element ('p')
c . appendChild (e)

const newContent = document . createTextNo
Console . log (new . content.) (' ')
e. appendchild (new content)

API - Application Programming Interface
↳ Mediater between front & Back
↳ PUT, GET, POST, DELETE.

---

fetch

fetch (url)

Fetch.

```
fetch(" api/.. ").then(res =>
res.json()).then(resp =>
resp.forEach(res => {
});
```

POST - Update
GET - Read
POST - Create
DELETE - delete

function

async → async function
await → used to wait.

async fun() ⇒ Executed one by one.

if async then await

API.

Used for getting data from db.

f.e Servlet → Database
R.E.

Events

1) onblur
2) onchange
3) ondoubleclick
4) ondrag
5) ondragend
6) ondragenter
7) onfocus
8) ondragleave
9) onkey(down, press, up)
10) on mouse (down, enter, leave, over, move, out, wheel)
11) onclick.
12)

Tokens

Handling login details.
if one we login the token will generate after that they access that page.

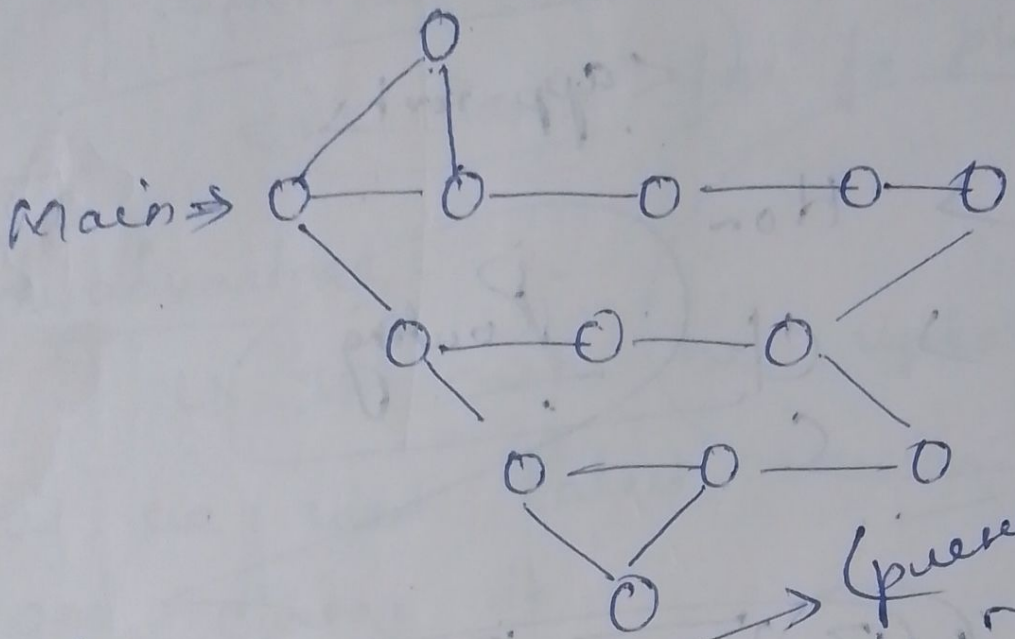
Source Code Management ← GIT → Devops

↳ Open Source ← It's enable multiple developers to work together.

Ans: Git download for windows.

↳ Download for windows.

↳ Github.com.



(present in code)

cmd

① git clone (URL) (Project URL)

② git checkout (-b) branchname (it can be anything)

(denotes new branch)

③ git push -u → (to Cloud → push)

(or)

git push --set-upstream origin test (branch name)

Function :

① function declaration (or) named function

```
function fun(parameter){
}
```

② function expression (or) anonymous function

```
const var = function(parameter){
}
```

③ arrow function

```
const var = (parameter) => {
}
```

---

① function declaration

```
function abi(a,b){
    return a+b
}
console.log(abi(2,4))
```

④ fun. expre.

```
const var = function(a,b){
} return a+b

co.lo(var(2,5))
```

⑤ arrow function

```
const var = (a,b) => {
    return a+b
}
co.lo(var(2,5))
```

(or)

⑥

```
const var = (name) => {
    return +name[5]
}
console.log( var("1256789010") + 1000 ).
```

+ name
↳ used for type change as number also used (type of)

Single line (No return)

```
const var = (name) => Number(name[5])
```

Immediate invoke function (No need to call)

```
( function fn () {
    console.log ("name"[2])
} ) ().
```

1) default parameter

```
const var = (a = "123456") => {
    return +a[2]
}
console.log(var())
```

2) Rest Parameter (Spread Operator (It is array)

```
const var = (... args) => {
    conside.log (args)
    return args.
}
conlo(var ("abi", "meena", "mani"))
```

Shallow Copy

```
var friend = {
    name : "Abi",
    age : 22,
}
```

} object

```
var fav = friend
fav.name : "Meena"
console.log ("friend", friend)
console.log ("fav", fav)
```

out

friend: Meena 22
fav. Meena 22

} both can change

(ie) shallow copy

deep copy - method (1) -> Spread operator

① var ~~person1~~ fav = {... ~~person~~ friend}

fav["name"] = "Meena"

console.log (friend)

console.log (fav)

var fav1 = {... fav, "now": "now"}

② method (~~1~~) ~~spread~~ Spread Operator (Error)

var person = {

name = "Abi",

age = 22,

gender = "Male",

address = {

street = "street",

Dist = "dist",

state = "state"}

}

9. var person1 = {... person}

person1.address.doorNo = "30"

Console.log (person1)

Console.log (person)

(or)

var person1 = {... person, address: {... person.address}}

person1.address.doorNo = "30"

Console.log (person1)

Console.log (person)

Task:

using Spread Operator with 2nd & 3rd level data change pincode.

Method (2) - deep copy

① JSON.

```
var person = {
    name : "Abi",
    age : 22,
    address : { street : "str",
                dist : "dist" }
}
```

in stringify key also in "quotes"

var p1 = JSON.stringify (Abi) →

var person1 = JSON.stringify (person)

var person2 = JSON.parse (JSON.stringify (person)) p1

Object destructure

d - structure

```
var person = { name : 'Abi',
               Address : { str :
                           dr : } }
```

{N, ...others} Ab

Abi = {N, Age, Make : {

```
Const {Address} = person
Console.log (Address)
```

const {Address, ... others} = person
(ob).

Console.log ("others", others)

Address
others

~~Array~~ Object destructure

Var person = { name = "Abi",

score = [1, 2, 3, 4, 5] }

Const {score} = person

Console.log (score)

Array destructure kind of Assignment [

const [m1, m2, ... remaining] = score
ab (ob)

console.log (m1)

console.log (m2)

console.log (remaining)

~~console.log (key~~

```
console.log (object.keys (person))
console.log (object.values (person))
```

Eg (String to array)

```
const name = "Abishek Raja".
console.log (name.split(" ").reverse().
                    join(" "))
```

```
const ff = () => {
const friends = ['Abi', 'Meena', 'Keerthi']
Const bf = ['Abi']
let f = []
for (const x of friends) {
    if (bf.includes (x)) {
        continue
    }
    else { f.push(x)
    } }
return f } (console.log(ff())
```

Array Methods

1. includes
2. some
3. every

```
const n = [1, 2, 3, 4, 5]
```

n[2] ⇒

① includes

```
console.log (n.includes (5))   // true
```

↳ that no.

```
Cosole.log (n.includes (3, 3))   // false
```

no position.

② Some

```
Console.log (n. some ((num, idx, a) => num < 4))
```

③ every :

```
console.log (n. every ((num, idx, a) => num < 4))
```

find & findIndex

```
console.log(n.find((num,idx,a)=>
                    num=10))

console.log(n.findIndex((num,idx,a)=>
                    num=10))
```

String Methods

1) includes
2) Starts with
3) ends with
4) Concat
5) Index of
6) last Index of
7) Length
8) replace
9) search
10) split
11) slice
12) trim
13) to upper case
14) to lower case.
15) substring

```
y
[20,0,6]
y.push(1)
y.push(2)

y.join(',')

y.join('')
```

① let str = "Abi is good boy"

② let s1 = str.slice (8, 12) or slice(8)

③ let s2 = str.substring (8, 12)

④ let s3 = str.substr (8)

Replace

let s = "Abi is good boy"

let x = s.replace ("Abi", "Abishek")

to upper case & lowercase

let s = "Abi is good boy"

let a = s.toUpperCase();

let b = s.toLowerCase();

trim

let s = " Abi"

let s1 = s.trim();

console.log (s.length)

console.log (s1.length)

Split

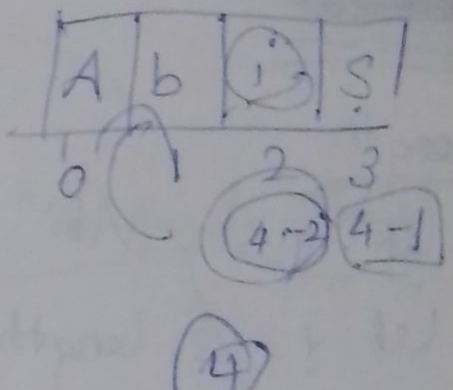
```
let s = "Abi, Meena, Mani"
let y = s.split(",")
```

search

```
let s = "Abi, Meena are friends."
let y = s.search("Abi")
```

includes

```
let s = "Abi is good boy"
s.includes("Abi")
y.includes("Abi", 8)
```



startsWith() (case sensitive)

```
let y = "Abi is good boy"
y.startsWith("Abi")
y.startsWith("good", 8)
```

EndsWith()

```
let y = "Abi is good boy"
y.endsWith("boy")
y.endsWith("good", 11)
```

IndexOf :

let s = "Abi is good boy"

s.indexOf ('good')

lastIndexOf()

let s = "He is good good"

s.lastIndexOf ("good")

~~Concat~~

length

let s = "Abi is good boy"

let y = s.length

concat

let x = "Abi";

let y = "shek";

let z = x.concat (y)

Promise

@ fulfilled / Pending / Rejected → 3 states

Promise Syntax :

```
const var name = new Promise ((resolve (var),
reject (var)) => { let x = 5;
                   if (x >= 4) {
                     resolve (x + " is greater than 4")
                   }
                   else
                   {
                     rejected (" Error ")
                   } } )

Var name
Promise . then ((res (var)) => {
  console . log (*res)
}) catch . ((reason) => {
  console . log (reason)
})
```

SetTimeOut

```
function Abi() {
    Console.log("Meena")
    setTimeout(() => {
        console.log("Abi")}, 3000)
    Console.log("Hello")
}
Abi()
```