1. **Display String**

Display `JSX is cool!` as an output.

|  |
| --- |
| **React Code** |
| import React from "react";  function Exercise1 () {}  export default Exercise1; |
| **Result** |
|  |
| **Solution** |
| import React from "react";  const Exercise1 = () => {    return <div>JSX is cool</div>;  };  export default Exercise1; |

1. **Display Array of Users**

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

|  |
| --- |
| **React Code** |
| import React from "react";  const users = [    { name: "John Doe", id: 1 },    { name: "Jane Doe", id: 2 },    { name: "Billy Doe", id: 3 },  ];  function Exercise2() {    return (      <>        <h3>User names</h3>        <ul></ul>      </>    );  }  export default Exercise2; |
| **Result** |
|  |
| **Solution** |
| import React from "react";  interface User {    name: string;    id: number;  }  const users: User[] = [    { name: "John Doe", id: 1 },    { name: "Jane Doe", id: 2 },    { name: "Billy Doe", id: 3 },  ];  const Exercise2: React.FC = () => {    return (      <>        <h3>User names</h3>        <ul>          {users.map((user) => (            <li key={user.id}>{user.name}</li>          ))}        </ul>      </>    );  };  export default Exercise2; |

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** |
| import React, { useState } from "react";  function Exercise3 () {    const [] = useState(true);    return (      <>        <button>Hide Element Below</button>        <div>Toggle Challenge</div>      </>    );  }  export default Exercise3; |
| **Result** |
|  |
| **Solution** |
| import React, { useState } from "react";  const Exercise3: React.FC = () => {    const [show, setShow] = useState<boolean>(false);    return (      <>        <button onClick={() => setShow(!show)}>          {show ? "Hide Element Below" : "Show Element Below"}        </button>        {show && <div>Toggle Challenge</div>}      </>    );  };  export default Exercise3; |

1. **Two way data binding**

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

|  |
| --- |
| **React Code** |
| import React, { useState } from "react";  function Exercise4 () {    const [] = useState("");    return (      <>        <input type="text" placeholder="Enter Text" />        <p></p>      </>    );  }  export default Exercise4; |
| **Result** |
|  |
| **Solution** |
| import React, { useState } from "react";  const Exercise4: React.FC = () => {    const [input, setInput] = useState<string>("");    return (      <>        <input          type="text"          placeholder="Enter Text"          value={input}          onChange={(e) => setInput(e.target.value)}        />        <p>{input}</p>      </>    );  };  export default Exercise4; |

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** |
| import React, { useState } from "react";  function Exercise5 () {    const [] = useState("");    return (      <>        <h3>Disable Button Challenge</h3>        <input type="text" />        <button>Submit</button>      </>    );  }  export default Exercise5; |
| **Result** |
|  |
| **Solution** |
| import React, { useState } from "react";  const Exercise5: React.FC = () => {    const [input, setInput] = useState<string>("");    return (      <>        <h3>Disable Button Challenge</h3>        <input type="text" value={input} onChange={(e) => setInput(e.target.value)} />        <button disabled={input.length < 1}>Submit</button>      </>    );  };  export default Exercise5; |

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** |
| import React, { useState } from "react";  interface ChildProps {    setValue: (value: string) => void;  }  const Child: React.FC<ChildProps> = ({ setValue }) => {    return (      <>        <div>Child</div>        <button onClick={() => setValue("Updated Text")}>Change Parent Value</button>      </>    );  };  const Parent: React.FC = () => {    const [value, setValue] = useState<string>("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>      </>    );  };  export default Parent; |

1. **React Children**

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

|  |
| --- |
| **React Code** |
| import React, { ReactNode } from "react";  function Child() {    return <div>This is children content</div>;  }  // Add code only here  function Parent() {    return (      <div>        <h3>Parent Component</h3>      </div>    );  }  function Exercise7 () {    return (      <Parent>        <Child />      </Parent>    );  }  export default Exercise7; |
| **Result** |
|  |
| **Solution** |
| import React, { ReactNode } from "react";  const Child: React.FC = () => {    return <div>This is children content</div>;  };  interface ParentProps {    children: ReactNode;  }  const Parent: React.FC<ParentProps> = ({ children }) => {    return (      <div>        <h3>Parent Component</h3>        {children}      </div>    );  };  const Exercise7: React.FC = () => {    return (      <Parent>        <Child />      </Parent>    );  };  export default Exercise7; |

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** |
| import React, { useState } from "react";  function Exercise8 () {    const [number1, setNumber1] = useState();    const [number2, setNumber2] = useState();    const [total, setTotal] = 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>    );  }  export default Exercise8; |
| **Result** |
|  |
| **Solution** |
| import React, { useState } from "react";  const Exercise8: React.FC = () => {    const [number1, setNumber1] = useState<number | undefined>(undefined);    const [number2, setNumber2] = useState<number | undefined>(undefined);    const [total, setTotal] = useState<number | undefined>(0);    function calculateTotal() {      if (typeof number1 === "number" && typeof number2 === "number") {        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>    );  };  export default Exercise8; |

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** |
| import React, { useState } from "react";  const Exercise9= () => {    const [] = useState(0);    return (      <div>        <h2>Counter: 0</h2>        <button>Increment</button>        <button>Decrement</button>      </div>    );  };  export default Exercise9; |
| **Result** |
|  |
| **Solution** |
| import React, { useState } from "react";  const Exercise9: React.FC = () => {    const [count, setCount] = useState<number>(0);    return (      <div>        <h2>Counter: {count}</h2>        <button onClick={() => setCount(count + 1)}>Increment</button>        <button onClick={() => setCount(count - 1)}>Decrement</button>      </div>    );  };  export default Exercise9; |

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** |
| import React, { useState} from "react";  function Exercise10 () {    const [userData, setUserData] = 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>    );  }  export default Exercise10; |
| **Result** |
|  |
| **Solution** |
| import React, { useState, useEffect } from "react";  const url = "https://jsonplaceholder.typicode.com/users/1";  interface UserData {    name?: string;    website?: string;    email?: string;    phone?: string;  }  const Exercise10: React.FC = () => {    const [userData, setUserData] = useState<UserData>({});    const fetchData = async () => {      try {        const response = await fetch(url);        if (!response.ok) {          throw new Error("Failed to fetch data");        }        const jsonData: UserData = await response.json();        setUserData(jsonData);      } catch (error) {        console.error(error);      }    };    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>    );  };  export default Exercise10; |

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** |
| import React from "react";  function Child() {    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] = useState(0);    return (      <>        <div>Counter: {count}</div>        <Child />      </>    );  }  export default Parent; |
| **Result** |
|  |
| **Solution** |
| import React, { createContext, useContext } from "react";  interface CountContextType {    count: number;    setCount: React.Dispatch<React.SetStateAction<number>>;  }  const CountContext = createContext<CountContextType | undefined>(undefined);  const Child: React.FC = () => {    const context = useContext(CountContext);    if (!context) {      throw new Error("Child component must be used within a CountContextProvider");    }    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>      </>    );  };  const Parent: React.FC = () => {    const [count, setCount] = React.useState<number>(0);    return (      <>        <CountContext.Provider value={{ count, setCount }}>          <div>Counter: {count}</div>          <Child />        </CountContext.Provider>      </>    );  };  export default Parent; |

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** |
| import React, { useState } from "react";  const Exercise12: React.FC = () => {    const [input, setInput] = useState<string>("");    const [list, setList] = useState<string[]>([]);    const deleteItem = (item: string) => {      setList(list.filter((ele) => ele !== item));    };    const addItem = (value: string) => {      if (input) {        setList([...list, value]);        setInput("");      }    };    const handleChange = (value: string) => {      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 key={index} onClick={() => deleteItem(item)}>                {item}              </li>            );          })}        </ul>      </>    );  };  export default Exercise12; |

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** | | |
| import React, { useState, useEffect, useMemo } from "react";  import axios from "axios";  interface Repo {    name: string;  }  interface Error {    message: string;  }  const App: React.FC<{ username: string }> = ({ username }) => {    const [repos, setRepos] = useState<Repo[]>([]);    const [loading, setLoading] = useState<boolean>(true);    const [error, setError] = useState<Error | null>(null);    // Fetch GitHub repositories for the provided username    const fetchData = () => {      axios        .get<Repo[]>(`https://api.github.com/users/${username}/repos`)        .then((response) => {          setRepos(response.data);          setLoading(false);        })        .catch((err) => {          setError(err);          setLoading(false);        });    };    useEffect(() => {      fetchData();    }, [username]);    // Memoize the list of repository names    const repoNames = 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>    );  };  function Exercise13() {    return <App username="filipeesf19"></App>;  }  export default Exercise13; | | |

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 "react";  import { Provider, useDispatch, useSelector } from "react-redux";  import { createSlice, configureStore } from "@reduxjs/toolkit";  // Define a Redux slice for the counter  interface CounterState {    value: number;  }  const counterSlice = createSlice({    name: "counter",    initialState: { value: 0 } as CounterState,    reducers: {      increment: (state) => {        state.value += 1;      },      decrement: (state) => {        state.value = state.value > 0 ? state.value - 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: CounterState) => state.value);    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  function Exercise14() {    return (      <Provider store={store}>        <Counter />      </Provider>    );  }  export default Exercise14; |

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 Exercise15= () => {    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 Exercise15; |
| **Solution** |
| import React, { useState } from "react";  interface Item {    id: number;    name: string;  }  interface ListItemProps {    item: Item;    onDelete: (id: number) => void;  }  const ListItem: React.FC<ListItemProps> = ({ item, onDelete }) => {    return (      <div>        <span>{item.name}</span>        <button onClick={() => onDelete(item.id)}>Delete</button>      </div>    );  };  const Exercise15: React.FC = () => {    const [items, setItems] = useState<Item[]>(      Array.from({ length: 1000 }, (\_, i) => ({ id: i, name: `Item ${i}` }))    );    const deleteItem = (id: number) => {      setItems(items.filter((item) => item.id !== id));    };    return (      <div>        <h1>Item List</h1>        <div style={{ height: "400px", overflowY: "auto" }}>          {items.map((item) => (            <ListItem key={item.id} item={item} onDelete={deleteItem} />          ))}        </div>      </div>    );  };  export default Exercise15; |

**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**