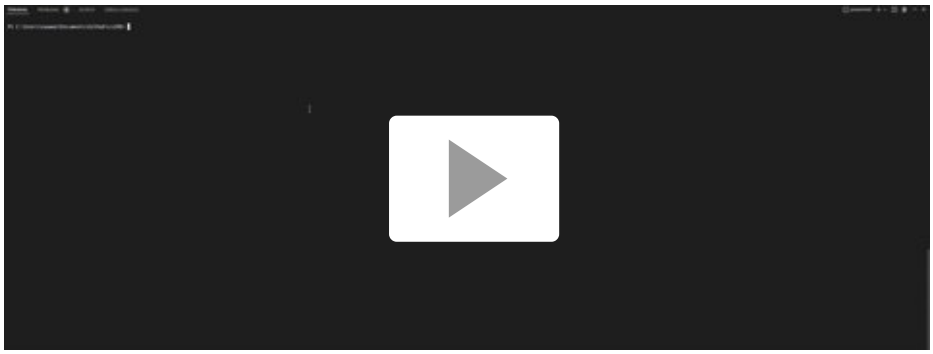
Exploration — React Components & JSX

In this exploration, we study how to write **components** using JSX.

React component files are like 'snippets' of code...not complete pages or complete functions, but just 'parts' in separate files that can be 'included' in other files to make a larger part or whole function or page. It allows large groups of programmers to work on small chunks independently of other chunks, which reduces errors.

We will also look at a feature of JavaScript called **destructuring expressions** because these are commonly used in React. We will frequently encounter example code that uses this feature, so it is well worth to understand it. After studying destructuring, we study what is JSX and how we use to write React components.

Destructuring Expressions



Object Destructuring

Consider an object `book` with two properties `title` and `price`. We can assign the value of these two properties to 2 variables `title` and `price` with two assignment statements as follows:

```
const book = { title : "Modern JavaScript", price: 21.99};
const title = book.title; // Value of title is set to "Modern JavaScript"
const price = book.price; // Value of price is set to 21.99
```

However, object destructuring expressions allow us to assign the members of an object to multiple variables **in just one expression.**

Declaring Variables and Assigning Values from an Object in One Expression

We can declare the variables and assign them values in the same destructuring expression. Here is an example of this:

```
const book = { title : "Modern JavaScript", price: 21.99};
const {title, price } = book; // This sets value of title to "Modern JavaScript" and price to 2
1.99
```

In the expression, `const {title, price } = book;`, we declare two variables `title` and `price`, and also assign them the values of the corresponding properties of the object `book`. Note that this requires that the **names of the variables must be the same the names of the properties** of the object that we want to destructure. JavaScript provides a different syntax that allows the names of the variables to be different from the properties in the object we want to destructure. But we will not be using that in the course. You can find more details about this in the textbook or at the MDN **page on destructuring assignments (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment)**.

Array Destructuring

Consider an array `vals` with 3 elements. We can assign the values of each of the 3 elements in this array to the 3 variables `a`, `b` and `c` using 3 assignment statements as follows:

```
const vals = [87, 42, 56];
const a = vals[0]; // Value of a is set to 87
const b = vals[1]; // Value of b is set to 42
const c = vals[2]; // Value of c is set to 56
```

However, array destructuring expressions allows us to use one expression to assign the value of multiple elements in an array to multiple variables.

Declaring Variables and Assigning Values from an Array in One Expression

We can declare the variables and assign them values in the same destructuring expression. Here is an example of this where we assign the value of the element at index 0 in the array `vals` to the variable `a`, the element at index 1 to the variable `b` and the element at index 2 to the variable `c`.

```
const vals = [87, 42, 56];
const [a, b, c] = vals; // Value of a is set to 87, b to 42 and c to 56
```

In the expression, `const [a, b, c] = vals;`, we declare three variables `a`, `b` and `c` and also assign them the values of the elements at index 0, 1 and 2 in the array `vals`.

JSX

JSX is a combination of JavaScript and XML. XML is a markup language which looks similar to HTML. We do not need to understand the details of XML to understand and write code in JSX. JSX allows us to define new components, which we can then use as tags along with existing HTML tags.

One difference with HTML is that in JSX every tag must be closed with either <componentName> </componentName> or with <componentName />. We can embed JavaScript expressions in JSX, and use these to pass JavaScript values to components. The JavaScript expressions we can use in JSX include primitives, arrays, objects, and functions. Other than string literals, all other types of expressions need to be surrounded by `{}` in JSX. A component.js file's function gets exported, and its name becomes a <componentName /> in another component file's function, or gets used in a page.js. So, 1) we import dependencies, then 2) call the function with a name, 3) return the HTML and/or <component /> tags, 4) then export that function to be used in other functions.

Example: Hello World

In the following example, we have modified the default `App` component created by `create-react-app` to print a "Hello World" message along with the current time.

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

function App() {
  const name = "Nauman";
  return (
    <div className="App">
      <main className="App-main">
        <p>Hello {name}!</p>
        <p>The current time is {Date()}</p>
      </main>
    </div>
  );
}

export default App;
```

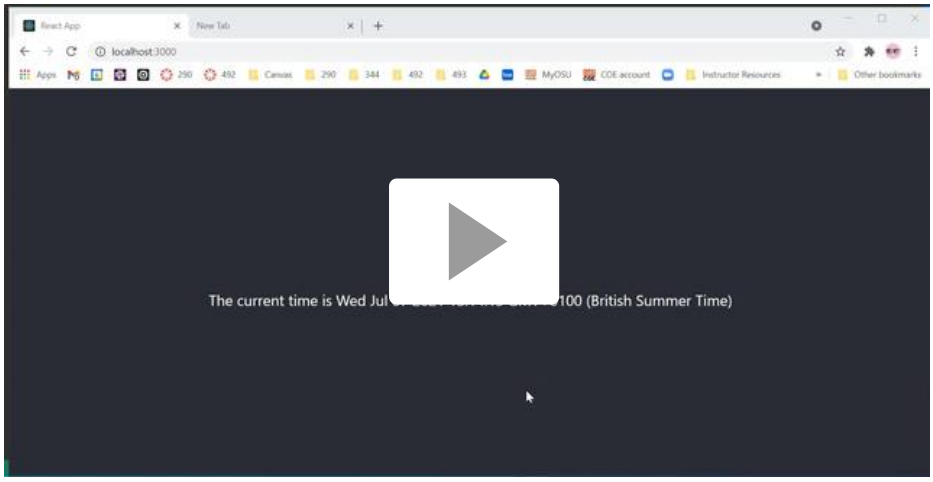
This example defines a React component named `App`. Recall that React components are functions that return JSX. This component is imported into the file `index.js` and is rendered by the browser as described in the previous exploration. We highlight a few things about JSX:

- Property name is `className` rather than `class`
  - HTML elements have an attribute `class`. However, `class` is a reserved word in JavaScript. Therefore, JSX uses `className` for the `class` attribute.
- JavaScript expressions are wrapped in curly braces, i.e., `{}`.
  - The `App` component has two JavaScript expressions, `{name}` and `{Date()}`.
  - We cannot use statements, such as, `if` or `for` within the curly braces.

Defining Components

We aim to define simple components in React. This leads to modular code as well as to reusability. Let us refactor our `App.js`. We will define a new component `Greeting` to print the greeting message and use this component in `App.js`.

> We encourage you to type along with the video to create the greeting React app. However, the app is also available in this file: **greeting.zip. (https://canvas.oregonstate.edu/courses/1879154/files/93929037?wrap=1)** (https://canvas.oregonstate.edu/courses/1879154/files/93929037/download?download_frd=1) To run the app, download the zip file, unzip it in a directory, go to that directory and run `npm install` and `npm start`.



Example: Greeting Component

Here is our modified `App.js` file that uses the component `Greeting` which is defined in the file `Greeting.js`.

```
import './App.css';
import Greeting from './Greeting';

function App() {
  const name = "Nauman";
  return (
    <div>
```

```
      <p
        <Greeting name={name} time={Date()}></Greeting>
      </p>
    </div>
  );
}

export default App;
```

We import the component using an `import` statement and use it just as we use the tag for a built-in HTML element. There are two attributes of the tag `Greeting`, i.e., `name` and `time`. These attributes will be passed to the function implementing the `Greeting` component as the properties of a JSON.

Here is the file `Greeting.js` that defines the `Greeting` component.

```
import React from 'react';

function Greeting(props) {
    return (
    <div>
      <p>Hello {props.name}!</p>
      <p>The current time is {props.time}</p>
    </div>
  );
}

export default Greeting;
```

We define a function `Greeting` which is passed one argument and which returns a React component with a `div` element containing two `p` elements. JSX expressions are used to get the values of the properties `name` and `time` from the argument, which we have named `props`.

Example: Using Object Destructuring

It is typical to use object desctructuring to assign the values of attributes passed to a component. Here is the above example rewritten using object destructuring where the JSON object passed to the function `Greeting` is destructured into the variables `name` and `time`.

```
import React from 'react';

function Greeting({ name, time }) {
    return (
      <div>
        <p>Hello {name}!</p>
        <p>The current time is {time}</p>
      </div>
    );
}

export default Greeting;
```

React Fragments

React does not allow two or more sibling elements as a component. For example, to take the previous example, we could not define `Greeting` as follows because the following function returns

two sibling `p` elements:

```
import React from 'react';

function Greeting({ name, time }) {
    return (
      <p>Hello {name}!</p>
      <p>The current time is {time}</p>
    );
}

export default Greeting;
```

This is why we had wrapped these two `p` elements in a `div`. Adding these `div` elements leads to unnecessary tags. React supports the notion of **React fragments** for these situations. A React fragment is enclosed in the tag `<React.Fragement>` or simply the empty tag `<>`, and can contain multiple sibling elements. Here is the `Greeting` component rewritten using a React fragment that contains the two `p` elements as siblings, but does not have to wrap them into a top level `div` element.

```
import React from 'react';

function Greeting({ time, name }) {
    return (
      <>
        <p>Hello {name}!</p>
        <p>The current time is {time}</p>
      </>
    );
}

export default Greeting;
```

Structure of React Apps

We just created a React app that used a component, Greeting, that was defined by us. We will now review this app to highlight the high-level structure which is common to React apps. Understanding this structure will help us understand existing React apps, and design and implement new React apps.

Component Tree

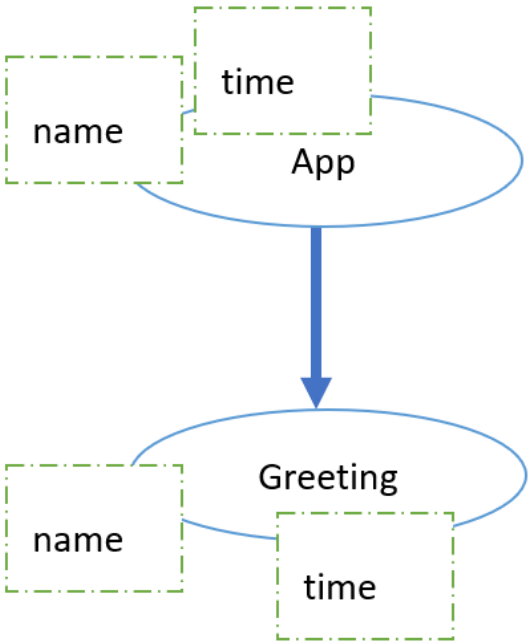
Components in a React app are arranged in a tree structure with the `App` component as the root of the tree. When we are thinking about a React app (either to understand an existing app or to design a new one), it is helpful to draw the components in the app as a tree to visualize the app at a high level.

Example: In our example Greeting app, the component tree has just 2 components. The `App` component is at the root of the tree and it has one child, the `Greeting` component. However, typical React apps will have many components and the component tree would be more complex.

Passing Data Via Properties

In React apps, data can be easily passed down from a parent component to a child component. This mechanism is built into React because when instantiating a child component, the parent component can pass data by setting the attributes of the child component. Thus data can flow along the React component tree from a node to any of its descendant nodes. This is called “top-down” or “unidirectional” data flow.

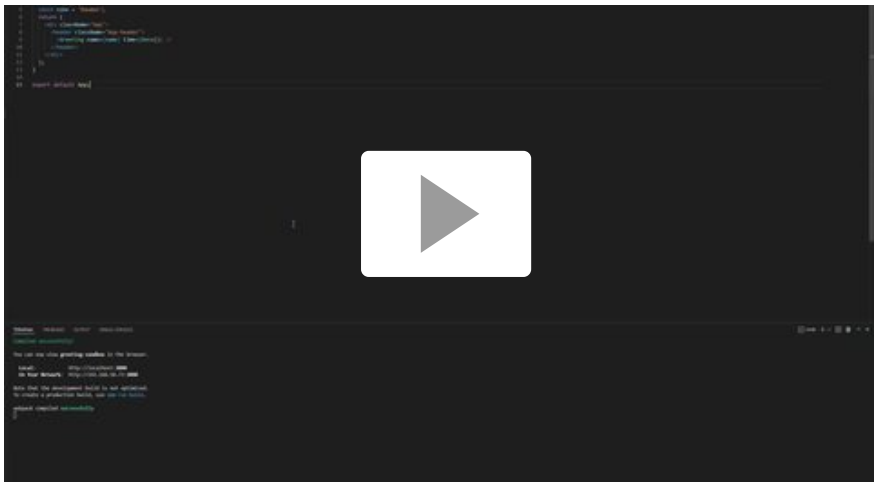
Example: In our example Greeting app, the `App` component passes down two properties, `name` and `time`, to the `Greeting` component.



The React component tree for the Greeting example. The root component is `App` and it has one child component, `Greeting`. `App` passes down two properties `name` and `time` to the `Greeting` component.

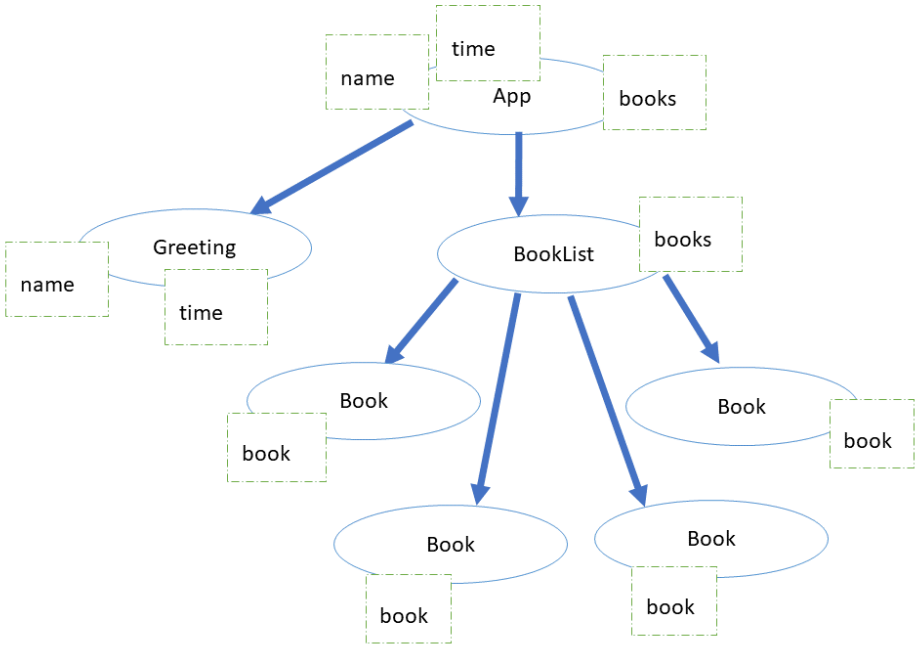
Common Errors and Debugging React Apps

Coding errors related to object destructuring are quite common when we start writing React apps. Let us look at some common errors and the tools we can use to debug these common errors.



Example: Book and BookList Components

Creating webpages that contain collection of items is a very common use case. Let us work through a detailed example of using React to create a webpage that display a collection of books.



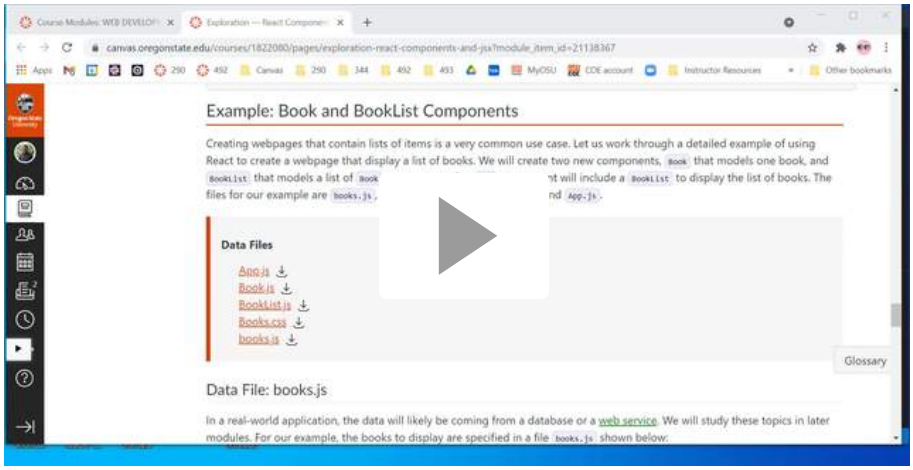
The React component tree for the BookList example. The root component is `App` and it has 2 child components, `Greeting` and `BookList`. `App` passes down two properties `name` and `time` to the `Greeting` component. `App` passes down one property `books`, which is an array, to the `BookList` component. The `BookList` component creates as many `Book` components as there are elements in the array `books`. The `BookList` component passes down one `book` object from the `books` array to each of the `Book` components.

We will create two new components, `Book` that models one book, and `BookList` that models a collection of `Book` components. Our `App` component will include a `BookList` to display the collection of books. Note that the component has the word "list" in the name. However, here the word "list" is used in the sense of a "collection," rather than referring to an HTML list.

Code and Data Files

The files for are examples are as follows

App.js (https://canvas.oregonstate.edu/courses/1879154/files/93831900?wrap=1)
(https://canvas.oregonstate.edu/courses/1879154/files/93831900/download?download_frd=1)
Book.js (https://canvas.oregonstate.edu/courses/1879154/files/93831898?wrap=1)
(https://canvas.oregonstate.edu/courses/1879154/files/93831898/download?download_frd=1)
BookList.js (https://canvas.oregonstate.edu/courses/1879154/files/93831899?wrap=1)
(https://canvas.oregonstate.edu/courses/1879154/files/93831899/download?download_frd=1)
Books.css (https://canvas.oregonstate.edu/courses/1879154/files/93831993?wrap=1)
(https://canvas.oregonstate.edu/courses/1879154/files/93831993/download?download_frd=1)
books.js (https://canvas.oregonstate.edu/courses/1879154/files/93831929?wrap=1)
(https://canvas.oregonstate.edu/courses/1879154/files/93831929/download?download_frd=1)



Data File: books.js

In a real-world application, the data will likely be coming from a database or a web service. We will study these topics in later modules. For our example, the books to display are specified in a file `books.js` shown below:

```
const books = [
    {
        title: 'Modern JavaScript for the Impatient', price: 26.55
    },
    {
        title: 'Learning React', price: 20.88
    },
    {
        title: 'Web Development with Node and Express', price: 30.99
    },
    {
        title: 'HTML and CSS: Design and Build Websites', price: 17
    }
];

export default books;
```

This file exports an array of JSON objects, each of which has the properties `title` and `price`. Place this file in a new directory `data` under the `src` directory.

The React Component: App

We modify `App.js` to import the data file `data/books.js` and assign it to the variable `books`, and the component `BookList` from the file `BookList.js`. The `App` component has two child components, the `Greeting` component and right after it the `BookList` component. The `App` component passes the data from the data file to the `BookList` component as the property named `books`.

```
  import './App.css';
  import Greeting from './Greeting';
  import BookList from './BookList';
  import books from './data/books.js';

function App() {
  const name = "Nauman";
  return (
    <div className="App">
      <main>
        <Greeting name={name} time={Date()}></Greeting>
        <BookList books={books}></BookList>
      </main>
    </div>
  );
}

export default App;
```

The React Component: BookList

The React component `BookList` is in the file `BookList.js` shown below:

```
import React from 'react';
import Book from './Book';
import './Books.css';

function BookList({ books }) {
    return (
      <div className="list-container">
        {books.map((book, i) => <Book book={book} key={i} />)}
      </div>
    );
}

export default BookList;
```

- The function `BookList` receives an object with one property that is named `books` which is the array of objects from the data file.
  - Note that we are using object destructuring to assign the property `books` of the object passed as the argument to the function `BookList` to the parameter named `books` in this function.
- It creates a `div` element.
- The contents of this `div` element are created by calling the `map` method on the array `books`. Note that in React, collection of components are created almost always using the `map` function.
- The `map` method is supplied the following arrow function `(book, i) => <Book book={book} key={i}`.
- `map` will call this arrow function on each object in the array `books` to create one `Book` component for each book in the array.
- Since we are using `Book` component in this file, we need to import this component.
- The `Book` component is passed two parameters,
  - One is `book` which corresponds to the current element in the `books` array and
  - The other is `key` which is set to the index `i` of the element in the `books` array.
- React requires that whenever we use map to display a list of components, we must assign a unique key to each component in the list. The key argument is thus for internal use of React and the `Book` component code does not need to bother with it.
- Note that we imported styles from the CSS stylesheet `Books.css` by using an `import` statement.

> **Note:** The BookList component uses the map method that was discussed in **Exploration — Functions and Functional Programming (https://canvas.oregonstate.edu/courses/1879154/pages/exploration-functions-and-functional-programming)** . The map method is passed an anonymous function expression defined using arrow function syntax. This topic was also discussed in the same exploration. If needed, review the relevant sections in that exploration. You can also find further **discussion of the**

> map method on MDN (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map).

The React Component: Book

The React component `Book`, in the file `Book.js`, displays a single book and is shown below:

```
import React from 'react';
import './Books.css';

function Book({ book }) {
    return (
      <div className="list-item">
        <h3>{book.title}</h3>
        <p>Price: {book.price}</p>
      </div>
    );
}

export default Book;
```

- The function `Book` receives an object with one property `book`, whose value is assigned to the parameter `book` using object destructuring. The value of this property is an object corresponding to one element of the array in `data/books.js`.
- The function `book` creates a `div` element that includes an `h3` heading with the title of the book and a `p` element listing the price of the book.

Summary

In this exploration, we looked at JSX and React components. We defined new components by creating new files that contain a function that return JSX. We exported the component from this file and imported the component in other files where we want to use the component. We looked at how to create a collection of React components using the `map` function. For reference we note that older versions of React created components using a class-based syntax, rather than the function-based syntax that we studied. Almost all new React code is written using function-based syntax. However, be aware that you come across legacy React code that uses the class-based syntax for defining components.

Additional Resources

Here are some references to learn more about the topics we discussed in this exploration.

- Destructuring expressions are described here (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment) on MDN including advanced topics that we didn't discuss and will not use in the course.
- Introduction to JSX (https://reactjs.org/docs/introducing-jsx.html) on the official React homepage.

- Discussion of **components and properties (https://reactjs.org/docs/components-and-props.html)** .
- React docs on use of **keys with lists (https://reactjs.org/docs/lists-and-keys.html)** .