Lesson:

Expressions and Statements in JSX

PW SKILLS

Topics to be covered

1. Expressions in JSX
2. Expressions in JSX
3. Pitfalls and Common Errors
4. Conclusion

In JSX, you can embed JavaScript expressions and statements directly within the markup. This allows you to dynamically generate content, apply conditional rendering, iterate over lists, and perform other complex logic within your JSX code. Understanding how to use expressions and statements in JSX is crucial for building dynamic and interactive React components.

Expressions in JSX

Expressions in JSX are JavaScript code snippets that evaluate to a value. You can use expressions inside curly braces {} to embed them within JSX tags or attributes. JSX expressions can be variables, function calls, arithmetic operations, or any valid JavaScript expression.

Here are a few examples of using expressions in JSX:

```
import React from 'react';
import { View, Text } from 'react-native';

const App = () => {
  const name = 'John Doe';
  const age = 25;
  const greeting = `Hello, ${name}! You are ${age} years old.`;

  return (
    <View>
      <Text style={{ fontSize: 24 }}>{greeting}</Text>
      <Text>The result of 5 + 3 is {5 + 3}.</Text>
    </View>
  );
};

export default App;
```

In the above example, we use expressions within the JSX tags and attribute values. The **name** and **age** variables are interpolated into the **greeting** expression, and the result of the arithmetic operation **5 + 3** is displayed in the paragraph.

Statements in JSX

JSX doesn't directly support statements like **if**, **for**, or **while**. However, you can use JavaScript's ternary operator **condition ? expression1 : expression2** to conditionally render JSX components based on a condition.

Here's an example of using a ternary operator to conditionally render JSX:

```jsx
import React from 'react';
import { View, Text } from 'react-native';

const App = () => {
  const isLoggedIn = true;

  return (
    <View>
      {isLoggedIn ? <Text>Welcome, user!</Text> : <Text>Please log in.</Text>}
    </View>
  );
};

export default App;
```

In the above example, the JSX expression **{isLoggedIn ? Welcome, user! : Please log in. }** conditionally renders the appropriate message based on the value of the **isLoggedIn** variable.

Pitfalls and Common Errors

While working with JSX expressions, it's important to be aware of some common pitfalls:

1. Forgetting to use Curly Braces

Remember to use curly braces **{}** to enclose expressions in JSX. Without curly braces, the content will be treated as a string literal instead of an expression.

```jsx
// Incorrect
return <View><Text>Counter: counter</Text></View>;

// Correct
return <View><Text>Counter: {counter}</Text></View>;
```

2. Using Undefined or Null Values

If an expression evaluates to undefined or null, it won't render anything. Ensure that the expressions used in JSX have valid values to avoid unexpected behavior.

```jsx
// Incorrect
return <View><Text>{undefined}</Text></View>;

// Correct
return <View><Text>{validValue}</Text</View>;
```

3. Missing Parentheses for Multiple Expressions
When using multiple expressions within a JSX tag or attribute, make sure to wrap them in parentheses () to avoid syntax errors.

```
// Incorrect
return <View><Text>{expression1} {expression2}</Text></View>;

// Correct
return <View><Text>{(expression1, expression2)}</Text></View>;
```

Conclusion
Expressions and statements in JSX allow you to bring dynamic behavior to your React components. You can embed JavaScript expressions within curly braces **{}** to generate dynamic content, apply conditional rendering, and perform complex logic.

Remember to use curly braces **{}** for expressions and use the ternary operator for conditional rendering. Avoid common pitfalls like missing curly braces, undefined or null values, and missing parentheses for multiple expressions.