What is ReactJS?

ReactJS is a **JavaScript** library used to build User Interfaces (UI). It significantly decreases the code with its components, states (i.e., hooks), etc.

Creating a React App

Open your terminal in the directory where you would like to create your application. Run this command to create a React application named `my-react-app`:

```
npx create-react-app my-react-app
```

What is ES6?

ES6 stands for ECMAScript 6. ECMAScript is a JavaScript standard intended to ensure a common language across different browsers. ES6 is the 6th version of ECMAScript.

Why ES6? / Features of ES6 / Upgrades in ES6

React uses ES6 and all of these new features will make your coding experience in React much better. You will be able to do things with much more ease and in fewer lines! Features like:

- **Arrow Functions:**

```
const hello = () => {  return "Hello World!";}
```

or

```
const hello = () => "Hello World!";
```

- **.map():** `.map` can be used for a lot of things. One of its use cases is that we can make any number of cards through a loop and just put it in JSX, like this:

```
const data = ['title1', 'title2', 'title3'];let cards =
data.map((item) => <card>{item}</card>);
```

- **Destructuring:**

Old Way:

```
const languages = ['JS', 'Python', 'Java'];const js =
languages[0];const python = languages[1];const java =
languages[2];
```

New Way:

```
const languages = ['JS', 'Python', 'Java'];const [js, python,
java] = languages;
```

- **Ternary Operator:** With this, you can write if/else conditions in one line. Its syntax is fairly simple like this:

```
condition ? <expression if true> : <expression if false>
```

Example:

```
let loading = false;const data = loading ?
<div>Loading...</div> : <div>Data</div>;
```

- **Spread Operator:**

```
const languages = ['JS', 'Python', 'Java'];const morelanguages
= ['C', 'C++', 'C#']; const allLanguages =
[...languages, ...morelanguages];
```

Output:

```
["JS", "Python", "Java", "C", "C++", "C#"]
```

React Components

There are two types of components:

1. Class Based Components
2. Function Based Components

Class Based Components

Before making a class based component, we need to inherit functions from `React.Component` and this can be done with `extends`, like this:

```
class Cat extends React.Component {  render() {    return <h1>Meow</h1>;  }}
```

It also requires a `render` method which returns HTML.

Function Based Components (Hooks)

In function components, it's simpler; we just need to return the HTML, like this:

```
function Cat() {  return <h1>Meow</h1>;}
```

Note: Component's name must start with an uppercase letter.

Rendering a Component

We made a component, now we want to render/use it. The syntax for using a component is:

```
<ComponentName />
```

Components in Files

To have less mess inside the main file (with all the components in the same file) and to reuse components on different pages, we have to make them separately. So that we can just import them in any file and use them!

For that, we will just make a new file called `Cat.js`, make a class or function based component there and `export default` that class/function! Like this:

```
function Cat() {  return <h1>Meow</h1>;} export default Cat;
```

Note: File name must start with an uppercase letter.

Props

As mentioned earlier, we can import the same component in different files and use it, but maybe in different files some changes in the component are needed. For that, we can use `props`! Like this:

Component:

```
function Cat(props) {  return <h1>Meow's color is
{props.color}</h1>;}
```

Main file:

```
<Cat color="purple" />
```

Class Based Components

Before making a class-based component, we need to inherit functions from `React.Component`, and this can be done with `extends`, like this:

```
class Cat extends React.Component {  render() {    return <h1>Meow</h1>;  }}
```

It also requires a `render` method which returns HTML.

Note: Component's name must start with an uppercase letter.

Component Constructor

The constructor gets called when the component is initiated. This is where you initiate the component's properties. In React, we have states which update on the page without reload. Constructor properties are kept in state.

We also need to add the `super()` statement, which executes the parent component's constructor, and the component gets access to all the functions of the parent component, like this:

```
class Cat extends React.Component {  constructor() {    super();    this.state = { color: "orange" };  }  render() {    return <h1>Meow's color is {this.state.color}</h1>;  }}
```

Props are arguments passed to React components via HTML attributes. Example:

Component:

```
function Cat(props) {  return <h1>Meow's color is {props.color}</h1>;}
```

Main file:

```
<Cat color="purple" />
```

Output: **Events**

Every HTML attribute in React is written in camelCase syntax. Event is also an attribute. Hence, written in camelCase.

As we learned, variables, states, and JavaScript operations are written in curly braces `{}`, and the same is true with React event handlers too! Like this: `onClick={show}`

```
<button onClick={show}>Show</button>
```

Arguments in Events

We can't pass arguments just like that; it will give a syntax error. First, we need to put the whole function in an arrow function, like this:

```
<button onClick={ () => show('true') }>Show</button>
```

React Event Object

Event handler can be provided to the function like this:

```
<button onClick={ (event) => show('true', event) }>Show</button>
```

React Lists

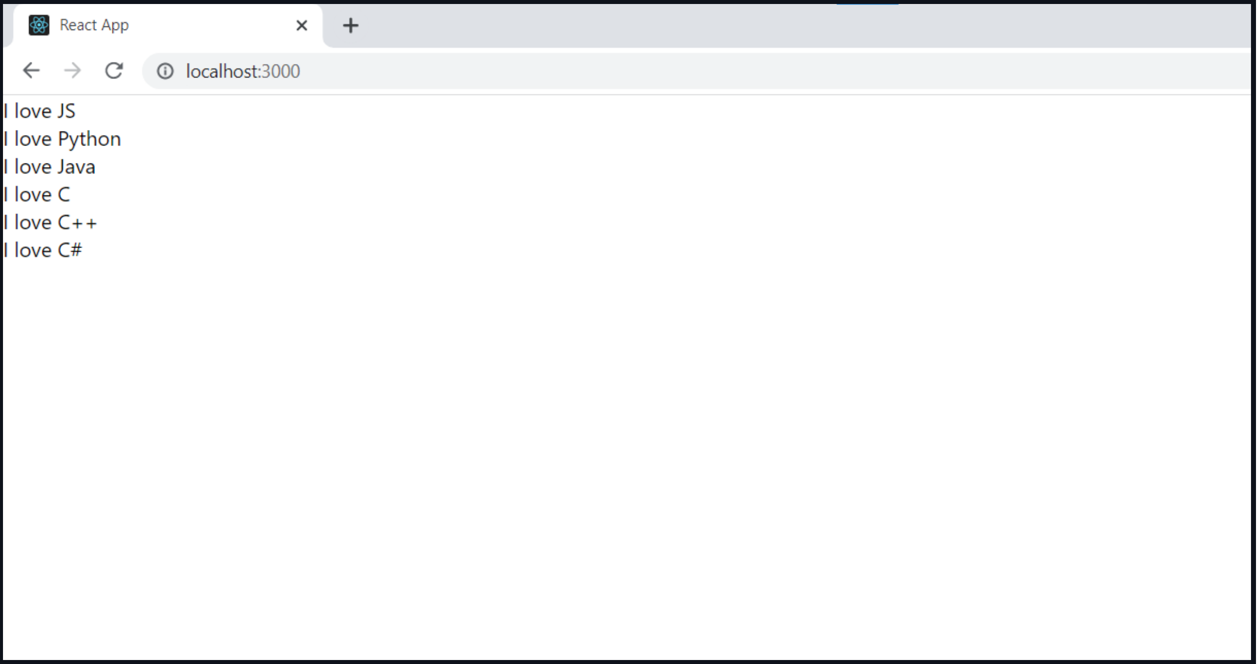
In React, we render lists with loops.

`map()`

An example of `map`:

```
function App() {  const languages = ['JS', 'Python', 'Java', 'C',
'C++', 'C#'];  return (    <div className="App">
{languages.map((language) => {        return <div>I love
{language}</div>      })}    </div>  );}
```

Output:



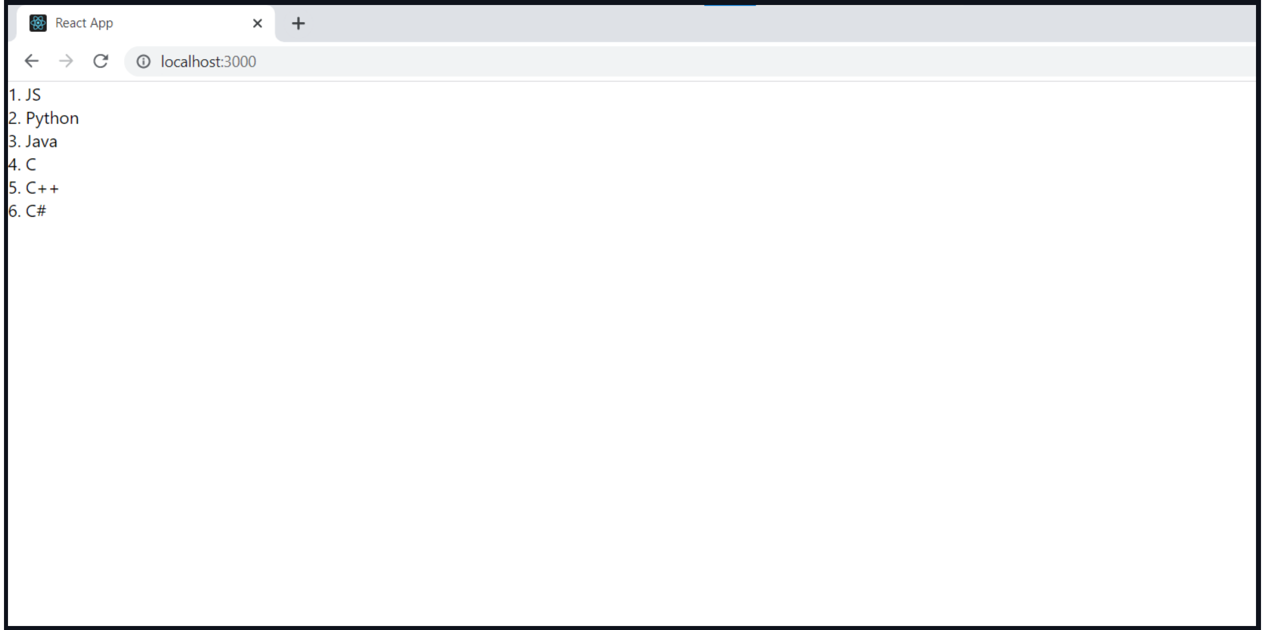
```
I love JS
I love Python
I love Java
I love C
I love C++
I love C#
```

Keys

With keys, React keeps a record of elements. This ensures that if an item is updated or removed, only that item will be re-rendered instead of the entire list. Example:

```
function App() {  const languagesDict = [    { id: 1, language:
'JS' },    { id: 2, language: 'Python' },    { id: 3, language:
'Java' },    { id: 4, language: 'C' },    { id: 5, language: 'C++' },
{ id: 6, language: 'C#' }  ];  return (    <div className="App">
{languagesDict.map((language) => {        return <div
key={language.id}>          {language.id}. {language.language}
</div>      })}    </div>  );}
```

Output:



```
1. JS
2. Python
3. Java
4. C
5. C++
6. C#
```

React Forms

React Forms are mostly like normal HTML forms, except we use `state` in this to handle inputs.

Handling Forms

In React, all the data is handled by the component and stored in the component state. We can change state with event handlers in the `onChange` attribute, like this:

```
import { useState } from 'react'; function Form() {  const [email, setEmail] = useState('');  return (     <form>       <label>         Enter your email: <input type="email" value={email} onChange={(e) => setEmail(e.target.value)} />       </label>     </form>   )} export default Form;
```

Submitting Form

We can submit the form with the `onSubmit` attribute for the `<form>`.

```
import { useState } from 'react'; function Form() {  const [email,
setEmail] = useState('');  const handleSubmit = (e) =>
{    e.preventDefault();    alert(`Your email is: ${email}`)  }
return (    <form onSubmit={handleSubmit}>      <label>        Enter
your email: <input type="email" value={email} onChange={(e) =>
setEmail(e.target.value)} />      </label>      <input type="submit"
/>    </form>  )} export default Form;
```

Multiple Inputs

We don't have to make multiple states for multiple inputs; we can save all values in one, like this:

```
import { useState } from 'react'; function Form() {  const [data,
setData] = useState({});  const handleChange = (e) =>
{    setData({...data, [e.target.name]: e.target.value})  }  const
handleSubmit = (e) => {    e.preventDefault();    alert(`Your email
is: ${data.email} and your name is: ${data.name}`)  }  return
(    <form onSubmit={handleSubmit}>      <label>        Enter your
email: <input type="email" name='email' value={data.email}
onChange={handleChange} />      </label>      <label>        Enter
your name: <input type="text" name='name' value={data.name}
onChange={handleChange} />      </label>      <input type="submit" />
</form>  )} export default Form;
```

Here in `handleChange`, we used **Spread Operators**. We are basically saying keep the whole data as it was, just change this name's value. Then it is saved as an object so we are getting that by `data.email` and `data.name`.

React CSS Styling

There are three ways to style in React:

1. Inline Styling
2. CSS Stylesheets
3. CSS Modules

Inline Styling

To inline style an element, we can make a JavaScript Object, like this:

```
const App = () => {  return (    <>      <h1 style={{ backgroundColor: "purple" }}>CodeWithHarry</h1>    </>  );}
```

In this, the first curly brace is to write JavaScript and the second is to make a JavaScript object. We can also write it like:

```
const App = () => {  const h1Style = {    backgroundColor: 'purple',    color: 'white'  }  return (    <>      <h1 style={h1Style}>CodeWithHarry</h1>    </>  );}
```

Note: CSS property names must be camelCase. `background-color` would be `backgroundColor`.

CSS Stylesheets

You can save the whole CSS in a separate file with the file extension `.css` and import it into your application.

`App.css`

```
body {  background-color: 'purple';  color: 'white';}
```

Here we are writing CSS, so we don't need to make a JS Object or do it in camelCase.

`Index.js`

Just import it like this:

```
import './App.css'
```

CSS Modules

In this, you don't have to worry about name conflicts as it is component-specific. The CSS is available only for the file in which it is imported.

Make a file with the extension `.module.css`, for example: `index.module.css`

Make a new file `index.module.css` and insert some CSS into it, like this:

```
.button {  background-color: 'purple';  color: 'white';}
```

Import it in the component like this:

```
import styles from './index.module.css'; const Home = () =>
{    return (        <button className={styles.button}>Click
me!</button>    )}; export default Home;
```