Solution: Create a light-dark theme switcher

Here is the completed solution code for the **ThemeContext.js** file:

```
import { createContext, useContext, useState } from "react";

const ThemeContext = createContext(undefined);

export const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState("light");

  return (
    <ThemeContext.Provider
      value={{
        theme,
        toggleTheme: () => setTheme(theme === "light" ? "dark" : "light"),
      }}
    >
      {children}
    </ThemeContext.Provider>
  );
};

export const useTheme = () => useContext(ThemeContext);

```

Here is the solution code for the **Switch/index.js** file:

```
import "./Styles.css";
import { useTheme } from "../ThemeContext";

const Switch = () => {
 const { theme, toggleTheme } = useTheme();
 return (
   <label className="switch">
     <input
       type="checkbox"
       checked={theme === "light"}
       onChange={toggleTheme}
     />
     <span className="slider round" />
   </label>
 );
};

export default Switch;

```

Steps

Step 1

To create the `ThemeProvider`, the first step is to create a new context object, `ThemeContext`, using `createContext`, a function that can be imported from React. The default value argument is only used when a component does not have a matching Provider above it in the tree. This default value can be helpful for testing components in isolation without wrapping them. For the purpose of this exercise, it's not relevant, so `undefined` can be used.

Then, inside the `ThemeProvider` component, you need to define a new piece of local state for the theme, which can be a string whose value is either `"light"` or `"dark"`. It can be initialized to `"light"`, which is usually the default theme for applications.

In the `return` statement, the `ThemeContext.Provider` component should be rendered and wrap the children.

Finally, recall that the value prop for `ThemeContext.Provider` is what gets injected down the tree as context. Since the application needs both the theme value and a way to toggle it, two values are injected: `theme` and `toggleTheme`.

`theme` is just the light-dark theme string value, whereas `toggleTheme` is a function that receives no parameters and just toggles the theme from light to dark and vice versa.

That completes the implementation of the `ThemeProvider` component, as per the code below:

```
import { createContext, useContext, useState } from "react";

const ThemeContext = createContext(undefined);

export const ThemeProvider = ({ children }) => {
 const [theme, setTheme] = useState("light");

 return (
  <ThemeContext.Provider
   value={{
    theme,
    toggleTheme: () => setTheme(theme === "light" ? "dark" : "light"),
   }}
  >
   {children}
  </ThemeContext.Provider>
 );
};
```

Step 2

The implementation for `useTheme` is quite simple. You just need to import the `useContext` hook from React and pass as an argument the `ThemeContext` object defined before. That allows your components to access both `theme` and `toggleTheme` values, which are the ones the `useTheme` custom hook returns.

```
export const useTheme = () => useContext(ThemeContext);
```

Step 3

The `Switch` component can then be connected to the `toggleTheme` function returned from `useTheme` as per the code below:

```
const Switch = () => {
 const { theme, toggleTheme } = useTheme();
 return (
  <label className="switch">
   <input
    type="checkbox"
    checked={theme === "light"}
    onChange={toggleTheme}
   />
   <span className="slider round" />
  </label>
 );
};

```

Step 4

And, finally, you should be able to use the switch widget on the top right corner to change the theme of the application: