// Question 1: How are events handled in React compared to vanilla JavaScript? Explain the concept of synthetic events.

/*

In React, events are handled via synthetic events. React creates a wrapper around native DOM events to normalize them across different browsers.

Unlike vanilla JS, you don't need to manually add event listeners; React automatically handles event delegation.

*/

```
const handleClick = () => alert('Button clicked!');

<button onClick={handleClick}>Click Me</button>
```

// Question 2: What are some common event handlers in React.js? Provide examples of onClick, onChange, and onSubmit.

/*

onClick: Triggered when an element is clicked.

Example:

*/

```
<button onClick={() => alert("Clicked!")}>Click Me</button>
```

/*

onChange: Triggered when the value of an input field changes.

Example:

*/

```
<input type="text" onChange={(e) => console.log(e.target.value)} />
```

/*

onSubmit: Triggered when a form is submitted.

```
  Example:
*/

<form onSubmit={(e) => { e.preventDefault(); alert('Form submitted!'); }}>
  <button type="submit">Submit</button>
</form>

// Question 3: Why do you need to bind event handlers in class components?

/*
  In class components, event handlers need to be bound to the class instance to maintain the correct this context.
*/

class MyComponent extends React.Component {
  constructor() {
    super();
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    alert(this);
  }

  render() {
    return <button onClick={this.handleClick}>Click Me</button>;
  }
}

// LAB EXERCISE
```

// Task 1: Create a button in a React component that, when clicked, changes the text from "Not Clicked" to "Clicked!" using event handling.

```
class ButtonClick extends React.Component {
  constructor() {
    super();
    this.state = { text: "Not Clicked" };
  }

  handleClick = () => {
    this.setState({ text: "Clicked!" });
  };

  render() {
    return (
      <div>
        <p>{this.state.text}</p>
        <button onClick={this.handleClick}>Click Me</button>
      </div>
    );
  }
}
```

// Task 2: Create a form with an input field in React. Display the value of the input field dynamically as the user types in it.

```
class DynamicInput extends React.Component {
  constructor() {
    super();
    this.state = { inputValue: "" };
  }
```

```
  handleInputChange = (event) => {
    this.setState({ inputValue: event.target.value });
  };

  render() {
    return (
      <div>
        <input
          type="text"
          value={this.state.inputValue}
          onChange={this.handleInputChange}
        />
        <p>{this.state.inputValue}</p>
      </div>
    );
  }
}

// Question 1: What is conditional rendering in React? How can you conditionally render elements in a React component?

/*
  Conditional rendering refers to rendering elements based on certain conditions.
  You can use JavaScript expressions like if, ternary operators, or logical && to conditionally render JSX.
*/

if (isLoggedIn) {
  return <button>Logout</button>;
} else {
  return <button>Login</button>;
}
```

// Question 2: Explain how if-else, ternary operators, and && (logical AND) are used in JSX for conditional rendering.

```
/*
  If-else: Not directly used in JSX, but can be done in functions.
  Ternary operator:
*/

{isLoggedIn ? <button>Logout</button> : <button>Login</button>}

/*
  && (Logical AND): Renders the right-hand side only if the left-hand side is true.
*/

{isLoggedIn && <button>Logout</button>}
```

// LAB EXERCISE

// Task 1: Create a component that conditionally displays a login or logout button based on the user's login status.

```
class LoginLogout extends React.Component {
  constructor() {
    super();
    this.state = { loggedIn: false };
  }

  toggleLoginStatus = () => {
    this.setState((prevState) => ({ loggedIn: !prevState.loggedIn }));
  };
```

```
  render() {
    return (
      <div>
        {this.state.loggedIn ? (
          <button onClick={this.toggleLoginStatus}>Logout</button>
        ) : (
          <button onClick={this.toggleLoginStatus}>Login</button>
        )}
      </div>
    );
  }
}

// Task 2: Implement a component that displays a message like "You are eligible to vote" if the user is over 18, otherwise display "You are not eligible to vote."

const VoteEligibility = ({ age }) => {
  return (
    <div>
      {age >= 18
        ? "You are eligible to vote"
        : "You are not eligible to vote"}
    </div>
  );
};

// Question 1: How do you render a list of items in React? Why is it important to use keys when rendering lists?

/*
  You can render a list using map() to iterate through the array of items and return JSX for each item.
```

```
  Keys are important because they help React efficiently update and manage the list by uniquely identifying each element.

*/


const fruits = ["Apple", "Banana", "Orange"];

const fruitList = fruits.map((fruit, index) => (

  <li key={index}>{fruit}</li>

));


// Question 2: What are keys in React, and what happens if you do not provide a unique key?


/*

  Keys are unique identifiers for each element in a list. Without a unique key, React will have trouble optimizing updates,

  which can lead to performance issues or incorrect rendering.

*/


// LAB EXERCISE


// Task 1: Create a React component that renders a list of items (e.g., a list of fruit names). Use the map() function to render each item in the list.


const FruitList = () => {

  const fruits = ["Apple", "Banana", "Orange"];

  return (

    <ul>

      {fruits.map((fruit, index) => (

        <li key={index}>{fruit}</li>

      ))}

    </ul>

  );

};
```

// Task 2: Create a list of users where each user has a unique id. Render the user list using React and assign a unique key to each user.

```
const UserList = () => {
  const users = [
    { id: 1, name: "John" },
    { id: 2, name: "Jane" },
    { id: 3, name: "Doe" },
  ];
  return (
    <ul>
      {users.map((user) => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
};
```

// Question 1: How do you handle forms in React? Explain the concept of controlled components.

```
/*
  Controlled components are React components that render a form element and control its value via React state.
  Form elements like inputs, selects, and textareas are bound to the state.
*/

class FormExample extends React.Component {
  constructor() {
    super();
    this.state = { name: "" };
  }
```

```
  handleChange = (event) => {
    this.setState({ name: event.target.value });
  };

  render() {
    return (
      <form>
        <input
          type="text"
          value={this.state.name}
          onChange={this.handleChange}
        />
      </form>
    );
  }
}

// Question 2: What is the difference between controlled and uncontrolled components in React?

/*
  Controlled components are those where the value is controlled by React state.
  Uncontrolled components store their state internally and are typically accessed using refs.
*/

// LAB EXERCISE

// Task 1: Create a form with inputs for name, email, and password. Use state to control the form and display the form data when the user submits it.

class Form extends React.Component {
```

```
constructor() {
  super();
  this.state = { name: "", email: "", password: "" };
}

handleChange = (event) => {
  this.setState({ [event.target.name]: event.target.value });
};

handleSubmit = (event) => {
  event.preventDefault();
  alert(`Name: ${this.state.name}, Email: ${this.state.email}`);
};

render() {
  return (
    <form onSubmit={this.handleSubmit}>
      <input
        type="text"
        name="name"
        value={this.state.name}
        onChange={this.handleChange}
      />
      <input
        type="email"
        name="email"
        value={this.state.email}
        onChange={this.handleChange}
      />
      <input
        type="password"
```

```
        name="password"
        value={this.state.password}
        onChange={this.handleChange}
      />
      <button type="submit">Submit</button>
    </form>
  );
 }
}
```

// Task 2: Add validation to the form created above. For example, ensure that the email input contains a valid email address.

```
handleSubmit = (event) => {
 event.preventDefault();
 const { email } = this.state;
 if (!email.includes("@")) {
  alert("Please enter a valid email.");
  return;
 }
 alert(`Name: ${this.state.name}, Email: ${email}`);
};
```