React js

React is a JavaScript library for building user interfaces.

What is React?

React is a JavaScript front-end library.
It is used for building user interfaces or reusable UI components.
It is created and maintained by Facebook.
React is specifically used in the development of single-page applications or mobile applications.

So React is a JavaScript library for building user interfaces.

React Libraries:

React and React DOM: Both are libraries. react is used for the components and react-dom is for rendering the components in the DOM. ... The react package holds the react source for components, state, props and all the code that is react. The react-dom package as the name implies is the glue between React and the DOM. It will take care "How to render/repaint components in DOM".
Webpack: Webpack is used for module packaging, development, and production pipeline automation.
Babel: Babel is a JavaScript compiler and transpiler used to convert one source code to others. It compiles React JSX and ES6 to ES5 JavaScript which can be run on all browsers. We need babel-loader for JSX file types, babel-preset-react (plugin) makes your browser update automatically when any changes occur to your code without losing the current state of the app.

React components:

Earlier, the developers write more than thousands of lines of code for developing a single page application. These applications follow the traditional DOM structure, and making changes in them was a very challenging task. If any mistake found, it manually searches the entire application and update accordingly. The component-based approach was introduced to overcome an issue. In this approach, the entire application is divided into a small logical group of code, which is known as components.

A Component is considered as the core building blocks of a React application. It makes the task of building UIs much easier. Each component exists in the same space, but they work independently from one another and merge all in a parent component, which will be the final UI of your application.

In ReactJS, we have mainly two types of components. They are:

1) Function Components
2) Class Components

Functional Components:

- The simplest way to define a component by writing a JavaScript function.
- A functional component is basically a JavaScript function that returns a React element (JSX).

Example:

App.js

```
function Welcome() {
  return (
    <h1>Hello world</h1>
  );
}
export default Welcome;
```

index.js

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

import Welcome from './App';
ReactDOM.render(
   <Welcome />,
  document.getElementById('root')
);
```

Class Components:

- Class components are more complex than functional components.
- Create a class defining class keyword which extends React.Component class.
- Now create a render function inside class which returns a React element.
- You can pass data from one class to other class components.

Example:

App.js

```
import React from 'react';

class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
export default Welcome;
```

index.js

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

import Welcome from './App';


ReactDOM.render(
    <Welcome name="UVPCE"/>,
  document.getElementById('root')
);
```

Note: class component is also known as a stateful component because they can hold or manage local state.

React State:

React components has a built-in state object.

The state object is used to store a property value that belongs to the component.

When the state object changes, the component re-renders.

A component with the state is known as stateful components. It is the heart of the react component which determines the behavior of the component and how it will render. They are also responsible for making a component dynamic and interactive.

Creating the state Object:

The state object is initialized in the constructor:

[A component gets initiated with constructor() function. The constructor function is where you initiate the component's properties. In React, component properties should be kept in an object called state.

The constructor function is also where you honor the inheritance of the parent component by including the super() statement, which executes the parent component's constructor function, and your component has access to all the functions of the parent component (React.Component).]

Example: Create a constructor function of Student component, and add a branch property:

Specify the state object in the constructor method:

Create state1.js file:

```
import React from 'react';
class Student extends React.Component {
  constructor(props) {
    super(props);
    this.state = {branch: "CE"};
  }
  render() {
    return (
      <div>
        <h1>Welcome to UVPCE </h1>
        He is in {this.state.branch} branch

      </div>
    );
  }
}

export default Student;
```

To define a state, you have to first declare a default set of values for defining the component's initial state. To do this, add a class constructor which assigns an initial state using this.state. The 'this.state' property can be rendered inside render() method.

React Component Life-Cycle

In ReactJS, every component creation process involves various lifecycle methods. These lifecycle methods are termed as component's lifecycle. These lifecycle methods are not very complicated and called at various points during a component's life. The lifecycle of the component is divided into **four phases**. They are:

1. Initial Phase
2. Mounting Phase - Birth of your component
3. Updating Phase - Growth of your component
4. Unmounting Phase - Death of your component

render()
The *render()* method is the most used lifecycle method. You will see it in all React classes. This is because *render()* is the only required method within a class component in React.
As the name suggests it handles the rendering of your component to the UI. It happens during the **mounting** and **updating** of your component.

Each phase contains some lifecycle methods that are specific to the particular phase. Let us discuss each of these phases one by one.

1. Initial Phase

It is the **birth** phase of the lifecycle of a ReactJS component. Here, the component starts its journey on a way to the DOM. In this phase, a component contains the default Props and initial State. These default properties are done in the constructor of a component. The initial phase only occurs once and consists of the following methods.

- **getDefaultProps()**
  It is used to specify the default value of this.props. It is invoked before the creation of the component or any props from the parent is passed into it.
- **getInitialState()**
  It is used to specify the default value of this.state. It is invoked before the creation of the component.

2. Mounting Phase

In this phase, the instance of a component is created and inserted into the DOM. It consists of the following methods.

- **componentWillMount()**
  This is invoked immediately before a component gets rendered into the DOM. In the case, when you call **setState()** inside this method, the component will not **re-render**.
- **componentDidMount()**
  This is invoked immediately after a component gets rendered and placed on the DOM. Now, you can do any DOM querying operations.
- **render()**
  This method is defined in each and every component. It is responsible for returning a single root **HTML node** element. If you don't want to render anything, you can return a **null** or **false** value.

3. Updating Phase

It is the next phase of the lifecycle of a react component. Here, we get new **Props** and change **State**. This phase also allows to handle user interaction and provide communication with the components hierarchy. The main aim of this phase is to ensure that the component is displaying the latest version of itself. Unlike the Birth or Death phase, this phase repeats again and again. This phase consists of the following methods.

- **componentWillRecieveProps()**
  It is invoked when a component receives new props. If you want to update the state in response to prop changes, you should compare this.props and nextProps to perform state transition by using **this.setState()** method.
- **shouldComponentUpdate()**
  It is invoked when a component decides any changes/updation to the DOM. It allows you to control the component's behavior of updating itself. If this method returns true, the component will update. Otherwise, the component will skip the updating.
- **componentWillUpdate()**
  It is invoked just before the component updating occurs. Here, you can't change the component state by invoking **this.setState()** method. It will not be called, if **shouldComponentUpdate()** returns false.
- **render()**
  It is invoked to examine **this.props** and **this.state** and return one of the following types: React elements, Arrays and fragments, Booleans or null, String and Number. If shouldComponentUpdate() returns false, the code inside render() will be invoked again to ensure that the component displays itself properly.
- **componentDidUpdate()**
  It is invoked immediately after the component updating occurs. In this method, you can put any code inside this which you want to execute once the updating occurs. This method is not invoked for the initial render.

4. Unmounting Phase

It is the final phase of the react component lifecycle. It is called when a component instance is **destroyed** and **unmounted** from the DOM. This phase contains only one method and is given below.

- **componentWillUnmount()**
  This method is invoked immediately before a component is destroyed and unmounted permanently. It performs any necessary **cleanup** related task such as invalidating timers, event listener, canceling network requests, or cleaning up DOM elements. If a component instance is unmounted, you cannot mount it again.

React Form:

- Just like in HTML, React uses forms to allow users to interact with the web page to gather information from the users.
- Forms can perform many tasks such as authentication of the user, adding user, searching, filtering, booking, ordering, etc.
- A form can contain text fields, buttons, checkbox, radio button, etc.

Creating Form:

React offers a stateful, reactive approach to build a form.
In React, the form is usually implemented by using controlled components.

Example:1

Create form that allows users to enter their name:

Form_ex1.js

```
import React from 'react';
class Form1 extends React.Component {
   render() {
     return (
       <form>
         <h1>Hello</h1>
         <p>Enter your name:</p>
         <input type="text" />
         <p>Enter Enrollment:</p>
         <input type="number" />
       </form>
     );
   }
 }
 export default Form1;
```

index.js

```
import Form1 from './Form_ex1';
ReactDOM.render(<Form1 />, document.getElementById('root'));
```

Now save file and see output in browser.

Now add Branch in this form using <select> tag:

Create an array of objects to make options for the select input. For this example, create array of departments.
Here Label and value may be different, that's why create array of objects.

```
import React from 'react';
const options = [
  { value: 'CE', label: 'Computer Engineering' },
  { value: 'IT', label: 'Information Technology' },
  { value: 'MC', label: 'Mechatronics' },
];
```

Next, using the options array, create a select tag inside the Form1 component with the different options. Assign the value property from the object to the value prop of the options element, so that the options are mapped correctly and the value for the select element can be retrieved.

```
class Form1 extends React.Component {
   render() {
    return (
     <form>
      <h1>Hello</h1>
      <p>Enter your name:</p>
      <input type="text" />
      <p>Enter Enrollment:</p>
      <input type="number" />
     <p>Enter Department:</p>
      <select>
       {options.map((option) => (
         <option value={option.value}>{option.label}</option>
       ))}
      </select>
     </form>
    );
  }
 }
 export default Form1;
```

In last add button to submit the form.

```
<br/><br/>
<input type="submit" value="Submit" />
```

React Events:

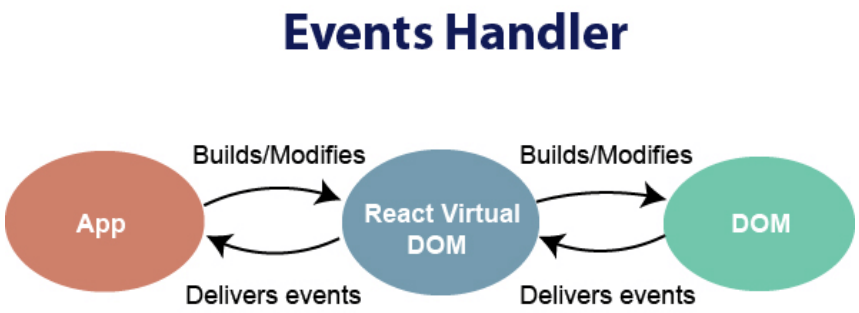
An event is an action that could be triggered as a result of the user action or system generated event. Just like HTML, React can perform actions based on user events.

React has the same events as HTML: For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called events.

What are event handlers in React?

Event handlers determine what action is to be taken whenever an event is fired. This could be a button click or a change in a text input.

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.
So whenever we are triggering an event in a React component, we are not actually dealing with the real DOM event, instead, we are dealing with React's custom event type which is called as synthetic event.



What is the onClick event in React?

The React onClick event handler enables you to call a function and trigger an action when a user clicks an element, such as a button, in your app.

Handling events with react have some syntactic differences from handling events on DOM. These are:

- React events are named as camelCase instead of lowercase.
- With JSX you pass a function as the event handler, rather than a string and React event handlers appear inside curly braces.

- For example:

Event declaration in plain HTML:	Event declaration in React:

<button onclick="showMessage()">
  Click Here
</button>
	<button onClick={showMessage}>
	  Hello JavaTpoint
	</button>

Example 1: Create button and on click of button see date in paragraph.

Open index.html and add paragraph:

Create event_ex1.js file and type following code:
import React from 'react';

```
// Events
class Hello extends React.Component {
   click_me() {
     document.getElementById("demo").innerHTML = Date();
   }
   render() {
     return (
       <button onClick={this.click_me}>See Date</button>
     );
   }
 }
 export default Hello;
```

Passing Arguments to Event Handlers:

If you want to send parameters into an event handler then two options are available.

1) **make an anonymous arrow function:**

Example 2: Create button and on click of button show alert box having greetings message.

```
import React from 'react';

// Events
class Hello extends React.Component {
   click_me=(a) =>{
   alert("Good" +a+"!");
   }
   render() {
     return (
       <button onClick={()=>this.click_me('Morning')}>Click Me</button>
     );
   }
 }
 export default Hello;
```

2) **Bind the event handler to this.**

   If you use regular functions instead of arrow functions you have to bind "this" keyword to the component instance using the bind() method:

Example:2

```
import React from 'react';

// Events
class Hello extends React.Component {
click_me(a,b)
{
  alert("Good "+a+" "+b+"!");
}
   render() {
     return (
```

```
        <button onClick={this.click_me.bind(this,"Evening","Rachana")}>Click Me</button>
    );
  }
 }
 export default Hello;
```

Handling Forms: onChange() event

Handling forms is about how you handle the data when it changes value or gets submitted.
In HTML, form data is usually handled by the DOM.
In React, form data is handled by the components.
When the data is handled by the components, all the data is stored in the component state.
You can control changes by adding event handlers in the **onChange attribute**:

Example-3:
Add an event handler in the onChange attribute, and let the event handler update the state object:

Note: You must initialize the state in the constructor method before you can use it.
Note: You get access to the field value by using the event.target.value syntax.

```
import React from 'react';
class Form2 extends React.Component {


 constructor(props) {
  super(props);
  this.state = { username: '' };    }
 myHandler = (event) => {
     console.log(event.target.value);
  //this.setState({username: event.target.value});
 }
 render() {
  return (
    <form>
    <h1>Hello {this.state.username}</h1>
    <p>Enter your name:</p>
    <input type='text' onChange={this.myHandler} />      </form>
  );
 }
}
export default Form2;
```

Submitting Forms: OnSubmit() event

You can control the submit action by adding an event handler in the onSubmit event:
Onsubmit event fires when from is submitted usually by pressing button.

Example-4:
Add a submit button and an event handler in the onSubmit attribute:

```
import React from 'react';
class Form4 extends React.Component {
 constructor(props) {
  super(props);
  this.state = { username: '' };
 }
 mySubmitHandler = (event) => {
  //event.preventDefault();
  alert(this.state.username);
 }
 myHandler = (event) => {
    this.setState({username: event.target.value});
 }
 render() {
  return (
    <form onSubmit={this.mySubmitHandler}>

    <p>Enter your name:</p>
    <input type='text' name="fname" onChange={this.myHandler} />
    <input type="submit" value="submit"/>
    </form>
  );
 }
}
export default Form4;
```

Submit Multiple Input Fields:

You can control the values of more than one input field by creating different Handlers. Update their state using setState() method by calling from onChange event.

Let's add more fields:

Example:5

```
import React from 'react';
class Form4 extends React.Component {
 constructor(props) {
  super(props);
  this.state = {
    name:'',
    address:'',
    branch:'',
  }
 }
 mynameHandler=(event)=>
 {
  this.setState({
    name:event.target.value
  });
```

```
  //document.getElementById("p1").innerHTML = this.state.name;
 }
 myaddressHandler=(event)=>
 {
  this.setState({
   address:event.target.value
  })
 }
 mybranchHandler=(event)=>
 {
  this.setState({
   branch:event.target.value
  })
 }
 myHandler = (event) => {
  alert("Name:"+this.state.name+"\nAddress:"+this.state.address+"\nBranch:"+this.state.branch);
  //event.prevantDefault();
  //document.getElementById("p1").innerHTML = this.state.name;
 }

 render() {
  return (
   <main>
    <center>
      <h1>Registartion Form</h1></center>
   <form onSubmit={this.myHandler}>
   <p>Enter your first name:</p>
   <input type='text' value={this.state.name} onChange={this.mynameHandler} />
   <p>Enter your address:</p>
   <textarea value={this.state.address} onChange={this.myaddressHandler}/>
   <p>Select Branch:</p>
   <select value={this.state.branch} onChange={this.mybranchHandler}>
    <option>CE</option>
    <option>IT</option>
    <option>ME</option>
    </select><br/><br/>
   <input type='submit'/>
   </form>
   <hr/>
   <p id="p1">Name:{this.state.name}<br/>
   Address:{this.state.address}<br/>
   Branch:{this.state.branch}
   </p>
   </main>
  );
 }
}
export default Form4;
```

Styling React Elements Using CSS:
There are many ways to style React elements with CSS: inline styling and External CSS stylesheet.

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

Example:

```
import React from 'react';

class MyHeader extends React.Component {
   render() {
     return (
      <div>
      <h1 style={{color: "red"}}>Shopping Cart</h1>
      <p>Buy Electronics items at Best Price in India</p>
      </div>
     );
   }
 }
 export default MyHeader;
```

Note: In JSX, JavaScript expressions are written inside curly braces, and since JavaScript objects also use curly braces, the styling in the example above is written inside two sets of curly braces {{}}.

If property name have two worlds, like background-color, then always write it in camel case syntax:

```
<h1 style={{backgroundColor:"green"}}>Shopping Cart</h1>
<h1 style={{textAlign:"center"}}>Shopping Cart</h1>
```

For Multiple attributes create an object to style some elements, and refer to it in the style attribute:
import React from 'react';

```
class MyHeader extends React.Component {
   render() {
      const style1 = {
         color: "white",
         backgroundColor: "Blue",
         textAlign:"center",
         fontFamily: "Arial",
         padding:"20px",
         margin:"0px"
        };
        const style2 = {
         margin:"0px",
         padding:"10px",
         textAlign:"center",
         backgroundColor: "Gray",
         fontFamily: "Arial",
```

```
    };

   return (
    <div>
    <h1 style={style1}>Shopping Cart</h1>
    <p style={style2}>Buy Electronics items at Best Price in India</p>
    </div>
   );
  }
 }
 export default MyHeader;
```

External CSS Stylesheet:

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.

P1.css

```
h1{
        color:white;
        background-color:green;
        text-align:center;
        font-family: Arial;
        padding:20px;
        margin:0px
}
```

P1.js

```
import React from 'react';
import './P1.css';
class MyHeader extends React.Component {
  render() {
    return (
    <div>
    <h1>Shopping Cart</h1>
    <p>Buy Electronics items at Best Price in India</p>
    </div>
   );
  }
 }
 export default MyHeader;
```