ADVANCED TOPICS IN SWIFT

James Blair — feature[23]

A BIT OF HISTORY

- Modern alternative to Objective-C
- Created by Chris Lattner and Apple
  - Begin work in 2010
  - Reached 1.0 release Summer 2014
- Took inspiration from many different languages

“[Swift] greatly benefited from the experiences hard-won by many other languages in the field, drawing ideas from Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list.”

– Chris Lattner

WHY SWIFT?

- A language that supports many paradigms
  - Object-oriented
  - Pure functional
  - Procedural
- Use higher-order functions with C-like performance
- **Safe**, unless you explicitly ask it not to be

At this point, @SwiftLang is probably a better, and more valuable, vehicle for learning functional programming than Haskell.

– Erik Meijer (Twitter), October 2015

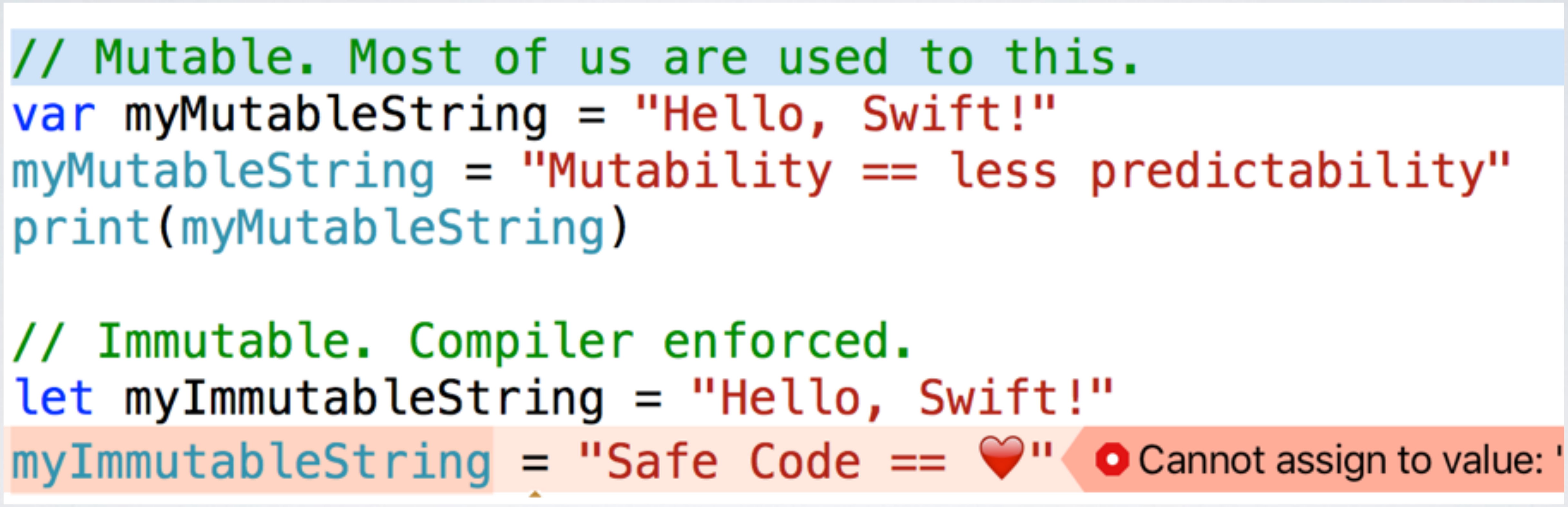
SOME KEY FEATURES

- Immutability, Value Types, and Copy Semantics
- Generics (Swift's implementation is a bit different than C# or Java)
- Protocols, Protocol Extensions, and "Traits"

SOME KEY FEATURES

- Enumeration Types (A fancy type of Tuple)
  - Associated Values / Pattern matching
- Optionals (A fancy type of Enumeration)
  - Optional Chaining / Safe unwrapping with `guard let`
- Property Observers

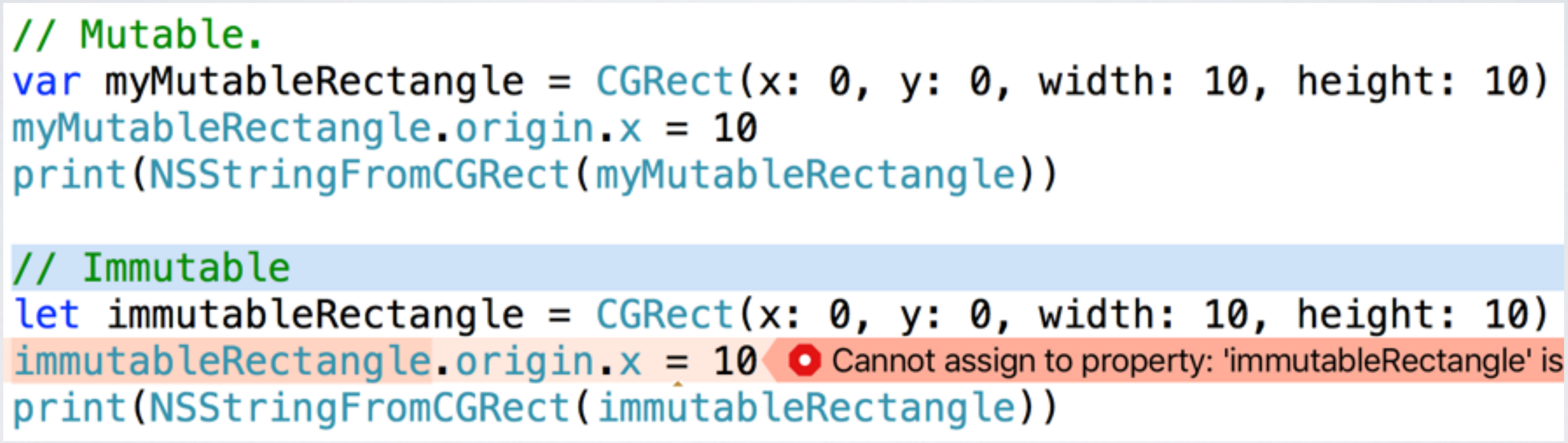
IMMUTABILITY



```
// Mutable. Most of us are used to this.
var myMutableString = "Hello, Swift!"
myMutableString = "Mutability == less predictability"
print(myMutableString)

// Immutable. Compiler enforced.
let myImmutableString = "Hello, Swift!"
myImmutableString = "Safe Code == ❤️"   Cannot assign to value: '
```

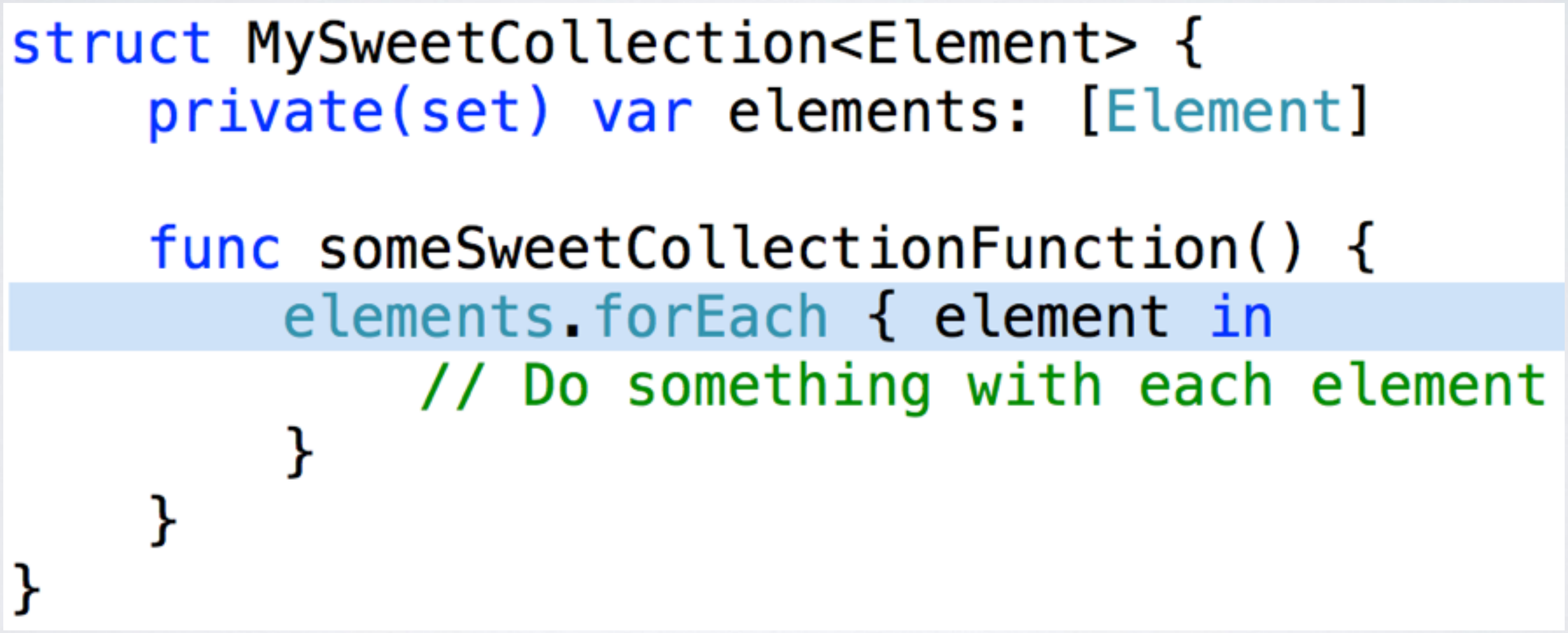
IMMUTABILITY

Works for Value-Type Members Too



```
// Mutable.
var myMutableRectangle = CGRect(x: 0, y: 0, width: 10, height: 10)
myMutableRectangle.origin.x = 10
print(NSStringFromCGRect(myMutableRectangle))

// Immutable
let immutableRectangle = CGRect(x: 0, y: 0, width: 10, height: 10)
immutableRectangle.origin.x = 10  Cannot assign to property: 'immutableRectangle' is
print(NSStringFromCGRect(immutableRectangle))
```

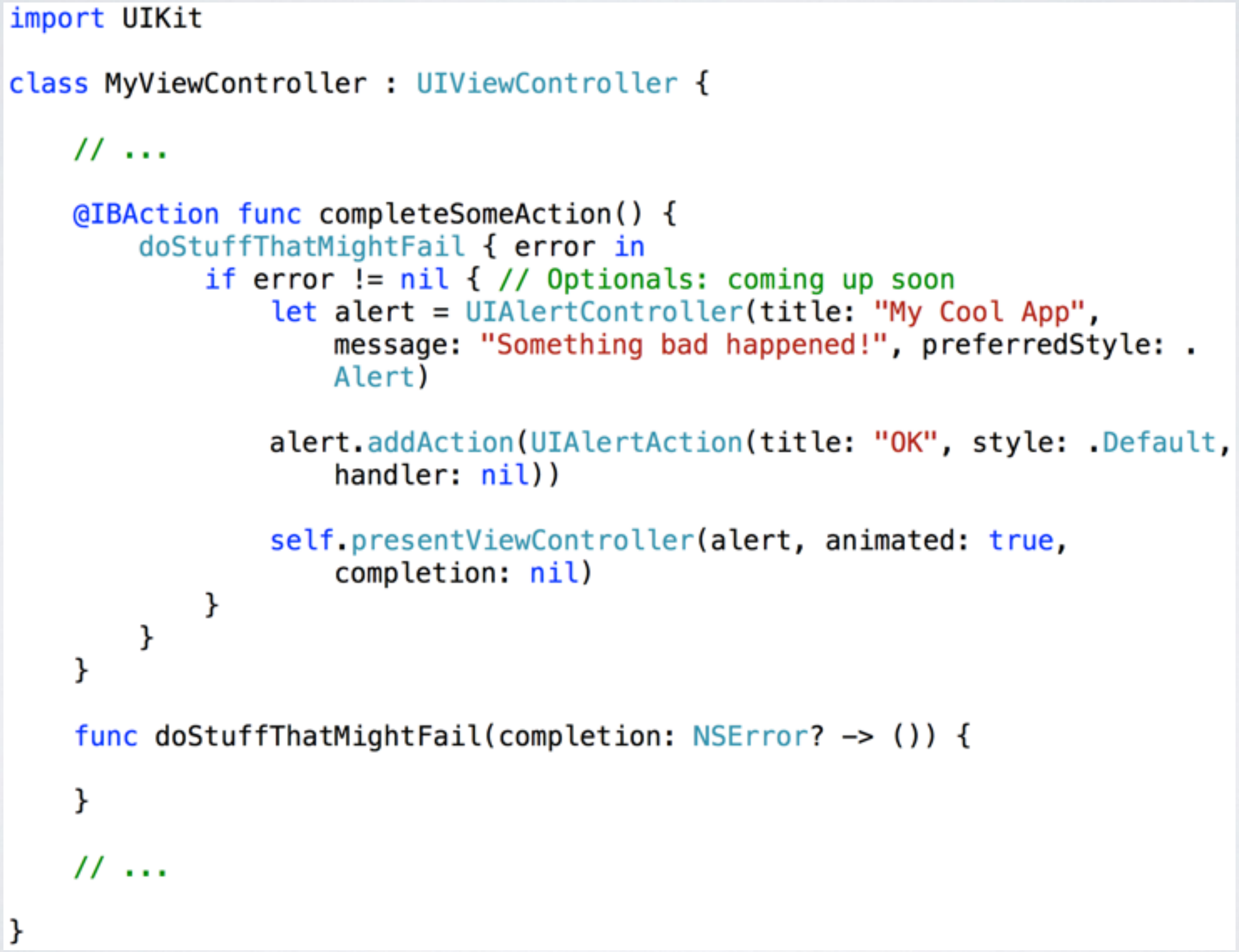
GENERICS



```swift
struct MySweetCollection<Element> {
    private(set) var elements: [Element]

    func someSweetCollectionFunction() {
        elements.forEach { element in
            // Do something with each element
        }
    }
}
```

PROTOCOLS

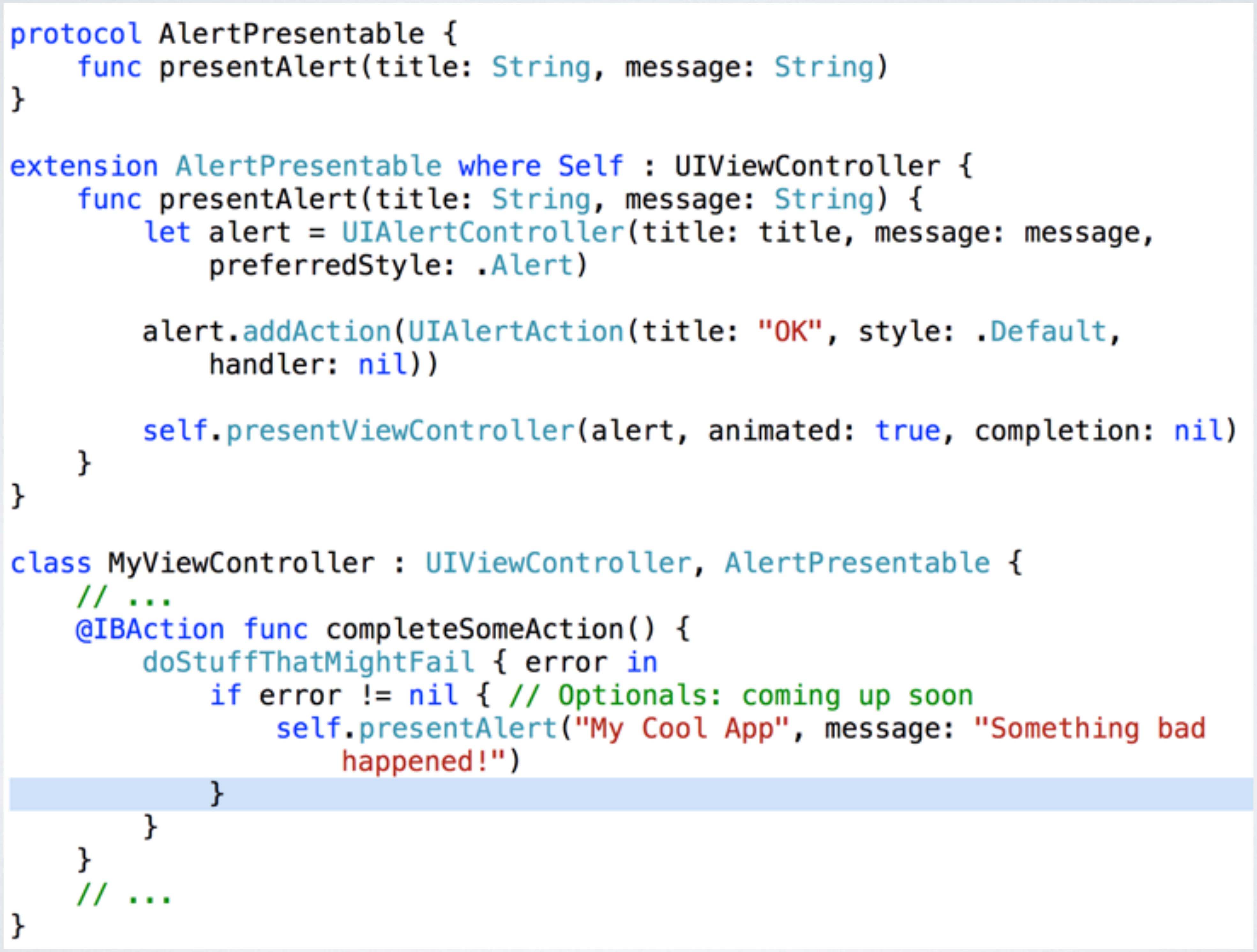
Extensions and "Traits"



```swift
import UIKit

class MyViewController : UIViewController {

    // ...

    @IBAction func completeSomeAction() {
        doStuffThatMightFail { error in
            if error != nil { // Optionals: coming up soon
                let alert = UIAlertController(title: "My Cool App",
                    message: "Something bad happened!", preferredStyle: .
                    Alert)

                alert.addAction(UIAlertAction(title: "OK", style: .Default,
                    handler: nil))

                self.presentViewController(alert, animated: true,
                    completion: nil)
            }
        }
    }

    func doStuffThatMightFail(completion: NSError? -> ()) {

    }

    // ...

}
```

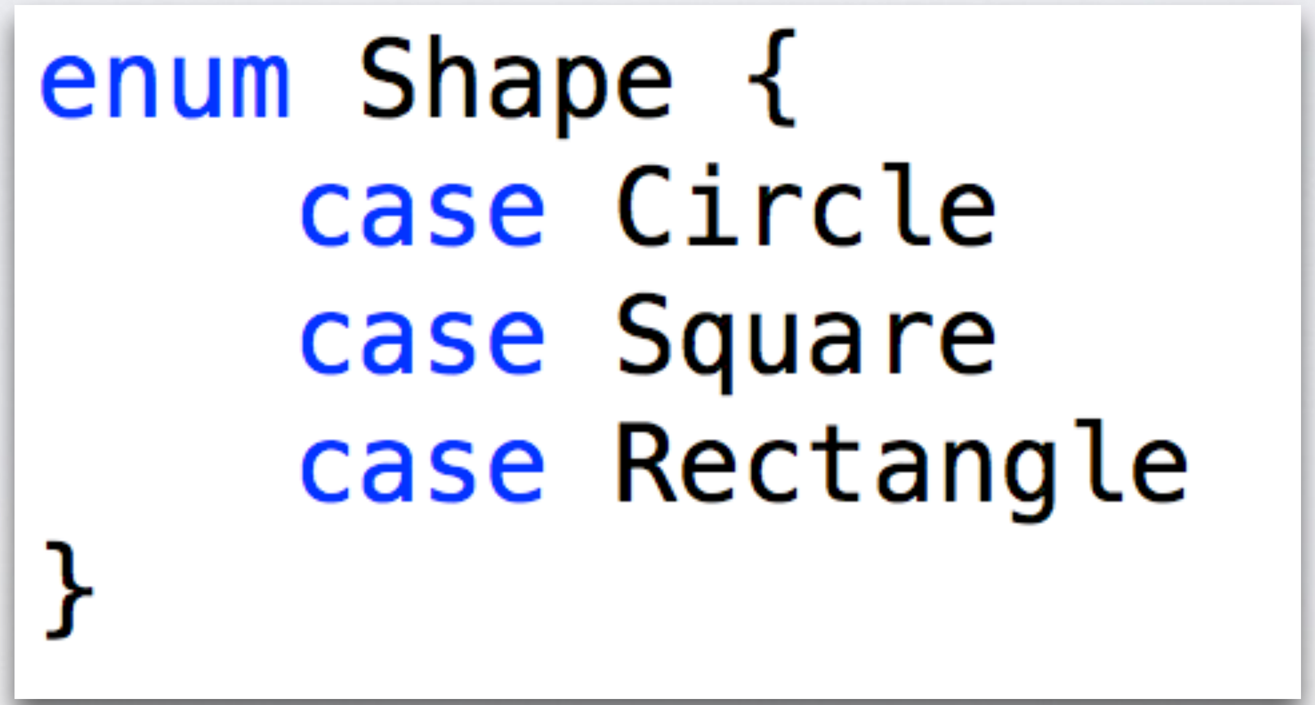
PROTOCOLS

Extensions and “Traits”



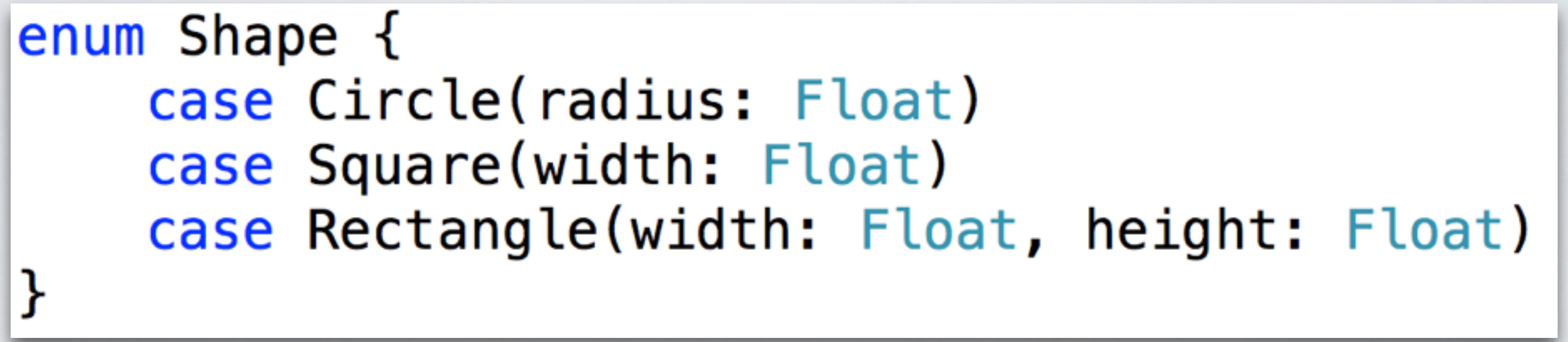
```swift
protocol AlertPresentable {
    func presentAlert(title: String, message: String)
}

extension AlertPresentable where Self : UIViewController {
    func presentAlert(title: String, message: String) {
        let alert = UIAlertController(title: title, message: message,
            preferredStyle: .Alert)

        alert.addAction(UIAlertAction(title: "OK", style: .Default,
            handler: nil))

        self.presentViewController(alert, animated: true, completion: nil)
    }
}

class MyViewController : UIViewController, AlertPresentable {
    // ...
    @IBAction func completeSomeAction() {
        doStuffThatMightFail { error in
            if error != nil { // Optionals: coming up soon
                self.presentAlert("My Cool App", message: "Something bad
                    happened!")
            }
        }
    }
    // ...
}
```

ENUMS


```
enum Shape {
    case Circle
    case Square
    case Rectangle
}
```

ENUMS

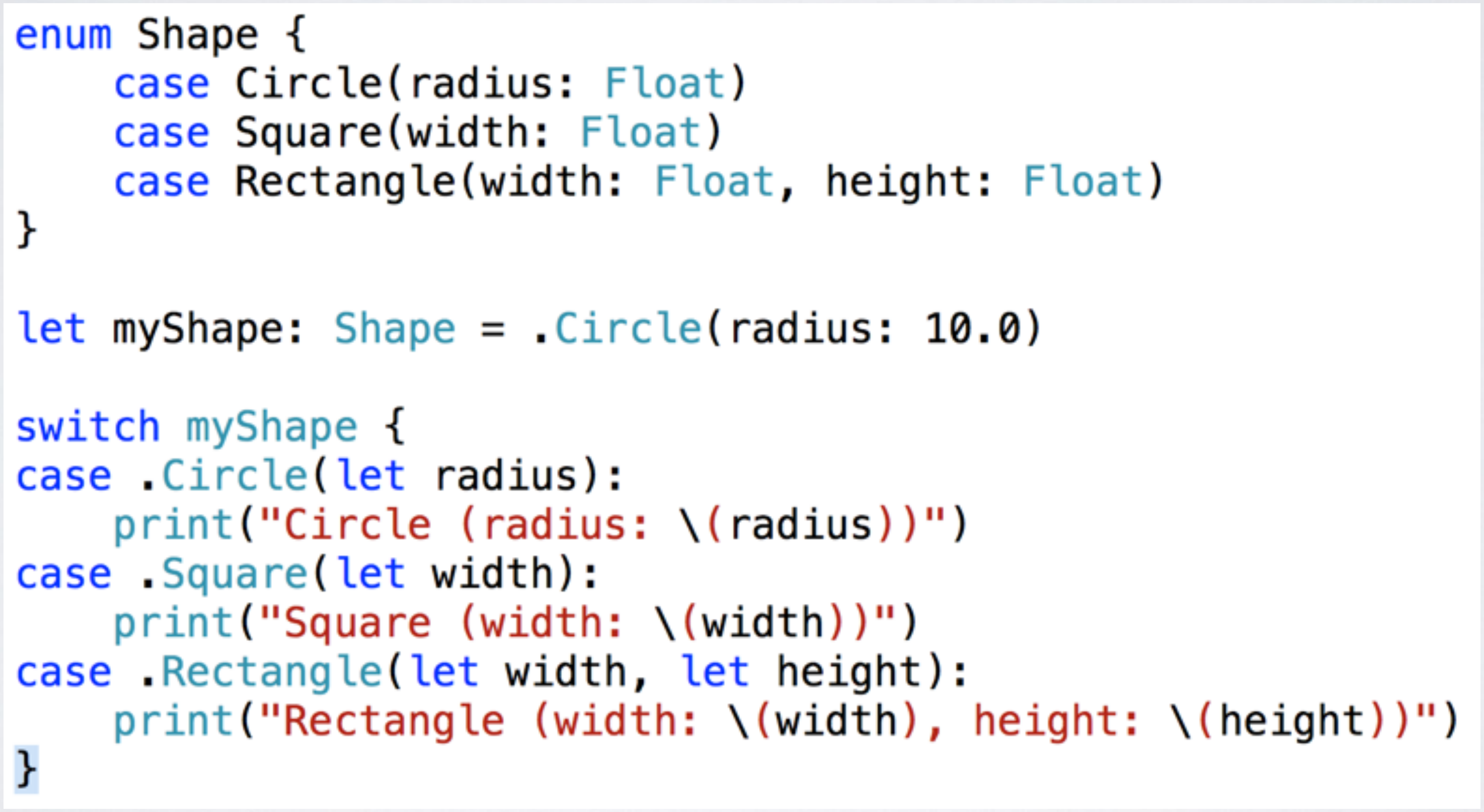
Associated Values



```
enum Shape {
    case Circle(radius: Float)
    case Square(width: Float)
    case Rectangle(width: Float, height: Float)
}
```

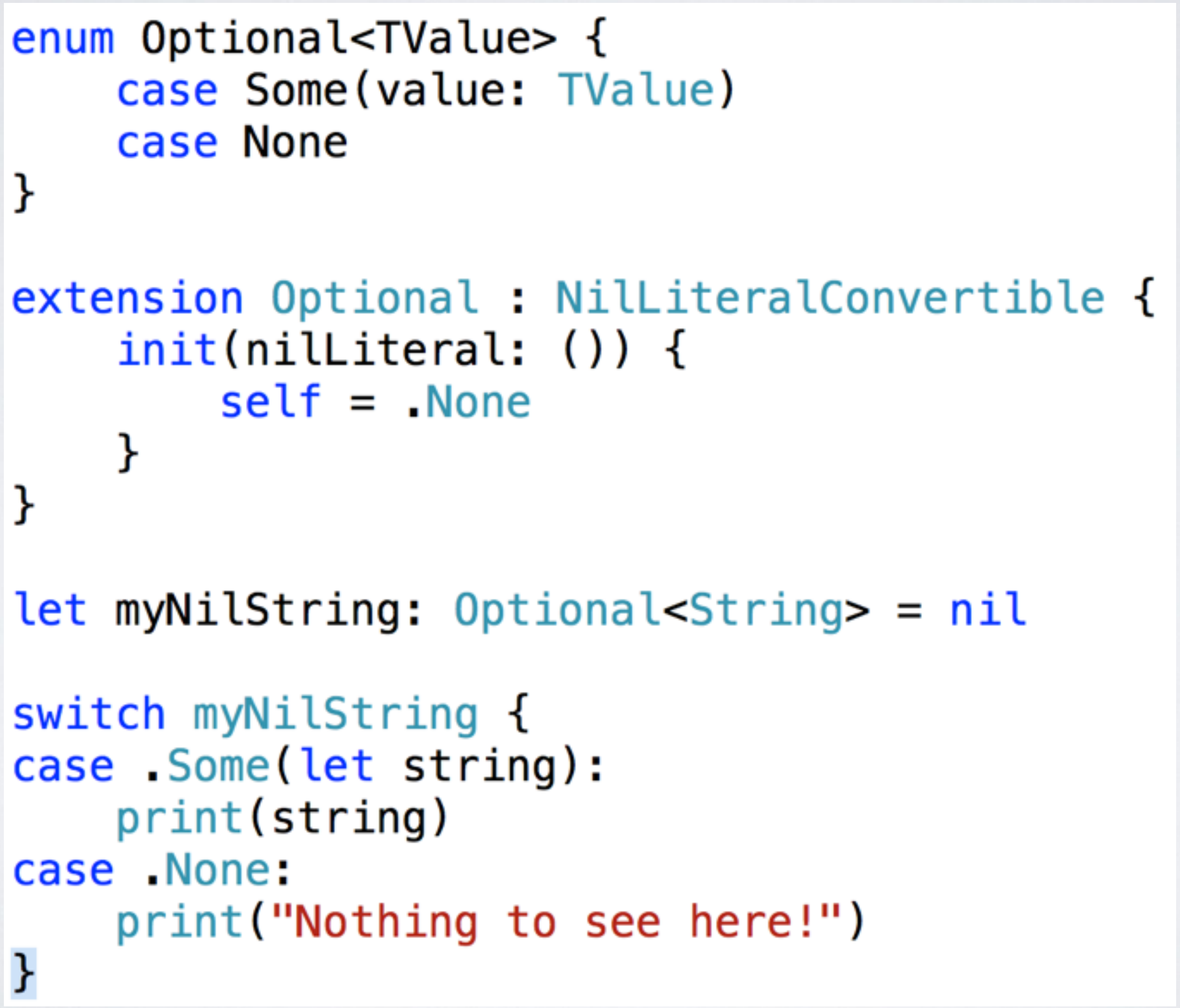
ENUMS

Pattern Matching



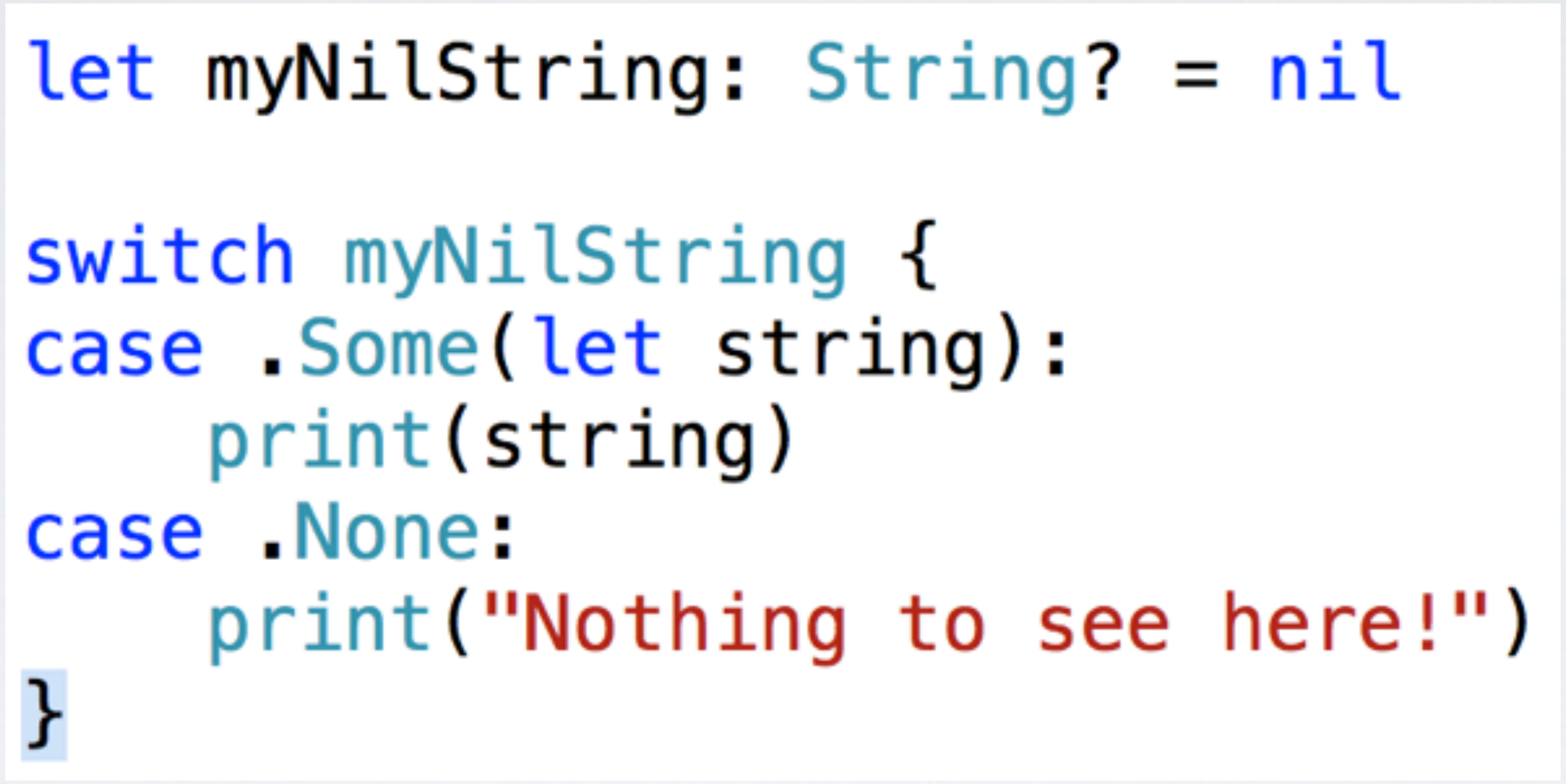
```swift
enum Shape {
    case Circle(radius: Float)
    case Square(width: Float)
    case Rectangle(width: Float, height: Float)
}

let myShape: Shape = .Circle(radius: 10.0)

switch myShape {
case .Circle(let radius):
    print("Circle (radius: \(radius))")
case .Square(let width):
    print("Square (width: \(width))")
case .Rectangle(let width, let height):
    print("Rectangle (width: \(width), height: \(height))")
}
```

OPTIONALS



```swift
enum Optional<TValue> {
    case Some(value: TValue)
    case None
}

extension Optional : NilLiteralConvertible {
    init(nilLiteral: ()) {
        self = .None
    }
}

let myNilString: Optional<String> = nil

switch myNilString {
case .Some(let string):
    print(string)
case .None:
    print("Nothing to see here!")
}
```

OPTIONALS

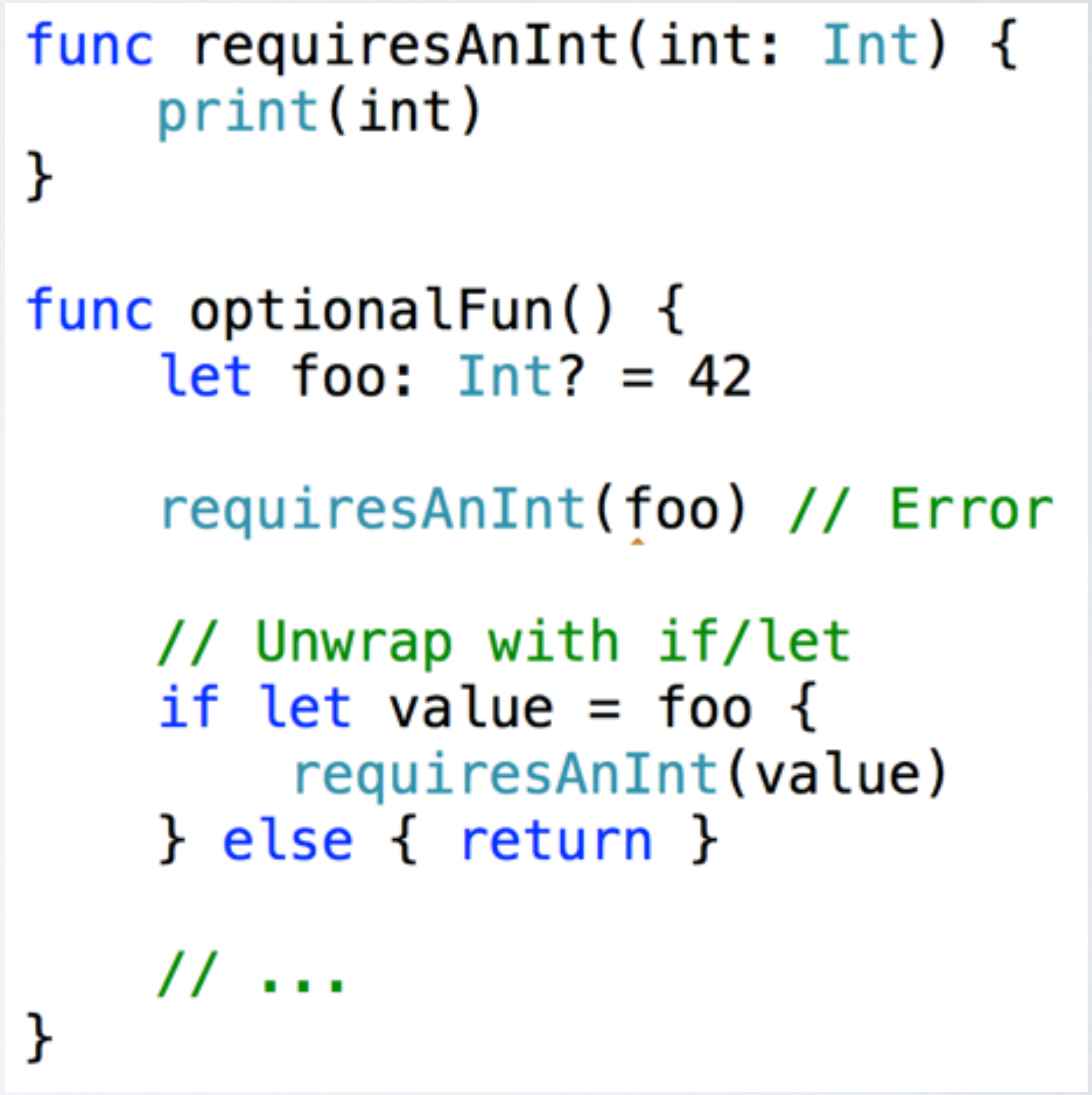


```
let myNilString: String? = nil

switch myNilString {
case .Some(let string):
    print(string)
case .None:
    print("Nothing to see here!")
}
```

OPTIONALS

Safe Unwrapping

- `if let`
- `guard let`
- `while let`
- `for let`

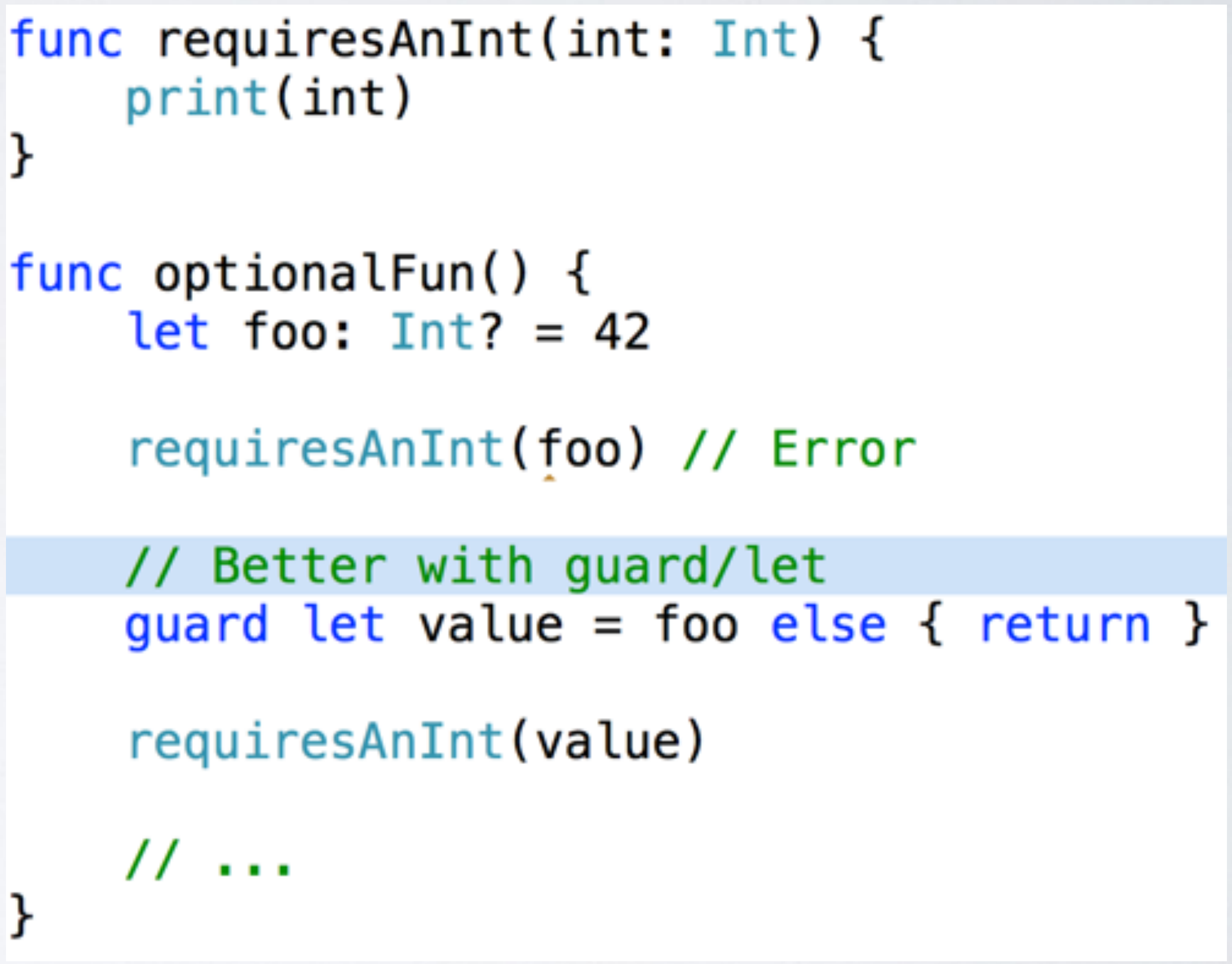


```swift
func requiresAnInt(int: Int) {
    print(int)
}

func optionalFun() {
    let foo: Int? = 42

    requiresAnInt(foo) // Error

    // Unwrap with if/let
    if let value = foo {
        requiresAnInt(value)
    } else { return }

    // ...
}
```

OPTIONALS

Safe Unwrapping

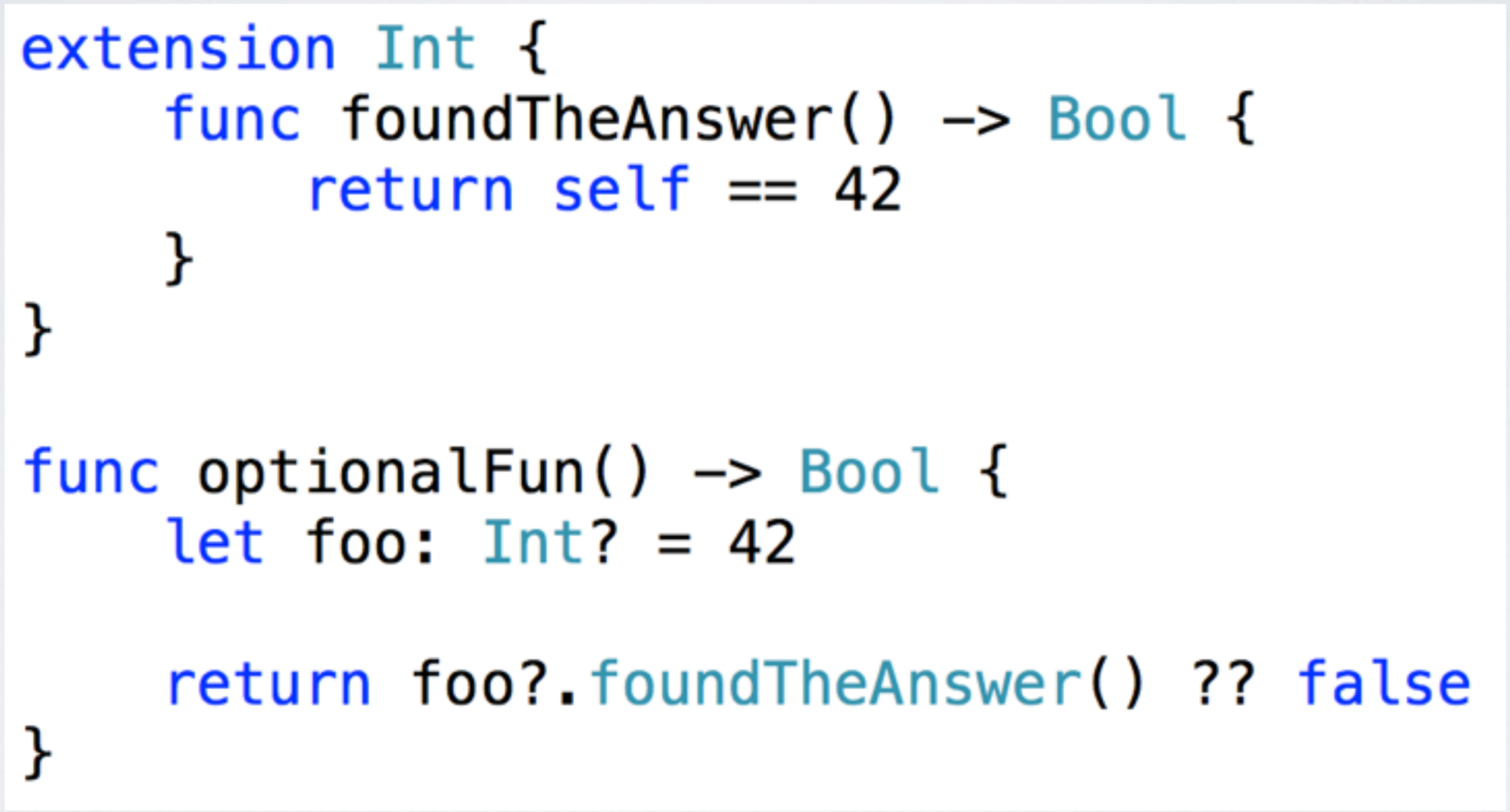
- `if let`
- `guard let`
- `while let`
- `for let`



```swift
func requiresAnInt(int: Int) {
    print(int)
}

func optionalFun() {
    let foo: Int? = 42

    requiresAnInt(foo) // Error

    // Better with guard/let
    guard let value = foo else { return }

    requiresAnInt(value)

    // ...
}
```

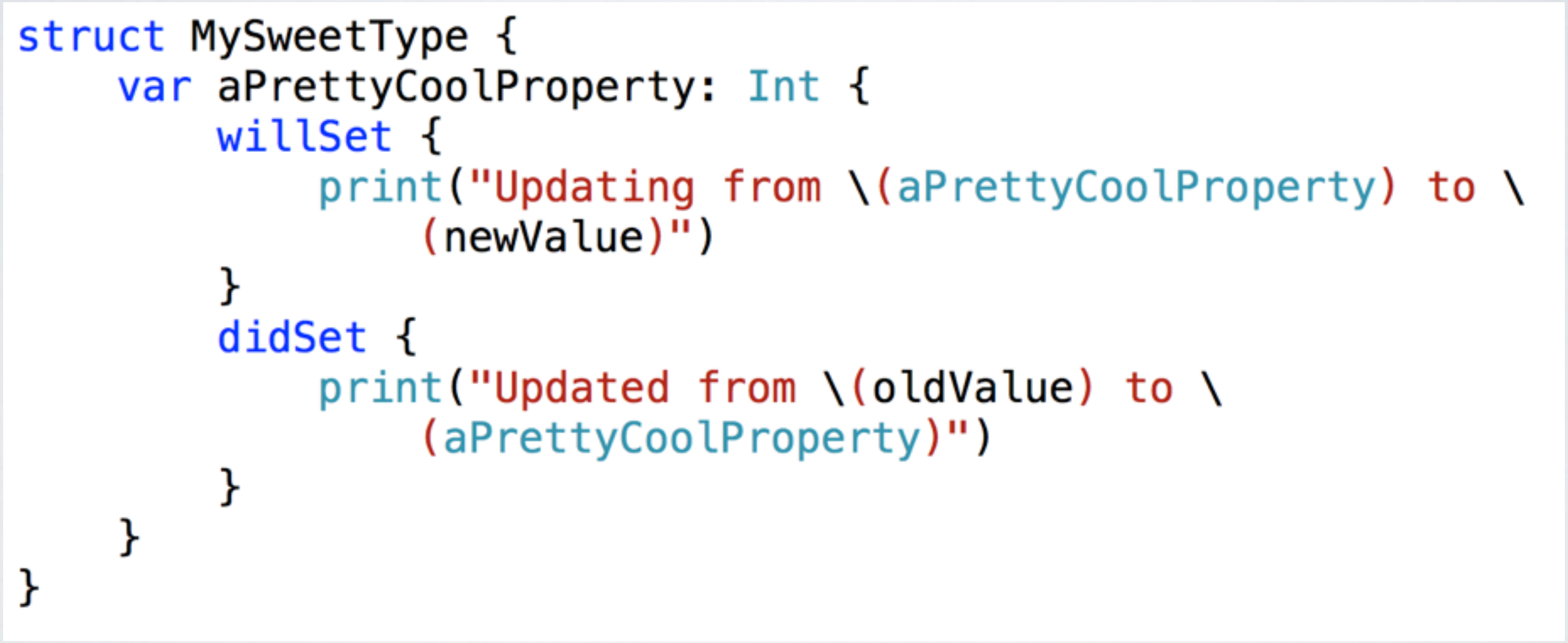
OPTIONALS

Optional Chaining



```
extension Int {
    func foundTheAnswer() -> Bool {
        return self == 42
    }
}

func optionalFun() -> Bool {
    let foo: Int? = 42

    return foo?.foundTheAnswer() ?? false
}
```

PROPERTY OBSERVERS



```swift
struct MySweetType {
    var aPrettyCoolProperty: Int {
        willSet {
            print("Updating from \(aPrettyCoolProperty) to \
                (newValue)")
        }
        didSet {
            print("Updated from \(oldValue) to \
                (aPrettyCoolProperty)")
        }
    }
}
```

Together, these language features can make your code much **safer**, more **deterministic**, and easier to **reason about**.

How can we use these features to improve a **real application**?

Let’s look at **MVC**

Let’s look at **MVC**

View **C**ontrollers

- MVC = **M**assive **V**iew **C**ontroller
- Modeling view state is **hard**
- How can we account for all possible states?
- How can we account for all possible state transitions?

DEMO

View Controllers

- All states are accounted for
- All transitions are accounted for
- Data are localized to each state

Let’s go further with **MVC**

Let's go further with **MVC**

View Layout

- Without IB / AutoLayout, view layout code can be large and complicated
- The math usually isn't difficult, but the code isn't always easy to read or maintain
- Views (or worse, controllers) become tightly coupled to a particular layout

DEMO

A Couple More Things

- Protocols and "Protocol-Oriented" Programming
- More ways to increase safety with enums

Protocol-Oriented Programming

- Provide default implementation of protocol methods
- If you're familiar with Scala, this looks like Traits
- Also looks a lot like multiple inheritance
  - But implementation determined at compile time
  - No v-table ambiguity and C++ style safety issues

Enums and Safety

- Enums are a great way to remove magic strings and numbers
- UIKit is riddled with magic strings (and sometimes numbers)
- prepareForSegue(segue: UIStoryboardSegue)
    - segue.identifier is an Optional<String> 😩

DEMO

References / Diving Deeper

1. Protocol and Value Oriented Programming in UIKit Apps (WWDC 2016, https://developer.apple.com/videos/play/wwdc2016/419/)

2. Improving Existing Apps with Modern Best Practices (WWDC 2016, https://developer.apple.com/videos/play/wwdc2016/213/)

3. Advanced Swift (Book, https://www.objc.io/books/advanced-swift/)

4. The Swift Programming Language (eBook, https://itunes.apple.com/us/book/swift-programming-language/id881256329)

https://github.com/jamesmblair/AdvancedTopicsInSwift

Questions?