React → used to create single page web.

HTML 5

⇒ It provides enhanced structure.

① Structural / Sectional tags

⇒ <article> : self contained content
<aside> :
<footer> : footer for section
<header> : header for section.
<main> : main content of doc
<nav> : navigation link
<section> :

② Media tags:

<audio> : embed audio
<video> : embed video
<track> : for text track (caption, subtitle)
<source> : Specify multimedia resource for <audio> or <video>

③ Interactive elements :

<details>
<summary>
<dialog>
<menu>

④ Graphics and multimedia

<canvas>
<svg>

⑤ Form-Related tags:

<datalist>
<Keygen>
<output>

html5, css

↓

Layout : display property => (flex, wrap, grid)

tailwind css :

Js → can be used as at client side & service side

→ powerful ~~powe~~ prg lang equivalent to C, C++, Py.

ReactJs → Libraries / Framework of JS.

Angular JS ↗

Dynamic → interactive / responsive.

Alter:

```
<html>
<body>
<h1> AIML Rockers </h1>
</body>
<script>
alert (" This is alert ")
</script>
</html>
```

let const.
var

```
<script>
var num = 100
console.log(num)
alert(num)

let b = 200
const c = 100
```

---

let, var ⇒ values can get effected.

const ⇒ Shows error / It don't change value.

Var

→ global scope
→ we can change values.
→ It has initial value as undefined. ie, used before declaration.

```
var a = 5 // → 5
console.log(a)
var a = 10
console.log(a) // → 10
```

Let

→ block scope. "{}" → only within it.
→ values can be changed.
→ can't use before declaration.

const:

→ block
→ can't change value.
→ can't use before declaration.

Loops

→ for
→ while
→ do while
→ switch case

If, If else

case i,

1. No.of ~~lin~~ lemons in hand = 7

~~Case ii~~ Shortage = 14

case ii,

No. of lemons in hand = 21

god 1 = 7 lemons

god 2 = 7 "

god 3 = 7 "

~~Shortage = 0~~ sufficient.

case 3

No. of lemons in hand = 15

god 1 = 7

god 2 = 7

god 3 = having 1 need 6

Shortage = 6

case 4

~~No of~~ lemons in hand = 67

god 1 = 7

god 2 = 7

god 3 = 7

Surplus = 46

O/P God 1 = 7 offer

God 2 = need 7

God 3 = need 7

---

print : No. of le

7.

God 1 =

---

prompt ("No. of lemon")

i/p = 7 (a = x - 7)

if (x > 7)

alert ("God 1 has 7")

else alert God 1 need

if ~~(mod % a / 7~~

a > 7)

alert (God 2 has 7)

else

alert (God 2 has "+a)

② Sam is having 75 candies. He gives half of it to his friend Angle. Since angle loves Sam a lot she gives back half of its position. Calculate & display individualy how many candies sam & Angle has.

Constraints:

-> use only 1 variable

-> use function.

a = 75 ⇒ 37.5

b = 75 / 2 ⇒ 37.5 Angle.

c = b / 2 ⇒ 18.7

sam → 37.5

(75) / 2 → (37.5)

18.7
37.5
11
37.5
18.7
56.2

ES6

-> ECMAScript 6.

-> has Arrow function.

07-01-2025

ES6

→ ECMAScript 6

→ has Array functions.

→ for efficiency in terms of space and ↑ readability

→ can create functions without name which is called as arrow function.

↓

Shorthand way to define functions in JS.

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

sum → variable

document.getElementById → method/functions

JS object

arrow_function

see in lap JS object → Json

JSON → JS object

```
<html>
<body>
<h1> id = "response" </h1>
<h1> id = "res" </h1>
</body>
```

id name

property

```
document.getElementById("response").innerHTML = howareyou();
```

```
<Scripts>
const howareyou = () => {
    return 100;
}
```

① simple calci by 2 i/p (numbers) display add, sub, product, quotient, remainder by creating individual arrow function the same.

$$sum = (a, b) \Rightarrow a + b$$

a - ①
b - ②

② create an array by taking array size & element from user. Extract all the perfect nos and even prime nos from array.

---

push

→ unshift ⇒ add element to 1st position

→ shift ⇒ remove element in 1st ~~element~~ position.

splice ⇒ arr.splice (1, 0, 99)

1 → index

0 → no. of elements it should delete after inserting that element.

99 → element that inserted in this index

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

⇒ [1, 99, 2, 3, 4, 5]

arr.splice (2, 1, 99)

arr = [1, 2, 99, 4, 5]

slice ⇒ print only the element in gn range

arr = [1, 2, 3, 4, 5, 6]
(0, 1, 2, 3, 4, 5)

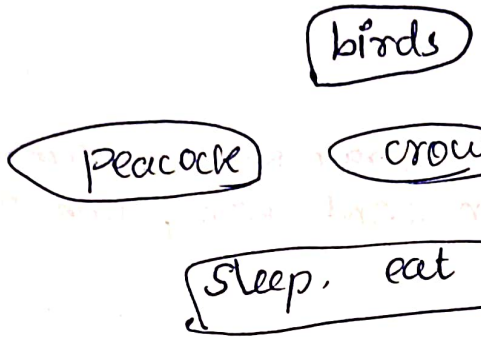
a = arr.slice (1, 4)

⇒ [2, 3, 4]

7-01-2025.

OOPs

→ Object Oriented Programming structure.



Class → blueprint (having everything).

JS Promise :

→ JavaScript obj.

→ It has 2 States.

Resolved [success]

Rejected [Failure]

Success → then

failure → catch

Callbacks ⇒ Resolve / Reject

Invoking a fun

promise → for APIs only.

8-01-2025.

① wrt a promise when Andra having bday party(bp).

A - 5 ~~Km~~ 5000 B- 3 ~~Km~~ 8000 C - 10 Km. c ⇒ 8000.

Promise Inbuild methods:

→ When their is more than one promise in order to review ~~them~~ we can we promise build in methods,

i) Promise.all()
ii) Promise.~~many~~any()
iii) Promise.~~se~~allsettle()
iv promise.race()

Promise.any ⇒ one with less time will be returned. ⇒ all r true

if any one is false ⇒ will display the one with less time.

Once it sees the promise false. it will stop. & gives shortest time promise. Status should be true.

promiseallSettled() ⇒ will display keys and value.

will display one among 3 three states.

(i) fullfilled
(ii) rejected
(iii) pending.

any ⇒ works for only true. [B→ false ⇒ then return least time of (A,C)].

race ⇒ false means then return its value thats it.

Closure:

React:

⇒ Lib / Framework of JS \ HTML ⇒ Youtube & Wikipedia
→ Eg: Amazon, Netflix

2 important folders:

~~node_modules~~
→ public
→ src.

3 Important file:

→ index.html
→ index.js
→ App.css

Note: As of now don't touch index files
Note: Initially do / wrt our code in App.js.

DOM:

Virtual-DOM / V-DOM ⇒ React.

do only necessary things.

→ React follows V-DOM; once here unlike HTML once DOM gets created the changes / manipulation what we do gets completed and only that part will be re-rended.
→ Where as in html, everytime we make change entire DOM will be re-rended.
→ In web application created by reactjs, each and every thing is called as components.

types of component
i) functional component
ii) class component.

JSX
→ writing html inside JS

```
function App() {
  return <h1>Hello World</h1>
}
```

01-01-2025

Props and States:

Stacks.

Props → Every components we have

Props
→ It won't change
Eg: Name : TATA. (Brand).

States:
→ It changes or we can change it.
Eg: waterbottle level (initially state
full → half ← updated state then it's empty).
current state

Flipkart website:

Home page

Grocery Mobiles Fashion ⇒ components.

Mobiles:
component name : Mobiles
Props name : version. price.
State : discount / offers, Stock available.

Passing props between components :

2 - files → parent.js
→ child.js

React Hooks

→ Earlier in IT industries they were using cls components but recently state concept was not available with functional components.

→ ~~New~~ Hooks is used to implement state in functional components.

types :

* useState
* useRef
* useEffect
* useReducer

Example for usestate is counter clock.

→ Stating the initial state as zero we can increment it, decrement, using hooks.

10-1-2025

Spread operator

→ passing array into useState

const combinearray = [...array1, ...array2];

Spread operator

useEffect Hook :

Upon the condition or action we apply in our functional components monitoring the impact or sideeffects can be done using useEffect hook.

→ It takes two arguments in which 2nd is optional.

1st → call back fun
2nd → dependency array

call back function is like constructure in java.

25-01-2025

MongoDB:

and db.products.find({$and:[{price:{$gt:80000}},{brand:'Apple'}]})

or db.products.find({$or:[{price:{$lt:10000}},{brand:'Apple'}]})

db.customers.find({$and:[{hobbies:{$exists:false}},{age:{$gt:40}}]})

Aggregation ⇒ group by

db.companies.aggregate([{$lookup:{from:'employees', localField:'_id', foreignField:'company_id', as:'Employees'}}])

→ local field ⇒ employees, foreign field ⇒ companies
new matched document will be added as an array in companies and the array name is employee.

Create a db 'Bank', create 2 collection under that (customer_personal and customer_account

Data Model for 1st

name → str, address → array, phone_no → obj (2 phone_no)
age → number.

5 records

Data Model & schema for 2nd

acc_no → int branch → str brand_name → str,
IFSC_code → str current_balance ~~→ int/float~~
acc_type → str overDraft ⟷ Yes | No

Query

① filter only Od categorities (Yes)

② Display only cus's address where branch names are same.

③ curren-bal 10,000 - 20,000 ⇒ only phone-no

④ only savings account (filter). full details.

⑤. Add field called 'Status' = should be same for who's IFSC code is same.

BackEnd

Backend is also called as middleware.

Custom modules ⇒ math modules

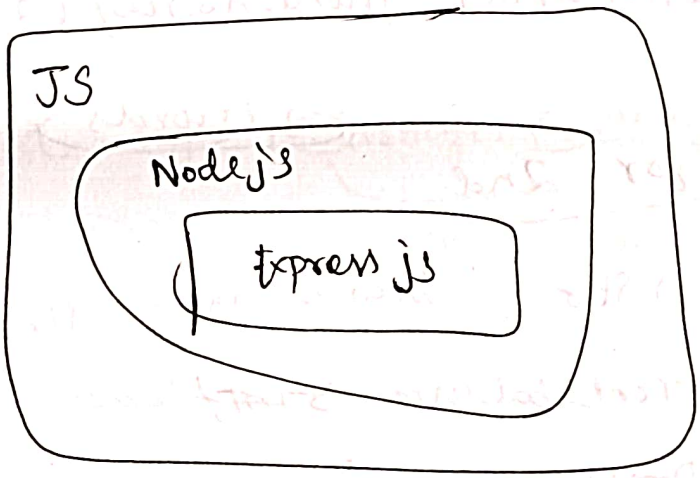
Pandas, Numpy ⇒ 3rd party module.

Modules only we will export

Node-js

→ packing library from js

→ In node we can use express js as middle ware.



require → Keyword in Node.js

```
const http = require ("http");      // Build-in module
const express = require ('express');   // 3rd party module
const sayHello = require ('./greet');  // custom module
```

27-01-2025

client → request, server → response.

[Always start ur server 1st and then go to client]

client → front end (React)

Nodemon :

→ runtime environment that allows you to run JavaScript on server-side.

Note :

→ maintain split terminal (two) in order to use client & server.

Run cmd :

→ start the server 1st, node server.js

→ command to run client ⇒ npm start

npm i express

get → send request post → get request

listen → keyword to active port.

We can delete - package.json to get it back

npm init -y

RunScript missing Error ← For this

→ Run the server.

Axios :

It's a popular js library used to make http requires from the browser / client

→ Axioms is known for easy & clean syntax and also flexibility. Especially works well with API and Rest API.

→ When we cort API for exclusive purpose then its called Rest APIs.

Cors :

→ Coss-origin Resorce Sharing

→ When a webpage requires information from a resource (from any other sites) whether to access the request and process it or not will be defined in the ~~root~~ style book. for this purpose we use CORS.

```
npm i axios
npm i cors
```

In the given example, we are requesting data from server "Hello, this is a msg from server".

- In DataComponent.js as client using 'http' get method via API /data.
Server response as json, from the json file. I want to filter only msg, so we are using cors response.data.message

Mongo CRUD:

28-01-2025

Inside src :

⇒ User.js ⇒ Create_User.js ⇒ Update_User.js

Model :

Name, age, email

Add User

Router, Axios, Express, Express cors ⇒ Dependencies.

App.js ⇒ Add routing for 3 elements.

create a folder 'server.js' ⇒ npm init

In server folder, wrt backend code in index.js

Create another folder named "models" inside 'server' and then inside models ⇒ create Users.js

server
↳ Models
↳ Users.js

Create a DataBase "Emp1"

Employee_id, name, salary, contact [array (2 nos)]

create Emp1 frontend component, it should be stored in DB.