Assignment 5 - Let's get Hooked!

1. **What is the difference between Named Export, default export and * as export ?**

- **Named Export:** Export multiple values with specific names.

```
// module.js
export const someValue = 42;
export function someFunction() {
  // ...
}
```

```
// anotherModule.js
import { someValue, someFunction } from './module';
```

- **Default Export:** Export a single "default" value, function, or object**.**

```
// module.js
const defaultExport = 123;
export default defaultExport;
```

```
// anotherModule.js
import myDefault from './module';
```

NOTE: We can also rename the default export during the import

```
import myRenamedDefault from './module';
```

- **Export All (* as Export):** Export all exports from a module and use a namespace to access them. This was introduced in ES2020

```
// module.js
export const value1 = 10;
export const value2 = 20;
```

```
// anotherModule.js
import * as myModule from './module';
console.log(myModule.value1); // 10
console.log(myModule.value2); // 20
```

2. **What is the importance of config.js file ?**

   The config.js file is an important file in many JavaScript projects. It is used to store configuration settings for the project, such as the database URL, the port number, and the environment (production, development, or testing).

   The config.js file is important because it allows us to centralize our configuration settings in one place. This makes it easier to manage our settings and to make changes to them. It also makes it easier to deploy our project to different environments, as we only need to change the config.js file for each environment.
   Eg:

```
const config = {
  // The environment the project is running in.
  environment: "development",

  // The database URL.
  database: "mongodb://localhost:27017/my_database",

  // The port number the project is running on.
  port: 3000,
};

module.exports = config;
```

3. **What are React Hooks?**

React Hooks are a set of functions introduced in React 16.8 that allow developers to use state and other React features without writing a class component. Hooks provide a more concise and readable way to manage component state, side effects, and other behaviors.

Before Hooks, stateful logic in React components was primarily managed through class components. Hooks provide an alternative approach that makes it easier to reuse stateful logic and keep components more focused and modular. Hooks are also easier to test than class methods, because they are not tied to a specific class hierarchy.
Some key React Hooks:

- useState
- useEffect
- useContext
- useReducer
- useCallback
- useMemo
- useRef
- useLayoutEffect
- useImperativeHandle
- useDebugValue
- useReducer

4. Why do we need useState hook?

The useState hook is a React hook that allows you to manage state in a functional component. State is data that can change over time in your React application. For example, you might use state to track the number of items in a shopping cart or the current user's location.

Before React Hooks, the only way to manage state in React was to use class components. Class components have a built-in state property that you can use to store data. However, class components can be difficult to understand and test.

Hooks make it much easier to manage state in functional components. The useState hook takes an initial value as an argument and returns an array with two values: the current state value and a function to update the state value. The current state value can be used in your component, and the update function can be used to change the state value.

Eg:

```jsx
import React, { useState } from 'react';

function Counter() {
  // Declare a state variable named "count" with an initial value of 0
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
    </div>
  );
}
```