# Introduction to React :

## 1. HTML, CSS, JavaScript — why React comes in?

•HTML, CSS, JS → are the core web technologies to build any webpage.

•But when a website becomes large and dynamic, manually updating HTML with JS becomes hard to manage.

•React is a JavaScript library that helps us build dynamic and reusable UI components easily.

**Why we use react in webpages :**

- Reusable components (e.g., Navbar, Button, Cards).
- Fast updates using Virtual DOM (updates only changed parts).
- Better performance.
- Easier state management.
- One-page applications (SPA) without reloading the page.

## 2. Difference between HTML Webpage and React Webpage ?

| FEATURE | HTML WEBPAGE | REACT WEBPAGE |
|---|---|---|
| **STRUCTURE** | Built manually with HTML tags | Built with reusable components |
| **DOM Handling** | Directly manipulates the DOM | Uses Virtual DOM for faster updates |
| **Performance** | Slower in large projects | Faster because of Virtual DOM |
| **Reusability** | Code repetition common | Components are reusable |
| **Data Flow** | Hard to handle dynamic data | Uses State and Props for data flow |
| **Scalability** | Difficult for big apps | Easy to scale for complex UIs |

**3. Difference between Angular and React ?**

| **FEATURE** | **REACT JS** | **ANGULAR JS** |
|---|---|---|
| **TYPE** | JavaScript Library | Complete Framework |
| **LANGUAGE** | JavaScript (optionally TypeScript) | TypeScript |
| **LEARNING CURVE** | Easier | Steeper |
| **DOM** | Virtual DOM | Real DOM |
| **DATA BINDING** | One-way | Two-way |
| **SIZE** | Lightweight | Heavy |
| **FLEXIBILITY** | You Choose extra tools | Everything is built-in |

**4. What is TypeScript**?

TypeScript is a programming language developed by Microsoft that adds type checking to JavaScript. It helps find errors early, makes code more structured and reliable, and is especially useful for large projects, while still running as normal JavaScript in the browser.

**Why TypeScript ?**

- Helps find errors early before running the program.
- Makes code easier to understand and maintain.
- Supports object-oriented programming features.
- Works well for large projects with many developers.
- Provides auto-completion and better debugging in editors like VS Code.

**5. What is Variables , Declaration , Initialization , Lexical Scope , Block Scope , Function Scope , Hoisting , Closures ?**

**Variables :**

A name used to store data

– Example :

```
let name = "Varun";

let age = 21;

console.log(name, age); //   Output: Varun 21
```

**Declaration :**

Telling JS that a variable exists

– Example :

```
let city;                          // Declaration
city = "Chennai";          // Initialization
console.log(city);        // Output: Chennai
```

**Initialization :**

Giving a variable a value.

– Example :

```
let language = "JavaScript";      // Declaration + Initialization
console.log(language);            // Output: JavaScript
```

**Lexical Scope :**

Scope determined by where code is written (nested functions can access outer variables)

– **Example :**

```
function outer() {
  let outerVar = "I am from outer function";
 function inner() {
    console.log(outerVar); //   Correct
  }
  inner();
}
outer();
                  // Output: I am from outer function
```

**Block Scope :**

Variables declared inside { } (like let or const) can’t be used outside

– Example :

```
{
  let a = 10;
  const b = 20;
  console.log(a, b);        //  Correct
}
console.log(a, b);        // Error: a and b are not defined
```

**Function Scope :**

Variables declared with var are visible throughout the function

– **Example :**

```
function test() {
  var x = 100;
  console.log(x);     // Correct
}
test();
console.log(x); //  Error: x is not defined
```

**Hoisting :**

JS moves declarations to the top of their scope

– **Example :**

```
console.log(num);     // Output: undefined
var num = 10;
```

```
// Function hoisting
greet(); // Output: Hello!
function greet() {
  console.log("Hello!");
}
```

**Closures :**

Function remembers variables from its outer scope even after the outer function has finished.

– **Example :**

```
function counter() {
  let count = 0;
  return function() {
    count++;
    console.log(count);
  };
}

const increment = counter();
increment();     // Output: 1
increment();    // Output: 2
increment();    // Output: 3
```

**6. In Web-development process why node is coming? and Why?**

Node.js is not a language — it's a runtime environment that lets you run JavaScript outside the browser, mainly on the server side.

**Why?**

It is fast, handles multiple requests efficiently, and has a large package library (npm) that helps in creating APIs and web applications quickly.

**7. Which one came first node or angular and why?**

- AngularJS was released by Google in 2010 as a frontend framework to build dynamic web pages.
- Node.js was released by Ryan Dahl in 2009 as a backend runtime environment for running JavaScript on the server.

So technically, **Node.js came first (2009)**, and **Angular** came later (2010).

**Why ?**

Node.js was created first to make JavaScript work on the server side, and after that, Angular was developed to make JavaScript more powerful and structured on the client side (frontend).

**8. CRA Project - Create Project -List of Files and Folders :**

**=> node-module :**

.bin , .cache , @adobe

⇨ **Public :**

- favicon.ico
- index.html
- logo192.png
- logo512.png
- manifest.json
- robots.txt

**=> SRC :**

⇨ app.cs
⇨ app.js
⇨ App test.js

- ⇨ index.css
- ⇨ index.js
- ⇨ logo.svg
- ⇨ reportWebVitals.js
- ⇨ setupTests.js
- ⇨ gitignore
- ⇨ package-lock.json
- ⇨ package.json
- ⇨ README.md

**VITE Project - Create Project -List of Files and Folders :**

- ⇨ .bin ,
- ⇨ .vite ,
- ⇨ .vite-temp ,
- ⇨ @babel

- ⇨ **Public**
- ⇨ vite.svg

- ⇨ **SRC :**

- ⇨ Assests :
- ⇨ react.svg
- ⇨ app.css
- ⇨ App.jsx
- ⇨ index.css
- ⇨ main.jsx
- ⇨ gitignore
- ⇨ eslint.config.js
- ⇨ index.html
- ⇨ package-lock.json
- ⇨ package.json
- ⇨ README.md
- ⇨ vite.config.js