codecademy

Advanced React

React CSS Styles

React supports inline CSS styles for elements. Styles are supplied as a `style` prop with a JavaScript object.

```
// Passing the styles as an object
const color = {
  color: 'blue',
  background: 'sky'
};
<h1 style={color}>Hello</h1>

// Passing the styles with an inline object,
as a shorthand
<h1 style={{ color: 'red' }}>I am red!</h1>
```

Style Names And Values

In React, style names are written in “camelCase”, unlike in CSS where they are hyphenated. In most cases, style values are written as strings. When entering numeric values, you don’t have to enter `px` because React automatically interprets them as pixel values.

```
// Styles in CSS:
// font-size: 20px;
// color: blue;

// Would look like this style object in
React:
const style = {
  fontSize: 20,
  color: 'blue',
};
```

Presentational and Container Components

A common programming pattern in React is to have presentational and container components. Container components contain business logic (methods) and handle state. Presentational components render that behavior and state to the user.
In the example code, `CounterContainer` is a container component and `Counter` is a presentational component.

```
class CounterContainer extends
React.Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
    this.increment
= this.increment.bind(this);
  }

  increment() {
    this.setState((oldState) => {
      return { count: oldState.count + 1 };
    });
  }

  render() {
    return <Counter count={this.state.count}
increment={this.increment} />;
  }
}

class Counter extends React.Component {
  render() {
    return (
      <div>
        <p>The count is {this.props.count}.
</p>
        <button onClick=
{this.props.increment}>Add 1</button>
      </div>
    );
  }
}
```

Static Property

In React, prop types are set as a static property (`.propTypes`) on a component class or a function component. `.propTypes` is an object with property names matching the expected props and values matching the expected value of that prop type. The code snippet above demonstrates how `.propTypes` can be applied.

```
class Birth extends React.Component {
  render() {
    return <h1>{this.props.age}</h1>
  }
}

Birth.propTypes = {
  age: PropTypes.number
}
```

`.isRequired`

To indicate that a prop is required by the component, the property `.isRequired` can be chained to prop types. Doing this will display a warning in the console if the prop is not passed. The code snippet above demonstrates the use of `.isRequired` .

```
MyComponent.propTypes = {
  year: PropTypes.number.isRequired
};
```

Type Checking

In React, the `.propTypes` property can be used to perform type-checking on props. This gives developers the ability to set rules on what data/variable type each component prop should be and to display warnings when a component receives invalid type props.
In order to use `.propTypes` , you'll first need to import the `prop-types` library.

```
import PropTypes from 'prop-types';
```

Controlled vs. Uncontrolled Form Fields

In React, form fields are considered either *uncontrolled*, meaning they maintain their own state, or *controlled*, meaning that some parent maintains their state and passes it to them to display. Usually, the form fields will be controlled.
The example code shows an uncontrolled and controlled input.

```
const uncontrolledInput = <input />;

const controlledInput = (
  <input value={this.state.value} onChange=
{this.handleInputChange} />
);
```

Controlled Components

A controlled form element in React is built with a change handler function and a `value` attribute.

```
const controlledInput = (
  <input value={this.state.value} onChange=
{this.handleInputChange} />
);
```