React JS

Components, Props, State, Events

React Components

- Components are like functions that return HTML elements
- Components are independent and reusable bits of code
- They serve the same purpose as JavaScript functions, but returns HTML via a render function
- Components come in two types
  - Class components
  - Function components

Class Component

- When creating a React component, the component's name MUST start with an uppercase letter
  - <section> is lowercase, so React knows we refer to an HTML tag.
  - <Profile /> starts with a capital P, so React knows that we want to use our component called Profile.
- The component must include the `extends React.Component` statement
  - This statement creates an inheritance to `React.Component`, and gives your component access to React.Component's functions
- The component also requires a `render()` method, this method returns HTML

Class Component (Example)

- Create a Class component called `Car`

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```

- To use this component in your application, use similar syntax as normal HTML: `<Car />`

```
ReactDOM.render(<Car />, document.getElementById('root'));
```

Function Component

- A Function component also returns HTML, and behaves pretty much the same way as a Class component
- React components are regular JavaScript functions except:
  - Their names always begin with a capital letter.
  - They return JSX markup.

```
function Car() {
  return <h2>Hi, I am also a Car!</h2>;
}
```

Component Constructor

- If there is a `constructor()` function in your component, this function will be called when the component gets initiated.
- The constructor function is where you initiate the component's properties.
- In React, component properties should be kept in an object called `state`.
- The `constructor` function is also where you honor the inheritance of the parent component by including the `super()` statement
  - which executes the parent component's constructor function, and your component has access to all the functions of the parent component (`React.Component`).

Car Component - Example

- Create a constructor function in the Car component, and add a color property

```
class Car extends React.Component {
  constructor() {
    super();
    this.state = {color: "red"};
  }
  render() {
    return <h2>I am a Car!</h2>;
  }
}
```

- Use the color property in the render() function

```
class Car extends React.Component {
  constructor() {
    super();
    this.state = {color: "red"};
  }
  render() {
    return <h2>I am a {this.state.color} Car!</h2>;
  }
}
```

Components in Components

- We can refer to components inside other components
- Use the Car component inside the Garage component

```
class Car extends React.Component {
  render() {
    return <h2>I am a Car!</h2>;
  }
}

class Garage extends React.Component {
  render() {
    return (
      <div>
      <h1>Who lives in my Garage?</h1>
      <Car />
      </div>
    );
  }
}

ReactDOM.render(<Garage />, document.getElementById('root'));
```

Components in Files

- React is all about re-using code, and it can be smart to insert some of your components in separate files
- To do that, create a new file with a `.js` file extension and put the code inside it:
- Note that the file must start by `importing React` and it has to end with the statement `export default Car`;

Components in Files - Example

- This is the new file, we named it “CarComponent.js":

```
import React from 'react';
import ReactDOM from 'react-dom';

class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}

export default Car;
```

- To be able to use the Car component, you have to import the file in your application

```
import Car from './CarComponent.js';
```

React Props

- Props (short for properties) are used to pass data from one component to another, much like arguments are passed to functions in JavaScript.
- They allow components to be dynamic and reusable by providing them with values or data from outside.
- Props are passed to components as attributes in JSX (similar to HTML attributes), and the receiving component can access those props through the props object.
- Props in React are like function arguments in JavaScript and attributes in HTML.
- Use `brand` attribute in component through `props`

Example

Add a "brand" attribute to the Car element:

```
const myelement = <Car brand="Ford" />;
```

Example

Use the brand attribute in the component:

```
class Car extends React.Component {
  render() {
    return <h2>I am a {this.props.brand}!</h1>;
  }
}
```

Props - Pass Data

- Props are also how you pass data from one component to another, as parameters

Example

Send the "brand" property from the Garage component to the Car component:

```
class Car extends React.Component {
  render() {
    return <h2>I am a {this.props.brand}!</h2>;
  }
}

class Garage extends React.Component {
  render() {
    return (
      <div>
      <h1>Who lives in my garage?</h1>
      <Car brand="Ford" />
      </div>
    );
  }
}

ReactDOM.render(<Garage />, document.getElementById('root'));
```

Props

Send Variable instead of String

Example

Create a variable named "carname" and send it to the Car component:

```
class Car extends React.Component {
  render() {
    return <h2>I am a {this.props.brand}!</h2>;
  }
}

class Garage extends React.Component {
  render() {
    const carname = "Ford";
    return (
      <div>
      <h1>Who lives in my garage?</h1>
      <Car brand={carname} />
      </div>
    );
  }
}

ReactDOM.render(<Garage />, document.getElementById('root'));
```

Props

Send Object instead of Variable or String

Example

Create an object named "carinfo" and send it to the Car component:

```
class Car extends React.Component {
  render() {
    return <h2>I am a {this.props.brand.model}!</h2>;
  }
}

class Garage extends React.Component {
  render() {
    const carinfo = {name: "Ford", model: "Mustang"};
    return (
      <div>
      <h1>Who lives in my garage?</h1>
      <Car brand={carinfo} />
      </div>
    );
  }
}

ReactDOM.render(<Garage />, document.getElementById('root'));
```

Props in the Constructor

- If your component has a constructor function, the props should always be passed to the `constructor` and to the `React.Component` via the `super()` method

```
class Car extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return <h2>I am a Car!</h2>;
  }
}

ReactDOM.render(<Car model="Mustang"/>, document.getElementById('root'));
```

Note: *React Props are read-only! You will get an error if you try to change their value*

React State

- React components has a built-in `state` object
- The `state` object is where you store property values that belongs to the component
- When the `state` object changes, the component re-renders
- The `state` object is initialized in the constructor

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {brand: "Ford"};
  }
  render() {
    return (
      <div>
        <h1>My Car</h1>
      </div>
    );
  }
}
```

React State

- The `state` object can contain as many properties as you like

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
  render() {
    return (
      <div>
        <h1>My Car</h1>
      </div>
    );
  }
}
```

Using the `state` Object

- Refer to the `state` object anywhere in the component by using the `this.state.propertyname` syntax

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
  render() {
    return (
      <div>
        <h1>My {this.state.brand}</h1>
        <p>
          It is a {this.state.color}
          {this.state.model}
          from {this.state.year}.
        </p>
      </div>
    );
  }
}
```

Changing the state Object

- To change a value in the state object, use the `this.setState()` method
- When a value in the `state` object changes, the component will re-render, meaning that the output will change according to the new value(s).
- Always use the `setState()` method to change the `state` object, it will ensure that the component knows its been updated and calls the `render()` method (and all the other lifecycle methods)

```
class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      brand: "Ford",
      model: "Mustang",
      color: "red",
      year: 1964
    };
  }
  changeColor = () => {
    this.setState({color: "blue"});
  }
  render() {
    return (
      <div>
        <h1>My {this.state.brand}</h1>
        <p>
          It is a {this.state.color}
          {this.state.model}
          from {this.state.year}.
        </p>
        <button
          type="button"
          onClick={this.changeColor}
        >Change color</button>
      </div>
    );
  }
}
```

Prop vs State

Props	State
Passed from parent to child	Managed within the component
Read-only (immutable)	Can be changed (mutable)
Controlled by parent component	Controlled by the component
Used for static or external data	Used for dynamic or internal data

React Events

- Just like HTML, React can perform actions based on user events.
- React has the same events as HTML: click, change, mouseover etc

Adding Events

- React events are written in **camelCase** syntax:

  `onClick` instead of `onclick`

- React event handlers are written inside **curly braces**:

  `onClick={shoot}` instead of `onClick="shoot()"`

React:

```
<button onClick={shoot}>Take the Shot!</button>
```

HTML:

```
<button onclick="shoot()">Take the Shot!</button>
```

Event Handlers

- A good practice is to put the event handler as a method in the component class

Example:

Put the `shoot` function inside the `Football` component:

```
class Football extends React.Component {
  shoot() {
    alert("Great Shot!");
  }
  render() {
    return (
      <button onClick={this.shoot}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

Bind `this`

- For methods in React, the `this` keyword should represent the component that owns the method.
- That is why you should use arrow functions. With arrow functions, `this` will always represent the object that defined the arrow function.

```
class Football extends React.Component {
  shoot = () => {
    alert(this);
    /*
    The 'this' keyword refers to the component object
    */
  }
  render() {
    return (
      <button onClick={this.shoot}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

In class components, the `this` keyword is not defined by default, so with regular functions the `this` keyword represents the object that called the method, which can be the global window object, a HTML button, or whatever

Bind `this`

If you must use regular functions instead of arrow functions you have to bind `this` to the component instance using the `bind()` method

Without the binding, the `this` keyword would return `undefined`

Make `this` available in the `shoot` function by binding it in the `constructor` function:

```
class Football extends React.Component {
  constructor(props) {
    super(props)
    this.shoot = this.shoot.bind(this)
  }
  shoot() {
    alert(this);
    /*
    Thanks to the binding in the constructor function,
    the 'this' keyword now refers to the component object
    */
  }
  render() {
    return (
      <button onClick={this.shoot}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

Passing Arguments

- If you want to send parameters into an event handler, you have two options:
  1. Make an anonymous arrow function:

Example:

Send "Goal" as a parameter to the `shoot` function, using arrow function:

```
class Football extends React.Component {
  shoot = (a) => {
    alert(a);
  }
  render() {
    return (
      <button onClick={() => this.shoot("Goal")}>Take the shot!</bu
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

Passing Arguments

2. Bind the event handler to `this`

Note that the first argument has to be `this`

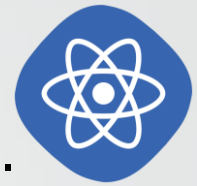
Note: If you send arguments without using the `bind` method, (`this.shoot(this, "Goal")` instead of `this.shoot.bind(this, "Goal")`), the `shoot` function will be executed when the page is loaded instead of waiting for the button to be clicked

Send "Goal" as a parameter to the `shoot` function:

```
class Football extends React.Component {
  shoot(a) {
    alert(a);
  }
  render() {
    return (
      <button onClick={this.shoot.bind(this, "Goal")}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

React Event Object



- Event handlers have access to the React event that triggered the function.
- In our example the event is the "click" event. Notice that once again the syntax is different when using arrow functions or not.
- With the arrow function you have to send the event argument manually

Arrow Function: Sending the event object manually:

```
class Football extends React.Component {
  shoot = (a, b) => {
    alert(b.type);
    /*
    'b' represents the React event that triggered the function,
    in this case the 'click' event
    */
  }
  render() {
    return (
      <button onClick={(ev) => this.shoot("Goal", ev)}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

React Event Object

- Without arrow function, the React event object is sent automatically as the last argument when using the `bind()` method:

With the `bind()` method, the event object is sent as the last argument:

```
class Football extends React.Component {
  shoot = (a, b) => {
    alert(b.type);
    /*
    'b' represents the React event that triggered the function,
    in this case the 'click' event
    */
  }
  render() {
    return (
      <button onClick={this.shoot.bind(this, "Goal")}>Take the shot!</button>
    );
  }
}

ReactDOM.render(<Football />, document.getElementById('root'));
```

References

- https://www.w3schools.com/react/default.asp
- https://reactjs.org/docs/hello-world.html