The New Objective-C

A ***Swift***er approach.

Oscar Swanros, 2014

Oscar Swanros, 2014

What's new?

Functional patterns

Protocols and extensions on structs

Pattern matching

Concise syntax

Closures

Generics

Fast iteration

Native collections

Optional types

Operator overloading

Object orientation

Namespaces

Tuples

Type inference

Read-Eval-Print-Loop (REPL)

Clear mutability syntax

Multiple return types

Compile to native code

Interactive playground

Oscar Swanros, 2014

A lot.

Oscar Swanros, 2014

```
var a = ["a", "b"];
a.reverse();
a[0];
```

Oscar Swanros, 2014

```
var a = ["a", "b"]; // => ["a", "b"]
a.reverse(); // => ["b", "a"]
a[0]; // => ?
```

Oscar Swanros, 2014

```
var a = ["a", "b"]; // => ["a", "b"]
a.reverse(); // => ["b", "a"]
a[0]; // => Swift => "a", JS => "b"
```

Oscar Swanros, 2014

```
- (NSArray *)constraintsForImageView
{
    NSMutableArray *constraints = [NSMutableArray array];

    [constraints addObject:[NSLayoutConstraint constraintWithItem:self.contentView
                                                        attribute:NSLayoutAttributeTop
                                                        relatedBy:NSLayoutRelationEqual
                                                           toItem:_imageView
                                                        attribute:NSLayoutAttributeTop
                                                        multiplier:1.0
                                                          constant:0]];

    [constraints addObject:[NSLayoutConstraint constraintWithItem:_imageView
                                                        attribute:NSLayoutAttributeWidth
                                                        relatedBy:NSLayoutRelationEqual
                                                           toItem:self.contentView
                                                        attribute:NSLayoutAttributeWidth
                                                        multiplier:1.0
                                                          constant:1]];

    [constraints addObject:[NSLayoutConstraint constraintWithItem:_imageView
                                                        attribute:NSLayoutAttributeHeight
                                                        relatedBy:NSLayoutRelationEqual
                                                           toItem:nil
                                                        attribute:NSLayoutAttributeNotAnAttribute
                                                       multiplier:1.0
                                                          constant:160]];

    return constraints;
}
```

Oscar Swanros, 2014

```
func constraintsForImageView() -> [NSLayoutConstraint] {
    var constraints: [NSLayoutConstraint] = Array()

    constraints.append(
        NSLayoutConstraint(item: contentView, attribute: .Top, relatedBy: .Equal, toItem: imageView, attribute: .Top, multiplier: 1.0, constant: 0)
    )

    constraints.append(
        NSLayoutConstraint(item: imageView, attribute: .Width, relatedBy: .Equal, toItem: contentView, attribute: .Width, multiplier: 1.0, constant: 0)
    )

    constraints.append(
        NSLayoutConstraint(item: imageView, attribute: .Height, relatedBy: .Equal, toItem: nil, attribute: .NotAnAttribute, multiplier: 1.0, constant: 160)
    )

    return constraints
}
```

Oscar Swanros, 2014

Swift *wants* your code to crash less, be more expressive.

Oscar Swanros, 2014

1. Optionals

Oscar Swanros, 2014

Optionals Demo

```
class Person {
  var address: String?
}

let tim = Person()
tim.address = "150, Spear St., SF, CA"

tim.address // => {Some: "150, Spear St., SF, CA"}
tim.address! // => "150, Spear St., SF, CA"

// Verify that an optional holds a value
if let address = tim.address {
  println(address) // => "150, Spear St., SF, CA"
}
```

Two types of Optionals

- Wrapped optionals

```
var a: String? // => nil
a = "Are you entertained yet?" // => {Some: "Are you entertained yet?"}
a! // => "Are you entertained yet?"
```

- Explicitly unwrapped optionals

```
var b: String! // => nil
b = "Are you entertained yet?" // => "Are you entertained yet?"
b // => "Are you entertained yet?"
```

Oscar Swanros, 2014

2. Enums

Oscar Swanros, 2014

URL Manager (Raw Values)

```
var APIBase = "http://evilplaninc.com/api/admin/v1/"

enum AdminEndpoint: String {
    case Users = "users"
    case Notes = "notes"


    func URL() -> NSURL {
        switch self {
        case .Users:
            return NSURL(string: APIBase + self.rawValue)!

        case .Notes :
            return NSURL(string: APIBase + self.rawValue)!
        }
    }
}

let usersEndpoint = AdminEndpoint.Users.URL()
let notesEndpoint = AdminEndpoint.Notes.URL()
```

Oscar Swanros, 2014

URL Manager (Associated Values)

```
var APIBase = "http://evilplaninc.com/api/v1/"

enum CollectionEndpoint {
    case Friends(userId: Int)
    case Notes(userId: Int)

    func URL() -> NSURL {
        switch self {
        case .Notes(let userId):
            return NSURL(string: APIBase + "notes?user_id=\(userId)")!

        case .Friends(let userId):
            return NSURL(string: APIBase + "friends?user_id=\(userId)")!
        }
    }
}

let notesURL = CollectionEndpoint.Notes(userId: 5).URL()
let friendsURL = CollectionEndpoint.Friends(userId: 5).URL()
```

Oscar Swanros, 2014

3. Operators, Generics, Functional

Oscar Swanros, 2014

JSON

```
{
  "stat" : "ok",
  "blogs" : {
      "blog" : [
          {
            "needspassword" : true,
            "id" : 3,
            "name" : "Objective Swift",
            "url" : "http://objectiveswift.com"
          },
          {
            ...
          }
      ]
  }
}
```

Model

```
struct Blog {
  let id: Int,
  let name: String,
  let needsPassword: Bool
  let url: NSURL
}
```

1

```
func parseBlog(blog: [String:AnyObject]) -> Blog? {
    if let id = blog["id"] as? NSNumber{
        if let name = blog["name"] as? String {
            if let needspassword = blog["needspassword"] as? Bool {
                if let url = blog["url"] as? String {
                    return Blog(
                        id: id.integerValue,
                        name: name,
                        needsPassword: needspassword,
                        url: NSURL(string: url)!
                    )
                }
            }
        }
    }

    return nil
}
```

Oscar Swanros, 2014

Get a String out of a Dictionary

```
func string(input: [String:AnyObject], key: String) -> String? {
    return input[key] as? String
}
```

Oscar Swanros, 2014

Get values

```
func int(input: [String:AnyObject], key: String) -> Int? {
    return input[key] as? Int
}

func bool(input: [String:AnyObject], key: String) -> Bool? {
    return input[key] as? Bool
}

func array(input: [String:AnyObject], key: String) -> [AnyObject]? {
    let maybeAny: AnyObject? = input[key]
    return maybeAny as? [AnyObject]
}
```

Oscar Swanros, 2014

2

```
func parseBlog(blog: [String:AnyObject]) -> Blog? {
    if let id = int(blog, "id") {
        if let name = string(blog, "name") {
            if let needspassword = bool(blog, "needspassword") {
                if let url = string(blog, "url") {
                    return Blog(
                        id: id,
                        name: name,
                        needsPassword: needspassword,
                        url: NSURL(string: url)!
                    )
                }
            }
        }
    }

    return nil
}
```

Oscar Swanros, 2014

Convert data

```
func toURL(s: String) -> NSURL {
    return NSURL(string: s)!
}

func asDict(x: AnyObject) -> [String:AnyObject]? {
    return x as? [String:AnyObject]
}
```

Oscar Swanros, 2014

Curry

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

Oscar Swanros, 2014

```
func add(x: Int, y: Int) -> Int {
    return x + y
}

add(1, 2)


func curriedAdd(x: Int) -> Int -> Int {
  return { y in return x + y }
}

add(1)(2)
```

Oscar Swanros, 2014

Custom operators

```
func flatten<A>(x: A??) -> A? {
    if let y = x { return y }
    return nil
}

infix operator >>>= {}
func >>>=<A, B>(optional: A?, f: A -> B?) -> B? {
    return flatten(optional.map(f))
}

infix operator <*> { associativity left precedence 150 }
func <*><A, B>(left: (A->B)?, right: A?) -> B? {
    if let l1 = left {
        if let r1 = right {
            return l1(r1)
        }
    }

    return nil
}
```

Oscar Swanros, 2014

3

```
func parseBlog(blog: AnyObject) -> Blog? {

    let makeBlog = curry { id, name, needsPassword, url in
      Blog(id: id, name: name, needsPassword: needsPassword, url: url)
    }

    return asDict(blog) >>>= {
        makeBlog <*> int($0, "id")
                 <*> string($0, "name")
                 <*> bool($0, "needspassword")
                 <*> (string($0, "url") >>>= toURL)
    }
}
```

Oscar Swanros, 2014

Finishing up

```
func join<A>(elements: [A?]) -> [A]? {
    var result: [A] = []
    for element in elements {
        if let x = element {
            result += [x]
        }else{
            return nil
        }
    }

    return result
}
```

Oscar Swanros, 2014

Done!

```
func getBlogs() {
    let blogs: [Blog]? = dictionary(parsedJSON, "blogs") >>>= {
        array($0, "blog") >>>= {
            join($0.map(parseBlog))
        }
    }

    blogs
}
```

Oscar Swanros, 2014