Swift & iOS 8

liaojinxing

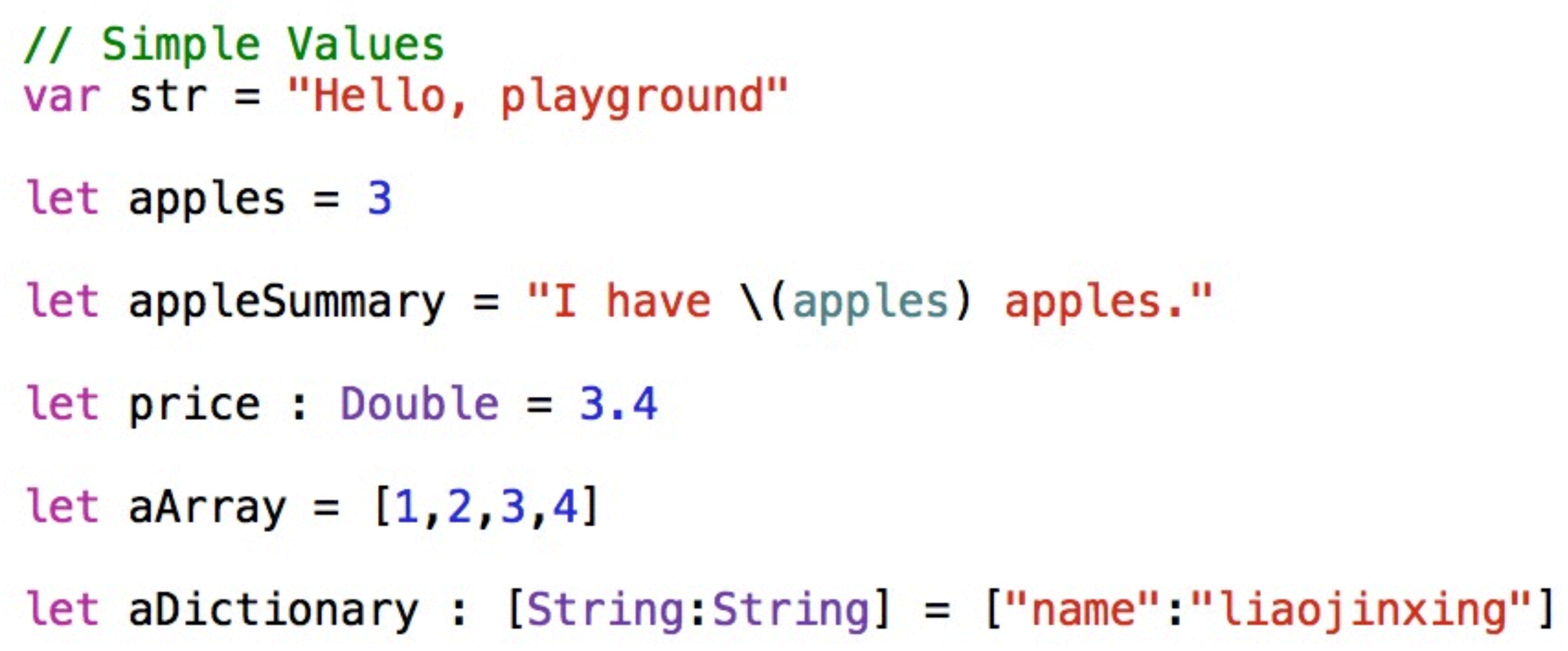
About Swift

- New language for iOS&OSX apps development.
- Easier, more flexible, more fun
- Backed by Cocoa and Cocoa Touch framework
- Industrial-quality systems programming language that is as expressive and enjoyable as a scripting language

Types

- Int, Double, Float, String, Bool, Tuple
- Collection types: Array, Dictionary
- var & let
- Optional values

Types



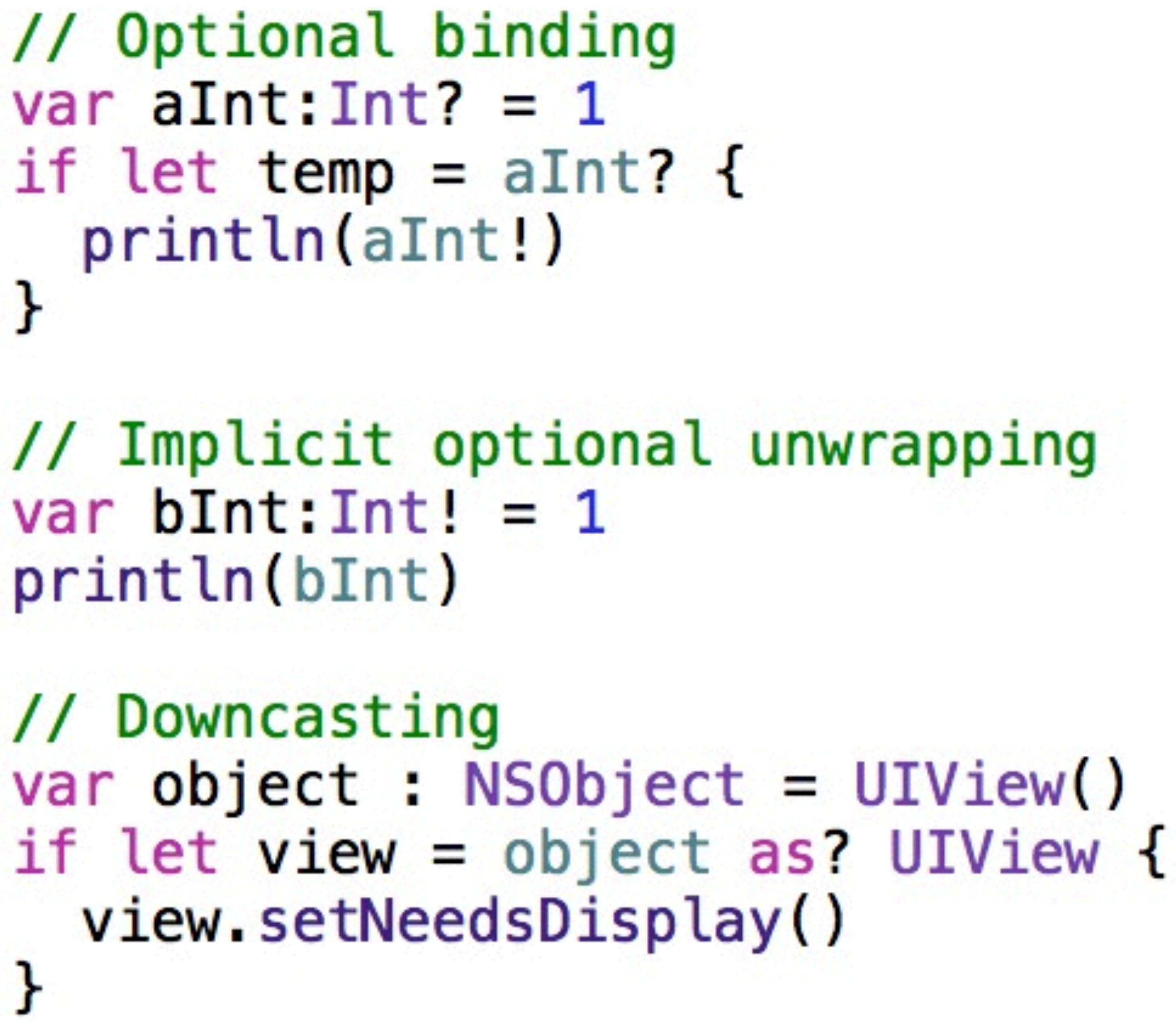
```swift
// Simple Values
var str = "Hello, playground"

let apples = 3

let appleSummary = "I have \(apples) apples."

let price : Double = 3.4

let aArray = [1,2,3,4]

let aDictionary : [String:String] = ["name":"liaojinxing"]
```

- Type safety, type checking, type inference

Optional values

- Forced unwrapping, Implicit optional unwrapping
- Optional binding
- Optional chaining
- Downcasting

Optional values



```swift
// Optional binding
var aInt:Int? = 1
if let temp = aInt? {
  println(aInt!)
}

// Implicit optional unwrapping
var bInt:Int! = 1
println(bInt)

// Downcasting
var object : NSObject = UIView()
if let view = object as? UIView {
  view.setNeedsDisplay()
}
```

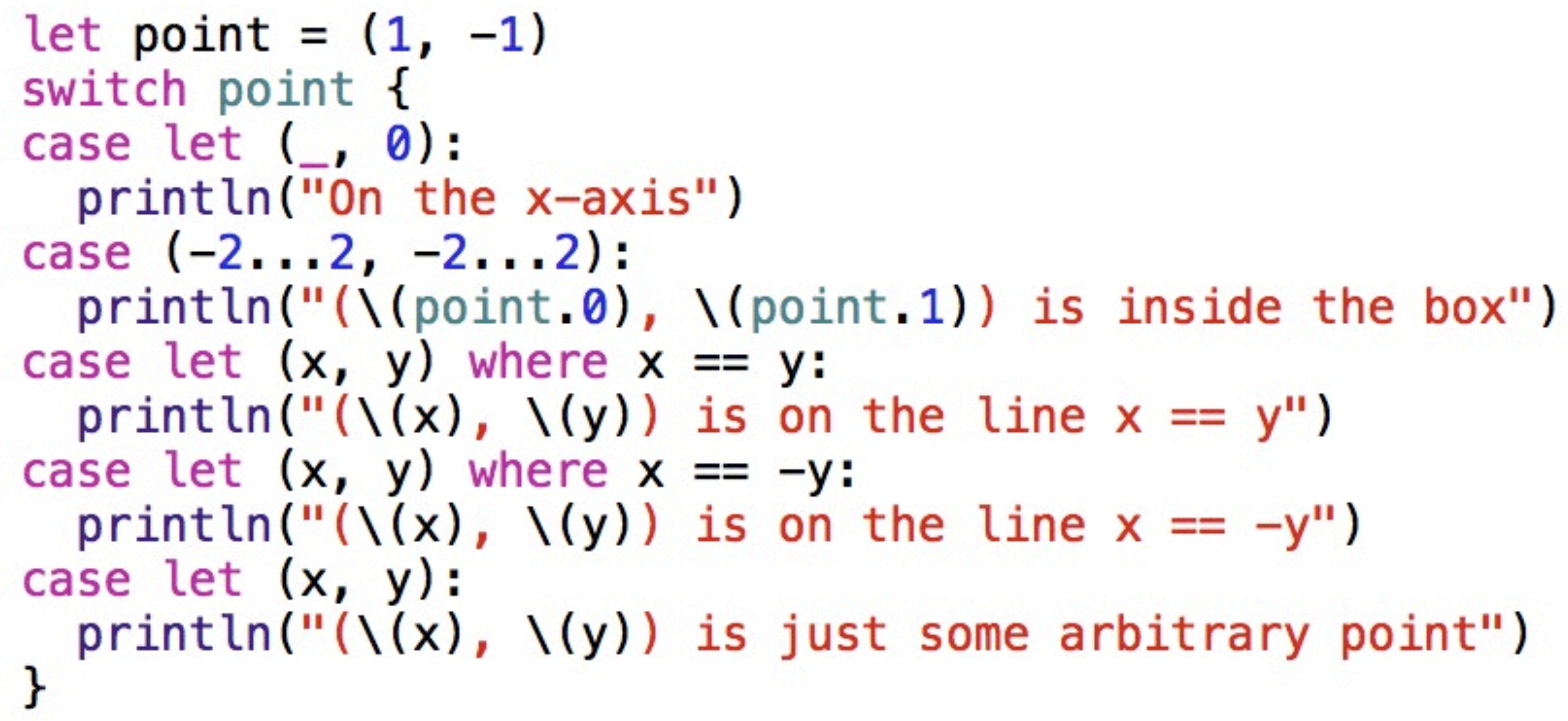
Optional chaining

```
if let beginsWithThe =
    john.residence?.address?.buildingIdentifier()?.hasPrefix("The") {
        if beginsWithThe {
            println("John's building identifier begins with \"The\".")
        } else {
            println("John's building identifier does not begin with \"The\".")
        }
}
```

Control Flow

- if
- for, for-in(…&..<), while, do-while
- switch(No implicit fallthrough, range matching, value bindings, where clause)
- labeled statements

switch

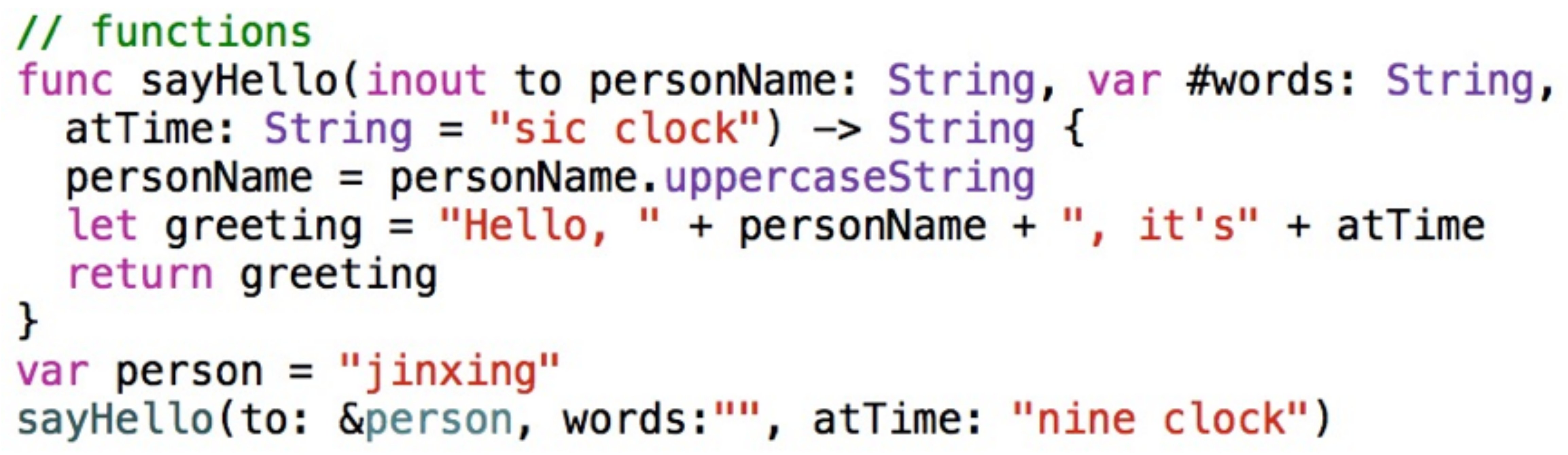


```swift
let point = (1, -1)
switch point {
case let (_, 0):
  println("On the x-axis")
case (-2...2, -2...2):
  println("(\(point.0), \(point.1)) is inside the box")
case let (x, y) where x == y:
  println("(\(x), \(y)) is on the line x == y")
case let (x, y) where x == -y:
  println("(\(x), \(y)) is on the line x == -y")
case let (x, y):
  println("(\(x), \(y)) is just some arbitrary point")
}
```

Functions

- Multiple parameters, multiple return values
- External parameter name
- Default parameter value
- Variadic Parameters
- Variable Parameters
- In-out parameters
- Function types as parameters types or return types
- Nested functions

Functions

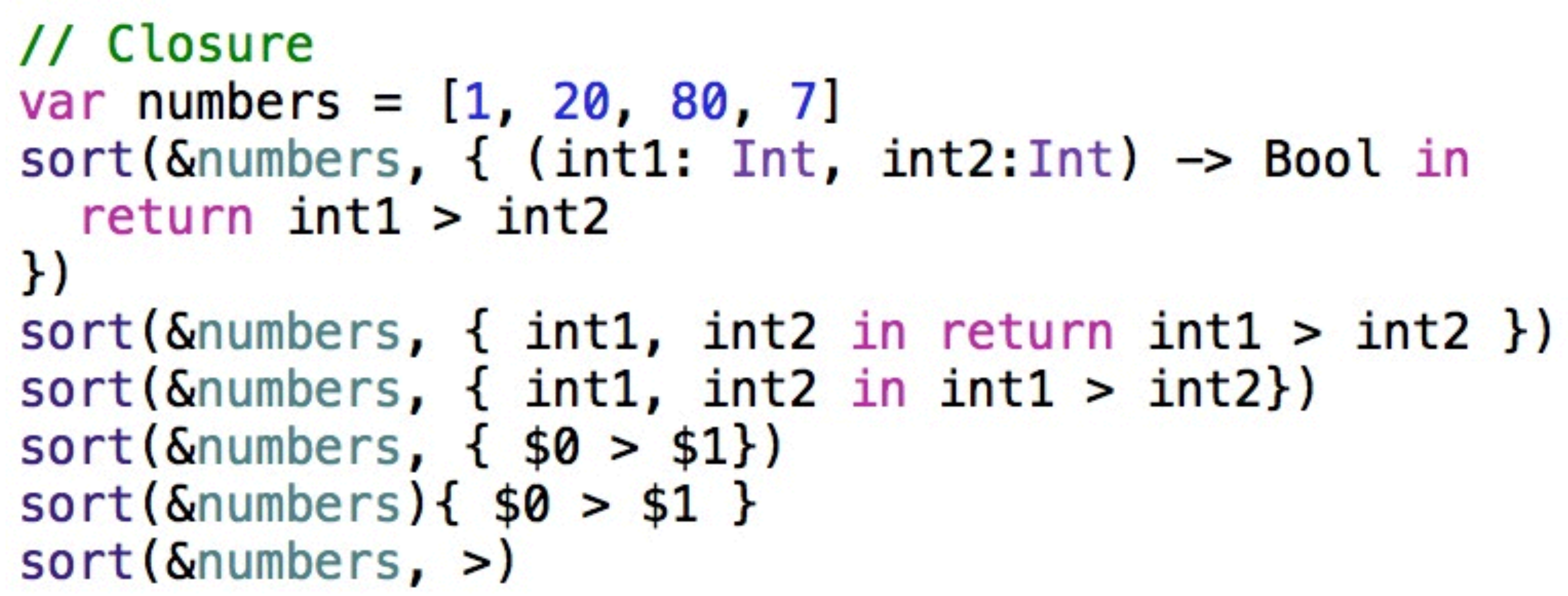


```
// functions
func sayHello(inout to personName: String, var #words: String,
  atTime: String = "sic clock") -> String {
  personName = personName.uppercaseString
  let greeting = "Hello, " + personName + ", it's" + atTime
  return greeting
}
var person = "jinxing"
sayHello(to: &person, words:"", atTime: "nine clock")
```

Closures

- Inferring parameter and return value types from context
- Implicit returns from single-expression closures
- Shorthand argument names
- Trailing closure
- Capturing values
- Functions and closures are reference types.

Closures

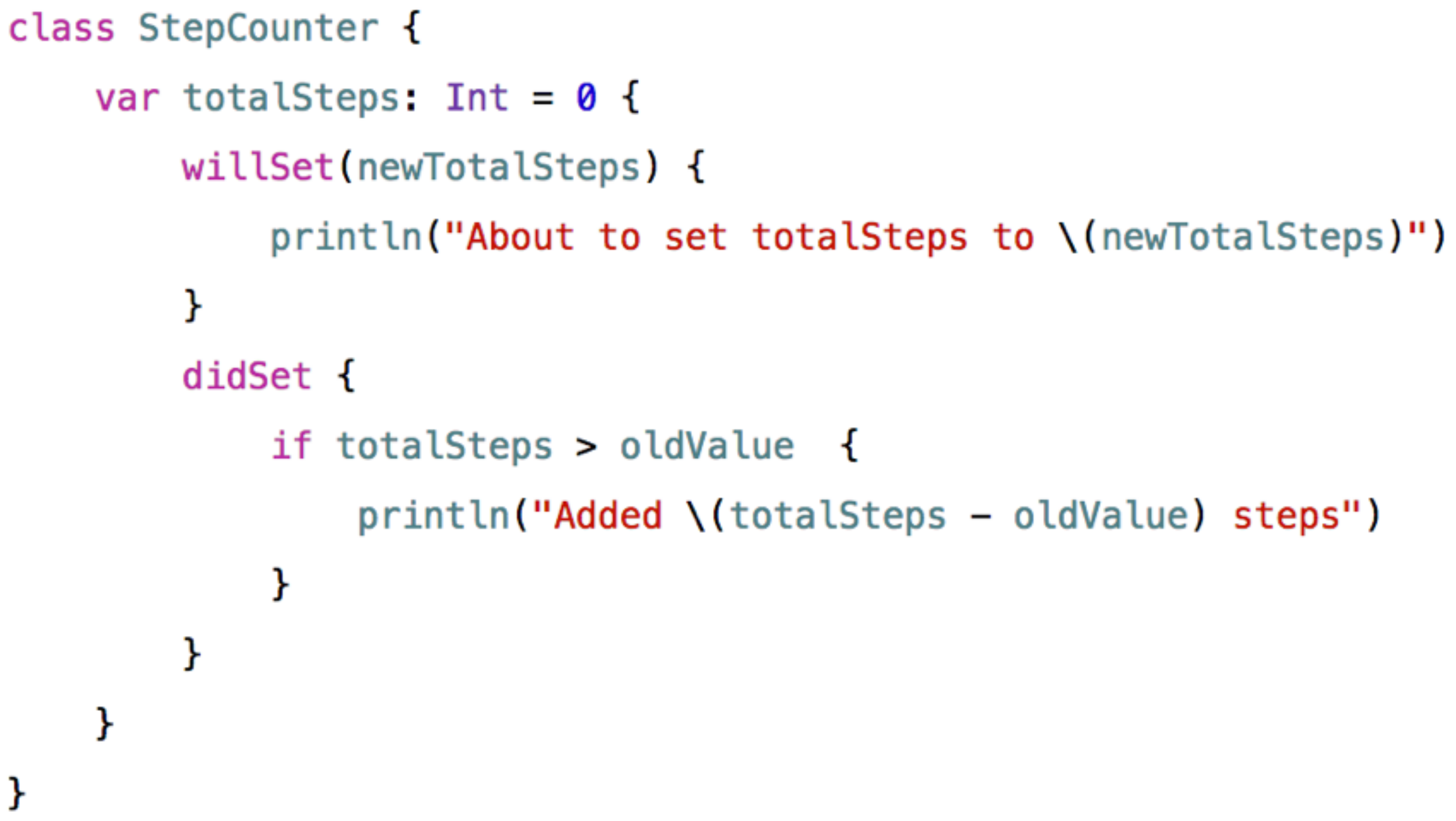


```
// Closure
var numbers = [1, 20, 80, 7]
sort(&numbers, { (int1: Int, int2:Int) -> Bool in
  return int1 > int2
})
sort(&numbers, { int1, int2 in return int1 > int2 })
sort(&numbers, { int1, int2 in int1 > int2})
sort(&numbers, { $0 > $1})
sort(&numbers){ $0 > $1 }
sort(&numbers, >)
```

Objects and Classes

- Structures are value types, classes are reference types.
- Stored properties and computed properties, property observer, type properties
- Instance methods and type methods(self)
- Subscripts

Property observer



```swift
class StepCounter {
    var totalSteps: Int = 0 {
        willSet(newTotalSteps) {
            println("About to set totalSteps to \(newTotalSteps)")
        }
        didSet {
            if totalSteps > oldValue  {
                println("Added \(totalSteps - oldValue) steps")
            }
        }
    }
}
```

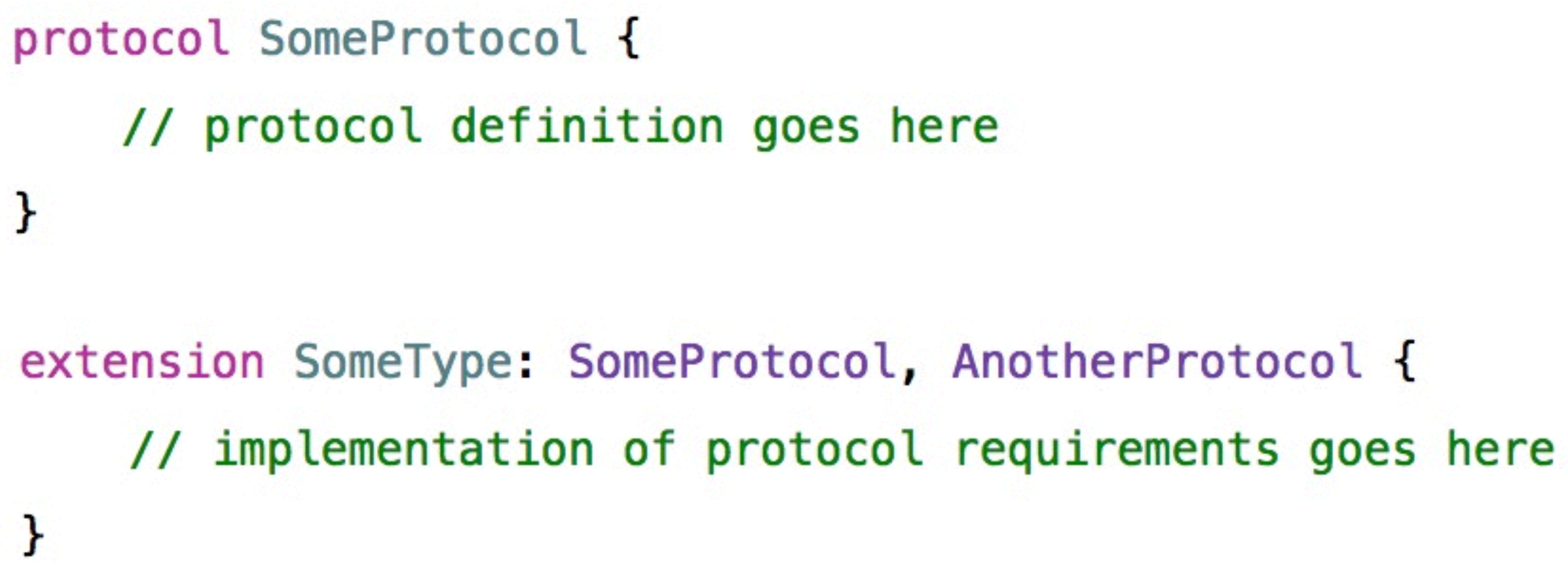
Subscript

```
subscript(index: Int) -> Int {
    get {
        // return an appropriate subscript value here
    }
    set(newValue) {
        // perform a suitable setting action here
    }
}
```

Initializer and Deinitializer

- Default initializer, memberwise initializer, initializer delegation
- Designated Initializers and Convenience Initializers
- Initializer chaining
- Two-phase initialization

Protocols & Extensions



```
protocol SomeProtocol {
    // protocol definition goes here
}

extension SomeType: SomeProtocol, AnotherProtocol {
    // implementation of protocol requirements goes here
}
```

- Unlike Objective-C categories, Swift extensions do not have names

Generics

- Generic functions & generic types
- Type constraints
- where clause

Generics

```
func allItemsMatch<
  C1: Container, C2: Container
  where C1.ItemType == C2.ItemType, C1.ItemType: Equatable>
  (someContainer: C1, anotherContainer: C2) -> Bool {

    if someContainer.count != anotherContainer.count {
      return false
    }
    for i in 0..<someContainer.count {
      if someContainer[i] != anotherContainer[i] {
        return false
      }
    }
    return true
}
```

Access control

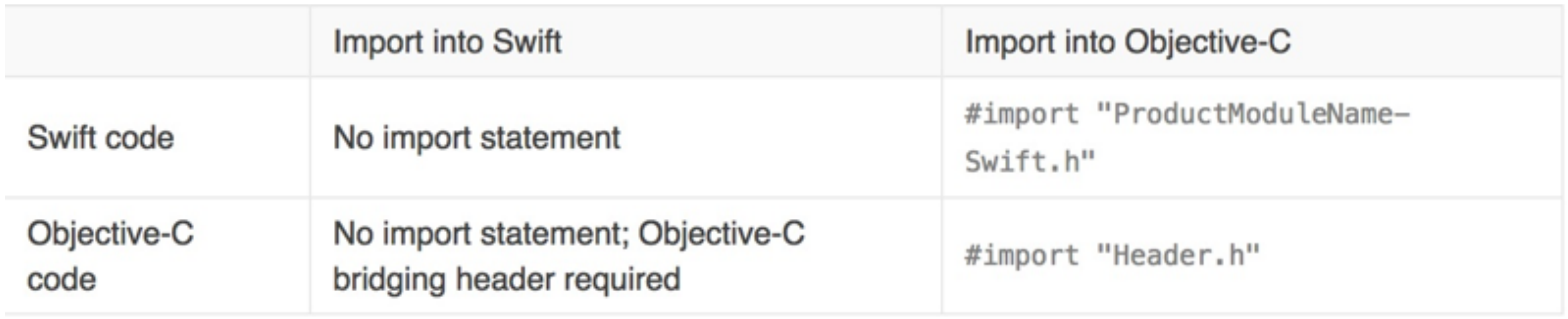
- public, internal, private. Default is internal.
- Tuple: the most restrictive access level of all types used in that tuple.
- Function: the most restrictive access level of the function’s parameter types and return type.
- Nested types and subclassing
- getter and setter
- Protocols

iOS with Swift

- Lister: A Productivity App

Using Swift with Cocoa&Objc

- @objc
- Swift-only features
- Mix and match



	Import into Swift	Import into Objective-C
Swift code	No import statement	`#import "ProductModuleName-Swift.h"`
Objective-C code	No import statement; Objective-C bridging header required	`#import "Header.h"`

iOS 8 for Developers

- 4,000 new APIs
- Amazing new features and capabilities
- Bold new technologies for game development

Add New Capabilities

- App Extensions
- Touch ID
- PhotoKit
- Manual Camera Controls
- HealthKit, HomeKit
- CloudKit
- Handoff

Games

- SceneKit
- SpriteKit
- Metal

TestFlight

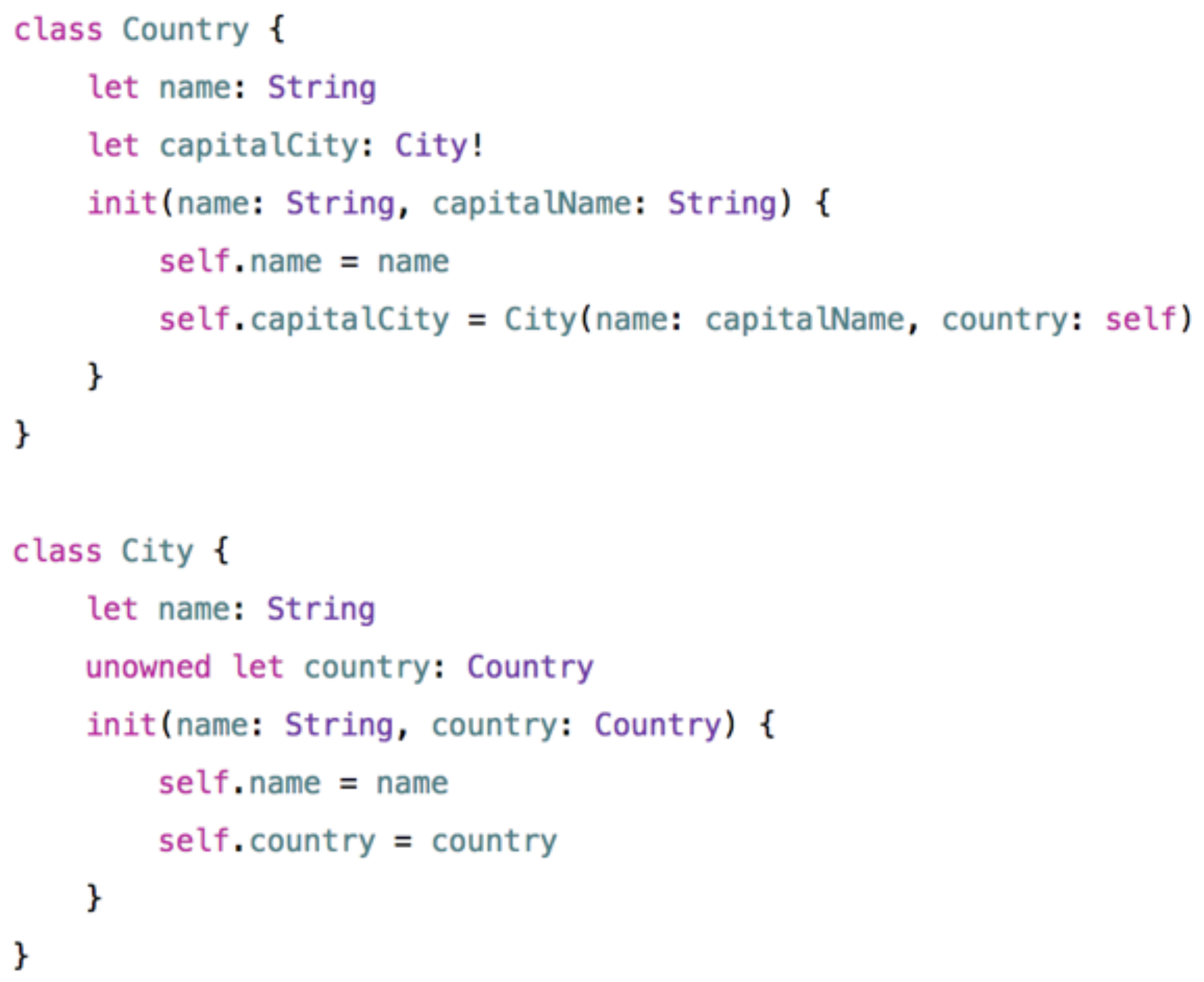
- Internal Testers: Up to 25 members * Up to 10 devices
- External Testers: Up to 1000 users
- Once a beta app is installed, TestFlight will notify testers each time a new build is available, provide instructions on where to focus, and offer an easy way to give feedback.

Q & A

Swift & ObjC

- Category vs Extension
- Closure vs block
- id vs AnyObject
- pointer vs optional
- Cocoa design patter

Implicit optional unwrapping



```
class Country {
    let name: String
    let capitalCity: City!
    init(name: String, capitalName: String) {
        self.name = name
        self.capitalCity = City(name: capitalName, country: self)
    }
}

class City {
    let name: String
    unowned let country: Country
    init(name: String, country: Country) {
        self.name = name
        self.country = country
    }
}
```