React:

Week 1 Workshop Presentation

Workshop Agenda

Activity	Estimated Duration
Set Up & Check-In	10 mins
Week 1 Review	60 mins
Assignment	60 mins
Break	15 mins
Assignment	80 mins
Check-Out (Feedback & Wrap-Up)	15 mins

Set up

Along with Zoom, please also be logged into Slack and the learning portal!

Introductions & check-in

Introductions

Please introduce yourselves briefly!

Check-In

How was this week? Any challenges or accomplishments?

Did you understand the Exercises and were you able to complete them?

You must complete all Exercises before beginning the Workshop Assignment.

Week 1 Review

Overview

React course overview	JSX
React course tips	create-react-app
Callbacks & higher-order functions	React components
Array.map() method	React components and JSX
Arrays of objects	React components and props
The DOM	I Heart React
React is declarative	

React course overview

- This will be the hardest course in the bootcamp, be prepared!
- Expect to work on: Same **nucampsite** project from Bootstrap - *React*-ified
- Beginning next week, your same Portfolio Project – *React*-ified
- Give yourself at *least* 2 hours a day to watch lectures, do exercises and challenges, additional study & practice concepts – plus more time for your portfolio project.
- 20-minute rule - Reach out during the week if you'd like help understanding something. Don't wait until the workshop!

React course tips

Accept the notion of a “Black Box”

- You will find yourself using code without knowing exactly how it works at first.
- Do your best, don't stress if it doesn't make sense immediately.
- You will repeat the same tasks over and over again and come to understand them better over time!
- Take it upon yourself to research more advanced JavaScript online, bit by bit, one concept at a time – build upon existing knowledge.
- Difficult learning curve at first, very rewarding afterward!

Callbacks & higher-order functions (1/3)

JavaScript has "**first-class functions**".

- This means that functions can be treated like variables/values.
- They can be passed into another function as an argument, and they can be returned from another function as a return value.

Callbacks & higher-order functions (2/3)

A function that is passed into another function, or returned from another function, is called a **callback function**, or just **callback.**

A function that takes a function into it as an argument, or returns another function from itself, is called a **higher-order function**.

These are programming concepts/terms that extend beyond just JavaScript or React.

Callbacks & higher-order functions (3/3)

Example:

```
// Declaration of a higher-order function:
const aHigherOrderFn = (aCallbackFn) => {
    console.log('This higher-order function "calls back" a callback function.');
    aCallbackFn();
}
// Invoking that higher-order function:
aHigherOrderFn(() => console.log('This is the body of that callback function.'));
```

Running the code above will result in the console.log() from the higher-order function's body running, followed by the console.log() from the callback function's body running.

The advanced array methods you learned this week are also examples of invoking higher-order functions.

Array.map() (1/3)

The **Array.map()** method, for example, takes a **callback** function as an argument:

```
const myArr = [1, 2, 3];
const squared = myArr.map((x) => x * x);
```

When using **callback** functions, the **arrow function** syntax is typically used to write it as an **anonymous callback** function, which is defined directly inside the parameter list without the function being assigned to a variable name.

In this case, the **callback** function is: **(x) => x * x**

Array.map() (2/3)

```
const myArr = [1, 2, 3];
const squared = myArr.map((x) => x * x);
```

The **map()** method will take the provided function and automatically loop it as many times as the length of the array it is invoked upon.

In this case, that array is **myArr**, and its length is **3**.

Each time the function runs, its first parameter (in this case, **x**) takes as its argument the next item from **myArr**:

First time: **x = myArr[0]**, second time: **x = myArr[1]**, third time: **x = myArr[2]**

The return value from each iteration is saved into a new array, which at the end of the iterations is returned from the **map()** method.

The value ultimately returned and saved into the **squared** variable is the array **[1, 4, 9]**

Array.map() (3/3)

```
const myArr = [1, 2, 3];
const squared = myArr.map((x) => x * x);
```

Since map() is a non-mutating method, it does not change the original array, myArr.
After the above code has run, the value of **myArr** will still be **[1, 2, 3]**, and the value of **squared** will be **[1, 4, 9].**

This is also an example of a **pure function**, which will always return the same outputs when given the same inputs and does not cause any side effects.

Arrays of objects (1/2)

```
const data = [
  {
    name: 'Butters',
    age: 3,
    type: 'dog'
  },
  {
    name: 'Lizzy',
    age: 6,
    type: 'dog'
  },
  {
    name: 'Red',
    age: 1,
    type: 'cat'
  },
  {
    name: 'Joey',
    age: 3,
    type: 'dog'
  }
];
```

Arrays are often used to contain objects.

Discuss: How would you use bracket notation to access the **age** property of the first item in the **data** array? (answer in next slide)

Arrays of objects (2/2)

```
const data = [
  {
    name: 'Butters',
    age: 3,
    type: 'dog'
  },
  {
    name: 'Lizzy',
    age: 6,
    type: 'dog'
  },
  {
    name: 'Red',
    age: 1,
    type: 'cat'
  },
  {
    name: 'Joey',
    age: 3,
    type: 'dog'
  }
];
```

Arrays are often used to contain objects.

Discuss: How would you use bracket notation to access the **age** property of the first item in the **data** array?

Answer: **data[0].age**

The DOM (1/2)

The DOM stands for **Document Model Object**.

- It is a way to model a webpage using the concept of objects, allowing us to use JavaScript to access and manipulate the elements of a webpage.
- The **document** object represents the webpage.
- We can use JavaScript methods on the **document** object to manipulate the webpage.

The DOM (2/2)

Example:

- The code below allows us to find the first element in a webpage that has the **id** of **'test'**, regardless of whether that element is a **div** or **span** or anything else:

  document.getElementById('test')

- We can manipulate that element with JavaScript. For example:

  document.getElementById('test').style.background = 'red';

- This would turn the background color of the element with the id of **'test'** to **red**.
- This is how React uses JavaScript to control the way webpages are rendered.

React is declarative

- DOM manipulation is performed under the hood by the React library.
- You don't actually manipulate the DOM yourself using React.
- You use React to declaratively describe what you want the webpage to do.
- React creates a virtual DOM based on your code.
- It then uses a process called **reconciliation** to **diff** (compare differences between) then match up the actual DOM with the **virtual DOM**.
- This allows React to efficiently apply updates based on only what has changed.

JSX (1/2)

- Under the hood, React uses **React.createElement()** to create React elements.
- React provides a syntax extension called **JSX** that is **syntactic sugar** over the **React.createElement()** method.
- Instead of using **React.createElement()** ourselves, which is more complicated, we can write **JSX**.
- JSX intentionally resembles HTML, though it is not HTML. It is **transpiled** (translated from one syntax to another) into JavaScript, which is then used to create HTML.

JSX (2/2)

- One difference between JSX syntax and HTML syntax is that the HTML **class** attribute is called **className** in JSX. This is because the word **class** already has a meaning in JavaScript that is different from the HTML/CSS meaning of **class**.

Example: HTML: ** **something** **
JSX: ** **something** **

React components

- React applications are mainly composed from **components**, reusable blocks of code conceptually similar to **functions**.
- We can create **components** using JavaScript functions or classes.
- A simple React **function component** can look like this:

```
function Hello() {
    return (
        <div>Hello world</div>
    );
}
```

- We can also use arrow functions to create function components, like this:

```
const Hello = () => {
    return (
        <div>Hello world</div>
    );
};
```

React components and JSX

- In this code example, the **h1** element in the return block is a JSX element:

```
const Hello = () => {
    return (
        <h1>Hello world</h1>
    );
};
```

- We could define another React component like this, which renders the **Hello** component inside a **div** as its return value:

```
const App = () => {
    return (
        <div className='myClass'>
            <Hello />
        </div>
    );
};
```

React components and props (1/2)

- The **props** object is a special object in React. We use it to pass **read-only** data from one component to another. Example: The **App** component can pass a name to the **Hello** component like this:

```
const App = () => {
  return (
    <div className='myClass'>
      <Hello name='Ludo' />
    </div>
  );
};
```

- This creates an object named **props** with a property of **name**, with its value as **'Ludo'**.
- We can pass this object to the **Hello** component and use it there like so:

```
const Hello = (props) => {
  return (
    <h1>Hello {props.name}</h1>
  );
};
```

React components and props (2/2)

- Note that in this example from the previous slide, **props.name** is a JavaScript variable and not JSX. Thus, since we are using it inside where JSX is expected (in the return statement of a React component), we must wrap it in curly braces to indicate that it is JavaScript:

```
const Hello = (props) => {
   return (
      <h1>Hello {props.name}</h1>
   );
};
```

I Heart React

- Recall this code from the final I Heart React project:

```
const HeartsList = () => {
    return (
        <div className='hearts-container'>
            {messages.map((message, index) => (
                <CandyColoredHeart key={index} text={message} />
            ))}
        </div>
    );
};
```

- Discuss: What is your understanding of what this code accomplished?

Workshop 1 Assignment

In this workshop, you will create a **CampsiteCard** component for your **nucampsite** project, then a **CampsitesList** component to render multiple **CampsiteCard** components using an array of objects (**CAMPSITES**). You will also create **Header** and **Footer** components for the app.

You must have finished the exercises in the book **The Course Project** before you can begin the Workshop Assignment.

You will be split up into groups to work on the assignment together.
Talk through each step out loud with each other, code collaboratively.
If your team spends more than 10 minutes trying to solve one problem,
ask your instructor for help!

Check-out

Submit your final workshop files (both the HTML and JS files) via the Nucamp Learning Portal.

Wrap up – Retrospective:

- What went well?
- What could improve?
- Action items?

Start on Week 2 if there's time, or continue discussing any Code Challenges/Quiz questions/any questions that remain from this week's materials.