The **private(set)** and **mutating** keywords in Swift can be used together to create properties that can only be set from within the scope in which they are defined, but can be mutated by methods within that scope.

This can be useful for encapsulating the state of an object and preventing unauthorized changes. For example, the following code defines a struct with a **private(set)** property called **value** and a **mutating** method called **increment()**:

Swift

```swift
struct Counter {
    private(set) var value = 0

    mutating func increment() {
        value += 1
    }
}
```

The **value** property can only be set from within the **Counter** struct, but it can be mutated by the **increment()** method. This means that the value of the **value** property can only be changed by calling the **increment()** method.

Here is an example of how to use the **private(set)** and **mutating** keywords together:

Swift

```swift
var counter = Counter()
print(counter.value) // Prints: 0

counter.increment()
print(counter.value) // Prints: 1

// Trying to set the value of the `value` property directly will not work.
// counter.value = 2 // Error: 'value' is immutable
```

The **private(set)** and **mutating** keywords can be a powerful combination for encapsulating the state of an object and preventing unauthorized changes.

Here are some additional benefits of using the **private(set)** and **mutating** keywords together in Swift:

- **Security:** The *private(set)* keyword can help to prevent unauthorized changes to the

state of an object.
- **Modularity:** The *private(set)* and *mutating* keywords can help to make code more modular and easier to maintain.
- **Readability:** The *private(set)* and *mutating* keywords can make code more readable and easier to understand.

Overall, the **private(set)** and **mutating** keywords are a valuable combination that can help you to write better code.