MERN: MongoDB, ExpressJS, NODEJS, ReactJS.

- Frontend : ReactJS.
- Backend : NodeJS, ExpressJS.
- DataBase : MongoDB.

SQL (optional) or as a secondary DB.

#Introduction:

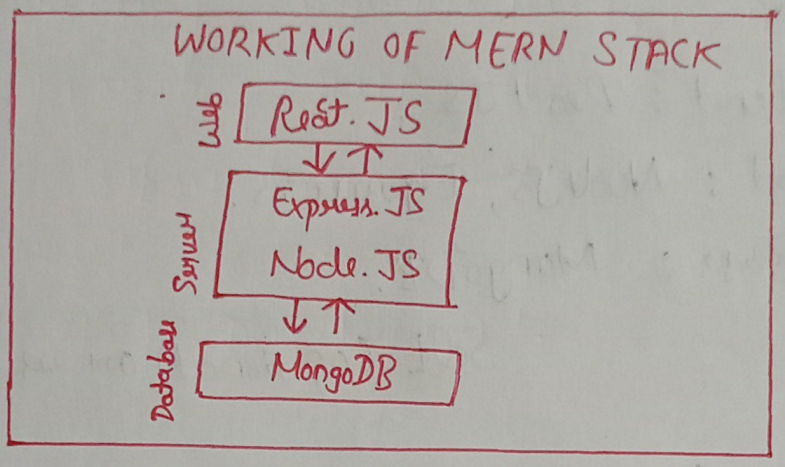
The MERN stack is a widely adopted full-stack development framework that simplifies the creation of modern web applications. Using JavaScript for both frontend and backend enables developers to efficiently build robust, scalable, and dynamic applications.

↳ What is MERN stack?

MERN stack is a JavaScript stack that is used for easier and faster deployment of full-stack applications. MERN stack comparies of 4 technology namely: MongoDB, Express, React and NodeJS. i.e.;

- MongoDB: Non Relational Database.
- ExpressJS: NodeJS web server.
- ReactJS: JavaScript Frontend Framework
- NodeJS: JavaScript Web Server or used for Backend related tasks.

↳ How MERN stack Works?



↳ Roadmap to become a MERN stack Developer:

- Learn basics of HTML, CSS, and JavaScript.
- Learn React which is a framework or frontend library for building UI.
- Learn NodeJS which is JavaScript Runtime environment.
- Learn ExpressJS, a framework built upon NodeJS to simplify the process of creating web application and API building.
- Learn MongoDB, a NoSQL database to store or retrieve data from database.

In this course, for learning MERN stack, we will first Revise the basics of JavaScript...

* Introduction to JavaScript...

JavaScript is a versatile, dynamically typed programming language used for interactive web applications, supporting both client-side and server-side development, and integrating seamlessly with HTML, CSS and a rich standard library.

- JS is a single-threaded language that executes one task at a time.
- It is an Interpreted Language which means it executes the code line by line.
- The data type of the variable is decided at run-time in JS that's why it called dynamically typed.

↳ Key features of JS:

(i) Client-side Scripting: JS runs on the user's browser, so has a faster response time without needing to communicate with the server.

(ii) Event-driven: JS can respond to user actions {clicks, keystocks} in real-time.

(iii) Asynchronous: JS can handle tasks like fetching data from servers without freezing the UI.

↳ Client-side nature of JS:

- Involves controlling the browser and its DOM (Document Object Model).
- Handles user events like clicks and form inputs.
- Common libraries includes AngularJS, ReactJS and VueJS.

↳ Server Side nature of JS:

- Involves interacting with databases, manipulating files and generating responses.
- Node JS and other frameworks like Express JS are widely used for server-side JavaScript, enabling full-stack Development.

↳ Applications of JS:

JavaScript is used in a wide range of applications, from enhancing websites to building complex applications. Like:

(i) Web Development: JavaScript adds interactivity and dynamic behavior to static website, with popular frameworks like Angular JS, React JS etc. enhancing development.

(ii) Server Application: Node.JS brings JavaScript to the server side, enabling powerful server application and full stack development.

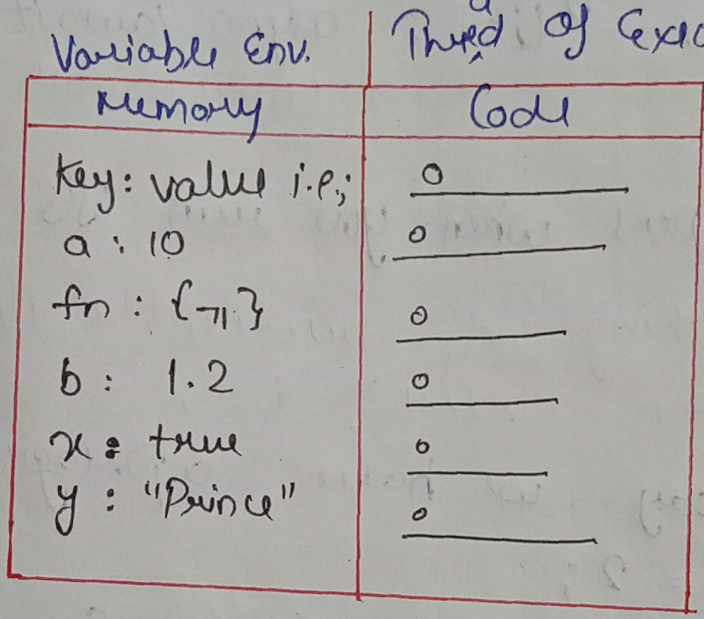
(iii) Game Development: JavaScript, combined with HTML5 and libraries like EaseJS enables creation of interactive games for the web.

* Working of JavaScript.

JavaScript is a dynamically typed, cross-platform threaded scripting and programming Language, it is an asynchronous and concurrent programming Language that offers a lot of flexibility.

It is single-threaded like Synchronous but also non-blocking like asynchronous.

↳ Execution Context;



In thread of Execution that is executed one line at a time.

In Memory component which is an environment in the CPU memory where variables are present in the form of key, value pairs, and function is present i.e.; whole function is present.

Since when you runs a JS code, the stack of execution context is created and stores all the data and variables in fact all the code in that stack in the form of key-value pair and as you known stack follows the principle of LIFO (Last in, first out).

Therefore, JS is a synchronous, single-threaded Languages.

- Single-Threaded means JavaScript executes one command at a time.
- Synchronous: It only executes one command at a time in a specific order i.e.; we can go through the next line of code once the current line has been completed executed.

*Note: we can make JS to work & or behave asynchronously by using various block of code like: async/await, promises etc. (will cover later).

↳ What happens when you run JS code:

"Everything in JavaScript happens inside the execution context" i.e.;

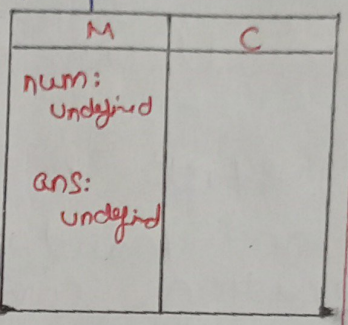
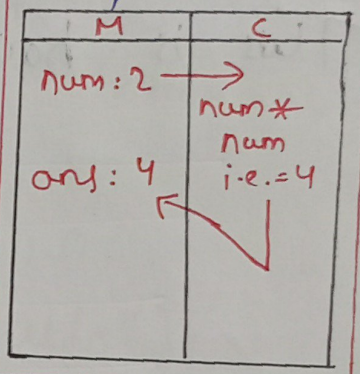
Let's say we have a program i.e.;

```
var a = 2;
function square(num) {
    var ans = num * num;
    return ans;
}
var square2 = square(2);
var square4 = square(4);
```

(num: Parameter; 2, 4: Argument)

↳ When we execute this code or any code, an execution context is created i.e.; also known as memory creation and code component creation phase;

It can be defined as;

Memory Component	Code Component
1st phase n: undefined Square: {...} Square 2: undefined Square 4: undefined	1st phase 
2nd phase n: 2 Square: {...} Square: 4 Square: 16	2nd phase 

Explaination:

(i) Step 1: In 1st phase JS allocate memory to every variables and functions and store undefined to the variable and whole code by function store in function.

(ii) Step 2: In 2nd phase, code execution phase, JS run through the whole program line by line i.e.; In phase2, all the calculation are done and values provided to the variable, also in this phase, functions are invoke and a brand new execution context of that function is created inside code components and we again go through these phases i.e.; phase 1 and phase 2.

(iii) Step 3: After completing the executing context, the executing context will be deleted.

Step4: when there are no. of function that are invoke then multiple execution context will created.

This all is done in a Call stack.

"Call stack is create global execution context and maintains the order of execution of execution contexts".

This is how JS runs Code.

* Hoisting in JS:

Hoisting is a phenomena is in JS by which you can use or get any variable and function even before programmer initialized it or put some value in it;

Lets say we have a code;

getName(); ⟶ It will have whole function.

console.log(x); → This will print or having value undefined

```
var x=7;
    function getName(){
        console.log("Hello");
    }
```

This is all done because of executing context; in which memory creation created these variables and function inside memory even before invoking or putting values in it.

"Even before the code starts executing the memory is allocated to this all variables and functions."

#note: Since Arrow functions will be created as an variable i.e.; it will print undefined.

React JS

Introduction to ReactJS :

ReactJS is a component-based JavaScript library used to build dynamic and interactive user interfaces. It simplifies the creation of Single-page applications (SPAs) with a focus on performance and maintainability.

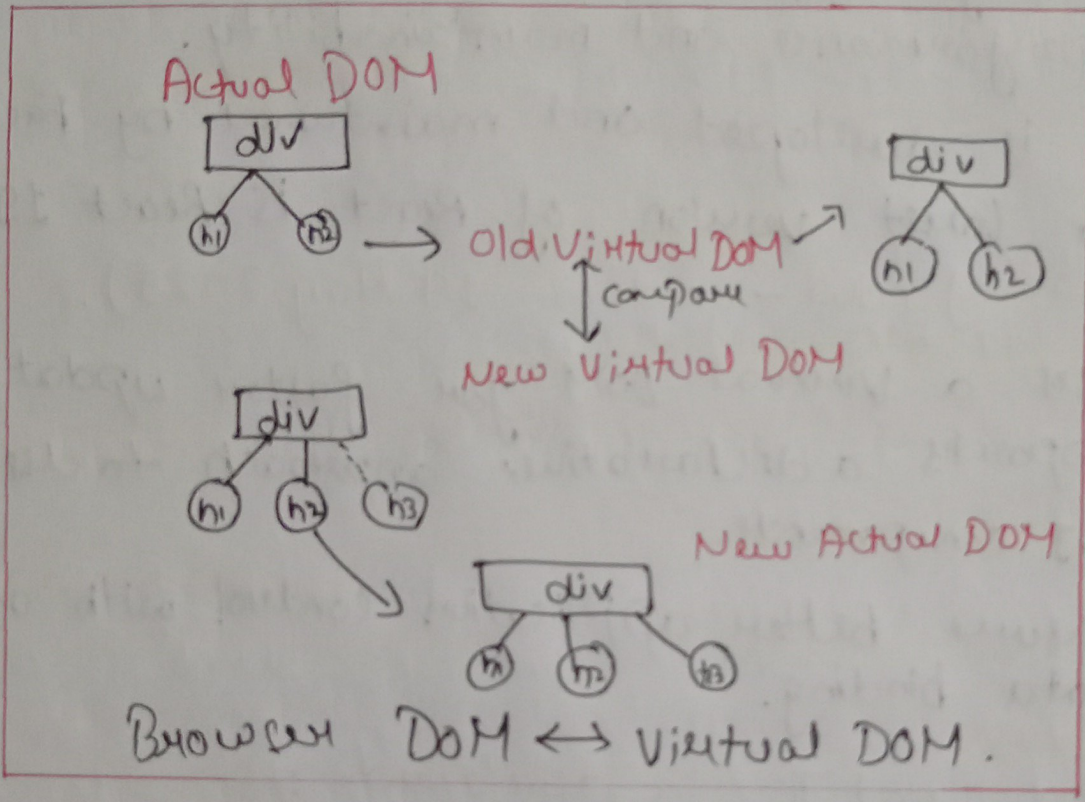
- It is developed and maintained by Facebook.
- The latest version of React is React 19 (as of now - on date 19 May 2025).
- Uses a virtual DOM for faster updates.
- Supports a declarative approach to design UI components.
- Ensures better application control with one-way data binding.

↳ Why ReactJS is developed :

ReactJS is was developed by Facebook to simplify building dynamic user interfaces. Before React, updating the UI was complex and messy. React introduced the virtual DOM to be efficiently update only the changed parts of a page, making web apps faster and easier to manage. It was created by Jordan Walke and released in 2013.

↳ How ReactJS works?

React operates by creating an in-memory virtual DOM rather than directly manipulating the browser's DOM. It performs necessary manipulations within the virtual representation before applying changes to the actual browser DOM. i.e.;



↳ Explaination:

(i) Actual DOM and Virtual DOM:

- Initially, there is an Actual DOM (Real DOM) containing a div with two children elements: h1 and h2.
- React maintains a previous Virtual DOM to track the UI state before any updates.

(ii) Detecting changes:

- When a change occurs (eg. adding a new h3 element), React generates a New Virtual DOM.

- React compares the previous Virtual DOM with the New Virtual DOM using a process called reconciliation.

(iii) Efficient DOM update:

- React identifies the differences (in this case, the new h3 element).
- Instead of updating the entire DOM, React updates only the changed part in the New Actual DOM, making the update process more efficient.

↳ "Hello World", Boiler Plate code for React;

```
import React from 'react';
function App(){
    return (
        <div>
          <h1> Hello, World </h1>
        </div>
    );
}
export default App;
```

* Note: We will use Vite + React app for creating React Apps. i.e.;

#Vite:

Vite is a build tool that provides a faster and more efficient development experience for modern web projects, and it works seamlessly with React. It leverages native ES modules to enable near-instant server start-up and fast hot module replacement (HMR).

→ To creat React project with Vite, run the following command in your terminal;

(make sure that node is installed in the pc).

Cmds:

- npm create vite@latest my-react-app --template react
- cd my-react-app
- npm install
- npm run dev.

↳ Why to use Vite with ReactJS;

Compared to older tools like Create React App (CRA), Vite offers;

Feature	Vite	CRA
• Development start time	Almost Instant	Slower (due to Bundling)
• Hot Reload	Fast & Efficient	Slower & less Reliable
• Build speed	Optimized with Rollup	webpack-based, slower

↳ Folder structure overview for Vite + React App:

```
my-react-app /
├── public/          ──→ Static files
├── src/             ──→ Application source code
│                        ↳ It contains components
│   ├── assets/      ──→ images and other static file
│   ├── App.css      ──→ Component-Specific Styles
│   ├── App.jsx      ──→ Main App Component
│   ├── index.css    ──→ Global Styles
│   └── main.jsx     ──→ Entry point of the app
├── .gitignore
├── index.html       ──→ Root HTML file
├── package.json
├── vite.config.js   ──→ Vite configuration
└── Readme.md
```

- For running the vite + React App we have to write the cmd;

  npm run dev.

* Quick Review on how to make Components;

- What is Components?

Ans. In React, a component is a reusable and independent building block of the User interface. Components divide the UI into smaller, manageable pieces, making it easier to develop, maintain, and reuse code. They are similar to JavaScript functions but work in isolation and return HTML via a render() function.

So basically, we have to create components of the code that is reusing in our website or repeating i.e.;

Lets say the Navebar, Header, Footer, Image Slider etc.

These parts are repeating in our website and are using repeatedly so we can using or writting code for multiple times we will create a component of it. or a separate component of it. and then call it or use it in our main file by using import. i.e.;

Lets say there is a Component image-slider that is in Image.JSX file, so we can import it as;

```
import image from "path_of_file";

<image>
```

* How React Renders in the browser from index.html.

Since, we know that React makes SPA (Single Page Application i.e.; it doesn't change or reload when changes occurs in the website that provide better UI experience for the user.

- So, how the whole React App Renders with the single index.html file's Root div.?

This is all done by the JavaScript i.e.; The root that is present in the index.html

```
<div id="root"></div>
```

This renders all the dyn data dynamically using JavaScript. Since the index.html file becomes an entry point for the # all the dynamic data and that data is entered into root element with the help of JavaScript.

- From where the root gets the dynamic data?

They gets the dynamic data through main.jsx file, this file tell React where and what to render inside the browser.

- Virtual DOM of React Renders to Real DOM.

React builds a Virtual DOM tree based on your JSX. and React efficiently updates the real DOM inside the #root div using different algorithms when state or prop changes.

This is how all things Done.

* JSX.

- What is JSX?

Ans. JSX is JavaScript Syntax Extension which is basically used to input JavaScript with HTML in react components.
JSX looks like HTML, But its not.

- const element = <h1> Hello World </h1>;

In JSX, we use;

- className instead of class.
- Returns only one parent element.
- All tags must be closed.

For writing JavaScript in JSX,

- we have use {} (curly Braces).
- Inside this, all the code is will JS. or dynamic code.

---

* Two Types of Components in React:

- Functional Components, (Modern & Preffered)
- Class based Components. (Older & less used).

(i) Functional: They are simple JS functions that return JSX.

Eg: (i)

```
function Greeting(){
    return <h1> Hello </h1>;
}
```

(ii) or using arrow function i.e;

```
const Greeting = () => <h1> Hello </h1>;
```

* Key Points of this type of Components:
  - Stateless or stateful using Hooks (useState, useEffect etc.).
  - Easier to read and test
  - Most Commonly used in Modern React.

(ii) Class: They are older & less used, They use JS classes and extend React.Component.

Eg:

```
import React, {Component} from 'React';
class Greeting extends Component {
    render(){
        return <h1> Hello </h1>;
    }
}
```

* Key points of this type of Components;
  - Used to be required for using state and lifecycle methods (like componentDidMount).
  - Now Mostly don't used.

Note: Import in used to import external files and styles/fonts/Package etc. as;

```
import logo from '..path-of-logo';
```

* Including Tailwind CSS with React;

- What is Tailwind css?

Ans. Tailwind CSS is a utility-first CSS framework for rapidly building modern websites and applications. Instead of writting custom CSS, you use pre-defined utility classes directly in HTML or JSX.

Since, There are some predefined classes like;

bg-blue-500, font-bold etc.

- How to use Tailwind?

We can use Tailwind either with CDN link or installing it. i.e.;

using Tailwind with Vite + React.

(Steps):

S1: Go to Tailwind CSS website; (tailwindcss.com).

S2: Then go to Docs and select vite (or using vite).

S3: Copy the CMDs written there.

- npm install tailwindcss @tailwindcss/vite.

S4: Copy the vite.config.js file and CSS like copy the vite content and paste to vite.config.js file and css cmds for our global CSS file. (Make sure that it is properly installed).

* Starting with React ;

Making our first React website ;

* Since for this ;
  - we will make a basic website,
  - we will use components like ; header, Footer, main etc.
  - we will use react-router-dom (Later we will cover this).

So, For making Componenents i.e. ;

In the SRC, make a Folder to Separate Components i.e. ; SRC
↳ Components.

Inside this we will make all the components like Header.jsx, Footer.jsx etc. and using the react-router-dom we will import these components to main file that is App.jsx.

To import components in the App.jsx file i.e. ;

First import them as ;

- import Header from 'Path-of Component';
- and under the return ;
  <Header />

It will automatically Render all the components accordingly.

Note: All the code is pushed on Github ;

Link: github.com/princebhatt03/MERN.

* Importing static data like images to website;

For importing static img images to website make sure that the image is present inside public/images folder and you can make images folder accordingly or use the links to render images;

I will use; public/images i.e.;

Now for importing static images i.e.;

- First, import the image;

  import logo from "path-of-images";

- Then used <img /> tag i.e.;

* If your are using import make sure that the images or static files are in assets/-- folder.

  Or if using public directory then use <img> tag or path for it.

* Props :

Since Props are properties in React are a way to pass data from one component to another i.e.; typically from a parent component to a child component.

↳ What is Props ?

Ans. Props are read only JavaScript objects that store the value of attributes passed to a React component. They help make components reusable and dynamic.

↳ How props works ?

Eg.
```
// Parent component
    function App() {
        return <Greeting name="Prince" />;
    }

// Child Component
    function Greeting (props) {
        return <h1> Hello, {props.name} </h1>;
    }
```

output : Hello, Prince.

Since, we can use props for different components and files for putting dynamic data to it.

Lets take another Eg. of Props as:

<Main SearchBar="True" />

(In App.jsx file).

{ props.SearchBar? <h1>Hello Search </h1> :
"No SearchBar" }

In this, there is prop that is SearchBar="True";
if this SearchBar = True, write Hello Search
or if SearchBar = false, write No SearchBar;
So this is how we can use props for a condition.

---

* Defaut Props: are used to spvify default values for a component's props in cas no value is provided by the parent component.

Eg.

```
function Greeting (props){
    return <h1> Hello, {props.name} </h1>;
    }
Greeting.defaultProps ={
    name: "Guest",
};
```

* Hooks :

Hooks are special functions introduced in React 16.8 that let you use state, lifecycle methods, and other React features in functional components without writing a class.

↳ Commonly Used Hooks ;

- useState() : Add state to a functional component
- useEffect() : Run side effects (eg. fetching data etc).
- useContext() : Accesses values from React's Context API
- useRef() : Creates a reference to a DOM
- useCallback() : Memoizes a callback function
- useReducer() : Manages complex state logic (like Redux but simpler).

↳ Eg. Using a useState Hook and useEffect Hook:

```
import React, {useState, useEffect} from 'react';

function Counter() {
    const [count, setCount] = useState(0);

    useEffect(() => {
      console.log('Component count:', count);
    }, [count]);

  return (
    <div>
      <p> You clicked {count} times </p>
      <button onClick={() => setCount(count + 1)}>
        Increase </button>
    </div>
  );
}
```

useState():

useState is a Hook that lets you add state to a functional components.

Before React Hooks, only class components could hold state using this.state.

- Syntax: const [state, setState] = useState(initialValue)
  - ↳ state : The current state value.
  - ↳ setState : A function to update the state.
  - ↳ initial Value : The starting value of the state. (can be a number, object, array, boolean, string etc

Node JS

Node JS is an open-source, cross-platform JavaScript runtime environment, that allows you to run JS on the server.

It is built on Chrome's V8 JavaScript engine for executing JavaScript code outside of a Browser. It provides an event-driven, non-blocking (asynchronous) I/O and cross-platform runtime environment for building highly scalable server-side applications using JS.

It is developed by Ryan Dahl in 2009, i.e., use for developing Restful APIs, microservers and web Application.

Ryan Dahl used the V8 engine of chrome and join this with C++ program and make NodeJS.

---

* NPM: is node package Manager that is used for managing node packages.

For initializing a project we use;

npm init (then enter all information)

//this is used to create package.json i.e., having all information related to projects.

↳ For install package ;

npm i express --save

or

npm i -g nodemon

-g is used to install it globally.

Modules; It is used for connecting two JS files
Let say we have two JS files;

index1.JS

```
const prince = require('./index2.');
c.log(prince);
```

index2.JS

```
prince = {
    name: "Prince",
    age: 21,
    developer: true,
};
Module.exports = Prince;
```

↳ Built-in Modules: that are already built in JS or nodeJS such as OS, etc. etc.

we can add these modules as same as ↑

```
const mod = require('OS');
//For adding external modules.
```

↳ Core Modules: Consider modules to be the same as JS libraries. A set of functions you want to include in your application.

↳ Two types of Modules → Sync or Asyn.

(i) Sync: That executes one function or line of code at a time.

↳ fs.readfileSync("demo.text");
↳ fs.writefileSync("demo.text", "Hello");

(ii) Async: That executes a function with a callback function.

↳ fs.readFile("demo.txt", "Hello", () => {
    clog("done");
});

#NPM: is Node Package Manager that is used to download JavaScript packages from Node Package Manager, whereas npx is used to execute JavaScript packages downloaded this way.

↳ for initializing NPM;

* npm init or npm init -y

for installing various packages we have to write;

npm i packageName (for folder)

or

npm i packageName -g (for global download)

* There are various of npm packages, we just to write;

const valid = require ('validator');

↳ for getting or plugin the external package.

#IIFE: Immediately Invoked function Expression (IIFE) are JavaScript functions that are excuted immediately after they are defined.

They are typically used to create a local scope for variables to prevent them from polluting the global scope.

↳
```
(function (){
    var a = 10;
    c.log(a);
})();
```

Creating a Web Server in Node JS;

Since like PHP and Other Backend Programming language, Node JS doesn't require any external application to create a web server;

There's a Built-in HTTP Module i.e., used to create a web-server in Node JS.

"Node Js has a built-in Module called HTTP, which allows Node Js to transfer data over the HTTP."

```
var http = require('http');
```

Eg.

```
const http = require('http');
http.createServer(function (req, res) {
    res.write('Hello World');
    res.end();
}).listen(8000);
```

or

```
const http = require('http');
const server = http.createServer((req,res)
    =>{
        res.end("Hello");
    });
server.listen(3000,() => {
    console.log('Server is listening on Port 3000');
});
```

#NodeJS Repl: The REPL feature of Node is very useful in experimenting with Nodejs codes & and to debug JS codes.

Repl stands for;
Read, Eval, Print, Loop. i.e.,
(i) Read User's Input.
(ii) Takes and evaluates the data structure.
(iii) Print the result.
(iv) Loops the above command until the user presses the ctrl+C twice.

For entering in the Node Repl, we have to open terminal then simply write "Node" then we enter into Repl.

#Modules; Modules are the JavaScript Libraries or same as the JS Library. It is a set of functions you want to include in your application.

NodeJS has a set of built-in Modules which you can use without any further installation. i.e.; Core Modules.

---

1) fs Modules: File System Modules;

```
const fs= require('fs');
fs.writeSync('read.text', 'Hello');
```

// It makes a file name read.text and Hello in it.
If we run again this then it overwrite the new sentence but in the same file.

for write next line to the file;
we use;

fs.appendFileSync("read.txt", 'Hello1');

and there are many method for it.
Such as read, link etc.

*Since read provides the Buffer_data i.e., Binary data.
For debugging data we have to use;

buf_data.toString();

then it will print original data.

---

b) Since There are all the Sync version of Modules; Now we will start the Async of them;

i) Async Modules requires a callback function;

```
fs.writeFile('read.txt', 'hello', err => {
    console.log('Done');
});
```

This is Async Modules in NodeJS.

Creating own Modules and Exporting them;

Let say we have 2 files ; index.js and index2.js and in ;

index2.js we have;

```
const sum = (a,b) => {
    return a+b;
};
module.exports = sum;
```

index.js we have;

```
const add = require('./index2.js');
const a = sumadd(5,5);
console.log(a);
```

∴ this is how we can export Modules to another file.

↳ Exporting External Modules or Packages from NPM;

(i) Express ; for this first we have to write

↳ npm init or npm init -y

then install the given Package i.e.;

↳ npm i express

↳ Now setup the Package ;

```
const express = require('express');
Const app = express();
app.get('/', function (req, res) {
    res.send('Hello World');
});
app.listen(3000);
```

Event Module

Node JS has a built-in Module called "Events", where you can create-, fire-, and listen for-your own events. To include the 'events' we have to create a class and instance of it;

```
const EventEmitter = require('events');
const events = new EventEmitter();

// class and instance of it ↑

events.on('eventName', () => {
        console.log("Prince");
    });

events.emit('eventName');
```

∴ we can call or emit multiple callbacks on events by the one EventEmitter;

Simply we have to make more functions of it...

∴ Also we can pass parameter in it;

```
event.on('checkPage', (status, msg) => {
        console.log(`Status code is ${status}
                and ${msg}`);
    });

event.emit("checkPage", 200, "done");
```

Streaming :

Streams are objects that let you read data from a source or write data to a destination in continous time. In Node.JS, there are 4 types of streams;

- Readable,
- Writeable,
- Duplex,
- Transform.

"Streaming means listening to music or watching video in real-Time, instead of downloding a file to your device."

Streaming can used by require (fs) and creating a http or express server.

↳

```
const http = require ('http');
const fs = require ('fs');
const server = http.creatServer ();
server.on ('request', (req, res) => {
    const stream = fs.createReadStream ('demo.txt');
    res.end
    stream.on ('data', data => {
        res.end (data);
    });
    stream.on ('end', () => {
        res.end ();
    });
server.listen (3000);
```

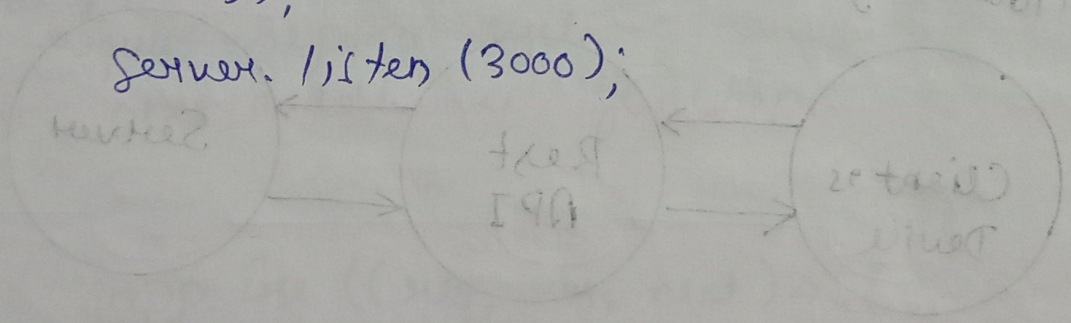
↳ Stream Pipe() : It is method used to take a readable stream and connect it to a writeable stream.

// Syntax ;

readable.pipe(destination [options])

// code ;

```
const fs = require('fs');
const http = require('http');
const server = http.createServer();
server.on('request', (req, res) => {
  const stream = fs.createReadStream('demo.txt');
  stream.pipe(res);
});
server.listen(3000);
```



Rest Full API:

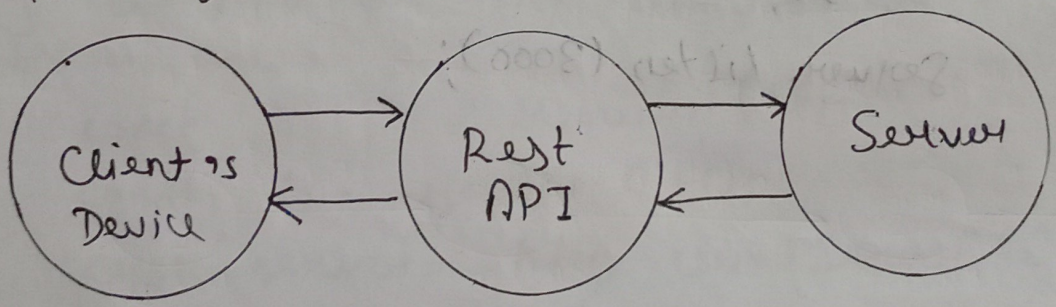
A Rest API (also known as Rest full API) is an application programming interface that two computer systems use to exchange information securely over the internet.

Here Rest and API words are;

- Rest: Representational State Transfer (REST) is a software architecture that imposes conditions on how API should work.
- API: Application programming interface (API) defines the rules that you must follow to communicate with other software systems.

Developers use or create APIs so that other applications can communicate with another software programmatically.

↳ How they work;



↳ Building an API;

```
app.get ('/', (req, res) => {
    res.send ('Home');
});

app.get ('/api', (req, res) => {
    res.send (users);
});
```

here users is an .json file which required...

Middleware; The Middleware in NodeJS is a function that will have all the access for requesting an object, responding to an object, and moving to the next middleware function the application request-respond cycle.

Express serves as a routing and Middleware - framework for handling the different routing of the webpage.

"It is Basically a middleman or function that has access to our req and respond to check whether we are a valid user or using a valid connection".

↳ Tasks Performed by Middlewares;

- Execute any code.
- Make changes to the request and the response object.
- End the request-respond cycle.
- Call the next middleware function.

↳ Code;

```
app.use((req, res, next) => {
    console.log('Hello Middleware');
    next();
});
```

∴ first the middleware function will run and then we will get the routes.

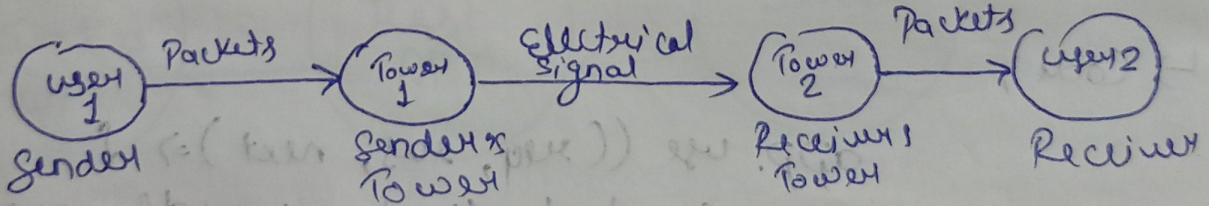
* Backend Development *

#Learn_what_Matters

The Backend Development involves the Logic, database, and other operations that are built behind the scenes to run the web server efficiently.
i.e., it is the server-side software.

↳ How Internet works;

When we send any request to the server or let say we are sending any data to another software which is in America then, the data from our device is sent in the form of packet to the nearest tower of * your service provider then it convert into electrical signal (i.e., packets are converted into electrical signal) and sent to the nearest toward in America from the user then again it convert the electrical signal into packets and the data is recived by the recivers.



Routers; Router is a network device that forwards data packets between computer networks. One or more devices can be connected to the routers.

"Routers have the MAC-Address of all the Devices connected to it and based on the mac-address it transmit data or response to their requests".

Server : A server is a computer program or device that provides a service to another computer program and its users, also known as the client.

" Server is Basically a computer device that is programmed (in any language) to act as a server. "

example; http server, express server. etc.

Server is follows the client-server Architecture ;

↳ client made a request and based on that request server provider or gives the response.

---

http or https : Since http is hyper text transfer protocol whereas https is hyper text transfer protocol secure ;

Express JS:

Express is a lightweight and flexible routing framework with minimal core features that can access through the use of Express middleware modules;

"Basically, Express is framework of NodeJs which is used for routing";

↳ Routing:

The process of choosing a path across one or more networks is known as network routing;

www.facebook.com/profile;

Here /Home,
/About are routings.

* There are various types of routing i.e.;
  - PUT, PATCH, Delete, POST, GET. etc.

* We can create a express server or server with the help of express;

// code:

```
const express = require('express');
const app = express();

app.get('/', (res, req) => {
    res.send("Hello From Home");
});
app.listen(3000);
```

Template engines: are used when you rapidly build web applications that are split into different components.
for example: Website components such as; Body, Navigation, footer, dashboard etc.

↳ Popular Template engines are: Ejs, Jade, Pug, Mustache, Blade etc.

EJS: is Embedded Javascript Templating is a Template engine used by NodeJS. It helps to create an HTML template with minimal code. It injects data into HTML template on the client side.

"We can say that it is HTML that is dynamic" by using;

<%= Username %>

Since we can display the real username...

↳ Steps for setup EJS:

(i) EJS install : npm i ejs.

(ii) Configure ejs :

app.set("view engine", "ejs");

(iii) Make a folder name "views" and make ejs files in it.

(iv) Replace send to render;

app.send ⟶ app.render('index');

↳ Static files in EJS: These files are the images, stylesheets, Frontend JS etc.

•Setup Static files; i) create a folder called public.
ii) create 3 folder in that folder as; images, stylesheets, Javascripts.

(iii) Configure the express static : app.use(express.static('./public'));
(iv) understand the path.

Error Handling; After the last route;

```
app.use(function errorHandler(err, req, res, next) {
    if (res.headersSent) {
        return next(err)
    }
    res.status(500)
    res.render('error')
})
app.listen(3000);
```

for throwing error;

```
throw Error("404");
```

Express Generator: is used to create all the folder and files with ejs so that it makes our task easy. i.e., for setting up ejs;

↳ Steps to use Express Generator;

(i) First install Express Globally as;

```
npm i express-generator -g
```

// for first time only.

(ii) Create new express app;

```
express appName --view=ejs
```

(iii) then follow the cmd shown on screen;

- cd appName,
- npm i,

} for open this in Vs code use; code .

Cokkies and Session ;

(localHost) : All are in same Laptop

Client	Server
(Browser)	(node Js)
↓	↓
Cok Cookie	Session.

* If we have to save any data from Browser (Client) to Server then it is called as use Session. and if we have to save any data from Server to client (Browser or FrontEnd) that then that is called as cookies.

∴ Cokies or Session are the data that is saves on Server or Client side ...

Express-Session : Express-session is used to save session on the server-side ;

For Setting-up Express-session ;

(i) npm i express-session

(ii) Config C-S ;

.app.js file :

```
# requier first;

app.use(session({
        resave: false,
        saveUninitialized: false,
        secret: "hey",
    })
);
```

(iii) create session or make ;

(a) create;

```
router.get('/', (req, res) => {
    req.session.ban = true;
    log(req.session);
});
```

(b) delete;

```
router.get('remove', (req, res) => {
    req.session.destroy(err => {
        if (err) throw err;
        res.send("Ban removed");
    });
});
```

#Cokie-Parser : Package for setting cookie; i.e., it is already install with express-generator and config also; used for setting cookies.

(i) use:

```
res.cookie("age", 22);
log(req.cookies);
```

(ii) delete;

```
res.clearCookie("age");
```

MONGODB.

#What is MongoDB?

MongoDB is an open-source, document-oriented NoSQL database management system.

i.e., it is database that stores information in the form of documents (in the form of JSON document).

It was created by a company called 10gen, which is now known as MongoDB, Inc. The company was founded by Eliot Horowitz and Merriman in 2007. The first version of MongoDB was released in 2009.

↳ Difference b/w SQL and MongoDB;

SQL	MongoDB
• SQL database are relational databases i.e., in the form of Table.	• NoSQL database are non-relational DB that is they are in the form of JSON document.
• They use structured tables to stores data in rows & column	• They provide flexibility in datastorage, allowing varied data types and structures.
• Suitable for applications with well-defined schemas and fixed data-structured.	• Ideal for applications with dynamic or evolving data models.
• E-commerce Platform, ~~Postgre~~ HR Management etc.	• CMS, Social Media Platforms.
• Examples: MySQL, PostgreSQL, Oracle.	• Examples: MongoDB; Cassandra

* what is JSON?

- JSON is JavaScript Object Notation is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects that contains of attribute-value pairs and arrays or key-value pairs.

```
[
  {
  "id": "1",
  "Name": "Princi"
  },
  {
  "id": "2",
  "Name": "Halua",
  "age": "22"
  }
]
```

---

* DB formation on Both sides;

CODE side	MongoDB side
DB setup	DB formation
Model	Collection
Schema	Documents.

DB → can be Amazon DB.

Models → collection of users, products, sales etc.

Schema → is the documents i.e., User;

Name : Prince,
email : 71-
Password : 71-
etc

MongoDB connection Default Code ;

- For Database/db.js file ;

```
Const mongoose = require ('mongoose');
mongoose.connect ('mongodb://localhost/MDB1').then (()=>
    {
     console.log ("Database connected");
    }).catch (err => {
       console.log ("Not Connected", err);
     });

module.exports = mongoose ;
```

// With • this code MongoDB is connected or a Database is created to the local or Atlas.

(i) Mongoose.connect : Database Created.
(ii) Mongoose.Schema : Document's data ;i.e.; name, age etc.
(iii) mongoose.model : Collections created. i.e.; User's collections, Product's etc. etc.

- For routes/users.js file or can any file ;

```
const mongoose = require ("path of db.js");
const userSchema = mongoose.Schema ({
    username : String,
    Password : String,    // and may more
});
```

module.exports = mongoose.models("user", userSchema);

∴ mongoose.models("user", userSchema);

↗ This is the collection name

↖ This is the Schema name.

Now, we have to require it in index.js file;

```
Const userModel = require("./users");
```

// Now we have to perform CRUD operation.

(i) For creating; (object)

```
router.get("/create", async function (req, res){
    Const userDone = await userModel.create({
        username: "Prince",
// Password: "1234",
    });
    res.send(userDone);
    console.log(userDone);
});
```

can be anything

(ii) For find; (Array)

```
router.get("/find", async function (req, res){
    const allUser = await userModel.find();
    res.send(allUser);
    console.log(allUser);
});
```

* There is also a findOne method i.e.;

```
const allUser = await userModel.findOne({
                                    username: "Prince"
                                });
// So it returns the array of the user
// whose username is Prince.
// If no user with the given username is present
// in the DB it will return null (Blank Json file).
```

(iii) For delete a users:

```
Router.get("/delete", async (req, res) => {
    const delUser = await userModel.findOneAnd
                                    Delete({
        name: "Prince"
    });
    res.send(delUser);
    console.log(delUser);
});
// If no user with the given username then null
or if 2 or more user with same name or
username so it will delete one (For First).
```

---

11. Object Orient Programming.

* Introduction: As the name suggests, object-oriented Programming or OOPs refers to languages that use objects in programming. OOPs aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The aim of OOPs is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

↳ Concept of OOPs;

- Class
- Objects
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing.

• Class: is a user-defined data type. It consists of data members and members functions, which can be accessed and used by creating an instance of that class. It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.

Eg: Consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels Speed limit etc. So here, Cars is the class, and wheels, Speed limit are their Properties.

• Object: is a basic unit of Object-Oriented Programming and represents the real life entities. An object is an Instances of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Eg: "Dog" is a real-life Object, which has some characteristics like color, Breed, Bark and Eats.

- Data Abstraction: is one of the most essential and important feature of OOPs. It refers to providing only essential information about the data to the outside world or function or class, and hiding the background details or implementation.

Eg. Consider a Man Driving a car, the man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of car. This is what abstraction is.

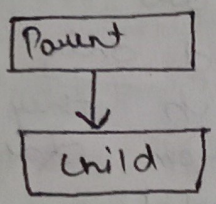
- Encapsulation: is defined as the wrapping up of data under a single unit. It is a mechanism that binds the bind data and code together into a single unit. In this, the variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared. As In this, the data is hidden from other classes, so it is also know as data-hidding.

Eg. Consider a company, there are different sections like the account section, finance section, sale section etc. All these sections have their own work or tasks. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular time. In this case, he is not allowed to directly access the data of the sales section, he first have to contact some other officer in the sales section. This term is know as encapsulation.

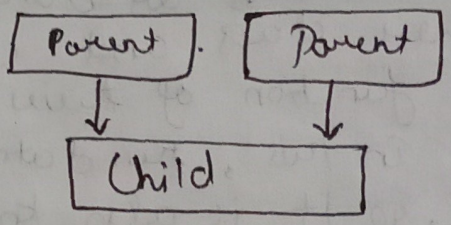
- **Inheritance**: is an important pillar of OOP (Object Oriented Programming). The capability of a class to derive properties and characteristics from another class is called Inheritance.

Inheritance is mechanism wherein a new class is derived from an existing class. In Programming classes may inherit or acquire the properties and methods of other classes. When we write a class, we inherit properties from other classes. So when we create a class, we do not need to write all the properties and functions again and again.

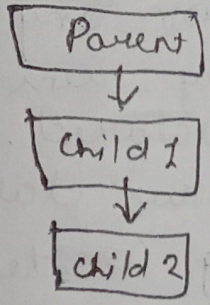
Since, a class derived from another class is called a subclass, whereas the class from which a subclass is derived is called a superclass.



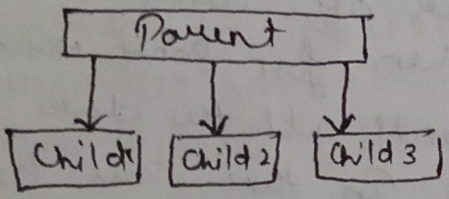
Single Inheritance



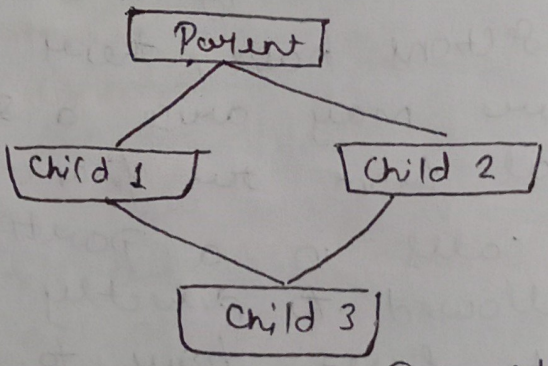
Multiple Inheritance



Multilevel Inheritance



Hierarchical Inheritance



Hybrid Inheritance.

There are some of the types of Inheritance.

In JS, `---proto--` is used to get the properties of Parent object. i.e., called Prototype.

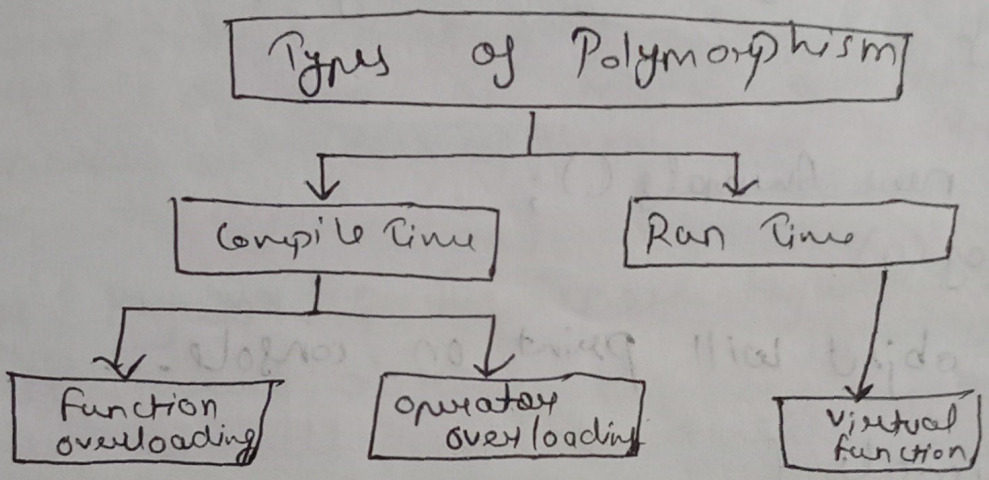
• Polymorphism :

The word polymorphism means having many forms. In simple words, we can define this as the ability ~~to be displayed~~ of a message to be displayed in more than one form.

Since it is the ability of any data to be processed in more than one form. (i.e., Poly means many and morphism means types)

Eg. A person at the same time can have different characteristic, like a man at the same time is a father, a husband, an employee etc.

This is called Polymorphism.



Syntax for creating a class in JS;

i)
```
class Animals {
    constructor() {
      console.log("Object is created");
    }
    eats(){
      console.log("I am eating");
    }
    jumps(){
      console.log("I am jumping");
    }
}
let a = new Animals();
console.log(a);
```

// Animals object will print on console...

(ii)
```
class Animals {
    constructor(name){
      this.name = name
      console.log("object");
    }
    eats(){
      clog("Done 1");
    }
    jumps(){
      eclog("Done 2");
    }
}
class Lion extends Animals {
    constructor(name){
      super(name)
      clog("Lion")
```


THANK YOU!