npm install -g create-react-app

You are now ready to create your first React application!

Run this command to create a React application

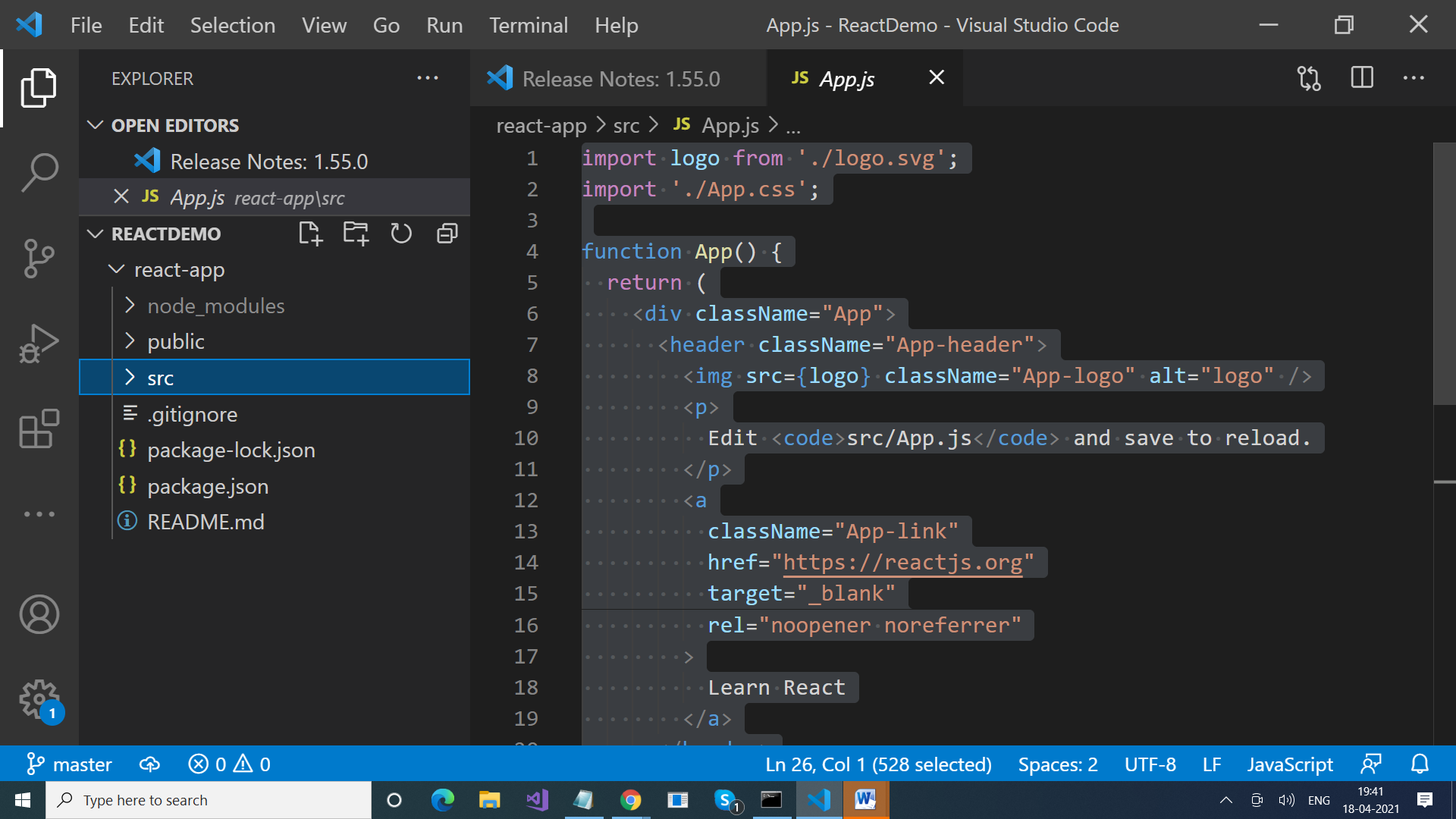
npx create-react-app myfirstreact

The create-react-app will set up everything you need to run a React application.

**npm-start :To build & run your application**

**We get three folders**

1. **Node\_modules**
2. **public**
3. **src**



**Index.html file**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="utf-8" />**

**<link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />**

**<meta name="viewport" content="width=device-width, initial-scale=1" />**

**<meta name="theme-color" content="#000000" />**

**<meta**

**name="description"**

**content="Web site created using create-react-app"**

**/>**

**<link rel="apple-touch-icon" href="%PUBLIC\_URL%/logo192.png" />**

**<!--**

**manifest.json provides metadata used when your web app is installed on a**

**user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/**

**-->**

**<link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />**

**<!--**

**Notice the use of %PUBLIC\_URL% in the tags above.**

**It will be replaced with the URL of the `public` folder during the build.**

**Only files inside the `public` folder can be referenced from the HTML.**

**Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC\_URL%/favicon.ico" will**

**work correctly both with client-side routing and a non-root public URL.**

**Learn how to configure a non-root public URL by running `npm run build`.**

**-->**

**<title>React App</title>**

**</head>**

**<body>**

**<noscript>You need to enable JavaScript to run this app.</noscript>**

**<div id="root"></div>**

**<!--**

**This HTML file is a template.**

**If you open it directly in the browser, you will see an empty page.**

**You can add webfonts, meta tags, or analytics to this file.**

**The build step will place the bundled scripts into the <body> tag.**

**To begin the development, run `npm start` or `yarn start`.**

**To create a production bundle, use `npm run build` or `yarn build`.**

**-->**

**</body>**

**</html>**

**Index.js file**

**import React from 'react';**

**import ReactDOM from 'react-dom';**

**import './index.css';**

**import App from './App';**

**import reportWebVitals from './reportWebVitals';**

**ReactDOM.render(**

**<React.StrictMode>**

**<App />**

**</React.StrictMode>,**

**document.getElementById('root')**

**);**

**// 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-vitals**

**reportWebVitals();**

**Default Component :**

**app.js**

**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;**

**Replace all the contents**

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

**class App extends Component {**

**render() {**

**return (**

**<div className="App">**

**<h1>Hello World!</h1>**

**</div>**

**);**

**}**

**}**

**export default App;**

**index.js:**

import React from 'react';

import ReactDOM from 'react-dom';

const myfirstelement = <h1>Hello React!</h1>

ReactDOM.render(myfirstelement, document.getElementById('root'));

**index.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>React App</title>

</head>

<body>

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

</body>

</html>

React Render HTML

React's goal is in many ways to render HTML in a web page.

React renders HTML to the web page by using a function called ReactDOM.render().

## The Render Function

The ReactDOM.render() function takes two arguments, HTML code and an HTML element.

The purpose of the function is to display the specified HTML code inside the specified HTML element.

Display a paragraph inside the "root" element:

ReactDOM.render(<p>Hello</p>, document.getElementById('root'));

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

<body>

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

</body>

**Index.html page**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport"

content="width=device-width, initial-scale=1" />

<title>React App</title>

</head>

<body>

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

</body>

</html>

**Index.js file**

import React from 'react';

import ReactDOM from 'react-dom';

ReactDOM.render(<p>Hello</p>, document.getElementById('root'));

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport"

content="width=device-width, initial-scale=1" />

<title>React App</title>

</head>

<body>

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

</body>

</html>

**Index.js**

import React from 'react';

import ReactDOM from 'react-dom';

const myelement = (

<table>

<tr>

<th>Name</th>

</tr>

<tr>

<td>John</td>

</tr>

<tr>

<td>Elsa</td>

</tr>

</table>

);

ReactDOM.render(myelement, document.getElementById('root'));

## The Root Node

The root node is the HTML element where you want to display the result.

It is like a container for content managed by React.

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport"

content="width=device-width, initial-scale=1" />

<title>React App</title>

</head>

<body>

<header id="sandy"></header>

</body>

</html>

**Index.js**

import React from 'react';

import ReactDOM from 'react-dom';

ReactDOM.render(<p>Hallo</p>, document.getElementById('sandy'));

# React Components

Components are like functions that return HTML elements.

## React Components

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

Components come in two types, Class components and Function components, in this tutorial we will concentrate on Class components.

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. This page provides an introduction to the idea of components.

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

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.

A Function component also returns HTML, and behaves pretty much the same way as a Class component, but Class components have some additions,

Create a Function component called Car

function Car() {

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

}

Display the Car component in the "root" element:

ReactDOM.render(<Car />, document.getElementById('root'));

Index.html

!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport"

content="width=device-width, initial-scale=1" />

<title>React App</title>

</head>

<body>

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

</body>

</html>

Index.js

import React from 'react';

import ReactDOM from 'react-dom';

function Car() {

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

}

ReactDOM.render(<Car />, document.getElementById('root'));

## Create a Class Component

When creating a React component, the component's name must start with an upper case letter.

The component has to include the extends React.Component statement, this statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a render() method, this method returns HTML.

Create a Class component called Car

class Car extends React.Component {

render() {

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

}

}

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

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

Display the Car component in the "root" element:

ReactDOM.render(<Car />, document.getElementById('root'));

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport"

content="width=device-width, initial-scale=1" />

<title>React App</title>

</head>

<body>

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

</body>

</html>

**Index.js**

import React from 'react';

import ReactDOM from 'react-dom';

class Car extends React.Component {

render() {

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

}

}

ReactDOM.render(<Car />, document.getElementById('root'));

You can also use an [ES6 class](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Classes) to define a component:

class Welcome extends React.Component {

render() {

return <h1>Hello, {this.props.name}</h1>;

}

}

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

}

const element = <Welcome name="Sara" />;ReactDOM.render(

element,

document.getElementById('root')

);

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')

);

**App.js file**

import logo from './logo.svg';

import './App.css';

function Welcome(props) {

  return <h1>Hello, {props.name}</h1>;

}

function App() {

  return (

    <div>

    <Welcome name="Sara" />      <Welcome name="Cahal" />      <Welcome name="Edite" /></div>

     );

}

export default App;

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

ReactDOM.render(

    <App />,

  document.getElementById('root')

);

**App.js**

import logo from './logo.svg';

import './App.css';

function Welcome(props) {

  return <h1>Hello, {props.name}</h1>;

}

function App1()

{

  return <h2> HI </h2>;

}

function App() {

  return (

    <div>

      <App1> </App1>

    <Welcome name="Sara" />      <Welcome name="Cahal" />      <Welcome name="Edite" /></div>

     );

}

export default App;

import logo from './logo.svg';

import './App.css';

function Welcome(props) {

  return  <div>Hello, {props.name}</div>;

}

function App1()

{

  return ( <font color='red'> <Welcome name="Sara" />      <Welcome name="Cahal" />      <Welcome name="Edite" /> </font>)

}

function App() {

  return (

      <App1> </App1>

     );

}

export default App;

React Props

Props are arguments passed into React components.

Props are passed to components via HTML attributes

## React Props

React Props are like function arguments in JavaScript and attributes in HTML.

To send props into a component, use the same syntax as HTML attributes:

Add a "brand" attribute to the Car element:

const myelement = <Car brand="Ford" />;

The component receives the argument as a props object:

Use the brand attribute in the component:

class Car extends React.Component {

render() {

return <h2>I am a {this.props.brand}!</h2>;

}

}

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport"

content="width=device-width, initial-scale=1" />

<title>React App</title>

</head>

<body>

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

</body>

</html>

Index.js

import React from 'react';

import ReactDOM from 'react-dom';

class Car extends React.Component {

render() {

return <h2>I am a {this.props.brand}!</h2>

}

}

const myelement = <Car brand="Ford" />;

ReactDOM.render(myelement, document.getElementById('root'));

## Pass Data

Props are also how you pass data from one component to another, as parameters.

### Example

Send the "brand" property from the Garage component to the Car component:

class Car extends React.Component {

render() {

return <h2>I am a {this.props.brand}!</h2>;

}

}

class Garage extends React.Component {

render() {

return (

<div>

<h1>Who lives in my garage?</h1>

<Car brand="Ford" />

</div>

);

}

}

ReactDOM.render(<Garage />, document.getElementById('root'));

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport"

content="width=device-width, initial-scale=1" />

<title>React App</title>

</head>

<body>

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

</body>

</html>

**Index.js**

import React from 'react';

import ReactDOM from 'react-dom';

class Car extends React.Component {

render() {

return <h2>I am a {this.props.brand}!</h2>;

}

}

class Garage extends React.Component {

render() {

return (

<div>

<h1>Who lives in my Garage?</h1>

<Car brand="Ford" />

</div>

);

}

}

ReactDOM.render(<Garage />, document.getElementById('root'));

If you have a variable to send, and not a string as in the example above, you just put the variable name inside curly brackets:

import React from 'react';

import ReactDOM from 'react-dom';

class Car extends React.Component {

render() {

return <h2>I am a {this.props.brand}!</h2>;

}

}

class Garage extends React.Component {

render() {

const carname = "Ford";

return (

<div>

<h1>Who lives in my Garage?</h1>

<Car brand={carname} />

</div>

);

}

}

ReactDOM.render(<Garage />, document.getElementById('root'));

Or if it was an object:

### Example

Create an object named "carinfo" and send it to the Car component:

class Car extends React.Component {

render() {

return <h2>I am a {this.props.brand.model}!</h2>;

}

}

class Garage extends React.Component {

render() {

const carinfo = {name: "Ford", model: "Mustang"};

return (

<div>

<h1>Who lives in my garage?</h1>

<Car brand={carinfo} />

</div>

);

}

}

ReactDOM.render(<Garage />, document.getElementById('root'));

## Props in the Constructor

If your component has a constructor function, the props should always be passed to the constructor and also to the React.Component via the super() method.

### Example

class Car extends React.Component {

constructor(props) {

super(props);

}

render() {

return <h2>I am a {this.props.model}!</h2>;

}

}

ReactDOM.render(<Car model="Mustang"/>, document.getElementById('root'));

React Props are read-only! You will get an error if you try to change their value.

# What is Constructor?

The constructor is a method used to initialize an object's state in a class. It automatically called during the creation of an object in a class.

The concept of a constructor is the same in React. The constructor in a React component is called before the component is mounted. When you implement the constructor for a React component, you need to call **super(props)** method before any other statement. If you do not call super(props) method, **this.props** will be undefined in the constructor and can lead to bugs.

### Syntax

1. Constructor(props){
2. **super**(props);
3. }

In React, constructors are mainly used for two purposes:

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

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

import React, { Component } from 'react';

class SecondApp extends Component {

  constructor(props){

    super(props);

    this.state = {

         data: 'Employees Data'

      }

    this.handleEvent = this.handleEvent.bind(this);

  }

  handleEvent(){

    console.log(this.props);

  }

  render() {

    return (

      <div className="SecondApp">

    <h2>React Constructor Example</h2>

    <input type ="text" value={this.state.data} />

        <button onClick={this.handleEvent}>Please Click</button>

      </div>

    );

  }

}

export default SecondApp;

**index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

import SecondApp from './SecondComp';

ReactDOM.render(

  <div>

  <App/>

    <SecondApp /> </div>,

  document.getElementById('root')

);

// 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-vitals

reportWebVitals();

**Constructors :** [**https://www.javatpoint.com/react-constructor**](https://www.javatpoint.com/react-constructor)

**React State :** [**https://www.javatpoint.com/react-state**](https://www.javatpoint.com/react-state)

**React Props :** [**https://www.javatpoint.com/react-props**](https://www.javatpoint.com/react-props)

**State Vs Props :** [**https://www.javatpoint.com/react-state-vs-props**](https://www.javatpoint.com/react-state-vs-props)