**Day 1 Morning**

**22-04-2025**

Application/Software:

Types:

Desktop apps: notepad, ms office, code editor

Web apps: flipkart, amazon, myntra, fb, youtube

mobile apps

Web apps development

How web app works

1. Client

2. Server

Noodles

Restaurant-> Server

Customer -> client

Web app-> live

ip address : unique address of a device on a network

192.168.10.13.1

host a web app

domain name : flipkart.com, amazon.in

Server & client

Server is just another computer system

Its the server on which a web app will be executing

Server will be containing your entire application

Browser send a request to server (client)

Server -> will accept and process the request, once processing is done then

it will send a response

development on local machine -> host the app on server

end users -> browser -> server

Development:

Two aspects are there for web development:

1. Frontend: UI -> display

HTML: structure a web page

HTML is used to display data into web page

CSS : will style a web page

JS: to perform operations onto the client side/browser

livescript

React, Angular, Vue,

2. Backend : Database -> store the data

Processing of the data

Database: MySQL, mongodb, postgre, sql server, Cassandra, etc

Process the data: JavaScript

Node js is an interpreter

Express

Registration form:

Frontend: write logic to display form with input fields, and one submit button

Backend: process the data -> store into database -> send a response to the client

Login Form:

Frontend: to display the fields for username password and button

Backend: validation of the received username and password

from request with the db username and password

Eery browser is having one component that executes the js code: JS Engine

interpreter executes the code

google chrome: v8 engine

firefox: spydermonkey

MongoDb/MySQL -> M

Express -> E

React -> R/A

Node Js -> N

MERN

MEAN

Technical Specifications of Request and Response:

Regardless of the technology we are using to develop backend, the specification is going to

remain the same

W3C has designed all the specs for request and response

Specs for request:

1. We need a url to send a request (endpoint)

2. mention the purpose for sending the request

w3c has created certain request methods : GET POST PUT DELETE PATCH

if we haven't specified any request method explicitly then GET will be considered by default

ip address

myapp.com -> translated into ip address DNS

**Day 1 Evening**

**22-04-2025**

HTML: Hyper Text Markup Language

XML: Xtensible Markup Language

Programming lang : c, c++, java

Scripting lang : python, php, js

Markup lang : display the data (HTML), represent the data (XML)

In order to write syntax for markup langs we need tags

<body> <BODY> <bOdY>

in HTML all the tags are predefined

HTML is case insensitive

HTML is error free

HTML will provide a predefined tag to display any UI element

types of tags:

1. Container tag: <body> </body>

2. Empty tag: <br> <img>

create a .html .htm file

<html>

<head>

<title>My first web page</title>

</head>

<body>

</body>

</html>

<h1>

<h2>...h6

display a link:

<a></a> anchor

text & link

<a href="https://www.google.com">Visit google page</a>

<a href="https://www.google.com">Visit google page</a>

<a href="https://www.flipkart.com" target="\_blank">Open flipkart</a>

Application/Software:

Types:

Desktop apps: notepad, ms office, code editor

Web apps: flipkart, amazon, myntra, fb, youtube

mobile apps

Web apps development

How web app works

1. Client

2. Server

Noodles

Restaurant-> Server

Customer -> client

Web app-> live

ip address : unique address of a device on a network

192.168.10.13.1

host a web app

domain name : flipkart.com, amazon.in

Server & client

Server is just another computer system

Its the server on which a web app will be executing

Server will be containing your entire application

Browser send a request to server (client)

Server -> will accept and process the request, once processing is done then

it will send a response

development on local machine -> host the app on server

end users -> browser -> server

Development:

Two aspects are there for web development:

1. Frontend: UI -> display

HTML: structure a web page

HTML is used to display data into web page

CSS : will style a web page

JS: to perform operations onto the client side/browser

livescript

React, Angular, Vue,

2. Backend : Database -> store the data

Processing of the data

Database: MySQL, mongodb, postgre, sql server, Cassandra, etc

Process the data: JavaScript

Node js is an interpreter

Express

Registration form:

Frontend: write logic to display form with input fields, and one submit button

Backend: process the data -> store into database -> send a response to the client

Login Form:

Frontend: to display the fields for username password and button

Backend: validation of the received username and password

from request with the db username and password

Eery browser is having one component that executes the js code: JS Engine

interpreter executes the code

google chrome: v8 engine

firefox: spydermonkey

MongoDb/MySQL -> M

Express -> E

React -> R/A

Node Js -> N

MERN

MEAN

Technical Specifications of Request and Response:

Regardless of the technology we are using to develop backend, the specification is going to

remain the same

W3C has designed all the specs for request and response

Specs for request:

1. We need a url to send a request (endpoint)

2. mention the purpose for sending the request

w3c has created certain request methods : GET POST PUT DELETE PATCH

if we haven't specified any request method explicitly then GET will be considered by default

Specs for response:

1. Response must contain the status : status conveys that what happened with the request

Status consists of 2 things: status code + status message

status code and its message is already defined by w3c

200 OK

201 CREATED

202 203.... 299 2xx in case of success

300 301 302 303.... 399 in case if redirection

400 401 402 … 499 in case of failure due to client

400 BAD\_REQUEST

404 NOT\_FOUND

500 501 502 … 599 error due to server

500 INTERNAL SERVER ERROR

server database -> 10 GB

2. Response may contain the data : html, array, object

HTML

**Day 2 Morning**

**23-04-2025**

<img src="">

<h1>Image demo</h1>

<img src="download.jpg">

<h1>Image demo</h1>

<img src="images/download.jpg">

<h1>Image demo</h1>

<img src="../images/download.jpg">

<img src="../images/download.jpg" width="50" height="50">

Lists:

series of items

types of list:

1. ordered list: <ol>

1. Mumbai <li>

2. Delhi

3. Chennai

4. Jaipur

<ol>

<li>Bahubali</li>

<li>Babu Rao</li>

<li>Raju</li>

</ol>

<ol start="4" type="a">

<li>Bahubali</li>

<li>Babu Rao</li>

<li>Raju</li>

</ol>

by default numeric sequence will be there

type attribute decides the type of sequence: 1 a A i I

2. unordered list <ul>

<ul type="square">

<li>Bahubali</li>

<li>Babu Rao</li>

<li>Raju</li>

</ul>

type : disc| circle | square

Table:

<table></table>

outer-box -> row -> columns

cellpadding="10" -> spaces inside of cell

cellspacing="0" -> spaces outside of the cell

<table border="1" cellpadding="10" cellspacing="0">

<tr>

<td>Id</td>

<td>Name</td>

<td>Phone</td>

</tr>

<tr>

<td>101</td>

<td>Selmun Khon</td>

<td>899911111</td>

</tr>

<tr>

<td>102</td>

<td>Black Bug</td>

<td>988110111</td>

</tr>

</table>

<table border="1" cellpadding="10" cellspacing="0">

<thead>

<tr>

<th>Id</th>

<th>Name</th>

<th>Phone</th>

</tr>

</thead>

<tbody>

<tr>

<td>101</td>

<td>Selmun Khon</td>

<td>899911111</td>

</tr>

<tr>

<td>102</td>

<td>Black Bug</td>

<td>988110111</td>

</tr>

</tbody>

</table>

<p>(a+b)<sup>2</sup></p>

<p>H<sub>2</sub>O</p>

Form:

<form action="https://www.myapp.com/register" method="post">

</form>

Note: form tag is not there to display the form elements

form tag is responsible to submit the data to the server

action: contain the url on which request has to be submitted

method: specify the request method using which request has to be submitted

now all the input fields should go inside of the form tag

html provides 3 types of buttons:

1. submit button: relevant inside the form tag only, and its functionality is already defined

when user hits the submit button then action attribute of form tag executes

2. plain button: is not having the functionality predefined

its the developer who is responsible for handling the click of the button

3. reset button: relevant inside of the form only, it will clear all the field values

**Day 2 evening**

**23-04-2025**

how to display content of one web page into another:

<iframe>

<iframe width="1039" height="520" src="https://www.youtube.com/embed/4ZfoJPDXVHs" title="iPhone 16 In-Depth Review After 120 Days ft.iPhone 15 ⚡ 1 Major Issue..." frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" referrerpolicy="strict-origin-when-cross-origin" allowfullscreen></iframe>

<html lang="en">

<head>

<title>Document</title>

</head>

<body>

<iframe width="1039" height="520" src="https://www.youtube.com/embed/4ZfoJPDXVHs" title="iPhone 16 In-Depth Review After 120 Days ft.iPhone 15 ⚡ 1 Major Issue..." frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" referrerpolicy="strict-origin-when-cross-origin" allowfullscreen></iframe>

<iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3774.2220331037097!2d72.83065662497337!3d18.921563832251486!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x3be7d1c06fffffff%3A0xc0290485a4d73f57!2sThe%20Taj%20Mahal%20Palace%2C%20Mumbai!5e0!3m2!1sen!2sin!4v1745412311730!5m2!1sen!2sin" width="400" height="300" style="border:0;" allowfullscreen="" loading="lazy" referrerpolicy="no-referrer-when-downgrade"></iframe>

</body>

</html>

<!DOCTYPE html> it is to denote that we are using latest version/latest features of the html

It is a good practice to always have this tag

<meta> metadata: data about data

size of the content 50 KB

language

size of media that page is rendering

search keywords into meta tag

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta charset="UTF-8"> specifies that current html page is using Unicode convention for character encoding

responsive page: a page that will adjust the layout/content according to the available area

div tag:

<nav>

<img>

<a></a>

<a></a>

<a></a>

</nav>

<div></div>

<footer></footer>

<section>

<footer>

<nav>

CSS: Cascading Style Sheet

Css is for styling of the web page

Css provides predefined properties for style customization

<style>

selector{

property-name:value;

.

.

.

}

</style>

selector:

types of selector:

1. tag selector: we can write name of the tag as selector and it will select all the tags with that name

h1{

color: green;

}

2. id selector: to identify an element uniquely we can give it an id

in the selector we have to write #id-value

#head1{

color: green;

}

<h1 id="head1">Welcome to my page</h1>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

#head1{

color: green;

}

#head2{

color: red;

}

#head3{

color: blue;

}

</style>

</head>

<body>

<h1 id="head1">Welcome to my page</h1>

<h1 id="head2">Hello</h1>

<h1 id="head3">Hii</h1>

</body>

</html>

3. class selector: to identify few elements commonly

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

.green-text{

color: green;

}

</style>

</head>

<body>

<h1>Welcome to my page</h1>

<h1 class="green-text">Hello</h1>

<h1>Hii</h1>

<p class="green-text">this is para 1</p>

<p>this is para 2</p>

<span>this is span 1</span>

<span>this is span 2</span>

<span class="green-text">this is span 3</span>

</body>

</html>

4. universal selector: \* it will select all the elements in the web page

\*{

color: grey;

}

background-color

background-image

font-size

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

#box{

background-color: red;

width: 450px;

height: 250px;

}

</style>

</head>

<body>

<div id="box">

<h1>Welcome to app</h1>

<p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Corrupti rerum soluta minima et. Id, amet. Laboriosam dolorum recusandae totam quo nihil. Fugiat veniam eos aut esse eligendi? Libero, at porro.</p>

<button>Click me</button>

</div>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<style>

\*{

font-family: 'Segoe Ui Light';

}

#box{

/\* background-color: red; \*/

width: 450px;

height: 250px;

border-width: 2px;

border-style: solid;

border-color: black;

}

</style>

</head>

<body>

<div id="box">

<h1>Welcome to app</h1>

<p>Lorem ipsum, dolor sit amet consectetur adipisicing elit. Corrupti rerum soluta minima et. Id, amet. Laboriosam dolorum recusandae totam quo nihil. Fugiat veniam eos aut esse eligendi? Libero, at porro.</p>

<button>Click me</button>

</div>

</body>

</html>

to manage spaces inside of any element we have padding property

top, right, bottom, left

padding-top

padding-right

padding-bottom

padding-left

padding: 10px 20px 40px 25px; clockwise T R B L

padding: 30px; all side same value

padding: 40px 20px; T R

**Day 3 Morning**

**24-04-2025**

padding & margin

padding: internal spaces

margin: spaces outside of any element

margin-top: 40px;

margin-left: 15px;

margin-bottom

margin-right

margin:

ways to represent color value:

1. name of the color

2. hexadecimal code #3b5598

https://materialui.co/colors

3. rgb function rgb(r 0-255,0-255,0-255)

element-selector:event-name{

}

h1:hover{

color: red

}

border-style

border-width

border-bottom-width

id>class>tag>\*

types of css:

1. internal css: css code written inside the style tag within html

home.html

about.html

contact.html

service.html

.

.

.

100 html files

2. external css: css code written into separate .css file

we manually have to link external css with html file

<link rel="stylesheet" type="text/css" href="loginstyle.css">

3. inline css: css code written directly inside of any tag

<p style="font-size: 20px;color: red;">Need Account? <a href="#">Sign Up</a></p>

inline css is never recommended

**Day 3 Evening**

**24-04-2025**

JavaScript: JS is for providing functionality to the web page at the client side

Client side scripting

logics that are executed at the client side (browser)

For JS we don't have compiler

For js only the interpreter is there i.e. JS Engine

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

</script>

</body>

</html>

for JS everything is an object

web page is an object

image is an object

every tag is also an object

browser window is an object

some of the predefined objects:

document -> this object represents the entire web page

window -> this object represents the browser's window

console -> this object represents the inspect window console

with the help of this object we can log some messages on the console

as a developer we don't need to declare these objects, as they are already defined

document.write("Hello World");

window.alert("Hello Alert");

alert();

note: the properties of window object can be accessed directly without window. (without calling them on window object with member access operator)

console.log("hello developers");

document.write("Hello World");

alert("Hello Alert");

console.log("hello developers");

variables

x=6;

developer can't decide the data type for variables

data types do exists but they are assigned at the runtime by looking at the value assigned

in JS we only have a single type for numeric values i.e "number"

in JS variables can change their type as well:

x = 6;

console.log(typeof x);

x = "hello";

console.log(typeof x);

js is not a type safe lang

TS -> compiler -> js

in js we don't have a separate type for character as we have in java and c c++

**Day 4 Morning**

**25-04-2025**

number

string

Boolean

js is not type safe

operators

Arithmetic : + - \* / %

a = 6;

b = 2;

c = a+b;

console.log("sum is "+c);

"" -> +

'' -> +

``

a = 6;

b = 2;

c = a+b;

console.log("sum of "+a+" and "+b+" is "+c);

console.log(`sum of ${a} and ${b} is ${c}`);

relational operators:

< > <= >= == != === !==

a = 2;

b = "2";

console.log(a==b); true

a = 2;

b = "2";

console.log(a===b); false

logical operators:

&& || !

assignment operator: =, += -= \*= /= %=

a=5;

a=a+2; a+=2;

conditional operator/ternary operator:

a=6;

b=13;

assign a value to a variable conditionally

v3 = condition ? v1:v2;

c=a>b?a:b

Control structures:

decides the way of execution of the statements

1. sequential -> all the statements will be executed in a sequence i.e. line by line

2. conditional -> to execute statements conditionally we will use conditional control structure

if, if else, if else if, nested if, switch case

3. looping -> to execute statements repeatedly loops are used

while, for, do while, foreach

num = -2;

if(num>0){

console.log("number is positive");

}

else{

console.log("number is negative");

}

roll 101 -> Gappu

roll 102 -> Tappu

roll 103 -> Pappu

not found

https://www.w3resource.com/c-programming-exercises/conditional-statement/index.php#google\_vignette

any number is between 1 to 10 or not

a=4;

if(a>=1 && a<=10){

}

else{

}

91-100 A+

81-90 A

71-80 B

<=70 Fail

m=84;

m = 77;

if(m>=91 && m<=100){

console.log("A+");

}

else if(m>=81){

console.log("A");

}

else if(m>=71){

console.log("B");

}

else{

console.log("Fail");

}

Nested if

// c1 is true and c2 is true

// c1 is true but c2 is false

// c1 is false but c2 is true

if(c1){

if(c2){

s1

}

else{

s2

}

}

else{

s3

}

Switch case :

switch(variable){

case value1:stmt1;

case value2:stmt2;

case value3:stmt3;

.

.

.

}

roll = 102;

switch (roll) {

case 101:console.log("gappu"); break;

case 102:console.log("tappu"); break;

case 103:console.log("pappu"); break;

default: console.log("not found");

}

quantity

rate

total price

total 1000 - 2999 5%

3000 - 6999 8%

>7000 12%

Total Price before discount:

Discount given:

Net Payable amount:

Loops:

while, for, do while

while(condition){

body

}

a=1;

while(a<=5){

console.log("hello");

a++;

}

a=5;

while(a>=1){

console.log(a);

a--;

}

for(x=1;x<=5;x++){

console.log(x);

}

nested loops

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

.

.

.

100 rows

for (a = 1; a <= 10; a++) {

for (b = 1; b <= 5; b++) {

document.write("\*");

}

document.write("<br>");

}

**Day 4 Evening**

**25-04-2025**

Array:

array is a collection of elements

syntax:

a = [10,20,"Hello",true,6.8]

a = [10,20,"Hello",true,6.8];

console.log(a);

console.log(a[0]);

console.log(a[1]);

console.log(a.length);

a = [10,20,"Hello",true,6.8];

for(i=0; i<a.length; i++){

console.log(a[i]);

}

a = [5,4,18,2,9,13,8];

for(i=0; i<a.length; i++){

if(a[i]%2==0){

console.log(a[i]);

}

}

a = [5,4,18,120,9,13,8];

max = a[0];

for(i=1; i<a.length; i++){

if(max<a[i]){

max=a[i];

}

}

console.log(max);

1d array [1,2,3,4,5,6]

2d array [[10,20,30],[40,50,60],[12,18]] array of arrays/ collection of 1d arrays

a = [[10,20,30],[40,50,60],[12,18]];

console.log(a);

console.log(a.length);

for(i=0; i<a.length; i++){

for(j=0; j<a[i].length; j++){

console.log(a[i][j]);

}

}

JSON: JavaScript Object Notation

JSON is a way to contain the data into key and value pairs

syntax:

{k:v,k:v,k:v,k:v}

student = {

id:101,

name:"Suresh",

marks:89,

phone:"999111"

}

console.log(student);

console.log(student.id);

console.log(student.name);

console.log(student.marks);

console.log(student.phone);

JSON supports all type of values

student = {

id: 101,

name: "Suresh",

marks: 89,

phone: ["999111","88111"]

}

console.log(student);

console.log(student.id);

console.log(student.name);

console.log(student.marks);

console.log(student.phone);

for(i=0; i<student.phone.length; i++){

console.log(student.phone[i]);

}

student = {

id: 101,

name: "Suresh",

marks: 89,

phone: ["999111","88111"],

address:{

houseNo:"A-186",

area:"Subhash Nagar",

city:"Nodepur",

state:"ReactPradesh"

}

}

console.log(student);

console.log(student.id);

console.log(student.name);

console.log(student.marks);

console.log(student.phone);

for(i=0; i<student.phone.length; i++){

console.log(student.phone[i]);

}

console.log(student.address.houseNo, student.address.area, student.address.state);

array of json objects:

students = [{

id: 101,

name: "Suresh",

marks: 89,

phone: ["999111","88111"],

},{

id: 102,

name: "Mukesh",

marks: 92,

phone: ["900011","76611","9044566"],

},{

id: 103,

name: "Priya",

marks: 96,

phone: ["70183663","89544"],

}]

students = [{

id: 101,

name: "Suresh",

marks: 89,

phone: ["999111","88111"],

},

{

id: 102,

name: "Mukesh",

marks: 92,

phone: ["900011","76611","9044566"],

},

{

id: 103,

name: "Priya",

marks: 96,

phone: ["70183663","89544"],

}]

console.log(students);

console.log(students[0].id, students[0].name, students[0].marks);

console.log(students[0].phone[0]);

for(i=0; i<students.length;i++){

console.log(students[i].id, students[i].name, students[i].marks);

for(j=0; j<students[i].phone.length; j++){

console.log(students[i].phone[j]);

}

}

console.log(students[0]["id"]);

**Day 5**

**26-04-2025**

**// ... spread operator**

a = [10,20,30];

b = [4,5,6,7];

c = [...a, ...b];

console.log(c);

sid=101;

sname="Manoj";

student = {

id:sid,

name:sname

}

console.log(student);

sid=101;

sname="Manoj";

student = {

sid,

sname

}

console.log(student);

x=8;

y=4;

obj = {x,y};

console.log(obj);

o1 = {

id:990,

name:"Amit"

};

o2 = {

marks:87,

phone:"99011233"

};

o3 = {...o1,...o2};

console.log(o3);

x = "id";

y = "name"

obj = {

[x]:198,

[y]:""

} // value of x should become the key-name for obj

console.log(obj);

function:

is a collection of statements that are grouped together

function provides code reusability

single responsibility principle

factorial -> {

10 statements

f1();

f2();

}

f2(){}

f3(){}

s1

s2

s3

s4

factorial()

.

.

.

.

factorial()

.

.

.

.

.

.

factorial()

100 places factorial()

return-type function-name(arguments){

s1

s2

s3

return value;

}

function name(arguments){

s1

s2

s3

return value;

}

function add(x,y){

z = x+y;

return z;

}

result = add(2,6);

console.log(result);

function checkEligibility(age){

if(age>=18){

console.log("eligible");

}

else{

console.log("tu to baccha hai");

}

}

checkEligibility(4);

**note: in JS functions are also considered as objects**

obj = {k:v}

obj = function show(){

console.log("hello world");

}

console.log(function show(){

console.log("hello world");

});

obj();

obj = function show(){

console.log("hello world");

}

console.log(obj);

obj();

show(); // show reference has been destroyed when show assigned into obj

**we can assign a function definition into a variable reference**

**and that variable now will become a function itself**

**sync/async**

ways of variable declaration/ keywords to declare a variable:

1. var

2. let

3. const

a=3;

function demo(){

a=5;

console.log("inside func a=",a);

}

demo();

console.log("outside func a=",a);

**without var**, let, const keyword the variable will become **window object** property and always it will be **having a global scope**

**1. var:**

to assign the relevant scope to the variables we use var keyword

function demo(){

var a=5;

console.log("inside func a=",a);

}

demo();

console.log("outside func a=",a); // can't use a here

var a=5;

function demo(){

console.log("inside func a=",a);

}

demo();

console.log("outside func a=",a); // we can use a here as it is global now

we can redeclare a variable using var keyword

var a=5;

console.log("a=",a);

var a=7;

console.log("a=",a);

System.out.println(a);

int a = 5;

**in js when declaration is not there:**

**Uncaught ReferenceError: a is not defined**

**if declaration is there but value is not assigned yet**

**var a;**

**console.log(a); undefined**

**hoisting:** everytime before the js code execution, the js engine will search for the declaration statements and it will place all those statements at the top of the code.

console.log(a);

var a=5;

function show(){

if(true){

var a=5;

console.log("inside if a=",a); accessible here

}

console.log("outside if a=",a); accessible here

}

show();

**2. let:**

**is used to give a block scope to a variable**

the block inside the let variable is declared, only inside that block you can access let variable

function show(){

if(true){

let a=5;

console.log("inside if a=",a); // accessible here

}

console.log("outside if a=",a); // not here

}

show();

Cannot redeclare block-scoped variable

console.log(a);

let a = 6;

**hoisting is there but we can't access let variable before its initialization**

**3. const** is exactly same as of let keyword but we can't change the value of const variable after its declaration

const a=5;

functions and their variations:

function show(){

console.log("hiii");

}

var obj = show;

console.log(obj);

we can pass a function as argument to another function

a function that is passed as an argument of another function

is called callback function

function show(){

console.log("iinside func show");

}

function demo(arg){

console.log("arg =",arg);

arg();

}

demo(show);

**in the above example, show is passed as argument into demo**

**so we can call show as callback function**

function demo(arg){

console.log("arg =",arg);

arg();

}

demo(function show(){

console.log("iinside func show");

});

**anonymous function: a function without a name**

if definition has to be passed an an argument i.e. we are writing definition into the argument directly then giving the name to this callback function is pointless, so we can opt out of giving it a name

function demo(arg){

console.log("arg =",arg);

arg();

}

demo(function(){

console.log("inside func show");

});

var sum = function(a,b){

var c = a + b;

return c;

}

const rs = sum(2,7);

console.log(rs);

a new syntax has been introduced in order to write the a**nonymou**s functions i.e. **arrow function**

to have syntactical benefits we hav**e arrow function**

var func = ()=>{

}

var sum = (a,b) => {

var c=a+b;

return c;

}

const rs= sum(2,8);

console.log(rs);

var sum = (a,b) => a+b;

const rs= sum(2,8);

console.log(rs);

const increment = n => n+5;

const rs = increment(12);

console.log(rs);

**array functions**

var arr = [1,8,6,3,5,10,4,9,7,12];

arr.forEach((value,index)=>{

console.log(index,value);

})

var arr = [1,8,6,3,5,10,4,9,7,12];

var processArrayValues = (value,index)=>{

console.log(index,value);

}

arr.forEach(processArrayValues);

forEach()

is a function, that will take a callback function as argument and it will execute that callback for each element of array

and in callback we will get current value, index number

var arr = [1,8,6,3,5,10,4,9,7,12];

var evenArr = [];

arr.**forEach**((value)=>{

if(value%2==0){

evenArr.push(value);

}

});

console.log(evenArr);

var arr = [1,8,6,3,5,10,4,9,7,12];

var evenArr = arr**.filter**(value=>value%2==0);

console.log(evenArr);

**map ->**

var arr = [1,8,6,3,5,10,4,9,7,12];

var newArr = arr.map(value=>value\*2);

console.log(newArr);

**reduce ->**

const numbers = [15.5, 2.3, 1.1, 4.7];

var newArr= numbers.reduce((total,value)=>{

return total+value;

});

console.log(newArr);

**Day 6**

**28-04-2025 Morning**

**functions/json object -> prototypes**

**=>** class Name {

functions/properties

}

class Student{

constructor(){

}

}

**=>** class Student{

constructor(){

console.log("in cons");

}

}

var s1 = new Student();

var s2 = new Student();

**=>**

class Student{

constructor(){

this.id=101;

this.name="Suresh";

}

}

var s1 = new Student();

console.log(s1.id, s1.name);

**->**

class Student{

constructor(id,name){

this.id=id;

this.name=name;

}

}

var s1 = new Student(101,"Rohan");

console.log(s1.id, s1.name);

var s2 = new Student(102,"Manoj");

console.log(s2.id, s2.name);

class Student{

var phone;

constructor(id,name){

this.id=id;

this.name=name;

}

display(){

console.log(this.id,this.name);

}

}

var s1 = new Student(101,"Rohan");

s1.display();

var s2 = new Student(102,"Manoj");

s2.display();

**->**

class Rectangle{

constructor(l,b){

this.l=l;

this.b=b;

}

getArea(){

return this.l\*this.b;

}

getPerimeter(){

return 2\*(this.l+this.b)

}

}

var r1 = new Rectangle(4,2);

const area = r1.getArea();

const perimeter = r1.getPerimeter();

console.log("area=",area);

console.log("perimeter=",perimeter);

Inheritance: acquire the properties of a class into another class

Parent and child

child always acquire the properties of parent class

class A{

}

class B extends A{

}

class A{

show(){

console.log("hello in A");

}

}

class B extends A {

}

var bObj = new B();

bObj.show();

class A{

constructor(){

this.x=18;

this.y=20;

}

show(){

console.log("hello in A");

}

}

class B extends A {

}

var bObj = new B();

bObj.show();

console.log(bObj.x,bObj.y);

class A{

constructor(){

this.x=18;

this.y=20;

}

show(){

console.log("hello in A");

}

}

class B extends A {

constructor(){

super();

this.z=30;

}

}

var bObj = new B();

bObj.show();

console.log(bObj.x,bObj.y, bObj.z);

var obj = {

x:5,

y:function(){

console.log(this);

}

}

obj.y();

if a function is inside a json object then this keyword will point towards the current/same json object

var obj = {

x:5,

y:()=>{

console.log(this.x);

}

}

obj.y()

with arrow function, this keyword will now point to window object even if it is inside the json

function keyword-> it will have its own this keyword context and that will refer to same object/current object

arrow function -> it is not having its own context of this keyword, always it will be referring to window object

types of process in js:

1. synchronous : blocking nature, when tasks are dependent on each other

2. asynchronous : non-blocking nature, when tasks are independent from each other

setTimeout(task,timeout);

setTimeout(()=>{

console.log("hello);

},2000);

**28-04-2025 Evening**

sync-> blocking

async-> non blocking

setTimeout-> async

setTimeout(() => {

console.log("hello")

}, 2000);

console.log("bye");

result:

bye

hello

callback functions never executes immediately

they wait for a particular event to occur then a callback will be called

with the help of callback functions we can make a process async

such that the callback function should not block the execution of rest of the program

setTimeout(() => {

console.log("hello")

}, 3000);

setTimeout(() => {

console.log("hii")

}, 1000);

setTimeout(() => {

console.log("bye")

}, 2000);

console.log("calculating sum....");

setTimeout(() => {

var x=2;

var y=7;

var z =x+y;

console.log("sum is ",z);

}, 1000);

console.log("calculating factorial...");

setTimeout(() => {

var num = 5;

var f=1;

for(var i=1;i<=num;i++){

f=f\*i;

}

console.log("factorial is",f);

}, 3000);

console.log("calculating square...");

setTimeout(() => {

var n=6;

var sq = n\*n;

console.log("square is ",sq);

}, 2000);

if already multiple async operations are there and we want to execute them synchronously then we have to go into nesting of callback functions

setTimeout(() => {

console.log("hello")

}, 3000);

setTimeout(() => {

console.log("hii");

}, 1000);

setTimeout(() => {

console.log("bye")

}, 2000);

fetch ids from server async non blocking

based on id fetch data async non blocking

based on data check for voting eligibility async

fetchIds(()=>{

fetchData(()=>{

checkEligibility();

})

});

console.log("calculating sum....");

setTimeout(() => {

var x = 2;

var y = 3;

var z = x + y;

console.log("sum is ", z);

console.log("calculating factorial...");

setTimeout(() => {

var f = 1;

for (var i = 1; i <= z; i++) {

f = f \* i;

}

console.log("factorial is", f);

console.log("calculating square...");

setTimeout(() => {

var sq = f \* f;

console.log("square is ", sq);

}, 2000);

}, 3000);

}, 1000);

note: when we go into nesting of callback functions then code readability gets affected and difficult to maintain

nesting of callback functions is known as callback hell

we should always avoid callback hell

To avoid callback hell, Promise is used

Promise:

For any task first create a promise

Now Promise will have 3 states:

1. Pending: the promise task is not executed yet

now if the task has been executed then

2. Fullfill: in case of successful execution

3. Rejected : in case of any failure

Now on the basis of promise state, we have to take suitable actions

**In JS:**

**1. create a promise object:**

var myPromise = new Promise((resolve,reject)=>{

// promise executor function

// logic for the task

if(){

resolve();

}

else{

reject();

}

});

it is the responsibility of a developer to change the state of promise either to resolve/fulfill or rejected

resolve & reject are the two arguments inside the promise executor callback function, they will be initialized automatically by the Promise object

resolve() : will mark the promise as fulfill

reject(): will mark the promise as rejected

if we haven't called none of them then even after execution of the promised task, state of promise will remain as pending only

now we can take actions on the basis of promise state i.e. promise handling

To handle the promise, two predefined functions are there

1. then()

2. catch()

myPromise.then(()=>{

s1

s2

}).catch(()=>{

s3

s4

});

**var evenPromise = new Promise((resolve,reject)=>{**

**var num = 9;**

**if(num%2==0){**

**resolve();**

**}**

**else{**

**reject();**

**}**

**});**

**evenPromise.then(()=>{**

**console.log("promise resolved and number is even");**

**}).catch(()=>{**

**console.log("promise rejected and number is not even");**

**});**

var evenPromise = new Promise((resolve,reject)=>{

var num = 8;

if(num%2==0){

resolve("promise resolved and number is even");

}

else{

reject("Error: promise rejected and number is not even");

}

});

evenPromise.then((result)=>{

console.log(result);

}).catch((error)=>{

console.log(error);

});

function getPromiseObject(num) {

var evenPromise = new Promise((resolve, reject) => {

if (num % 2 == 0) {

resolve("promise resolved and number is even");

}

else {

reject("Error: promise rejected and number is not even");

}

});

return evenPromise;

}

getPromiseObject(7).then((result)=>{

console.log(result);

}).catch((error)=>{

console.log(error);

});

**29-04-2025 (Day -7)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<script>

// sum-1, factorial->3, square -> 2

// sum 2 3 -> factorial(sum result) -> square(fact result)

// console.log("calculating sum....");

// setTimeout(() => {

// var x = 2;

// var y = 3;

// var z = x + y;

// console.log("sum is ", z);

// console.log("calculating factorial...");

// setTimeout(() => {

// var f = 1;

// for (var i = 1; i <= z; i++) {

// f = f \* i;

// }

// console.log("factorial is", f);

// console.log("calculating square...");

// setTimeout(() => {

// var sq = f \* f;

// console.log("square is ", sq);

// }, 2000);

// }, 3000);

// }, 1000);

function sum(a, b) {

return new Promise((resolve, reject) => {

console.log("calculating sum...");

setTimeout(()=>{

var c = a + b;

if(c){

resolve(c);

}

else{

reject("Error: something went wrong in sum");

}

},1000);

});

}

function factorial(num){

return new Promise((resolve,reject)=>{

console.log("calculating factorial...");

setTimeout(() => {

var f=1;

for(var i=1;i<=num;i++){

f=f\*i;

}

if(f){

resolve(f);

}

else{

reject("Error: cannot find factorial");

}

}, 3000);

});

}

function square(n){

return new Promise((resolve,reject)=>{

console.log("calculating square...");

setTimeout(() => {

var sq=n\*n;

if(sq){

resolve(sq);

}

else{

reject("Error: cannot find square");

}

}, 2000);

});

}

sum(2,3)

.then((result)=>{

console.log("sum is ",result);

var obj = factorial(result);

return obj;

})

.then((factResult)=>{

console.log("factorial is",factResult);

var sqPromise = square(factResult);

return sqPromise;

})

.then((sqResult)=>{

console.log("square is",sqResult);

})

.catch((error)=>{

console.log(error);

});

</script>

</body>

</html>

function fetchIds(){

return new Promise((resolve,reject)=>{

console.log("fetching ids...");

setTimeout(() => {

var ids = [167,124,181,101]

if(ids){

resolve(ids);

}

else{

reject("Error: cannot find square");

}

}, 2000);

});

}

function fetchData(id){

return new Promise((resolve,reject)=>{

console.log("fetching data for...",id);

setTimeout(() => {

var userData={userId:id,name:"Aman",phone:"87777111",age:23};

if(userData){

resolve(userData);

}

else{

reject("Error: cannot find data");

}

}, 1000);

});

}

function checkEligibility(age){

return new Promise((resolve,reject)=>{

console.log("checking age...");

setTimeout(() => {

if(age>=18){

resolve("User is eligible");

}

else{

reject("Error: tu to baccha hai..!");

}

}, 1000);

});

}

fetchIds()

.then((idResult)=>{

console.log("ids are",idResult);

return fetchData(idResult[1]);

}).

then((dataResult)=>{

console.log(dataResult);

return checkEligibility(dataResult.age);

})

.then((msg)=>{

console.log(msg);

})

.catch((error)=>{

console.log(error);

});

Here in JS we have two ways to handle the promise:

**1. then & catch functions**

**2. async & await keywords**

**await**: when a function call is returning a promise then we can apply await keyword onto that call

await keyword handles the promise, and it will make rest of the statement wait for the current promise task to be executed completely then only rest of the code will continue

var obj = await sum(2,3); // obj will be having now sum result i.e. 5

var rs = await fetchIds();

if await keyword is there then that expression will return the value which has been passed into the resolve argument while defining the promise

**try{**

**var obj = await sum(2,3); // obj will be having now sum result i.e. 5**

**var rs = await fetchIds();**

**}**

**catch(error){**

**}**

if any function is returning a promise then onto its signature we can apply async keyword and if any function is containing await in the definition then async keyword must be applied on the signature

**30-04-2025 Morning(Day -8)**

DOM Manipulation:

What is DOM ?

Document Object Model:

whenever any html file is loaded into the browser, at that time before displaying the web page content it prepares a tree like structure of all the tags

This tree structure is referred as DOM

predefined document object is the object of DOM

<html>

<head>

<title></title>

</head>

<body>

<button>fetch data</button>

<div>

<h1>hii</h1>

<a href="https://www.flipkart.com">Visit e-commerce</a>

</div>

<form>

<input><p>password is in correct format</p>

</form>

</body>

</html>

once the page is loaded in the browser and dom has been prepared

then in that case if dynamically we are updating the dom structure then it is known as dom manipulation

1. we can introduce a new element into the web page

2. we can remove an existing element

3. we can change the content of existing elements

4. we can get the content of elements

5. we can get the value of attributes

6. we can change the value of attributes

7. we can change the style of elements

note: with the help of document object we can perform all the above operations

to perform any manipulation on the html elements, first we need to get their objects

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<h1 id="myheading">This is heading</h1>

<script>

var obj = document.getElementById("myheading");

console.log(obj);

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<h1 id="myheading">This is heading</h1>

<script>

var obj = document.getElementById("myheading");

console.log(obj.innerHTML); // innrHTML will return the content inside of the tag

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<h1 id="myheading">This is heading</h1>

<button onclick="getContent()">Get content</button>

<script>

function getContent(){

var obj = document.getElementById("myheading");

console.log(obj.innerHTML);

}

// innrHTML will return the content inside of the tag

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<h1 id="myheading">This is heading</h1>

<button onclick="changeContent()">Update content</button>

<script>

function changeContent(){

var obj = document.getElementById("myheading");

obj.innerHTML = "Updated content...!"

}

// innrHTML will return the content inside of the tag

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<h1 id="myheading" >This is heading</h1>

<div id="container" >

</div>

<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Animi rem, nesciunt illum ad tempora distinctio sed, facilis voluptatibus earum ipsum officiis, sint quidem. Corporis voluptas aperiam dolor magnam impedit vero.</p>

<button onclick="addElement()" >Add new heading</button>

<script>

function addElement(){

var heading = document.createElement("h2");

heading.innerHTML = "Hello h2 heading";

var containerDiv = document.getElementById("container");

containerDiv.appendChild(heading);

}

// innrHTML will return the content inside of the tag

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<h1 id="myheading">This is heading</h1>

<p id="para">Lorem ipsum dolor sit amet consectetur adipisicing elit. Animi rem, nesciunt illum ad tempora distinctio sed, facilis voluptatibus earum ipsum officiis, sint quidem. Corporis voluptas aperiam dolor magnam impedit vero.</p>

<button onclick="removeElement()">Remove element</button>

<script>

function removeElement(){

var paraObj = document.getElementById("para");

paraObj.remove();

}

// innrHTML will return the content inside of the tag

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<input type="text" id="textfield">

<a href="https://www.flipkart.com" id="shopping-link">online shopping</a>

<button onclick="getElementAttributes()">get attributes</button>

<script>

**function getElementAttributes(){**

**var textfield = document.getElementById("textfield");**

**var anchorObj = document.getElementById("shopping-link");**

**var typeValue = textfield.getAttribute("type");**

**var hrefValue = anchorObj.getAttribute("href");**

**console.log(typeValue,hrefValue);**

**}**

// innrHTML will return the content inside of the tag

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<input type="text" id="textfield">

<a href="https://www.flipkart.com" id="shopping-link">online shopping</a>

<button onclick="changeElementAttributes()">get attributes</button>

<script>

**function changeElementAttributes(){**

**var textfield = document.getElementById("textfield");**

**var anchorObj = document.getElementById("shopping-link");**

**textfield.setAttribute("type","password");**

**anchorObj.setAttribute("href","https://www.amazon.in");**

**}**

// innrHTML will return the content inside of the tag

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<input type="password" id="passwordfield">

<button onclick="togglePassword()" id="btn">show</button>

<script>

**function togglePassword() {**

**var passwordfield = document.getElementById("passwordfield");**

**var btn = document.getElementById("btn");**

**if (btn.innerHTML == "show") {**

**passwordfield.setAttribute("type", "text");**

**btn.innerHTML = "hide";**

**}**

**else{**

**passwordfield.setAttribute("type","password");**

**btn.innerHTML = "show";**

**}**

**}**

// innrHTML will return the content inside of the tag

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<input type="text" id="field1">

<input type="text" id="field2">

<button onclick="add()">add</button>

<h2 id="result"></h2>

<script>

function add(){

var n1 = document.getElementById("field1").value;

var n2 = document.getElementById("field2").value;

var n3 = parseInt(n1)+parseInt(n2);

document.getElementById("result").innerHTML = "sum is "+n3;

}

// innrHTML will return the content inside of the tag

</script>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<input type="text" id="textfield" onkeyup="validate()">

<p id="validation-msg"></p>

<script>

function validate(){

var textfield = document.getElementById("textfield");

var para = document.getElementById("validation-msg");

if(textfield.value.length != 0){

para.innerHTML="value is valid !"

para.style.color="green";

}

else{

para.innerHTML="Value is required!";

para.style.color="red";

}

}

// innrHTML will return the content inside of the tag

</script>

</body>

</html>

**30-04-2025 Evening(Day -9)**

Ajax: Asynchronous JavaScript And Xml

From browser using ajax, we can send a request to the server (for any purpose) and get the response without blocking other process in browser and also without reloading the browser

steps:

1. create an object that will send request

var xhr = new XMLHttpRequest();

2. mention url and request method

xhr.open(method, url)

3. send the request

xhr.send()

4. read response

to read the response we need to get the state of the request

to get the state of the request there is a property of xhr object i.e. readyState

readyState is having values ranging from 0 to 4

0 -> XMLHttpRequest object is created

1 -> when request has been opened i.e. when open() has been called

2 -> when request has been sent

3 -> when server accepts the request and starts processing it

4 -> when response is ready at the client side/ response received from server

xhr.onreadystatechange=()=>{

}

when ready state is changed then onreadystatechange event will be fired by browser

"[{},{},{}]"

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<button onclick="fetchData()">Fetch Data</button>

<table border="1" cellpadding="10" cellspacing="0">

<thead>

<tr>

<th>Id</th>

<th>Name</th>

<th>Phone</th>

<th>Username</th>

<th>Email</th>

</tr>

</thead>

<tbody id="tbody">

</tbody>

</table>

<script>

function fetchData() {

var xhr = new XMLHttpRequest();

xhr.open("GET", "https://jsonplaceholder.typicode.com/users");

xhr.send();

xhr.onreadystatechange=()=>{

// console.log(xhr.readyState);

if(xhr.readyState === 4){

if(xhr.status==200){

console.log("got response...");

// console.log([10,20,40]);

// console.log(xhr.responseText);

var data = JSON.parse(xhr.responseText);

console.log(data);

var tbody = document.getElementById("tbody");

for(var i=0;i<data.length;i++){

var tr = document.createElement("tr");

var idCol = document.createElement("td");

var nameCol = document.createElement("td");

var phoneCol = document.createElement("td");

var usernameCol = document.createElement("td");

var emailCol = document.createElement("td");

idCol.innerHTML = data[i].id;

nameCol.innerHTML = data[i].name;

phoneCol.innerHTML = data[i].phone;

usernameCol.innerHTML = data[i].username;

emailCol.innerHTML = data[i].email;

tr.appendChild(idCol);

tr.appendChild(nameCol);

tr.appendChild(phoneCol);

tr.appendChild(usernameCol);

tr.appendChild(emailCol);

tbody.appendChild(tr);

}

}

// else if(){

// to handle other status code apart from 200

// }

}

}

}

Xhr.onload = function(){

If (this.status===200){

Console.log(“this.responseText”)

}

Else{

Console.log(“Some error”)

}

}

**On loade ka matalb process complete ho gayi hai**

</script>

</body>

</html>

async function getUsersData(){

try {

const response = await fetch("https://jsonplaceholder.typicode.com/users"); // promise based if(response.status==200){

const data = await response.json();

console.log(data);

}

} catch (error) {

console.log(error);

}

}

Bootstrap:

media queries and flex related properties

grid system :

entire browser width will be divided into 12 equal sections

col-n: col-1, col-2, col-3, …, col-12

**1-05-2025 Morning(Day -10)**

**Node JS:**

it is a runtime environment for js

node js enables a developer to run a js program directly without using the browser

module: collection of functions, properties

types of modules:

1. in-built modules : modules which are predefined by node js

eg: fs, http, etc

2. user defined modules: module created by a developer according to the requirement

3. 3rd party module : modules provided by 3rd party (neither by node js nor a developer has created), we manually have to download and install them

eg: express js, react js, etc

npm: node package manager : it is a cli tool

demomodule -> show(), hide()

index.js

var obj = require("demomodule");

obj.show();

obj.hide();

const fs = require("fs");

fs.writeFile("myfile.txt","hello world", (error)=>{

if(!error){

console.log("write operation completed!..");

}

else{

console.log("write operation failed !");

}

});

console.log("after writing into file....!");

const fs = require("fs");

fs.writeFileSync("demofile.txt","this is a dummy text data")

console.log("after writing into file....!");

const fs = require("fs");

fs.readFile("myfile.txt",(error,data)=>{

if(!error){

console.log(data.toString());

}

else{

console.log(error);

}

});

console.log("after reading file...!");

const fs = require("fs");

var data = fs.readFileSync("myfile.txt");

console.log(data.toString());

console.log("after reading file...!");

**1-05-2025 Evening(Day -10)**

CRUD operations:

C: Create

R: Read

U: Update

D: Delete

npx json-server data.json

"editform.html?postId=10&name=hhh&age=17"

1. .html file create -> list, save.html, edit.html

2. save.html -> create a form

handle the submit event

read form data

submit it to server using fetch()

3. list.html -> create a empty table

in script write code to send request to fetch data

once data is there in response, using dom manipulation write that data into table

create a action column that will contain delete & edit button

first implement delete button

then implement edit button

4. edit.html -> read id from url

fetch data by the id

fill input fields

handle the submit event

read form data

submit it to server using fetch()