Class 5: Assignment

- What is the difference between **Named** export, **Default** export and *** as** export?
  - In **named export**, we can have multiple exports per file. We can just use the keyword export before the component or function.

```
export const Function1 = () => {
}

export const Function2 = () => {
}
```

    Different ways to import it:

```
import {Function1} from "./MyFile";
import {Function1, Function2} from "./MyFile";
import {Function2 as NewFunction} from "./MyFile";
```

  - In **Default export**, we can have e only one default export per file.

```
const MyComponent = () => {
}

export default MyComponent;
```

    To import it:

```
import MyComponent from "./MyComponent";
```

  - In * as export, we can use it to import the whole module as a component and access the components from inside the module.

```
export const MyComponent = () => {}
export const MyComponent2 = () => {}
export const MyComponent3 = () => {}
```

To import it:

```
import * as MainComponents from "./MyComponent";
```

To use them:

```
<MainComponents.MyComponent />
<MainComponents.MyComponent2 />
<MainComponents.MyComponent3 />
```

▼ What is the importance of the config.js file?

- They help to keep code organized and maintainable and can be used to store values that are used throughout the codebase.
- Defining constants in a separate file also allows for better scalability. If changes need to be made to the constants, they can be done quickly and easily without having to search through the entire codebase.
- Additionally, it's best practice to store constants in an object or array so that they can be accessed more easily.
- Named exports allow for more flexibility when importing the constants. Instead of having to import all of the constants at once, you can choose which ones you want to use in a particular file. This makes it easier to keep track of what is being used and where.

▼ What are React hooks?

React Hooks are simple JavaScript functions that we can use to isolate the reusable part from a functional component. Hooks can be stateful and can manage side effects.

React provides a bunch of standard in-built hooks:

- `useState`: To manage states. Returns a stateful value and an updater function to update it.
- `useEffect`: To manage side-effects like API calls, subscriptions, timers, mutations, and more.
- `useContext`: To return the current value for a context.
- `useReducer`: A `useState` alternative to help with complex state management.
- `useCallback`: It returns a memorized version of a callback to help a child component not re-render unnecessarily.
- `useMemo`: It returns a memoized value that helps in performance optimizations.
- `useRef`: It returns a ref object with a `.current` property. The ref object is mutable. It is mainly used to access a child component imperatively.
- `useLayoutEffect`: It fires at the end of all DOM mutations. It's best to use `useEffect` as much as possible over this one as the `useLayoutEffect` fires synchronously.
- `useDebugValue`: Helps to display a label in React DevTools for custom hooks.

▼ Why do we need **useState** hook?

- `useState`: To manage states.
- It is a React Hook that lets you add a state variable to your component.
- Returns a stateful value and an updater function to update it.

▼ What is a state?

Components often need to change what’s on the screen as a result of an interaction.

Typing into the form should update the input field, clicking “next” on an image carousel should change which image is displayed, and clicking “buy” should put a product in the shopping cart.

> Components need to “remember” things: the current input value, the current image, and the shopping cart.
>
> In React, this kind of component-specific memory is called a ***state***.

The `useState` Hook provides those two things:

1. A **state variable** to retain the data between renders.
2. A **state setter function** to update the variable and trigger React to render the component again.

To add a state variable, import `useState` from React at the top of the file:

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

const [index, setIndex] = useState(0);
```

- `index` is a state variable and `setIndex` is the setter function.
- The `[` and `]` syntax here is called array restructuring and it lets you read values from an array. The array returned by `useState` always has exactly two items.
- When you call `useState`, you are telling React that you want this component to remember something: In this case, you want React to remember the `index`.
- The only argument to `useState` is the **initial value** of your state variable. In this example, the `index`’s initial value is set to `0` with `useState(0)`.

Every time your component renders, `useState` gives you an array containing two values:

1. The **state variable** (`index`) with the value you stored.
2. The **state setter function** (`setIndex`) which can update the state variable and trigger React to render the component again.