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

```
<button className='button-primary'>
   <div>
      Submit
   </div>
</button>
```

○
```
{
    type: Button,
    props: {
        className: "button-primary",
        children: "div",
    },
}
```

[✓]
```
{
    type: "button",
    props: {
        className: "button-primary",
        children: {
            type: "div",
            props: {
                children: "Submit",
            }
        },
    },
}
```

○
```
{
    type: "button",
    props: {
        className: "button-primary",
        children: {
            type: "div",
            children: "Submit"
        },
    },
}
```

✓ **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 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.

[✓ A component defined as a special case of another more generic component.]

✓ **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

```
const buttonElement = {
  type: SubmitButton,
  props: {
    color: "green",
    children: "Submit!",
  },
};
```

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" });`

◉
```
{
  type: SubmitButton,
  props: {
    color: "blue",
    children: "Submit!",
    disabled: true,
  },
};
```

○
```
{
  type: SubmitButton,
  props: {
    disabled: true,
    color: "blue",
  },
};
```

○
```
{
  type: SubmitButton,
  props: {
    color: "green",
    children: "Submit!",
    disabled: true,
  },
};
```

✓ **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

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

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

○
```
{ title: "cake", cal: 500 }
```

◉
```
{ title: "tiramisu", cal: 500 }
```

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

○
```
{ title: "cake", cal: 400 }
```

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**

☑
```
<div>{null}</div>
```

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

☑
```
<div>{false}</div>
```

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

☑
```
<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.

☑
```
<div>{undefined}</div>
```

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