

Started on	Tuesday, 5 August 2025, 4:54 PM
State	Finished
Completed on	Tuesday, 5 August 2025, 5:06 PM
Time taken	11 mins 56 secs
Marks	9.00/16.00
Grade	56.25 out of 100.00

Question 1

Complete

Mark 1.00 out of 1.00

How do you pass props to a functional component?

```
const MyComponent = ({name}) => {  
  return <div>{name}</div>;  
};
```

- ☐ a. By using useState.
- ☐ b. By using setProps.
- ☐ c. By using the this.props syntax.
- ☒ d. By passing them as function arguments.

Question 2

Complete

Mark 1.00 out of 1.00

How would you conditionally render a component in React?

```
const MyComponent = () => {  
  const isLoggedIn = true;  
  return (  
    <div>  
      {isLoggedIn ? <p>Welcome, User!</p> : <p>Please Log In</p>}  
    </div>  
  );  
};
```

- ☐ a. The component will display both messages.
- ☐ b. The message will display 'Please Log In' regardless of the condition.
- ☒ c. The message will display 'Welcome, User!' because isLoggedIn is true.
- ☐ d. The component will throw an error due to improper JSX usage.

Question 3

Complete

Mark 1.00 out of 1.00

What is the correct usage of useEffect to log a message when the component mounts?

```
const MyComponent = () => {  
  useEffect(() => {  
    console.log('Component mounted');  
  }, []);  
  return <div>Hello</div>;  
};
```

- ☐ a. The message will not log because the useEffect hook is incorrect.
- ☐ b. The message will log every time the component renders.
- ☐ c. The message will log every time the component's state updates.
- ☒ d. The message will log once, when the component is first mounted.

Question 4

Complete

Mark 0.00 out of 1.00

What is the output of the following code?

```
const Parent = () => {  
  const [counter, setCounter] = useState(0);  
  return (  
    <div>  
      <Child counter={counter} />  
      <button onClick={() => setCounter(counter + 1)}>Increment</button>  
    </div>  
  );  
};  
const Child = React.memo(({ counter }) => {  
const Child = React.memo(({ counter }) => {  
  console.log('Rendering Child');  
  return <div>{counter}</div>;  
}
```

- ☐ a. "Rendering Child" will never be printed.
- ☐ b. "Rendering Child" will be printed on every re-render of Parent.
- ☐ c. "Rendering Child" will be printed only the first time the component is rendered.
- ☒ d. "Rendering Child" will be printed every time the button is clicked.

Question 5

Complete

Mark 0.00 out of 1.00

What is the output of the following code?

```
const MyComponent = () => {  
  const [count, setCount] = useState(0);  
  const memoizedCallback = useMemo(() => () => setCount(count + 1), [count]);  
  return <button onClick={memoizedCallback}>{count}</button>;  
};
```

- ☒ a. The application will crash because of improper useMemo usage.
- ☐ b. The count will increment correctly but in an inefficient way.
- ☐ c. The button will never update its count because the increment function will not re-render.
- ☐ d. The button text will increment correctly when clicked.

Question 6

Complete

Mark 0.00 out of 1.00

What is the output of the following code?

```
const MyComponent = () => {  
  const [name, setName] = useState('John');  
  useEffect(() => {  
    setName('Doe');  
  }, [name]);  
  return <div>{name}</div>;  
};
```

- ☒ a. The name will always stay 'John'.
- ☐ b. The component will result in an infinite loop of re-renders.
- ☐ c. The component will display 'John' initially and 'Doe' afterwards.
- ☐ d. The component will display an empty string.

Question 7

Complete

Mark 0.00 out of 1.00

What is the output of the following code?

```
const MyComponent = () => {  
  const [counter, setCounter] = useState(0);  
  const increment = useMemo(() => () => setCounter(counter + 1), []);  
  return <button onClick={increment}>{counter}</button>;  
};
```

- ☐ a. The button will display '0' but will not update after the first render.
- ☒ b. The button will throw an error.
- ☐ c. The button will display '0' and increment correctly on click.
- ☐ d. The button will display 'NaN'.

Question 8

Complete

Mark 1.00 out of 1.00

What is the result of this code snippet?

```
const MyComponent = () => {
  const [items, setItems] = useState(['Apple', 'Banana']);
  const handleAddItem = (item) => setItems([...items, item]);
  return (
    <div>
      <button onClick={() => handleAddItem('Orange')}>Add Orange</button>
      <ul>{items.map(item => <li key={item}>{item}</li>)}</ul>
    </div>
  );
};
```

- ☐ a. The 'Orange' item will be added, but it won't be rendered correctly.
- ☐ b. The list will be empty even after the button is clicked.
- ☒ c. The 'Orange' item will be added and displayed in the list when the button is clicked.
- ☐ d. The button will throw an error because setItems is improperly used.

Question 9

Complete

Mark 1.00 out of 1.00

What will be the output of the following code?

```
const MyComponent = () => {
  const [count, setCount] = useState(0);
  const increment = () => setCount(count + 1);
  return <button onClick={increment}>{count}</button>;
}
```

- ☐ a. An infinite loop will occur.
- ☐ b. `0` will always be displayed.
- ☐ c. The button text will change from 0 to 1, but won't increment further.
- ☒ d. The button text will keep incrementing when clicked.

Question 10

Complete

Mark 1.00 out of 1.00

What will be the result of the following code?

```
const Parent = () => {
  const [state, setState] = useState({ name: 'Alice', age: 25 });
  const changeName = () => setState((prevState) => ({ ...prevState, name: 'Bob' }));
  return <button onClick={changeName}>{state.name}</button>;
};
```

- ☒ a. The button text will change from 'Alice' to 'Bob' when clicked.
- ☐ b. The button will throw an error due to object immutability.
- ☐ c. The button will never render anything.
- ☐ d. The name will remain 'Alice' even after clicking the button.

Question 11

Complete

Mark 0.00 out of 1.00

What will happen when you call `useEffect` with an empty dependency array?

```
useEffect(() => {  
  console.log('Effect runs only once');  
}, []);
```

- ☐ a. The effect will run on every render.
- ☒ b. The effect will run after the first render and every time the state changes.
- ☐ c. The effect will run only once.
- ☐ d. The effect will run only when the component unmounts.

Question 12

Complete

Mark 1.00 out of 1.00

What will the following code output?

```
const MyComponent = () => {  
  const [count, setCount] = useState(0);  
  useEffect(() => {  
    setCount(count + 1);  
  }, [count]);  
  return <div>{count}</div>;  
};
```

- ☐ a. The value will always stay at 0.
- ☐ b. The value will increment correctly every time the component re-renders.
- ☒ c. An infinite loop will occur.
- ☐ d. The value will never update due to a circular state update.

Question 13

Complete

Mark 0.00 out of 1.00

What will the following code output?

```
const MyComponent = () => {  
  const [state, setState] = useState({});  
  setState({...state, name: 'John'});  
  return <div>{state.name}</div>;  
};
```

- ☐ a. The component will throw an error.
- ☒ b. `state.name` will be 'John'.
- ☐ c. `state.name` will be 'undefined' because the state is overwritten.
- ☐ d. `state.name` will stay empty.

Question 14

Complete

Mark 0.00 out of 1.00

What will the following code render?

```
const MyComponent = () => {  
  const [count, setCount] = useState(0);  
  const increment = useCallback(() => setCount(count + 1), [count]);  
  return <button onClick={increment}>{count}</button>;  
};
```

- ☐ a. The counter will show stale values due to the closure.
- ☒ b. The button will not update after the first render.
- ☐ c. The button will render correctly, incrementing the counter.
- ☐ d. The application will crash due to a closure issue.

Question 15

Complete

Mark 1.00 out of 1.00

Which of the following hooks should you use for handling form inputs in a functional component?

```
const MyComponent = () => {  
  const [value, setValue] = useState('');  
  return <input value={value} onChange={(e) => setValue(e.target.value)} />;  
};
```

- ☒ a. useState
- ☐ b. useRef
- ☐ c. useEffect
- ☐ d. useMemo

Question 16

Complete

Mark 1.00 out of 1.00

Which statement is true about React.StrictMode?

- ☒ a. It performs an extra render to detect potential problems.
- ☐ b. It enables hooks automatically.
- ☐ c. It allows for performance optimization.
- ☐ d. It reduces the size of the React bundle.