W.P.T

React -

React dom

- structure how elements are placed
  - container
    - table container
      - Rows
        - columns
  - Button Group

Re javascript dom

document.link =
- all =
- element

React Virtual dom

- container
  - table container
    - Rows
      - column
  - Button Group

⇒ whenever something is changed than React takes snapshot of virtual DOM and compares updated Vs snapshot and than update the UI

⊛ keys

```
<ul>                          <ul>
  <li> Bruce <li>  ..........    <li> Danial <li>
  <li> Clark <li>  ----------    <li> Bruce <li>
<ul>                            <li> clark <li>
                              <ul>
```

they are same but as React will not detect they are changed as all dont have keys

⊛ If keys are assigned then it will render only "Danial" else without key complete list is rerender

⊛ JSX

→ full form Javascript Xml it allows us to write "HTML" in "JS"

Without JSX →
```
const text = React.CreateElement('p', {}, 'this is text');
Const Container = React.createElement('div', '{}', text);
ReactDOM.render(container, rootElement);
```

With JSX ⇒
```
const container = () => {
    Return ( <div>
        <p> this is text <p>
    </div>);
```

18

Functional Component

Declaration

- Before hooks function were stateless, and were behind class Component
- they are widly used than class

class Component

- class were with state hence were used with component lifecycle

---

Declaration

```
const addPatient = (p)=>{
    return (
        <div>{p.name}</div>
    );
}
```

```
class addPatient extends React.Component{
    constructor (prop) {
        super(p);
    }
    render(){
        return (<div>{p.na
            this.p.name</div>
        );
    }
}
```

```
<addPatient name="deepak"/>
```

---

state handling

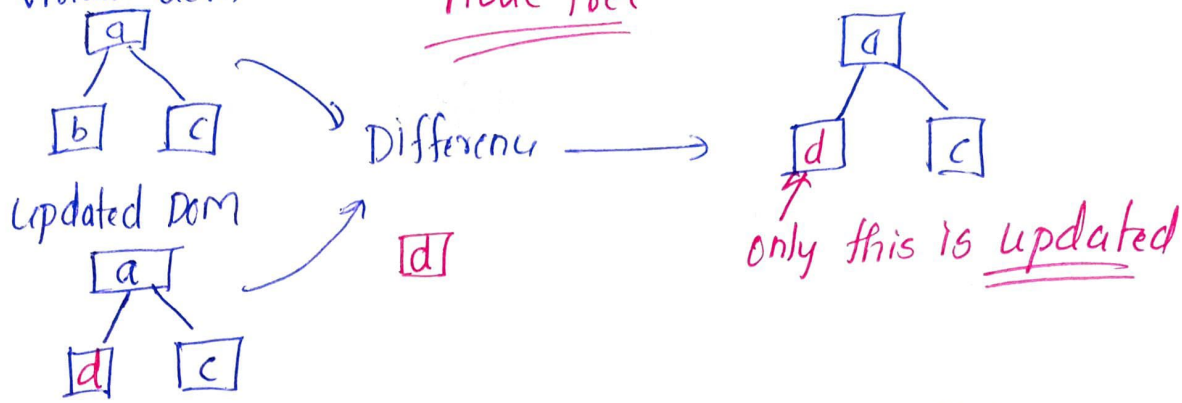
```
const ClassRoom = () =>{
  const [student, setStudent] = useState(0)
  const Cont = ()=>{
      setStudent(student++);
  }
  onclick={Cont}
}
```

Return 2 items
① Current state
② function to update state.

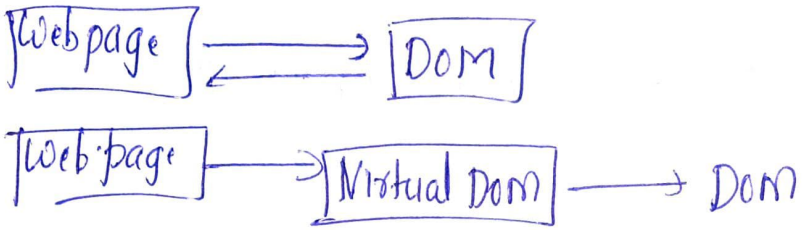
```
constructor(p):
    this.state = {student : 0}
    this.addStudent = this.addStudent
                      .bind(this)
}
addStudent=()=>{this.addStudent.bind
                (this)
}
addCount (){
    this.setState((prevState)
    => { return { StudentCount
        : prevState.State
    }
```

React →

* Virtual Dom? → Javascript Object



* was introduce to remove the complete updatn of "dom" and improve the speed of Render & efficiency



Props: (read only.
It is used to send data from parent component to child Component

```
<Patient patient={Patient}/>.        const patient(prop)=>{
                                       Return (<div>{Prop.patient.name}
                                       </div>)
```

React Hooks

* are built in functions which enable developer to use state, and life cycle method of Component (16.8 onwards)

* life cycle methods → mount, unmount, update, } from functional Components.
  → property (prop), state.

* they cannot be used in class components

* called at top, and from Functn component

UseEffect :- with these hook we inform react that you of Compon. will run after 1st render & after update Component require to do something after rendering or on change. state, e.g is used fetching, setting up

① use Effect → runs 1st at after first render
② after dependancy changes state

Component life cycle

① Initialisation ② mounting (componentwill/did mount)
③ updating (component will/did Update)
④ unmounting (componentWillUnmount)

React-Router

→ standard library used for routing in react, used for single Page App
& without refreshing the page

→

BrowserRouter → parent element uses HTML5 history API
↳ Routes
↳ Route
↳ Link.

Conditional Rendering

① Return (
<h1> Hello </h1>
{ cars.length > 0 && <h2> you are fit </h2> }

② { Condition ? true : false } =>
return (
{ isGoal ? <madeGoal> : <missedGoal> }

State :- Internal storage of component

Advantages of React (its js library) (fro view → mvc

① Virtual Dom => speed ↑ ② Reusable Components.
③ server side rendering ④ usage of "JSX" Code readability ↑
⑤ ↑ app performance. ⑥ Open source - front-end technology

React → (single page applicatn) speciality

limitation of React

① It is just a library not framework
② library is large & takes time
③ uses inline templating "JSX"

ES6 Vs ES5

① require Vs import

ES5 → var react = require('react')

ES6 → import react from 'react

② Export Vs Exports

ES5 module.exports = component;

ES6 export default Component;

React Vs Angular

	React	Angular
① architecture	'View' of MVC	MVC
② Rendering	server side rendering	client side rendering
③ Dom	uses Virtual Dom	uses Real Dom
④ Author	Facebook	google
⑤ Data binding	one way data binding	two way databinding
⑥ Debugg	Compile time	Runtime
⑦ Reusable Components		

mutable
State :- is object of react component that holds some information
Immutable that may change over the life time of Component
→ as it exist in complete life cycle hence initialisatn is required
Prop :- is read only object that hold some informatn to control the behaviour of the particular Component

⑧ when state changes Component Re-renders

26 "React-router-dom":-

→ Router V6

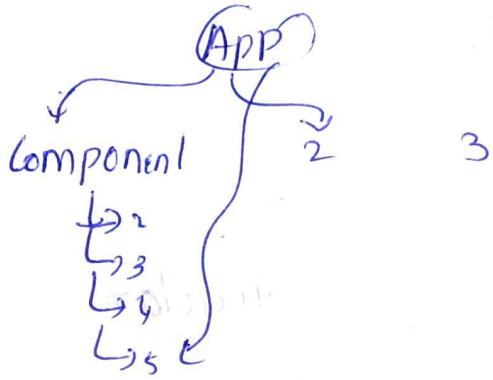
```
<BrowserRouter>
  <Routes>
    <Route path="*" element={<NoPage />} />
  <Routes>
<BrocoserRouter>
```

⊗ used for single page application, without Refreshing. whole page

Redux

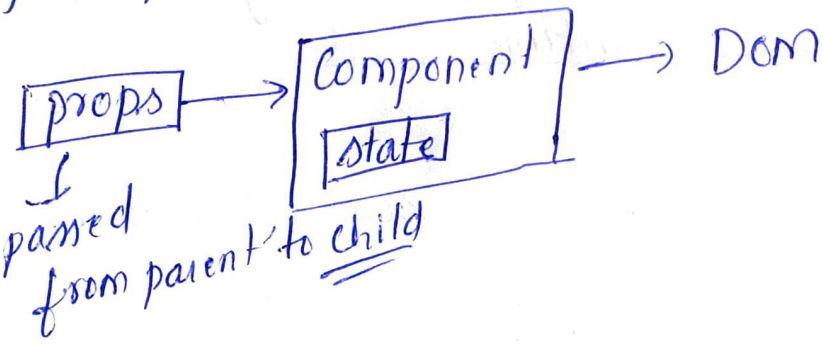
⊗ state management

⊗ to transfer to Data b/w component.



Pure Component

React. Pure Component it is same as react. Component except pure Component handles should Component Update (), if there are changes than only component is rendered or screen, in case of compones every change.

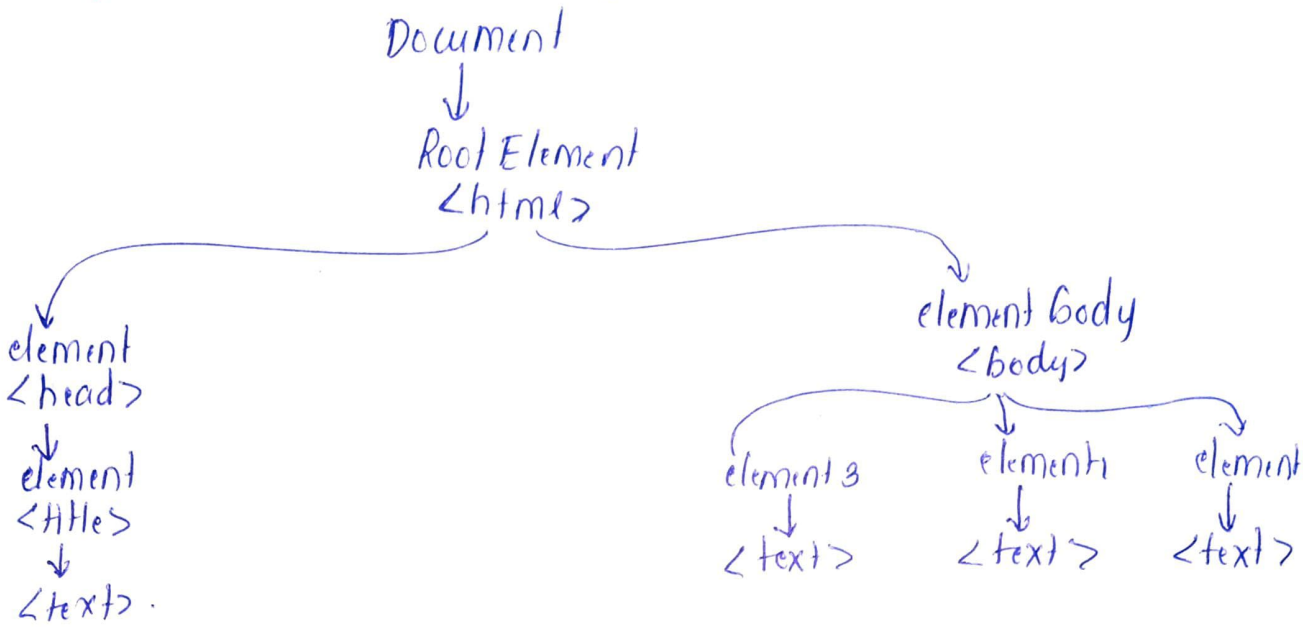


React :-

feature of react

① reusable component ② virtual dom ③ one way data binding - parent component nests child components, ④ JSX ⑤ High performance ⑥ small size docker image ⑦

Dom **virtual Dom** :- stored in **memory**



~~JSX~~ **event** <Button onPress={lightUp}/> → in html only string allowed
↳ name of function

Comment singleline → #
multiline → {/* */}

React

- mainly used to create single page application

```
ReactDOM.render(myelement,
    document.getElement*("root"));
```

- **Functional Component**
- they are stateless
- Where as class based Component are statefull, smart component

```
class myElement extends Component{
    render(){
        return <div>ab</div>.
    }
```

To show Dynamic content

```
<h1>{deepak.surname}</h1>
```

Comment

```
{ /*    */}
```

Props

- read only
- passed through html attribute
- any data allowed

```
<Patient patientDetails="{Details"/>

<H3>Hello {props.patientDetails}</>
```

events

```
onClick={updateServer}
```

↑ no {}

React State

- internal object to store information about component.
- whenever state change component rerender

React Hooks

- allows to hook into state & life cycle methods

```
import {useState} from "react"
```

→ allowed only in Function Component
→ called at top level
→ useState()

- track state in functional component

Initial state → useState() → current state
→ Function to set state

- whenever state changes component rerenders.

hook

props	useState
① changes lead to re-render() to update DOM	① changes lead to re-render() of component to update DOM
② parent → child	component state change within
③ not modifiable	modifiable
read only	read/write

```
(event) => {
    console.log(event.target.value)
}
```

Keys :- It helps react while updating it will match the key and update only required or "else" complete list get re rendered

Conditional Rendering

```
{ condition ? True : elseState }

{ flag && statements }
```

class Based component

```
class HelloComponent extends Component{
  render(){
      return (<h3>Hello</h3>)
  }
}
export default Hello
```

passing prop

```
<div><Hello name="dkrush" /></div>
```

```
class Hello extends C.. {
    return ( <h1> Hello {this.prop.name}
}
```

State :-

```
class Statefull extends C{
state = {
    name : "shrilata"
    email : "@gmail.com"
    address: "pune"
}
render(){
  return (
    <div>
      name: {this.state.name}
            {this.state.email}
            {      address}
    </div>)
```

Life Cycle Events

1. Initial
2. mounting
3. Updating
4. Unmounting

```
class HelloComponent extends C {
  constructor(){
    super
    this.state = { mess: "hello"}
  }
  render(){
    return(<h1> <>)
  }
```

Initialisatn phase

Constructor(props) → when new Component created

ComponentWillMount :- immediate before mounting Component

render() :- it returns the mkp

Component Did mount → immediate after mounting Component. on Dom

↳ here all ajax calls are done

axios

```
class Hell extends Compon {
    Component Did Mount(){
        Console ( mounted)
    }
    render(){
        <h1>Hello</h1>
    }
}
```

Update Methods

→ Change to state.

Component will recieve props

↓

Should Component Update() → return Boolean

↓ True ↓ false

Component will updat.()

↓

render

↓

Component did Update()

Component will Unmount()

Error Boundries

```
Const person() {                    → normal error
    If (rnd > 0.7)                  → complete ap stops
        throw new Error ("Something Went Wrong")
}
```

Catch errors while rendering.

```
class ErrorBounds (person).
{
    Component Did Catch = (error, info) => {
        Set State ({ hasError True, msg: error})
    }

    render(){
        If ( hasError)
            return <h1> error </h1>
        if(else
            return <h1> no error </h1>
```

Error Boundry

```
App(){
    return(
        <div>
            <h1> Welcome </1 h1>
            <Error Boundry>
                <person />
            < Error Boundry>.
```

Forms & Validation

Uncontrolled component → Browser handles them, we take from Dom.
Controlled component → React completely handles it, saved in component state

① Html → form data handled by Dom
② React → component
↳ all data stored in state

---

ref → is used to recieve value from "Dom" for uncontrolled component

---

React.createRef()

⇒ this.myRef = React.createRef()

<input type="text" ref={this.myRef} />

↳ creation of ref
↳ assign ref to input node from Dom

Composition vs inheritance

Combining different component to form larger one

Parent
↓
derived class
↓
derived class

Context

const userContext = React.createContext
({
currentUser: null;
});

→ Provider

<UserContext.Provider
value={{ currentUser: user }}>
<AccountSummary />
<AccountProfile />
</UserContext.Provider>

---

consumer

const AccountSummary = (ps) => {
return (
<AccountHeader />
)

const AccountHeader = (prop) => {
const context = React.useContext(
UserContext)

return (
<h2>{context.currentUser
.name}</h2>
)
}

useContext(

50) Redux

Create react app

npm install redux react-redux

① Store ② Action ③ Reducer

④ Dispatch

SOREDA

① create store →

import { createStore } from redux.

const mystore = createStore (reducer-name)

② Create action

```
const increament = () => {
    return{
        type: 'INCREAMENT' //action name
    }
}

const Decreament = () => {
    return{
        type: "DECREAMENT"
    }
}
```

Action name

Create Reducer

```
const counter = (state = 0, action) => {
  switch (action.type) {
      case "INCREAMENT":
        return state + 1;
      case "DECREAMENT":
        return state - 1;
  }
}

let store = createStore(counter)
```

① display store =

console.log (store.getstore())

④ Dispatch Actn

store.dispatch (increament());

store.subscribe (() => { })

to dispatch changes to store

to get changes of store subscribe component

Store → A js object with special functions

to update store ⇒ create action object that ~~subscribes~~ describes. then dispatch to store.

store → store runs reducer function

Action → tells store what happened in application by type

reducer → f takes, action a state and update a returns new state

Dispatch store method, to update state, store.dispatch();

and pass action object

store.disp (increamin())

useSelector() → to get state from store

useDispatch() → to update

Node.js

- Node is framework for scalable server
- open source, serverside javaScript
- Node.js ⇒ runtime + library
- all Apis of Node are async Non blocking
- it is non waiting
- used for Data Intensive rather than CPU Intensive
- It uses single threaded Event loop
- another way to execute Code of machine

REPL (Read-Eval-Print-loop)

↳ Node Shell

↳ we can give commands or Node hello.js file

To include module → require(" ")
filepath/filename

Module

core → require("react") ("http")

local → require("./modules")
modules.exports.answer
var ans = require("./modules")
sysout(ans.answer)

third party

exports :- special object

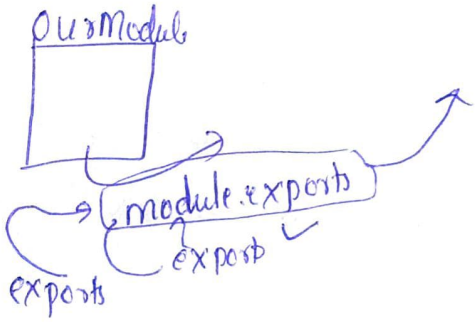
Exporting function

module2
```
exports.sayHello = function () {
        return "hello"
    };
exports.sayHola = function () {
        return "hala"
    }
```

```
var greet = require("./module2")
console(greet.sayhello
        sayhalo)
```

Exports Variation page 13, 14

exports is just module.exports little helper



e.g.
exp module.Exp

```
exports.func1 = function () {
        console.log("Hello");
    };

module.exports.func2 = function () {
        consol.("Bello")
    }
```

```
var temp = require("./module.Exp")
f1.func1()
f2.func2()
```

function sayHello(){
console("Hello")
}

const userName = "Sheelaka"

module.exports = {username, sayHello};

Or

module.exports.username = username
module.exports.sayhello = sayHello

Or

exports.username = username
exports.sayHello = sayHello.

const {username, sayHello}
= require("./sample")

Or

const helloModule = require("./sample")
helloModule.sayHello();
(helloModule.username)

functions, properties → exports.

Objects, json objects → module.exports.

npm (node Package manager)

↳ used to load third party Modules

→ installation
↳ globally → development tools & CLI packages (based)
npm install -g <package-name>
locally → used install framework & libraries
⊛ scope is its directory only

npm uninstall ↑
npm update ↑

Package.json

⊛ heart of entire app
↳ contains metaData of project
↳ Configuration file
↳ human-readable metadata (project name, Version, dependency list discription)

npm init → creates package.json
--save → to update package.json

package-lock.json

guarantees product is 100% reproducable by storing ~~package~~ version of installed packa
↓
currently installed versions

---

fs → file system manupulation interaction

fs.writeFileSync();
fs.readFileSync();
} Sync

fs.writeFile()
fs.readFile()

⊛ async function always taken Completion Callback as its last argument

---

URL Core module

↳ provides utilities for URL Resolution & Parsing

→ q = url.parse
q.host, q.pathname, q.search, q.href
q.protocol, q.query

fun(req, res){
res.write.
}

req.url → as string
req.method → Get, post, Delete
req.header →

```
var fs = require("fs")
function process_request(req, res){
    fs.readFile('bigSample.txt', fun(err, data)
    {
        res.writeHead(200, {
            Content-Type: 'text/plain'})
        res.write(data)
        res.end();
    });
var s = http.createServer(process_request)
s.listen(1337, '127.0.0.1');
s.listen(1337) → default local host
```

Routing :- use switch case to match url patterns

Promise

New feature of ES6

↳ use to write asynchronous code

↳ it represent Completn of async Functn

↳ Is object

① Create Promise

```
let promise = new Promise(function(resolve, reject)
{
});   → executer code
```

```
let promis = new Promise(function(res, rej){
    // things to do accomplish
    if (it worked fine)
        resolve('stuf word')
    else
        reject(new Error('it broke'))
```

* then()

Method in promise executed when code run reaches resolve

* it returns promise

① takes two arguments
callback functn ① success
② failure of promis.

```
p.then(onFullfilled(), onReject())
```

```
ex -> promise.then(
    function(result){ },
    function(error){ }
);
```

Consuming Promise

```
Const promise = new promise(
    (resolve, reject) => { if(true)
        resolve("resolv
    })
prom.then(msg => console.log("+msg
```

promise().then().catch()
success error

e.g

```
Const promise = new Promise(
    (resolve, reject) => {
        return reject("Hi");
promis.catch(err => Console.log(err))
```

Chained promise

* promise is mostly used in fetching data from server.

Ajax

```
function load(url){
  return new Promise(function(
  resolve, reject){
    const request= new XMLHttpRequest();
    request.onreadystatechange= function
      (e){
        if(this.readystate===4){
          if(this.status==200){
            resolve(this.response);
          }
          else{
            reject(this.status);
          }
    request.open('Get',url,true);
    request.send();
  });
}

load("url").then(res=>{
          const result= JSON.parse(res)
          $("#msg").text(result.msg)
        },
      error=> $("#msg").text(error)
        );
}
```

Fetch

```
fetch('url').then((res)=>{
    return response.json()
  }).then((data)=>{
        console.log(data)
  }).catch((err)=>{
        console.error(err)
  })
```

Async Await

→ added to ECMAScript 2017 (es 8)

→ async → put in front of function declaration turns that functn in async function

* async function returns promise

```
function hello(){
    return "hello"
  }
hello();
async function hello(){
    return "Hello"
  }
async ()=>{return "hello"}
```

---

e.g `hello().then((v)=>console(v))`

```
let f = async () => {
    console('async functn');
    return Promise.resolve("hello");
}
f().then((res) => {console(res)})
```

Await() (to pause) untill promise fullfilled
used inside async fun^n to wait for async operatn

* can be used infront of any async promise based function to pause your code

```
let result = await promise
```

example

```
let promise = new Promise((resolve, rej)
  => {
    setTimeOut(() => {resolve(
    "Hello")}, 4000);
  }
)

async function fun2() {
  let result = await promise
  console(result)
}
fun2();
```

Next Gen JS

→ Const JS 1.5
  to assign const
→ let → to define block scoped variable.
→ Arrow function

⑦ Exports & imports

```
Const person => {
    name: "krishna"
}
export default person
import prs from "Person.js"
import person from "./Person.js"
```

name is optional

```
import {baseData} from utility.js
import {clean} fr ---
```

utility.js — name must match

```
export const clean = () => {}
export const baseData = 10;
```

example

```
export function sum(x, y) {return x+y}
export var pi = 3.14
import * as math from "/_"
import {sum, pi}
```

Classes properties & Methods

- Constructor name is Constructor
- function keyword removed in class.

```
class stud {
    Constructor (name){
        this.sname = name
    }
    GetName() {
        console.log(this.sname);
    }
```

- super usage is same as of Java
- only one constructor allowed
- Spread & rest operator

```
numbers = [1,2,3
Const newNumbers = [...numbers, 4]
    Console(newNumbers)
        → [1,2,3,4]
Const newNumbers = [numbers, 4]
        [[1,2,3],4].
```

---

rest → merge arguments into array.

```
function sort(...args){
    args.sort();
}
sort(1,2,3,4,5)
```

Destructing

extracting array elements and object elements into resp variable

```
Var {name, age} = {name: 'krishna', age: 19}
```

e.g `Var rect = {x:0, y:10, width:15}`

```
Var {x, y, width} = rect
rect.x = 10; → value changed
({x, y, width} = rect) // Again assigning
```

Array Destructing

```
[a, b] = ['Hello', 'max']
```

Array functions

Array.filter

- filter(callback)

↳ result will be another array with filter element

Array.map

returns array with operation on each element

- map(callback)

---

objects

```
Const person = {
    name: 'max
}
Person.name = "krishna"
Const Second = {
    ...person
};
```

template string ${ }.

Variable → string.

- any thing that return value can be added to ${ }.

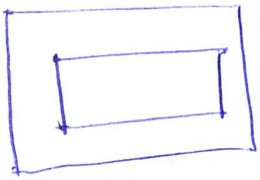
Bootstrap 6

↳ opensource frontend framework given by twitter

① Container . container class
- container-fluid

↳ it has grid system

② deals with margin
- Container → fixed width container with responsiveness
- container-fluid → width can vary according to need

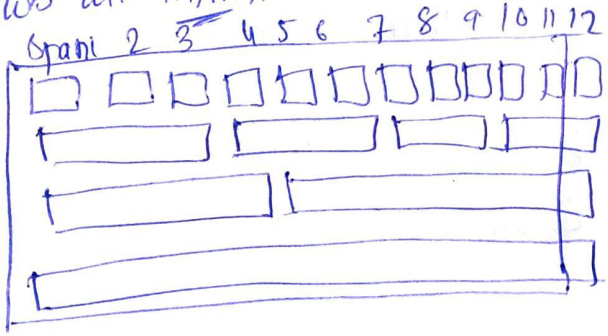


• container

• container-fluid

grid-system

screen divided upto
columns → upto 12/row
rows are infinite



class = "row"
- col — extra small device < 576px
- col-sm → small device
- col-md → medium device
- col-lg → large device
- col-xl → X-large devices

Styles (h1 - h6)

class = "text-muted"

<small class = "text-muted"> faded text </small>

I Love Krushna

73

global size = 1rem = 16px

class = "lead"
↳ text stands out

text-left
- center
- right →
- lowercase - the india
- uppercase → THE INDIA
- capitalize → The India

Text-color

① muted → gray
② text-primary → blue
③ success → green
④ info → light blue
⑤ warning → orange
⑥ Danger → red.

text background

"bg-success"

Bold italic

Tables

① basic table - class = "table".
② table-striped
③ table-bordered
④ table-condensed
⑤ table-dark

} Can be used at row col level

Jumbotron

big box for getting attention

↳ grey box with rounded corners

* class = "jumbotron"

Images

<img-rounded>  500px radius

<img-circle> 

<img-thumbnail> 

Icons

Alerts

class = "alert alert-success"

" alert-dismissable"

List

<ul class = "list-inline">

<li class = "list-inline-item" />

</>

list-Group

class = "list-group"

"list-group -item"

</>

Badge

Home

14 .

class = "list-group-item

list-group-item-success"

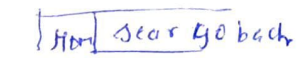
Navs

simple horizontal menu

<nav class = "nav">

<a class = "nav-item > Home</i>.

</nav>

? "nav-tabs" 

"nav-pills"

Vertical → "nav flex-column"

Togglable

Forms

<form

 → adds proper spacing

? input / <textarea>

↑ <select>

</>

<input class = "form-control" />

HTML

extension .html or .htm

↳ case insensitive

Head Tags

<base>, <meta>, <link> <script>

<style> <title>

Body section

bgcolor
background → image to background
text = "red"
link = "red", alink = "blue",
↳ to specify link colour

<body background = "http.... img"/>

Bold → <b> | shikethrough = <strike> or <s>
i → <i>
Under → <u>

subscript → <sub>
superscript → <sup>

<font size = 7 > Hello </>
<font colour = "Blue"> </>

---

new line →
paragraph <p>

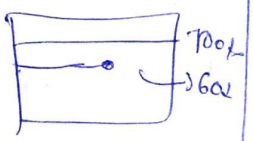
 → inline cointainer
to mark part of text

<!-- !> → comment

<hn align = " " > < > ← left right center

horizontal lines

<hr size = "5" >
<hr width = "100" > ↳ "60%"
<hr colour = " " >



Numbered list

 ind </>
 Sri </>

1. indi
2. srilanki

<ol start = 11 > ← change numbering start

Type = A, a, I, i
Upper case letter
small letter
cap roman
small roman

Unorded list

 type = Disc ●
 square □
 circle ○

Image

<img
alt → alternative text to display if image failed
Src = "home.png"
height = "100px"
width = "100px"

Table

attribute → Border = "2"
align = "center"
width
<td> → left align + regular
<th> → center + bold

HTML forms

<form method = " " action = " " >
</form>

<Input>
= text, password, hidden

Radio, checkbox, file selector

Button, submit, Reset

<textarea>
<select>
<button>

Text field

<textarea rows="10" cols="50" >
</textarea> [disabled] [readonly]

submit, reset,

checkbox

<input type=" " name=" " value=" "
[checked] [disabled] >

Radio button

<input type="radio" name, value [checked] [disabled]>

Password

name, size = n, value = "value" [disabled]

dropdown list

① <select> name, multiple = "True/False" size = n [disabled] >

<select>

→ <option> [selected] [disable] value = " " >

HTML 5

<nav> <summary> <header>

<section> <article> <aside>

<summary>

<video> <audio>

HTML 5

<canvas>

→ calender, date, time, email, url, search

<Input>

① placeholder
hold text in fade text

② Auto focus.
↳ to set focus of browser onit on reload

③ Required

<Input required>

④ Tel url, ⑧ Color color → to display colour palet

⑤ Number ⑥ Range

↳ simplified text

<Input type="range"
min='1', max="20"
value = 0 />

Date, Week, month

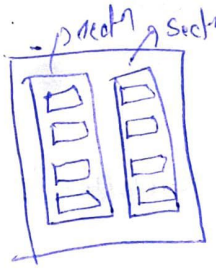
type = "date" ③ week. ⑤ datetime
② month ④ time ⑦ datetime-local

Canvas

↳ rectangular space where we can use Javascript to draw shapes

<canvas id= width="200" hight="100"></canvas>

Section



<nav> major navigatn block.

CSS

cascading style sheet

* selector { } declaration

{ colour: blue; font-size: 12px }
prop value prop value

<head>
<style>
p {
colour: red;
text-align: center;
}
</style>
<p>

Insert style sheet
① Embedded
② Linked style sheet

one.css

<head>
<Link rel="stylesheet"
type="text/css"
href="mystyle.css">
</head>

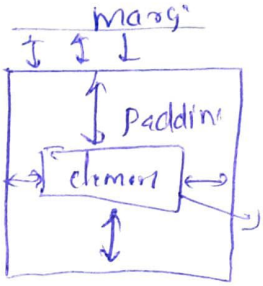
③ Inline
<input style="colour: red; width"/>

Types of selector
"HTML" → html tag selector { }
"class selector" .myClass { }
"id selector" → #para
H1.myClass { }
<h1 class="myclass">

CSS Box model

⊗ all elements can be considered as box
⊗ magin ⊗ Border
⊗ padding ⊗ Content

margin → no bg color
clears area around border, is transparent
padding → clears area around content, is affected by background color



p { border: 5px solid red }

CSS Background

background - colour
- Image
- position
- repeat → how image will be repeated

url('xyz')

{background-image: url('xyz.png')

Color
direction → writting direction
letter-spacing
text-align → center left right
text-decorate →
text-indent
text-shadow

text-transform
white-space
word-spacing

text-decoratn : overline abc
line-through ~~abc~~
underline abc

text-transform: uppercase;
lowercase;
capitalize;

p.date { }
p.main { }

<p class="date">

Text-shado :-

h1 { text-shado : 2px 2px red }

Font-styling

font-family
font-size
style →
variant → small caps / caps
weight → weight

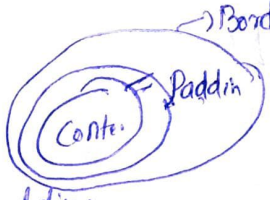
p {font-variant : small-caps; }

List styling

list-style : imag
- one
- two
- thr

list-style-position → position w.r.t content flow
-type

Border →

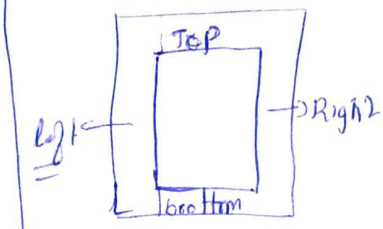


goes on content & padding

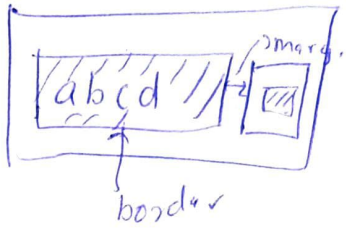
Margin → clears area around Border,
→ doesnot have background color
→ its transparent

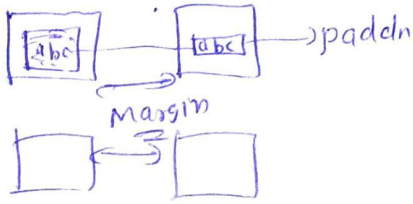
Padding :- clears area around content
padding takes pa background color

Margin



* It is invisible around content
* whereas padding is visible





border → width style color

background : url (" ");

Shadow Effect

tex-shadow → text
box-shadow → element

- C1 { text-shadow : 2px 2px; }
- C2 { box-shadow : 10px 10px grey }

Javascript

- object based programming language
- js can be in header body or other file

```
<script type="text/javascript">
</script>

<script src="imagin.js"
```

OnLoad → on loaging of page

- javascript is freeform language

Datatype:

Number (45, 125) | String " " | Boolean True/false | Null (absence of value)

Variables are loosly typed

- first letter can be a letter or underscore

Case

```
switch(variable){
  case outcom1:{
      // statem
      break;}

  case outcome2:{

      break:
      }

  default:{

  }
}
```

Function

- function can be assigned to variable

predefined functions

isFinite( ) → checks if arg is finite or not

isNaN( ); → check if not a numb

parseInt(str) →
parseFloat(str);

parseInt("3 bind") => 3
parseFloat("3.14meter") = 3.14
parseInt(0xFF) = 255
parseFloat("0.1") = 0.1
parseInt("0.1") = NaN

predefined object

```
var myString = new String("char")
```

String props

1. length
2. charAt(index)
3. concat()
4. IndexOf() (Returns -1)
5. IndexOf("Hello", 2) → start index
6. toLowerCase()
7. toUpperCase()
8. slice(start index, end index);
9. "Hello".slice(0,1) => H

String Functions

"Zero one two three four".split(" ")

↓

returns array of string

split("delimiter", [, limit Integer])

Var myArray = myString.split(",")

Substring, slice → 0 to end-1

str.replace("H", "h")

Slice(-3)

World "rld"

Array Methods

array.reverse()

array.slice(start, end)

(end-1)

array.join(seperatorString) **X

array.push()

array ⇒ ['fire', 'Air', 'water']

Array.join()

→ By default join by ',' comma

array.join(SeperatorString)

↳ ',', ' ', -,

join('-')

fire-air-water

All dates are from

Jan 1 1970 00:00:00

Dates before are invalid

Vaid = new Date() //current date

Date Methods

getDate() (1-31)

getDay() (0-6) 0 → sunday)

getMonth() (0-11, 0 → Jan)

getFullYear() 4 digit year

Array :-

~~delete()~~ → does not reduce array length

myArray

delete myArray[2]

del removes all indexes

Objects

person = { "name": "amit",
"age": 23
}

object initialiser

Var myfirstObj = { }

myfirstobj.name = "deep";

myfirstobj.lastname = "yadav";

(2) Var myO = {

name: "krushna",

lastname: "yadav

}

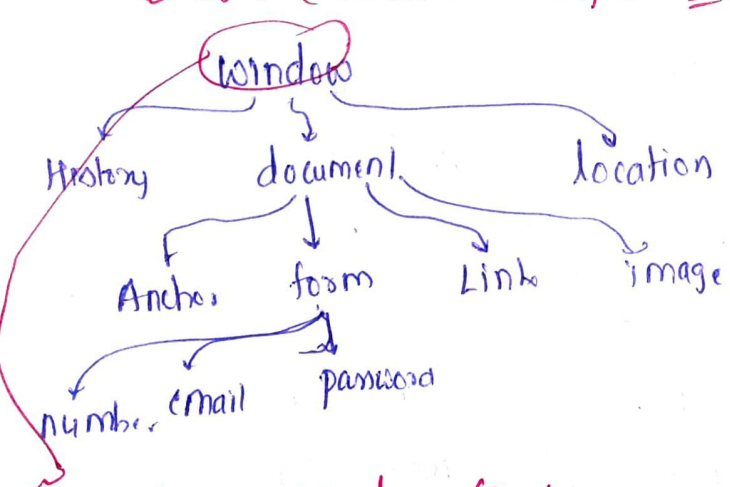
3

```
var myO = { }
myO.name = "krushna",
myO["name"] = "krushna"
console ( myO["name"])
```

adding methods to object

```
var person = {
    "name": "Andrew",
    "age": 21,
    info: function(){
        console(this.name)
    }
}
```

DOM (document object model)



window object methods

```
window.alert( )
window.confirm( ) "Exit now ?"
window.prompt ("enter x,y")
var input.
```

```
window.open (Url, name, specs)
```

*) Url → specifies the url of page to open, if no url a new blank window is opened

Name → name of windo

SetInterval SetTimeout

```
var int = self.setInterval ("clock()", 58)
                              ↑
                           function

function clock() {
    var ctime = new Date()
    document.getElementById("clock").value
    window.clearInterval(int) = ctime
    window.setTimeout ("alert('5 sec')"
    after 5sec                , 5000)
        alert will pop
```

Document object

Ⓐ HTML ~~obj~~ document loaded into browser it becomes document object

Ⓐ document object becomes root node of HTML document and owns all other nodes

document.anchors → returns all anchors in document

document.baseURI → returns absolute base uri of doc

document.cookie → Returns all cookies in document in name value pairs

document.forms → returns collection of forms

document.getElementById() → return element with that attribute

document.getElementsByName() →
- Access all element with specified name

- getElementsByTagName() → Node List with specified tag

- images, - lastModified → returns date & time last document was modified

- links → returns collection of links

- referrer → returns url of

- title → title of document

- Url → return url of document

- write() → to write Javascript code to document

- writeln() → write + new line

Modifying Content

document.getElementById("p1").innerHTML = "Hello world"

1 2
1 2

innerHTML = abcd

innerText = abcd

Regular Expression (RE)

RegExp() constructor

special literal syntax //

var re = // → pattern to match space

re = //g → match on global basis

/web/i → insentive case match

g → dont stop after first occuch

i → case insensitive

Word Boundry

\b →

\B → not word boundry

"222"

\d → searches for digit 0-9

\D →

/\d\d\d/ → "222"

/\D\D\D/ → "ABC"

↳ not digit

\s → single whitespace

/over\sbite/ → over bite ✓

overbite ✗

. → every character except newline

/.../ "ABC" "1+3" "A 3"

2 or 3 char.

[...]

/[AN]BC/ → AN B ✗ AB ✓ NBC ✓

[^...] Negate cha.

/[^AN]B/ → ABC ✗ NBC ✗ BBC ✓

positional metachar

"^" → at the begining of string or lin.

"$" → at the end of string or line

/^Fred/ => Fred ✓ "i am Fred" ✗

/Fred$/

Counting

* → 0 or more.

/ja*va/ → jva, java, jaaava

? → Zero or one

/ja?va/ → jva ✓ java ✓

+ → one or more

{n} → exactly n tim.

/ja{2}va/ → jaava

{n,} → N or more

{n,m} → least n max m

r. Regex Object

⊗ reg = /pattn/[gli]

new regExp (pattern, flag)

Eg

var zip = new Regex ("\\d{5}");

Where to us.

exec(), test(), search()

replace(), split()

/^\d{5}$/ → end with

start 0-9 5 99999

e.g 999-999-9999

/^[0-9]{3}[\-]?{2}[0-9]{4}$/

AJAX & JSON

Values supported in JSON

string → " " double quots

Numbers →

Booleans → True/false

Ajax

asynchronous Javascript and Xml

⊗ It is technique to mak. Web responsive

- it allows to fetch data without complete refresh

XMLHttpRequest (object of window)

- it is Javascript object

Methods

1. open ("method", "url", sync/async)
2. send ("content") → sends DOM, string
3. abort ( ) → terminate current request
4. getAllResponseHeaders ( )
   getHeader ( )
   SetReq header ("label", "value") → set headers before sending

XMLHttp Request → properties

1. onReadyStateChange → event handler that fires on each state change
2. readyState → current status of Req (0,1,2,3,4)
3. status → Http status returned by re.
4. responseText → get Response data as string
   responseXML → as XML

→ we assign callback function to it

JSON process

Var obj = JSON.parse (xmlhttp.responseText);

JQuery :-

- is used to manipulate DOM easily

Why JQuery

- it can locate element with class
- it can apply styles to element
  - $(document).ready(() => { alert("hell.. });

Selectors in JQuery // same as CSS

- .myclass element
- #myid element

Syntax

$ (selectorExpression)
jQuery ( )
$(selector).action()

$("h2").css("color", "red")
<h2>

jQuery("h1").html("Welcar")

Seled by tag nam.

$('p'), $('h1'), $('a')
↓ all p, all h, all a

Var para = $('p')

Selecting by ID

var ele = $("#myid") 19

- html(" ")
- css ( )
- hide ( )

Select by class name

$(".intro")

Selecting by input type

$('input[type="text"]').css(
"background", "yellow")

To select all i/p

$(":input") →

$(':input[type="radio"]')

$(':text') → all text

(':password') → all password

Basic filters

Index related selectors

:eq(), :lt(), :even, :odd

e.g → $("p:eq(1)")

→ select 2nd element

0, (1)

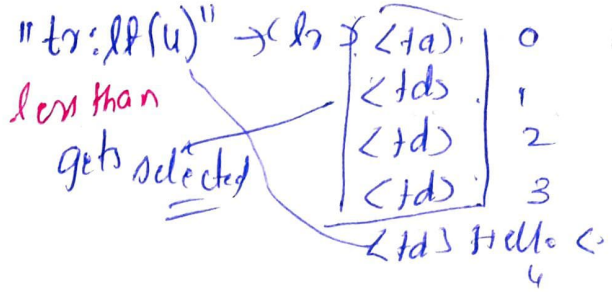
∴ $("p:second-child")

$("element:even") → even index

" odd" → odd index

:gt (grether than

"tr:lt(4)" → <tr>



:header

h1, h2 ...

$(":header").css( )

:not (invert above props)

Content filter

$("div:contains("hello")")

↳ search that div which

:empty → element with no child

$("p").filter(".intro")

all p with class .intro

children() returns all children() of particular element

method chaining

$("div").fadeOut().css("color","red").text("hello world");

Iterating through Nodes

- each(function(index, element))

```
$("li").each(function(){
        console.( $(this).text())
    })
```

Getting Content

text(), html(), val()

- text() → Content of selected elements
- html() →
- val() → Set or return value.

used for Setting & getting

Getting → $("#html").$("#test").html().

add & remove element

- append() | prepend()
- appendTo()

To wrap <H1> <H2>

- wrap("<H1>")
- remove()

$(#btn).onclick(function)

$("#div1").css({"color", "red", "background", "blue"}).html('changed')

Adding/removing class

addClass()

$('p').addClass('one')

hasClass() → returns True/False.

$('p').hasClass("one") → True

removeClass()

$('p').removeClass();

on/off

attaches one or more event handler to the element

$("#tabl.data").on("click", ()=>{})

Show / hide

show(duatn, callback)
hide(durath, callback).
toggle(durath, callback)
↳ visi → invisi

Slide down / slide up / toggle.

$(selector).slideDown(speed, callback)
slow/fast Normal.
or millisecond

fadeIn / fadeOut / fadeTo

gradually change opacity

fadeTo(speed, opacity, callback) → 0.1, 0.2 ...
fadeIn(speed, callback).

AJax

$(selector).load(url, data, callback)
$.get(url, data, callback, datatype)

```
$.get ('Getcontent.html', (data)=>{
        $('#targetDiv').html(data)
    },'html')

$.getJSON(url, data, callback)
```

AJax

```
$("button").click(()=>{
        $.ajax({
            url: "demo_test.txt"
          success: (r)=>{
                $("div").html(r)
            }
        });
```

Get Vs Post

- both are not secure
- Get → data → url (limited)(1024)
- post → " → body
- Get → Binary, files not allowed
  post → " allowed
- cryptojs → to encrypt the data

→ Https → uses

Http → 80

Https → 443 → signed certificates are used

SSL → Secure socket layer