Congratulations! You passed!

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

Go to next item

You increased your skill scores!

Computer Programming
Your score: **266** (↑7) Intermediate
Well done! At an intermediate level, you have a solid understanding of the material and are able to pass intermediate content. You can apply key concepts on most tasks.

Web Development Your score: **219** (↑9) Intermediate
Well done! At an intermediate level, you have a solid understanding of the material and are able to pass intermediate content. You can apply key concepts on most tasks.

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

- [x] The fact that some components don't know their children ahead of time.
  - Correct
    Correct, they leverage the `children` prop.
- [x] A component that acts as a generic box.
  - Correct
    Correct, like a Dialog or Alert.
- [ ] A special case of other components.
- [x] 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.

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

- (•) `children`
- ( ) `render`
- ( ) `type`
  - Correct
    Correct, all components have an implicit `children` prop.

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

- [x] An intermediary representation that describes a component instance.
  - Correct
    Correct, JSX gets transformed into that intermediary representation that is a descriptive object.
- [x] A JavaScript object that represents the final HTML output.
  - Correct
    Correct, they represent what the UI should look like.
- [ ] A React Component that represents a simple DOM node, like a button.

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 and component containment.
- ( ) Component containment.
- ( ) Component specialization.

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**

- [x] Extend the functionality of children components.

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

- [x] Add to children properties.

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

- [x] 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**

```
    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>
  );
};
```

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

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

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

const MyButton = withClick(Button);

export default function App() {
```

```
20    return <MyButton onClick={() => console.log("AppClick")}>Submit</MyButton>;
21  }
```

- (●) `"AppClick"`
- ( ) `"ButtonClick"`
- ( ) `"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**

- [x] Custom hooks.

Correct
Correct, that's one possible abstraction.

- [x] Render props pattern.

Correct
Correct, that's one possible abstraction.

- [ ] Components that consume context.
- [x] Higher order components.

Correct
Correct, that's one possible abstraction.

9. What does the screen utility object from react-testing-library represent when performing queries against it? **1 / 1 point**

- ( ) Your laptop screen
- ( ) The whole virtual DOM
- (●) The whole page or root document

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`
- ( ) `toHaveBeenCalled`
- ( ) `toBeInTheDocument`

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