# 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>);
  }
}

// Example usage: <ShoppingList name="Mark" />
```

## Perks of learning React

1. React is very popular. As a developer, it's quite likely that you're going to work on a React project in the future. Perhaps an existing project, or maybe your team will want you to work on a brand new app based on React.
2. A lot of tooling today is built using React at the core. Popular frameworks and tools like Next.js,

Gatsby and many others use React under the hood.

3. As a front-end engineer, React is probably going to come up in a job interview.

### React Components

You just saw how to create your first React application. This application comes with a series of files that do various things, mostly related to configuration, but there's one file that stands out: App.js. App.js is the first React Component you meet.
Its code is this:

```
import React from 'react'
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="lo
        <p>
          Edit <code>src/App.js</code> and save to r
        </p>
        <a
          className="App-link"
          href="<https://reactjs.org>"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  )
}
export default App
```

An application built using React, or one of the other popular front end frameworks like Vue and Svelte for example, is built using dozens of components. But let's start by analyzing this first component.

```
import React from 'react'
import logo from './logo.svg'
import './App.css'

function App() {
  return /* something */
}

export default App
```

You can see a few things here. We import some things, and we export a function called App.

The things we import in this case are a JavaScript library (the react npm package), an SVG image, and a CSS file.

create-react-app is set up in a way that allows us to import images and CSS to use in our JavaScript, but this is not something you need to care about now. You need to care about the concept of a component

**App** is a function that in the original example returns something that at first sight looks quite strange.

It looks like HTML but it has some JavaScript embedded into it. That is JSX, a special language used to build a component's output. We'll talk more about JSX in the next section.

In addition to defining some JSX to return, a component has several other characteristics.

A component can have its own state, which means it encapsulates some variables that other components can't access unless this component exposes this state to the rest of the application.

A component can also receive data from other components. In this case, we talk about props.

## Using JSX to compose UI

As introduced in the last section, one of the main benefits of JSX is to make it very easy to build a UI.
In particular, in a React component you can import other React components, and you can embed them and display them.
A React component is usually created in its own file, because that's how we can easily reuse it (by importing it) in other components.
But a React component can also be created in the same file of another component, if you plan to only use it in that component. There's no "rule" here, you can do what feels best to you.
I generally use separate files when the number of lines in a file grows too much.
To keep things simple let's create a component in the same App.js file.
We're going to create a WelcomeMessage component

```
function WelcomeMessage() {
  return <p>Welcome!</p>
}
```

See? It's a simple function that returns a line of JSX that represents a p HTML element.
We're going to add it to the App.js file. Now inside the App component JSX we can add <WelcomeMessage /> to show this component in the user interface.

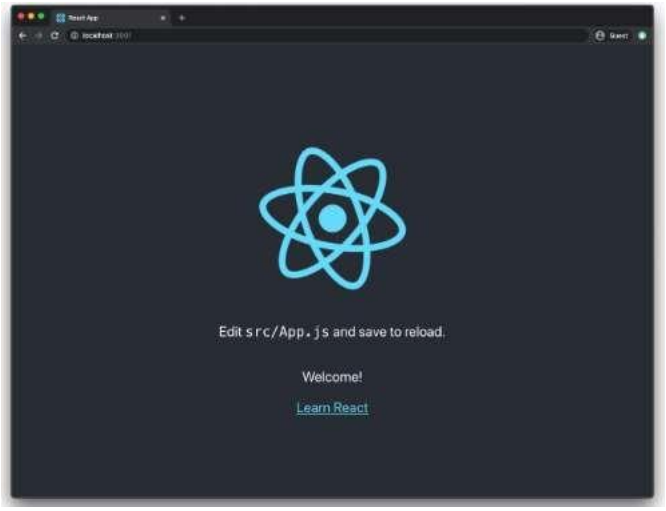
```
import React from 'react'
import logo from './logo.svg'
import './App.css'

function WelcomeMessage() {
  return <p>Welcome!</p>
}

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

export default App
```

And here's the result. Now you can see the "Welcome!" message in the screen?




We say WelcomeMessage is a child component of App, and App is its parent component.
We're adding the <WelcomeMessage /> component like if it was part of the HTML language.
That's the beauty of React components and JSX: we can compose an application interface and use it like we're writing HTML.
With some differences, as we'll see in the next section.

## Rendering Elements

## Components And Props

## State and Lifecycle

## Handling Events

## Conditional Rendering

## Lists and Keys

## Forms

## Lifting State Up

## Accessibility

## Context

## Error Boundaries

Forwarding Refs

Fragments

Higher Order Components

Type Checking

Uncontrolled Components

Hooks

Routing



For any queries, you can reach us at
support@newtonschool.co