✓ Congratulations! You passed!

Grade received 100% **Latest Submission Grade** 100% **To pass** 80% or higher

Retake the assignment in **7h 58m**

Go to next item

1. What are some of the features of component containment? Select all that apply. **1 / 1 point**

[✓] A component that uses the **`children`** prop to pass **`children`** elements directly as their content.

✓ **Correct**
Correct, all the content of the generic box is provided via the **`children`** prop.

[✓] The fact that some components don't know their children ahead of time.

✓ **Correct**
Correct, they leverage the **`children`** prop.

[✓] A component that acts as a generic box.

✓ **Correct**
Correct, like a Dialog or Alert.

[ ] A special case of other components.

2. What are the props that all components have by default? Select all that apply. **1 / 1 point**

● **`children`**

○ **`type`**

○ **`render`**

✓ **Correct**
Correct, all components have an implicit **`children`** prop.

3. What is a React Element? Select all that apply. **1 / 1 point**

[ ] A React Component that represents a simple DOM node, like a button.

[✓] A JavaScript object that represents the final HTML output.

✓ **Correct**
Correct, they represent what the UI should look like.

[✓] An intermediary representation that describes a component instance.

✓ **Correct**
Correct, JSX gets transformed into that intermediary representation that is a descriptive object.

4. Assuming you have the below component, what are all the features implemented from component composition with children? **1 / 1 point**

```
function ConfirmationDialog() {
  return (
    <Dialog color="blue">
      <h1 className="Dialog-title">
        Thanks!
      </h1>
      <p className="Dialog-message">
        We'll process your order in less than 24 hours.
      </p>
    </Dialog>
  );
}
```

- ○ Component specialization.
- [✓ Component specialization and component containment.]
- ○ Component containment.

> ✓ **Correct**
> Correct, **`ConfirmationDialog`** is a special case of **`Dialog`** and the **`Dialog`** is an example of a generic box (containment) that uses children to lay out the content.

5. What are some of the use cases that the **`React.cloneElement`** API allows you to achieve? Select all that apply. **1 / 1 point**

- [✓ Extend the functionality of children components.]

> ✓ **Correct**
> That's correct. The **`React.cloneElement`** API allows you to extend the functionality of children components.

- [✓ Add to children properties.]

> ✓ **Correct**
> That's correct. The **`React.cloneElement`** API allows you to add to children's properties.

- [✓ Modify children's properties.]

> ✓ **Correct**
> That's correct. The **`React.cloneElement`** API allows you to modify children's properties.

6. Assuming you have the following **`Row`** component that uses **`React.Children.map`** to perform some dynamic transformation in each **`child`** element, in order to add some custom styles, what's wrong about its implementation? Select all that apply. **1 / 1 point**

```
const Row = ({ children, spacing }) => {
  const childStyle = {
    marginLeft: `${spacing}px`,
  };

  return(
    <div className="Row">
      {React.Children.map(children, (child, index) => {
        child.props.style = {
          ...child.props.style,
          ...(index > 0 ? childStyle : {}),
        };

        return child;
      })}
    </div>
  );
};
```

- ○ Each child is missing a key, causing potential problems if the list order changes.
- ○ You can't use the spread operator in the style prop.
- [✓ Each child is being mutated.]

> ✓ **Correct**
> Correct, props are being mutated and that is a React breaking rule. You should use **`React.cloneElement`** to create a copy of the elements first.

7. Assuming you have the following set of components, what would be logged into the console when clicking the Submit button that gets rendered on the screen? **1 / 1 point**

```
const Button = ({ children, ...rest }) => (
  <button onClick={() => console.log("ButtonClick")} {...rest}>
    {children}
  </button>
);

const withClick = (Component) => {
  const handleClick = () => {
    console.log("WithClick");
  };

  return (props) => {
    return <Component onClick={handleClick} {...props} />;
  };
};

const MyButton = withClick(Button);

export default function App() {
  return <MyButton onClick={() => console.log("AppClick")}>Submit</MyButton>;
}
```

- ( ) `“ButtonClick”`
- [✓ `“AppClick”`]
- ( ) `“WithClick”`

✓ **Correct**
Correct, due to the order of the spread operator in the different components, the original **`onClick`** prop passed to **`MyButton`** takes precedence.

8. Among the below options, what are valid solutions to encapsulate cross-cutting concerns? Select all that apply

1 / 1 point

- [✓ Render props pattern.]

✓ **Correct**
Correct, that’s one possible abstraction.

- [✓ Custom hooks.]

✓ **Correct**
Correct, that’s one possible abstraction.

- [✓ Higher order components.]

✓ **Correct**
Correct, that’s one possible abstraction.

- ☐ Components that consume context.

9. What does the screen utility object from react-testing-library represent when performing queries against it?

1 / 1 point

- [✓ The whole page or root document]
- ( ) The whole virtual DOM
- ( ) Your laptop screen

✓ **Correct**
That’s correct, the screen utility object from react-testing-library represents the root document when performing queries against it.

10. When writing tests with Jest and react-testing-library, what matcher would you have to use to assert that a button is disabled?

1 / 1 point

- [✓ `toHaveAttribute`]
- ( ) `toBeInTheDocument`
- ( ) `toHaveBeenCalled`

✓ **Correct**
That’s correct, When writing tests with Jest and react-testing-library, you would use **`toHaveAttribute`** to assert that a button is disabled.