Certainly! `useContext` is a hook in React that allows you to subscribe to React context without introducing nesting. It's particularly useful when you have deeply nested components and you want to avoid prop drilling.

Here's a step-by-step guide to using `useContext` with examples:

Step 1: Create a Context

First, you need to create a context using `React.createContext`. This creates a context object with `Provider` and `Consumer` components.

```
// MyContext.js
import { createContext } from 'react';

const MyContext = createContext();

export default MyContext;
```

Step 2: Create a Context Provider

Wrap your application (or part of it) with a `Provider` component. The `Provider` component accepts a `value` prop, which will be the data you want to share.

```
// App.js
import React from 'react';
import MyContext from './MyContext';

const App = () => {
  const contextValue = {
    // Your shared data or functions go here
    message: 'Hello from Context!',
  };

  return (
    <MyContext.Provider value={contextValue}>
      {/* The rest of your components */}
    </MyContext.Provider>
  );
};

export default App;
```

Step 3: Use `useContext` in a Component

Now you can use `useContext` in any component that is a child of your `Provider`.

```
// MyComponent.js
import React, { useContext } from 'react';
import MyContext from './MyContext';

const MyComponent = () => {
  const context = useContext(MyContext);

  return (
    <div>
      <p>{context.message}</p>
    </div>
  );
};

export default MyComponent;
```

Step 4: Access Context Outside of the JSX

If you need to access context outside of JSX (for example, in event handlers), you can still use `useContext` at the top level of your component.

```
// AnotherComponent.js
import React, { useContext } from 'react';
import MyContext from './MyContext';

const AnotherComponent = () => {
  const context = useContext(MyContext);

  const handleClick = () => {
    console.log(context.message);
  };

  return (
    <div>
      <button onClick={handleClick}>Click me</button>
    </div>
  );
};

export default AnotherComponent;
```

Step 5: Render Components

Now, you can render your components in the main file (`index.js` or `App.js`).

```
// index.js
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import MyComponent from './MyComponent';
import AnotherComponent from './AnotherComponent';

ReactDOM.render(
  <React.StrictMode>
    <App />
    <MyComponent />
    <AnotherComponent />
  </React.StrictMode>,
  document.getElementById('root')
);
```

Now, the `MyComponent` and `AnotherComponent` can access the shared context data without passing props through each level of the component tree. Keep in mind that the `useContext` hook will return the current context value for the given context. If the context provider is higher up in the tree, it will look for the nearest provider and use its value.