//Activity1: User understands that in scenarios when child component has functions passed as prop, child components gets re-rendered unnecessarily when parent re-renders, due to the function reference changing.

```
import React from 'react';

import ReactDOM from 'react-dom';


const ParentComponent = () => {

  const handleClick = () => {

    console.log('Button clicked');

  };


  return (

    <div>

      <h1>Parent Component</h1>

      <ChildComponent onClick={handleClick} />

    </div>

  );

};


const ChildComponent = ({ onClick }) => {

  console.log('ChildComponent rendered');

  return <button onClick={onClick}>Click me</button>;

};


ReactDOM.render(<ParentComponent />, document.getElementById('root'));
```

//activity2: User should be able to use useCallback() to avoid function reference change when a component re-renders

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

import ReactDOM from 'react-dom';


const ParentComponent = () => {

  const [count, setCount] = useState(0);
  const handleClick = () => {
    console.log('Button clicked');
  };

  const memoizedHandleClick = useCallback(() => {
    handleClick();
  }, []); // No dependencies

  return (
    <div>
      <h1>Parent Component</h1>
      <p>Counter: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment Parent</button>
      <ChildComponent onClick={memoizedHandleClick} />
    </div>
  );
};

const ChildComponent = ({ onClick }) => {
  console.log('ChildComponent rendered');
  return <button onClick={onClick}>Click me</button>;
};

ReactDOM.render(<ParentComponent />, document.getElementById('root'));
```

//Activity3: User should be able ot use React.Memo() along with useCallback() to resolve the unnecessary re-render issue

```
import React, { useState, useCallback } from 'react';
import ReactDOM from 'react-dom';

const ChildComponent = React.memo(({ data }) => {
  return (
    <div>
      <h2>User Details</h2>
      <p>{data ? `Name: ${data.name}, Email: ${data.email}` : 'No user data fetched'}</p>
    </div>
  );
});

const ParentComponent = () => {
  const [data, setData] = useState(null);
  const [userId, setUserId] = useState(1);

  const fetchData = useCallback(async () => {
    try {
      const response = await fetch(`https://jsonplaceholder.typicode.com/users/${userId}`);
      const fetchedData = await response.json();
      setData(fetchedData);
    } catch (error) {
      console.error('Error fetching data:', error);
    }
  }, [userId]);

  return (
    <div>
      <ChildComponent data={data} />
      <button onClick={fetchData}>Fetch User Data</button>
      <input
        type="number"
        value={userId}
        onChange={(e) => setUserId(e.target.value)}
        placeholder="Enter User ID"
      />
    </div>
  );
};

ReactDOM.render(
  <ParentComponent />,
  document.getElementById('root')
);

ReactDOM.render(<ParentComponent />, document.getElementById('root'));
//Activity 4: User understands the need of dependency array in useCallback Hook
import React, { useState, useCallback } from 'react';
import ReactDOM from 'react-dom';

const ParentComponent = () => {
  const [count, setCount] = useState(0);

  const handleClick = useCallback(() => {
    setCount(prevCount => prevCount + 1);
  }, [setCount]);

  return (
    <div>
      <h2>Parent Component</h2>
      <p>Count: {count}</p>
      <ChildComponent onClick={handleClick} />
    </div>
  );
};
```

```
const ChildComponent = ({ onClick }) => {
  return (
    <div>
      <h2>Child Component</h2>
      <button onClick={onClick}>Click Me</button>
    </div>
  );
};

const App = () => {
  return (
    <div>
      <ParentComponent />
    </div>
  );
};

ReactDOM.render(<App />, document.getElementById('root'));
//Activity 5: User should be able to use dependency array with useCallback
import React, { useState, useCallback } from 'react';
import ReactDOM from 'react-dom';

const ParentComponent = () => {
  const [data, setData] = useState(null);
  const [userId, setUserId] = useState(1);

  const fetchData = useCallback(async () => {
    try {
      const response = await fetch(https://jsonplaceholder.typicode.com/users/${userId});
      const fetchedData = await response.json();
      setData(fetchedData);
    } catch (error) {
      console.error('Error fetching data:', error);
    }
  }, [userId]);

  return (
    <div>
      <h2>User Details</h2>
      <p>{data ? Name: ${data.name}, Email: ${data.email} : 'No user data fetched'}</p>
      <button onClick={fetchData}>Fetch User Data</button>
      <input
        type="number"
        value={userId}
        onChange={(e) => setUserId(e.target.value)}
        placeholder="Enter User ID"
      />
    </div>
  );
};

ReactDOM.render(
  <ParentComponent />,
  document.getElementById('root')
);
```