

# ReactJS

## React Introduction

- ✓ React is a declarative, efficient, and flexible JavaScript library for building user interfaces.
- ✓ React is used to build single-page applications.
- ✓ React allows us to create reusable UI components.
- ✓ It is an open-source, component-based front-end library which is responsible only for the view layer of the application.
- ✓ It was created by **Jordan Walke**, who was a software engineer at **Facebook**.
- ✓ It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.

## 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.

## Installation ReactJS on Windows

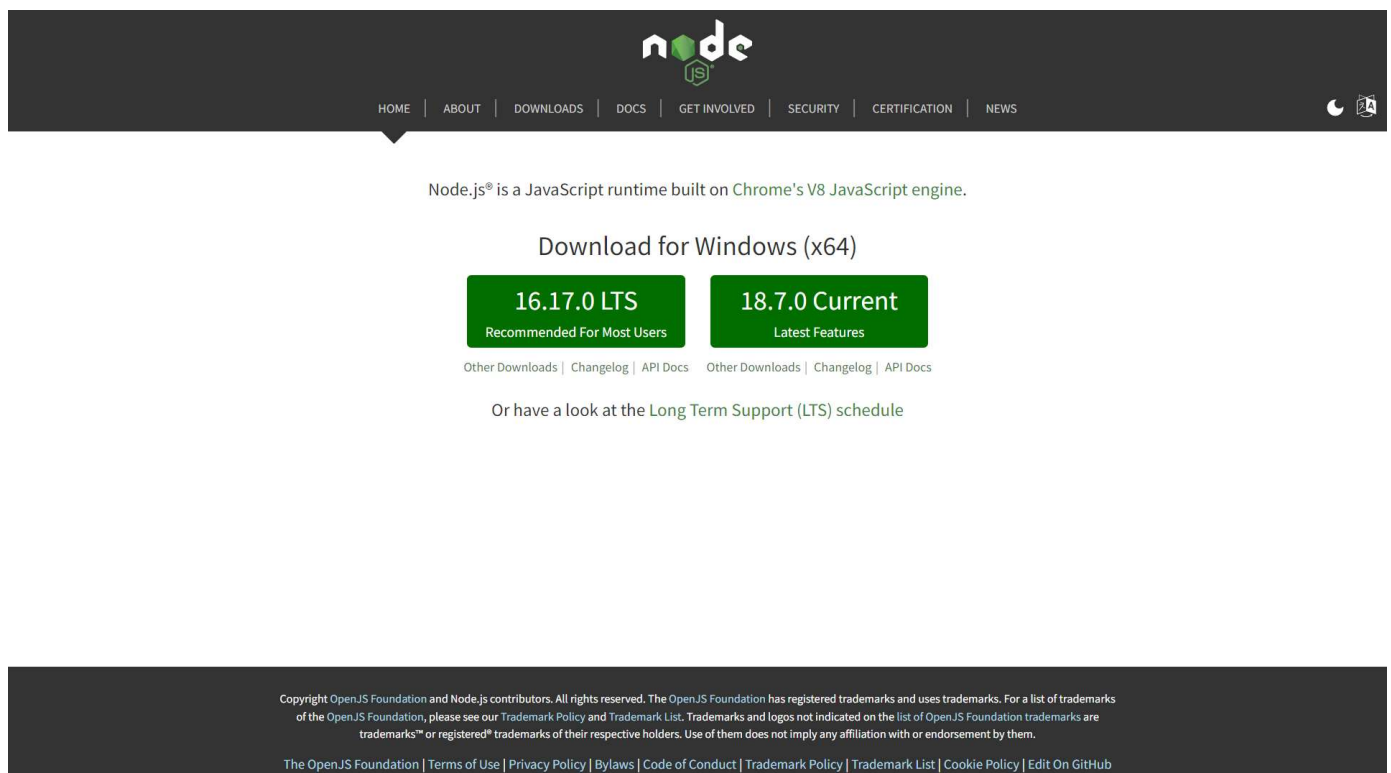
### Step 1:

Install Node.js installer for windows.

Click on <https://nodejs.org/en/>

Here install the LTS version (the one present on the left).

Once downloaded open NodeJS without disturbing other settings, click on the **Next** button until it's completely installed.



node

HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | CERTIFICATION | NEWS

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for Windows (x64)

**16.17.0 LTS**  
Recommended For Most Users

**18.7.0 Current**  
Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) | [Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [Long Term Support \(LTS\) schedule](#)

Copyright OpenJS Foundation and Node.js contributors. All rights reserved. The OpenJS Foundation has registered trademarks and uses trademarks. For a list of trademarks of the OpenJS Foundation, please see our [Trademark Policy](#) and [Trademark List](#). Trademarks and logos not indicated on the list of OpenJS Foundation trademarks are trademarks™ or registered® trademarks of their respective holders. Use of them does not imply any affiliation with or endorsement by them.

[The OpenJS Foundation](#) | [Terms of Use](#) | [Privacy Policy](#) | [Bylaws](#) | [Code of Conduct](#) | [Trademark Policy](#) | [Trademark List](#) | [Cookie Policy](#) | [Edit On GitHub](#)

### Step 2:

Open command prompt to check whether it is completely installed or not type the command

```
node -v
```

```
Microsoft Windows [Version 10.0.22000.856]
(c) Microsoft Corporation. All rights reserved.

C:\Users\murah>node -v
v16.17.0

C:\Users\murah>
```

If the installation went well, it will give you the version you have installed.

### Step 3:

Now in the terminal run the below command:

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

```
Microsoft Windows [Version 10.0.22000.856]
(c) Microsoft Corporation. All rights reserved.

C:\Users\murah>npm install -g create-react-app
npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security
updates. Please upgrade asap.

changed 67 packages, and audited 68 packages in 4s

4 packages are looking for funding
  run `npm fund` for details

2 high severity vulnerabilities

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
```

It will globally install react app for you. To check everything went well run the command

```
create-react-app --version
```

```
C:\Users\murah>create-react-app --version
5.0.1

C:\Users\murah>
```

If everything went well it will give you the installed version of react app

### Step 4:

Now Create a new folder where you want to make your react app using the below command

```
mkdir newfolder
```

**Note:** The **newfolder** in the above command is the name of the folder and can be anything.

Move inside the same folder using the below command

```
cd newfolder (your folder name)
```

## React create-react-app

Starting a new React project is very complicated, with so many build tools. It uses many dependencies, configuration files, and other requirements such as Babel, Webpack, ESLint before writing a single line of React code. Create React App CLI tool removes all that complexities and makes React app simple. For this, we need to install the package using NPM, and then run a few simple commands to get a new React project.

The **create-react-app** is an excellent tool for beginners, which allows you to create and run React project very quickly. It does not take any configuration manually. This tool is wrapping all of the required dependencies like **Webpack**, **Babel** for React project itself and then you need to focus on writing React code only. This tool sets up the development environment, provides an excellent developer experience, and optimizes the app for production.

### Create a new React project

Once the React installation is successful, we can create a new React project using create-react-app command. Here, I choose "firstproject" name for my project.

```
> create-react-app firstproject
```

```
C:\Tutorials\react_pro>create-react-app firstproject

Creating a new React app in C:\Tutorials\react_pro\firstproject.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[ ] - idealTree:firstproject: sill idealTree buildDeps
```

The above command will take some time to install the React and create a new project with the name "firstproject." Now, we can see the terminal as like below.

```
npm audit fix --force
Run `npm audit` for details.

Success! Created firstproject at C:\Tutorials\react_pro\firstproject
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd firstproject
  npm start

Happy hacking!

C:\Tutorials\react_pro>
```

The above screen tells that the React project is created successfully on our system.

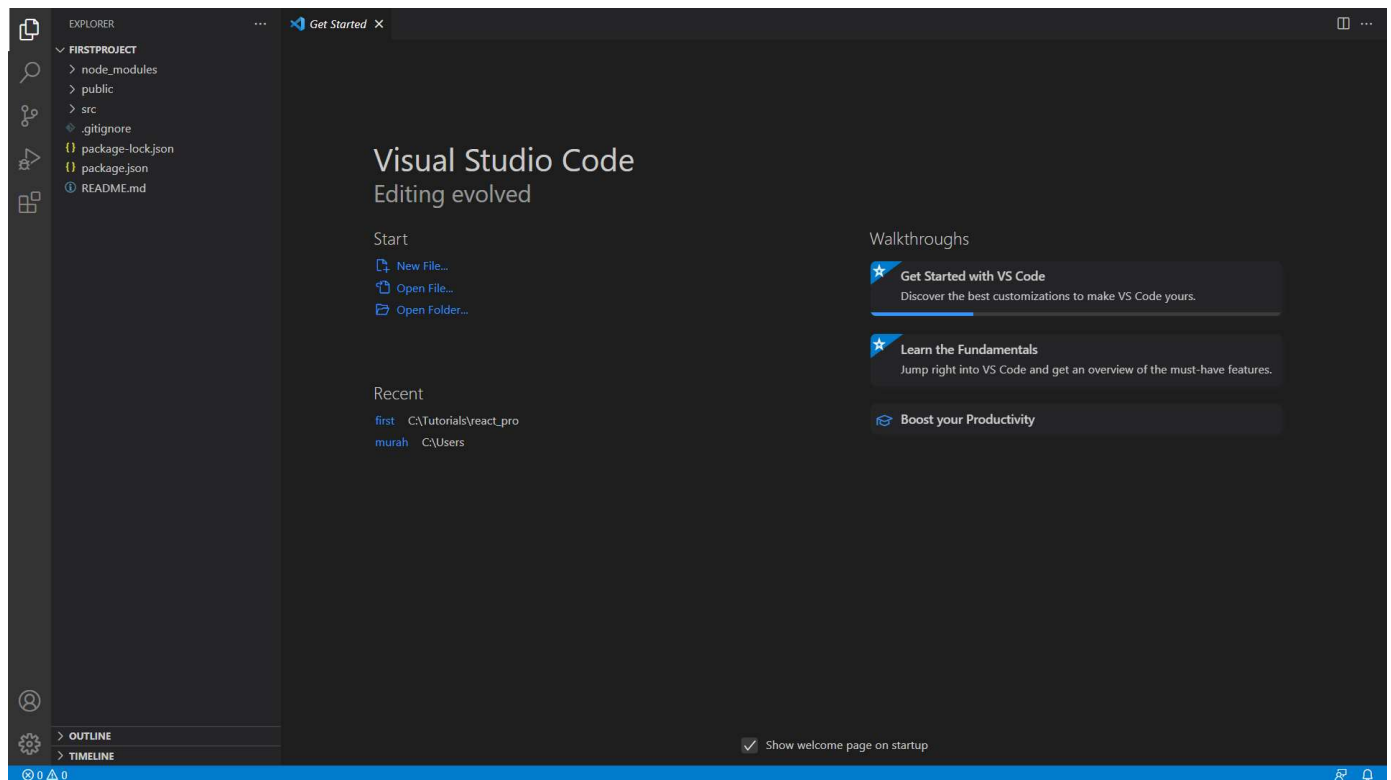
First, Change to the "firstproject" directory.

Next, enter into project by using visual studio code editor.

```
C:\Tutorials\react_pro>cd firstproject  
C:\Tutorials\react_pro\firstproject>code .
```

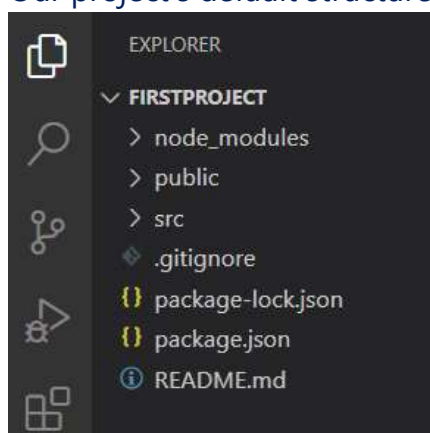
Note:

code . → this command is used to enter the visual studio editor.



Enter into the above screen is a visual studio editor with react project.

Our project's default structure looks like as below image.



In React application, there are several files and folders in the root directory. Some of them are as follows:

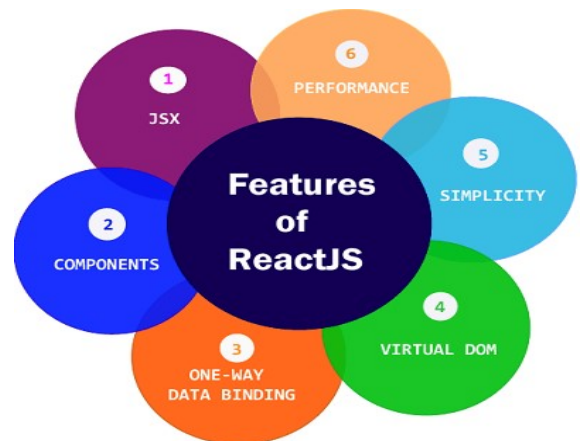
1. **node\_modules**: It contains the React library and any other third party libraries needed.
2. **public**: It holds the public assets of the application. It contains the index.html where React will mount the application by default on the `<div id="root"></div>` element.

3. **src**: It contains the App.css, App.js, App.test.js, index.css, index.js, and serviceWorker.js files. Here, the App.js file always responsible for displaying the output screen in React.
4. **package-lock.json**: It is generated automatically for any operations where npm package modifies either the node\_modules tree or package.json. It cannot be published. It will be ignored if it finds any other place rather than the top-level package.
5. **package.json**: It holds various metadata required for the project. It gives information to npm, which allows to identify the project as well as handle the project's dependencies.
6. **README.md**: It provides the documentation to read about React topics.

## React Features

Currently, ReactJS gaining quick popularity as the best JavaScript framework among web developers. It is playing an essential role in the front-end ecosystem. The important features of ReactJS are as following.

- ✓ JSX
- ✓ Components
- ✓ One-way Data Binding
- ✓ Virtual DOM
- ✓ Simplicity
- ✓ Performance



### JSX

JSX stands for JavaScript XML. It is a JavaScript syntax extension. Its an XML or HTML like syntax used by ReactJS. This syntax is processed into JavaScript calls of React Framework. It extends the ES6 so that HTML like text can co-exist with JavaScript react code. It is not necessary to use JSX, but it is recommended to use in ReactJS.

### Components

ReactJS is all about components. ReactJS application is made up of multiple components, and each component has its own logic and controls. These components can be reusable which help you to maintain the code when working on larger scale projects.

### One-way Data Binding

ReactJS is designed in such a manner that follows unidirectional data flow or one-way data binding. The benefits of one-way data binding give you better control throughout the application. If the data flow is in another direction, then it requires additional features. It is because components are supposed to be immutable and the data within them cannot be changed. Flux is a pattern that

helps to keep your data unidirectional. This makes the application more flexible that leads to increase efficiency.

### Virtual DOM

A virtual DOM object is a representation of the original DOM object. It works like a one-way data binding. Whenever any modifications happen in the web application, the entire UI is re-rendered in virtual DOM representation. Then it checks the difference between the previous DOM representation and new DOM. Once it has done, the real DOM will update only the things that have actually changed. This makes the application faster, and there is no wastage of memory.

### Simplicity

ReactJS uses JSX file which makes the application simple and to code as well as understand. We know that ReactJS is a component-based approach which makes the code reusable as your need. This makes it simple to use and learn.

### Performance

ReactJS is known to be a great performer. This feature makes it much better than other frameworks out there today. The reason behind this is that it manages a virtual DOM. The DOM is a cross-platform and programming API which deals with HTML, XML or XHTML. The DOM exists entirely in memory. Due to this, when we create a component, we did not write directly to the DOM. Instead, we are writing virtual components that will turn into the DOM leading to smoother and faster performance.

## Pros and Cons of ReactJS

### Pros

1. **virtual DOM:** Enables developers to make changes in a small component, without interfering in the whole application.
2. **Reusable Component:** Allow developers to code components one time and then reuse them in the future without the need of rewriting the whole program.
3. **community:** react is supported by a huge community of developers.
4. **SEO-friendly:** react is an SEO-friendly library that allows the applications to be indexed by the google search engine.
5. **Easy to learn:** The library is slightly easier to learn then other JavaScript frameworks.
6. **Downward data flow:** An open-source library that by working with react results in a simplified app that is easier to maintain.
7. **Redux:** One-way dataflow (or Downward data flow) prevents the app from being unstable.

### Cons

1. **Lack of proper documentation:** fast changing and updates in React make it hard to write proper documentation.
2. **Development speed:** the pace of React's development may make developers feel Exhausted by the need for relearning processes.
3. **JSX:** JSX may make new developers confused but it's optional.
4. **SEO Problem:** React had problems with Google and indexing.

# 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.

## Coding JSX

JSX allows us to write HTML elements in JavaScript and place them in the DOM without any `createElement()` and/or `appendChild()` methods.

## Example

### JSX

```
<div>Hello World</div>
```

### Corresponding without JSX

```
React.createElement("div", null, "Hello World");
```

## Why use JSX?

- ✓ It is faster than regular JavaScript because it performs optimization while translating the code to JavaScript.
- ✓ Instead of separating technologies by putting markup and logic in separate files, React uses components that contain both. We will learn components in a further section.
- ✓ It is type-safe, and most of the errors can be found at compilation time.
- ✓ It makes easier to create templates.

## Example without JSX

### App.js (function Component)

```
import React from 'react'
function App() {
  return (
    React.createElement("div", null,
      React.createElement("h1", null, "Hello World!!!"))
  )
}
export default App
```

## Example with JSX

### App.js (class Component)

```
import React, { Component } from 'react'
class App extends Component {
  render() {
    return (<div>
      <h1>Hello world</h1>
    </div> )
  }
}
export default App
```



## App.js (function Component)

```
import React from 'react'

function App() {
  return (
    <div>
      <h1>Hello world</h1>
    </div>
  )
}

export default App
```

## Nested Elements in JSX

To use more than one element, we need to wrap it with one container element. Here, we use **div** as a container element which has **three** nested elements inside it.

### App.js

```
import React from 'react'

function App() {
  return (
    <div>
      <h1>Codetantra</h1>
      <h3>it is a software Company...</h3>
      <p>Training/Support/Development</p>
    </div>
  )
}

export default App
```

## Expressions in JSX

- ✓ With JSX we 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 with JSX Expression

#### App.js

```
import React from 'react'

function App() {
  return (
    <div>
      <h1>sum of 20 + 30 : {20 + 30}</h1>
    </div>
  )
}

export default App
```



## JSX Comments

- ✓ JSX allows us to use comments that begin with `/*` and ends with `*/` and wrapping them in curly braces `{}` just like in the case of JSX expressions.

### Example

```
import React from 'react'

function App() {
  return (
    <div>
      <h1>Codetantra</h1>
      { /*<h3>Software Company</h3>*/ }
    </div>
  )
}

export default App
```

## JSX Styling

- ✓ React always recommends to use **inline** styles.
- ✓ To set inline styles, you need to use **camelCase** syntax.

### Example

```
import React from 'react'

function App() {
  const mystyle = {
    color: "Red",
    border: "2px solid green",
    textAlign: "center"
  }
  return (
    <div>
      <h1 style={mystyle}>Codetantra</h1>
    </div>
  )
}

export default App
```

## Trenary Expression:

```
import React from 'react'

function App() {
  const x = 5;
  return (<div>
    <h1>{x<=5?"true":"false"}</h1>
  </div>)
}

export default App
```

## 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.
- ✓ Components come in two types, Class components and Function components.

### Class Component

- ✓ A class component must include the extends `React.Component` statement.
- ✓ This statement creates an inheritance to `React.Component`, and gives your component access to `React.Component`'s functions.
- ✓ The component also requires a `render()` method, this method returns HTML.

### Example

```
import React from 'react'

class App extends React.Component {
  render() {
    return (
      <div>
        <h1>Class Component</h1>
      </div>
    )
  }
}

export default App
```

### Functional Components

- ✓ Functional components are simply javascript functions.
- ✓ We can create a functional component in React by writing a javascript function.

```
import React from 'react'

function App() {
  return (
    <div>
      <h1>Functional Component</h1>
    </div>
  )
}

export default App
```

## React State

- ✓ React components has a built-in **state** object.
- ✓ The **state** object is where you store property values that belongs to the component.
- ✓ When the **state** object changes, the component re-renders.
- ✓ The state is an instance of React Component Class can be defined as an object of a set of **observable** properties that control the behavior of the component. In other words, the State of a component is an object that holds some information that may change over the lifetime of the component.

### Example-1:

App.js

```
import React, { Component } from 'react'

class App extends Component {
  state = {name:"Raj",age:20,address:"Hyd"}
  render() {
    return (
      <div>
        <h1 style={{color:"green"}}>Name:{this.state.name}</hr>
        Age:{this.state.age}</hr>
        Address:{this.state.address}</hr>
      </h1>
    </div>
    )
  }
}

export default App
```

### Example-2:

App.js

```
import React, { Component } from 'react'

class App extends Component {
  constructor() {
    super()
    this.state = {name:"Raj"}
  }
  changeHandler = () => this.setState({name:"Ravi"})
  render() {
    return (
      <div>
        <center>{this.state.name}<br/>
        <button onClick={this.changeHandler}>click</button>
      </center>
    </div>
    )
  }
}

export default App
```

## React Props

- ✓ Props stand for "Properties." They are **read-only** components.
- ✓ It is an object which stores the value of attributes of a tag and work similar to the HTML attributes.
- ✓ It gives a way to pass data from one component to other components.
- ✓ It is similar to function arguments.
- ✓ Props are passed to the component in the same way as arguments passed in a function.
- ✓ Props are **immutable** so we cannot modify the props from inside the component.
- ✓ Inside the components, we can add attributes called props.
- ✓ These attributes are available in the component as **this.props** and can be used to render dynamic data in our render method.

### Example 1: (Class Component)

#### App.js

```
import React, { Component } from 'react'
import Car from './Car'
class App extends Component {
  render() {
    return (
      <div>
        <center>
          <Car brand="BMW"/>
        </center>
      </div>
    )
  }
}

export default App
```

#### Car.js

```
import React, { Component } from 'react'

export class Car extends Component {
  render() {
    return (
      <div>
        <center>
          Car Brand: {this.props.brand}
        </center>
      </div>
    )
  }
}

export default Car
```

## Example 2: (Functional Component)

### App.js

```
import React from 'react'
import Car from './Car'
function App() {
  return (
    <div>
      <Car brand = "BMW"/>
    </div>
  )
}

export default App
```

### Car.js

```
import React from 'react'

function Car(props) {
  return (
    <div>
      <center>
        Car brand: {props.brand}
      </center>
    </div>
  )
}

export default Car
```

## React Props validation

Props are an important mechanism for passing the **read-only** attributes to React components. The props are usually required to use correctly in the component. If it is not used correctly, the components may not behave as expected. Hence, it is required to use **props validation** in improving react components.

Props validation is a tool that will help the developers to avoid future bugs and problems. It is a useful way to force the correct usage of your components. It makes your code more readable. React components used special property **PropTypes** that help you to catch bugs by validating data types of values passed through props, although it is not necessary to define components with propTypes. However, if you use propTypes with your components, it helps you to avoid unexpected bugs.

### Validating Props

**App.propTypes** is used for props validation in react component. When some of the props are passed with an invalid type, we will get the warnings on JavaScript console. After specifying the validation patterns, we will set the App.defaultProps.

## Syntax:

```
class App extends React.Component {  
  render() {}  
}  
Component.propTypes = { /*Definition */};
```

## ReactJS Props Validator

ReactJS props validator contains the following list of validators.

SN	PropType	Description
1.	PropTypes.array	The props should be an array.
2.	PropTypes.bool	The props should be a boolean.
3.	PropTypes.func	The props should be a function.
4.	PropTypes.number	The props should be a number.
5.	PropTypes.object	The props should be an object.
6.	PropTypes.string	The props should be a string.
7.	PropTypes.symbol	The props should be a symbol.
8.	PropTypes.instanceOf	The props should be an instance of a particular JavaScript class.
9.	PropTypes.isRequired	The props must be provided.
10.	PropTypes.element	The props must be an element.
11.	PropTypes.oneOf()	The props should be one of several types of specific values.

## Example

Here, we are creating an App component which contains all the props that we need. In this example, **App.propTypes** is used for props validation. For props validation, you must have to add this line: **import PropTypes from 'prop-types'** in **App.js** file.

### App.js

```
import React, { Component } from 'react'  
import PropTypes from 'prop-types'  
  
class App extends Component {  
  render() {  
    return (  
      <div>  
        number--{this.props.propNumber}<br/>  
        numberValidation --{this.props.propNumber?"True":"False"}<br/>  
        string--{this.props.propString}<br/>  
        stringValidation--{this.props.propString?"True":"False"}  
      </div>  
    )  
  }  
}  
  
App.propTypes = {  
  propNumber :PropTypes.number,  
  propString :PropTypes.string  
}
```

```
App.defaultProps = {
  propNumber: 1,
  propString: "Murahari"
}

export default App;
```

## React Constructor

- ✓ The constructor is a method used to initialize an object's state in a class.
- ✓ It is automatically called during the creation of an object in a class.
- ✓ The concept of a constructor is the same in React.
- ✓ The constructor in a React component is called before the component is mounted.
- ✓ When you implement the constructor for a React component, you need to call `super(props)` method before any other statement.
- ✓ If you do not call `super(props)` method, `this.props` will be undefined in the constructor and can lead to bugs.

## Syntax

```
Constructor(props){
  super(props);
}
```

In React, constructors are mainly used for two purposes:

1. It is used for initializing the local state of the component by assigning an object to `this.state`.
2. It is used for binding event handler methods that occur in your component.

we cannot call `setState()` method directly in the `constructor()`. If the component needs to use local state, you need directly to use `'this.state'` to assign the initial state in the constructor. The constructor only uses `this.state` to assign initial state, and all other methods need to use `set.state()` method.

## Example

```
import React, { Component } from 'react'

class App extends Component {
  constructor(props) {
    super(props)
    this.state = {
      name: "Murahari",
      color: "Red"
    }
  }
  ChangeColor = () =>{
    this.setState({color: "green"})
  }
  render() {
    return (
      <div>
        <center>
```



```

        <h1 style={{color:this.state.color}}>{this.state.name}</h1>
        <button onClick={this.ChangeColor}>Change</button>
    </center>
</div>
)
}
}
}

export default App

```

## React Component API

- ✓ ReactJS component is a top-level API.
- ✓ It makes the code completely individual and reusable in the application.

Here, we are going to explain the three most important methods available in the React component API.

- ✓ `setState()`
- ✓ `forceUpdate()`
- ✓ `findDOMNode()`

### `setState()`

- ✓ This method is used to update the state of the component.
- ✓ This method does not always replace the state immediately.
- ✓ Instead, it only adds changes to the original state.
- ✓ It is a primary method that is used to update the user interface(UI) in response to event handlers and server responses.

**Note:** In the ES6 classes, `this.method.bind(this)` is used to manually bind the `setState()` method.

### Syntax

```
this.setState(object newState[, function callback]);
```

In the above syntax, there is an optional `callback` function which is executed once `setState()` is completed and the component is re-rendered.

### Example

```

import React, { Component } from 'react'

class App extends Component {
  constructor(props) {
    super(props)
    this.state = {
      msg:"Codetantra",
    }
  }
  ChangeName = () =>{

```

```

    this.setState({msg:"CT"})
  }
  render() {
    return (
      <div>
        <center>
          <h1>{this.state.msg}</h1>
          <button onClick={this.ChangeName}>Change</button>
        </center>
      </div>
    )
  }
}

export default App

```

## forceUpdate()

This method allows us to update the component manually.

### Syntax

```
Component.forceUpdate(callback);
```

### Example

```

import React, { Component } from 'react'

class App extends Component {
  constructor(props) {
    super(props)
    this.forceUpdateState = this.forceUpdateState.bind(this) ;
  }
  forceUpdateState(){
    this.forceUpdate()
  }
  render() {
    return (
      <div>
        <center>
          <h3>{parseInt(Math.random()*100)} </h3>
          <button onClick={this.forceUpdateState}>Change</button>
        </center>

      </div>
    )
  }
}

export default App

```

## findDOMNode()

For DOM manipulation, you need to use `ReactDOM.findDOMNode()` method. This method allows us to find or access the underlying DOM node.

### Syntax

```
ReactDOM.findDOMNode(component);
```

### Example

```
import React, { Component } from 'react'
import ReactDOM from 'react-dom'

class App extends Component {
  constructor(props) {
    super(props)
    this.changeColor = this.changeColor.bind(this);
  }

  changeColor(){
    var myDiv = document.getElementById('mydiv');
    ReactDOM.findDOMNode(myDiv).style.color = "green";
  }

  render() {
    return (
      <div>
        <center>
          <div id='mydiv'>NODE</div>
          <button onClick={this.changeColor}>Click</button>
        </center>
      </div>
    )
  }
}

export default App
```

### Example 2:

#### App.js

```
import React, { Component } from 'react'

export default class App extends Component {
  constructor(props) {
    super(props)
    this.state = {
      count:0
    }
  }

  sub = () =>{
    this.setState({count:this.state.count-1})
  }
```

## Output



Example:1  
App.js

```
import React, { Component } from 'react'

class App extends Component {
  constructor(props) {
    super(props)

    this.state = {
      username: ''
    }
  }

  changeUsername = (event) => {
    this.setState({username: event.target.value})
  }

  render() {
    return (
      <div>
```

```

        <center>
        <form>
            <input type="Text" placeholder='username' onChange={this.changeUsername}/><br/>
            UserName: {this.state.username}
        </form>
        </center>
    </div>
    )
}
}
}

export default App

```

### Example2:

```

import React, { Component } from 'react'

class App extends Component {
  constructor(props) {
    super(props)

    this.state = {
      username: ''
    }
  }
  changeUsername = (event) => {
    this.setState({username:event.target.value})
  }

  submitHandler = (event) => {
    alert(`username: ${this.state.username}`);
    event.preventDefault();
  }
  render() {
    return (
      <div>
        <center>
          <form onSubmit={this.submitHandler}>
            <input type="Text" placeholder='username' onChange={this.changeUsername}/><br/>
            <input type="submit" value="submit"/>
          </form>
        </center>
      </div>
    )
  }
}

export default App

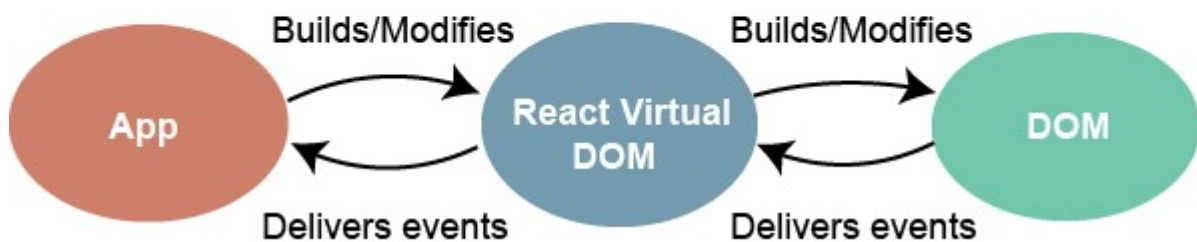
```

## React Events

An event is an action that could be triggered as a result of the user action or system generated event. For example, a mouse click, loading of a web page, pressing a key, window resizes, and other interactions are called 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.

## Events Handler



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

1. React events are named as camelCase instead of lowercase.
2. With JSX, a function is passed as the event handler instead of a string. For example.

### Example1:

```
import React, { Component } from 'react'

class App extends Component {
  constructor(props) {
    super(props)

    this.state = {
      username: ''
    }
  }
  changeUsername (event) {
    this.setState({username:event.target.value})
  }

  submitHandler = (event) => {
    alert(`username: ${this.state.username}`);
    event.preventDefault();
  }

  render() {
    return (
      <div>
        <center>
```

```

        <form onSubmit={this.submitHandler}>
          <input type="Text" placeholder='username'
onChange={this.changeUsername.bind(this)}/><br/>
          <input type="submit" value="submit"/>
        </form>
      </center>
    </div>
  )
}
}
}

export default App

```

## React Conditional Rendering

In React, we can create multiple components which encapsulate behavior that we need. After that, we can render them depending on some conditions or the state of our application. In other words, based on one or several conditions, a component decides which elements it will return. In React, conditional rendering works the same way as the conditions work in JavaScript. We use JavaScript operators to create elements representing the current state, and then React Component update the UI to match them.

### Example

#### App.js (Class Component)

```

import React, { Component } from 'react'

class App extends Component {
  constructor() {
    super()
    this.state = {count:0}
  }
  changeHandler = () => this.setState({count:this.state.count+1})

  render() {
    if(this.state.count<=5)
    return (
      <div>
        <center>{this.state.count}<br/>
        <button onClick={this.changeHandler}>click</button>
        </center>
      </div> )
    else
    return(
      <div>
        {this.setState({count:0})}
        {this.state.count}<br/>
        <button onClick={this.changeHandler}>click</button>
      </div>
    )
  }
}

export default App

```



## Example

### App.js (functional Component)

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

function App() {
  let [count, setCount] = useState(0)
  const ClickHandler = () => setCount(count+1)
  if(count<=5)
  return (
    <div>
      <center>
        <h1>{count}</h1>
        <button onClick={ClickHandler}>click</button>
      </center>
    </div>
  )
  else{
    count = setCount(0)
    return (
      <div>
        <center>
          <h1>{count}</h1>
          <button onClick={ClickHandler}>click</button>
        </center>
      </div>
    )
  }
}

export default App
```

## React Lists

- ✓ Lists are used to display data in an ordered format and mainly used to display menus on websites.

### Example:

#### App.js

```
import React, { Component } from 'react'
import './App.css'

export default class App extends Component {
  render() {
    const myli = ["Home", "News", "contact", "about"]
    const listitem = myli.map((li) => <li>{li}</li>)
    return(
      <div>
        <ul>{listitem}</ul>
      </div>
    )
  }
}
```

## App.css

```
ul{
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333;
}
li{
  float:left;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}
```

## React Keys

A key is a unique identifier. In React, it is used to identify which items have changed, updated, or deleted from the Lists. It is useful when we dynamically created components or when the users alter the lists. It also helps to determine which components in a collection needs to be re-rendered instead of re-rendering the entire set of components every time.

Keys should be given inside the array to give the elements a stable identity. The best way to pick a key as a string that uniquely identifies the items in the list.

## Example

### App.js

```
import React, { Component } from 'react'

export default class App extends Component {
  render() {
    const myli = ["Home", "News", "contact", "about"]
    const listitem = myli.map((li) => <li key = {li.id}>{li}</li>)
    return(
      <div>
        <ul >{listitem}</ul>
      </div>
    )
  }
}
```

## React Refs

Refs is the shorthand used for **references** in React. It is similar to **keys** in React. It is an attribute which makes it possible to store a reference to particular DOM nodes or React elements. It provides a way to access React DOM nodes or React elements and how to interact with it. It is used when we want to change the value of a child component, without making the use of props.

## How to create Refs

In React, Refs can be created by using `React.createRef()`. It can be assigned to React elements via the `ref` attribute. It is commonly assigned to an instance property when a component is created, and then can be referenced throughout the component.

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.callRef = React.createRef();
  }
  render() {
    return <div ref={this.callRef} />;
  }
}
```

## How to access Refs

In React, when a ref is passed to an element inside render method, a reference to the node can be accessed via the current attribute of the ref.

```
const node = this.callRef.current;
```

## Example

### App.js

```
import React, { Component } from 'react'

class App extends Component {
  constructor(props) {
    super(props)
    this.inputref = React.createRef();
  }

  clickHandler = () => {
    this.inputref.current.focus();
    alert(this.inputref.current.value)
  }

  render() {
    return (
      <div>
        <center>
          <input type = "Text" ref={this.inputref}/><br/>
          <input type = "button" value = "click" onClick={this.clickHandler}/>
        </center>
      </div>
    )
  }
}

export default App
```

## React Fragments

In React, whenever we want to render something on the screen, we need to use a render method inside the component. This render method can return **single** elements or **multiple** elements. The render method will only render a single root node inside it at a time. However, if you want to return multiple elements, the render method will require a **'div'** tag and put the entire content or elements inside it. This extra node to the DOM sometimes results in the wrong formatting of your HTML output and also not loved by the many developers.

### Syntax

```
<React.Fragment>
  <h2> child1 </h2>
  <p> child2 </p>
  .. .....
</React.Fragment>
```

### Example

```
import React, { Component } from 'react'

export default class App extends Component {
  render() {
    return (
      <React.Fragment>
        <h1>Hello world</h1>
      </React.Fragment>
    )
  }
}
```

## React CSS

There are many ways to style React with CSS, this tutorial will take a closer look at three common ways:

1. Inline styling
2. CSS stylesheets
3. CSS Modules

### Inline Styling

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

#### Example:

#### App.js

```
import React, { Component } from 'react'

export default class App extends Component {
  render() {
    return (
```

```

    <div>
      <h1 style={{color:"white",backgroundColor:"black",textAlign:"center"}}>React
inline styles</h1>
    </div>
  )
}
}

```

**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 `{{}}`.

## 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.

### App.css

```

body {
  background-color: #282c34;
  color: white;
  padding: 40px;
  font-family: Sans-Serif;
  text-align: center;
}

```

### App.js

```

import React, { Component } from 'react'
import './App.css'

export default class App extends Component {
  render() {
    return (
      <div>
        <h1>React inline styles</h1>
      </div>
    )
  }
}

```

## CSS Modules

- ✓ Another way of adding styles to your application is to use CSS Modules.
- ✓ CSS Modules are convenient for components that are placed in separate files.

Create the CSS module with the `.module.css` extension, example: `mystyle.module.css`.

### Example:

#### mystyle.module.css

```

.bigblue {
  color: DodgerBlue;
  padding: 40px;
  font-family: Sans-Serif;
  text-align: center;
}

```

## Car.js

```
import React, { Component } from 'react'
import styles from './mystyle.module.css'

class Car extends Component {
  render() {
    return (
      <div>
        <h1 className={styles.bigblue}>Hello Car!</h1>
      </div>
    )
  }
}

export default Car
```

## App.js

```
import React, { Component } from 'react'
import Car from './Car'

export default class App extends Component {
  render() {
    return (
      <div>
        <Car/>
      </div>
    )
  }
}
```

## React Map

A map is a data collection type where data is stored in the form of pairs. It contains a unique key. The value stored in the map must be mapped to the key. We cannot store a duplicate pair in the map(). It is because of the uniqueness of each stored key. It is mainly used for fast searching and looking up data.

In React, the `map()` method is used to traverse and display a list of similar objects of a component. A map is not the feature of React. Instead, it is the standard JavaScript function that could be called on any array. The `map()` method creates a new array by calling a provided function on every element in the calling array.

## Example

### App.js

```
import React, { Component } from 'react'

export default class App extends Component {
  render() {
    const myli = ["Home", "News", "contact", "about"]
    const listitem = myli.map((li) => <li key = {li.id}>{li}</li>)
```

```

    return(
      <div>
        <ul >{listitem}</ul>
      </div>
    )
  }
}

```

## React Table

A table is an arrangement which organizes information into rows and columns. It is used to store and display data in a structured format.

Example:

App.js

```

import React, { Component } from 'react'

export default class App extends Component {
  render() {
    const data = [
      { name: "Anom", age: 19, gender: "Male" },
      { name: "Megha", age: 19, gender: "Female" },
      { name: "Subham", age: 25, gender: "Male"},
    ]

    return (
      <div>
        <table>
          <tr>
            <th>Name</th>
            <th>Age</th>
            <th>Gender</th>
          </tr>
          {data.map((val, key) => {
            return (
              <tr key={key}>
                <td>{val.name}</td>
                <td>{val.age}</td>
                <td>{val.gender}</td>
              </tr>
            )
          })}
        </table>
      </div>
    )
  }
}

```



## React Higher-Order Components

Higher-order components or HOC is the advanced method of reusing the component functionality logic. It simply takes the original component and returns the enhanced component.

### Syntax:

```
const EnhancedComponent = higherOrderComponent(OriginalComponent);
```

### Reason to use Higher-Order component:

- ✓ Easy to handle
- ✓ Get rid of copying the same logic in every component
- ✓ Makes code more readable

### Example1:

#### Name.js

```
import React from 'react'

const EnhancedComponent = (OriginalComponent) => {
  class NewComponent extends React.Component {
    render() {
      return <OriginalComponent name="ReactJS" />
    }
  }
  return NewComponent
}

export default EnhancedComponent;
```

#### App.js

```
import React from "react";
import EnhancedComponent from './Name'

class App extends React.Component {
  render() {
    return <h1>{this.props.name}</h1>
  }
}

export default EnhancedComponent(App);
```

### Example2:

#### Example.js

```
import React from 'react'

const EnhancedComponent = (OriginalComponent) => {
  class NewComponent extends React.Component {

    constructor(props) {
      super(props);
      // Set initial count to be 0
      this.state = { count: 0 };
    }
  }
}
```

```

    handleClick = () => {
      // Incrementing the count
      this.setState({ count: this.state.count + 1 });
    }

    render() {

      // passed a handleClick and count in the originalComponent
      // as a props for calling and adding the functionality
      return <OriginalComponent
        handleClick={this.handleClick}
        show={this.state.count} />
    }
  }
  // Returns the new component
  return NewComponent
}
// Export main Component
export default EnhancedComponent

```

## App.js

```

import React from 'react'
import './App.css'
// importing HighOrder file
import EnhancedComponent from './Name'

class App extends React.Component {
  render() {
    // Destructuring the props
    const { show, handleClick } = this.props

    // Calling out the props
    return (<center><button onClick={handleClick}>{show}</button></center>)
  }
}

export default EnhancedComponent(App);

```

## React Code splitting

### Example

### Name.js

```

import React, { Component } from 'react'

export default class Name extends Component {
  render() {
    return (
      <div>ReactJS</div>
    )
  }
}

```

## App.js

```
import React, { Component } from 'react'
const Name = React.lazy(()=>import('./Name'))

export default class App extends Component {
  render() {
    return (
      <div>

        <Name/>
      </div>
    )
  }
}
```