Week 1

Tuesday 09/24/24: Starting the React Bootcamp

Intro to JSX

- Hello World
    - Code belongs to a javascript file
    - The part that looks like HTML is called JSX
- What is JSX
    - A syntax extension for JavaScript
        - Syntax extension means that web browsers can't read it
    - If a javascript file contains JSX, then that file needs to be compiled

JSX Elements

- A basic unit of JSX
    - Example: Hello world

JSX Elements and Their Surroundings

- JSX elements are treated as JavaScript expressions
    - Can be saved in a variable, passed to a function, stored in an object or array, etc

Attributes in JSX

- JSX elements can have attributes
    - Stored within the first angled brackets -> <>

Nested JSX

- Can nest JSX elements inside of other JSX elements
- If a JSX expression takes up more than 1 line, must wrap it in parenthesis
- Nested JSX expressions can also be saved to variables, passed to functions etc

JSX Outer Elements

- A JSX expression must have exactly one outermost element
- The first opening tag and the final closing tag of a JSX expression must belong to the same JSX element

Rendering JSX

- To render means to make it appear on screen
- React relies on two things to render
    - What content to render
    - Where to place the content

```
const container = document.getElementById('app');
const root = createRoot(container);
root.render(<h1>Hello world</h1>);
```

- First line
    - Uses the document object, which represents web page
    - Uses the getElementbyID() method of document to get the Element object representing the HTML element with the passed in ID (app)
    - Stores the element in container
- Second line
    - Use createRoot() from the react-dom/client library, which creates a React root from container and stores it in root. root can be used to render a JSX expression. This is the “where to place the content” part of React rendering
- Last line
    - uses the render() method of root to render the content passed in as an argument. Here we pass an <h1> element, which displays Hello world. This is the “What content to render” part of React rendering

Passing a Variable to render()

- The render() method’s argument does not need to be JSX, but it should evaluate to a JSX expression.
    - Arg could also be a variable, so long as that variable evaluates to a JSX expression

The Virtual DOM

- React’s render() only updates DOM elements that have changed
    - Means: if you render the exact same thing twice in a row, the second render will do nothing
- Significant because updating the necessary DOM elements is a large part of what makes React so successful

REVIEW:

- React is a modular, scalable, flexible and popular front-end framework
- JSX is a syntax extension for JavaScript
    - They can be stored in variables, objects, arrays and more
- JSX elements can have attributes and be nested within each other
- JSX must have exactly one outer element, and other elements can be nested inside
- createRoot() from react-dom/client can be used to create a React root at the specified DOM element
- A React root's render() method can be used to render JSX on the screen
- A React root's render() methods only updates DOM elements that have changed using the virtual DOM

Friday: 09/27/24: Continuing React Bootcamp

The Problem: DOM manipulation is at the heart of modern interactive web, but it's slow, and made worse by the inefficient way JavaScript frameworks update the DOM.

- Let's say there's a list with 10 items. You want to check off the first item. 1 item is changed in the list, but most Javascript frameworks rebuild the entire list: UNNECESSARY

Solution: React uses something called the Virtual DOM. In React, for every DOM object, there's a corresponding virtual DOM object (a lightweight copy). Manipulating the virtual DOM is much faster because nothing gets drawn on screen.

How the solution helps

- When a JSX element is rendered, every single DOM object gets updated, sounds inefficient, but cost is insignificant because the virtual DOM can update so quickly
- Once virtual DOM updated, React compares the virtual DOM with a virtual DOM *snapshot* that was taken right before the update.
- By comparing the new virtual DOM with pre-updated version, React sees exactly which Virtual DOM objects have changed, called *diffing*
- Then React updates only those DOM objects that have changed on the real DOM

class vs className

- In JSX, use className instead of class

Self-Closing Tags

- Self-closing Tag: Some HTML elements such as <img> and <input> only use one tag. The tag belongs to a single-tag element is not an opening tag or a closing tag; it's a self closing tag
- In JSX, you have to include the forward slash when writing a self closing tag

JavaScript in you JSX in your Javascript

- Regular JavaScript, written inside of a JSX expression, written inside of a JavaScript file

```
root.render(<h1>2 + 3</h1>)
```

    - This shows 2+3, not 5
    - This is because any code in between <h1> and </h1> will be read as JSX, not javaScript
    - In order to get JavaScript, use curly braces!

Week 2

Tuesday: 10/01/24: Continuing the Bootcamp

Variables in JSX

- When you have JavaScript embedded in JSX, that JavaScript is part of the same environment as the rest of the JavaScript in the file
    - Means you can access variables while inside of a JSX expression, even if those variables were declared outside of te JSX code block

Variable Attribites in JSX

- When writing JSX, common to use variables to set attributes
    - Can also use object properties to set attributes
    - Make sure to wrap variable names and object properties in curly braces, so that file reads it as JavaScript in JSX
- Remember that when you render, to put the variable of the thing you want to render in the parenthesis

Event Listeners in JSX

- JSX elements can have event listeners
- Can create an event listener by giving a JSX element a special attribute:
  - `<img onClick={clickAlert} />`
- An event listener attributes name should be something like onClick or onMouseOver
  - The word one plus the type of event that you're listening for
- An event listener attribute's value should be a function. The above example would onle work if clickAlert were a valid function that had been defined elsewhere
- In JSX, event listener names are written in camelCase

JSX Conditionals

- You can not nject an if statement into a JSX expression
  - There are other ways to have it so things only render if a certain condition is met
- One option is to write an if statement and not inject it into JSX, have the conditional wrap the JSX element.
- The Ternary Operator
  - A more compact way to write conditionals in JSX
  - Example: x ? y : z
    - x, y, and z are all JavaScript expressions
    - When code is executed, x is evaluated as either "truthy" or "falsy". If x returns truthy, then the entire ternary operator returns y. If x is falsy, the the entire ternary operator returns z
    - == is equality (attempts to convert and then compare)
    - === is strict equality (only compares the same types)
- &&
  - One last way to write conditionals
  - Works best for conditionals that will sometimes do an action, but other times do nothing at all
  - Example: x && <h1>ya mama</h1>
    - If x then ya mama, otw nothing

.map in JSX

- .map() is the most efficient way of creating a list of JSX elements

```
const strings    'Home'  'Shop'  'About Me'

const listItems   strings map string    <li> string </li>

<ul> listItems </ul>

root.render(<ul>{listItems}</ul>);
```

- In the above example, start out with an array of strings. We call .map() on this array of strings, and the .map() call returns a new array of s
- On the last line of the of the example, note that {listItems} will evaluate to an array, because it's the returned value of .map()

Keys

- When making a list in JSX, sometimes list will need to include keys:

```
<ul>
  <li key "li-01">        </li>
  <li key "li-02">        </li>
  <li key "li-03">        </li>
</ul>
```

- A key is a JSX attribute. The attribute's name is key. The attributes value should be something unique, similar to an id attribute
- Keys don't do anything visible, React uses the internally to keep trac of lists. If keys are not used, React might scramble the order of list
- A list only needs keys if either of the following is true
    - The list items have memory from one render to the next. For instance, when a to-do list renders, each item must "remember" whether it was checked off
    - A list's order might be shuffled. For instance, a list of search results might be shuffled from one render to the next

```
const peopleList = people.map((person,i) =>
  // expression goes here:
  <li key = {'person_' + i} >{person}</li>
);
```

React.createElement

- Can write React code without using JSX at all
- For example, the JSX expression:
  const h1 = <h1>Hello World</h1>;
  Can be written as

```
const h1 = React createElement(
  "h1"
  null
  "Hello world"
 )
```

- When a JSX element is compied, the compiler transforms the JSX element into the method that you see above: React.createElement(). Every JSX element is secretly a call to React.createElement()

React Components

- React applications are made of components
- What is a component?
  - A small, reusable chunk of code that is responsible for one job. That job is often to render some HTML and re-render it when some data changes

Import React

- `import React from 'react'`
- Creates an object named React, which contains methods necessary to use the React library. React is imported from the ‘react’ package, which should be installed in your project as a dependency. With the object, can start utillizing features of the react library

Import ReactDOM

- `import ReactDOM from 'react-dom/client'`
  The methods imported from ‘react-dom’ interact with the DOM

Create a Function Component

- Components are building blocks that make up a React application.

Week 3

Create a Function Component

- Function Component: A React component defined by JavaScript functions
- Can use it to create as many instances of that component as we want

- Functions are declared with PascalCase (diff from camelCase
- Functional components are very similar to JavaScript functions, except their job is to assemble a portion of the interface based on instructions given

Importing and Exporting React Components
- React application typically has 2 core files
    - App.js - top level
    - index.js - entry point
- Export from App.js
- Import from index.js

Using and Rendering a Component

```
ReactDOM createRoot document getElementById 'app'    render <MyComponent
/>
```

- Goes in index.js

Use Multiline JSX in a Component
- In order to return multi line JSX components, use parenthesis

Use a Variable Attribute in a Component
- Given a variable with attributes, can use these attributes in a component by doing {variable.attr}

Putting Logic in a Function Component
- Can also do logic before any returns in a function component
- Can also do conditionals

Event Listener and Event Handlers in a Component
- Function components can include event handlers
- When calling the function within the JSX element, do not include parenthesis, bc that would call the function immediately (we only want to call it once the event listener is triggered)

Create a React App
- Moving onto big things >_>

Returning Another Component

-

Apply a Component in a Render Function

Review

- A React application can contain multiple components.
- Components can interact with each other by returning instances of each other.
- Components interacting allows them to be broken into smaller components, stored into separate files, and reused when necessary.

Props

- Information that gets passed from one component to another

Week 4

Access a Component's props

- Component's props are an object
  - Holds info abt that component
- To access a component's `props` object, you can reference the `props` object and the dot notation for its properties
  - props.name

Pass 'props to a Component

- To pass props, give the component an attribute

player.js

```
import React from 'react'


function Player props

  return

      <h1> props songName </h1>

      <h2> props artist </h2>


}
```

```
export default Player
```

app.js

```
import React from 'react'

import Player from './Player'

function App
  return
    <Player songName   "aaa" artist   "bbb"/>


export default App
```

- Can also use props in if statements

Putting an Event Handler in a Function Component

- Will often pass functions as props, especially common to pass event handler functions

```
mport React from 'react'
import Button from './Button'

function Talker
  function talk
    let speech   ''
    for  let i   0  i   10000  i
      speech    'blah '

    alert speech
```

```
  return <Button />


export default Talker
```

Pass an Event Handler as a Prop

- <Button beef = {talk} />
    - Note: no () after the function talk

Receive an Event Handler as a prop

- <button onClick = {props.talk}>
    - touch me
- </button>

Props.children

- Every component's props object has a property named children
- props.children will return everything btw a component's opening and closing JSX tags

Giving Default values to props

- Components should have a default value

That completes our lesson on props! Here are some of the skills that you've learned:

- Passing a prop by giving an *attribute* to a component instance.
- Accessing a passed-in prop via `props.propName`.
- Displaying a prop.
- Using a prop to make decisions about what to display.
- Defining an event handler in a function component.
- Passing an event handler as a prop.
- Receiving a prop event handler and attaching it to an event listener.
- Naming event handlers and event handler attributes according to a convention.
- Accessing `props.children`.
- Assigning default values to

    - Preview: Docs Loading link description
    - props

Why Use Hooks?

- Hooks are functions that let us manage the internal state of components and handle post-rendering side effects from our function components
- Using hooks, we can determine what we want to show the users by declaring how our user interface should look based on the state
    - Examples include: useState(), useEffect(), useContext(), useReducer, and useRef()

Update Function Component State

- State Hook
    - `import React, { useState } from 'react';`
    - when we call useState() function, it returns an array with two values:
        - The current state
        - The state setter: a function that we can use to update the value of this state
    - Can use these 2 values to track the current state of a data value or property and change it when we need to. Can extract values as so:
        - `const [currentState, setCurrentState] = useState();`
    - useState() allows React to keep track of the current value of the state from one render to the next

Initialize State

- Can also use State Hook to manage the value of any primitive data type and data collections like arrays and objects
    - `const [isLoading, setIsLoading] = useState(true);`
    - true is now the default value of the state variable isLoading

Use State Setter Outside of JSX

- This

```
const handleChange = (event) => {
  const newEmail = event.target.value;
  setEmail(newEmail);
}
```

- Can simplify to this:
    - `const handleChange = (event) => setEmail(event.target.value);`
- Or this
    - `const handleChange = ({target}) => setEmail(target.value);`

Set From Previous State

- React state updates are asynchronous
    - There are some scenarios where portions of code will run before state is finished updating
    - Good: grouping state updates can improve performance
    - Bad: outdated state values
- Pass a callback function as the argument for setState() instead of a value

Arrays in State

- One can also use arrays as state variables