Swift 2.0

Anthony Mittaz
@nsfounder
github.com/sync

MARKETPLACER™
That's what we do

It’s funny…

Last week ‘Result’, ‘Decoded’ or ‘Either’ was the only sanctioned way for purist to handle errors in Swift

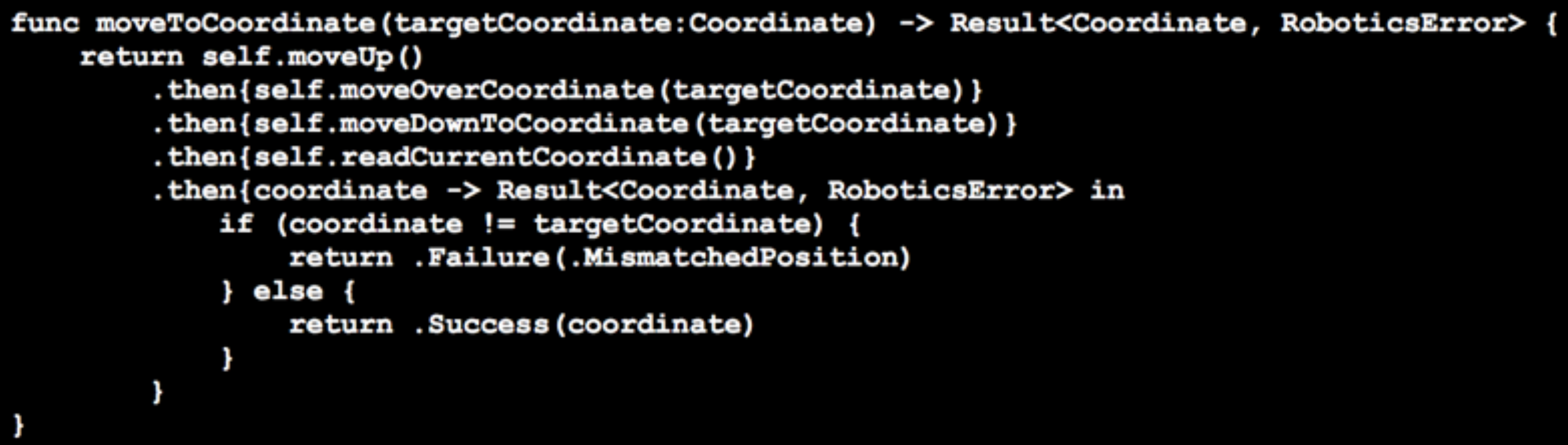
Result<Value, Error> values are either successful (wrapping Value) or failed (wrapping Error).

This is similar to Swift's native Optional type, with the addition of an error value to pass some error code, message, or object along to be logged or displayed to the user.

https://github.com/antitypical/Result

It had all the shiny

(then could have been map)



```
func moveToCoordinate(targetCoordinate:Coordinate) -> Result<Coordinate, RoboticsError> {
    return self.moveUp()
        .then{self.moveOverCoordinate(targetCoordinate)}
        .then{self.moveDownToCoordinate(targetCoordinate)}
        .then{self.readCurrentCoordinate()}
        .then{coordinate -> Result<Coordinate, RoboticsError> in
            if (coordinate != targetCoordinate) {
                return .Failure(.MismatchedPosition)
            } else {
                return .Success(coordinate)
            }
        }
}
```

And all the cool kids were using it

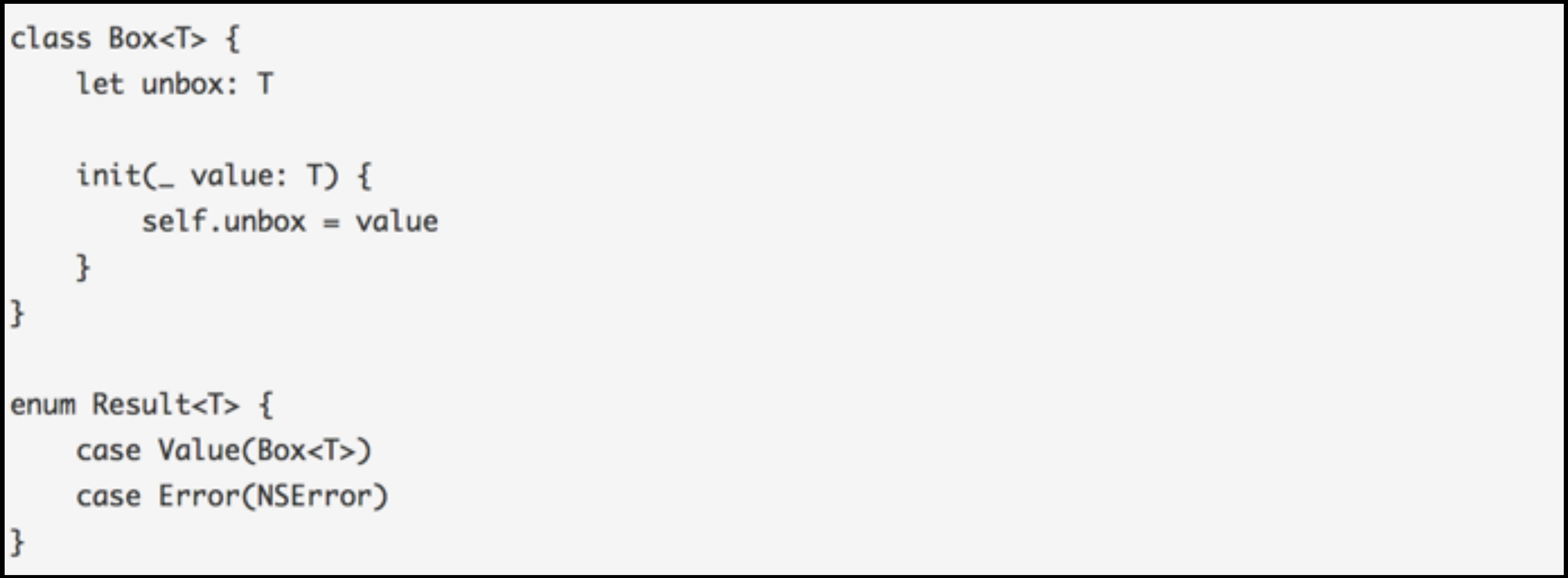
Carthage, Argo, ReactiveCocoa, Future, Realm, …

It came from Haskell

“So it’s cool man.”

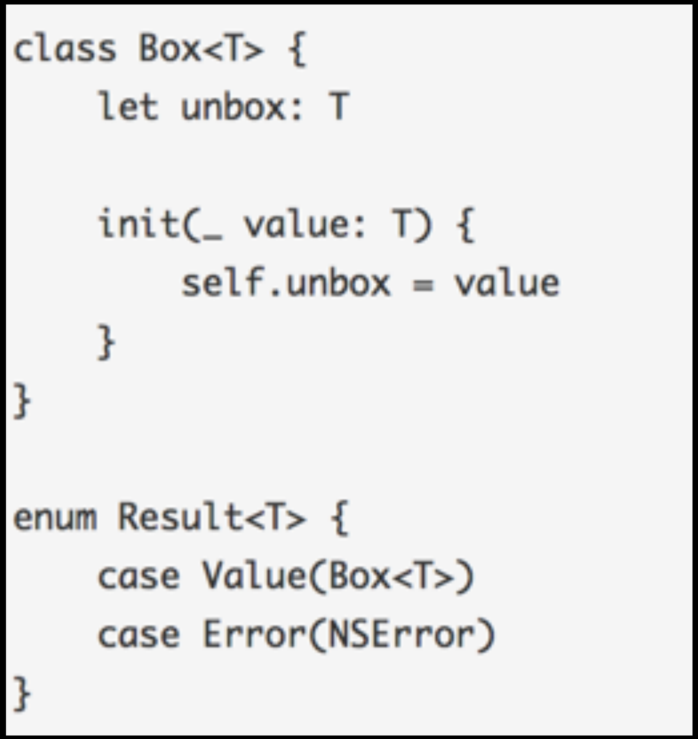
We had to ‘Box’ the 💩 out of it

to make it work



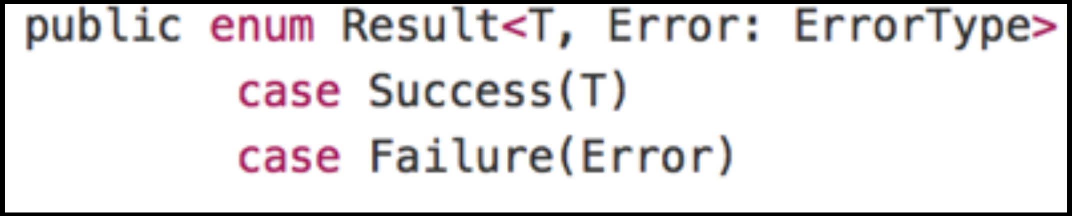
```
class Box<T> {
    let unbox: T

    init(_ value: T) {
        self.unbox = value
    }
}

enum Result<T> {
    case Value(Box<T>)
    case Error(NSError)
}
```

Thanks to Swift 2.0 we can achieve the same thing with less boilerplate



```
class Box<T> {
    let unbox: T

    init(_ value: T) {
        self.unbox = value
    }
}

enum Result<T> {
    case Value(Box<T>)
    case Error(NSError)
}
```

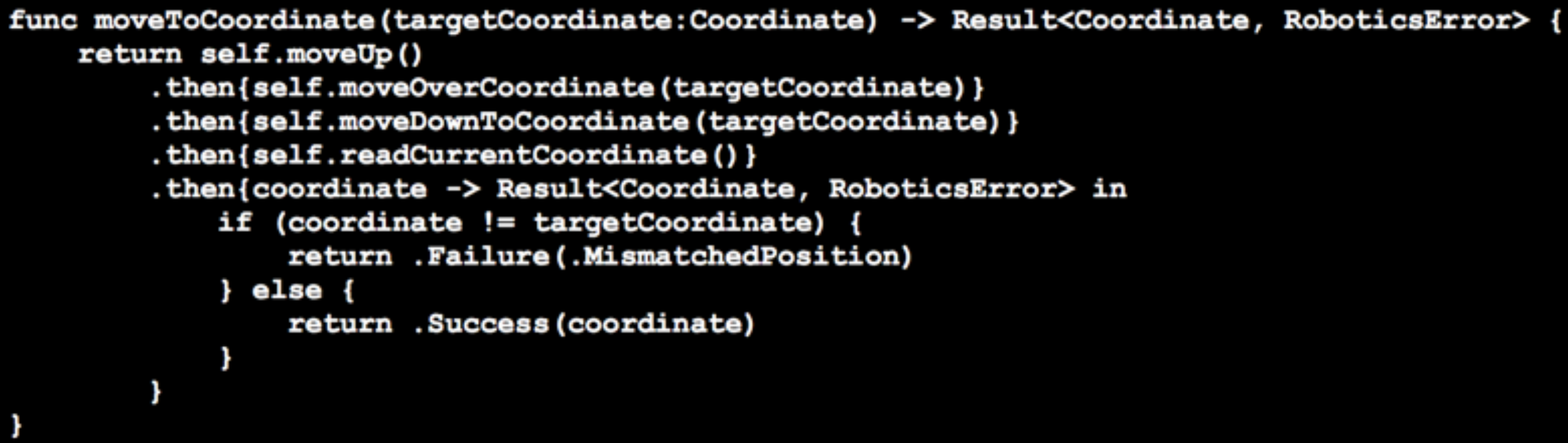
VS



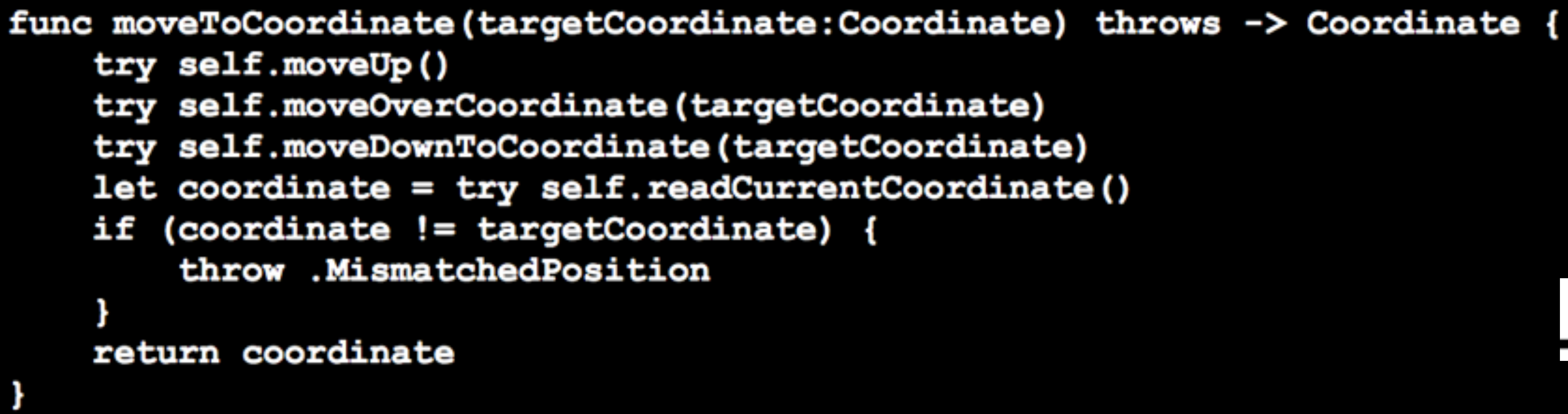
```
public enum Result<T, Error: ErrorType>
        case Success(T)
        case Failure(Error)
```

Try - Throws is great

read this http://www.sunsetlakesoftware.com/2015/06/12/swift-2-error-handling-practice



```
func moveToCoordinate(targetCoordinate:Coordinate) -> Result<Coordinate, RoboticsError> {
    return self.moveUp()
        .then{self.moveOverCoordinate(targetCoordinate)}
        .then{self.moveDownToCoordinate(targetCoordinate)}
        .then{self.readCurrentCoordinate()}
        .then{coordinate -> Result<Coordinate, RoboticsError> in
            if (coordinate != targetCoordinate) {
                return .Failure(.MismatchedPosition)
            } else {
                return .Success(coordinate)
            }
        }
}
```

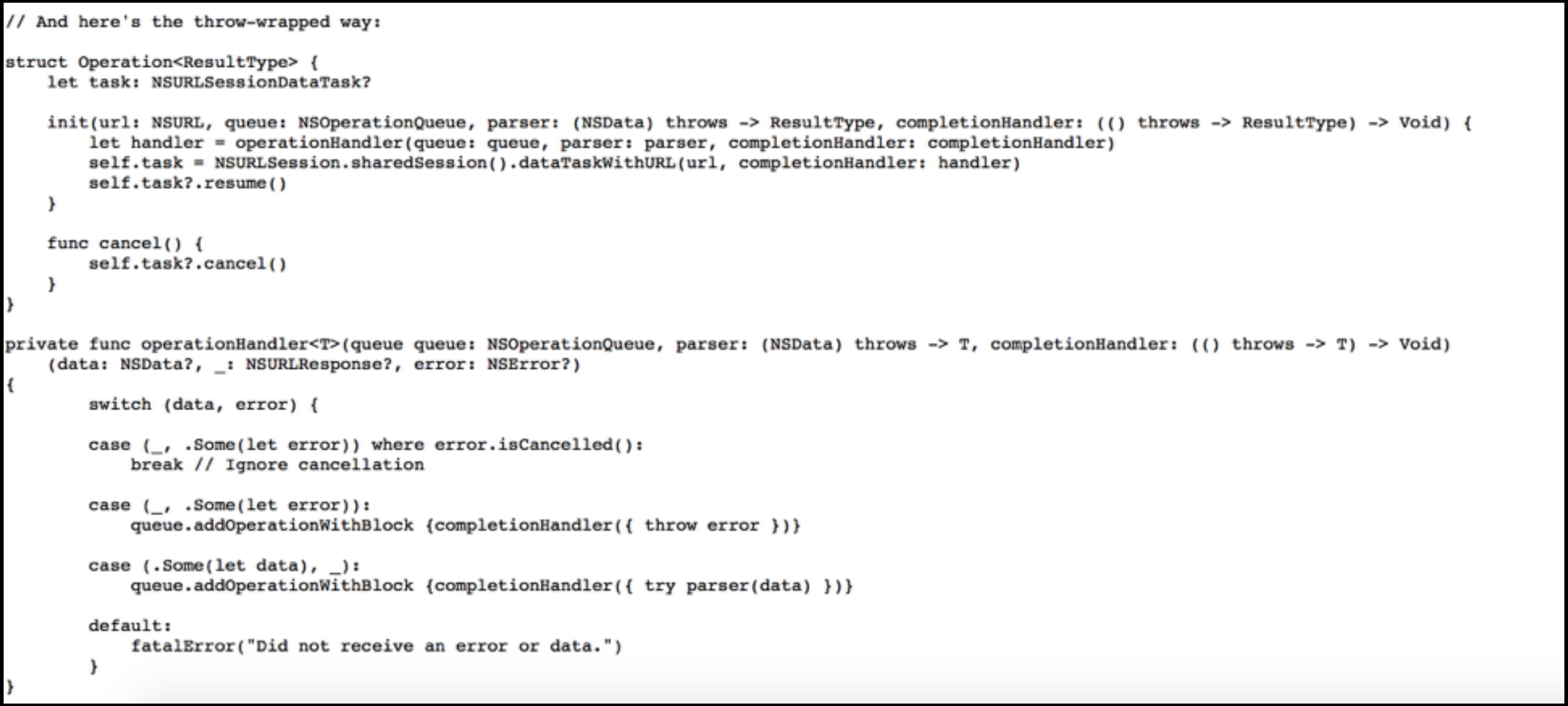


```
func moveToCoordinate(targetCoordinate:Coordinate) throws -> Coordinate {
    try self.moveUp()
    try self.moveOverCoordinate(targetCoordinate)
    try self.moveDownToCoordinate(targetCoordinate)
    let coordinate = try self.readCurrentCoordinate()
    if (coordinate != targetCoordinate) {
        throw .MismatchedPosition
    }
    return coordinate
}
```

Throw isn’t really a great model when working with async code

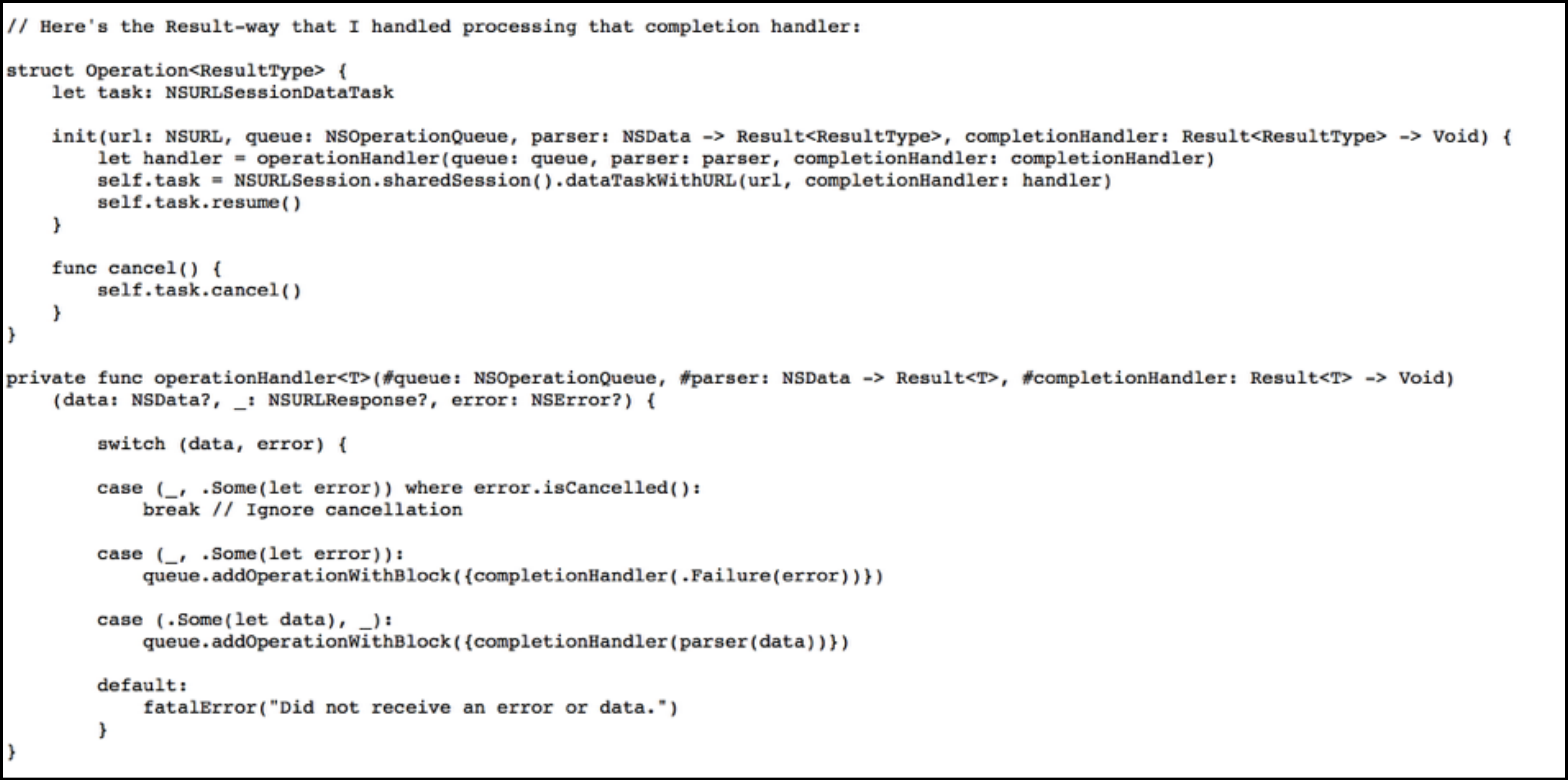
It can be done

see https://gist.github.com/rnapier/b1f13be8d018bf4d145b



```
// And here's the throw-wrapped way:

struct Operation<ResultType> {
    let task: NSURLSessionDataTask?

    init(url: NSURL, queue: NSOperationQueue, parser: (NSData) throws -> ResultType, completionHandler: (() throws -> ResultType) -> Void) {
        let handler = operationHandler(queue: queue, parser: parser, completionHandler: completionHandler)
        self.task = NSURLSession.sharedSession().dataTaskWithURL(url, completionHandler: handler)
        self.task?.resume()
    }

    func cancel() {
        self.task?.cancel()
    }
}

private func operationHandler<T>(queue queue: NSOperationQueue, parser: (NSData) throws -> T, completionHandler: (() throws -> T) -> Void)
    (data: NSData?, _: NSURLResponse?, error: NSError?)
{
        switch (data, error) {

        case (_, .Some(let error)) where error.isCancelled():
            break // Ignore cancellation

        case (_, .Some(let error)):
            queue.addOperationWithBlock {completionHandler({ throw error })}

        case (.Some(let data), _):
            queue.addOperationWithBlock {completionHandler({ try parser(data) })}

        default:
            fatalError("Did not receive an error or data.")
        }
}
```

1.2 Swift Result version



```
// Here's the Result-way that I handled processing that completion handler:

struct Operation<ResultType> {
    let task: NSURLSessionDataTask

    init(url: NSURL, queue: NSOperationQueue, parser: NSData -> Result<ResultType>, completionHandler: Result<ResultType> -> Void) {
        let handler = operationHandler(queue: queue, parser: parser, completionHandler: completionHandler)
        self.task = NSURLSession.sharedSession().dataTaskWithURL(url, completionHandler: handler)
        self.task.resume()
    }

    func cancel() {
        self.task.cancel()
    }
}

private func operationHandler<T>(#queue: NSOperationQueue, #parser: NSData -> Result<T>, #completionHandler: Result<T> -> Void)
    (data: NSData?, _: NSURLResponse?, error: NSError?) {

        switch (data, error) {

        case (_, .Some(let error)) where error.isCancelled():
            break // Ignore cancellation

        case (_, .Some(let error)):
            queue.addOperationWithBlock({completionHandler(.Failure(error))})

        case (.Some(let data), _):
            queue.addOperationWithBlock({completionHandler(parser(data))})

        default:
            fatalError("Did not receive an error or data.")
        }
}
```

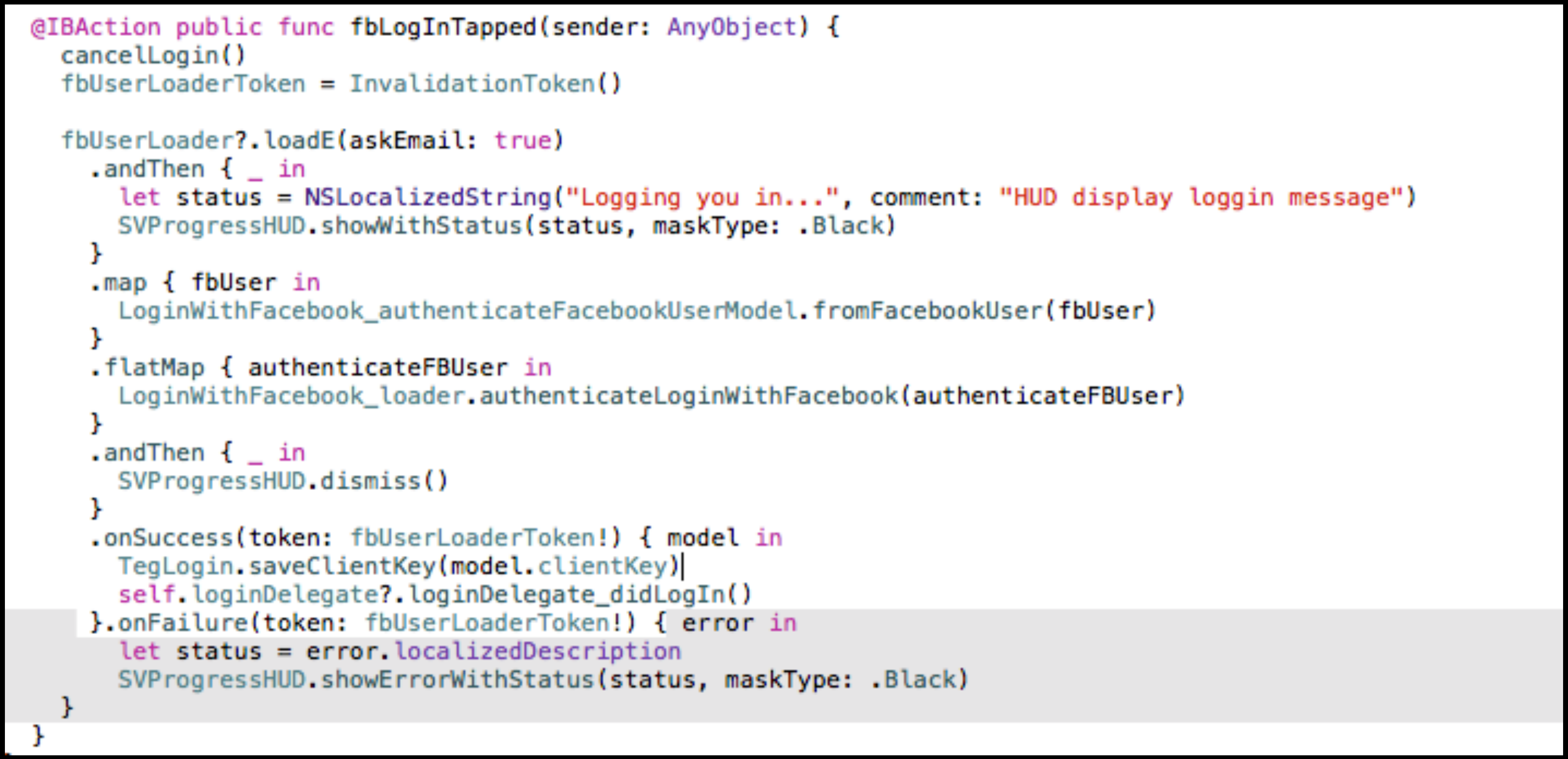
But it doesn’t feel right

read this https://gist.github.com/nicklockwood/21495c2015fd2dda56cf

"But try only works in this one situation, with sequential imperative statements. If Brad Larson was working on something like a network library instead of a robot controller, result types would work much better for error propagation than Swift 2's exceptions, because exception-style errors don't really work for asynchronous, callback-based APIs."

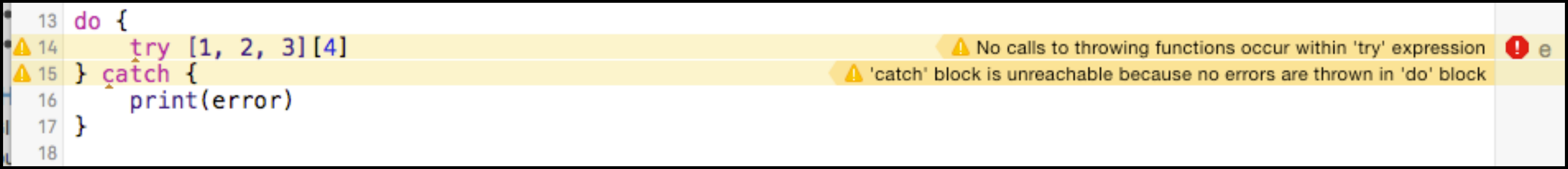
–Nick Lockwood

Future or Promise



```swift
@IBAction public func fbLogInTapped(sender: AnyObject) {
  cancelLogin()
  fbUserLoaderToken = InvalidationToken()

  fbUserLoader?.loadE(askEmail: true)
    .andThen { _ in
      let status = NSLocalizedString("Logging you in...", comment: "HUD display loggin message")
      SVProgressHUD.showWithStatus(status, maskType: .Black)
    }
    .map { fbUser in
      LoginWithFacebook_authenticateFacebookUserModel.fromFacebookUser(fbUser)
    }
    .flatMap { authenticateFBUser in
      LoginWithFacebook_loader.authenticateLoginWithFacebook(authenticateFBUser)
    }
    .andThen { _ in
      SVProgressHUD.dismiss()
    }
    .onSuccess(token: fbUserLoaderToken!) { model in
      TegLogin.saveClientKey(model.clientKey)
      self.loginDelegate?.loginDelegate_didLogIn()
    }.onFailure(token: fbUserLoaderToken!) { error in
      let status = error.localizedDescription
      SVProgressHUD.showErrorWithStatus(status, maskType: .Black)
  }
}
```

It’s not a replacement for array out of bounds check for example

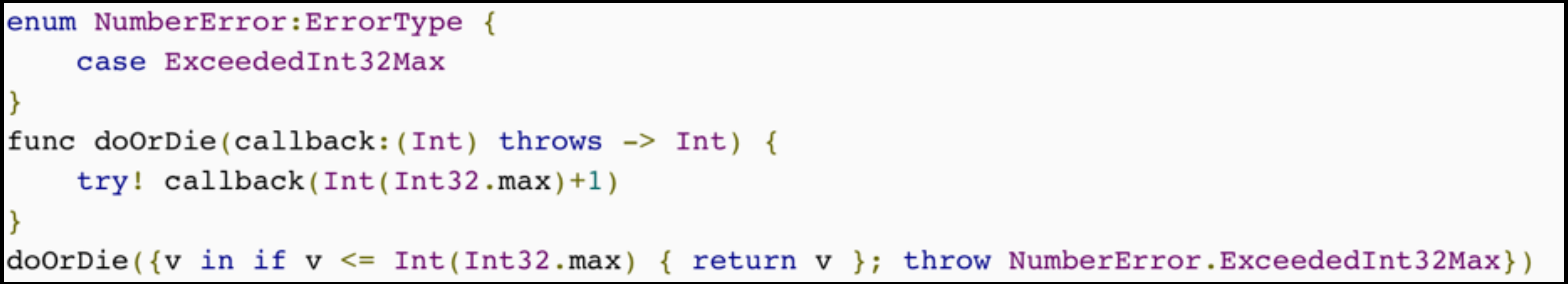


```
do {
    try [1, 2, 3][4]
} catch {
    print(error)
}
```

No calls to throwing functions occur within 'try' expression

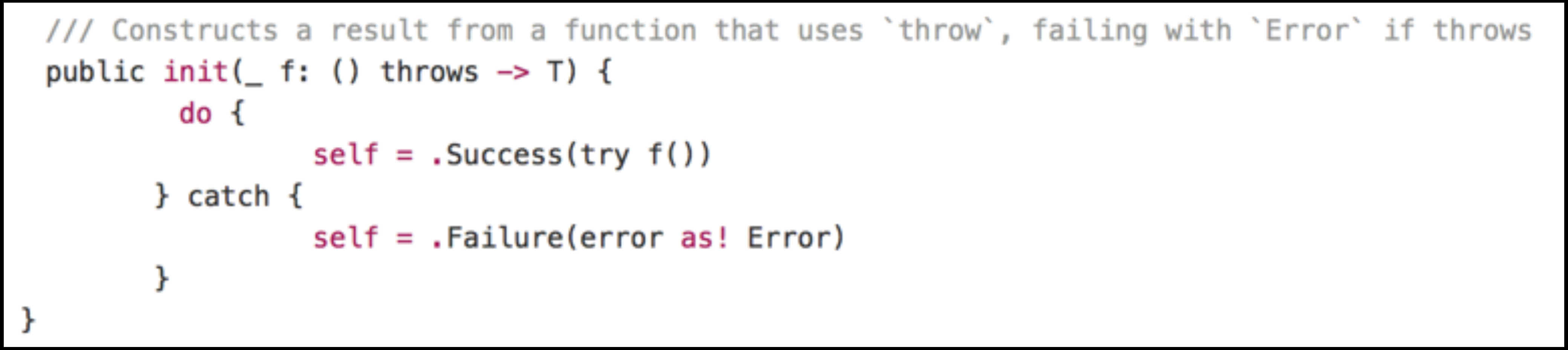
'catch' block is unreachable because no errors are thrown in 'do' block

Try! the impossible error

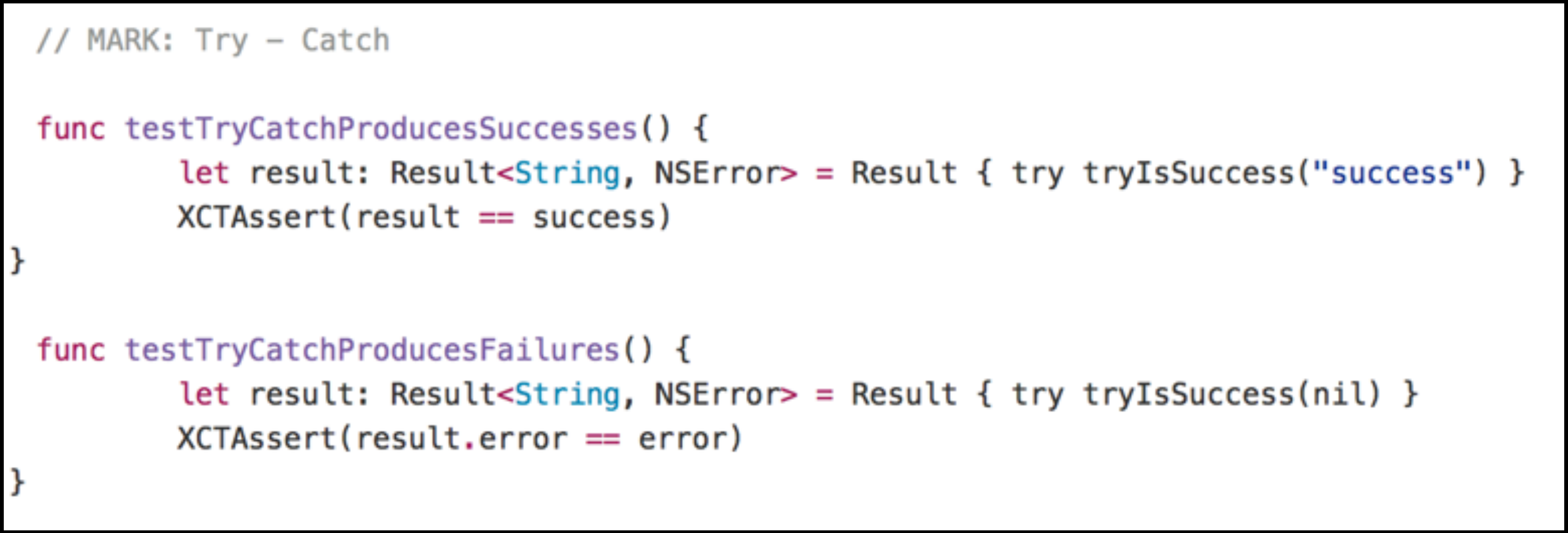


```swift
enum NumberError:ErrorType {
    case ExceededInt32Max
}
func doOrDie(callback:(Int) throws -> Int) {
    try! callback(Int(Int32.max)+1)
}
doOrDie({v in if v <= Int(Int32.max) { return v }; throw NumberError.ExceededInt32Max})
```

For the haters you can convert a function that throws into a Result

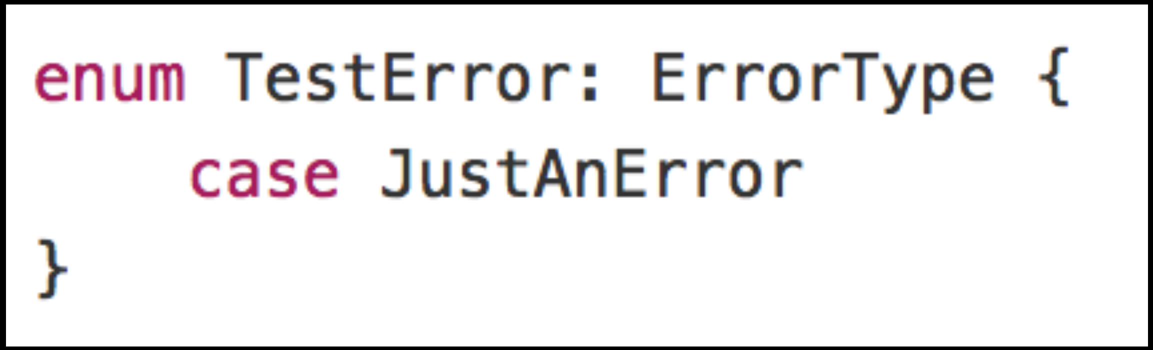


```swift
  /// Constructs a result from a function that uses `throw`, failing with `Error` if throws
  public init(_ f: () throws -> T) {
          do {
                  self = .Success(try f())
        } catch {
                  self = .Failure(error as! Error)
        }
}
```



```swift
 // MARK: Try - Catch

 func testTryCatchProducesSuccesses() {
         let result: Result<String, NSError> = Result { try tryIsSuccess("success") }
         XCTAssert(result == success)
}

 func testTryCatchProducesFailures() {
         let result: Result<String, NSError> = Result { try tryIsSuccess(nil) }
         XCTAssert(result.error == error)
}
```

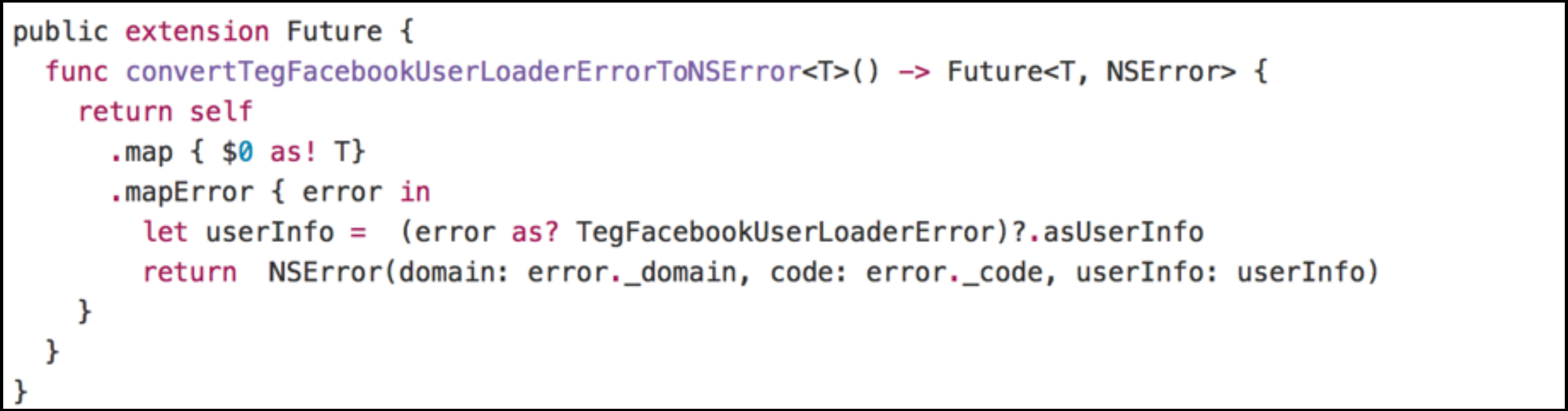
ErrorType



```swift
enum TestError: ErrorType {
    case JustAnError
}
```

```swift
public enum TegFacebookUserLoaderError: ErrorType {
  case Parsing
  case GraphRequest(error: NSError?)
  case LoginFailed(error: NSError?)
  case LoginCanceled
  case AccessToken
}

extension TegFacebookUserLoaderError: Equatable {}

public func == (lhs: TegFacebookUserLoaderError, rhs: TegFacebookUserLoaderError) -> Bool {
  switch (lhs, rhs) {
  case (.Parsing, .Parsing): return true
  case (.GraphRequest(let lError), .GraphRequest(let rError)): return lError == rError
  case (.LoginFailed(let lError), .LoginFailed(let rError)): return lError == rError
  case (.LoginCanceled, .LoginCanceled): return true
  case (.AccessToken, .AccessToken): return true
  default: return false
  }
}

extension TegFacebookUserLoaderError {
  var asUserInfo: [NSObject : AnyObject]? {
    var underlyingError: NSError?
    switch self {
    case .GraphRequest(let error):
      underlyingError = error
    case .LoginFailed(let error):
      underlyingError = error
    default: break
    }

    return underlyingError.map { [NSUnderlyingErrorKey: $0] }
  }
}
```

ErrorType -> NSError



```
public extension Future {
  func convertTegFacebookUserLoaderErrorToNSError<T>() -> Future<T, NSError> {
    return self
      .map { $0 as! T}
      .mapError { error in
        let userInfo =  (error as? TegFacebookUserLoaderError)?.asUserInfo
        return  NSError(domain: error._domain, code: error._code, userInfo: userInfo)
    }
  }
}
```

Repeat-while

Boring…

Rethrows

😷

```swift
func foo(x: Int throws -> Int) rethrows -> Int {
  return try x(0)
}

enum Foo: ErrorType { case Foo }

foo { print($0); return $0 }
try foo { _ in throw Foo.Foo }
```

“A throwing method can't override a rethrowing method, and a throwing method can't satisfy a protocol requirement for a rethrowing method. That said, a rethrowing method can override a throwing method, and a rethrowing method can satisfy a protocol requirement for a throwing method.”

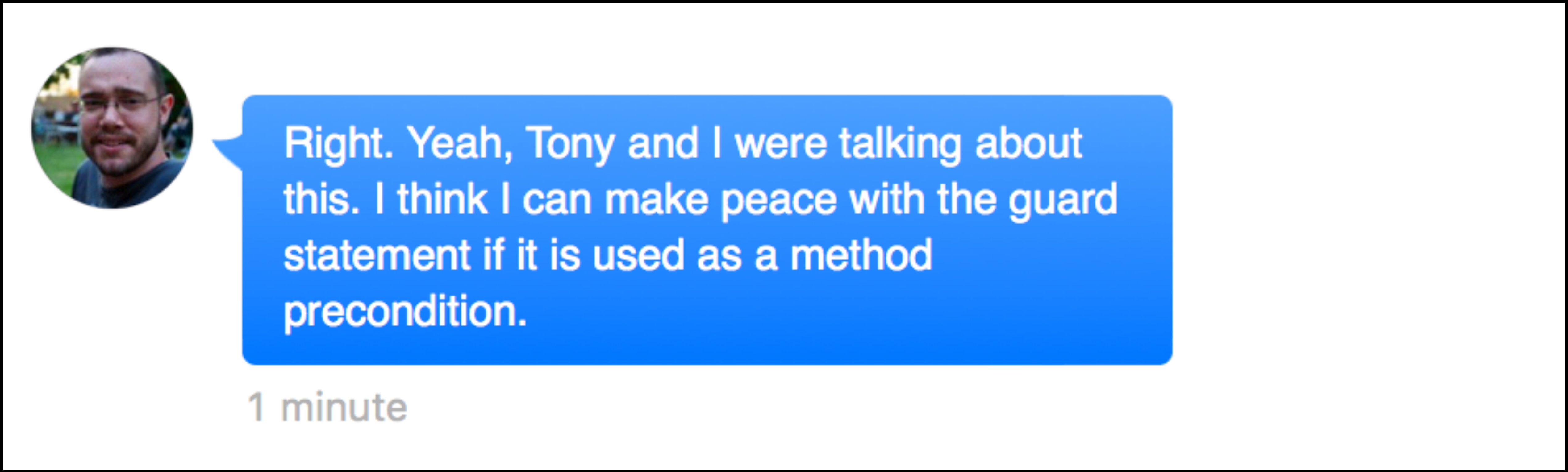
–David Steuber

Guard is fantastic

But Oliver would say…

Happy path isn’t first

But he is ok with it



Right. Yeah, Tony and I were talking about this. I think I can make peace with the guard statement if it is used as a method precondition.

1 minute

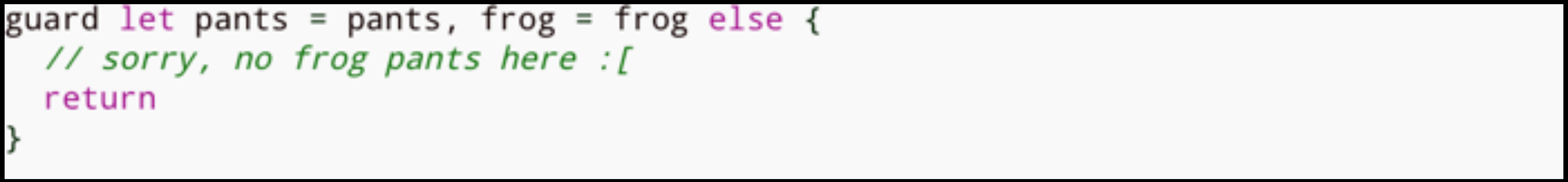
Compare…

```swift
func numericiseIPAddress(ipAddr: String) -> IPResult {
  let components = split(ipAddr.characters) { $0 == "."}
  if components.count == 4 {
    if let firstOctet = Int(String(components[0]))
      where firstOctet >= 0 && firstOctet < 256 {
        if let secondOctet = Int(String(components[1]))
          where secondOctet >= 0 && secondOctet < 256 {
            if let thirdOctet = Int(String(components[2]))
              where thirdOctet >= 0 && thirdOctet < 256 {
                if let fourthOctet = Int(String(components[3]))
                  where fourthOctet >= 0 && fourthOctet < 256 {
                    let value = fourthOctet
                      + thirdOctet * 256
                      + secondOctet * (256*256)
                      + firstOctet * (256*256*256)
                    return .Success(value)
                } else {
                  return .Failure("fourth octet was bad")
                }
            } else {
              return .Failure("third octet was bad")
            }
        } else {
          return .Failure("second octet was bad")
        }
    } else {
      return .Failure("first octet was bad")
    }
  } else {
    return .Failure(components.count < 4 ? "too few components" : "too many components")
  }
}
```

To this…

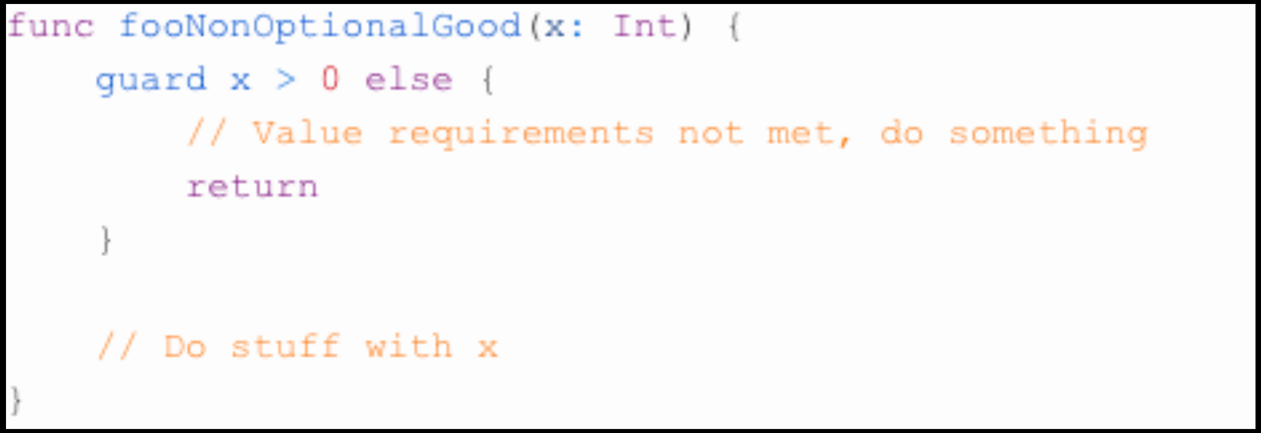
```
func numericiseIPAddress(ipAddr: String) -> IPResult {
  // Split dot-decimal string into its (presumably) four components
  let components = split(ipAddr.characters) { $0 == "."}

  // Ensure split happened correctly
  guard components.count == 4 else {
    return .Failure(components.count < 4 ? "too few components" : "too many components")
  }

  // Get first octet
  guard let firstOctet = Int(String(components[0]))
    where firstOctet >= 0 && firstOctet < 256 else {
      return .Failure("first octet was bad")
  }
  // Get second octet
  guard let secondOctet = Int(String(components[1]))
    where secondOctet >= 0 && secondOctet < 256 else {
      return .Failure("second octet was bad")
  }
  // Get third octet
  guard let thirdOctet = Int(String(components[2]))
    where thirdOctet >= 0 && thirdOctet < 256 else {
      return .Failure("third octet was bad")
  }
  // Get fourth octet
  guard let fourthOctet = Int(String(components[3]))
    where fourthOctet >= 0 && fourthOctet < 256 else {
      return .Failure("fourth octet was bad")
  }
  // Calculate numerical value of IP address, given the values of each octet
  let value = fourthOctet
    + thirdOctet * 256
    + secondOctet * (256*256)
    + firstOctet * (256*256*256)
  return .Success(value)
}
```

Also this is legit



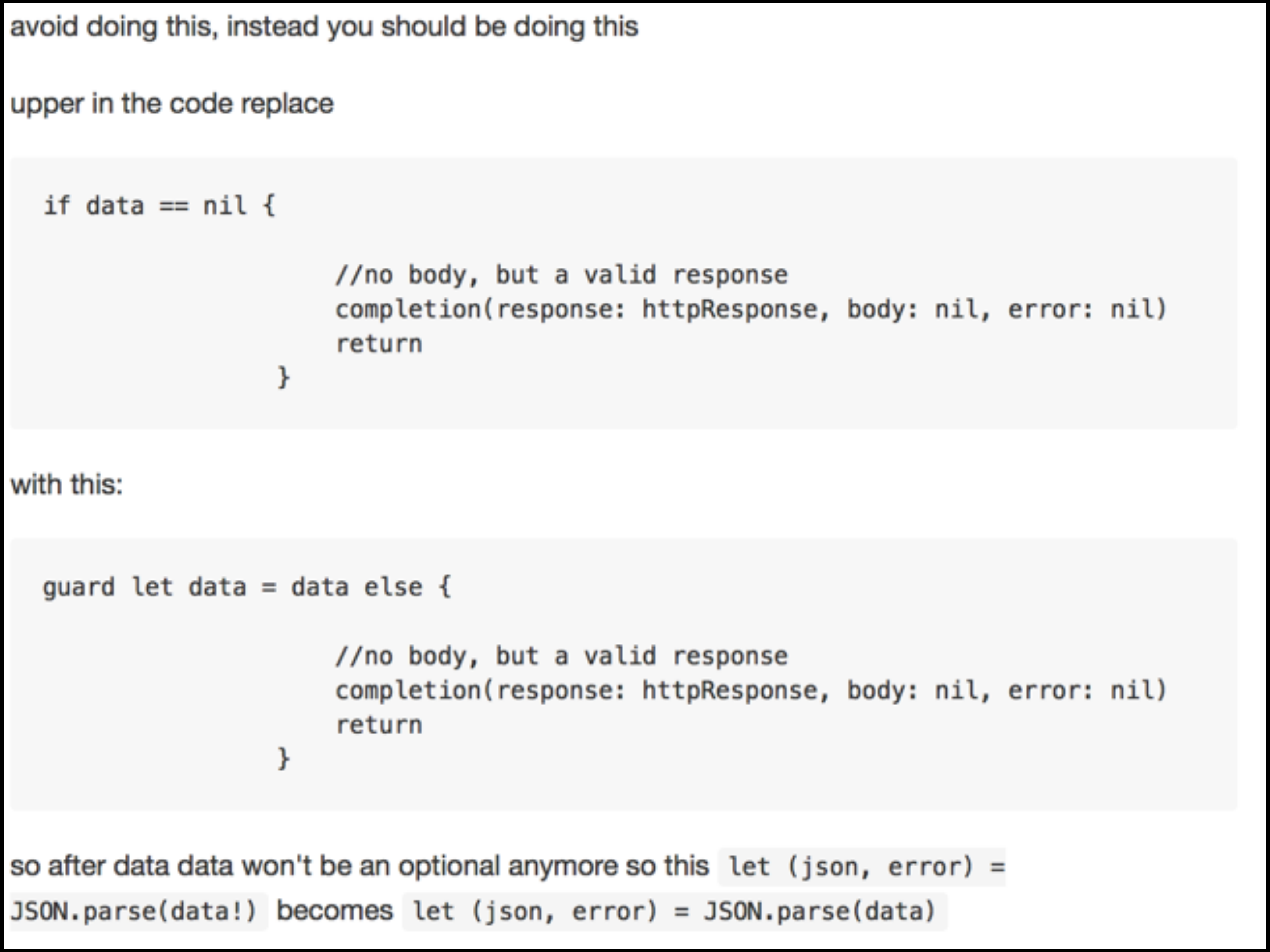
```
guard let pants = pants, frog = frog else {
  // sorry, no frog pants here :[
  return
}
```

Same pattern holds true for non-optional values



```
func fooNonOptionalGood(x: Int) {
    guard x > 0 else {
        // Value requirements not met, do something
        return
    }

    // Do stuff with x
}
```

Don’t forget to use it



avoid doing this, instead you should be doing this

upper in the code replace

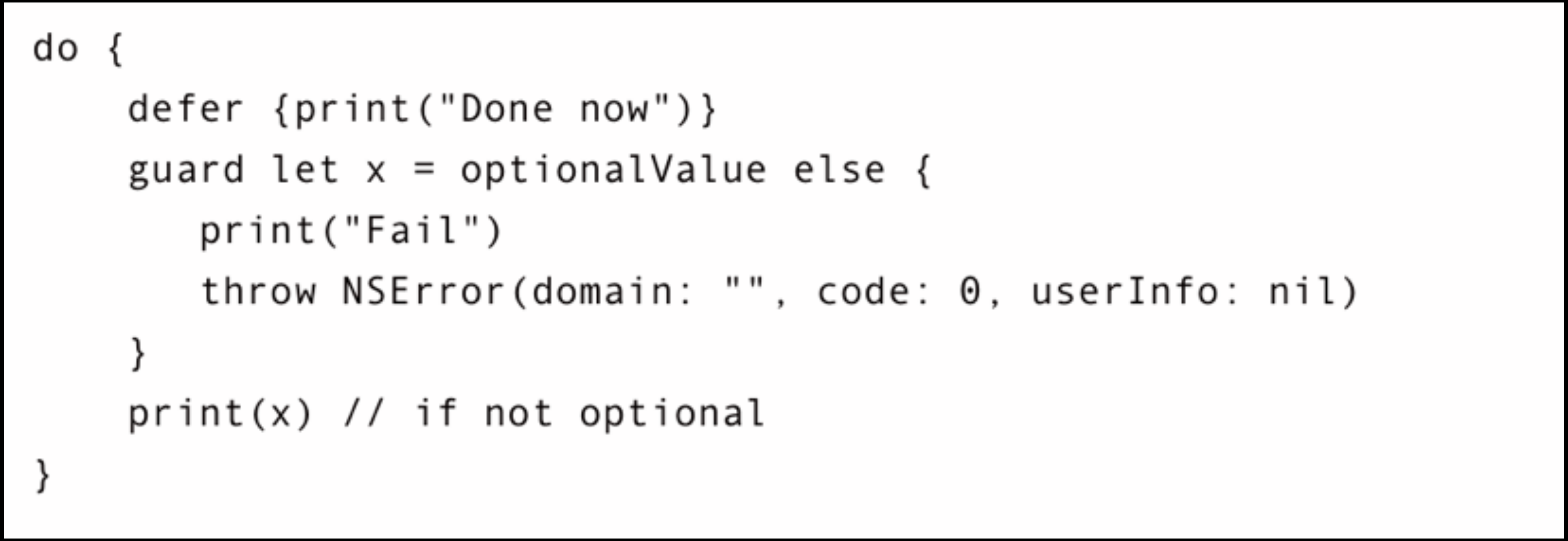
```
if data == nil {

                    //no body, but a valid response
                    completion(response: httpResponse, body: nil, error: nil)
                    return
                }
```

with this:

```
guard let data = data else {

                    //no body, but a valid response
                    completion(response: httpResponse, body: nil, error: nil)
                    return
                }
```

so after data data won't be an optional anymore so this `let (json, error) = JSON.parse(data!)` becomes `let (json, error) = JSON.parse(data)`

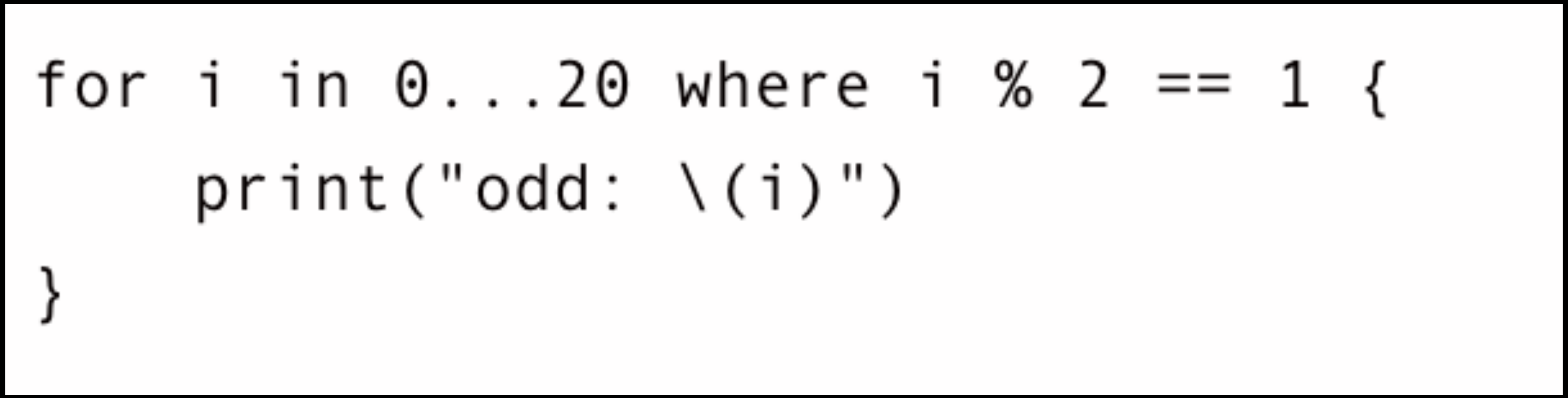
It goes hand in hand with Defer, Do and Error handling



```
do {
    defer {print("Done now")}
    guard let x = optionalValue else {
       print("Fail")
       throw NSError(domain: "", code: 0, userInfo: nil)
    }
    print(x) // if not optional
}
```

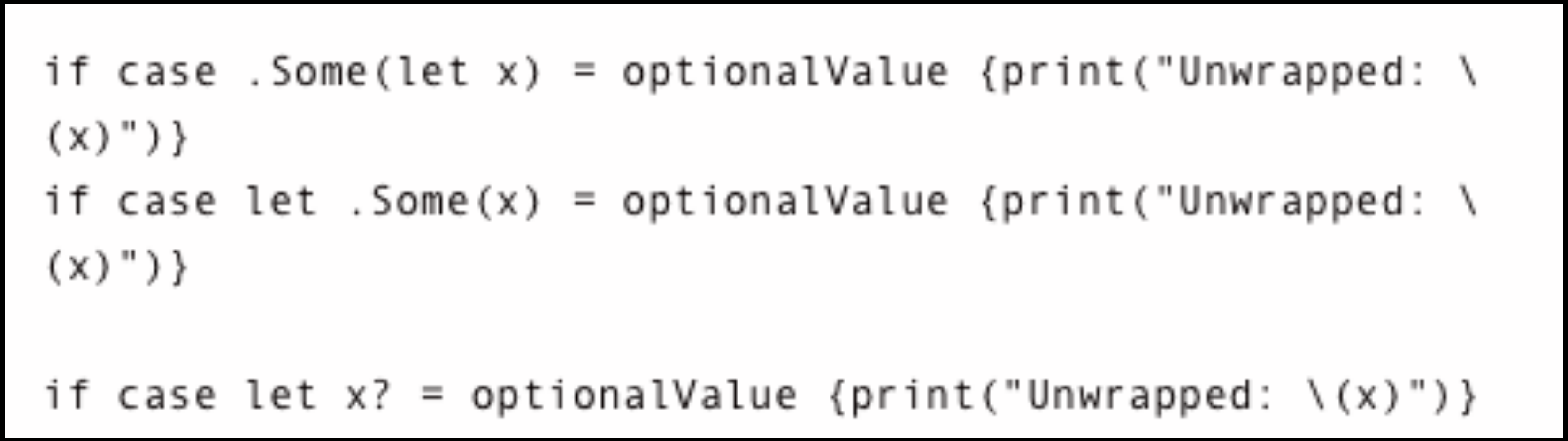
For-in where clauses

Python to the rescue

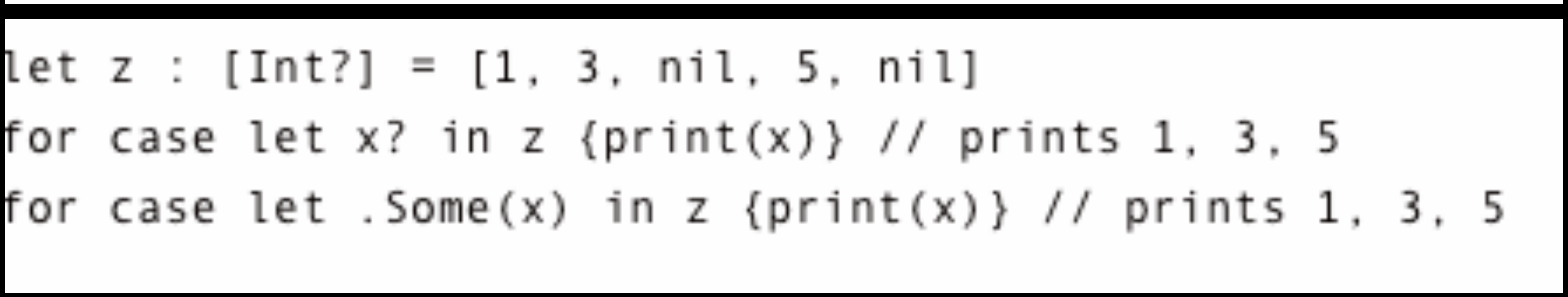


```
for i in 0...20 where i % 2 == 1 {
    print("odd: \(i)")
}
```

Case clauses in conditional

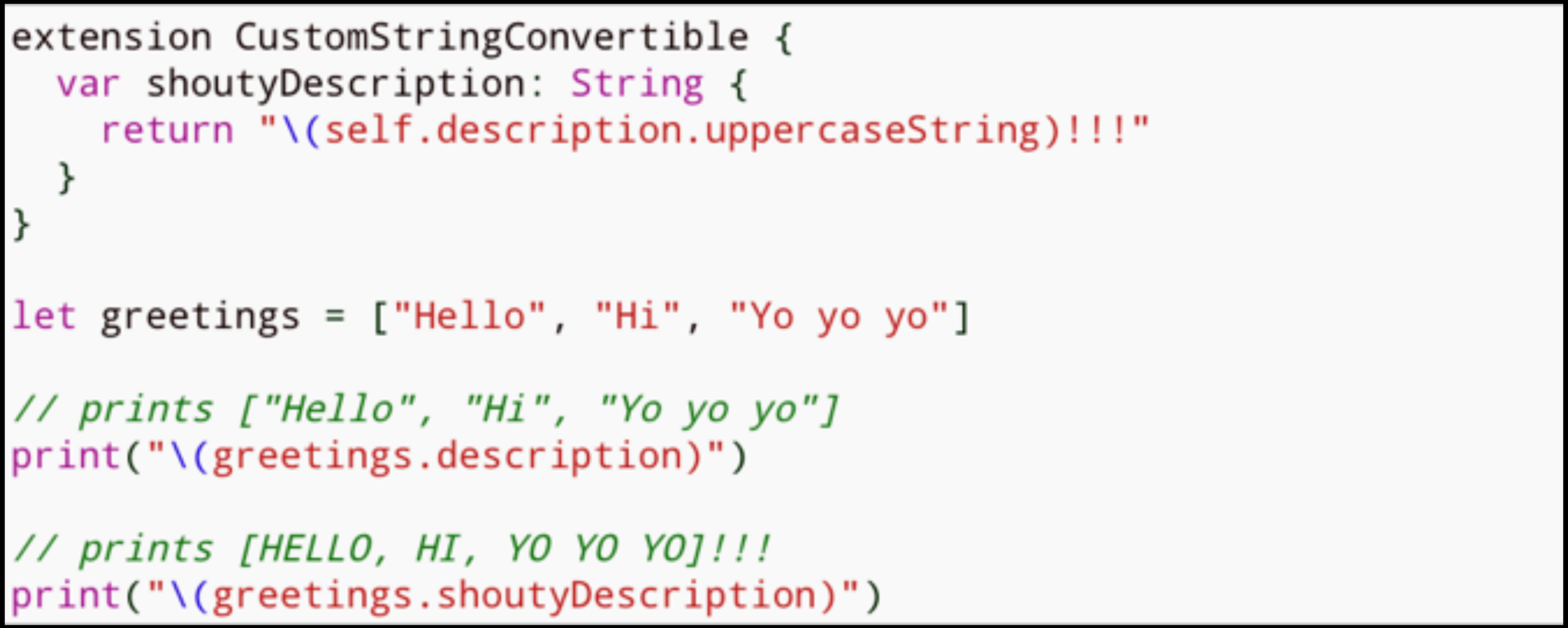


```
if case .Some(let x) = optionalValue {print("Unwrapped: \
(x)")}
if case let .Some(x) = optionalValue {print("Unwrapped: \
(x)")}

if case let x? = optionalValue {print("Unwrapped: \(x)")}
```



```
let z : [Int?] = [1, 3, nil, 5, nil]
for case let x? in z {print(x)} // prints 1, 3, 5
for case let .Some(x) in z {print(x)} // prints 1, 3, 5
```

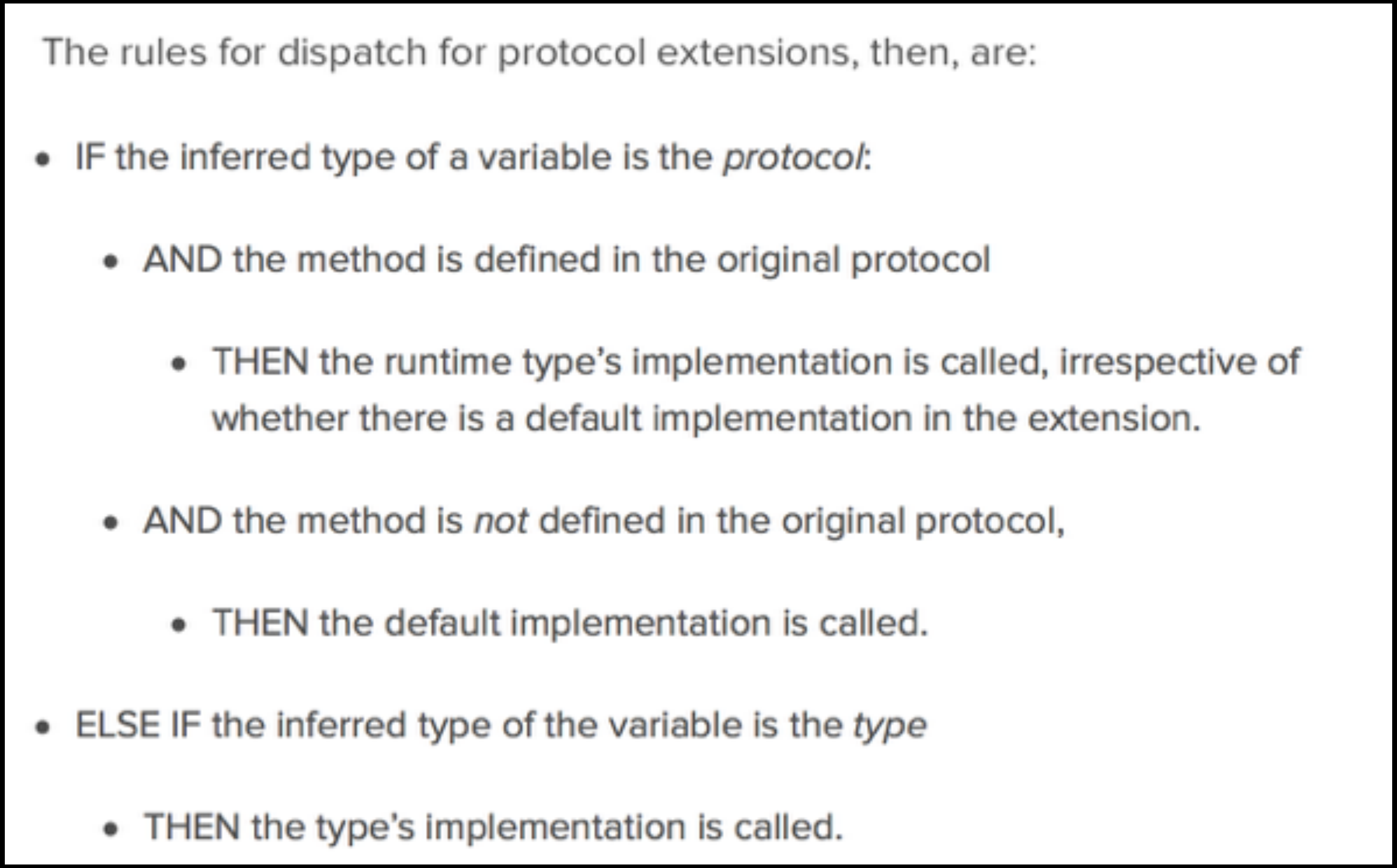
Protocol extensions



```
extension CustomStringConvertible {
  var shoutyDescription: String {
    return "\(self.description.uppercaseString)!!!"
  }
}

let greetings = ["Hello", "Hi", "Yo yo yo"]

// prints ["Hello", "Hi", "Yo yo yo"]
print("\(greetings.description)")

// prints [HELLO, HI, YO YO YO]!!!
print("\(greetings.shoutyDescription)")
```

But there is a catch, be extremely careful

read this http://nomothetis.svbtle.com/the-ghost-of-swift-bugs-future



The rules for dispatch for protocol extensions, then, are:

- IF the inferred type of a variable is the *protocol*:
    - AND the method is defined in the original protocol
        - THEN the runtime type's implementation is called, irrespective of whether there is a default implementation in the extension.
    - AND the method is *not* defined in the original protocol,
        - THEN the default implementation is called.
- ELSE IF the inferred type of the variable is the *type*
    - THEN the type's implementation is called.

Mirror

JSON again? see http://chris.eidhof.nl/posts/swift-mirrors-and-json.html

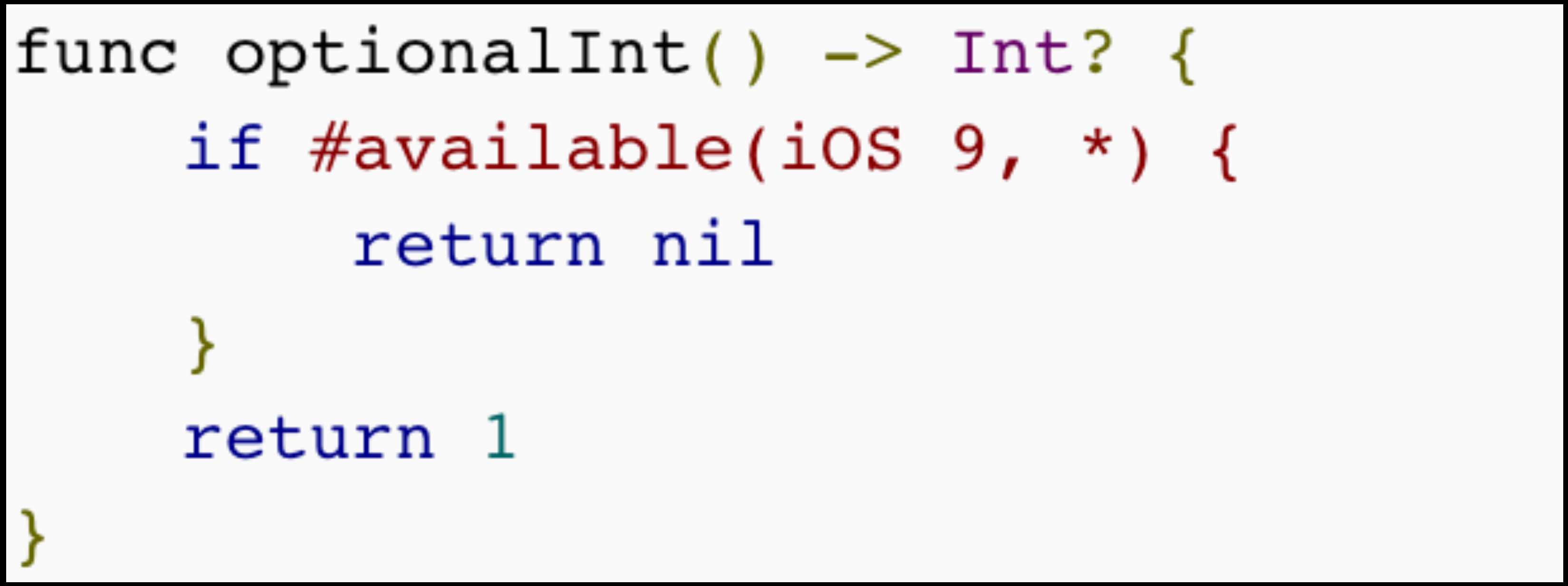
Pattern matching

```
var username: String?
var password: String?

switch (username, password) {
case let (.Some(username), .Some(password)):
    print("Success!")
case let (.Some(username), .None):
    print("Password is missing")
case let (.None, .Some(password)):
    print("Username is missing")
case (.None, .None):
    print("Both username and password are missing")
}
```

```
var username: String?
var password: String?

switch (username, password) {
case let (username?, password?):
    print("Success!")
case let (username?, nil):
    print("Password is missing")
case let (nil, password?):
    print("Username is missing")
case (nil, nil):
    print("Both username and password are missing")
}
```

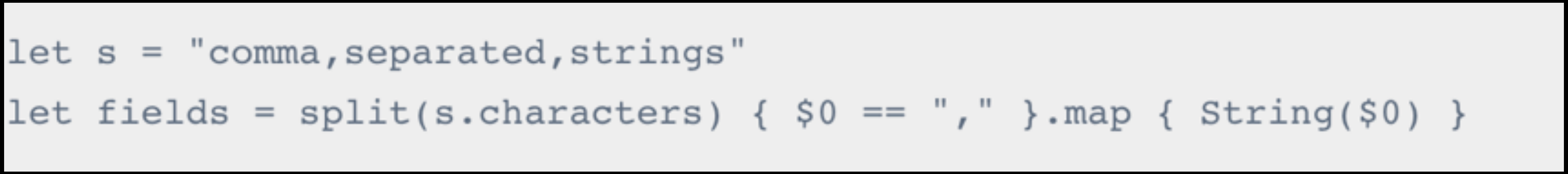
#available

compiler enforced!, Xcode Fix it integration, also great use with a Factory



```
func optionalInt() -> Int? {
    if #available(iOS 9, *) {
        return nil
    }
    return 1
}
```

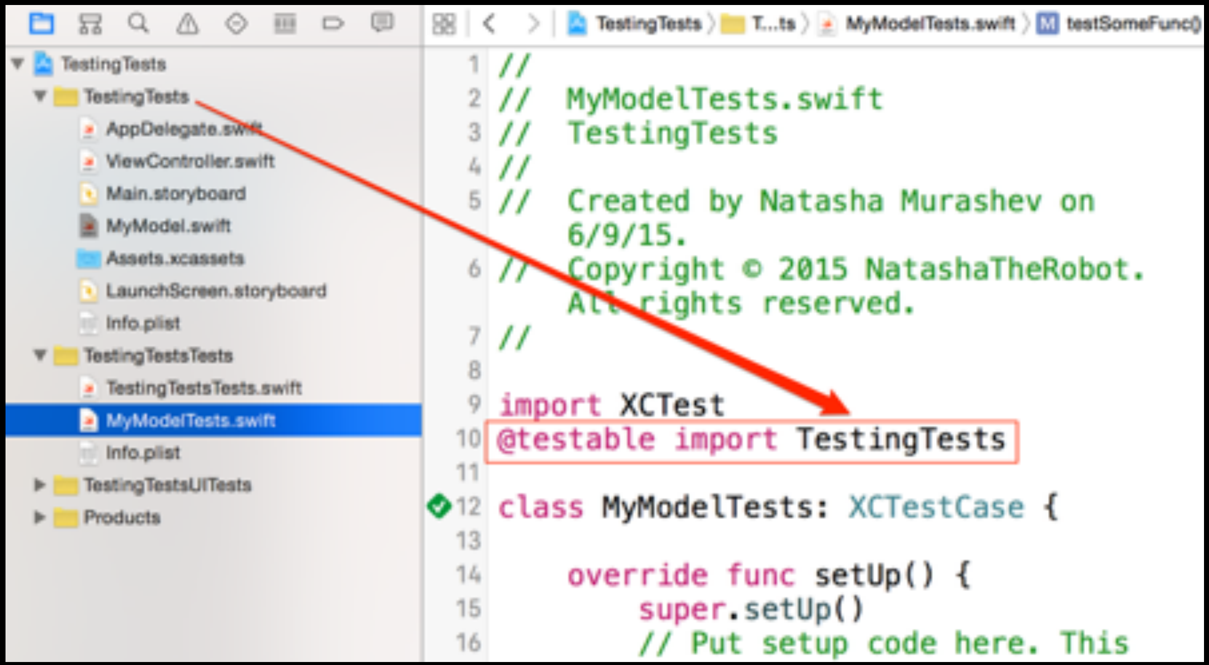
Strings are no longer collections



```
let s = "comma,separated,strings"
let fields = split(s.characters) { $0 == "," }.map { String($0) }
```

@testable

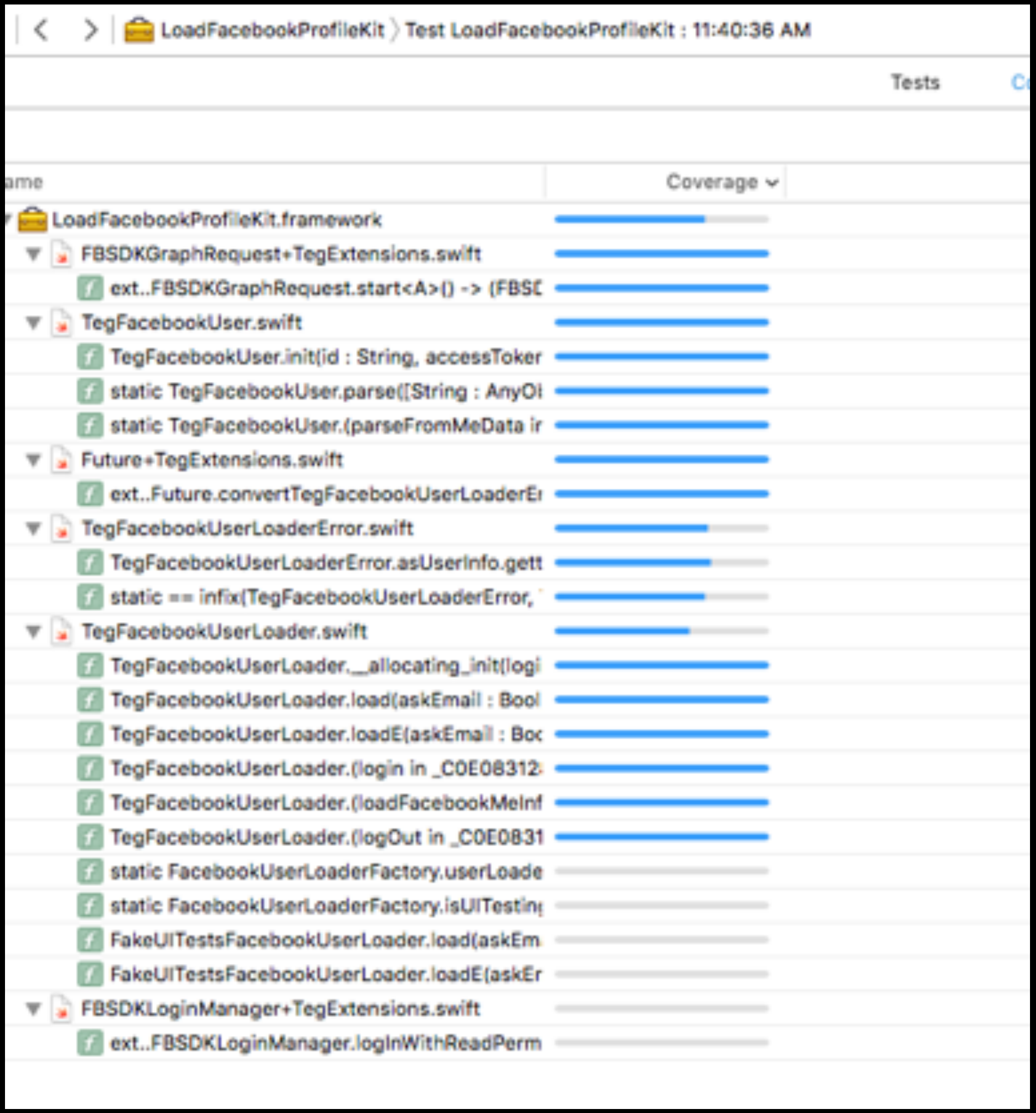
public all the things no more!



```
//
//  MyModelTests.swift
//  TestingTests
//
//  Created by Natasha Murashev on
    6/9/15.
//  Copyright © 2015 NatashaTheRobot.
    All rights reserved.
//

import XCTest
@testable import TestingTests

class MyModelTests: XCTestCase {

    override func setUp() {
        super.setUp()
        // Put setup code here. This
```

Xcode 7

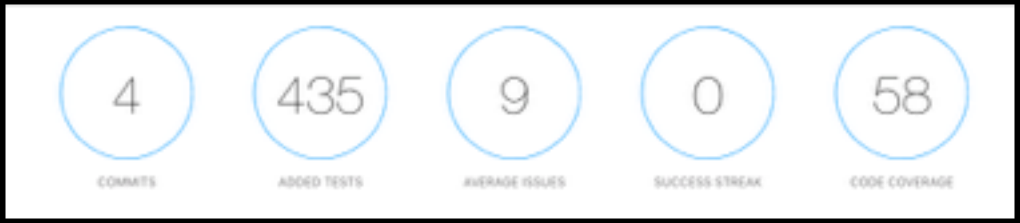
Code Coverage



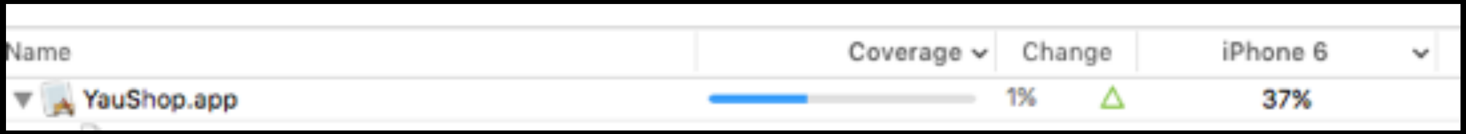
LoadFacebookProfileKit 〉 Test LoadFacebookProfileKit : 11:40:36 AM

Tests

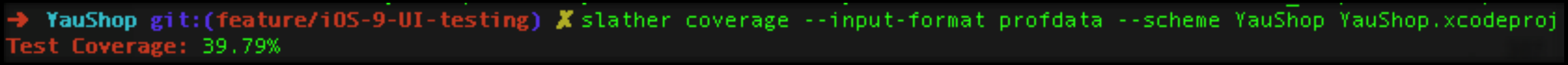
ame	Coverage
LoadFacebookProfileKit.framework	
FBSDKGraphRequest+TegExtensions.swift	
ext..FBSDKGraphRequest.start<A>() -> (FBSE	
TegFacebookUser.swift	
TegFacebookUser.init(id : String, accessToker	
static TegFacebookUser.parse([String : AnyOl	
static TegFacebookUser.(parseFromMeData ir	
Future+TegExtensions.swift	
ext..Future.convertTegFacebookUserLoaderEr	
TegFacebookUserLoaderError.swift	
TegFacebookUserLoaderError.asUserInfo.gett	
static == infix(TegFacebookUserLoaderError,	
TegFacebookUserLoader.swift	
TegFacebookUserLoader.__allocating_init(logi	
TegFacebookUserLoader.load(askEmail : Bool	
TegFacebookUserLoader.loadE(askEmail : Boo	
TegFacebookUserLoader.(login in _C0E08312	
TegFacebookUserLoader.(loadFacebookMeInf	
TegFacebookUserLoader.(logOut in _C0E0831	
static FacebookUserLoaderFactory.userLoade	
static FacebookUserLoaderFactory.isUITesting	
FakeUITestsFacebookUserLoader.load(askEm	
FakeUITestsFacebookUserLoader.loadE(askEr	
FBSDKLoginManager+TegExtensions.swift	
ext..FBSDKLoginManager.logInWithReadPerm	



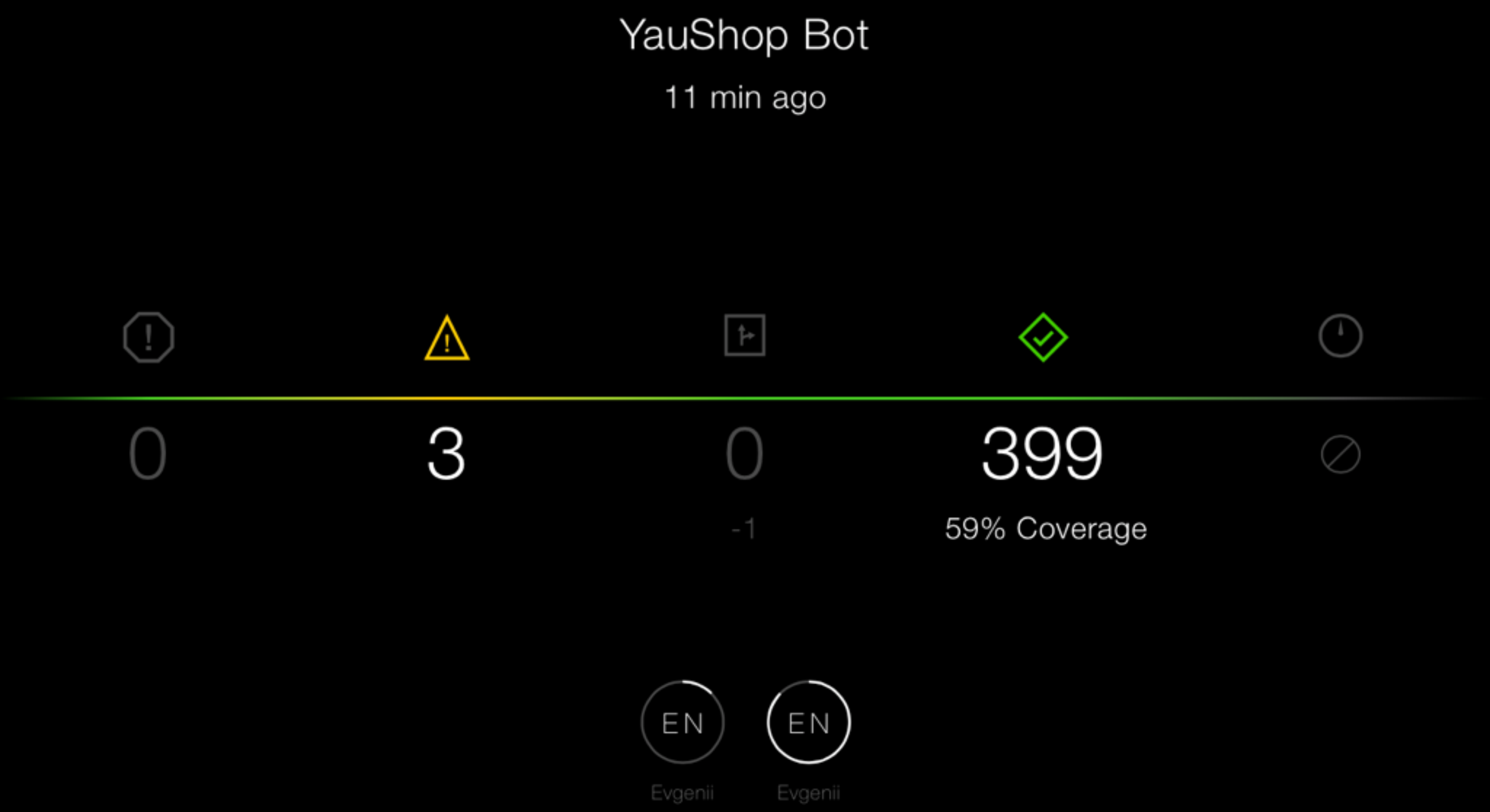
4	435	9	0	58
COMMITS	ADDED TESTS	AVERAGE ISSUES	SUCCESS STREAK	CODE COVERAGE



Name	Coverage	Change	iPhone 6
YauShop.app		1% △	37%



```
➔ YauShop git:(feature/iOS-9-UI-testing) ✗ slather coverage --input-format profdata --scheme YauShop YauShop.xcodeproj
Test Coverage: 39.79%
```



YauShop Bot

11 min ago

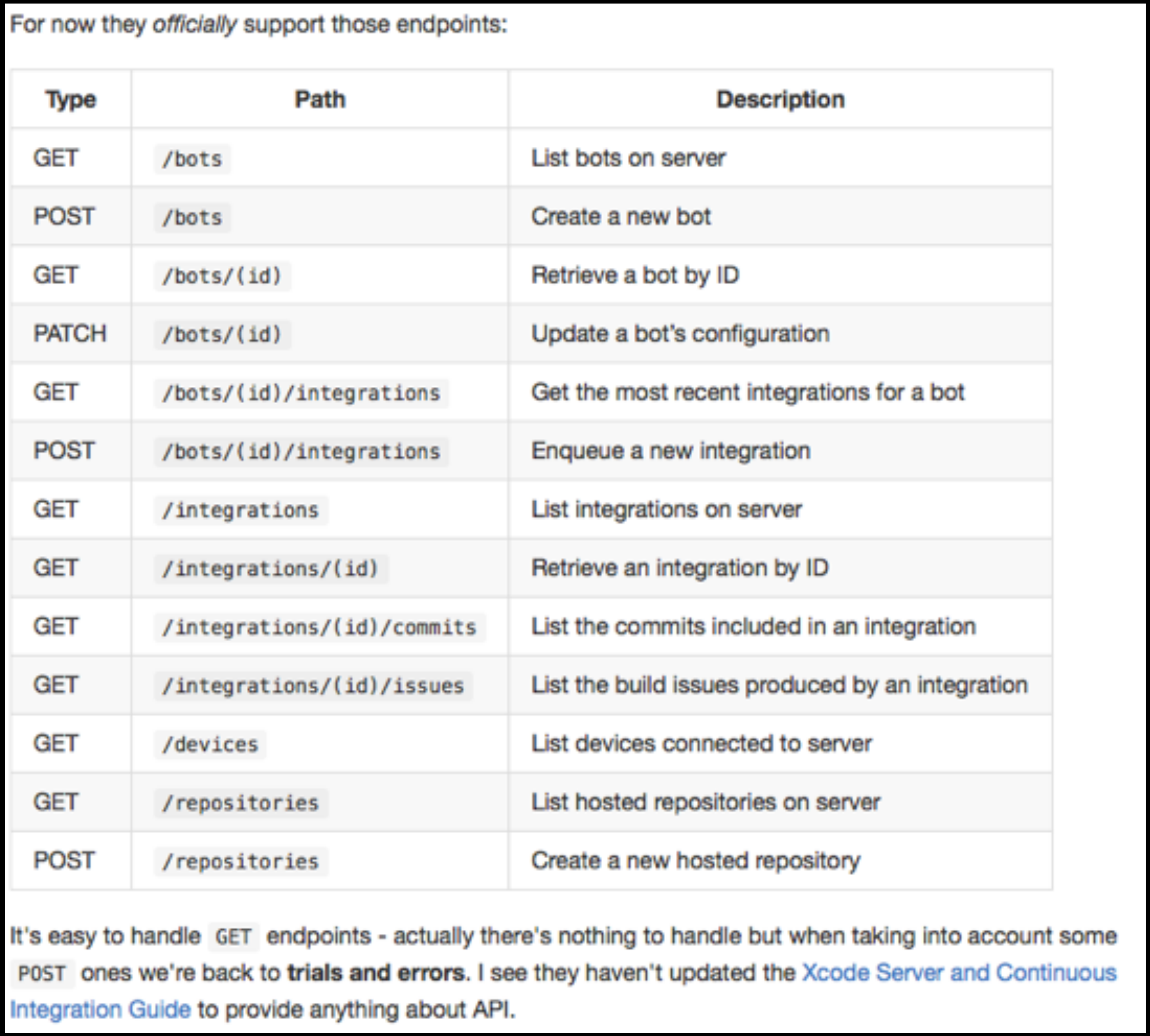
⬣	⚠	▣	◇✓	◷
0	3	0	399	⊘
		-1	59% Coverage	

EN Evgenii

EN Evgenii

Xcode 🚢 API

see https://github.com/czechboy0/XcodeServerSDK



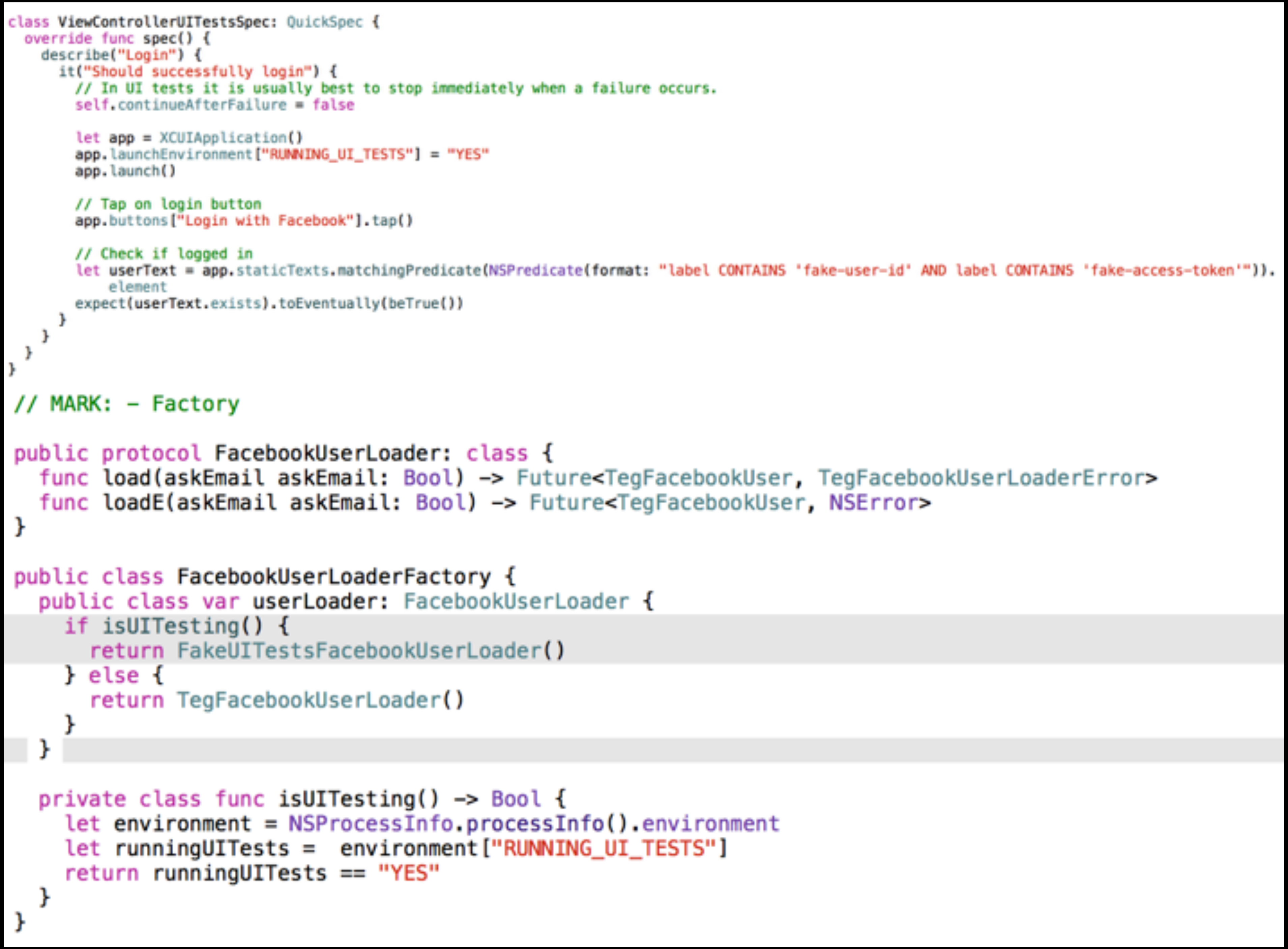
For now they *officially* support those endpoints:

Type	Path	Description
GET	`/bots`	List bots on server
POST	`/bots`	Create a new bot
GET	`/bots/(id)`	Retrieve a bot by ID
PATCH	`/bots/(id)`	Update a bot's configuration
GET	`/bots/(id)/integrations`	Get the most recent integrations for a bot
POST	`/bots/(id)/integrations`	Enqueue a new integration
GET	`/integrations`	List integrations on server
GET	`/integrations/(id)`	Retrieve an integration by ID
GET	`/integrations/(id)/commits`	List the commits included in an integration
GET	`/integrations/(id)/issues`	List the build issues produced by an integration
GET	`/devices`	List devices connected to server
GET	`/repositories`	List hosted repositories on server
POST	`/repositories`	Create a new hosted repository

It's easy to handle `GET` endpoints - actually there's nothing to handle but when taking into account some `POST` ones we're back to **trials and errors**. I see they haven't updated the Xcode Server and Continuous Integration Guide to provide anything about API.

UI Testing

it isn’t KIF, your app is running on a separate process. If you want to alter the behaviour of your app use environment variables

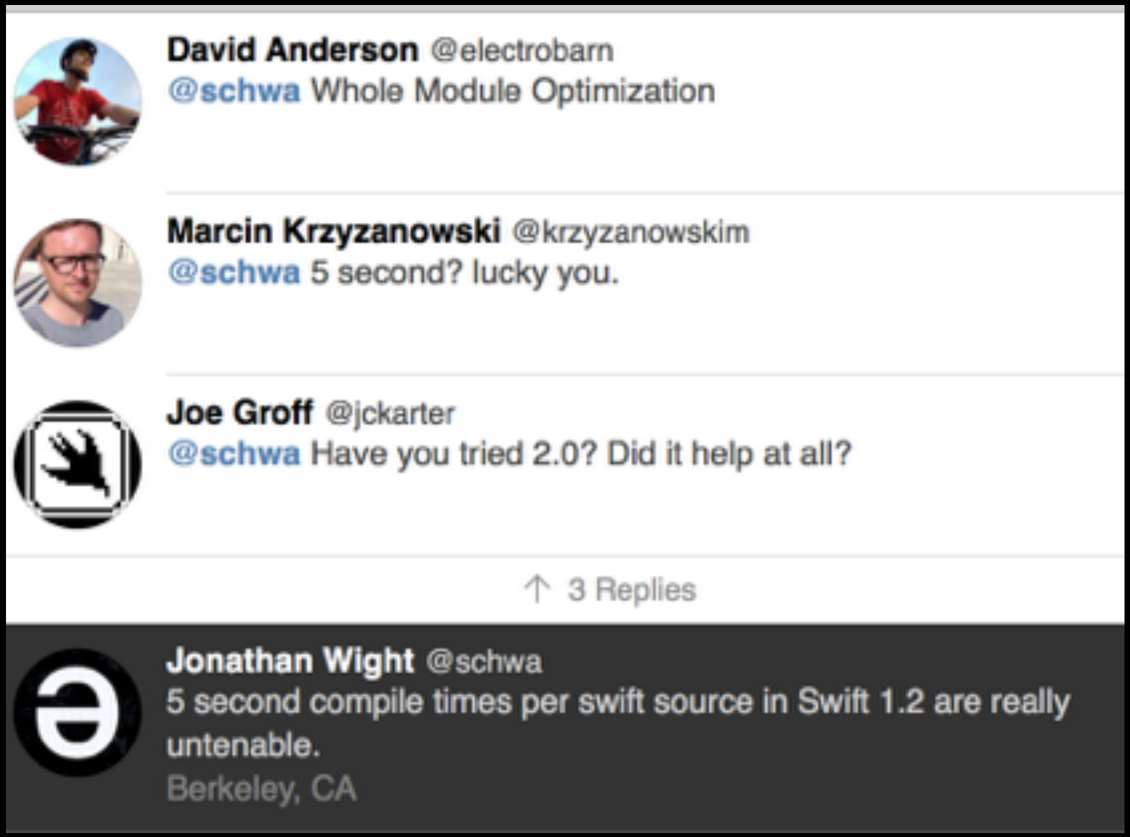


```
class ViewControllerUITestsSpec: QuickSpec {
  override func spec() {
    describe("Login") {
      it("Should successfully login") {
        // In UI tests it is usually best to stop immediately when a failure occurs.
        self.continueAfterFailure = false

        let app = XCUIApplication()
        app.launchEnvironment["RUNNING_UI_TESTS"] = "YES"
        app.launch()

        // Tap on login button
        app.buttons["Login with Facebook"].tap()

        // Check if logged in
        let userText = app.staticTexts.matchingPredicate(NSPredicate(format: "label CONTAINS 'fake-user-id' AND label CONTAINS 'fake-access-token'")).
            element
        expect(userText.exists).toEventually(beTrue())
      }
    }
  }
}

// MARK: - Factory

public protocol FacebookUserLoader: class {
  func load(askEmail askEmail: Bool) -> Future<TegFacebookUser, TegFacebookUserLoaderError>
  func loadE(askEmail askEmail: Bool) -> Future<TegFacebookUser, NSError>
}

public class FacebookUserLoaderFactory {
  public class var userLoader: FacebookUserLoader {
    if isUITesting() {
      return FakeUITestsFacebookUserLoader()
    } else {
      return TegFacebookUserLoader()
    }
  }

  private class func isUITesting() -> Bool {
    let environment = NSProcessInfo.processInfo().environment
    let runningUITests =  environment["RUNNING_UI_TESTS"]
    return runningUITests == "YES"
  }
}
```

Stability greatly improved

Swift 1.2 vs 2.0

a ‘whole’ new world



David Anderson @electrobarn
@schwa Whole Module Optimization

Marcin Krzyzanowski @krzyzanowskim
@schwa 5 second? lucky you.

Joe Groff @jckarter
@schwa Have you tried 2.0? Did it help at all?

Jonathan Wight @schwa
5 second compile times per swift source in Swift 1.2 are really untenable.
Berkeley, CA

95% of the code samples on this slides aren’t mine but theirs…

- https://gist.github.com/nicklockwood/21495c2015fd2dda56cf
- https://medium.com/swift-programming/keep-your-swift-exceptions-clean-easy-to-update-and-future-proof-20b997d0b46c
- http://austinzheng.com/2015/06/08/swift-2-control-flow/
- https://medium.com/the-traveled-ios-developers-guide/protocol-oriented-programming-9e1641946b5c
- http://radex.io/swift/error-conversions/
- https://gist.github.com/rnapier/b1f13be8d018bf4d145b
- http://chris.eidhof.nl/posts/swift-mirrors-and-json.html
- http://natashatherobot.com/swift-2-pattern-matching-unwrapping-multiple-optionals/
- http://matthewfecher.com/app-developement/swift-2-0s-new-take-on-defensive-design-with-guard/
- http://www.raywenderlich.com/108522/whats-new-in-swift-2
- http://sketchytech.blogspot.com/2015/06/living-in-post-oop-world-protocols-rule.html
- http://ericcerney.com/swift-guard-statement/
- http://nomothetis.svbtle.com/error-handling-in-swift

- http://www.sunsetlakesoftware.com/2015/06/12/swift-2-error-handling-practice
- http://robnapier.net/initial-guards
- http://nomothetis.svbtle.com/the-ghost-of-swift-bugs-future
- http://ericasadun.com/2015/06/12/swift-diffs/
- http://natashatherobot.com/swift-2-error-handling/
- http://sketchytech.blogspot.com.au/2015/06/closures-that-throw-rethrows-in-swift-20.html?m=1
- http://jamesonquave.com/blog/swift-2-whats-new/
- http://sketchytech.blogspot.com/2015/06/whats-new-in-swift-20-repeat-while.html
- https://github.com/rnapier/WikipediaSearcher/blob/master/WikiSearch/JSON.swift
- http://airspeedvelocity.net/2015/06/09/changes-to-the-swift-standard-library-in-2-0-beta-1/
- http://ericasadun.com/2015/06/09/swift-why-try-and-catch-dont-work-the-way-you-expect/
- https://gist.github.com/jckarter/85a8313201356bae465a
- http://natashatherobot.com/swift-2-xcode-7-unit-testing-access/
- https://gist.github.com/jckarter/8f21f11ca46e67e6735a

https://github.com/sync/talks

“Share more. What you have learnt, and your code.”

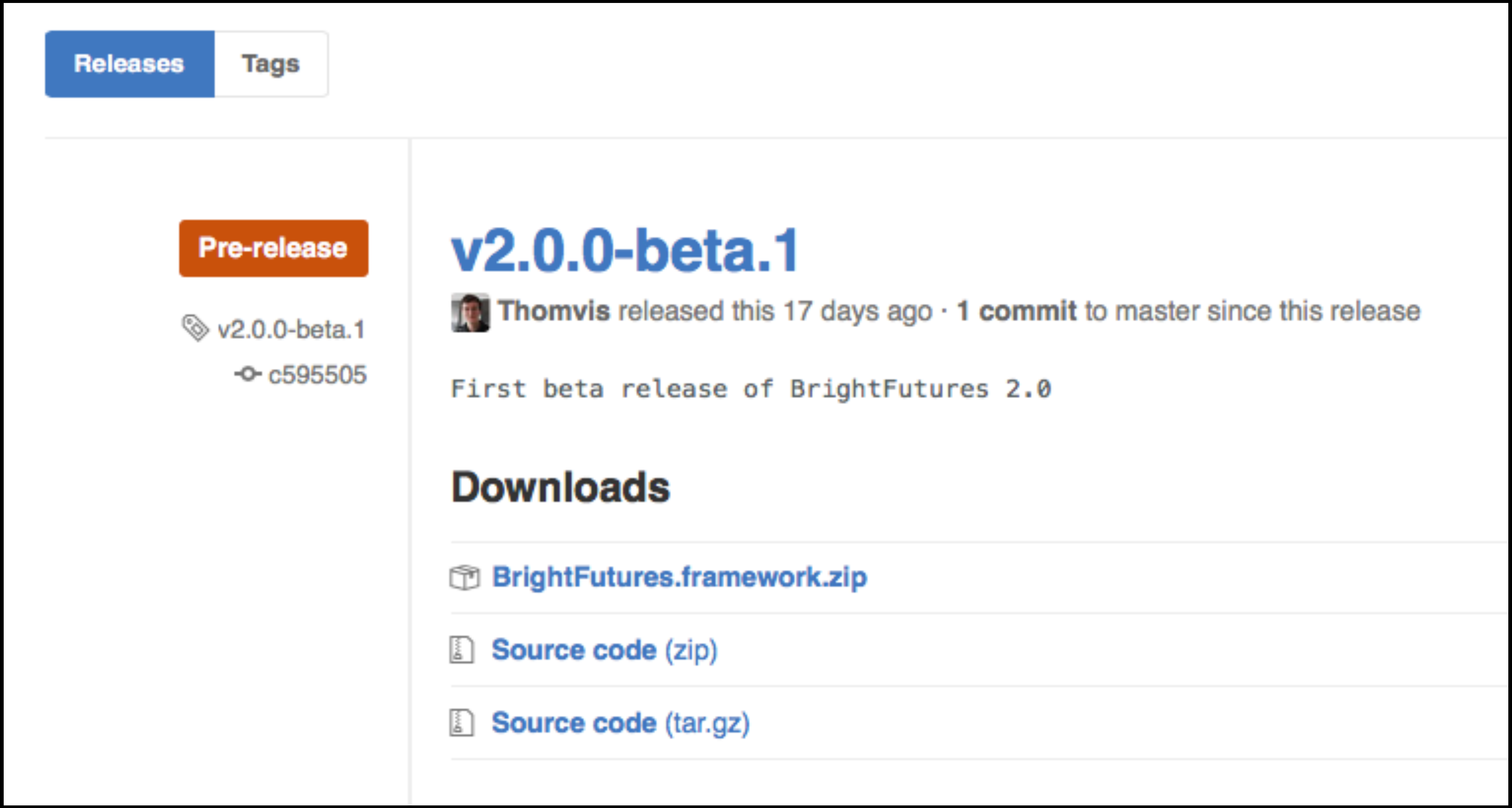
–Brian Gesiak

❤️

Carthage

Please tag your releases and attach precompiled framework

```
carthage build --no-skip-current
carthage archive BrightFutues
```



v2.0.0-beta.1
c595505

v2.0.0-beta.1

Thomvis released this 17 days ago · 1 commit to master since this release

```
First beta release of BrightFutures 2.0
```

Downloads

BrightFutures.framework.zip
Source code (zip)
Source code (tar.gz)

Questions ?