React is a JavaScript library for building user interfaces.

React is used to build single-page applications.

React allows us to create reusable UI components.

**ReactJS** is an open-source front-end JavaScript library for building user interfaces. **ReactJS** is maintained by Facebook and a community of individual developers and companies. It is widely used as a base in building single-page websites and mobile applications. It is very easy to use, and it allows users to create reusable UI components.

## Features of ReactJS

**JSX :**JSX is an extension to javascript. Though it is not mandatory to use JSX in react, it is one of the good features and easy to use.

**Components**: Components are like pure javascript functions that help make the code easy by splitting the logic into reusable independent code. We can use components as functions and components as classes. Components also have a state, props which makes life easy. Inside a class, the state of each of the props is maintained.

**Virtual DOM:**React creates a virtual dom, i.e., in-memory data -structure cache. Only the final changes of DOM has later updated in the browsers DOM.

**Javascript Expressions:**JS expressions can be used in the jsx files using curly brackets, for example {}.

## Advantages of ReactJS

Here, are important pros/benefits of using ReactJS:

* ReactJS uses virtual dom that makes use of in-memory data-structure cache, and only the final changes are updated in browsers dom. This makes the app faster.
* You can create components of your choice by using the react component feature. The components can be reused and also helpful in code maintenance.
* Reactjs is an open-source javascript library, so it is easy to start with.
* ReactJS has become very popular in a short span and maintained by Facebook and Instagram. It is used by many famous companies like Apple, Netflix, etc.
* Facebook maintains ReactJS, the library, so it is well maintained and kept updated.
* ReactJS can be used to develop rich UI for both desktop and mobile apps.
* Easy to debug and test as most of the coding is done in [Javascript](https://www.guru99.com/interactive-javascript-tutorials.html) rather than on Html.

**Disadvantages of ReactJS**

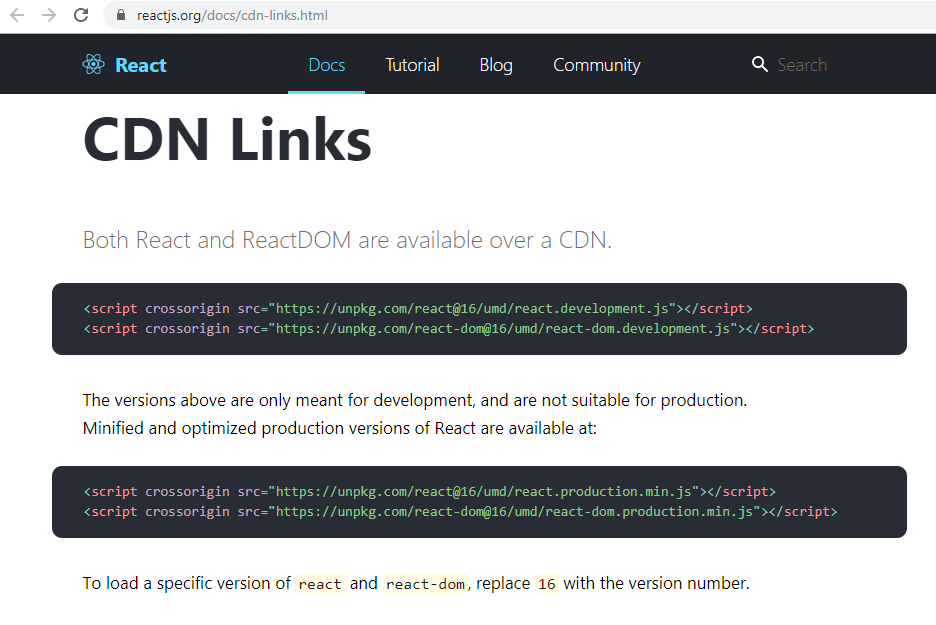
Here, are cons/ drawbacks of using ReactJS:

* Most of the code is written in JSX, i.e., Html and css are part of javascript, it can be quite confusing as most other frameworks prefer keeping Html separate from the javascript code.
* The file size of ReactJS is large.

## Using ReactJS from CDN

To start working with react, we need to first install reactjs. You can easily get started to use reactjs by using the CDN javascript files, as shown below.

Go to the official site of reactjs to get the CDN links, i.e., <https://reactjs.org/docs/cdn-links.html> and you will get the required files to explain the following image.



### For dev

<script crossorigin src="https://unpkg.com/react@version/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@version/umd/react-dom.development.js"></script>

### For prod:

<script crossorigin src="https://unpkg.com/react@**version**/umd/react.production.min.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@**version**/umd/react-dom.production.min.js"></script>

Replace **version** with the latest react version for both react-development.js and react-dom.developement.js. You can host the files at your end to start working with reactjs.

In case if you are planning to use the CDN files, make sure to keep the cross-origin attribute, to avoid cross-domain issues. Reactjs code cannot be executed directly in the browser and needs to be transpired using Babel to javascript before executing in the browser.

Here is the BabelJS script that can be used:

<script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>

Here is the working ReactJS example using cdn files and babeljs script.

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>Hello World</title>

<script src="https://unpkg.com/react@16/umd/react.development.js"></script>

<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>

<script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>

</head>

<body>

<div id="app"></div>

<script type="text/babel">

ReactDOM.render(

<h1>Hello, from Guru99 Tutorials!</h1>,

document.getElementById('app')

);

</script>

</body>

</html>

**Output:**



We will get into the details of the code in the next chapter, let us see the working here with CDN files. It is said that using babel script directly is not a good practice, and newcomers can just use it to learn reactjs for now. In production, you will have to install react using npm package.

## Using NPM Packages

Make sure you have nodejs installed. If not installed, go through this tutorial for nodejs (<https://www.guru99.com/node-js-tutorial.html>) installation.

Once you have nodejs installed, create a folder **reactproj/**.

To start with project setup, run command **npm init**.

This is how the folder structure will look like:

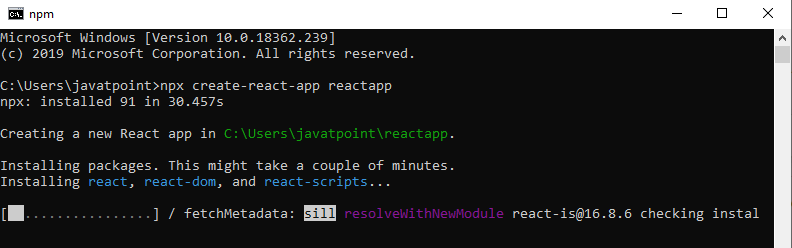
### Create a new React project

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

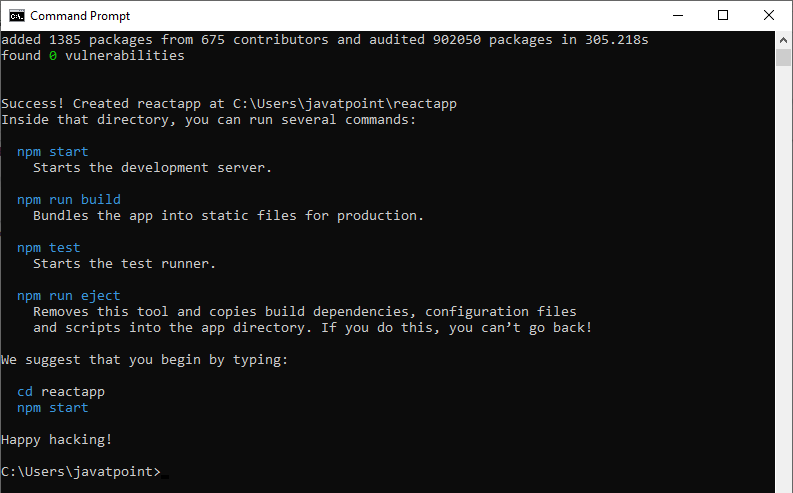
1. C:\Users\javatpoint> create-react-app reactproject

#### NOTE: We can combine the above two steps in a single command using npx. The npx is a package runner tool which comes with npm 5.2 and above version.

1. C:\Users\javatpoint> npx create-react-app reactproject



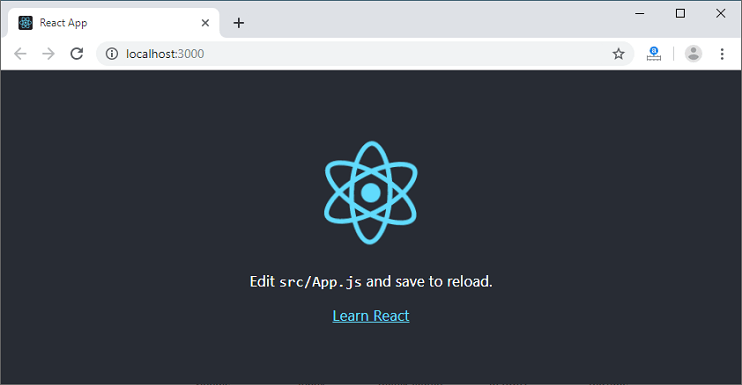
The above command will take some time to install the React and create a new project with the name "reactproject." Now, we can see the terminal as like below.



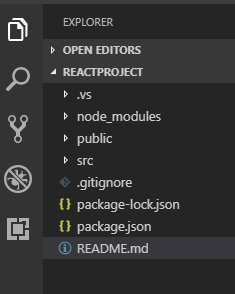
The above screen tells that the React project is created successfully on our system. Now, we need to start the server so that we can access the application on the browser. Type the following command in the terminal window.

1. $ cd Desktop
2. $ npm start

NPM is a package manager which starts the server and access the application at default server [http://localhost:3000](http://localhost:3000/). Now, we will get the following screen.



Next, open the project on Code editor. Here, I am using Visual Studio Code. 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 Environment Setup

Now, open the **src >> App.js** file and make changes which you want to display on the screen. After making desired changes, **save** the file. As soon as we save the file, Webpack recompiles the code, and the page will refresh automatically, and changes are reflected on the browser screen. Now, we can create as many components as we want, import the newly created component inside the **App.js** file and that file will be included in our main **index.html** file after compiling by Webpack.

Next, if we want to make the project for the production mode, type the following command. This command will generate the production build, which is best optimized.

1. $ npm build

## Features of ReactJS

**JSX :**JSX is an extension to javascript. Though it is not mandatory to use JSX in react, it is one of the good features and easy to use.

**Components**: Components are like pure javascript functions that help make the code easy by splitting the logic into reusable independent code. We can use components as functions and components as classes. Components also have a state, props which makes life easy. Inside a class, the state of each of the props is maintained.

**Virtual DOM:**React creates a virtual dom, i.e., in-memory data -structure cache. Only the final changes of DOM has later updated in the browsers DOM.

**Javascript Expressions:**JS expressions can be used in the jsx files using curly brackets, for example {}.

## Advantages of ReactJS

Here, are important pros/benefits of using ReactJS:

* ReactJS uses virtual dom that makes use of in-memory data-structure cache, and only the final changes are updated in browsers dom. This makes the app faster.
* You can create components of your choice by using the react component feature. The components can be reused and also helpful in code maintenance.
* Reactjs is an open-source javascript library, so it is easy to start with.
* ReactJS has become very popular in a short span and maintained by Facebook and Instagram. It is used by many famous companies like Apple, Netflix, etc.
* Facebook maintains ReactJS, the library, so it is well maintained and kept updated.
* ReactJS can be used to develop rich UI for both desktop and mobile apps.
* Easy to debug and test as most of the coding is done in [Javascript](https://www.guru99.com/interactive-javascript-tutorials.html) rather than on Html.

## Disadvantages of ReactJS

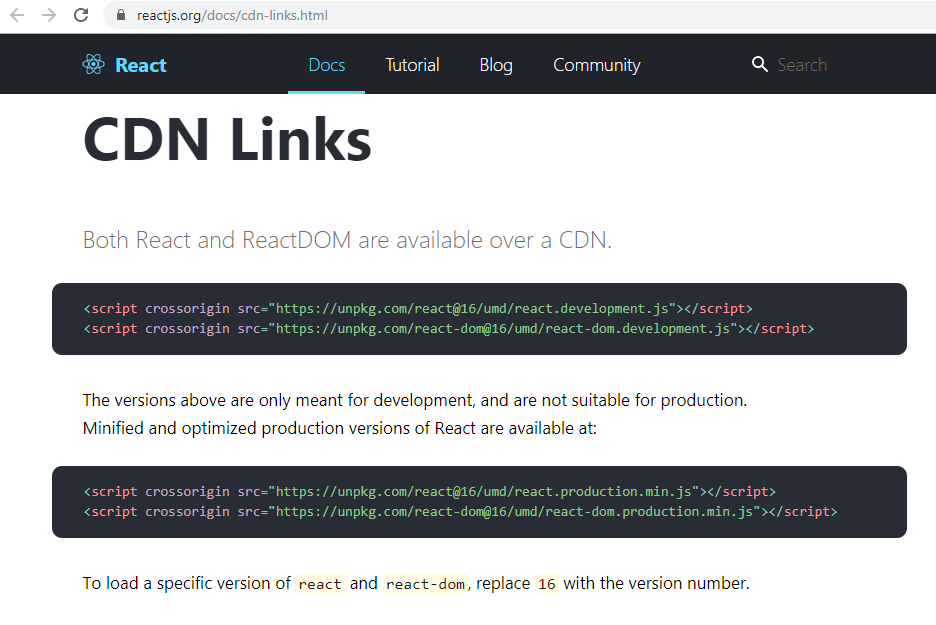
Here, are cons/ drawbacks of using ReactJS:

* Most of the code is written in JSX, i.e., Html and css are part of javascript, it can be quite confusing as most other frameworks prefer keeping Html separate from the javascript code.
* The file size of ReactJS is large.

## Using ReactJS from CDN

To start working with react, we need to first install reactjs. You can easily get started to use reactjs by using the CDN javascript files, as shown below.

Go to the official site of reactjs to get the CDN links, i.e., <https://reactjs.org/docs/cdn-links.html> and you will get the required files to explain the following image.



### For dev

<script crossorigin src="https://unpkg.com/react@version/umd/react.development.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@version/umd/react-dom.development.js"></script>

### For prod:

<script crossorigin src="https://unpkg.com/react@**version**/umd/react.production.min.js"></script>

<script crossorigin src="https://unpkg.com/react-dom@**version**/umd/react-dom.production.min.js"></script>

Replace **version** with the latest react version for both react-development.js and react-dom.developement.js. You can host the files at your end to start working with reactjs.

In case if you are planning to use the CDN files, make sure to keep the cross-origin attribute, to avoid cross-domain issues. Reactjs code cannot be executed directly in the browser and needs to be transpired using Babel to javascript before executing in the browser.

Here is the BabelJS script that can be used:

<script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>

Here is the working ReactJS example using cdn files and babeljs script.

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>Hello World</title>

<script src="https://unpkg.com/react@16/umd/react.development.js"></script>

<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>

<script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>

</head>

<body>

<div id="app"></div>

<script type="text/babel">

ReactDOM.render(

<h1>Hello, from Guru99 Tutorials!</h1>,

document.getElementById('app')

);

</script>

</body>

</html>

**Output:**



We will get into the details of the code in the next chapter, let us see the working here with CDN files. It is said that using babel script directly is not a good practice, and newcomers can just use it to learn reactjs for now. In production, you will have to install react using npm package.

## Using NPM Packages

Make sure you have nodejs installed. If not installed, go through this tutorial for nodejs (<https://www.guru99.com/node-js-tutorial.html>) installation.

Once you have nodejs installed, create a folder **reactproj/**.

To start with project setup, run command **npm init**.

This is how the folder structure will look like:

reactproj\

package.json

Now we will install the packages that we need.

Here are the list of packages for reactjs:

npm install react --save-dev

npm install react-dom --save-dev

npm install react-scripts --save-dev

Open the command prompt and run above commands inside the folder reactproj/.

Create a folder **src/** where all the js code will come in that folder. All the code for reactjs project will be available in the src/ folder. Create a file index.js and add import react and react-dom, as shown below.

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

ReactDOM.render(

<h1>Hello, from Guru99 Tutorials!</h1>,

document.getElementById('root')

);

We have returned the basic code for reactjs. We will explain the details of it in the next chapter. We want to display **Hello, from Guru99 Tutorials**and the same is given to the dom element with id “root”.It is taken from the index.html file, which is the start file, as shown below.

Create a folder public/ and add index.html in that as shown below

**index.html**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>ReactJS Demo</title>

</head>

<body>

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

</body>

</html>

The package react-scripts will take care of compiling the code and starting the server to display the html file i.e index.html. You need to add the command in package.json that will take care of using react-scripts to compile the code and start server as shown below:

"scripts": {

"start": "react-scripts start"

}

After installing all the packages and adding the above command, the final package.json is as follows:

**Package.json**

{

"name": "reactproj",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"start": "react-scripts start"

},

"author": "",

"license": "ISC",

"devDependencies": {

"react": "^16.9.0",

"react-dom": "^16.9.0",

"react-scripts": "^3.1.1"

}

}

To start testing reactjs run the command

npm run start

C:\reactproj>npm run start

> reactproj@1.0.0 start C:\reactproj

> react-scripts start

It will open browser with url http://localhost:3000/ as shown below:

**public/index.html**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>ReactJS Demo</title>

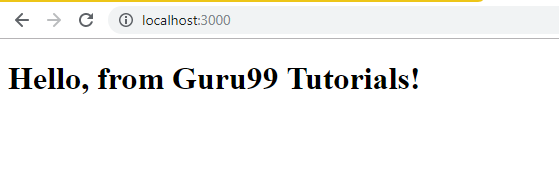
</head>

<body>

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

</body>

</html>



We are going to use the same process to execute the javascript files in the next chapters too. Add all your js and .jsx file in src/ folder .The file structure will be as follows:

reatproj/

src/

index.js

node\_modules/

public/

index.html

package.json

### How to Create Your First React Project Setup

Here is a step by step guide in this ReactJS Tutorial to start with the first react application.

**Step 1)** Import the react packages.  
1. To start with ReactJS, we need to first import the react packages as follows.

import React from 'react';

import ReactDOM from 'react-dom';

2. Save the file as index.js in src/ folder

**Step 2)** Write Simple Code.  
We will write a simple code in this tutorial React JS, wherein we will display the message **“Hello, from Guru99 Tutorials!”**

ReactDOM.render(

<h1>Hello, from Guru99 Tutorials!</h1>,

document.getElementById('root')

);

**ReactDOM.render**will add the <h1> tag to the element with id root. Here is the html file we are having:

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>ReactJS Demo</title>

</head>

<body>

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

</body>

</html>

**Step 3)** Compile the Code.

Next in this React.js Tutorial, we need to compile the code to get the output in the browser.

Here is the folder structure:

reactproj/

node\_modules/

src/

index.js

package.json

public/

index.html

We have added the commands to compile the final file in package.json as follows:

"scripts": {

"start": "react-scripts start"

},

To compile the final file run following command:

npm run start

When you run above command, it will compile the files and notify you if any error, if all looks good, it will open the browser and the run the index.html file at http://localhost:3000/index.html

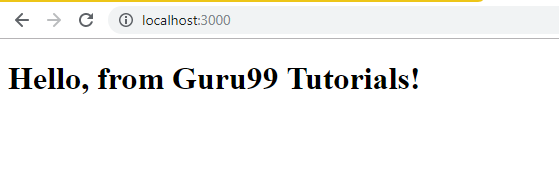
Command: npm run start:

C:\reactproj>npm run start

> reactproj@1.0.0 start C:\reactproj

> react-scripts start

**Step 4)** Check Output.  
The URL http://localhost:3000 will open in the browser once the code is compiled as shown below:



## What is JSX?

JSX is an extension to javascript. It is a template script where you will have the power of using HTML and Javascript together.

Here is a simple example of a JSX code.

const h1tag = "<h1>Hello, from Guru99 Tutorials!</h1>";

### Why we need JSX in React?

For a UI, we need Html, and each element in the dom will have events to be handled, state changes, etc.

In case of React, it allows us to make use of Html and javascript in the same file and take care of the state changes in the dom in an efficient manner.

### Expressions in JSX

Here is a simple example of how to use expressions in JSX.

In earlier ReactJS examples, we had written something like :

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

ReactDOM.render(

<h1>Hello, from Guru99 Tutorials!</h1>,

document.getElementById('root')

);

We will now change the above code to add expressions. Expressions are used inside curly brackets {}, and they are expanded during run time. Expressions in react are the same as javascript expressions.

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

const display = "Hello, from Guru99 Tutorials!";

const h1tag = "<h1>{display}</h1>";

ReactDOM.render(

h1tag,

document.getElementById('root')

);

Let us now test the same in the browser.



You can see that the {display} expression is not replaced. React does not know what to do when an expression is used inside the .js file.

Let us now add changes and create a .jsx file, as shown below:

**test.jsx**

import React from 'react';

import ReactDOM from 'react-dom';

const display = "Hello, to Guru99 Tutorials";

const h1tag =<h1>{display}</h1>;

export default h1tag;

We have added the required code and will use the text.jsx file in index.js.We want the **h1tag**variable to be used inside script.js, so the same is exported as shown above in the test.jsx

Here is the modified code in index.js

import React from 'react';

import ReactDOM from 'react-dom';

import h1tag from './test.jsx';

ReactDOM.render(

h1tag,

document.getElementById('root')

);

To use the test.jsx in index.js we have to import it first as shown below:

import h1tag from './test.jsx';

We can use the h1tag now in the ReactDOM.render as shown below:

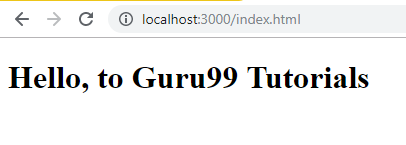
ReactDOM.render(

h1tag,

document.getElementById('root')

);

**Here is the output when we check the same in the browser:**



## What are Components in ReactJS?

Components are like pure javascript functions that help make the code easy by splitting the logic into reusable independent code.

### Components as functions

**test.jsx**

import React from 'react';

import ReactDOM from 'react-dom';

function Hello() {

return <h1>Hello, from Guru99 Tutorials!</h1>;

}

const Hello\_comp = <Hello />;

export default Hello\_comp;

We have created a function called **Hello** that returned h1 tag as shown above. The name of the function acts as an element, as shown below:

const Hello\_comp = <Hello />;

export default Hello\_comp;

The Component**Hello** is used as an Html tag, i.e., **<Hello />**and assigned to**Hello\_comp**variable and the same is exported using export.

Let us now use this component in index.js file as shown below:

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import Hello\_comp from './test.jsx';

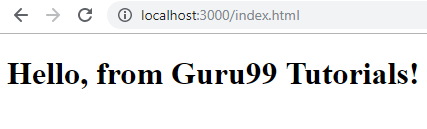
ReactDOM.render(

Hello\_comp,

document.getElementById('root')

);

**Here is the output in the browser:**



### Class as Component

Here is a ReactJS example that uses a class as a component.

**test.jsx**

import React from 'react';

import ReactDOM from 'react-dom';

class Hello extends React. Component {

render() {

return <h1>Hello, from Guru99 Tutorials!</h1>;

}

}

export default Hello;

We can use Hello component in index.js file as follows:

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import Hello from './test.jsx';

ReactDOM.render(

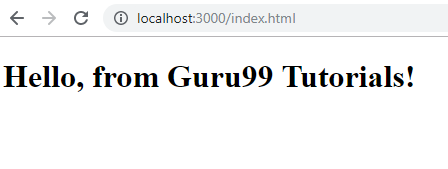
<Hello />,

document.getElementById('root')

);

The Component Hello is used as an Html tag i.e., **<Hello />.**

**Here is the output of the same.**



## What is a State in ReactJS?

A state is a javascript object similar to props that have data to be used with the reactjs render. The state data is a private object and is used within components inside a class.

### Example of State

Here is a working example on how to use state inside a class.

**test.jsx**

import React from 'react';

import ReactDOM from 'react-dom';

class Hello extends React.Component {

constructor(props) {

super(props);

this.state = {

msg: "Hello, from Guru99 Tutorials!"

}

}

render() {

return <h1>{this.state.msg}</h1>;

}

}

export default Hello;

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import Hello from './test.jsx';

ReactDOM.render(

<Hello />,

document.getElementById('root')

);

This is what we get when we test it in the browser:



## What are Props in ReactJS?

Props are properties to be used inside a component. They act as global object or variables which can be used inside the Component.

### Props to Function Component

Here is an example of passing props to a function component.

import React from 'react';

import ReactDOM from 'react-dom';

function Hello(props) {

return <h1>{props.msg}</h1>;

}

const Hello\_comp = <Hello msg="Hello, from Guru99 Tutorials!" />;

export default Hello\_comp;

As shown above, we have added **msg**attribute to <Hello /> Component. The same can be accessed as **props**inside Hello function, which is an object that will have the **msg**attribute details, and the same is used as an expression.

The component is used in index.js as follows:

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import Hello\_comp from './test.jsx';

ReactDOM.render(

Hello\_comp,

document.getElementById('root')

);

**Here is the output in the browser:**



### Props to Class Component

To access the props in a class we can do it as follows:

**test.jsx**

import React from 'react';

import ReactDOM from 'react-dom';

class Hello extends React.Component {

render() {

return <h1>{this.props.msg}</h1>;

}

}

export default Hello;

The **msg**attribute is passed to the component in index.js as follows:

import React from 'react';

import ReactDOM from 'react-dom';

import Hello from './test.jsx';

ReactDOM.render(

<Hello msg="Hello, from Guru99 Tutorials!" />,

document.getElementById('root')

);

**This is the output in the browser:**



**Also Check:-**[AngularJS Tutorial for Beginners: Learn AngularJS Step by Step](https://www.guru99.com/angularjs-tutorial.html)

## Life Cycle of a Component

A component life cycle is divided into Initialization, Mounting, Update, and UnMounting stages.

Here is a detail explanation about each Component.

A component in reactjs has the following stages :

**Initialization**: This is the first stage of the component life cycle.

Here it will have the default props and the state at the initial level.

**Mounting**: In this phase, the Component is rendered inside the dom. We having exposure to following methods in the mounting state.

* componentDidMount(): This is also called when the Component is just added to the dom.
* render(): You have this method for all the components created. It returns the Html node.

**Update**: In this state, the dom is interacted by a user and updated. For example, you enter something in the textbox, so the state properties are updated.

Following are the methods available in update state:

* shouldComponentUpdate() : called when the component is updated.
* componentDidUpdate(): after the component is updated.

**UnMounting:**this state comes into the picture when the Component is not required or removed.

Following are the methods available in unmount state:

Component willUnmount(): called when the Component is removed or destroyed.

## Working Example

Here is a working example which shows the methods called at each state.

**Example: complife.jsx**

import React from 'react';

import ReactDOM from 'react-dom';

class COMP\_LIFE extends React.Component {

constructor(props) {

super(props);

this.state = {name: ''};

this.UpdateName = this.UpdateName.bind(this);

this.testclick = this.testclick.bind(this);

}

UpdateName(event) {

this.setState({name: event.target.value});

}

testclick(event) {

alert("The name entered is: "+ this.state.name);

}

componentDidMount() {

console.log('Mounting State : calling method componentDidMount');

}

shouldComponentUpdate() {

console.log('Update State : calling method shouldComponentUpdate');

return true;

}

componentDidUpdate() {

console.log('Update State : calling method componentDidUpdate')

}

componentWillUnmount() {

console.log('UnMounting State : calling method componentWillUnmount');

}

render() {

return (

<div>

Enter Your Name:<input type="text" value={this.state.name} onChange={this.UpdateName} /><br/>

<h2>{this.state.name}</h2>

<input type="button" value="Click Here" onClick={this.testclick} />

</div>

);

}

}

export default COMP\_LIFE;

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import COMP\_LIFE from './complife.jsx';

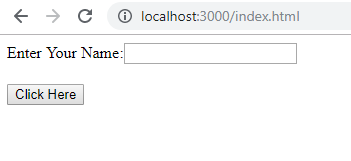
ReactDOM.render(

<COMP\_LIFE />,

document.getElementById('root')

);

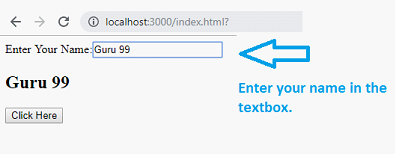
**When you check the output in the browser**



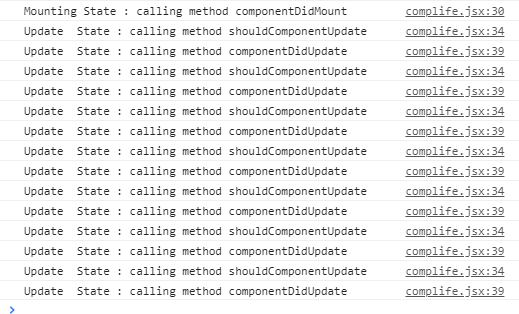
**In browser console you get :**

browser console Output

**When the user enters in the textbox:**



**In console following messages are displayed:**



## Working with Forms

In reactjs Html input elements like <input />, <textarea /> and <select /> has their own state and needs to be updated when user interacts using the setState() method.

In this chapter, we will see how to work with forms in reactjs.

**Here is a working example:**

**form.jsx**

import React from 'react';

import ReactDOM from 'react-dom';

class Form extends React.Component {

constructor(props) {

super(props);

this.state = {name: ''};

this.UpdateName = this.UpdateName.bind(this);

this.formSubmit = this.formSubmit.bind(this);

}

UpdateName(event) {

this.setState({name: event.target.value});

}

formSubmit(event) {

alert("The name entered is: "+ this.state.name);

}

render() {

return (

<form>

Enter Your Name:<input type="text" value={this.state.name} onChange={this.UpdateName} /><br/>

<h2>{this.state.name}</h2>

<input type="submit" value="Submit" onClick={this.formSubmit} />

</form>

);

}

}

export default Form;

For the input fields, we need to maintain the state, so for that react provides a special method called **setState,**which helps to maintain the state whenever there is a change.

We have used events onChange and onClick on the textbox and submit button. When the user enters inside the textbox the onChange event is called, and the name field inside state object state is updated as shown below:

UpdateName(event) {

this.setState({name: event.target.value});

}

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import Form from './form.jsx';

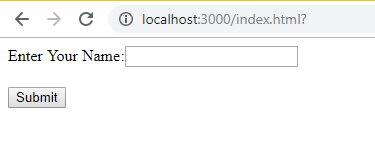
ReactDOM.render(

<Form />,

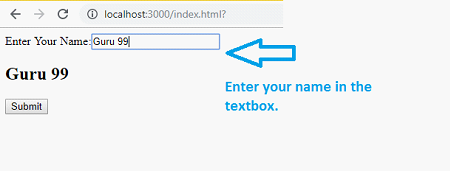
document.getElementById('root')

);

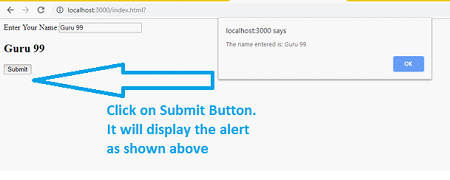
**The output in the browser is as follows:**



Step 1) Enter your name in the textbox:



1. Click on submit button



## Working with Events in ReactJS

Working with events in reactjs is same as how you would have done in javascript. You can use all the event handlers that are used in javascript. The setState() method is used to update the state when the user interacts with any Html element.

Here is a working example of how to use events in reactjs.

**events.jsx**

import React from 'react';

import ReactDOM from 'react-dom';

class EventTest extends React.Component {

constructor(props) {

super(props);

this.state = {name: ''};

this.UpdateName = this.UpdateName.bind(this);

this.testclick = this.testclick.bind(this);

}

UpdateName(event) {

this.setState({name: event.target.value});

}

testclick(event) {

alert("The name entered is: "+ this.state.name);

}

render() {

return (

<div>

Enter Your Name:<input type="text" value={this.state.name} onChange={this.UpdateName} /><br/>

<h2>{this.state.name}</h2>

<input type="button" value="Click Here" onClick={this.testclick} />

</div>

);

}

}

export default EventTest;

For the input fields, we need to maintain the state, so for that react provides a special method called **setState,**which helps to maintain the state whenever there is a change.

We have used events **onChange**and **onClick**on the textbox and button. When the user enters inside the textbox the **onChange**event is called, and the name field inside state object state is updated as shown below:

UpdateName(event) {

this.setState({name: event.target.value});

}

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import EventTest from './events.jsx';

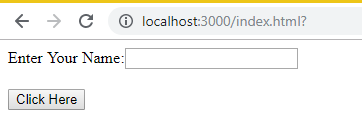
ReactDOM.render(

<EventTest />,

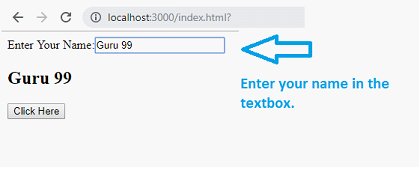
document.getElementById('root')

);

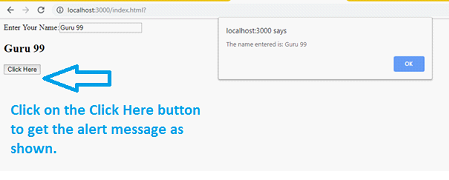
**Here is the output in the browser:**



**When a user enters the name :**



**When the user clicks on the Click Here button:**



## Working with Inline CSS in ReactJS

Will take a look at a working example to understand the working of inline css in reactjs.

**addstyle.jsx**

import React from 'react';

import ReactDOM from 'react-dom';

const h1Style = {

color: 'red'

};

function Hello(props) {

return <h1 style={h1Style}>{props.msg}</h1>;

}

const Hello\_comp = <Hello msg="Hello, from Guru99 Tutorials!" />;

export default Hello\_comp;

I have added color: ‘red’ style to the h1 tag.

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import Hello\_comp from './addstyle.jsx';

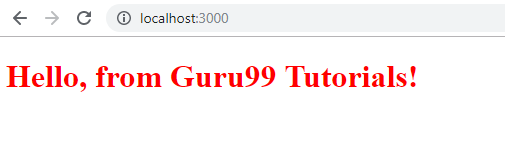
ReactDOM.render(

Hello\_comp,

document.getElementById('root')

);

**The output in the browser is as follows:**



You can create an object of style that you want on the element and use an expression to add style, as shown in the example above.

## Working with External CSS in ReactJS

Let us create a external css , for that create a folder css/ and add style.css in it.

**style.css**

.h1tag {

color:red;

}

**Add the style.css to your index.html file**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<title>ReactJS Demo</title>

<link href="css/style.css" rel="stylesheet" type="text/css" />

</head>

<body>

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

<script src = "out/script.min.js"></script>

</body>

</html>

Now let us add the class to the h1 tag in .jsx file

**addstyle.jsx**

import React from 'react';

import ReactDOM from 'react-dom';

let classforh1 = 'h1tag';

function Hello(props) {

return <h1 className={classforh1}>{props.msg}</h1>;

}

const Hello\_comp = <Hello msg="Hello, from Guru99 Tutorials!" />;

export default Hello\_comp;

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import Hello\_comp from './addstyle.jsx';

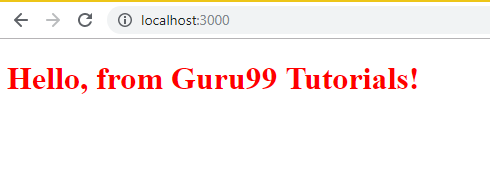
ReactDOM.render(

Hello\_comp,

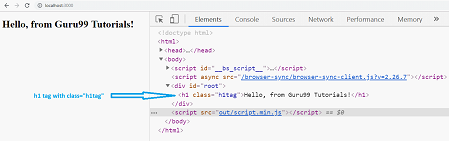
document.getElementById('root')

);

The className attribute is given the class details. Now let us test in the browser.



This is what you see when you inspect the h1 tag in browser:



You can see that the class=” h1tag” is added successfully to the h1 tag.

## Summary:

* ReactJS is an open-source front-end JavaScript library to build the user interface. It is maintained by Facebook and used by many companies today for UI development.
* The core features of ReactJS includes JSX, components(functional components and class-based components), the life cycle of a component, props, and state support for a component, working with javascript expressions.
* Project setup of ReactJS is explained using CDN files and also using npm packages to build the project.
* JSX is an extension to javascript. It is a template script where you will have the power of using Html and javascript together.
* Components are like pure javascript functions that help make the code easy by splitting the logic into reusable independent code.
* A state is a javascript object similar to props that have data to be used with the reactjs render. The state data is a private object and is used within components inside a class.
* Props are properties to be used inside a component.
* A component life cycle is divided into Initialization, Mounting, Update, and UnMounting stages.
* In reactjs html input elements like <input />, <textarea /> and <select /> has their own state and needs to be updated when user intereacts using the setState() method.
* Working with events in reactjs is same as how you would have done in javascript. You can use all the event handlers that are used in javascript. The setState() method is used to update the state when the user interacts with any Html element.
* ReactJS allows you to work with external css as well as inline css using javascript expression.