Functional programming in Swift

Michał Januszewski

Swift ?

WWDC 2014

LLVM

unctional patterns

Protocols and extensions on structs

Pattern matching

oncise syntax

Closures

Generics

Fast iteration

ative collections

Optional types

perator overloading

Object orientation

amespaces

Tuples

Type inference

lear mutability syntax

Read-Eval-Print-Loop (REPL)

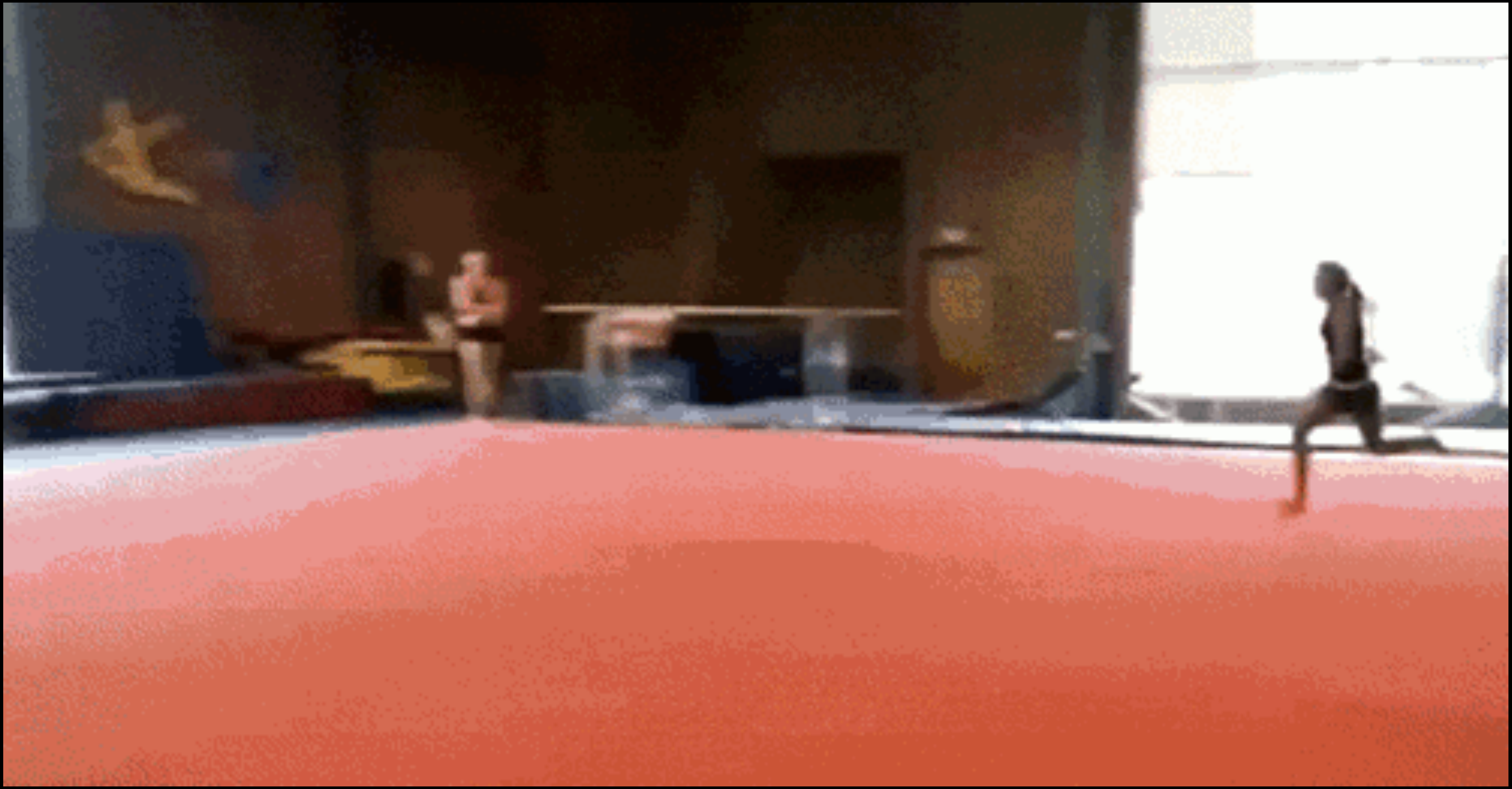
nteractive playground

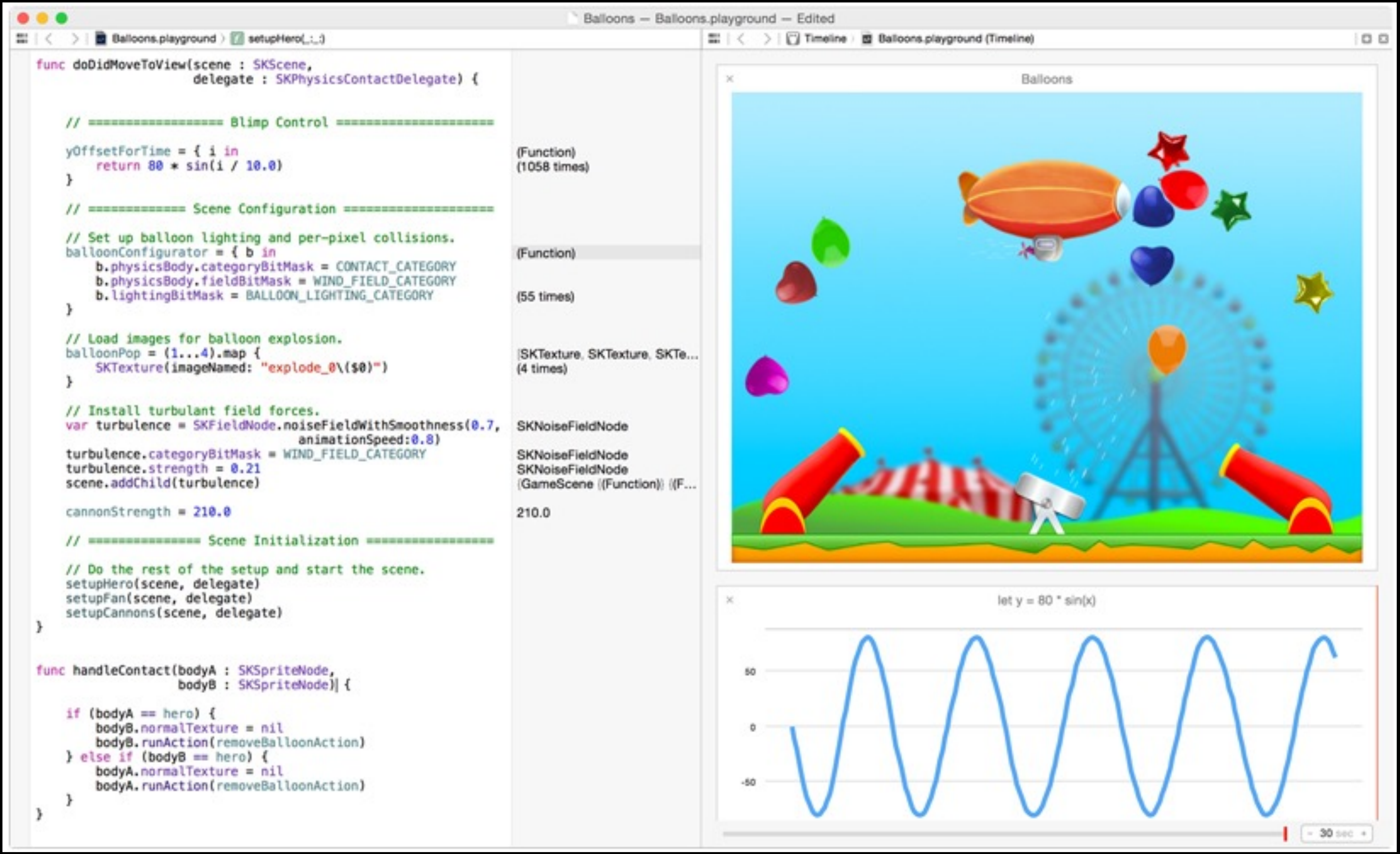
Multiple return types

Compile to native code

Swift !!!1one

4GIFs.com





```
func doDidMoveToView(scene : SKScene,
                     delegate : SKPhysicsContactDelegate) {


    // ================== Blimp Control =====================

    yOffsetForTime = { i in                                        (Function)
        return 80 * sin(i / 10.0)                                  (1058 times)
    }

    // ============= Scene Configuration ====================

    // Set up balloon lighting and per-pixel collisions.
    balloonConfigurator = { b in                                   (Function)
        b.physicsBody.categoryBitMask = CONTACT_CATEGORY
        b.physicsBody.fieldBitMask = WIND_FIELD_CATEGORY
        b.lightingBitMask = BALLOON_LIGHTING_CATEGORY              (55 times)
    }

    // Load images for balloon explosion.
    balloonPop = (1...4).map {                                     [SKTexture, SKTexture, SKTe...
        SKTexture(imageNamed: "explode_0\($0)")                    (4 times)
    }

    // Install turbulant field forces.
    var turbulence = SKFieldNode.noiseFieldWithSmoothness(0.7,     SKNoiseFieldNode
                                       animationSpeed:0.8)
    turbulence.categoryBitMask = WIND_FIELD_CATEGORY               SKNoiseFieldNode
    turbulence.strength = 0.21                                     SKNoiseFieldNode
    scene.addChild(turbulence)                                     {GameScene {(Function)} {(F...

    cannonStrength = 210.0                                         210.0

    // =============== Scene Initialization =================

    // Do the rest of the setup and start the scene.
    setupHero(scene, delegate)
    setupFan(scene, delegate)
    setupCannons(scene, delegate)
}


func handleContact(bodyA : SKSpriteNode,
                   bodyB : SKSpriteNode) {

    if (bodyA == hero) {
        bodyB.normalTexture = nil
        bodyB.runAction(removeBalloonAction)
    } else if (bodyB == hero) {
        bodyA.normalTexture = nil
        bodyA.runAction(removeBalloonAction)
    }
}
```

https://developer.apple.com/swift/

Agenda

- Closures
- High-order functions
- Laziness
- Currying
- Pattern matching
- Optionals

Closures

```
{ (params) -> returnType in statements }
```

```
func applyFunction(value: Int, f: Int -> Int) -> Int {
    return f(value)
}
```

```
applyFunction(2, { value in return value * 2 })
applyFunction(2, { value in value * 2 })
applyFunction(2, { $0 * 2 })
```

```
let a3 = applyFunction(2) { $0 * 2 }
```

http://fuckingclosuresyntax.com

High-order functions

```
let sports = [„swimming”, „cycling”, „running”]

let filtered = sports.filter { $0.hasPrefix("c") }
let capitalized = sports.map { $0.capitalizedString }
let numbers = [1,2,3,4,5]

let sum = numbers.reduce(0, combine: (+))

let product = numbers.reduce(1, combine: (*))
```

Laziness

Objective-C

```
@property (nonatomic, strong) NSMutableArray *posts;

- (NSMutableArray *)posts {
    if (!_posts) {
        _posts = [[NSMutableArray alloc] init];
    }
    return _posts;
}
```

Swift

```
lazy var posts = [Post]()
lazy var posts: [Post] = self.initPosts()

func initPosts() -> [Post] {
    let posts = [Post(„First post”), Post(„Second
post”)]
    return posts
}
```

Generators

```
for i in 1...5 {
  println(i)
}
```

```
var range = Range(start: 1, end: 5)
for i in range {
  println(i)
}
```

```
var sequence = Range(start: 1, end: 5)
var generator = sequence.generate()
while let i = generator.next() {
  println(i)
}
```

Custom generators

```
var count = 0;
var gen = GeneratorOf<Int> {
    count++
    return count >= 5 ? nil : count
}
for x in gen {
    println(x)
}
```

```
class Blog {
    private var posts: [Post]!

    init(posts: [Post]) {
        self.posts = posts
    }
}

for post in blog {
    println(post)
}
```

```
struct Post {
    let author: String
    let content: String

    func description() -> String {
        return "\(author): \(content)"
    }
}
```

```swift
struct Post {
    let author: String
    let content: String

    func description() -> String {
        return "\(author): \(content)"
    }
}

class Blog: SequenceType {
    private var posts: [Post]!

    init(posts: [Post]) {
        self.posts = posts
    }

    func generate() -> GeneratorOf<Post> {
        var nextIdx = posts.count-1
        return GeneratorOf<Post> {
            if (nextIdx < 0) {
                return nil
            }
            return self.posts[nextIdx--]
        }
    }
}
```

Currying

```swift
struct User {
    let name: String
    let password: String
    let age: Int

    static func create(name: String, password: String, age: Int) -> User {
        return User(name: name, password: password, age: age)
    }
}
```

```swift
struct User {
    let name: String
    let password: String
    let age: Int

    static func create(name: String, password: String, age: Int) -> User {
        return User(name: name, password: password, age: age)
    }
}

func curry<A,B,C,R>(f: (A,B,C) -> R) -> A -> B -> C -> R {
    return { a in { b in { c in return f(a,b,c) } } }
}
```

```swift
struct User {
    let name: String
    let password: String
    let age: Int

    static func create(name: String, password: String, age: Int) -> User {
        return User(name: name, password: password, age: age)
    }
}

func curry<A,B,C,R>(f: (A,B,C) -> R) -> A -> B -> C -> R {
    return { a in { b in { c in return f(a,b,c) } } }
}

let curried = curry(User.create)
let result = curried("A")("B")
let result2 = result(1)
```

Pattern matching

```
func fizzBuzz(number: Int) -> String {
    switch (number % 3, number % 5) {
    case (0, 0):
        // number divides by both 3 and 5
        return "FizzBuzz!"
    case (0, _):
        // number divides by 3
        return "Fizz!"
    case (_, 0):
        // number divides by 5
        return "Buzz!"
    case (_, _):
        // number does not divide by either 3 or 5
        return "\(number)"
    }
}
```

```
enum Status {
    case OnTime
    case Delayed(minutes: Int)
}

let goodNews = Status.OnTime
let badNews = Status.Delayed(minutes: 90)

class Train {
    var status = Status.OnTime
}
```

```
extension Train: Printable {

    var description: String {

        switch status {

        case .OnTime:
           return "On time"

        case .Delayed(let minutes) where 0...5 ~= minutes:
            return "Slight delay of \(minutes) min"

        case .Delayed(_):
            return "Delayed"

        }

    }

}
```

```
let trainOne = Train()
let trainTwo = Train()
let trainThree = Train()

trainTwo.status = .Delayed(minutes: 2)
trainThree.status = .Delayed(minutes: 8)
```

```
trainOne.description

trainTwo.description

trainThree.description
```

https://developer.apple.com/swift/blog/downloads/
Patterns.zip

Optionals

```
let dict = ["a": 100, "b": 200, "c": 300]

if let value = dict["d"] {
    println("Value: \(value)")
} else {
    println("Value for key d does not exits")
}
```

```
struct User {
    let name: String
    let password: String
    let age: Int

    static func create(name: String)(password: String)(age: Int) -> User {
        return User(name: name, password: password, age: age)
    }
}
```

```
let dict = ["name": "Foo", "password": "bar", "age": 30]
```

```
var user: User?

if let name = dict["name"] as? String {
    if let pass = dict["password"] as? String {
        if let age = dict["age"] as? Int {
            user = User.create(name, password: pass, age: age)
        }
    }
}
```

```
func GetString(value: AnyObject) -> String? {
    return value as? String
}

func GetInt(value: AnyObject) -> Int? {
    return value as? Int
}

func GetFloat(value: AnyObject) -> Float? {
    return value as? Float
}
```

```
infix operator >>> { associativity left precedence 150 }

func >>><A, B>(a: A?, f: A -> B?) -> B? {
    if let x = a {
        return f(x)
    } else {
        return .None
    }
}
```

fmap

```
infix operator <^> { associativity left }

func <^><A, B>(f: A -> B, a: A?) -> B? {
    if let x = a {
        return f(x)
    } else {
        return .None
    }
}
```

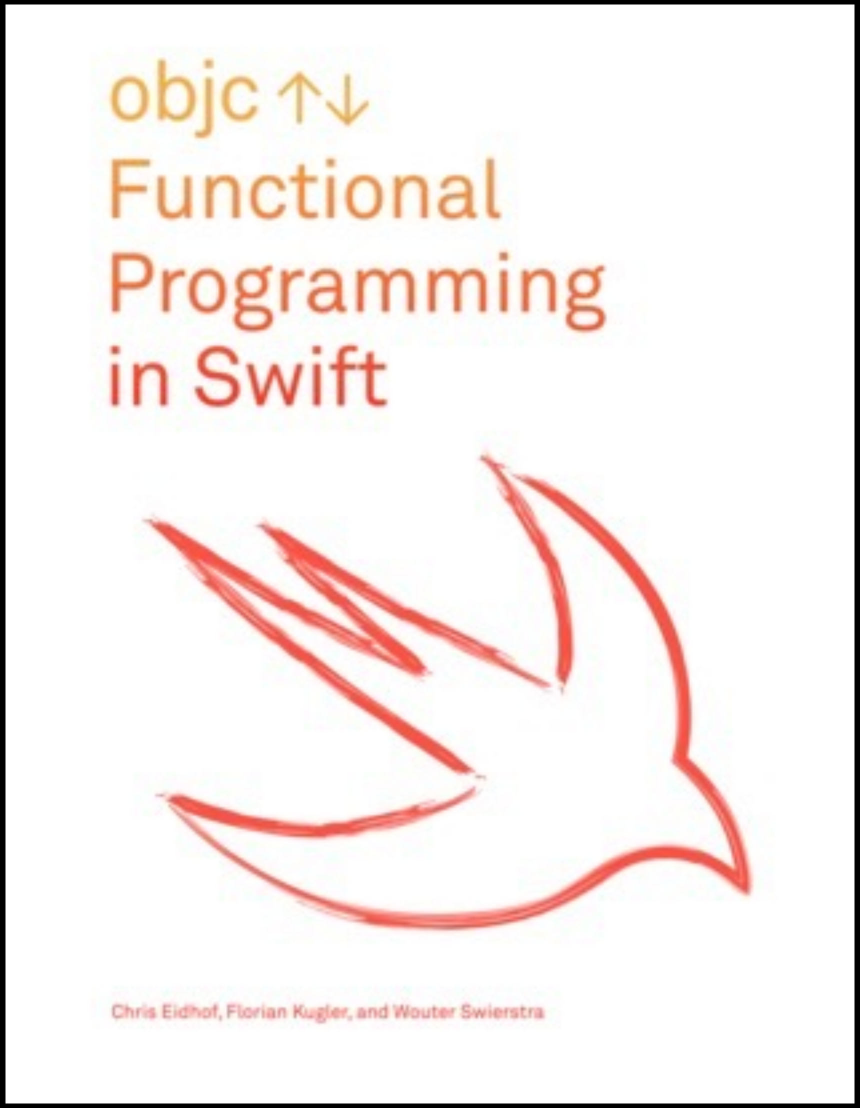
apply

```
infix operator <*> { associativity left }

func <*><A, B>(f: (A -> B)?, a: A?) -> B? {
    if let x = a {
        if let fx = f {
            return fx(x)
        }
    }
    return .None
}
```

```
let user = User.create <^>
         dict["name"] >>> GetString <*>
         dict["password"] >>> GetString <*>
         dict["age"] >>> GetInt
```

Pytania?



objc ↑↓

Functional Programming in Swift

Chris Eidhof, Florian Kugler, and Wouter Swierstra

Swift Warsaw

http://swiftwarsaw.com

Dziękuję za uwagę.

@mjanuszewski