# Unit-6 React.js-1

## Introduction

### What is React?

React, sometimes referred to as a frontend JavaScript framework, is a JavaScript library created by Facebook.

React is a tool for building UI components.

### How does React Work?

React creates a VIRTUAL DOM in memory.

Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM.

React only changes what needs to be changed!

React finds out what changes have been made, and changes **only** what needs to be changed.

### React.JS History

Current version of React.JS is V18.0.0 (April 2022).

Initial Release to the Public (V0.3.0) was in July 2013.

React.JS was first used in 2011 for Facebook's Newsfeed feature.

Facebook Software Engineer, Jordan Walke, created it.

Current version of `create-react-app` is v5.0.1 (April 2022).

### Why we use ReactJS?

The main objective of ReactJS is to develop User Interfaces (UI) that improves the speed of the apps. It uses virtual DOM (JavaScript object), which improves the performance of the app. The JavaScript virtual DOM is faster than the regular DOM. We can use ReactJS on the client and server-side as well as with other frameworks. It uses component and data patterns that improve readability and helps to maintain larger apps.

### React Environment Setup

There are two ways to set up an environment for successful ReactJS application. They are given below.

# Unit-6 React.js-1

Using the npm command

Using the create-react-app command

**1. Using the npm command**

**Install NodeJS and NPM**

NodeJS and NPM are the platforms need to develop any ReactJS application. You can install NodeJS and NPM package manager.

After that Run the below command in your terminal or command prompt to install the React.

```
npm install -g create-react-app
```

**2. Using the create-react-app command**

```
npx create-react-app myApp
```

it will create one react app name 'myApp'. After creating app

```
cd myApp
npm start
```

npm start will start react server on browser **localhost:3000** and run your react app.

## React Render HTML

React renders HTML to the web page by using a function called `createRoot()` and its method `render()`.

### The createRoot Function

The `createRoot()` function takes one argument, an HTML element.

The purpose of the function is to define the HTML element where a React component should be displayed.

### The render Method

The `render()` method is then called to define the React component that should be rendered.

But render where?

# Unit-6 React.js-1

There is another folder in the root directory of your React project, named "public". In this folder, there is an `index.html` file.

### Example

Display a paragraph inside an element with the id of "root":

```
const container = document.getElementById('root');

const root = ReactDOM.createRoot(container);

root.render(<p>Hello</p>);
```

The result is displayed in the `<div id="root">` element:

```
<body>

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

</body>
```

## React JSX

### What is JSX?

JSX stands for JavaScript XML.

JSX allows us to write HTML in React.

JSX makes it easier to write and add HTML in React.

Using JSX is not compulsory but it is highly recommended for programming in React as it makes the development process easier as the code becomes easy to write and read.

JSX creates an element in React that gets rendered in the UI. It is transformed into JavaScript functions by the compiler at runtime.

Error handling and warnings become easier to handle when using JSX

**sample JSX code:**

```
const ele = <h1>This is sample JSX</h1>;
```

The above code  looks like HTML and it also uses a JavaScript-like variable but is neither HTML nor JavaScript, it is JSX(JavaScript XML). With the help of JSX, we have directly written the HTML syntax in JavaScript

# Unit-6 React.js-1

## Why JSX?

- It is faster than normal JavaScript as it performs optimizations while translating to regular JavaScript.
- It makes it easier for us to create templates.
- Instead of separating the markup and logic in separate files, React uses *components* for this purpose. We will learn about components in detail in further articles.
- As JSX is an expression, we can use it inside of if statements and for loops, assign it to variables, accept it as arguments, or return it from functions.

## Expressions in JSX

With JSX you can write expressions inside curly braces { }.

The expression can be a React variable, or property, or any other valid JavaScript expression.

JSX will execute the expression and return the result:

**Example** Execute the expression `5 + 5`:

```
const myElement = <h1>React is {5 + 5} times better with JSX</h1>;
```

we declare a variable called `name` and then use it inside JSX by wrapping it in curly braces:

```
const name = 'Josh Perez';

const element = <h1>Hello, {name}</h1>;
```

### Passing expression

If you want to dynamically specify the src or alt text in img tag. You could use a value from JavaScript by replacing " and " with { and }

```
function Avatar()
 {
                const pic = ‘img.png';
                const description = 'test image';
        return (
       <img  className="pic"  src={pic}  alt={description} />
);
}
export default Avatar;
```

## How to Apply style in JSX?

There are many ways to style React with CSS, this tutorial will take a closer look at **inline styling**, and **CSS stylesheet**.

### Inline Styling

To style an element with the inline style attribute, the value must be a JavaScript object:

```
import React from 'react';
import ReactDOM from 'react-dom';

const Header = () => {

  return (
    <>
      <h1 style={{color: "red"}}>Hello Style!</h1>
    </>
  );
}
```

Since the inline CSS is written in a JavaScript object, properties with two names, like `background-color`, must be written with camel case syntax

Use `backgroundColor` instead of `background-color`:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const Header = () => {
  return (
    <>
      <h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
}
```

You can also create an object with styling information, and refer to it in the style attribute:

```
import React from 'react';
import ReactDOM from 'react-dom/client';

const Header = () => {
  const myStyle = {
    color: "white",
    backgroundColor: "DodgerBlue",
    padding: "10px",
    fontFamily: "Sans-Serif"
  };
  return (
    <>
      <h1 style={myStyle}>Hello Style!</h1>
      <p>Add a little style!</p>
    </>
  );
}
```

You can write your CSS styling in a separate file, just save the file with the `.css` file extension, and import it in your application.

**App.css:**

Create a new file called "App.css" and insert some CSS code in it:

```
p {
  background-color: #282c34;
  color: white;
  padding: 40px;
  font-family: Arial;
  text-align: center;
}
```

Import the stylesheet in your application:
**App.js**

```
import React from 'react';
import ReactDOM from 'react-dom';
import './App.css';

const App = () => {

    return (
      <div>
      <h1>Hello Style!</h1>
      <p>Add a little style!.</p>
      </div>
    );
  }
```

## JSX Comments

To write comments in React (JSX), we need to wrap **them in curly braces**.

```
Function comment() {
Return(
{/* this works */ }
)
}
```

# Unit-6 React.js-1

## Attribute class = className

The class attribute is a much used attribute in HTML, but since JSX is rendered as JavaScript, and the class keyword is a reserved word in JavaScript, you are not allowed to use it in JSX.

Use attribute className instead.

## Example

Use attribute className instead of class in JSX:

```
const myElement = <h1 className="myclass">Hello World</h1>;
```

## React Components

Components are like functions that return HTML elements.

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.

React components are regular JavaScript functions except:
o Their names always begin with a capital letter.
o They return JSX markup.

### Function Component

### Example

Create a Function component called Car

Car.js

```
function Car() {

  return <h2>Hi, I am a Car!</h2>;

}

Export default Car;
```

### Rendering a Component

Now your React application has a component called Car, which returns an <h2> element.

To use this component in your application, use similar syntax as normal HTML: `<Car/>`

## Example

Display the `Car` component in the "root" element:

**Index.js**

```
import Car from './Car'

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<React.StrictMode><Car /><React.StrictMode>);
```

**Program:**
Build basic react app that display “Hello World” in browser.
Here file named ex1.js is a component and import this component in App.js file.

**Ex1.js**

```
import React from 'react';
function Ex1() {
return(
<div>
<h1>Hello World!</h1>
</div>
)
}
export default Ex1;
```

**App.js**

```
import Ex1 from "./Ex1";
function App() {
return (
<div>
<Ex1/>
</div>
);
}
export default App;
```

## React Props

Props stand for "Properties."
It is an object which stores the value of attributes of a tag and work similar to the HTML attributes.
React components use props to communicate with each other.
Each component can pass some information to another components by giving them props. Props are similar to function arguments.
Props are passed to the component in the same way as arguments passed in a function.

**Example:**
**Step 1: Pass props to the component**
**App.js**

```
import Ex2 from "./Ex2";
function App() {
var n = “ABC”;
return (
<div>
<Ex2 name={n} rollnum="101" marks="20" />
<Ex2 name="DEF" rollnum="102" marks="16" />
<Ex2 name="GHI" rollnum="103" marks="22.5" />
</div>
);
}
export default App;
```

**Step 2: Read props inside the component**
**Ex2.js**

```
import React from 'react';
const Ex2 = (props)=>{
return(
<div>
<ul>
<li>{props.name}</li>
<li>{props.rollnum}</li>
<li>{props.marks}</li>
</ul>
</div>
);
}
export default Ex2;
```

# Unit-6 React.js-1

## Program:

Product Name and its price are declared as props in **Example.js** file and the value of props passed in **app.js** file. A variable can also be passed as props value.

### app.js :

```
import Example from "./Example";
function App() {
const xyz="Computer";
return (
<div>
<Example Name={xyz} Price="70000" />
<Example Name="Mobile" Price="20000" />
</div>
);
}
export default App;
```

### Example.js :

```
import React from 'react'
const Example = (props)=>{
return(
<div>
<h1>{props.Name} : {props.Price}</h1>
</div>
);
}
export default Example;
```

## Program:

Write a React code to print car's brand name and its model name which are passed as props using JSON.

### Example.js :

```
import React from 'react'
function Example(props) {
return(
<h2>I am a { props.brand.model } of { props.brand.name }!</h2>
);
}
export default Example;
```

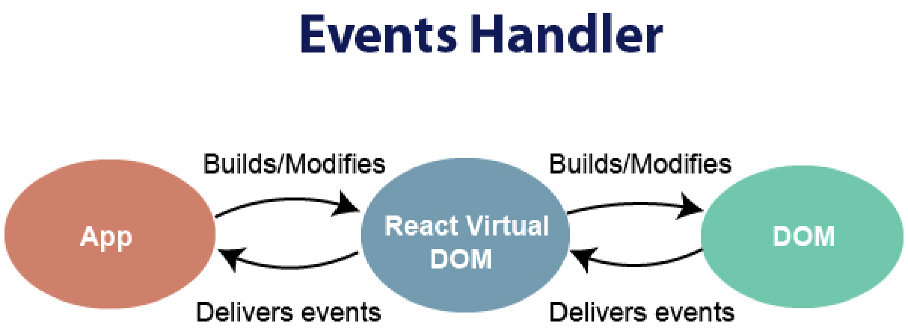
### App.js :

```
import Example from "./Example";
function App() {
const carInfo = { name: "Ford", model: "Mustang" };
return (
<div>
<h1>Who lives in my garage?</h1>
<Example brand={ carInfo }/>
</div>
);
}
export default App;
```

## React Events

Just like HTML DOM events, React can perform actions based on user events.

React has its own event handling system which is very similar to handling events on DOM elements. The react event handling system is known as Synthetic Events. The synthetic event is a cross-browser wrapper of the browser's native event.



React has the same events as HTML: click, change, mouseover etc.

React events are written in camelCase syntax:

`onClick` instead of `onclick`.

React event handlers are written inside curly braces:

`onClick={shoot}` instead of `onClick="shoot()"`.

### In HTML:

```
<button onclick="shoot()">Take the Shot!</button>
```

### In React:

```
<button onClick={shoot}>Take the Shot!</button>
```

### Example:

Put the shoot function inside the Football component:

```
function Football() {

  const shoot = () => {

    alert("Great Shot!");

  }

  return (

    <button onClick={shoot}>Take the shot!</button>

  );

}
```

## Passing Arguments

To pass an argument to an event handler, use an arrow function.

### Example:

Send "Goal!" as a parameter to the shoot function, using arrow function:

```
function Football() {

  const shoot = (a) => {

    alert(a);

  }

  return (

    <button onClick={() => shoot("Goal!")}>Take the shot!</button>

  );

}
```

In react, we cannot return false to prevent the default behaviour. We must call preventDefault event explicitly to prevent the default behavior.

For example:

In plain HTML, to prevent the default link behaviour of opening a new page, we can write:

```
<a href="#" onclick="console.log('You had clicked a Link.'); return false">

    Click_Me

</a>
```

In React, we can write it as:

```
function ActionLink() {

    function handleClick(e) {

        e.preventDefault();

        console.log('You had clicked a Link.');

    }

    return (

        <a href="#" onClick={handleClick}>

              Click_Me

        </a>

    );

}
```

## onclick event

Write react js script to display alert box with text “Welcome to LJU” by clicking on button.

**Event1.js**

```
import React from 'react';
function Event1() {
const mystyle = {
color : "white",
backgroundColor: "#000000",
padding: "10px 20px",
margin: "200px"
};
Function handleClick () {
alert ('Welcome to LJU');
}
return (
<div>
<center>
<button style = {mystyle} onClick={handleClick}>
Click me
</button>
</center>
</div>
);
}

export default Event1;
```

**App.js**

```
import Event1 from "./Event1"
function App() {
return (
<div>
<Event1/>
</div>
);
}
export default App;
```

## onchange event

**Write react js script to display values in console while changing it in text box.**
**Event2.js**

```
import React from 'react';
function Event2() {
function handleChange(event) {
console.log (event.target.value);
}
return (
<input type=”text” name="firstName" onChange={handleChange} />
);
}
export default Event2;
```

**App.js**

```
import Event2 from "./Event2"
function App() {
return (
<div>
<Event2/>
</div>
);
}
export default App;
```

## onDoubleClick event

**Write react js script to display alert box with text "welcome to lju" only on double click.**

**Event3.js**

```
import React from 'react';
function Event3() {
const mystyle = {
color : "white",
backgroundColor : "#000000",
padding: "10px 20px",
margin:"30px"
};
const doubleClickHandler = (event) => {
alert("Welcome to LJU");
}
return (
<>
<button style={mystyle} onDoubleClick = {doubleClickHandler}>Double Click
here</button>
</>
);
}
export default Event3;
```

**App.js**

```
import Event3 from "./Event3"
function App() {
return (
<div>
<Event3/>
</div>
);
}
export default App;
```

## onsubmit

**Write react js script to display alert box with text "' You clicked submit" only on form submit**

**Form.js**

```
import React from 'react';
const Form= ()=> {
function handleSubmit (e) {
e.preventDefault ();
alert (' You clicked submit.');
}
return (
<form onSubmit = {handleSubmit}>
<button type="submit">Submit</button>
</form>
);
}

export default Form;
```

**App.js**

```
import Event3 from "./Event3"
function App() {
return (
<div>
<Form/>
</div>
);
}
export default App;
```

**preventDefault** is used to prevent the default form behaviour of submitting

## React Map

The maps are used for traversing or displaying the list of similar objects of a component.

A new array is made using the map() method, and a function is called on each element of the array.

### Syntax

```
array.map(callback[, thisObject]);
```

### Parameter Details

- **callback** − Function that produces an element of the new Array from an element of the current one.
- **thisObject** − Object to use as **this** when executing callback.

### Return Value

Returns the created array.

**Example:- Arraymap.js**

```
import React from 'react'

const Arraymap = () => {
   const arr=[1,2,3,4,5];
  return (
    <div>
        <h1>using arraymap function</h1>
        {  arr.map((value)=>
             {
           return <h1>array values= {value}</h1>

              })
        }
    </div>
  )
}

export default Arraymap
```

# Unit-6 React.js-1

**Program:**
Write a program to create ReactJS application having an array of strings and convert it in Uppercase using MAP method.
**Arraymap.js :**

```
import React from 'react'
const Arraymap = () => {
const arr=["a","b","c","d","e"];
return ( <div>
            <h1>map function</h1>
            { arr.map((value)=>
                {
                    return <p>array values= {value.toUpperCase()}</p>
                 })
            }
        </div>
      )
 }
export default Arraymap
```

**Program:**
We have an array of numbers and we want to multiply each of these numbers by 5. Create react js app to display these multiplied numbers using map function.
**Map1.js**

```
import React from 'react';
function Map1() {
let arr = [2, 4, 6, 3, 10, 12]
return (
<div>
<h1>Multiplication of numbers are as under: </h1>
{ arr.map((value)=>
{
return <h2>{value} * 5 = {value * 5}</h2>
})
}
</div>
)
}
export default Map1
```

# Unit-6 React.js-1

**Program:** Create react app which displays images using map function.

**Map2.js**

```
import React from 'react';
import img1 from "./img1.jpg"
import img2 from "./img2.jpg"
import img3 from "./img3.jpg"
import img4 from "./img4.jpg"
function Map2() {
const images=[img1,img2,img3,img4]
        return (
            <div>
              {    images.map((val) => {
                     return
                     <img src={val.pic} heigth="200px" width="200px" alt="logo" />
                    })
              }
            </div>
        )
}
export default Map2
```

## React Filter

It's the process of looping through an array and including or excluding elements inside that array based on a condition that you provide.

we have apply filter to skip digit “3” from the array and display all remaining digits of the array.

**Example:-   Arrayfilter.js**

```
import React from 'react'

const Arrayfilter = () => {
    const arr=[1,2,3,4,5];
    const newarr=arr.filter((num)=>
    {
        if(num==3)
        {return false;}
        else
        return true;
    }
    );
    return (
    <div>
```

```
      <h1>Array before filter {arr}</h1>
      <h1>Array after filter {newarr}</h1>

    </div>
  )
}

export default Arrayfilter
```

## React Routing

React Router is a standard library for routing in React. It enables navigation between views from different components in a React application, allows the browser URL to be changed, and keeps the UI in sync with the URL.
To understand how React Router works, let's create a simple application for React. The application will include the home, about, and contact components. We will be using React Router to navigate between these components.
Routing can be server-side routing or client-side routing. However, React Router is a part of client-side routing.
There are 3 different packages for React Routing.
1. **react-router:** It is the heart of react-router and provides core functionalities for routing in web applications and all the packages for react-router- dom and react-router-native

2. **react-router-native**: It is used for routing in mobile applications

3. **react-router-dom:** It is used to implement dynamic routing in web applications.

**Router:** A <router> that uses the hash part of the URL (i.e., window.location.hash) to keep your UI in sync with the URL.

**Routes:** To read a single component, wrap all routes inside the Routes component.

Switch groups across multiple routes, iterate over them and find the first one that matches the path. Thus, the corresponding component of the path is rendered.

**Route:** The route component will help us establish the link between the component's UI and the URL. To include the route in the application, add the below code to your app.js

**Add React Router**
To add React Router in your application, run below command in the terminal from the root directory of the application:

# Unit-6 React.js-1

```
npm i -D react-router-dom
```

**Folder Structure**

To create an application with multiple page routes, let's first start with the file structure.

Within the **src** folder, we'll create a folder named **components** with several files:

**src\ components **

- Home.js
- Product.js
- Contact.js
- Navbar.js

Each file will contain a very basic React component.

## App.js

```
import './App.css';
import Navbar from'./components/Navbar';
function App() {
    return (
          <div>
             <Navbar/>
           </div>
        );
}
export default App
```

**( Note:  to use bootstrap  , copy link of bootstrap 5 and paste it in index.html in <head>)**

**Navbar.js**

```
import React from 'react'
import {BrowserRouter as Router,Route,Routes,Link} from "react-router-dom"
import Home from "./Home"
import Contact from "./Contact"
import Shop from "./Shop"
const Navbar = () => {

    return (
   <Router>
     <div>
    <nav className="navbar navbar-inverse">
       <div className="container-fluid">
           <ul className="nav navbar-nav">
              <li className="active"><Link to="/">Home</Link></li>
              <li><Link to="/Shop">Shop</Link></li>
              <li><Link to="/Contact">Contact</Link></li>
         </ul>
     </div>
</nav>
  </div>

    <Routes>
        <Route path="/" element={<Home/>}/>
        <Route path="Shop" element={<Shop/>}/>
        <Route path="Contact" element={<Contact/>}/>
    </Routes>
    </Router>
  )
}
export default Navbar
```

**1. Home.js (**`<Route exact path="/" element={<Home/>}/>)`

```
import React from 'react'
const Home = () =>{
         return (
               <div>
               <h1 >Mobile shopping Website</h1>
               </div>
            )
      }
export default Home
```

**2. Contact.js**

```
import React from 'react'
const Contact = () => {
return (
           <div>
             <h1>Contact Detail</h1>
              <h3> DM me For Order</h3>
           </div>
        )
     }
export default Contact
```

**3. Product.js**

```
import React from 'react'
import '../App.css';
const Product = () => {
return (
          <div>
           <h1>This is product page</h1>
          </div>
      )
}
export default Product
```