

✔ Congratulations! You passed!

Grade received 100% To pass 80% or higher

Go to next item

1. Let's suppose you have the below JSX that gets returned from a component, what would be the equivalent object representation (Element) that React will create internally?

1 / 1 point

```
1 <button className='button-primary'>
2   <div>
3     | Submit
4   </div>
5 </button>
```

☐

```
1 {
2   type: "button",
3   props: {
4     className: "button-primary",
5     children: {
6       type: "div",
7       children: "Submit"
8     },
9   },
10 }
```

☒

```
1 {
2   type: "button",
3   props: {
4     className: "button-primary",
5     children: {
6       type: "div",
7       props: {
8         children: "Submit",
9       }
10    },
11  },
12 }
```

☐

```
1 {
2   type: Button,
3   props: {
4     className: "button-primary",
5     children: "div",
6   },
7 }
```

✔ Correct

That's correct, the **children** key also should point to an element, so it has to have two keys: **type** and **props**.

2. What is the concept of component specialization?

1 / 1 point

- ☒ A component defined as a special case of another more generic component.  
☐ A component that is designed to fulfill one specific purpose and nothing else.  
☐ A component that doesn't know its children ahead of time and acts as a generic box.

✔ Correct

That's correct. For example, a **SubmitButton** component is a more specialized version of a **Button** component.

3. You would like to clone a React element by using the `React.cloneElement` API, where the particular element has the below structure:

1 / 1 point

```
1 const buttonElement = {
2   type: SubmitButton,
3   props: {
4     color: "green",
5     children: "Submit!",
6   },
7 };
```

What would be the value of the variable output when using the API with the following parameters?

```
const output = React.cloneElement(buttonElement, {disabled: true, color: "blue"});
```

☒

```
1 {
2   type: SubmitButton,
3   props: {
4     color: "blue",
5     children: "Submit!",
6     disabled: true,
7   },
8 }
```

☐

```
1 {
2   type: SubmitButton,
3   props: {
4     disabled: true,
5     color: "blue",
6   },
7 };
```

☐

```
1 {
2   type: SubmitButton,
3   props: {
4     color: "green",
5     children: "Submit!",
6     disabled: true,
7   },
8 }
```

✔ Correct

That's correct, the **color** props gets overridden and a new prop called **disabled** with a value of **true** gets added.

4. Imagine you are using the spread operator in the below component as follows:

1 / 1 point

```
1 const props = { title: "tiramisu", cal: 400 };
2 const element = <Component title="cake" {...props} cal={500} />;
```

What would be the value of `element.props`?

☐

```
1 { title: "tiramisu", cal: 400 }
```

☐

```
1 { title: "cake", cal: 400 }
```

☒ 1 `{ title: "tiramisu", cal: 500 }`

☐ 1 `{ title: "cake", cal: 500 }`

☒ Correct

That's correct, that's the final value.

5. Amongst the below expressions, select all that will not render anything on the screen when being used in JSX.

1/1 point

☒ 1 `<div>{false}</div>`

☒ Correct

That's correct, the `false` value will not render anything.

☒ 1 `<div>{null}</div>`

☒ Correct

That's correct, the `null` value will not render anything.

☒ 1 `<div>{undefined}</div>`

☒ Correct

That's correct, the `undefined` value will not render anything

☒ 1 `<div>{(() => true)()}</div>`

☒ Correct

That's correct, the function gets invoked instantly in place and returns a value of `true`, which will not render anything.