React Session 2

Mohamed Emary

July 28, 2024

1 JS Review

1.1 Array Destructuring

In JS you can put any type of data inside a function, even an array

```
function sayHello() {
  console.log('Hello');
}

let [x, y] = [5, sayHello];
y(); // Hello
```

1.2 Ternary Operator

When using ternaly operator, you can only use one statement, if you want more you need to use a function:

Suppose we want to check if `x > y`, print `x`, assign the sum of `x` and `y` to `res` else print `y`, assign the difference of `x` and `y` to `res`

```
let x = 5,
  y = 10,
  res;

// ERROR You can only have one statement in a ternary operator
x > y ? console.log(x); res = x + y : console.log(y); res = x - y;
```

To fix this, you need to use a function:

```
let x = 5,
  y = 10,
  res;

x > y
  ? (() => {
```

```
      console.log(x);
      res = x + y;
    })()
  : (() => {
      console.log(y);
      res = x - y;
    })();

console.log(res); // -5
```

You can also use a named function, and call it by in the ternary operator

---

If we have an `if else if else` statement, we can make its equivalent using ternary operators by nesting them

```
let x = 5,
  y = 10,
  res;

x > y ? console.log(x) : x < y ? console.log(y) : console.log('Equal');
```

1.3 `filter`

Suppose we have this array `[23, 45, 89, 12, 34, 11]`, and we want to filter out all the numbers that are less than 20:

```
let nums = [23, 45, 89, 12, 34, 11];

let filtered = nums.filter((num) => num >= 20);

console.log(filtered); // [23, 45, 89, 34]
```

`filter` doesn't change the original array, it returns a new array.

1.4 `map`

Suppose we have this array `[1, 2, 3, 4, 5]`, and we want to double each number:

```
let nums = [1, 2, 3, 4, 5];

let doubled = nums.map((num) => num * 2);

let nums_index = nums.map((num, index) => [num * 2 , index]);

console.log(doubled); // [2, 4, 6, 8, 10]
console.log(nums_index); // [[2, 0], [4, 1], [6, 2], [8, 3], [10, 4]]
```

`map` doesn't change the original array, it returns a new array.

2 React Session 1 Review

2.1 Creating a React App

When creating a React app, you can use `npm create vite@latest` to create a new React app using Vite. You don't have to use `@latest` though, it's the default anyway.

Then we install the dependencies using `npm install`, then we run the app using `npm run dev`.

2.2 Project Files

There are two `JSX` files in the project, `App.jsx` and `main.jsx`.

- `App.jsx` is the main component of the app.
- `App.css` is the CSS file for the `App.jsx` component ONLY. (Can't be used in other components)
- `main.jsx` is the entry point of the app. It links the main component which is `App.jsx` to the `index.html` file.
- `index.css` is the main CSS file that you can use to apply styles to ANY COMPONENT in the app. It's the main CSS file because it's the file that is linked to the `index.html` file.
    - `index.css` is linked to `main.jsx` using `import './index.css';`. It's similar to how we import modules in JS, but in modules we specify the name of the module with the path, while in CSS we only specify the path.

2.3 Function Component

For a function component to be a valid React component, it must return a JSX element, and the name of the component should start with a capital letter.

To create a react function using the react snippet, you can type `rfc` then press `tab`.

Function component name has to be uppercase, if it's lowercase, React will treat it as a custom HTML tag.

To make a comment in JSX, you can use `{/* Comment */}`.

2.4 Functional and Class Components

React supported class components untill version 16.8, but now it's deprecated. Functional components are much easier since in class components you have to bind `this` to the function.

3 Note on `<React.StrictMode>`

If you log something in the console in any component and call that component in in `App.jsx` inside which is called inside `main.jsx`, it will be logged twice because `main.jsx` has `<React.StrictMode>` which runs the component twice.

4 Binding in JSX

Suppose we have this component:

```
export default function Hello() {
  let username = "Mohamed";
  return (
    <div>Hello username</div>
  )
}
```

If we want to use the variable `username`, so it prints the value of the variable, back in JS we would select the element and use something like `.innerHTML` or `.innerText`, but in JSX we use `{}`:

```
export default function Hello() {
  let username = "Mohamed";
  return (
    <div>Hello {username}</div>
  )
}
```

This is called **binding**. Binding is used to add JS code inside JSX **in only one line (same idea as in ternary operator)**.

Another example:

Suppose you have these two classes in `index.css`:

```
.bg-red{
  background-color: red;
}

.bg-blue{
  background-color: blue;
}
```

And you want to add a class to a div based on a condition:

```
export default function Hello() {
  let isRed = true;
  return (
    <div className={isRed ? 'bg-red' : 'bg-blue'}>Hello</div>
  )
}
```

From that we conclude that binding can also be used to values to attributes.

We can eve use backticks to use template literals `className={`bg-red pad-${isRed ? 'small' : 'big'}`}`. This will add the class `bg-red` and `pad-small` if `isRed` is true, otherwise it will add `bg-red` and `pad-big`.

5 Events in JSX vs HTML

In HTML, we use `onclick`, `onmouseover`, `onmouseout`, etc. In JSX, we use `onClick`, `onMouseOver`, `onMouseOut`, etc.

In JSX, we use camelCase so instead of `onclick`, we use `onClick`.

In JSX we pass the name of the function without the parentheses, while in HTML we pass the name of the function with the parentheses:

- In HTML: `<button onclick="sayHello()">Click me</button>`
- In JSX: `<button onClick={sayHello}>Click me</button>`

And if the function needs a parameter, we use an arrow function:

```
<button onClick={() => sayHello('Mohamed')}>Click me</button>
```

Consider this example component:

```
export default function Incrementer() {
  let counter = 100;

  function increment() {
    counter++;
    console.log(counter);
  }

  return (
    <div>
      <h1>{counter}</h1>
      <button onClick={increment}>Increment</button>
    </div>
  );
}
```

When you try this code, you will see that the updated counter is logged to the console, but the value of the counter on the screen is not updated. This is because React doesn't know that the **state** has changed, so it doesn't re-render the component.

That was the reason why people in the past used class components, because they had a method called `setState` that would update the state and re-render the component.

But as of React 16.8, we have hooks, and the `useState` hook is used to create a state variable and a function to update it.

Back when we were studying CRUD operations with JS, we had to use a function that displays the data, and with each change in the data we had to call that function again to display the updated data.

6 State & Hooks

6.1 State

A state is a container that holds data that can be changed. When the state changes, the component gets re-rendered.

6.1.1 State vs Variable

A variable is a container that holds data, but when the variable changes, the component doesn't get re-rendered, on the other hand, when the state changes, the component gets re-rendered.

After rendering, the variable will return to its initial value, while the state will keep the updated value.

	Variable	State
Re-rendering on change:	No	Yes
Value after re-render:	Initial	Updated

And that was the difference between class and functional components, class components had a state, while functional components were stateless.

6.2 Hooks

Hooks were introduced in React 16.8, and **they are functions** that let you use state and other React features without writing a class.

A Hook:

- Is a function.
- Starts with `use**`, like `useState()`.
- Can't be used outside a functional component.

6.2.1 `useState`

`useState` is a Hook that lets you add state to a functional component.

`useState` returns an array with two elements:

1. A state value.
2. A function that lets you update the state value, and re-renders the component when the state changes.

If you change that value without using the function, the component won't re-render.

Now let's update the `Incrementer` component using `useState`:

```
1  import { useState } from 'react';
2
3  export default function Incrementer() {
4    const [counter, setCounter] = useState(100);
5
6    function increment() {
7      setCounter(counter + 1);
8    }
9
10   return (
11     <div>
12       <h1>{counter}</h1>
```

```
      <button onClick={increment}>Increment</button>
    </div>
  );
}
```

When you click the button, the value of the counter will be updated and the component will be re-rendered.

You can use any number of `useState` hooks in a component, for example to set the username:

```
import { useState } from 'react';

export default function Incrementer() {
  const [counter, setCounter] = useState(100);

  function increment() {
    setCounter(counter + 1);
  }
  const [username, setUsername] = useState("");

  function setUsernameHandler() {
    setUsername("Mohamed");
  }

  return (
    <div>
      <h1>{counter}</h1>
      <button onClick={increment}>Increment</button>
      <h1>{username}</h1>
      <button onClick={setUsernameHandler}>Set Username</button>
    </div>
  );
}
```

`useStateSnippet` can be used to create a new state variable and its setter.

You can pass any value to `useState`, it doesn't have to be a number, it can be a string, an object, an array, etc.

7 Child Components

A child component is a component that is nested inside another component.

For example if we have a `Parent` component, and we want to add a `Child` component inside it:

```
// Your imports here

export default function Parent() {
  return (
    <div>
      <h1>Parent</h1>
      <Child />
    </div>
```

```
  );
}
```

And the `Child` component:

```
export default function Child() {
  return <h1>Child</h1>;
}
```

Then in `App.jsx` we can call the `Parent` component:

```
export default function App() {
  return (
    <div>
      <Parent />
    </div>
  );
}
```

You can even call the `<Child />` multiple times inside the same parent component.

7.1 Why to use components?

Components make your code resuable. Since components are actually functions, and you can call them multiple times, you can use the same component in multiple places.

For example, if you have a website that has a navbar in every page, you can create a `Navbar` component and use it in every page.

7.2 Sending Data Between Components

To be able to send data from component to another, the component that sends the data must be the **parent** component of the component that receives the data.

To send data from the **parent** to the **child**, you can use **props (properties)**.

Props are passed from the parent to the child component as attributes, and you can send any number of prop***s***.

The child component receives the props in the first argument of its function component as an object and can access them using dot notation.

For example, if we have a `Parent` component that sends `name` and `age` to the `Child` component:

```
import Child from "../Child/Child";

export default function Parent() {
  let username = "Mohamed";

  return (
    <div>
      <h1>Parent</h1>
      <Child name={username} age={20} />
    </div>
  );
}
```

Notice that to send 20 as a number, you have to use curly braces {}.

Receives the props in the first argument

```
export default function Child(props) {
  return (
    <div>
      <h1>{props.name}</h1>
      <h1>{props.age}</h1>
    </div>
  );
}
```

You can also use destructuring to get the props, so instead of `Child(props)`, you can use `Child({name, age})`. And we use `{}` not `[]` to destruct because props is an object not an array.

Suppose you have an object `product` that contains the name, price, and brand of a product, and you want to send this object to a `Product` component:

```
import Product from "../Product/Product";

export default function Parent() {
  let product = {
    name: "iPhone",
    price: 1000,
    brand: "Apple",
  };

  return (
    <div>
      <h1>Parent</h1>
      <Product product={product} />
    </div>
  );
}
```

And the `Product` component:

```
export default function Product({ product }) {
  return (
    <div>
      <h1>{product.name}</h1>
      <h1>{product.price}</h1>
      <h1>{product.brand}</h1>
    </div>
  );
}
```

What if the product is an array of objects?

The may think we should use a for loop to loop over the array and send each object to the `Product` component, but that's not possible because the for loop is not a one-liner.

So we are limited to one-liners, and we can use the `map` function to loop over the array and send

each object to the `Product` component.

```
import Product from "../Product/Product";

export default function Parent() {
  let products = [
    {
      name: "iPhone",
      price: 1000,
      brand: "Apple",
    },
    {
      name: "Galaxy",
      price: 900,
      brand: "Samsung",
    },
  ];

  return (
    <div>
      <h1>Parent</h1>
      {products.map((product) => (<Product product={product} />))}
    </div>
  );
}
```

But how were you able to use `<Product product={product} />` inside the binding `{}` which should only contain JS code?

In JSX, when you use curly braces `{}`, you're embedding JavaScript expressions within the JSX.

JSX elements are expressions, so `<Product product={product} />` is a JSX element, but in JSX, it is also a JavaScript expression that evaluates to a React element. The curly braces allow you to use any JavaScript expression, including function calls, object properties, and even JSX elements.

8 Side note (difference between `map` and `forEach`)

The main difference between `map` and `forEach` is that `map` **returns a new array** with the new modifications, while `forEach` **doesn't return anything**, it just iterates over the array do do some operations and **don't change the original array**.

```
let x = [1, 2, 3, 4, 5];
let y = x.map((x) => x * 2);
let z = x.forEach((x) => x * 2);

console.log(x); // [1, 2, 3, 4, 5]
console.log(y); // [2, 4, 6, 8, 10]
console.log(z); // undefined
```

That is the reason why we use `map` to loop over an array and send each object to a component, because `forEach` doesn't return anything.

9 Installing UI Libraries

9.1 Tailwind CSS

Tailwind provides multiple ways to install it, depending on the project you're working on. We are using React with Vite, so we will use the `npm` method.

To install Tailwind CSS, run:

```
npm install -D tailwindcss postcss autoprefixer
```

The `-D` flag is used to install the packages as dev dependencies.

Then you need to create a `tailwind.config.js` file by running:

```
npx tailwindcss init -p
```

The `-p` flag is used to create a `postcss.config.js` file.

Now in the `content` of the `tailwind.config.js` file, you need to add the files that you want Tailwind to scan for classes:

`tailwind.config.js`:

```
module.exports = {
  content: ["./index.html", "./src/**/*.{js,jsx,ts,tsx}"],
  theme: {
    extend: {},
  },
  plugins: [],
};
```

The `**` is used to scan all the files in any subdirectory level in the `src` directory, and the `*.{js,jsx,ts,tsx}` is used to scan only the files with these extensions.

9.2 Font Awesome & Bootstrap

To install Font Awesome & Bootstrap, you need to run:

```
npm install @fortawesome/fontawesome-free bootstrap
```

You should see them now in the `dependencies` in the `package.json` file.

Now you should decide if you want to use those libraries for all components or for a specific component.

If you want to use them for all components, you should import them in the `main.jsx` file.

If you want to use them for a specific component, you should import them in that component.

Example using both libraries in the `main.jsx` file:

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import "@fortawesome/fontawesome-free/css/all.min.css";
import "bootstrap/dist/css/bootstrap.min.css                ";
import "bootstrap/dist/js/bootstrap.bundle.min.js ";
```

```

import "./index.css";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Notice that you should always import the `index.css` file after the libraries, because the libraries might have some styles that you want to override.

10 Child Example Continued

If you had a property called `onSale` that has a boolean value, and you want to display a message if the product is on sale, you can use a ternary operator:

```
export default function Product({ product }) {
  return (
    <div>
      <h1>{product.name}</h1>
      <h1>{product.price}</h1>
      <h1>{product.brand}</h1>
      <h1>{product.onSale ? "On Sale" : ""}</h1>
    </div>
  );
}
```

Or you can even make a card on the top right corner of the product card that says “Sale!” if the product is on sale using Tailwind CSS:

```
export default function Product({ product }) {
  return (
    <div className="border p-4 m-4">
      <h1>{product.name}</h1>
      <h1>{product.price}</h1>
      <h1>{product.brand}</h1>
      {product.onSale ? (
        <div className="absolute top-0 right-0 bg-red-500 text-white p-2">
          Sale!
        </div>
      ) : (
        ""
      )}
    </div>
  );
}
```