



# Introduction To React

Ms. Sonam Mittal

# Introduction to React

React is a javascript library for building the user interfaces.

- React is a Javascript library created by Jordan Walke, an engineer at Facebook, in 2011
- According to Jordan Walke, React is an efficient, declarative, and flexible open-source JavaScript library for building simple, fast, and scalable frontends of web applications.
- Today, there are over 220,000 live websites using React JS. Not only that, but industry giants like Apple, Netflix, Paypal, and many others have also already started using React JS in their software productions.

# What is React?

- React is a declarative, efficient, and flexible JavaScript library for building user interfaces.
- It lets you compose complex UIs from small and isolated pieces of code called “components”.
- React has a few different kinds of components, but we’ll start with React.Component subclasses.

```
class ShoppingList extends React.Component {  
  render() {  
    return (  
      <div className="shopping-list">  
        <h1>Shopping List for {this.props.name}</h1>  
        <ul>  
          <li>Instagram</li>  
          <li>WhatsApp</li>  
          <li>Oculus</li>  
        </ul>  
      </div>  
    ); } }  
}
```

- Use components to tell React what we want to see on the screen.
- When our data changes, React will efficiently update and re-render our components.
- Here, ShoppingList is a React component class. A component takes in parameters, called props (short for “properties”), and returns a hierarchy of views to display via the render method.
- The render method returns a description of what you want to see on the screen. React takes the description and displays the result.
- Most React developers use a special syntax called “JSX” which makes these structures easier to write.
- The `<div>` tag can be transformed at build time by `React.createElement('div')` method.

```
return React.createElement('div', {className: 'shopping-list'},  
  React.createElement('h1', /* ... h1 children ... */),  
  React.createElement('ul', /* ... ul children ... */)  
);
```

# Why React?

1. **Simplicity:** It Can be easily learnt using java Script and helps to build professional website. The component-based approach, well-defined lifecycle. React uses a special syntax called JSX and can also be used with JavaScript. But JSX is much easier to use.
2. **Easy to learn:** React can be understand with basic programming skills.. To react, you just need CSS and HTML.
3. **Native Approach:** It can also be used to create mobile applications (React Native). Extensive code reusability is supported. So at the same time with the same code you can design IOS, Android and Web applications.
4. **Data Binding:** It uses one-way data binding and an application architecture called Flux, controls the flow of data to components through one control point – the dispatcher.
5. **Easy to Debug:** It's easier to debug self-contained components of large ReactJS apps.
6. **Performance:** React does not offer any concept of a built-in container for dependency. You need to browse the code using JS, EcmaScript 6 modules which we can use via Babel. ReactJS-di to inject dependencies automatically.
7. **Testability:** Easy to test. We can easily manipulate with the state we pass to the ReactJS view and take a look at the output and triggered actions, events, functions, etc.

- A Single Page Application (SPA) is an application that allows you to work inside a browser and does not require reloading the page when a person is using it.
- Many of the apps we use every day are single page applications. Navigation apps, many social media platforms and some email providers are SPAs. Like, Gmail, Google Maps, AirBNB, Netflix, Pinterest, Paypal, and many more
- SPAs display stunning (User experience) UX that act like a browser. It does so by maintaining the minimum possible code, or “shell” of a page.
- This code is usually dependent on JavaScript frameworks, and when used ensures the high performance of the SPA.
- SPAs, due to the presence of their HTML shells, can easily be converted to progressive web applications (PWA) leading to a more seamless experience for the user.
- Likewise, their JavaScript-dependent shell also allows it to be able to load preloaded pages while offline. Known as offline caching, this allows users to never lose track of their data while on the app.
- The SPA only needs you to send one request, store all data, and then can freely access all of this data while offline.

- Native apps are applications that are installed through an app store such as Google or Apple's App Store.
- The advantage of SPAs is that they're Javascript-based.
- JavaScript forms the backbone of most native applications, making these apps easier to create and maintain. Because of that, technologies such as React Native can convert your cloud-based web application into a native app (software program that is developed for use on a particular platform or device) with ease.
- The beauty of SPAs is in their fluidity and ease of performance. Many SPAs have a frontend that is decoupled from its back end.
- SPAs use HTML and JavaScript features which serve as their frontend and a separate framework consequently serves as its functional back end. So if one fails, there's not much of an issue in performance.
- <https://media.giphy.com/media/52FwconEHyJH73EuFt/giphy.mp4>

- Multi-page applications, or MPAs, request rendering each time for a new page from the server in the browser.
- They're perfect for applications larger than SPAs, and due to the amount of content, they have different levels of UI. They are often known as the “traditional” way of app development.
- There are multiple levels of UI, this has recently been resolved due to the developments of AJAX.
- AJAX allows the transfer of a large amount of complex data between the servers and browsers, however; this also brings some problems.
- With its ability to transfer data, there's a new layer of complexity that often challenges JavaScript developers.
- most popular websites like Amazon, eBay have opted to remain as MPAs
- <https://media.giphy.com/media/Xpj7yINzSulzLa8Oq7/giphy.gif>



# SPA vs MPA – which one to choose?

- **Speed:** SPA's load faster because it loads the majority of app resources just once. The page doesn't reload entirely whenever the user requests a new piece of data. MPA is slower as the browser must reload the entire page from scratch whenever the user wants to access new data or moves to a different part of the website. The optimal loading time for a website is [0.4 seconds](#). If your website or app is image-heavy, then choosing a SPA is a safer option.
- **Coupling:** SPA is strongly decoupled, meaning that front-end and back-end are separate. Single-page applications use APIs developed by server-side developers to read and display data. In MPA's, front-end and back-end are more interdependent. All coding is usually housed under one project.
- **Search Engine Optimization:** One of the weaknesses of the SPA is SEO. Unfortunately, they aren't as SEO friendly as MPA's. It's primarily because most single-page applications are run on JavaScript, which most search engines do not support.
- **User experience:** SPA's are more mobile-friendly, and it's something worth remembering as a lot of traffic comes from mobile devices. Even Google started to prioritize mobile experience over the desktop. MPA's, enable better information architecture. You can create as many pages as required, and you can include as much information on a page as you need without any limits. Navigation is clear.
- **Security:** Larger the website, the more effort it takes to secure it. In MPA, you'll have to secure every webpage. SPA's are more prone to hacker attacks, as they run on JavaScript, which doesn't perform code compilation making it more vulnerable to malware.
- **Development process:** SPA's provides the reusable backend code. You can apply the same code you used in your web app to your native mobile app. As applications and websites are frequently used on mobile devices. The front-end and back-end, both parts can be developed simultaneously, which speeds up the entire development process. MPA's take longer to develop as in most cases, the server-side has to be coded from the beginning.
- **JavaScript dependency:** SPA lives and breathes JavaScript, which can be problematic. More search engines started to support JavaScript but with varying results. The level of support highly depends on the JS framework used. If the app is run on a browser with disabled JavaScript, it can cause app functionality problems, which might result in higher bounce rates and lower conversion. MPA's can be built without any JavaScript dependency.

- Both have pros and cons. SPA wins in terms of speed and code reusability, but it has deficiencies in SEO optimization. Using an MPA will help you rank higher in Google and is more scalable but much slower than SPA's.
- SPA's are better used in social networking applications, SaaS platforms – anywhere where SEO rankings aren't a deal-breaker.
- MPA's are best used in e-commerce apps, business catalogues, and marketplaces. If you're a large company that offers a wide variety of products, then an MPA is the best choice for you.

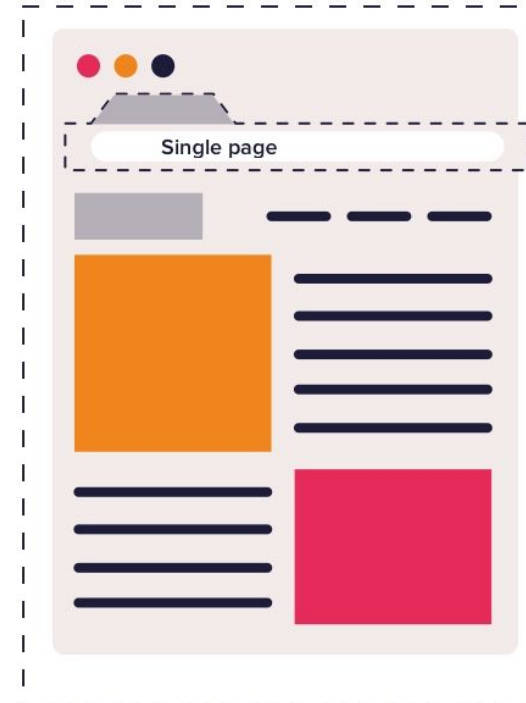
# SPA Vs. MPA



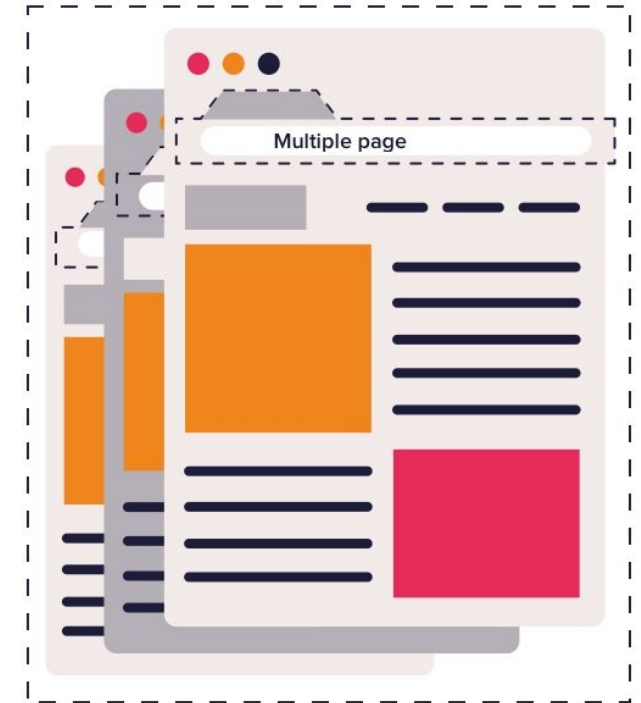
## SPA MPA

Sleek UX	✓
Easy SEO	✓
Security	✓
Less server load	✓
Offline functionality	✓
Mobile adaptability	✓
Application scalability	✓
UI/data separation	✓
Speed	✓
Fast launch	✓
Works without JavaScript	✓

## SP



## MP



- React developer tools are a series of extensions, frameworks, and libraries designed to simplify React development. Developers use testing utilities, code generators, debugging extensions, and other React tools to create more robust, more stable code while saving valuable development time.
- Many React developer tools work as extensions on browsers such as Chrome or Firefox.
- Perhaps “required” is too strong a word. The only developers who want to create better code and do it faster should use React tools. That goes for full-stack developers too!
- You can inspect and debug your application far easier and more conveniently.
- You can install developer tools seamlessly onto your browser, so you have a powerful resource within easy reach whenever you need it.

- There is an excellent selection of React developer tools to choose from, so let's explore the top dozen tools that every developer should know in 2021. All of these tools are open source.
- [Belle](#). Belle is a set of easy to use configurable React components. It enables programmers to quickly import any of these components to their applications: Button, Card & Select, ComboBox, Rating, TextInput, and Toggle.
- [BIT](#). BIT is a CLI tool created to solve problems arising from React components sharing. BIT lets you organize and distribute user interface (UI) components with your team members. You can also use departed components anywhere needed.
- [Create React App](#). This single command-line tool was created by Facebook, helping developers speed up setting up an environment for new React projects. Create React App offers a frontend build pipeline, arranges the developer's environment, and optimizes the application for subsequent production. Consequently, developers need not spend time with configuration tasks.
- [Evergreen](#). Evergreen is an out-of-the-box UI framework for React that is recognized for its extensive documentation. Evergreen boasts a wide selection of ready-to-use components, though it allows for customization.
- [Gatsby](#). This React-based framework helps developers build light and fast applications and websites. Gatsby works with many different data sources, such as Markdown files, CMS like Contentful or WordPress, REST, or GraphQL API, so you can easily manage content.

- [Jest](#). Facebook created this JavaScript testing framework to test React components. It's a versatile framework, working for other JS solutions such as Angular, Babel, Node, TypeScript, and Vue. Since Jest was developed by the same folks who created React, it's the best choice for testing.
- [React 360](#). React 360 creates interactive 360 experiences designed to run in web browsers. It brings together React's propositional power with modern APIs like WebGL and WebVR to help developers create applications that can be accessed via different types of devices. React 360 takes advantage of both the robust React ecosystem and modern web technologies to simplify the creation of cross-platform 360 experiences.
- [React Proto](#). Proto is short for "prototype." This tool lets developers create an application architecture backward. Developers begin with a visual design, and Proto provides the application files required for further development. It then helps developers define the props in ReactJS and states.
- [React Sight](#). This React tool provides developers with a visual representation of the React app structure. Users must first to install React Developer Tools for Chrome. Consequently, you must also add it as a Chrome, which then adds a new "React Sight" panel to DevTools. React Sight also provides support for React Router and Redux.
- [Redux](#). Redux is a highly popular JavaScript container that holds an application's state in a store that allows any component to access and use it. Redux also offers the Redux Toolkit, useful for programmers who want to write Redux logic easily.

- [Rekit](#). Rekit is a complete toolkit, an all-in-one solution designed for cutting-edge React applications. Rekit creates apps and provides programmers with project management tools such as Rekit Studio. This tool comes with a convenient command line interface and tools that manage actions, components, pages, and reducers.
- [Storybook](#). Storybook is designed for user interfaces (UI). Developers use it to create, develop, and test UI components, providing you with both a UI component playground and a development environment. Storybook lets developers benefit from the UI component development environment and provides a means of quickly testing and displaying them.

# Top websites using React JS



Top Websites Using React JS	Website Monthly Visits
Yandex.ru	3.3 billion
yahoo.com	1.3 billion
cnn.com	695.7 million
bbc.co.uk	625.8 million
paypal.com	580.2 million
roblox.com	519.8 million
amazon.co.uk	437.7 million



# Advantages of React JS

## **Speed**

The React basically allows developers to utilize individual parts of their application on both client-side and the server-side, which ultimately boosts the speed of the development process.

## **Flexibility**

Compared to other frontend frameworks, the React code is easier to maintain and is flexible due to its modular structure. This flexibility, in turn, saves huge amount of time and cost to businesses.

## **Performance**

React JS was designed to provide high performance in mind. The core of the framework offers a virtual DOM program and server-side rendering, which makes complex apps run extremely fast.

## **Usability**

Deploying React is fairly easy to accomplish if you have some basic knowledge of JavaScript. In fact, an expert JavaScript developer can easily learn all ins and outs of the React framework in a matter of a day or two.

# Advantages of React JS



## Reusable Components

React JS has potential to reuse its components. It saves time for developers as they don't have to write various codes for the same features. Furthermore, if any changes are made in any particular part, it will not affect other parts of the application.

## Simplified Scripting

React JS features a free syntax extension called JSX. This makes your HTML markup within the library much easier. Its writing shortcuts allow you to make your code simpler and cleaner, converting your HTML mockups into ReactElement trees. JSX not only helps avert code injections, but it makes your whole application run faster.

## Ease of Maintenance

When an app has complicated logic, making changes after the fact can be a long, painful, and expensive process. But one of the key ReactJS advantages is its modular design. This makes it easy to make small changes to small components without disturbing the others. Programmers are also able to reuse assets, re-employing the same object, which also saves enormous amounts of programming and maintenance time.

# Create-react-app

Create React App is an officially supported way to create single-page React applications. It offers a modern build setup with no configuration. The Create React App is maintained by **Facebook** and can work on any **platform**, for example, macOS, Windows, Linux, etc.

## Requirements

To create a React Project using create-react-app, you need to have installed the following things in your system.

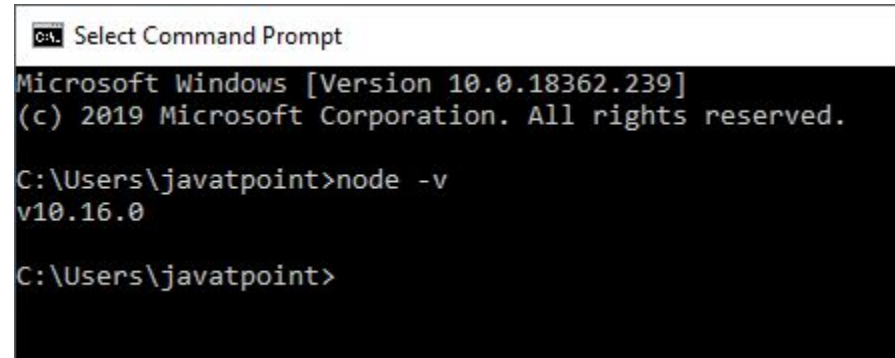
Node version  $\geq 8.10$

NPM(Node Package Manager) version  $\geq 5.6$

Let us check the current version of **Node** and **NPM** in the system.

1. Run the following command to check the Node version in the command prompt.

**\$ node -v**



```
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

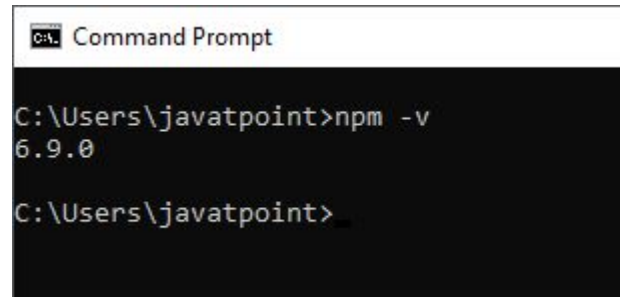
C:\Users\javatpoint>node -v
v10.16.0

C:\Users\javatpoint>
```

# Installing React Developer tool

2. Run the following command to check the NPM version in the command prompt.

**\$ npm -v**



```
Command Prompt
C:\Users\javatpoint>npm -v
6.9.0
C:\Users\javatpoint>
```

## Installation

Here, we are going to learn how we can install React using **CRA** (Create-react-app ) tool. For this, we need to follow the steps as given below.

## Install React

We can install React using npm package manager by using the following command. There is no need to worry about the complexity of React installation. The create-react-app npm package manager will manage everything, which needed for React project.

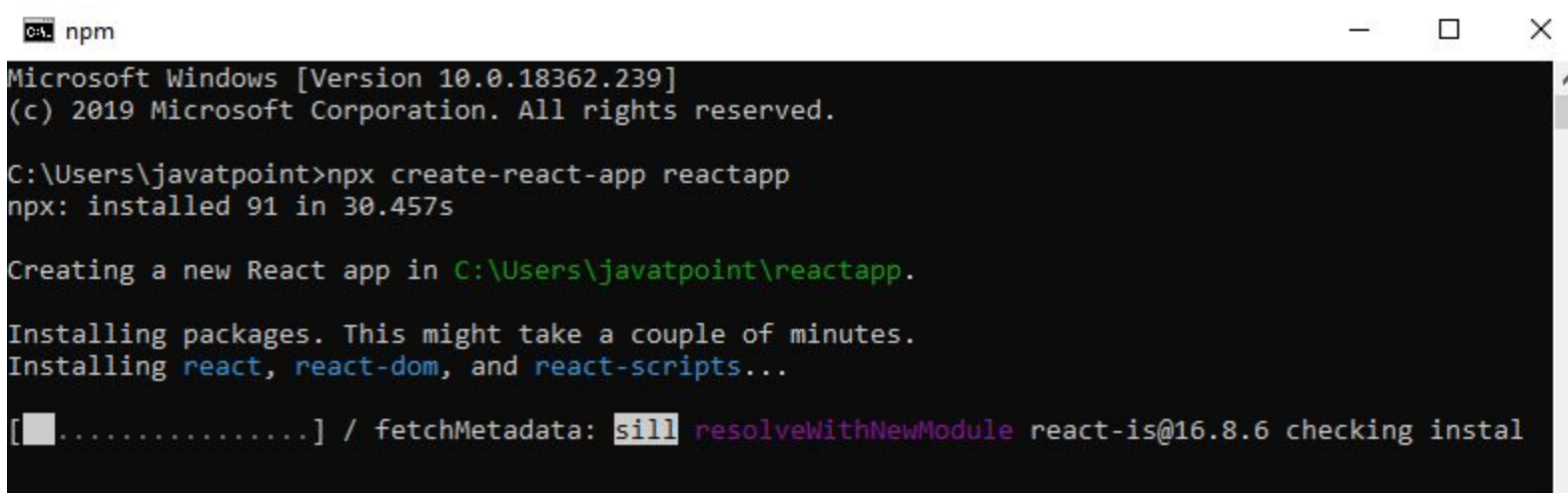
```
C:\Users\javatpoint> npm install -g create-react-app
```

# Create 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.

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

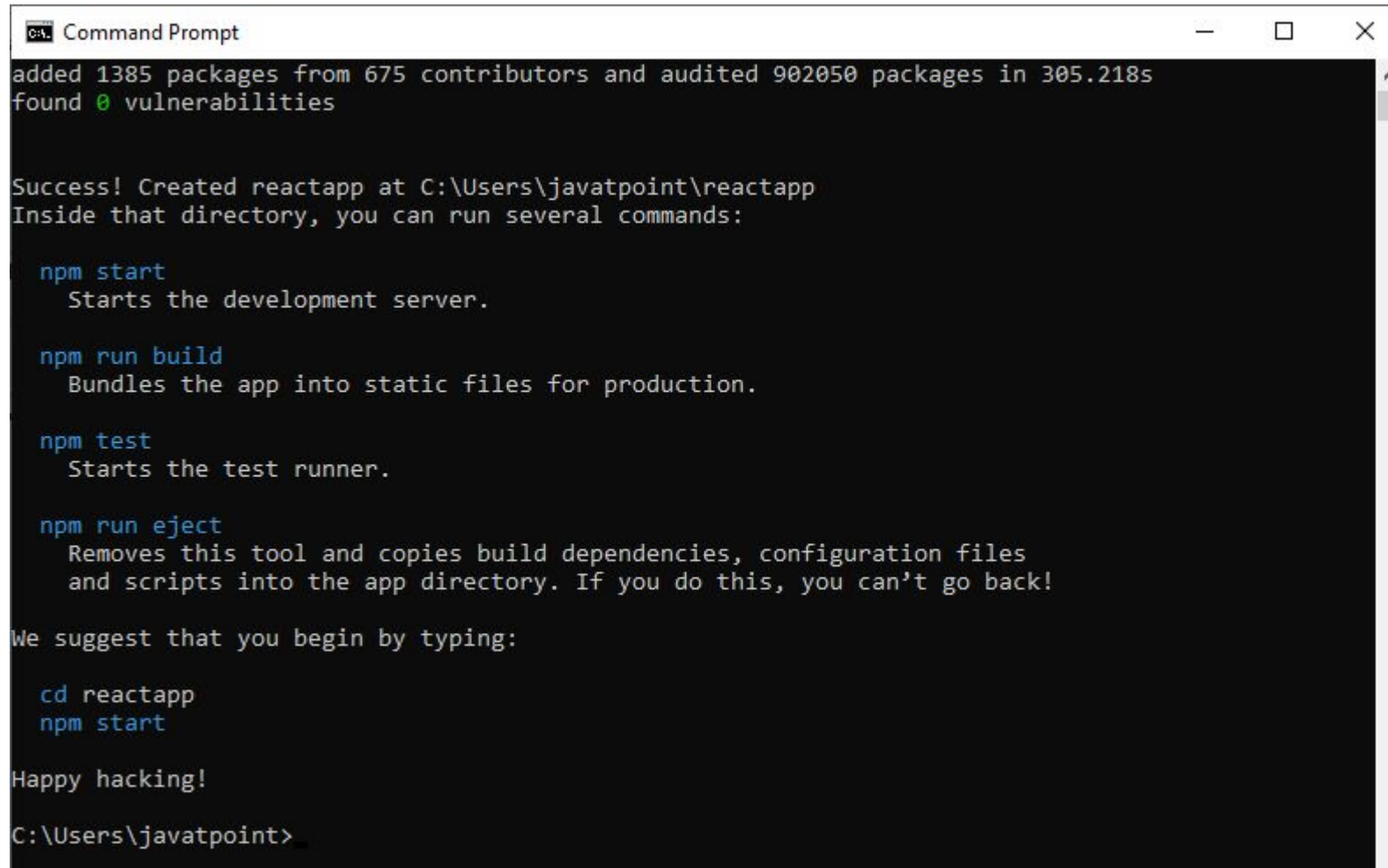


```
npm  
Microsoft Windows [Version 10.0.18362.239]  
(c) 2019 Microsoft Corporation. All rights reserved.  
  
C:\Users\javatpoint>npx create-react-app reactapp  
npx: installed 91 in 30.457s  
  
Creating a new React app in C:\Users\javatpoint\reactapp.  
  
Installing packages. This might take a couple of minutes.  
Installing react, react-dom, and react-scripts...  
[.....] / fetchMetadata: sill resolveWithNewModule react-is@16.8.6 checking instal
```

The above command will take some time to install the React and create a new project with the name "reactproject."

# Contd..

Now, we can see the terminal as like below.



```
Command Prompt
added 1385 packages from 675 contributors and audited 902050 packages in 305.218s
found 0 vulnerabilities

Success! Created reactapp at C:\Users\javatpoint\reactapp
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 reactapp
  npm start

Happy hacking!
C:\Users\javatpoint>
```

# Steps for Recat Installation

- <https://nodejs.org/en/download/current/> : for nodejs installation
- `node -v` : to check version of nodejs
- `npm -v` :to check version of npm
- `npx create-react-app my-app` : to create react app folder
- `cd my-app`
- `npm start`

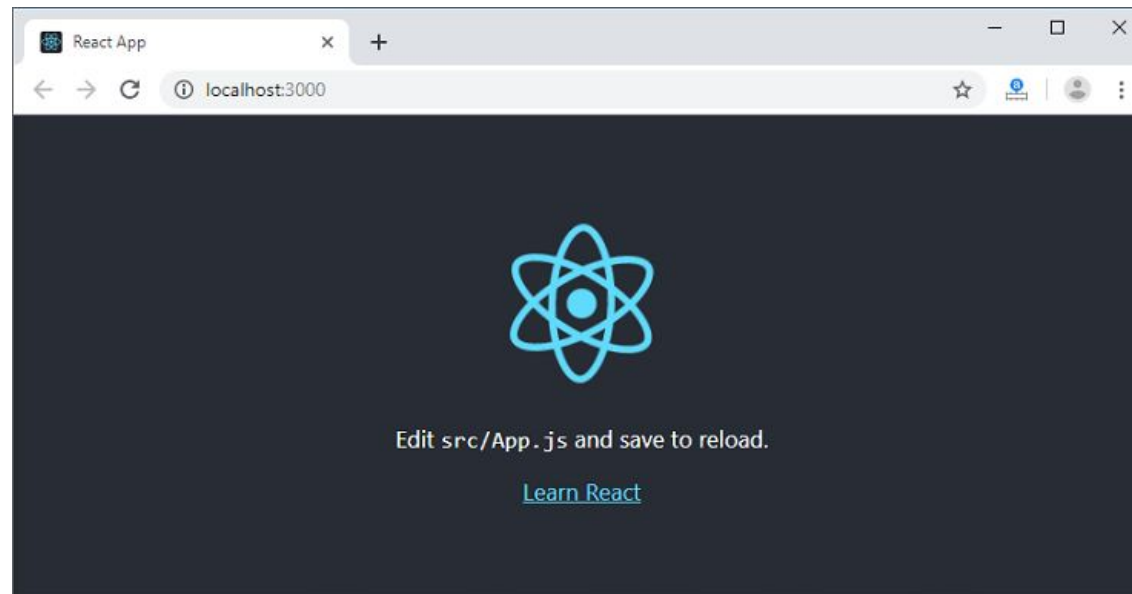
# Contd..

To start the server so that we can access the application on the browser. Type the following command in the terminal window.

```
$ cd my app
```

```
$ npm start
```

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





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.

For the environment setup of react js, 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.

Then, 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.

```
$ npm build
```

# Folder Structure

After creation, the project should look like this:

**my-app/**

README.md

node\_modules/

package.json

public/

index.html

favicon.ico

**src/**

App.css

App.js

App.test.js

index.css

index.js

logo.svg

For the project to build, these files must exist with exact filenames:

- public/index.html is the page template;
- src/index.js is the JavaScript entry point.

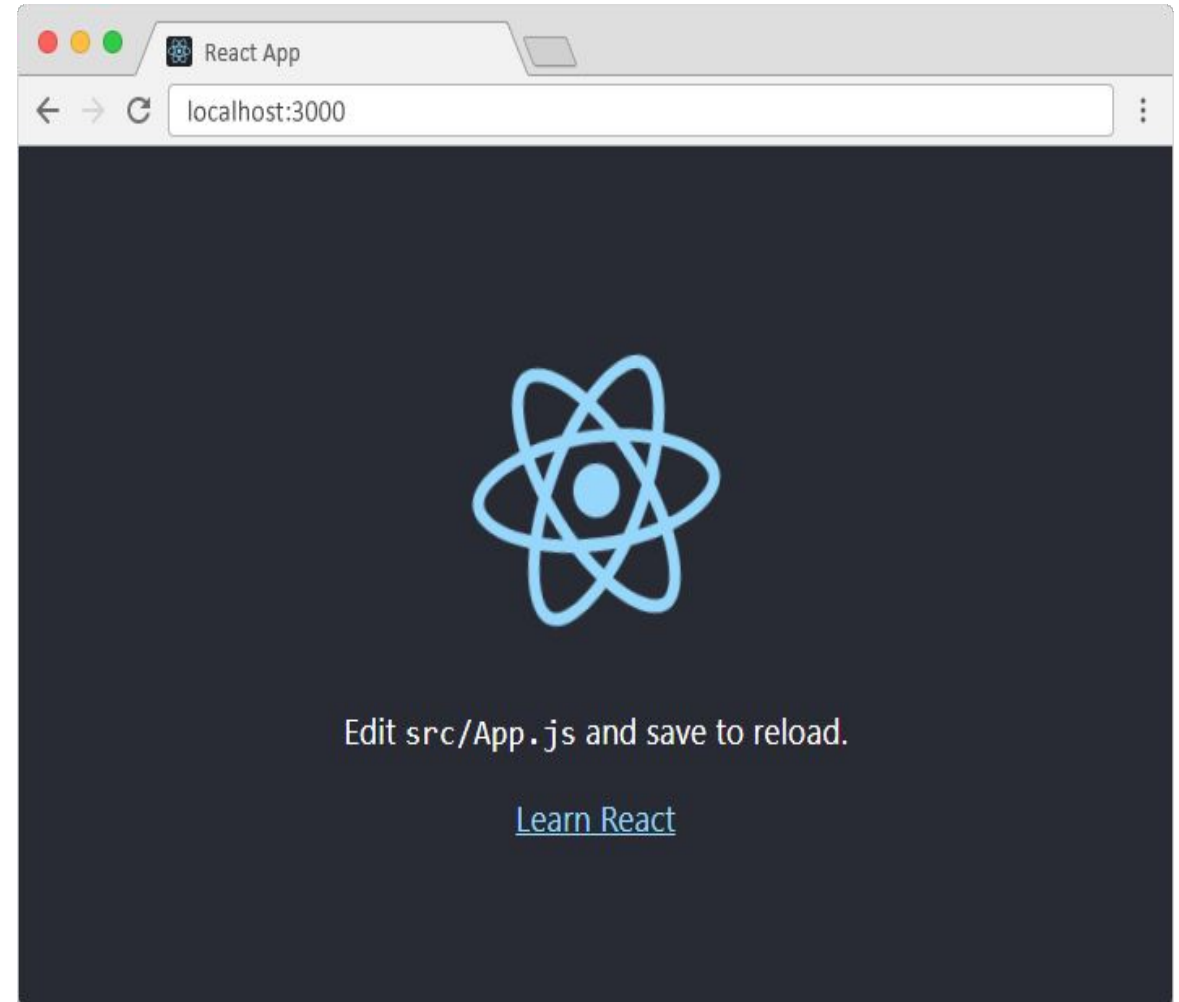
# Getting start with react



- Open App.js file in src folder having content as given below:

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );} export default App;
```



# App1.js and index.js file (changes to be done)

## App1.js File

```
import logo from './logo.svg';
import './App.css';

function App1() {
  return (
    <div className="App1">
      <h1>Hello World!</h1>
    </div>
  );
}

export default App1;
```

## Index.js File

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App1 from './App1';
import reportWebVitals from './reportWebVitals';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App1 />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function// to
// log results (for example: reportWebVitals(console.log))// or send to an
// analytics endpoint. Learn more: https://bit.ly/CRA-vitalsreportWebVitals();
```

# Understanding 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.
- Example the App component in which we were returning the HTML elements i.e.

```
import React, { Component } from 'react';
import './App.css';
class App extends Component {
  // JSX Content
  render() {
    return (
      <div className="App">
        { /* Other Component can be nested here */ }
        <h1>Root Of Application</h1>
      </div>
    )
  }
}
export default App;
```

# Understanding JSX



- React components have a **render** function. The render function specifies the HTML output of a React component.
- JSX(JavaScript Extension), is a React extension which allows writing JavaScript code that looks like HTML.
- JSX is an HTML-like syntax used by React that extends ECMAScript so that **HTML-like** syntax can co-exist with JavaScript/React code. The syntax is used by **preprocessors** (i.e., transpilers like babel) to transform HTML-like syntax into standard JavaScript objects that a JavaScript engine will parse.
- JSX provides you to write HTML/XML-like structures (e.g., DOM-like tree structures) in the same file where you write JavaScript code, then preprocessor will transform these expressions into actual JavaScript code. Just like XML/HTML, JSX tags have a tag name, attributes, and children.



# Use of 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.
- It is type-safe, and most of the errors can be found at compilation time.
- It makes easier to create templates.

# Nested Elements in JSX

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

## App.JSX

```
import React, { Component } from 'react';
class App extends Component{
  render(){
    return(
      <div>
        <h1>React JS</h1>
        <h2>Training of JSX</h2>
        <p>This website contains the best CS tutorials.</p>
      </div>
    );
  }
}
export default App;
```

# JSX Attributes

JSX use attributes with the HTML elements same as regular HTML.

JSX uses standard naming convention of HTML such as a class in HTML becomes **className** in JSX because the class is the reserved keyword in JavaScript. We can also use our own custom attributes in JSX.

For custom attributes, we need to use **data- prefix**. In the below example, we have used a custom attribute **data-demoAttribute** as an attribute for the **<p>** tag.

```
import React, { Component } from 'react';
class App extends Component{
  render(){
    return(
      <div>
        <h1>React JS</h1>
        <h2>Training of JSX</h2>
        <p data-demoAttribute = "demo">This website contains the best CS tutorials.</p>
      </div>
    );
  }
}
export default App;
```

# JSX Restrictions

Some restrictions we face for example className.

## ClassName:

1. We cannot use class attribute in the JSX because javascript itself **reserve the word**(class). In application we already use to create the class this is why we have to use className.
2. All the HTML elements we are using in the render() method are managed by the React Library.
3. We are not using the real HTML text, React is converting the behind the scene. React finds the attributes in quotation marks we can define on all these elements.

We cannot return same elements for example;

```
import React, { Component } from 'react';
import './App.css';
class App extends Component {
  // JSX Content
  render() {
    return (
      <div className="App">
        <h1>Root Of Application</h1>
      </div>
      <h1>Root Of Application</h1> { /* Cannot add same element twice */ }
    )
  }
}
export default App;
```

# What are components?

- React component is the building block of a React application.
- A React component represents a small part of user interface in a webpage.
- The primary job of a React component is to render its user interface and update it whenever its internal state is changed.
- Rendering of the UI, manages the events belongs to its user interface.
- React component provides below functionalities.
  - Initial rendering of the user interface.
  - Management and handling of events.
  - Updating the user interface whenever the internal state is changed.
- React component accomplish these feature using three concepts –
  - **Properties** – Enables the component to receive input.
  - **Events** – Enable the component to manage DOM events and end-user interaction.
  - **State** – Enable the component to stay stateful which updates its UI with respect to its state.

# React Components

**In React Js, there are mainly two types of components;**

1. Functional Components
2. Class Components

## **Functional Components:**

In React, function components are a way to write components that only contain a render method and don't have their own state. They are simply JavaScript functions that may or may not receive data as parameters. We can create a function that takes props(properties) as input and returns what should be rendered. A valid functional component can be shown in the below example.

```
function WelcomeMessage(props) {  
  return <h1>Welcome to the , {props.name}</h1>;  
}
```

# React Components

## Class Components:

Class components are more complex than functional components. It requires you to extend from `React.Component` and create a render function which returns a React element. You can pass data from one class to other class components. You can create a class by defining a class that extends `Component` and has a render function. Valid class component is shown in the below example.

```
class MyComponent extends React.Component {  
  render() {  
    return (  
      <div>This is main component.</div>  
    );  
  }  
}
```

AWD

```
//function component  
function Hello() {  
  return <div>Hello</div>  
}
```

The same functionality can be done using ES6 class component with little extra coding.

```
//Class component  
import React, {Component} from 'react';  
class App2 extends React.Component {  
  render() {  
    return (  
      <div>Hello, I am a Class Component</div>  
    );  
  }  
}
```



- Class components supports state management out of the box whereas function components does not support state management. But, React provides a hook, *useState()* for the function components to maintain its state.
- Class component have a life cycle and access to each life cycle events through dedicated callback apis. Function component does not have life cycle. Again, React provides a hook, *useEffect()* for the function component to access different stages of the component.

# Reusable Components

Reusable components can be **requirements specifications, design documents, source code, user interfaces, user documentation, or any other items associated with software**. All products resulting from the software development life cycle have the potential for reuse.

For example;

```
src/App.js

import React, { Component } from 'react';
import './App.css';
import Person from './Person/Person';
class App extends Component {
  render() {
    return (
      <div className="App">
        <h1> Heading</h1>
        <p> Paragraph </p>
        {/* Reusing the same component Multiple time. */}
        <Person />
        <Person />
        <Person />
      </div>
    );
  }
}
export default App;
```

# Working with Components & Re-using them

If our Application contains more and more components and use them wherever we need to use in our app.

`<Person />`: This is now effectively our custom HTML element. We also can configure it though before we do that, let's change something else about our react code because right now is all static, still we have our custom component but in there we are still using some static HTML in the end.

```
src/Person/Person.js

import React from 'react';
const person = () => {
  return <p> I am a Person</p>;
};
export default person;
```

JSX code should be dynamic and it should output different thing depending on the state of our application or on some user input.

# Example

```
import logo from './logo.svg';
import './App.css';

import React, {Component} from 'react';
import Person from './Person/Person';
class App2 extends React.Component {
  render() {
    return (
      <div>Hello, I am a Class Component
        <Person/>
        <Person/>
        <Person/>
        <Person/>
      </div>
    );
  }
}
export default App2;
```

```
//Person.js file in Person folder in src folder
import React, {Component} from 'react';

const Person=()=>{
  return <p> I am a person </p>;
};
export default Person;
```

`<p> I am a Person and I am X year old! </p>` but X should actually be a random number, and we can simply do that. We can replace X with a random number.

## Src/Person/Person.js

```
import React from 'react';  
const person = () => {  
    return <p> I am a Person and i am Math.floor(Math.random() * 30) year old!</p>;  
};  
export default person;
```

## To execute;

`Math.floor(Math.random() * 30) //as javascript`

If we have some dynamic content in our JSX part which we want to run as javascript code and not interpret as text.

For that we have to wrap into single curly braces i.e. `{ /* Javascript code */}`

## such as

```
return <p> I am a Person and i am {Math.floor(Math.random() * 30)} year old!</p>;
```

Props are variables which are used to pass data between React components by its parent component.

React's data flow between components is unidirectional (from parent to child only).

## How to pass data with props?

```
class ParentComponent extends Component {  
  render()  
  {  
    return (  
      <ChildComponent name="First Child" />  
    );  
  }  
}
```

```
const ChildComponent = (props) => {  
  return <p>{props.name}</p>;  
};
```

Firstly, we need to define/get some data from the parent component and assign it to a child component's "prop" attribute.

```
<ChildComponent name="First Child" />
```

"Name" is a defined prop here and contains text data. Then we can pass data with props like we are giving an argument to a function:

```
const ChildComponent = (props) => {  
  // statements  
};
```

And finally, we use dot notation to access the prop data and render it:

```
return <p>{props.name}</p>;
```

# Working with Props

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.

## Src/App.js

```
import React, { Component } from 'react';
import './App.css';
import Person from './Person/Person';
class App extends Component {
  render() {
    return (
      <div className="App">
        <h1> Heading</h1>
        <p> Paragraph </p>
        { /* Setting Attribute */ }
        <Person name="Lucy" age="23" />
        <Person name="Max" age="12">
          My Hobbies : Dancing { /* Some Additional Data */ }
        </Person>
        <Person name="Mike" age="34" />
      </div>
    );
  }
}
export default App;
```



# Using React Properties

The Person component can take the attribute and give us access inside the receiving component on object named props.

In javascript file;

```
import React from 'react';
const person = (props) => {
  return <p></p>;
};
export default person;
```

One argument which is passed into it by default in React which is an object with all the properties of this component.

We have props through which we can get access for that name and age.

In javascript file;

```
import React from 'react';
const person = props => {
  return (
    <p>
      I am a {props.name} and i am {props.age} year old!
    </p>
  );
};
export default person;
```

# Understanding the “Children” Props

## Children Property:

Now we also want to output whatever we pass between the opening and closing tag of our custom component.

```
import React, { Component } from 'react';
import './App.css';
import Person from './Person/Person';
class App extends Component {
  render() {
    return (
      <div className="App">
        <Person name="Max" age="12">
          My Hobbies : Dancing { /* Some Additional Data */ }
        </Person>
      </div>
    );
  }
}
export default App;
```

# Understanding the “Children” Props

## Children Property:

In the Person component, where we want to receive it in the end will wrap our paragraph in normal parentheses.

```
import React from 'react';
const person = props => {
  return (
    <div>
      <p>
        I am a {props.name} and i am {props.age} year old!{' '}
      </p>
      <p> {props.children} </p>
    </div>
  );
};
export default person;
```

New p element inside the container div element. The second p element will output the content which we have passed between the Person element in src/App.js.

# Understanding Using State

The state in a component can change over time. The change in state over time can happen as a response to user action or system event. 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. For example, In **App.js** file,

```
import React, { Component } from 'react';
import './App.css';
import Person from './Person/Person';
class App extends Component {
  render() {
    return (
      <div className="App">
        <h1> Heading</h1>
        <button>Switch Name</button>
        <Person name="Lucy" age="23" />
        <Person name="Max" age="12">
          My Hobbies : Dancing
        </Person>
        <Person name="Mike" age="34" />
      </div>
    );
  }
}
export default App;
```

# Understanding Using State

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 the state. The state property can be rendered inside render() method.

```
import React, { Component } from 'react';
import './App.css';
import Person from './Person/Person';
class App extends Component {
  state = {
    persons: [
      {name: 'Lucy', age: 23},
      {name: 'Max', age: 12},
      {name: 'Mike', age: 34},
    ]
  }
  ...
}
export default App;
```

# Example

```
import logo from './logo.svg';
import './App.css';
import React, {Component} from 'react';
import Person from './Person/Person';
class App2 extends React.Component {
  state={
    person:[ {name: 'Sonam'}, {name: 'Soni'}, {name: 'Shagun'}]
  }
  render() {
    return (
      <div>Hello, I am a Class Component
        <h1> states</h1>
        <button> CLICK</button>
        <Person name={this.state.person[1].name}/>
        My hobbies: Reading

      </div>

    );
  }
}
export default App2;
```

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

const Person=props=>{
  return <p> I am a Person having name
{props.name} and i am
{Math.floor(Math.random() * 30)} year old!</p>;
  <p>{props.children}</p>
};
export default Person;
```

React has another special built-in object called state, which allows components to create and manage their own data. So unlike props, components cannot pass data with state, but they can create and manage it internally.

The state can be initialized by props.

## How to use State?

```
import logo from './logo.svg';
import './App.css';
import React, {Component} from 'react';
class App2 extends React.Component {
  constructor(props) {
    super(props);
    this.state={
      id: "1",
      name: "test"
    };
  }
  render() {
    return (
      <div>
        <p> Hello my id is: {this.state.id}</p>
        <p>{this.state.name}</p> </div>);
    }
  }
  export default App2;
```

State should not be modified directly, but it can be modified with a special method called `setState( )`.

```
this.state.id = "2020";    // wrong  
this.setState({          // correct  
id: "2020"  
});
```



# Example.....

```
import logo from './logo.svg';
import './App.css';
import React, {Component} from 'react';
class App2 extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
    this.changeColor = () => { this.setState({color: "blue"}); }
    render() {
      return (
        <div>
          <h1>My {this.state.brand}</h1>
          <p> It is a {this.state.color}
            {this.state.model}
            from {this.state.year}. </p>
          <button type="button" onClick={this.changeColor} >Change color</button>
        </div> );
    }
  }
  export default App2
```

# Props vs State

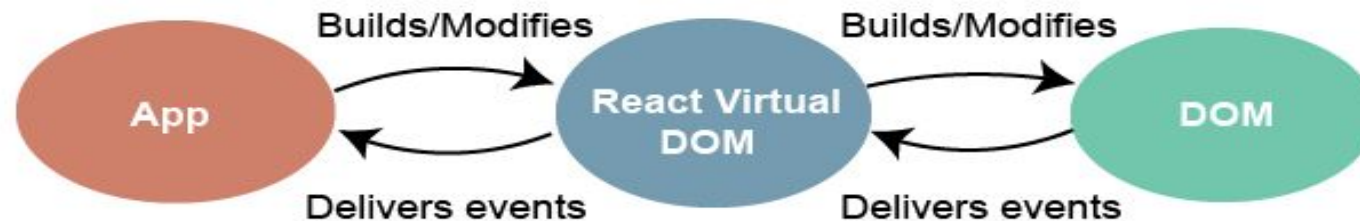


SN	Props	State
1.	Props are read-only.	State changes can be asynchronous.
2.	Props are immutable.	State is mutable.
3.	Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
4.	Props can be accessed by the child component.	State cannot be accessed by child components.
5.	Props are used to communicate between components.	States can be used for rendering dynamic changes with the component.
6.	Stateless component can have Props.	Stateless components cannot have State.
7.	Props make components reusable.	State cannot make components reusable.
8.	Props are external and controlled by whatever renders the component.	The State is internal and controlled by the React Component itself.

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.

## Events Handler





## Event Declaration in React:

- React event handlers are written inside curly braces.
- React events are named as **camelCase** instead of **lowercase**.

```
<button onClick={showMessage}>  
  Hello JavaTpoint  
</button>
```

## Examples:

```
//Example 1.
import React, {Component} from 'react';
import Person from './Person/Person';

function App2() {
  const shoot = () => {
    alert("Great Shot!");
  }
  return <div>
    <button onClick={shoot}>Take the
    shot!</button>
  </div>
}
export default App2;
```

```
//Example 2
import logo from './logo.svg';
import './App.css';
import React, {Component} from 'react';
import Person from './Person/Person';
function App2() {
  function addUser(event){
    event.preventDefault();
    console.log('Add user event clicked');
  }
  return <div>
    <button onClick={ addUser }>
    Add User </button>
  </div>
}
export default App2;
```

# Function in class component

```
import logo from './logo.svg';
import './App.css';
import React, {Component} from 'react';
import Person from './Person/Person';
class App2 extends React.Component {
  saySomething(something) {
    console.log(something);
  }

  handleClick(e) {
    this.saySomething("element clicked");
  }

  componentDidMount() {
    this.saySomething("component did mount");
  }

  render(){
    return( <div>
    <button onClick={this.handleClick.bind(this)} value="Click me" > CLICK
    ME</button>
    </div> ); } }
export default App2;
```

```
// parameterised function in an event
import logo from './logo.svg';
import './App.css';

import React, {Component} from 'react';
import Person from './Person/Person';
function App2() {
  const shoot = (a, b) => {
    alert(b.type);
    document.write("my ", a); }
  return (
    <button onClick={ (event) =>
    shoot("Goal!", event)}>Take the
    shot!</button>
  );
}
export default App2;
```

# Life cycle methods for class components

```
class App2 extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {date: new Date()};  }  
  componentDidMount() {  
    this.timerID = setInterval(  
      () => this.tick(), 1000  
    ); }  
  componentWillUnmount() {  
    clearInterval(this.timerID);  
  }  
  tick() {  
    this.setState({  
      date: new Date()  
    }); }  
  render() {  
    return (  
      <div>  
        <h1>Hello, world!</h1>  
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>  
      </div>  
    ); } }  
}
```

- We want to set up a timer whenever the Clock is rendered to the DOM for the first time. This is called “mounting” in React using `componentDidMount` method.
- We also want to clear that timer whenever the DOM produced by the Clock is removed. This is called “unmounting” in React using `componentWillUnmount` method.

Let's quickly recap what's going on and the order in which the methods are called:

- When `<Clock />` is passed to `root.render()`, React calls the constructor of the Clock component. Since Clock needs to display the current time, it initializes `this.state` with an object including the current time. We will later update this state.
- React then calls the Clock component's `render()` method. This is how React learns what should be displayed on the screen. React then updates the DOM to match the Clock's render output.
- When the Clock output is inserted in the DOM, React calls the `componentDidMount()` lifecycle method. Inside it, the Clock component asks the browser to set up a timer to call the component's `tick()` method once a second.
- Every second the browser calls the `tick()` method. Inside it, the Clock component schedules a UI update by calling `setState()` with an object containing the current time. Thanks to the `setState()` call, React knows the state has changed, and calls the `render()` method again to learn what should be on the screen. This time, `this.state.date` in the `render()` method will be different, and so the render output will include the updated time. React updates the DOM accordingly.
- If the Clock component is ever removed from the DOM, React calls the `componentWillUnmount()` lifecycle method so the timer is stopped.



## Adding an Event Listener

You can create an event listener in a React app by using the `window.addEventListener` method:

```
1 import React from 'react';
2
3 const App = (props) => {
4   window.addEventListener('keydown', (event) => {
5     // ...
6   });
7   return (
8     <div className='container'>
9       <h1>Welcome to the Keydown Listening Component</h1>
10    </div>
11  );
12 };
```

There are a couple of problems with the code above.

1. You only want the event listener added when the component finishes rendering.
2. You need to remove the event listener when the component is unmounted.

# Event Listener in React in class and functional Component

Now the **addEventListener** is being called at the correct time, but there is still a problem: the way it is written now will add a new listener on all component changes. This is because you need to tell the **useEffect** hook to only run when the component first renders. To do this you can simply add an empty dependency array as the second argument of **useEffect**:

```
class App2 extends React.Component {  
  componentDidMount() {  
    window.addEventListener ('click', (event)=>{  
      console.log("Shoot ME");  
    });  
  }  
  render(){  
    return( <div>  
      <p> Welcome to my page </p>  
    </div> );  
  }  
}  
export default App2;
```

```
import logo from './logo.svg';  
import './App.css';  
import React, {Component} from 'react';  
import Person from './Person/Person';  
  
const App2 = (props) => {  
  React.useEffect(() => {  
    window.addEventListener ('keydown', (event)=>{  
      console.log("Shoot ME");  
    });  
  }, []);  
  
  return (  
    <div> <p> welcome to my page</p>  
    </div>  
  );  
}
```

# Removing an Event Listener



Now that you've added the event listener to your component, you will also need to know how to remove that listener when the component is no longer mounted. The function for removing an event listener is `window.removeEventListener`.

## 1. `window.removeEventListener('keydown', handlekeyDown);`

You'll notice that the second argument to `removeEventListener` is not an anonymous function, and is instead a variable. This is necessary for identifying what event listener to remove, since technically you can have multiple `keydown` event listeners running at the same time.

Now you just need to call `removeEventListener` when the component unmounts, but how do you do that in a functional component? There is no `componentWillUnmount` lifecycle method, but the `useEffect` hook helps you out with an optional return value in the callback function. You can return a function in the `useEffect` callback and that function will run when the component unmounts.

Here is what your code should look like after successfully removing an event listener:

```
1  const App = (props) => {
2    const handleKeyDown = (event) => {
3      console.log('A key was pressed', event.keyCode);
4    };
5
6    React.useEffect(() => {
7      window.addEventListener('keydown', handleKeyDown);
8
9      // cleanup this component
10     return () => {
11       window.removeEventListener('keydown', handleKeyDown);
12     };
13   }, []);
14
15   return (
16     <div className='container'>
17       <h1>Welcome to the Keydown Listening Component</h1>
18     </div>
19   );
20 };
```

# Example

- Always cleanup your event listeners. Do this with `window.removeEventListener` when your component unmounts. By cleaning up, you'll avoid listening to events multiple times and memory leaks.

```
class App2 extends React.Component {
  componentDidMount() {
    window.addEventListener('click', (event)=>{
      console.log("Shoot ME");
    });
  }
  componentWillUnmount(){
    window.removeEventListener('click', (event)=>{
      console.log("Shoot ME");});
  }
  render(){
    return( <div>
    <p> Welcome to my page </p>
    </div> );
  } }
export default App2;
```

```
const App2 = (props) => {
  React.useEffect(()=> {
    const handleClick = event => {
      console.log('Button clicked');
    };

    window.addEventListener('click',handleClick );
    return () => {
      window.removeEventListener('click', handleClick);
    };
  },[]);
  return (
    <div>
      <p> welcome to my page</p>
    </div>
  );
}
```

- useState() is a Hook that allows you to have state variables in functional components. You pass the initial state to this function and it returns a variable with the current state value (not necessarily the initial state) and another function to update this value.
- React Hooks are functions that add state variables to functional components and instrument the lifecycle methods of classes. They tend to start with use.
- useState enables you to add state to function components. Calling React.useState inside a function component generates a single piece of state associated with that component. Whereas the state in a class is always an object, with Hooks, the state can be any type. Each piece of state holds a single value, which can be an object, an array, a boolean, or any other type you can imagine.
- It's especially useful for local component state, but larger projects might require additional state management solutions.

## Declaring state in React

useState is a named export from react.

To use it, you can write:

**React.useState Or to import it just write useState:**

```
import React, { useState } from 'react';\n
```

The useState Hook allows you to declare only one state variable (of any type) at a time which is shown as follows:

```
import React, { useState } from 'react';\nconst Message= () => {\n  const messageState = useState( " ");\n  const listState = useState( [] );\n}
```

useState takes the initial value of the state variable as an argument.

# Example

```
import React, { useState } from 'react';
function App2() {
  // Declare a new state variable, which we'll call
  "count"
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
export default App2;
```

```
import React, { useState } from 'react';
class App2 extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1
        }}}>
          Click me
        </button>
      </div>
    );
  }
}
export default App2;
```



Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen.

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

## Function and Class Components

The simplest way to define a component is to write a JavaScript function:

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

This function is a valid React component because it accepts a single “props” (which stands for properties) object argument with data and returns a React element. We call such components “function components” because they are literally JavaScript functions.



**We can also define a component using a class:**

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

Previously, we only encountered React elements that represent DOM tags:

```
const element = <div />;
```

However, elements can also represent user-defined components:

```
const element = <Welcome name="Sara" />;
```

When React sees an element representing a user-defined component, it passes JSX attributes and children to this component as a single object. We call this object “props”.

For example, this code renders “Hello, Sara” on the page:

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
const element = <Welcome name="Sara" />;  
ReactDOM.render(  
  element,  
  document.getElementById('root') );
```

Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail. A button, a form, a dialog, a screen: in React apps, all those are commonly expressed as components.

For example, we can create an App component that renders Welcome many times:

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
function App() {  
  return (  
    <div>  
      <Welcome name="Sara" />  
      <Welcome name="Cahal" />  
      <Welcome name="Edite" />  
    </div> );  
}  
ReactDOM.render(  
  <App />,  
  document.getElementById('root')  
);
```

# Example.

```
<html>
<head>
  <script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>
  <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="root"></div>

  <script type="text/babel">
    function Welcome(props) {
      return <h1>Hello, {props.name}</h1>;
    }
    function Hello() {
      return <h1>Hello World!
        <Welcome name="Sara" /></h1>;
    }
    ReactDOM.render(<Hello />, document.getElementById('root'))

  </script>
</body>
</html>
```

We can split components into smaller components.  
For example, consider this Comment component:

```
function Comment(props) {  
  return (  
    <div className="Comment">  
      <div className="UserInfo">  
        <img className="Avatar"  
          src={props.author.avatarUrl}  
          alt={props.author.name}  
        />  
        <div className="UserInfo-name">  
          {props.author.name}  
        </div>  
      </div>  
      <div className="Comment-text">  
        {props.text} </div>  
      <div className="Comment-date">  
        {formatDate(props.date)} </div> </div> ); }  
}
```

It accepts author (an object), text (a string), and date (a date) as props, and describes a comment on a social media website.

This component can be tricky to change because of all the nesting, and it is also hard to reuse individual parts of it. Let's extract a few components from it.

First, we will extract Avatar:

```
function Avatar(props) {  
  return (  
    <img className="Avatar"  
      src={props.user.avatarUrl}  
      alt={props.user.name} /> );  
}
```

We can now simplify Comment a tiny bit:

```
function Comment(props) {  
  return (  
    <div className="Comment">  
      <div className="UserInfo">  
        <Avatar user={props.author} />  
        <div className="UserInfo-name">  
          {props.author.name}  
        </div>  
      </div>  
    </div>  
  );  
}
```



```
</div>
</div>
<div className="Comment-text">
  {props.text}
</div> <div className="Comment-date">
  {formatDate(props.date)}
</div> </div> );
}
```

Next, we will extract a UserInfo component that renders an Avatar next to the user's name:

```
function UserInfo(props) {
  return (
    <div className="UserInfo">
      <Avatar user={props.user} /> <div className="UserInfo-name"> {props.user.name}
    </div> </div> ); }
```





This lets us simplify Comment even further:

```
function Comment(props) {  
  return (  
    <div className="Comment"> <  
      UserInfo user={props.author} />  
    <div className="Comment-text"> {props.text} </div>  
    <div className="Comment-date"> {  
      formatDate(props.date)} </div> </div> ); }  
}
```

# Stateless vs Stateful Components

Stateful and stateless components have many different names.

They are also known as:

- **Container vs Presentational components**
- **Smart vs Dumb components**

The literal difference is that one has state, and the other doesn't. That means the stateful components are keeping track of changing data, while stateless components print out what is given to them via props, or they always render the same thing.

## **Stateful/Container/Smart component:**

```
class Main extends Component {  
  constructor() {  
    super()  
    this.state = {  
      books: []  
    }  
  }  
  render() {  
    return <BooksList books={this.state.books} />;  
  }  
}
```



## Stateless/Presentational/Dumb component:

```
const BooksList = ({books}) => {  
  return (  
    <ul>  
      {books.map(book => {  
        return <li>book</li>  
      })}  
    </ul>  
  )  
}
```



Stateful Component	Stateless Component
1. Stores info about component's state change in memory	1. Calculates the internal state of the components
2. Have authority to change state	2. Do not have the authority to change state
3. Contains the knowledge of past, current and possible future changes in state	3. Contains no knowledge of past, current and possible future state changes
4. Stateless components notify them about the requirement of the state change, then they send down the props to them.	4. They receive the props from the Stateful components and treat them as callback functions.

# Passing Method References Between Components



**src/App.js**

...

<Person

click={this.switchNameHandler}

name={this.state.persons[1].name}

age={this.state.persons[1].age}

/> ...

In Person component we will add a new property name click. There we will pass a reference to this.switchNameHandler and we can use this click property Person.js file.

**src/Person/Person.js**

import React from 'react';

const person = props => {

return (

<div>

<p onClick={props.click}>

I am a {props.name} and i am {props.age} year old!{' '}

</p> <p> {props.children} </p> </div> );

export default person;

In Person.js we can simply call props.click because click is the name of the property defined in App.js, And that will execute the switchNameHandler function which we pass as a reference.

In case if the method i.e. switchNameHandler accepts an argument like we want to pass a value to our function.

## **src/App.js**

```
switchNameHandler = (newName) => {  
  this.setState({  
    persons: [{ name: 'Lucy Stifert', age: 23 }, { name: newName, age: 12 }, { name: 'Mike', age:  
34 }],  
  });  
};
```

## How do we pass the data?

There are two ways of doing that:

- 1) `{ this.switchNameHandler.bind(this, 'NewName') }` : The first argument this is a list of arguments actually which will be passed into our function. and the second argument will be the new name.
- 2) `{ () => this.switchNameHandler() }` : This will return switchNameHandler function i.e a function call that is why we have added parentheses.

# Example

```
import React from 'react';
import Person from './Person/Person';

class App extends React.Component {
  state = {
    person:[
      {name: "Paul", location: "New York"},
      {name: "David", location: "Atlanta"},
      {name: "Raj", location: "London"}
    ],
    otherDetails: "I am working as Software Developer"
  }
  clickHandler = () => {
    this.setState({
      person:[
        {name: "Duke", location: "New York"},
        {name: "David", location: "Atlanta"},
        {name: "Raj", location: "Delhi"},
      ]
    })
  }
  render(){
    return (
      <div>
        <Person name={this.state.person[0].name} location={this.state.person[0].location} paraClick = {this.clickHandler}/>
        <Person name={this.state.person[1].name} location={this.state.person[1].location}
      >{this.state.otherDetails}</Person>
        <Person name={this.state.person[2].name} location={this.state.person[2].location} />
      </div>
    );
  }
}
export default App;
```

```
//Person.js
import React from 'react';

const Person = (props) => {
  return (
    <div>
      <p onClick={props.paraClick}> My name is {props.name} and I stay in
        {props.location}! {props.children}</p>
    </div>
  );
}

export default Person;
```



- Data Binding is the process of connecting the view element or user interface, with the data which populates it. Data binding is a way to synchronise the data between the model and view components automatically.
- In **ReactJS**, components are rendered to the user interface and the component's logic contains the data to be displayed in the view(UI). The connection between the data to be displayed in the view and the component's logic is called data binding in ReactJS.
- **One-way Data Binding:** ReactJS uses one-way data binding.

**1. Component to View:** Any change in component data would get reflected in the view, .i.e., any change in the state variable i.e. subject will reflect in the view part.

**2. View to Component:** Any change in View would get reflected in the component's data.

- Two-way binding — implicitly enforcing that some value in the DOM is always consistent with some React state — is concise and supports a wide variety of applications



```
import React, { Component } from 'react';
class App2 extends Component {

  constructor() {
    super();
    this.state = {
      subject: "hello"
    };
  }
  render() {
    return (
      <div>
        <h4>Welcome to my Web Page</h4>
        <p><b>{this.state.subject}</b> Everyone</p>

      </div>
    )
  }
}
export default App2;
```

# View to Component:



```
import React, { Component } from 'react';
class App2 extends Component {
  constructor() {
    super();
    this.state = {
      subject: ""
    };
  }
  handleChange = event => {
    this.setState({
      subject: event.target.value
    })
  }
  render() {
    return (
      <div>
        <h4> Welcome to my Web Page</h4>
        <input placeholder="Enter Subject"
          onChange={this.handleChange}></input>
        <p><b>{this.state.subject}</b> Everyone</p>

      </div>
    )
  }
}
export default App2;
```



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

function App(){
  const [name,setName] = useState("");

  const handleChange = (e)=>{
    setName(e.target.value);
  }
  return (
    <div>
      <input onChange={handleChange} value={name} />
      <h1>{name}</h1>
    </div>
  )
}
export default App;
```