

✔ Congratulations! You passed!

Grade received 90% Latest Submission Grade 90% To pass 80% or higher

Retake the assignment
in 7h 54m

Go to
next
item

1. You are building a form using both Formik and Yup libraries, where one of the inputs is an email. Before the form gets submitted to the server, you would like to set up some client validation with Yup to make sure the field has an email that is valid, otherwise a message "Invalid email address" would be shown on the screen. This field is also required. Choose the correct validation code from the three code snippets.

1 / 1 point

☒

```
1  Yup.string().email("Invalid email address").required("Required")
```

☐

```
1  Yup.email("Invalid email address").required("Required")
```

☐

```
1  Yup.email().string("Invalid email address").required("Required")
```

✔ Correct

Correct, first Yup needs to know the type of value (string) and then chain the different validation rules with their associated error message to show.

2. You have the following React application where you have a `ToDo` component that has two text labels and an uncontrolled text input and the entry point App component that renders a list of two Todos and a button to reverse the order of the Todos. To avoid a React keys warning, a key is provided to each `ToDo` component, with the index as its value. Suppose that the next sequence of events happen in the application:

1 / 1 point

1. You write "Wash dishes" in the first `ToDo` input
2. You write "Buy groceries" in the second `ToDo` input
3. You click the button to reverse the order

What would happen on the screen after that?

```
1  const ToDo = props => (  
2    <tr>  
3      <td>  
4        <label>{props.id}</label>  
5      </td>  
6      <td>  
7        <input />  
8      </td>  
9      <td>  
10       <label>{props.createdAt}</label>  
11     </td>  
12   </tr>  
13 );  
14  
15  
16 function App() {  
17   const [todos, setTodos] = useState([  
18     {  
19       id: 'todo1',  
20       createdAt: '18:00',
```

```

21   },
22   {
23     id: 'todo2',
24     createdAt: '20:30',
25   }
26   });
27
28   const reverseOrder = () => {
29     // Reverse is a mutative operation, so we need to create a new array first.
30     setTodos([...todos].reverse());
31   };
32
33   return (
34     <div>
35       <button onClick={reverseOrder}>Reverse</button>
36       {todos.map((todo, index) => (
37         <ToDo key={index} id={todo.id} createdAt={todo.createdAt} />
38       ))}
39     </div>
40   );

```

- ☐ todo1 Buy groceries 18:00
todo2 Wash dishes 20:30
- ☒ todo2 Wash dishes 20:30
todo1 Buy groceries 18:00
- ☐ todo2 Buy groceries 20:30
todo1 Wash dishes 18:00

✓ **Correct**

Correct, when reversing the order React understands they are still the same nodes with key=1 and key=2, so it will preserve their internal state (input value). Since the props are different though, it will just update the node with the new prop values.

3. True or false: There are at least two errors in the code below.

1 / 1 point

```

1  import { createContext, useContext, useState } from "react";
2
3  const ThemeContext = createContext(undefined);
4
5  export const ThemeProvider = () => {
6    const [theme, setTheme] = useState("light");
7
8    return(
9      <ThemeContext.Provider
10        value={{
11          theme,
12          toggleTheme: () => setTheme(!theme),
13        }}
14      >
15        </ThemeContext.Provider>
16      );
17  };

```

- ☒ True
- ☐ False

✓ **Correct**

Correct, there are two errors in this code. First, the **toggleTheme** implementation is incorrect and should be: **toggleTheme: () => setTheme(theme === "light" ? "dark" : "light")**. Second, **ThemeProvider** should use the children prop and pass it as a direct child of **ThemeContext.Provider**.

4.

1 / 1 point

True or False: The type of a React element can be a DOM node, such as, for example, an HTML button.

- ☒ True

✓ Correct

Correct, the type can be a DOM node.

☐ False.

5. True or false: When the user clicks the Submit button, the "WithClick" string will never be output to the console.

1 / 1 point

```
1  const Button = ({ children, ...rest }) => (  
2    <button onClick={() => console.log("ButtonClick")} {...rest}>  
3      {children}  
4    </button>  
5  );  
6  
7  const withClick = (Component) => {  
8    const handleClick = () => {  
9      console.log("WithClick");  
10   };  
11  
12   return(props) => {  
13     return<Component {...props} onClick={handleClick} />;  
14   };  
15 };  
16  
17 const MyButton = withClick(Button);  
18  
19 export default function App() {  
20   return <MyButton onClick={() => console.log("AppClick")}>Submit</MyButton>;  
21 }
```

☐ True

☒ False

✓ Correct

Correct. It is false to claim that the WithClick string will never be output to the console. Actually, due to the order of the spread operator in the different components, the **withClick** Higher-order component (HOC) takes precedence, and is the thing to be console logged.

6.

1 / 1 point

True or False: Using jest and react-testing-library, to assert that a function has been called with some specific arguments, you would need to use the **toHaveBeenCalledWith** matcher.

☐ False.

☒ True.

✓ Correct

Correct, this is the proper matcher to check the arguments of the function call.

7.

1 / 1 point

Among the following code examples, what are valid implementations of the render props pattern?

✓

```
1  <MealProvider render={data => (  
2    <p>Ingredients: {data.ingredients}</p>  
3  )} />
```

✓ Correct

Correct, it uses a render type prop that is a function that returns JSX.

✓

```
1  <Row renderIcon={() => <Icon name="add" />} />
```



Correct

Correct, it uses a render type prop that is a function that returns JSX. Even though the name is not exactly "render", explicit variations are also valid.



```
1 <LoginUser renderUser={<p>Mark</p>} />
```

8. You need the below code snippet to run only after the initial render. What updates (if any) do you need to make to the code?

1 / 1 point

```
1 React.useEffect(()=> {
2   console.log('The value of the toggle variable is', toggle)
3 })
```

- ☒ Add an empty dependency array.
- ☐ You shouldn't make any updates.
- ☐ You should remove the toggle variable .



Correct

Correct! To run the effect only on the initial render, you need an empty dependency array.

9. You are given the below piece of code.

0 / 1 point

```
1 import {useState} from "react";
2
3 export default function App() {
4   const [restaurantName, setRestaurantName] = useState("Lemon");
5
6   function updateRestaurantName() {
7     setRestaurantName("Little Lemon");
8   };
9
10  return (
11    <div>
12      <h1>{restaurantName}</h1>
13      <button onClick={updateRestaurantName}>
14        Update restaurant name
15      </button>
16    </div>
17  );
18 };
```

True or false: The `restaurantName` variable's value will always be reset between re-renders of the App component.

- ☒ True
- ☐ False



Incorrect

Not quite. Please revise the lesson item in Module 2, Lesson 1, titled, Working with Complex Data in useState.

10. Is this valid code?

1 / 1 point

```
1  if (data !== '') {  
2    useEffect(() => {  
3      | setData('test data');  
4    });  
5  }
```

☒ No

☐ Yes

☒ **Correct**

Correct. If you use a hook in a condition, you're breaking rules! Thus, the below code is invalid.