Swift

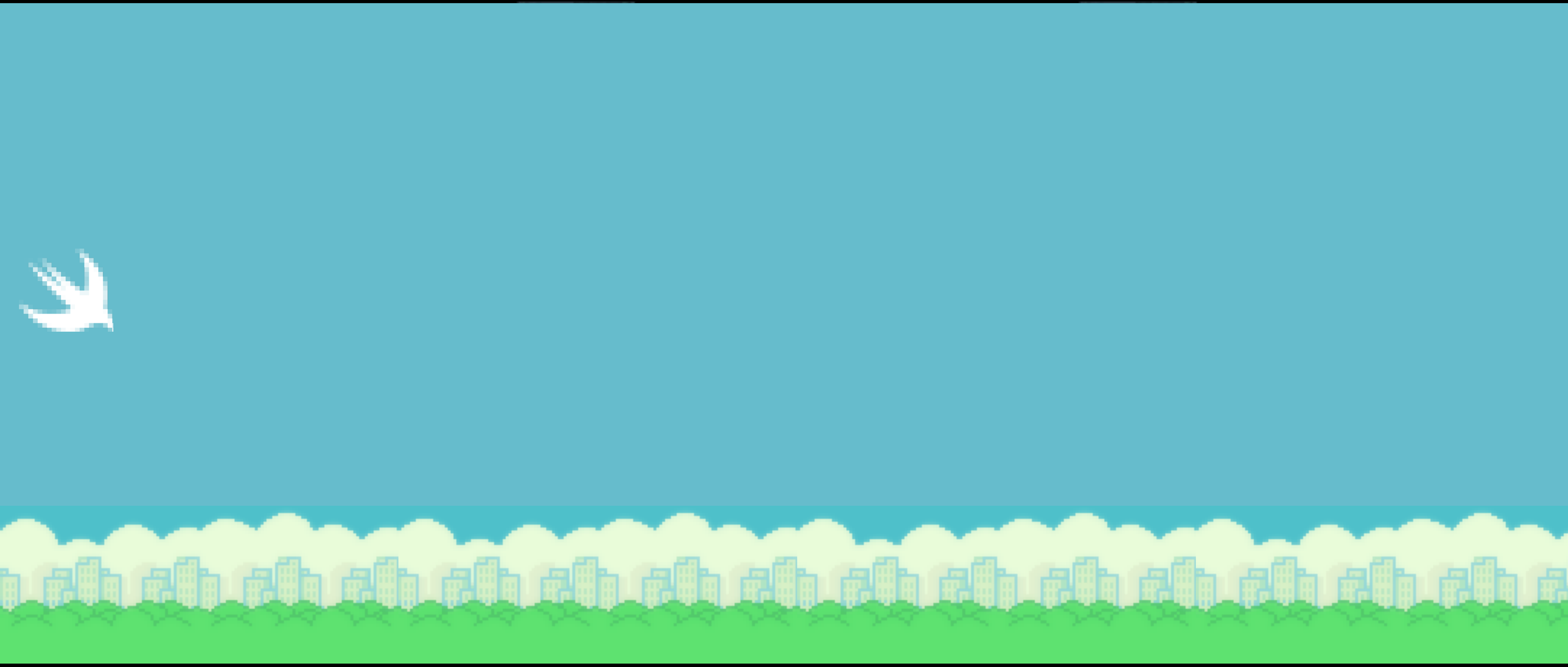
Erste Flugstunde

“Swift is a pragmatic, not a religious, language. It tries to get the defaults right, but allows you to change them if it doesn’t suit you.”

@clattner_llvm

“It’s designed to scale from “hello, world” to an entire operating system.”

The Swift Programming Language



```
class MyClass {
    var myProperty = 5.0

    func myFunc() -> Int {
        return 5
    }
}
```

```
struct Money: Printable {
    var value = 0.0
    var denomination: Denomination

    var description: String {
        return "\(value) \(denomination)"
    }
}

let moneten = Money(value:5, denomination:.Euro)

moneten
```

```
enum Denomination: Printable {
    case Dollar
    case Euro

    var description: String {
        switch self {
        case .Dollar:
            return "$"
        case .Euro:
            return "€"
        }
    }
}
```

```
class MyClass {
    var myProperty = 5.0

    func myFunc() -> Int {
        return 5
    }
}



struct Money: Printable {
    var value = 0.0
    var denomination: Denomination

    var description: String {
        return "\(value) \(denomination)"
    }
}

let moneten = Money(value:5, denomination:.Euro)

moneten
```

```
enum Denomination: Printable {
    case Dollar
    case Euro

    var description: String {
        switch self {
        case .Dollar:
            return "$"
        case .Euro:
            return "€"
        }
    }
}
```

```
class MyClass {
    var myProperty = 5.0

    func myFunc() -> Int {
        return 5
    }
}
```

```
enum Denomination: Printable {
    case Dollar
    case Euro

    var description: String {
        switch self {
        case .Dollar:
            return "$"
        case .Euro:
            return "€"
        }
    }
}
```

```
struct Money: Printable {
    var value = 0.0
    var denomination: Denomination

    var description: String {
        return "\(value) \(denomination)"
    }
}
```

```
let moneten = Money(value:5, denomination:.Euro)

moneten
```

```
class MyClass {
    var myProperty = 5.0

    func myFunc() -> Int {
        return 5
    }
}
```

```
struct Money: Printable {
    var value = 0.0
    var denomination: Denomination

    var description: String {
        return "\(value) \(denomination)"
    }
}
```

```
enum Denomination: Printable {
    case Dollar
    case Euro

    var description: String {
        switch self {
        case .Dollar:
            return "$"
        case .Euro:
            return "€"
        }
    }
}
```

```
let moneten = Money(value:5, denomination:.Euro)

moneten
```

“5.0 €”

Protocols

```
protocol InheritingProtocol: SomeProtocol, AnotherProtocol {
    // protocol definition goes here
}

protocol Container {
    typealias ItemType  // Associated Type
    mutating func append(item: ItemType)
    var count: Int { get }
    subscript(i: Int) -> ItemType { get }
}
```

Protocols sind Typen!

```
let things: Container[]
```

Extensions

```
extension Array {
    func first () -> Any? {
        return self[0]
    }
    func rest () -> Array {
        if self.count >= 1 {
            return Array(self[1..self.endIndex])
        } else {
            return []
        }
    }
}
```

Closures / Functions

```
var words = ["xyz", "abc", "def"]

words.sort({(a: String, b: String) -> Bool in  return a < b })

words.sort({a, b in a < b}) // "Type inference"
words.sort({ $0 < $1 })  // "Implicit argument names"
words.sort {$0 < $1}  //"Trailing argument"

func mySorter(a: String, b: String) -> Bool {
    return a < b
}

words.sort(mySorter)
```

func ist im Wesentlichen ein Closure mit Namen :)

Reference Cycles

Wie eh und je. Closures sind ARC-Objekte, die Teile ihrer Umgebung einfangen. Dazu wird auch kein __block benötigt, wenn man mal was verändern will im Closure.

Aber Vorsicht:

Reference Cycles are still a thing!

```
@lazy var someClosure: () -> String = {
        [unowned self] in
        return self.whatever()
    }
```

ie Lorriman/ Solent News

switch / enum / Pattern matching

```
struct Train {
    var state: TrainState = .OnTime
}

enum TrainState {
    case OnTime
    case Delayed(Int)
}
```

switch / enum / Pattern matching

```
struct Train {
    var state: TrainState = .OnTime
}

enum TrainState {
    case OnTime
    case Delayed(Int)
}
```

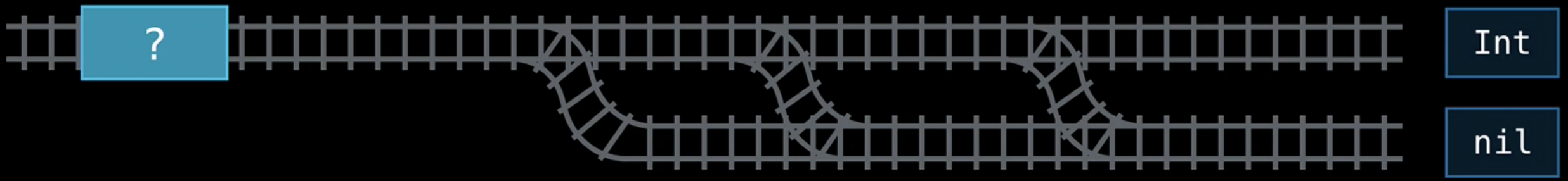
```
let train = Train(state:.Delayed(10))

switch train.state {
    case .OnTime:
        println("cool")
    case .Delayed(1):
        println("nearly on time")
    case .Delayed(2..10):
        println("almost here, I swear")
    case .Delayed(_):
        println("it’ll get here when it’ll get here")
}
```

Optionals

Können einen Wert haben, oder auch “nix”.
Müssen “entpackt” werden, um an den Inhalt zu kommen.

Optional Chaining

```
addressNumber = paul.residence?.address?.buildingNumber?.toInt()
```



Non-Optionals (also alles ohne “?”) haben garantiert immer einen Wert, nie nil!


© Fran Trabal

Optionals + Pattern Matching

```
func parsePlist(list: Dictionary<String, AnyObject>) -> () {
    switch (list["name"], list["population"], list["abbr"]) {
    case (.Some(let listName as NSString),
          .Some(let pop as NSNumber),
          .Some(let abbr as NSString))
        where abbr.length == 2:
            // do something with listName, pop, abbr
            println("\(listName) \(pop) + \(abbr)")
    default:
        println("nothing found")
    }
}
```

Generics

```
func bla<T>(someThing: T) -> T { return someThing }
func swap<T>(inout a: T, inout b: T) { /* not /* swapping */ anything */ }
func indexOf<T: Equatable>(sought: T, inArray array:[T]) -> Int? {
    for i in 0..array.count {
        if array[i] == sought { return i }
    }
    return nil
}
```

Memoization

```
func memoize<T: Hashable, U>( body: (T) -> U ) -> (T) -> U {
    var memo = Dictionary<T, U>()
    return { x in
        if let q = memo[x] { return q}
        let r = body(x)
        memo[x] = r
        return r
    }
}
```

Fun stuff

```
extension String {
    subscript (i: Int) -> String {
        return String(Array(self)[i])
    }
}

struct Thing {
    var name = "Apple"
}

extension Thing {
    var nameWithArticle: String {
    switch name[0] {
    case "a", "e", "i", "o", "u":
        return "an \(name)"
    default:
        return "a \(name)"
        }
    }
}
```

```
// zuerst muss man den operator erstellen
operator infix ~ {}

func ~ (decorator: (Thing) -> String,
    object: Thing) -> String {
        return decorator(object)
} // globale Funktion!

func an(object: Thing) -> String {
    return object.nameWithArticle
}


let obst = Thing(name: "orange")
let gemüse = Thing(name: "tomato")

"Let's eat \(an ~ obst)”
// “Let’s eat an orange”

"Let's eat \(an ~ gemüse)”
// “Let’s eat a tomato”
```

Fun stuff

Currying

```
func addTwoNumbers(a: Int)(b: Int) -> Int {
    return a + b
}

func addTwoNumbers(a: Int) -> (Int -> Int) {
    func addTheSecondNumber(b: Int) -> Int {
        return a + b
    }
    return addTheSecondNumber
}
```

addTwoNumbers(4)(5) // Returns 9

Swift ist noch β (oder α?)

Performance ist noch nicht “optimal”. -Ofast hilft, aber ändert Semantik der Sprache (z.B. keine Overflow und Out-of-bounds checks mehr) — Compiler ist noch buggy und nicht genug optimiert.

NSString vs. String

```
let bla:String = “\U0001F496"          //💖
let nsbla:NSString = “\U0001F496"      //💖

nsbla.length                           // 2
countElements(bla)                     // 1
```

@selector() vs Selector()

geht, aber ist nicht Swiftig

Access Modifier fehlen (private, etc.)

Regex-Literale fehlen

KVO?? “Information forthcoming.”

Xcode 6 Beta 2: 9 Seiten Known Issues

Mehr coole Sachen: @IBInspectable, @IBDesignable

http://www.weheartswift.com/make-awesome-ui-components-ios-8-using-swift-xcode-6/

Swift Mailingliste @ Google Groups

https://groups.google.com/forum/#!forum/swift-language

Swift Radars

http://swiftradar.tumblr.com

Terrible Swift Ideas

http://terribleswiftideas.tumblr.com

Jede Menge Swift-Session-Videos!

Protip: Alt+Klick auf Vorspulen in QuickTime → mit Tempo z.B 1,5x schauen 👍🏻