

# Lesson:

# Touchable Opacity



The `TouchableOpacity` component in React Native provides a touchable area that reduces the opacity of its content when pressed. It is a wrapper component that enables touch interactions for its child components. The `TouchableOpacity` component is useful when you want to add interactive elements to your app that provide visual feedback upon being pressed.

## Using the TouchableOpacity Component

To use the `TouchableOpacity` component, wrap your content inside it and provide an `onPress` callback function to handle the press event. Here's an example:

```
1 import React from 'react';
2 import { View, TouchableOpacity, Text, StyleSheet } from 'react-native';
3
4 const App = () => {
5   const handlePress = () => {
6     console.log('Button Pressed!');
7   };
8
9   return (
10     <View style={styles.container}>
11       <TouchableOpacity
12         style={styles.button}
13         onPress={handlePress}
14       >
15         <Text style={styles.buttonText}>Press Me</Text>
16       </TouchableOpacity>
17     </View>
18   );
19 };
20
21 const styles = StyleSheet.create({
22   container: {
23     flex: 1,
24     justifyContent: 'center',
25     alignItems: 'center',
26   },
27   button: {
28     backgroundColor: '#F2F2F2',
29     padding: 10,
30     borderRadius: 8,
31   },
32   buttonText: {
33     fontSize: 16,
34     color: '#333333',
35   },
36 });
37
38 export default App;
```

In this example, we have a simple button implemented using the `TouchableOpacity` component. The `TouchableOpacity` wraps the text "Press Me", and the `onPress` prop is assigned to the `handlePress` callback function, which logs a message to the console when the button is pressed.

The `TouchableOpacity` component reduces the opacity of its content when pressed, providing a visual feedback to indicate the interaction. This opacity effect is automatically applied by the component.

## Props of TouchableOpacity

The **`TouchableOpacity`** component accepts several props that allow you to customize its behavior and appearance. Here are the commonly used props with their default values:

- **`onPress`** (function, required): Specifies the callback function to be called when the component is pressed.
- **`activeOpacity`** (number): Determines the opacity of the button when it is pressed. The default value is 0.2.
- **`style` (object)**: Defines the style for the container of the `TouchableOpacity`. It allows you to specify dimensions, positioning, and other styling properties.
- **`disabled` (boolean)**: Disables the button if set to true. The default value is false.

## Use Cases

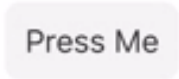
The **`TouchableOpacity`** component is useful in various scenarios where you want to add interactivity to your app's user interface. Here are some common use cases for using `TouchableOpacity`:

- **Buttons**: Create interactive buttons that change their appearance upon being pressed, providing visual feedback to the user.
- **Links**: Implement clickable links or text elements that provide a visual indication of being pressed.
- **Cards and tiles**: Add touchable areas within cards or tiles to enable actions like expanding or collapsing the content.
- **Interactive images**: Allow users to interact with images by making them touchable, such as image galleries or product thumbnails.

By utilizing the **`TouchableOpacity`** component in these scenarios, you can enhance the user experience by making your app more interactive and responsive to user input.

## Note on Platform-specific Styling

Both `TouchableOpacity` and `TouchableHighlight` automatically apply platform-specific styling. On iOS, they use the iOS default highlighting effect, while on Android, they use a ripple effect. This ensures that the components blend seamlessly with the native look and feel of the platform, providing a consistent user experience.

 Press Me