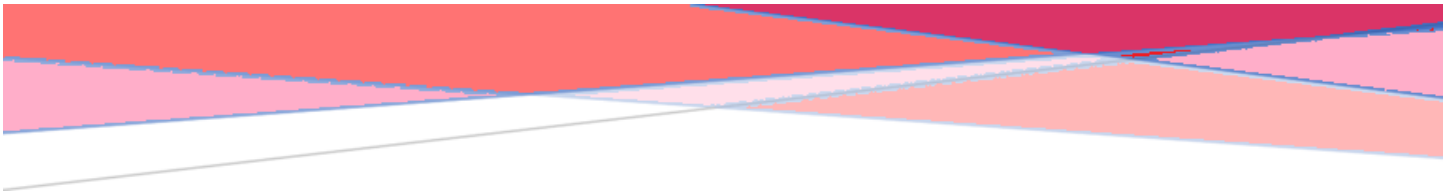


REACT

ReactJS Props and Classes

Abstract

This is quick start into building React applications. We start out with Props on the first day, then on the second day we move into re-purposing an existing HTML app into a React application.

Axle Barr

https://github.com/axleb/reactjs-2day-bootcamp-day01

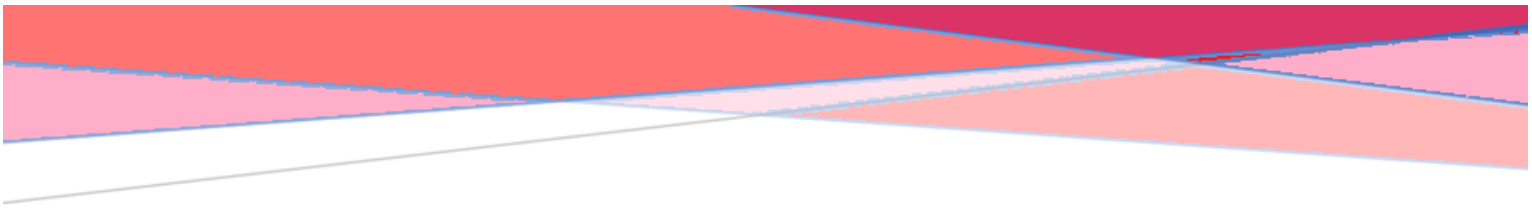


Table of Contents

PART 01 – BOILERPLATE APP BUILD

1. From a terminal window, pointing to your parent folder (create one for this bootcamp), run the command
   >**sudo npm install -g create-react-app** (remember to do this globally).

2. Choose a name for your app and run the command to create a new app, in my case I will be running the command **create-react-app bcprops**. Run this command in your parent folder**.**
   So *bcprops*, is the name of my React application.

3. CD into the **bcprops** folder and open the application with **VSCode** (or another editor) and most of our work will be in the app folder, which acts like the parent folder.

4. After you CD into the bcprops folder run the command >**npm run start**, your browser should open and you should see the default React UI

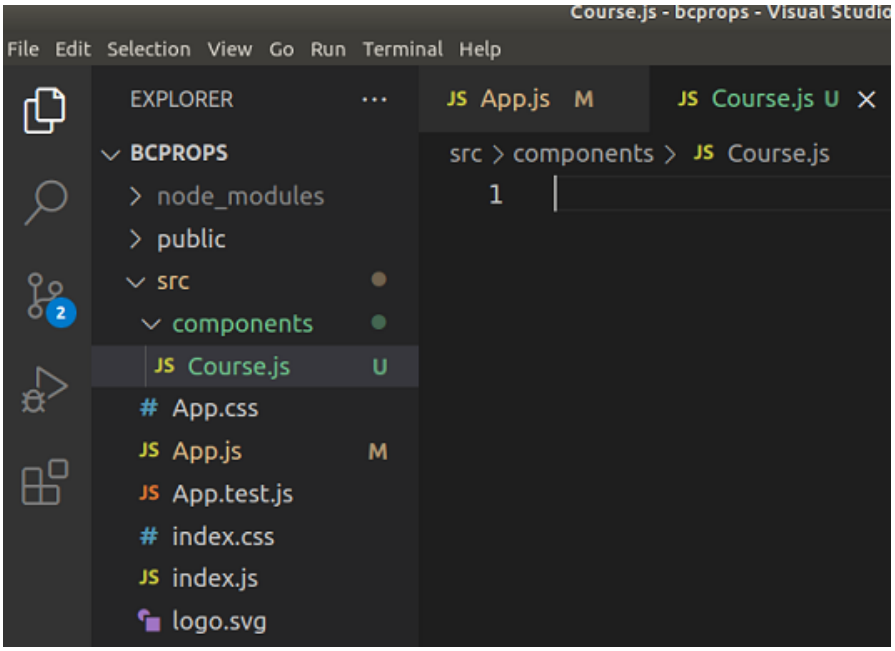
5. In VSCode, view the code of app.js, this is the file that feeds the default page that shows up on the browser at port 3000. Change the text between the anchor tags, so change *Learn React* to *Learn React Now* or something similar, hit **save**.

6. Remove all the code between the `<div>` tags in App.js, this is the `div` with the `className` attribute with a value of "App". Also remove the line that imports the logo:

```
import './App.css';
function App() {
  return (
    <div className="App">
     <h2>Courses offered by Skillsoft</h2><p></p>
    </div>
  );
}
export default App;
```

   The `<p>` tags are just for spacing

7. In VSCode or from your file system, create a new folder under the src folder called components. In the components folder create a file called Course.js:



Notice the upper-case **C**, more on that later.

8. In Course.js, here is the basic code needed to create a function. Arrow functions can be used also, but the function name must be *exported*:

```
function Course(){

}
export default Course;
```

9. Every component should return something, usually **JSX**. Once the component gets incorporate into another component or App.js, the return statement will be rendered:

```
function Course(){
  return(
    <div>JavaScript</div>
  )
}
```

Note: return is a function and it can only return one parent JSX element. Render simply means take this code and throw it onto the browser window is some element.

PART 02 – RENDERING THE COMPONENT

1. We can pass single or multiple components to App.js. Simply import the file housing the component and dispay that component like a custom JSX/HTML element:

```
import Course from "./components/Course";
function App() {
  return (
    <div className="App">
      <h2>Courses offered by Skillsoft</h2>
      <p></p>
      <Course />
    </div>
  );
```

2. This means that we can do something like this in App.js:

```
return (
  <div className="App">
    <h2>Courses offered by Skillsoft</h2>
    <p></p>
    <Course />
    <Course />
    <Course />
  </div>
);
```

3. Since a component is just a function, it can have properties. These properties get interpreted by the React engine. We made this change in Course.js but it shows up on the browser window:

```
const mostPopular = "JavaScript";
function Course(){
  return(
    <div>Course Name: {mostPopular}</div>
  )
}
```

You can remove the second and third component call from App.js

4. Most times, values of properties are passed to the child component from the parent component, App.js in this case:

```
const mostPopular = "JavaScript";
function App() {
  return (
    <div className="App">
      <h2>Courses offered by Skillsoft</h2>
      <p></p>
      <Course popCourse={mostPopular}/>
    </div>
  );
}
```

We would pass the value of the `mostPopular` property in App.js to Course.js as an attribute of the `<Course>` element.

5. In Course.js if we want to use that value, we have to pass **props** as an argument when we define the function:

```
//const mostPopular = "JavaScript";
function Course(props){
  return(
    <div>Course Name: { }</div>
  )
}
```

The name we use can be anything but it is convention to use **props**. Notice that the original property of this function is now commented out

6. To see the value of what we passed, we now access it via props

```
function Course(props){
  return(
    <div>Course Name: {props.popCourse}</div>
  )
}
```

7. Here is the entire Course.js up to this point:

```
function Course(props){
  return(
    <div>Course Name: {props.popCourse}</div>
  )
}
//
export default Course;
```

8. Here is the entire App.js up to this point:

```
import './App.css';
import Course from "./components/Course";
const mostPopular = "JavaScript";
function App() {
  return (
    <div className="App">
      <h2>Courses offered by Skillsoft</h2>
      <p></p>
      <Course popCourse={mostPopular}/>
    </div>
  );
}
export default App
```

PART 03 – PASSING TWO PROPERTIES

1. We can pass multiple values via an array from App.js, or any parent component:

```
const mostPopular = "JavaScript";
const leastPopular = "VueJS";
//
function App() {
  return (
    <div className="App">
      <h2>Courses offered by Skillsoft</h2>
      <p></p>
      <Course popCourse={[mostPopular, leastPopular]}/>
    </div>
  );
```

There are other ways, but for multiple items, this works well for a small number

2. In Course.js we would have to use the array to get values, so one way is to do this:

```
function Course(props){
  return(
    <div>Course Name: {props.popCourse[0]}</div>
  )
}
```

3. We can return two `<div>` but we can create a parent pair of `<div>` and return multiple elements inside of that parent `<p>` in this case:

```
  return(
    <div>
      <p>Course Name: {props.popCourse[0]}</p>
      <p>Course Name: {props.popCourse[1]}</p>
    </div>
  )
```

4. In App.js a popular way to pass values is via an object:

```
const courses = {
  mostPopular : "JavaScript",
  leastPopular : "VueJS"
};
//
function App() {
  return (
    <div className="App">
      <h2>Courses offered by Skillsoft</h2>
      <p></p>
      <Course
      mostPopular={courses.mostPopular}
      leastPopular={courses.leastPopular}
      />
    </div>
  );
```

When the **Course** component is invoked, two properties are passed to the component via **props**.

5. In Course.js we would only have to use **props** then the name of the property:

```
    return(
      <div>
        <p>Course Name: {props.mostPopular}</p>
        <p>Course Name: {props.leastPopular}</p>
      </div>
    )
```

6. (Optional) Pass the object in App.js and extract in Courses.js:

App.js

```
  return (
      <div className="App">
        <h2>Courses offered by Skillsoft</h2>
        <p></p>
        <Course courses={courses}/>
      </div>
```

Course.js

```
  return(
      <div>
        <p>Course Name: {props.courses.mostPopular}</p>
        <p>Course Name: {props.courses.leastPopular}</p>
      </div>
    )
```

PART 04 – PASSING TO THREE LEVELS

1. Add a third key/value pair to the `courses` object in App.js:

```
    import Course from "./components/Course";
    const courses = {
      mostPopular : "JavaScript",
      leastPopular : "VueJS",
      futureCourse: "FSD with NodeJS"
    };
    function App() {
```

Notice the comma after VueJS

2. Also, add a new component which we will call from Course.js, call it Future.js

```
    const Future = () => {
      return(
        <div>
          <p>Coming Soon : { }</p>
        </div>
      )
    }
    //
    export default Future;
```

I used function literal here for demonstration purposes but the regular function will work as well.

3. Now we call this new component from Course.js NOT from App.js

```
return(
  <div>
    <p>Course Name: {props.mostPopular}</p>
    <p>Course Name: {props.leastPopular}</p>
    <div><Future /></div>
  </div>
)
```

Remember to import this component into Course.js.

4. All that is left now is to pass the values into the third level component *Future*. However there is no direct relationship between App.js and Future.js. Remember it is in App.js is where we declared our object. Continue passing props down to this level and access values from props:

```
    const Future = props => {
      return(
        <div>
          <p>Coming Soon : { props.futureCourse}</p>
        </div>
      )
    }
    //
    export default Future;
```

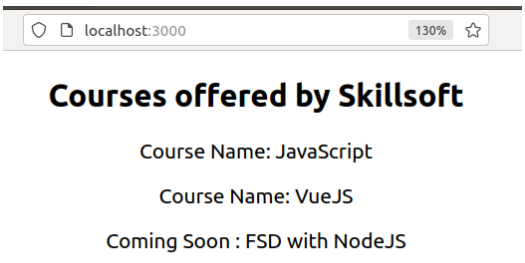
This still did not work, we have to first add an *attribute* in Course.js

5. To pass this value all the way to the third component we first have to pass it through **Course**:

```
  <div>
    <p>Course Name: {props.mostPopular}</p>
    <p>Course Name: {props.leastPopular}</p>
    <Future futureCourse={props.futureCourse}/>
  </div>
```

6. Then in App.js add this third attribute to Course:

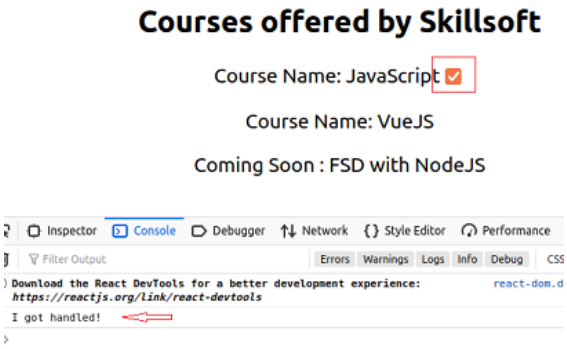
```
  <div className="App">
    <h2>Courses offered by Skillsoft</h2>
    <p></p>
    <Course
    mostPopular={courses.mostPopular}
    leastPopular={courses.leastPopular}
    futureCourse={courses.futureCourse}
    />
  </div>
```



PART 05 – DOM EVENTS

1. For a simple *event* demo, add a checkbox to the first course (in Course.js) and add a handler function. All JSX events start witn the word **on**.

```
function Course(props){
  function handleOnChange() {
    console.log("I got handled!");
  };
  return(
    <div>
      <p>
        Course Name: {props.mostPopular}
        <input
          type="checkbox"
          onChange={handleOnChange}
        />
      </p>
      <p>Course Name: {props.leastPopular}</p>
```



2. However we may want to update one of the values, for example we may have a property like *uRecommended* and initially set to false:

```
function Course(props){
  let uRecommended = false;
  function handleOnChange() {
    console.log("I got handled!");
  };
  return(
```

3. When the checkbox is checked, we want to change the value of uRecommend:

```
function Course(props){
  let uRecommended = false;
  function handleOnChange() {
    uRecommended = true;
    console.log(uRecommended);
  };
return(
```

We also print the value to make sure it was changed from *false* to *true*.

4. Soon we will use this value to change the appearance of a course that was recommended. For now we can just setup the code:

```
          <div>
            <p>
              Course Name:
              <span className={uRecommended ? 'strongText' : 'normalText'}>
                {props.mostPopular}
              </span>
              <input
                type="checkbox"
```

Based on the value of *uRecommend*, the course name itself will be either *bolded* or *normal* font weight. This code wont work at the moment.

5. Just to round off this part, I added two classes in App.css to match the styles in #4 above:

```
.normalText {
  font-weight: normal;
}

.strongText {
  font-weight: bold;
}
```

6. The reason this will not work is that the component does not get refreshed. We need to inform React that one of the components did change and that it should re-draw that particular component on the browser window. Before we utilize this feature of React, lets move the recommendation variable and value to App.js where it can be passed via props:

```
const courses = {
  mostPopular : "JavaScript",
  leastPopular : "VueJS",
  futureCourse: "FSD with NodeJS",
  recommendation: false
};
function App() {
```

Add one more key/value pair to obtain an initial value for recommendation

7. We we call the Course component, pass the initial value along with the others:

```
      <Course
      mostPopular={courses.mostPopular}
      leastPopular={courses.leastPopular}
      futureCourse={courses.futureCourse}
      recommendation={courses.recommendation}
      />
    </div>
```

8. We first need to import **useState** from the React library, then add it INSIDE of the function where it is being used. Do this in Course.js:

```
import { useState } from "react";
import Future from "./Future";
function Course(props){
```

9. Next, de-structure the two elements of the `useState()` function:

```
function Course(props){
  const [recommendation, setRecommended] =  useState(props.recommendation);
  function handleOnChange() {
```

The `useState()` function will always return the original property value **and** a method to update that value. When that inner function is called(the second element), the component in which it lives, gets re-painted onto the browser window. In this case the component is **Course**.

10. We can now update the *uRecommended* value via *recommendation* that we got from `useState()`. Then inside the original `handleChange()` function we use the second element of `useState()`, the function, to update that *uRecommended*:

```
function Course(props){
  const [recommendation, setRecommended] =  useState(props.recommendation);
  let uRecommended = recommendation;
  console.log(uRecommended);
  function handleOnChange() {
    setRecommended(!uRecommended);
    console.log(uRecommended);
  };
  return(
```

The two `console.log()` functions are there to check that this mechanism works, once you are satisfied that it works, remove the log lines.

PART 06 – JSX FORMS

We will no longer be using the Future.js file, so you can comment it out or just delete it. Also on Course.js remove/comment the line for least popular course. Remember to remove the *import* line for Future.js.

1. Create a new file NewCourse.js and define a form with 1 field and a submit button

```
function NewCourse(props){
  return(
    <form>
      <br /><hr /><br />
      <div>
        <label>Course Name: </label>
        <input
        type='text'
        />
      </div><br />
      <div>
        <button type='submit'>Add Course</button>
      </div>
    </form>
  )
}
export default NewCourse
```

2. Now we want to capture what use enters into fields. For this we can hook into the JSX `onChange` event and point that event to a handler function:

```
function NewCourse(props){
  const courseHandler = (event) => {
    console.log(event.target.value);
  }
  return(
```

For now, lets just log the value we get from the field

3. Also we need to handle form submission, so again we create a handler for that:

```
function NewCourse(props){
  const courseHandler = (event) => {
    console.log(event.target.value);
  }
  const submitHandler = (event) => {
    event.preventDefault();
  }
  return(
```

4. Now we want to capture what user enters into fields. For this we can hook into the JSX `onChange` event and point that event to a handler function:

```
      <form>
        <br /><hr /><br />
        <div>
          <label>Course Name: </label>
          <input
          type='text'
          onChange={courseHandler}
          />
        </div><br />
        <div>
```

We wrote the `courseHandler` function in #2

5. Now we point to the handler from our form upon submission:

```
      console.log(newCourse);
    }
  //
    return(
      <form onSubmit={submitHandler}>
        <br /><hr /><br />
        <div>
```

6. Import this NewCouse.js file in App.js and add it to the components display:

```
…
import Course from "./components/Course";
import NewCourse from "./components/NewCourse";
const courses = {
…
        <p></p>
        <Course
          mostPopular={courses.mostPopular}
          leastPopular={courses.leastPopular}
          futureCourse={courses.futureCourse}
          recommendation={courses.recommendation}
        />
        <NewCourse />
      </div>
    );
```

Note: not all the lines in this file are shown, only what is important for NewCourse

PART 07 – ADD STATE

In order to work with forms and fields, especiall if we want to use the values entered in other components, we must import and use **state**.

1. The form needs a special functional hook from the react library, it is called `useState()`. `useState` is a **Hook** that lets us add state to components

```
import { useState } from "react";
function NewCourse(props){
  const courseHandler = (event) => {
    console.log(event.target.value);
  }
```

`useState()` returns two values: the **current state** and a function that **updates** the current state. *We used this feature in Part05*.

2. In order to use the two elements, we must **de-structure** them like this:

```
import { useState } from "react";
function NewCourse(props){
  const [courseName, setCourseName] = useState('');
  const courseHandler = (event) => {
```

The first element is the state **property** while the second element is a **function** that will change that initial property value, if required.

3. We can now update state in the course handler:

```
function NewCourse(props){
  const [courseName, setCourseName] = useState('');
  const courseHandler = (event) => {
    setCourseName(event.target.value);
  }
```

4. At this point we can construct an object made up of the values of the form fields, we do this in the `submitHandler` function:

```
const submitHandler = (event) => {
  event.preventDefault();
  const newCourse = {
    courseName:courseName
  };
  console.log(newCourse);//remove later
}
return(
```

You can add a log line just to see if this code works, we can remove it later

5. Also we can complete the value field of the input box respectively:

```
<div>
  <label>Course Name: </label>
  <input
  type='text'
  onChange={courseHandler}
  value ={courseName}
  />
</div><br />
```

PART 08 – PASS DATA FROM FORM TO APP

At the moment, we have just the two courses and we display them individually. However we should use some kind of loop to do this, since the amount of array elements may change. We will soon be adding new courses.

We need to make a few changes before proceeding so that the demonstration is clearer and work with less code.

1. In App.js turn the **courses** object into an **array of objects**, also change the name from courses to courseList:

```
import NewCourse from "./components/NewCourse";
const courseList = [
  {
    courseName : "JavaScript",
  },
  {
    courseName : "C-Sharp",
  }
];
function App() {
```

2. Still in App.js import the useState hook:

```
import { useState } from 'react';
import './App.css';
import Course from "./components/Course";
import NewCourse from "./components/NewCourse";
```

3. Use the **useState** hook to extract the array data from **courseList**:

```
function App() {
 const[courseInfo, setCourseInfo] = useState(courseList);
 return (
   <div className="App">
```

4. Now when we call **Course**, we pass *courseInfo* instead of individual components:

```
            <p></p>
            <Course courses={courseInfo} />
            <NewCourse />
          </div>
```

Remember `courseInfo` was de-structured and now contains the `courseList`

5. Here is the entire function in App.js

```
import { useState } from 'react';
import './App.css';
import Course from "./components/Course";
import NewCourse from "./components/NewCourse";
const courseList = [
  {    courseName : "JavaScript",  },
  {    courseName : "C-Sharp",  }
];
//
function App() {
  const[courseInfo, setCourseInfo]=useState(courseInfo);
  return (
    <div className="App">
      <h2>Courses offered by Skillsoft</h2>
      <p></p>
      <Course courses={courseList} />
      <NewCourse />
    </div>
  );
}
export default App;
```

PART 09 – USING A LOOP TO DISPLAY COMPONENTS

1. In Course.js remove the pair of `<p>` tags that showed the least popular course and the future course:

```
          <div>
            <p>Course Name:
              <span className={uRecommended ? 'strongText' : 'normalText'}>
                {props.mostPopular}
              </span>
              <input
                type="checkbox"
                onChange={handleOnChange}
              />
            </p>
          </div
```

2. In Course.js the return statement needs to change in order to render the courses array:

```
return(
  <div>
    {props.courses.map()}
    <p>Course Name:
      <span className={uRecommended ? 'strongText' : 'normalText'}>
        {props.mostPopular}
      </sp
```

We now access the array of objects passed from App.js via **props.courses**

3. The `map()` function takes a function as an argument and this function will operate on **each** element in our **courses** data array:

```
return(
  <div>
    {props.courses.map(()=>()}
  </div>
)
```

We use the parenthesis instead of {} since we are processing JSX

4. Carefully cut all the code including the wrapper <p> tags and insert that block of code between the parenthesis of the map inner function:

```
return(
  <div>
    {props.courses.map(()=>(
      <p>Course Name:
        <span className={uRecommended ? 'strongText' : 'normalText'}>
          {props.mostPopular}
        </span>
        <input
          type="checkbox"
          onChange={handleOnChange}
        />
      </p>
    ))}
  </div>
)
```

This setup will print one line each of each course in `courseList`. There will be a warning in the Console window, we will fix it shortly.

5. Each course name is now locked inside of `props.courses`, so we have to extract this data from the inner function of map via a JS closure:

```
return(
  <div>
    {props.courses.map((course)=>(
      <p>Course Name:
        <span className={uRecommended ? 'strongText' : 'normalText'}>
          { props.mostPopular }
        </span>
```

6. Once we have access to each *course* object in the array passed from App.js we can access the values:

```
        return(
          <div>
            {props.courses.map((course)=>(
              <p>Course Name:
                <span className={uRecommended ? 'strongText' : 'normalText'}>
                  { course.courseName }
                </span>
```

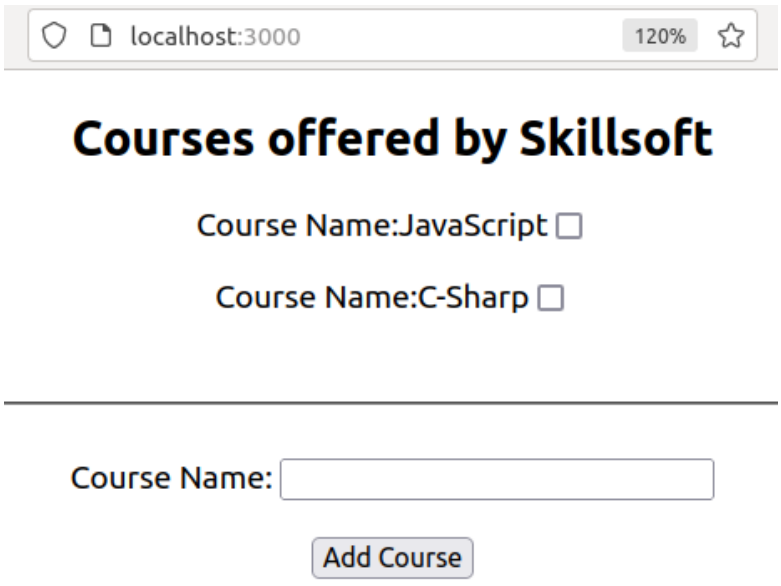
7. In order to remove the error we should identify each of our `div`s that contain content. A quick way to do this to access the *index* property that comes automatically with the `map()` method:

```
            <div>
             {props.courses.map((course, index) => (
               <div>
                 Course: <strong>{ course.courseName }</strong> 
               </div>
             ))}
```

8. Each time we iterate over the courses array, `map()` provides us with a count and React provides us with an attribute called *key*, we simply pass the count to key:

```
             {props.courses.map((course, index)=>(
               <p key={index}>Course Name:
                 <span className={uRecommended ? 'strongText' : 'normalText'}>
                   {course.courseName}
```

The warning should now go away. Also, the checkbox wont work properly, this can be a task for users to solve on your own.



PART 10 – ADDING A NEW COURSE

1. We would need to add a new course received via the form (NewCourse) to our existing array. This would be done in App.js since we have access to the form data after submission. We would need to add a function that triggers the update of the new course:

```
function App() {
  const[courseInfo, setCourseInfo]=useState(courseList);
  const saveCourse = newCourse => {

  };
  };
  return (
    <div className="App">
```

This function will be triggered from NewCourse.js. Remember it is App.js that calls Course.js and it is Course.js that shows the list of courses.

2. In `saveCourse` () we insert an inner function. This inner function called `setCourseInfo()` will have to handle adding the current course just entered by user, to the existing array:

```
  const[courseInfo, setCourseInfo]=useState(courseList);
  const saveCourse = newCourse => {
    setCourseInfo(prevCourseData => {

    });
  };
  return (
```

So, **prevCourseData** will represent the existing array of course objects. The function `setCourseInfo()` was de-structured from `useState()`.

3. In `setCourseInfo()` we return the new data, attached to the existing data:

```
  const[courseInfo, setCourseInfo]=useState(courseList);
  const saveCourse = newCourse => {
    setCourseInfo(prevCourseData => {
      return [newCourse, ...prevCourseData];
    });
  };
  return (
```

So we combine previous data with what was sent by the form.

4. Now when we call **NewCourse** component, we pass a pointer to the `saveCourse()` function on App.js:

```
      <p></p>
      <Course courses={courseInfo} />
      <NewCourse onSaveCourse={saveCourse}/>
    </div>
  );div>
      ))};
```

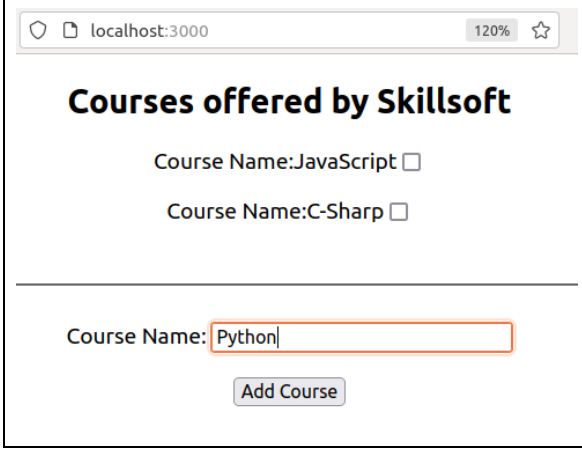
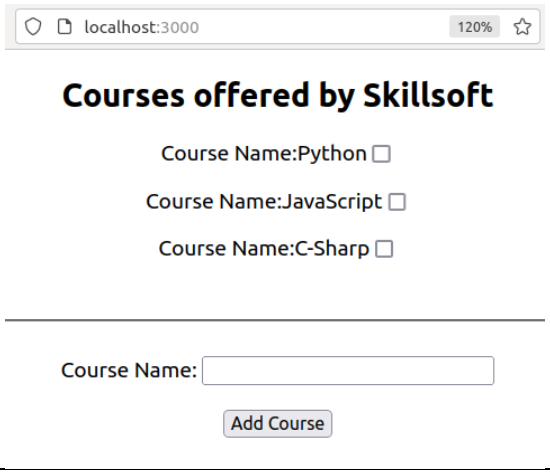
5. Over in the NewCourse.js file, we trigger the `saveCourse()` function in App.js by simply calling it via *props*:

```
    }
    const submitHandler = (event) => {
      event.preventDefault();
      const newCourse = {
        courseName:courseName
      };
      props.onSaveCourse(newCourse);
      setCourseName('');
    }
  //
    return(
```

We can also clear out any values at the same time. So props not only can pass properties/values, it can also pass functions.

6. We can remove any `console.log()` lines we had in the various files and add a new course. We could add a conditional statement, if the array is empty we could deliver different content to the browser. Do this in Course.js:

```
    };
    if(props.courses.length === 0){
      return(<div>No Data</div>)
    };
    return(
      <div>
```

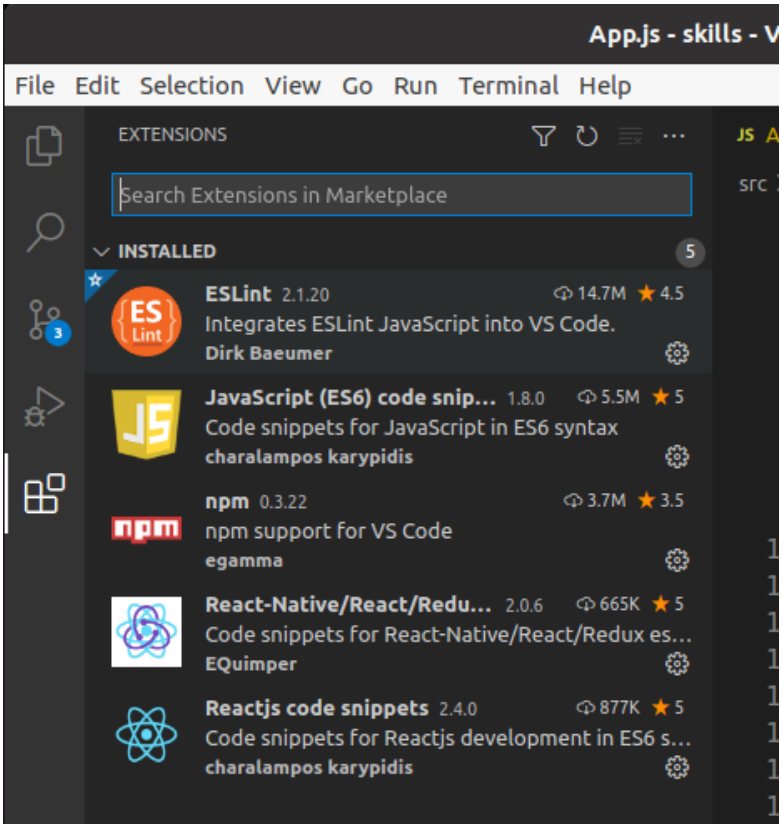
Adding Python:	After:
	

Appendix A – Pre-requisite Courses

1. Asynchronous Programming in JavaScript - Ron Sumida
   This course references asynchronous code in JS
2. JavaScript: Objects - Kishan Iyer
   This course references the ***this*** keyword in JS
3. JavaScript Front-end Development: Functions & Objects - Axle Barr
   References arrow functions (lambda expressions)

Appendix B – ES6 Code Snippets for VS Code

1. Ctrl + Shift + P
2. Install extensions -> search for the following and install them via VS Code

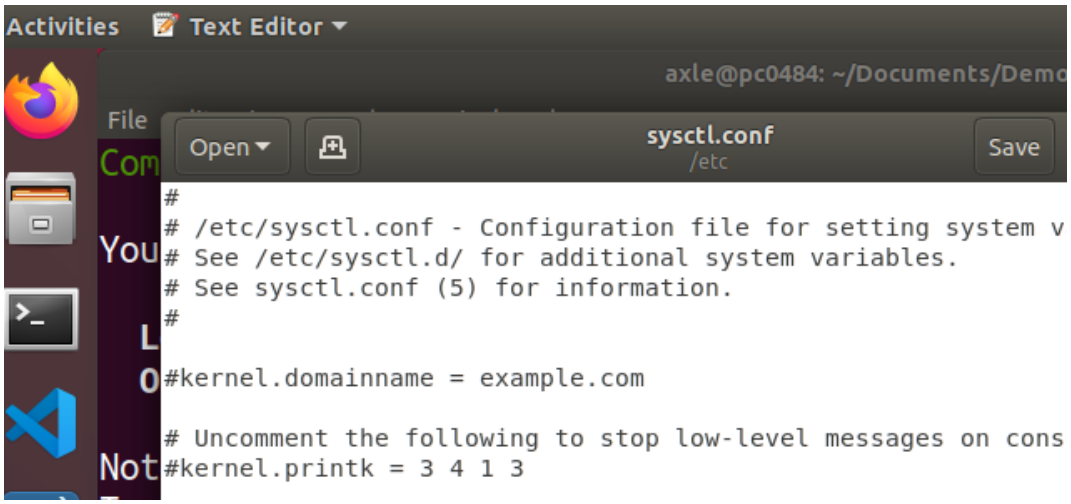


APPENDIX C – FILE WATCHERS LIMIT REACHED (ENOSPC)

1. If you are on Linux you may get an error about file watchers limit, this is due to the number of times that React recompiles and hotloads. Hidden from view is a package called inotify that Linux systems use to track this sort of activity. You would need to adjust this value using the steps below:

Run the following command at any terminal prompt: **sudo gedit /etc/sysctl.conf**

This would bring up the sysctl.conf file for editing



Add a new line with this value: **fs.inotify.max_user_watches=524288**

Make sure that there is no # sign in front of the line.

Save the file and try restarting the app with **npm start**

To solve this, at least temporarily, run the following command:
sudo sysctl -w fs.inotify.max_user_watches=524288