Namaste React Course by Akshay Saini

Chapter 05 - Let's get Hooked!

Q: What is the difference between `Named export`, `Default export`, and `* as export`?

A: ES6 provides us to import & export a module and use it in other files. ES6 provides two ways to export a module from a file: `named export` and `default export`.
In `Named export`, one can have multiple named exports per file. Then import the specific exports they want surrounded in {} braces. The name of imported module has to be the same as the name of the exported module.
In `Named export`, the component is exported from MyComponent.js file like:

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

and the component is imported from MyComponent.js file like: here we must use {} in MyComponent.

```
// ex. importing a single named export
import { MyComponent } from "./MyComponent";

// ex. importing multiple named exports
import { MyComponent, MyComponent2 } from "./MyComponent";

// ex. giving a named import a different name by using "as":
import { MyComponent2 as MyNewComponent } from "./MyComponent";
```

In `Default export`, One can have only one default export per file. The naming of import is completely independent in default export and we can use any name we like.
In `Default export`, the component is exported from MyComponent.js file like:

```
const MyComponent = () => {}
export default MyComponent;
```

and the component is imported from MyComponent.js file like: here we must omit {} in MyComponent.

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

In `* as export`, it is used to import the whole module as a component and access the components inside the module.
In `* as export`, the component is exported from MyComponent.js file like:

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

and the component is imported from MyComponent.js file like:

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

Now we can use them in JSX as:

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

We can use `Named export` and `Default export` together. So you should export like:

```
export const MyComponent2 = () => {}
const MyComponent = () => {}
export default MyComponent;
```

and import like:

```
import MyComponent, {MyComponent2} from "./MyComponent";
```

Q: What is the importance of `config.js` file?

A: `config.js` files are essentially editable text files that contain information required for the successful operation of a program. The files are structured in a particular way, formatted to be user configurable.
Most of the computer programs we use: whether office suites, web browsers, even video games are configured via menu interfaces.
Configuration files are very simple in structure. For instance, if you were to write an application, and the only thing it ever needed to know was its user's preferred name, then its one and only config file could contain exactly one word: the name of the user. For example:

```
Shivam
```

Usually, though, an application needs to keep track of more than just one piece of information, so configuration often uses a key and a value:

```
NAME='Shivam'
SURNAME='Kumar'
```

Q: What are `React Hooks`?

A: In React version 16.8, React introduced a new pattern called 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.
Hooks allow you to reuse stateful logic without changing your component hierarchy. This makes it easy to share Hooks among many components or with the community.

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.

Q: Why do we need `useState` Hook?

A: `useState` hook is used to maintain the state in our React application. It keeps track of the state changes so basically useState has the ability to encapsulate local state in a functional component.
The useState hook is a special function that takes the `initial state` as an `argument` and `returns an array` of two entries. UseState encapsulate only singular value from the state, for multiple state need to have useState calls.

Syntax for useState hook

```
const [state, setState] = useState(initialstate);
```

Importing: To use useState you need to import useState from react as shown below:

```
import React, { useState } from "react";
```

we can use Hooks in Functional Components

```
const Example = (props) => {
  // You can use Hooks here!
  return <div />;
}
```