The scope of a view modifier is the view to which it is applied. This means that a view modifier will only affect the view to which it is applied and its descendant views.

For example, the following code applies the `.foregroundColor()` modifier to a Text view:

```
Swift
Text("Hello, world!")
    .foregroundColor(.red)
```

This will change the text color of the Text view to red. However, the text color of any descendant views will not be affected.

To change the text color of all descendant views, you can use the `.modifier()` method on the parent view. For example, the following code applies the `.foregroundColor()` modifier to a VStack view and its descendant views:

```
Swift
VStack {
    Text("Hello, world!")
    Text("This is a descendant view.")
}
.foregroundColor(.red)
```

This will change the text color of the VStack view and its descendant views to red.

View modifiers can also be applied to multiple views at once. For example, the following code applies the `.foregroundColor()` modifier to two Text views:

```
Swift
Text("Hello, world!")
    .foregroundColor(.red)

Text("This is another view.")
    .foregroundColor(.blue)
```

This will change the text color of the first Text view to red and the text color of the second Text view to blue.

View modifiers are a powerful tool for customizing the look and feel of your SwiftUI views. By understanding the scope of view modifiers, you can use them to create complex and beautiful user interfaces.

Here are some additional tips for using view modifiers:

- You can use the `.modifier()` method to apply the same view modifier to multiple views at once.
- You can chain multiple view modifiers together to create more complex effects.
- You can create your own custom view modifiers.
- You can use view modifiers to create animations and transitions.

Overall, view modifiers are a valuable tool for any SwiftUI developer. They allow you to create custom and unique views that can be used to build beautiful and engaging user interfaces.