React Study Notes

Introduction to React

1. Where does React put all of the elements I create in JSX when I call root.render()?

All the elements I render get put inside the div with the id of "root" (or whatever other element I might select when calling createRoot).

2. What would show up in my console if I were to run this line of code?

```
console.log(<h1>Hello world!</h1>)
```

An object! Unlike creating an HTML element in vanilla DOM JS, what gets created from the JSX we have in our React code is a plain JS object that React will use to fill in the view.

3. What's wrong with this code?

```
root.render(
    <section>
        <h1>Hi there</h1>
        <p>This is my website!</p>
    </section>
)
```

You can only render 1 parent element at a time. That parent element can have as many children elements as you want.

4. What does it mean for something to be "declarative" instead of "imperative"?

Imperative: We need to give specific step-by-step instructions on how to accomplish a task.
Declarative: We can write our code to simply "describe" what should show up on the page and allow the tool (React, e.g.) to handle the details on how to put those things on the page.

5. What does it mean for something to be "composable"?

We have small pieces that we can put together to make something larger or greater than the individual pieces themselves.

Props

1. What do props help us accomplish?

Make a component more reusable.

2. How do you pass a prop into a component?

```
<MyAwesomeHeader title="???" />
```

3. Can I pass a custom prop (e.g. blahblahblah={true}) to a native DOM element?

No, because the JSX we use to describe native DOM elements will be turned into REAL DOM elements by React. And real DOM elements only have the properties/attributes specified in the HTML specification. (Which doesn't include properties like blahblahblah)

4. How do I receive props in a component?

```
function Navbar(props) {
    console.log(props.blahblahblah)
    return (
        <header>
            ...
        </header>
    )
}
```

5. What data type is props when the component receives it?

An object!

Array.map() Method

Example 1: Squaring Numbers

```
const nums = [1, 2, 3, 4, 5]
const squares = nums.map(function(num) {
    return num * num
})
// Result: [1, 4, 9, 16, 25]
```

Example 2: Capitalizing Strings

```
const names = ["alice", "bob", "charlie", "danielle"]
const capitalized = names.map((name) => {
    return name[0].toUpperCase() + name.slice(1)
})
// Result: ["Alice", "Bob", "Charlie", "Danielle"]
```

Example 3: Wrapping in HTML Tags

```
const pokemon = ["Bulbasaur", "Charmander", "Squirtle"]
const paragraphs = pokemon.map(mon => `<p>${mon}</p>`)
// Result: ["<p>Bulbasaur</p>", "<p>Charmander</p>", "<p>Squirtle</p>"]
```

Map Quiz

1. What does the .map() array method do?

Returns a new array. Whatever gets returned from the callback function provided is placed at the same index in the new array. Usually we take the items from the original array and modify them in some way.

2. What do we usually use .map() for in React?

Convert an array of raw data into an array of JSX elements that can be displayed on the page.

3. Why is using .map() better than manually creating components?

- We often don't have the data ahead of time when we're building the app, so we simply can't manually type them out.
- It makes our code more "self-sustaining" - not requiring additional changes to the code whenever the data changes.

useState Hook

Counter Example

```
import React from "react"

export default function App() {
    const [count, setCount] = React.useState(0);

    function add() {
        setCount(prevCount => prevCount + 1);
    }

    function subtract() {
        setCount(prevCount => prevCount - 1);
    }

    return (
        <main className="container">
            <h1>How many times will Bob say "state" in this section?</h1>
            <div className="counter">
                <button className="minus" onClick={subtract}>–</button>
                <h2 className="count">{count}</h2>
                <button className="plus" onClick={add}>+</button>
            </div>
        </main>
    )
}
```

useState Quiz

> 1. What are the 2 options for what you can pass to a state setter function?

> 1. Pass the new version of state that we want to use as the replacement for the old version of state.
> 2. Pass a callback function. Must return what we want the new value of state to be. Will receive the old version of state as a parameter so we can use it to help determine what we want the new value of state to be.

> 2. When would you pass the first option?

> Whenever we don't really care about (or need) the old value, we simply want to set a new value.

> 3. When would you pass the second option?

> Whenever we do care about the previous value in state and need it to help us determine what the new value should be.

Updating State Objects

Key Concept: When updating objects in state, use the spread operator (...) to create a new object while preserving unchanged properties.

```
const [contact, setContact] = React.useState({
    firstName: "John",
    lastName: "Doe",
    phone: "+1 (212) 555-1212",
    email: "itsmyrealname@example.com",
    isFavorite: false
})

function toggleFavorite() {
    setContact(prevContact => ({
        ...prevContact,
        isFavorite: !prevContact.isFavorite
    }))
}
```

Conditional Rendering

1. What is "conditional rendering"?

When we want to only sometimes display something on the page based on some kind of condition.

2. When would you use &&?

When you want to either display something or NOT display something.

3. When would you use a ternary?

When you need to decide which of 2 things to display.

4. What if you need to decide between more than 2 options?

Use if...else if...else conditional or maybe a switch statement.

useEffect Hook

useEffect Quiz

1. In what way are React components meant to be "pure functions"?

- Given the same props or state, the component will always return the same content, or UI
- Rendering and re-rendering a component will never have any kind of side effect on an outside system

2. What is a "side effect" in React? Examples?

Side Effect: Any code that affects or interacts with an outside system
Examples: local storage, API calls, websockets, DOM manipulation

3. What is NOT a "side effect" in React?

Anything that React is in charge of: maintaining state, keeping the UI in sync with data, rendering DOM elements

4. When does React run your useEffect function?

- As soon as the component renders for the first time
- On every re-render of the component (assuming no dependencies array)
- Will NOT run the effect when the values of the dependencies in the array stay the same between renders

5. What is the "dependencies array"?

- Second parameter to the useEffect function
- A way for React to know whether or not it should re-run the effect function

useEffect Examples

Fetching API Data

```
export default function App() {
    const [starWarsData, setStarWarsData] = React.useState({})
    const [count, setCount] = React.useState(1)

    React.useEffect(() => {
        fetch(`https://swapi.dev/api/people/${count}`)
            .then(res => res.json())
            .then(data => setStarWarsData(data))
    }, [count])

    return (
        <div>
            <h2>The count is {count}</h2>
            <button onClick={() => setCount(prevCount => prevCount + 1)}>
                Get next character
            </button>
            <pre>{JSON.stringify(starWarsData, null, 2)}</pre>
        </div>
    )
}
```

Window Resize Listener

Note: This practice needs cleanup function to remove event listener!

```
export default function WindowTracker() {
    const [windowWidth, setWindowWidth] = React.useState(window.innerWidth)

    React.useEffect(() => {
        window.addEventListener("resize", function() {
            setWindowWidth(window.innerWidth)
        })
    }, [])

    return (
        <h1>Window width: {windowWidth}</h1>
    )
}
```

Key Takeaways

- React uses a **declarative** approach to building UIs
- Components should be **pure functions** for predictable rendering
- Use **props** to make components reusable
- Use **.map()** to transform data into JSX elements
- Use **useState** for managing component state
- Use **useEffect** for side effects like API calls
- Always include **dependency arrays** in useEffect

Print / Save as PDF