A computed property in Swift is a property whose value is derived from other properties. Computed properties are calculated on demand, rather than storing their values. This makes them useful for properties that are difficult or impossible to store directly, such as the age of a user or the total price of a shopping cart.

To define a computed property, you use the `var` or `let` keyword, followed by the property name and a getter closure. The getter closure is a block of code that is executed to calculate the value of the property.

For example, the following code defines a computed property called `fullName` that returns the full name of a person:

```Swift
struct Person {
    var firstName: String
    var lastName: String

    var fullName: String {
        return "\(firstName) \(lastName)"
    }
}
```

To use the `fullName` property, you would simply access it like any other property:

```Swift
let person = Person(firstName: "John", lastName: "Doe")

print(person.fullName) // Prints "John Doe"
```

Computed properties can also have setters, which allows you to change their values. However, it is important to note that setters for computed properties are not always possible. For example, if a computed property relies on external data, such as the current time or the state of a database, then it may not be possible to set its value directly.

Here is an example of a computed property with a setter:

```Swift
class Rectangle {
    var width: Int
    var height: Int
```

```
    var area: Int {
        get {
            return width * height
        }
        set {
            width = newValue / height
        }
    }
}
```

In this example, the `area` property calculates the area of the rectangle based on its `width` and `height` properties. The `area` property also has a setter, which allows you to set the area of the rectangle by specifying a new value.

Computed properties are a powerful feature of Swift that allows you to create properties that are derived from other data. This can make your code more concise and expressive, and it can also help you to avoid storing redundant data.