1. **Display String**

Display `JSX is cool!` as an output.

|  |
| --- |
| **React Code** |
| function App() {}  ReactDOM.render(<App />, document.getElementById("root")); |
| **Result** |
|  |
| **Solution** |
| function App() {    return <div>JSX is cool</div>;  }  ReactDOM.render(<App />, document.getElementById("root")); |

1. **Display Array of Users**

Display all users to the browser. Small Hint: Use array map with react key

|  |
| --- |
| **React Code** |
| const users = [    { name: "John Doe", id: 1 },    { name: "Jane Doe", id: 2 },    { name: "Billy Doe", id: 3 },  ];  function App() {    return (      <>        <h3>User names</h3>        <ul></ul>      </>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |
| **Result** |
|  |
| **Solution** |
| const users = [    { name: "John Doe", id: 1 },    { name: "Jane Doe", id: 2 },    { name: "Billy Doe", id: 3 },  ];  function App() {    return (      <>        <h3>User names</h3>        <ul>          {users.map((user, index) => (            <li key={user.id}>{user.name}</li>          ))}        </ul>      </>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |

1. **Show/Hide Element**

Make the button functional. A click on button should toggle (show/hide) the string `Toggle Challenge` each time it is pressed.

|  |
| --- |
| **React Code** |
| function App() {    const [] = React.useState(true);    return (      <>        <button>Hide Element Below</button>        <div>Toggle Challenge</div>      </>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |
| **Result** |
|  |
| **Solution** |
| function App() {    const [show, setShow] = React.useState(false);    return (      <>        <button onClick={() => setShow(!show)}>          {show ? "Hide Element Below" : "Show Elemnent Below"}        </button>        {show && <div>Toggle Challenge</div>}      </>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |

1. **Two way data binding**

User should be able to type in any characters on input and those character should show below.

|  |
| --- |
| **React Code** |
| function App() {    const [] = React.useState("");    return (      <>        <input type="text" placeholder="Enter Text" />        <p></p>      </>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |
| **Result** |
|  |
| **Solution** |
| function App() {    const [input, setInput] = React.useState("");    return (      <>        <input          type="text"          placeholder="Enter Text"          value={input}          onChange={(e) => setInput(e.target.value)}        />        <p>{input}</p>      </>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |

1. **Disable Button is Input is empty**

Make button disabled when there is no character on the input field. Enable the `Submit` button (remove button from being disabled) when there is at least one character.

|  |
| --- |
| **React Code** |
| function App() {    const [] = React.useState("");    return (      <>        <h3>Disable Button Challenge</h3>        <input type="text" />        <button>Submit</button>      </>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |
| **Result** |
|  |
| **Solution** |
| function App() {    const [input, setInput] = React.useState("");    return (      <>        <h3>Disable Button Challenge</h3>        <input          type="text"          value={input}          onChange={(e) => setInput(e.target.value)}        />        <button disabled={input.length < 1}>Submit</button>      </>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |

1. **Update Parent State from Child**

Parent text (I need to be updated from my child) should be updated when Child button below is clicked. Feel free to use any string to update the parent's current string.

|  |
| --- |
| **React Code** |
| function Child() {    return (      <>        <div>Child</div>        <button>Change Parent Value</button>      </>    );  }  function Parent() {    const [value, setValue] = React.useState(      "I need to be updated from my child"    );    return (      <>        <h3>Update Parent State Challenge (Using Callback)</h3>        <div className="wrapper">          <div>Parent</div>          <div className="box-wrapper">{value}</div>        </div>        <div className="wrapper">          <Child />        </div>      </>    );  }  ReactDOM.render(<Parent />, document.getElementById("root")); |
| **Result** |
| **🡪** |
| **Solution** |
| function Child({ setValue }) {    return (      <>        <div>Child</div>        <button onClick={() => setValue("Updated Text")}>          Change Parent Value        </button>      </>    );  }  function Parent() {    const [value, setValue] = React.useState(      "I need to be updated from my child"    );    return (      <>        <h3>Update Parent State Challenge (Using Callback)</h3>        <div className="wrapper">          <div>Parent</div>          <div className="box-wrapper">{value}</div>        </div>        <div className="wrapper">          <Child setValue={setValue} />        </div>      </>    );  }  ReactDOM.render(<Parent />, document.getElementById("root")); |

1. **React Children**

Show entire Child component content inside Parent component. Only add code on Parent Component below

|  |
| --- |
| **React Code** |
| function Child() {    return <div>This is children content</div>;  }  // Add code only here  function Parent() {    return (      <div>        <h3>Parent Component</h3>      </div>    );  }  function App() {    return (      <Parent>        <Child />      </Parent>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |
| **Result** |
|  |
| **Solution** |
| function Child() {    return <div>This is children content</div>;  }  // Add code only here  function Parent({ children }) {    return (      <div>        <h3>Parent Component</h3>        {children}      </div>    );  }  function App() {    return (      <Parent>        <Child />      </Parent>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |

1. **Sum of two numbers**

Make this app work like a calculator that can add two numbers. When user place numbers on first and second input and hit the button. The sum should appear on the `Total: ` as an output.

|  |
| --- |
| **React Code** |
| function App() {    const [number1, setNumber1] = React.useState();    const [number2, setNumber2] = React.useState();    const [total, setTotal] = React.useState(0);    return (      <div>        <h2>Adding Two Numbers</h2>        <input placeholder="First Number" type="number" />        <input placeholder="Second Number" type="number" />        <button>Add Two Numbers</button>        <p>Total:</p>      </div>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |
| **Result** |
|  |
| **Solution** |
| function App() {    const [number1, setNumber1] = React.useState();    const [number2, setNumber2] = React.useState();    const [total, setTotal] = React.useState(0);    function calculateTotal() {      setTotal(number1 + number2);    }    return (      <div>        <h2>Adding Two Numbers</h2>        <input          placeholder="First Number"          type="number"          value={number1}          onChange={(e) => setNumber1(+e.target.value)}        />        <input          placeholder="Second Number"          type="number"          value={number2}          onChange={(e) => setNumber2(+e.target.value)}        />        <button onClick={calculateTotal}>Add Two Numbers</button>        <p>Total: {total || ""}</p>      </div>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |

1. **Counter App**

Pressing `Increment` button should increase the counter count by one. Pressing `Decrement` button should decrease the counter count by one.

|  |
| --- |
| **React Code** |
| const App = () => {    const [] = React.useState(0);    return (      <div>        <h2>Counter: 0</h2>        <button>Increment</button>        <button>Decrement</button>      </div>    );  };  ReactDOM.render(<App />, document.getElementById("root")); |
| **Result** |
|  |
| **Solution** |
| function App() {    const [number1, setNumber1] = React.useState();    const [number2, setNumber2] = React.useState();    const [total, setTotal] = React.useState(0);    function calculateTotal() {      setTotal(number1 + number2);    }    return (      <div>        <h2>Adding Two Numbers</h2>        <input          placeholder="First Number"          type="number"          value={number1}          onChange={(e) => setNumber1(+e.target.value)}        />        <input          placeholder="Second Number"          type="number"          value={number2}          onChange={(e) => setNumber2(+e.target.value)}        />        <button onClick={calculateTotal}>Add Two Numbers</button>        <p>Total: {total || ""}</p>      </div>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |

1. **Fetch Data from API**

Given the url below, make this app fetch the data and display them in the browser (Small Hint: use `React.useEffect and fetch api`)

// Sample Response

{

id: 1,

name: "Leanne Graham",

username: "Bret",

email: "Sincere@april.biz",

phone: "1-770-736-8031 x56442",

website: "hildegard.org"

}

|  |
| --- |
| **React Code** |
| function App() {    const [userData, setUserData] = React.useState({});    // No need to touch code below    return (      <div className="App">        <h2>User Data</h2>        <p>          <strong>Name: </strong>{" "}          {userData.name || "(Need to populate name here)"}        </p>        <p>          <strong>Website: </strong>          {userData.website || "(Need to populate website here)"}        </p>        <p>          <strong>Email: </strong>          {userData.email || "(Need to populate email here)"}        </p>        <p>          <strong>Phone: </strong>          {userData.phone || "(Need to populate phone here)"}        </p>      </div>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |
| **Result** |
|  |
| **Solution** |
| function App() {    const [userData, setUserData] = React.useState({});    const fetchData = async () => {      const response = await fetch(url);      const jsonData = await response.json();      setUserData(jsonData);    };    React.useEffect(() => {      fetchData();    }, []);    return (      <div className="App">        <h2>User Data</h2>        <p>          <strong>Name: </strong>{" "}          {userData.name || "(Need to populate name here)"}        </p>        <p>          <strong>Website: </strong>          {userData.website || "(Need to populate website here)"}        </p>        <p>          <strong>Email: </strong>          {userData.email || "(Need to populate email here)"}        </p>        <p>          <strong>Phone: </strong>          {userData.phone || "(Need to populate phone here)"}        </p>      </div>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |

1. **Counter with Parent and Child**

Create a counter where the state value is in the Parent component, and the button that triggers the render is in the Child component. Use React Context to pass the Props.

|  |
| --- |
| **React Code** |
| function Child() {    const { count, setCount } = context;    const decrease = () => setCount(count - 1);    const increase = () => setCount(count + 1);    return (      <>        <button onClick={increase}>Increment</button>        <button onClick={decrease}>Decrement</button>      </>    );  }  function Parent() {    const [count, setCount] = React.useState(0);    return (      <>        <div>Counter: {count}</div>        <Child />      </>    );  }  ReactDOM.render(<Parent />, document.getElementById("root")); |
| **Result** |
|  |
| **Solution** |
| const CountContext = React.createContext();  function Child() {    const context = React.useContext(CountContext);    const { count, setCount } = context;    const decrease = () => setCount(count - 1);    const increase = () => setCount(count + 1);    return (      <>        <button onClick={increase}>Increment</button>        <button onClick={decrease}>Decrement</button>      </>    );  }  function Parent() {    const [count, setCount] = React.useState(0);    return (      <>        <CountContext.Provider value={{ count, setCount }}>          <div>Counter: {count}</div>          <Child />        </CountContext.Provider>      </>    );  }  ReactDOM.render(<Parent />, document.getElementById("root")); |

1. **Adding and Deleting Items from list**

Make an Add button that adds the text inside of an input box (if there is any text). After adding, empty the input box. Also, when clicking on the items of the list, delete them.

|  |
| --- |
| **Result** |
|  |
| **Solution** |
| function App() {    const [input, setInput] = React.useState("");    const [list, setList] = React.useState([]);    const deleteItem = (item) => {      setList(list.filter((ele) => ele !== item));    };    const addItem = (value) => {      if (input) {        setList([...list, value]);        setInput("");      }    };    const handleChange = (value) => {      setInput(value);    };    return (      <>        <input          type="text"          value={input}          onChange={(e) => handleChange(e.target.value)}        ></input>        <button onClick={() => addItem(input)}>Add</button>        <ul>          {list.map((item, index) => {            return <li onClick={() => deleteItem(item)}>{item}</li>;          })}        </ul>      </>    );  }  ReactDOM.render(<App />, document.getElementById("root")); |

1. **Data Fetching with Axios and Memoization**

You are tasked with building a React app that displays a list of GitHub repositories for a given user. The challenge has two parts:

**Part 1: Data Fetching**

* Create a component called GitHubRepos that takes a GitHub username as a prop.
* Inside the GitHubRepos component, use Axios to fetch the list of repositories for the provided username from the GitHub API (e.g., [https://api.github.com/users/{username}/repos](https://api.github.com/users/%7busername%7d/repos)).
* Store the fetched data in state.
* Display the list of repository names as a simple unordered list (<ul>) in the component's render.
* Use conditional rendering to display a loading screen while the user waits for the data, and an error screen in case the url doesn’t exist.

**Part 2: Memoization for Optimization**

* Use the useMemo hook to memoize the list of repository names so that it only recomputes when the fetched data changes.
* Whenever the username prop changes, the component should re-fetch data from the API.
* However, the list of repository names should not be recomputed unless the fetched data (the list of repositories) changes.

|  |  |  |
| --- | --- | --- |
| **Result** | | |
| **Successful** | **While Loading** | **If user doesn’t exist** |
|  |  |  |
| **Solution** | | |
| function GitHubRepos({ username }) {    const [repos, setRepos] = React.useState([]);    const [loading, setLoading] = React.useState(true);    const [error, setError] = React.useState(null);    // Fetch GitHub repositories for the provided username    const fetchData = () => {      axios        .get(`https://api.github.com/users/${username}/repos`)        .then((response) => {          setRepos(response.data);          setLoading(false);        })        .catch((err) => {          setError(err);          setLoading(false);        });    };    React.useEffect(() => {      fetchData();    }, [username]);    // Memoize the list of repository names    const repoNames = React.useMemo(() => {      return repos.map((repo) => repo.name);    }, [repos]);    if (loading) {      return <div>Loading...</div>;    }    if (error) {      return <div>Error: {error.message}</div>;    }    return (      <div>        <h2>GitHub Repositories for {username}</h2>        <ul>          {repoNames.map((repoName) => (            <li key={repoName}>{repoName}</li>          ))}        </ul>      </div>    );  }  ReactDOM.render(    <GitHubRepos username="filipeesf18" />,    document.getElementById("root")  ); | | |

1. **Counter with React Toolkit**

Create a React application with a counter component that includes the following features:

* Display a counter value in the center of the screen.
* Include two buttons:
* "Increment" button to increase the counter value by 1.
* "Decrement" button to decrease the counter value by 1.
* Ensure that the "Decrement" button is disabled when the counter is at 0.
* Use React Redux Toolkit library to manage the states.

|  |
| --- |
| **Result** |
|  |
| **Solution** |
| import React from "https://cdn.skypack.dev/react@17.0.1";  import ReactDOM from "https://cdn.skypack.dev/react-dom@17.0.1";  import redux from "https://cdn.skypack.dev/redux@4.0.5";  import reactRedux, {    Provider,    useDispatch,    useSelector,  } from "https://cdn.skypack.dev/react-redux@7.2.2";  import ReduxjsToolkit, {    createSlice,    configureStore,  } from "https://cdn.skypack.dev/@reduxjs/toolkit@1.4.0";  // Define a Redux slice for the counter  const counterSlice = createSlice({    name: "counter",    initialState: 0,    reducers: {      increment: (state) => state + 1,      decrement: (state) => (state > 0 ? state - 1 : 0),    },  });  const { increment, decrement } = counterSlice.actions;  // Create a Redux store  const store = configureStore({    reducer: counterSlice.reducer,  });  // Counter component  function Counter() {    const count = useSelector((state) => state);    const dispatch = useDispatch();    return (      <div>        <h1>Counter: {count}</h1>        <button onClick={() => dispatch(increment())}>Increment</button>        <button onClick={() => dispatch(decrement())} disabled={count === 0}>          Decrement        </button>      </div>    );  }  // Render the Counter component within the Redux Provider  ReactDOM.render(    <Provider store={store}>      <Counter />    </Provider>,    document.getElementById("root")  ); |

1. **Performance improvement**

Improve the performance of the following code.

|  |
| --- |
| **React Code** |
| import React, { useState } from "react";  const ListItem = ({ item, onDelete }) => {    return (      <div>        <span>{item.name}</span>        <button onClick={() => onDelete(item.id)}>Delete</button>      </div>    );  };  const App = () => {    const [items, setItems] = useState(      Array.from({ length: 1000 }, (\_, i) => ({ id: i, name: `Item ${i}` }))    );    const deleteItem = (id) => {      setItems(items.filter((item) => item.id !== id));    };    return (      <div>        <h1>Item List</h1>        {items.map((item) => (          <ListItem key={item.id} item={item} onDelete={deleteItem} />        ))}      </div>    );  };  export default App; |
| **Solution** |
| import React, { useState } from "react";  import { FixedSizeList } from "react-window";  const ListItem = React.memo(({ item, onDelete }) => {    return (      <div>        <span>{item.name}</span>        <button onClick={() => onDelete(item.id)}>Delete</button>      </div>    );  });  const App = () => {    const [items, setItems] = useState(      Array.from({ length: 1000 }, (\_, i) => ({ id: i, name: `Item ${i}` }))    );    const deleteItem = (id) => {      setItems(items.filter((item) => item.id !== id));    };    return (      <div>        <h1>Item List</h1>        <FixedSizeList          height={400}          width={300}          itemSize={50}          itemCount={items.length}        >          {({ index, style }) => (            <ListItem              key={items[index].id}              item={items[index]}              onDelete={deleteItem}              style={style}            />          )}        </FixedSizeList>      </div>    );  };  export default App; |

**Optimizations Made:**

* We wrapped the **ListItem** component with **React.memo** to prevent unnecessary re-renders when the props remain the same.
* We replaced the regular mapping of items with **FixedSizeList** from **react-window** for virtualization. This ensures that only the visible items in the list are rendered, improving performance significantly, especially for large lists.
* We specified the **itemSize** and **itemCount** for the virtualized list, and used the provided index and style props to render each item. This approach is much more efficient than mapping over the entire list.

1. **React Query and useTransition**