iOS at RetailMeNot

Adam Chalmers

@adam_chal
github.com/adamchalmers/
achalmers@rmn.com
About Me

- University of Sydney, philosophy and CS degrees
- Did one semester of UT in 2013
  - Couldn’t get into this class… damn waitlist
- “Bit of everything” programmer
  - Formerly game programmer
  - Web dev
  - Android
  - Backend/server
  - Now iOS
  - Rust?? Haskell?? Something else???
- Austin/RetailMeNot for a year
Why I love mobile

- Huge growth
  - Especially the next billion users
- Huge demand for developers
  - There’s 3 generations of Java/C++ programmers
  - There’s only 1 generation of iPhone/Android programmers
- Sensors and hardware
- Intimately integrated into the user’s life
- User intent
Why I hate mobile

- Huge number of APIs
- Best practices constantly changing
- Wide range of devices
  - CPU
  - Firmware
  - Screen size
- Slow networks
- Code duplication
Mobile at RetailMeNot

- Core team of twenty across iOS and Android
  - iPad app discontinued - too much work, not enough users
- Mweb sits with web, not mobile
- Critical for strategy
  - People always have their phones when buying things
- Autonomy
- Server sends us ViewModels
Why Swift?

- Powerful type system
  - No NullPointerExceptions!
- Immutable variables/objects
- Protocols and composition
- Tooling... sort of
- Server language (thanks IBM)
- Android language (experimental)
General advice

- Small objects and functions are easier to read, debug and unit test!
- Learn the languages that inspired it
  - Typesystem from Haskell
  - Protocol-based from Go
  - Object orientation from Java
  - Functional programming from Haskell/Lisp
- Learn to write testable code
  - Web developers can deploy bugfixes in 2 minutes
  - Mobile developers can deploy bugfixes in 2 days
- Know your platform’s UI/UX patterns
- Books, talks, conferences
- INTERNSHIPS
- 80% of your job is communicating
**Idiomatic Swift: optionals**

```swift
func find(array: [Int], target: Int) -> Int {
    for (i, elem) in array.enumerated {
        if elem == target {
            return i
        }
    }
    return -1
}
```

```swift
let i = find(array, 123)
array[i] *= 2
```

```swift
func find(array: [Int], target: Int) -> Int? {
    for (i, elem) in array.enumerated {
        if elem == target {
            return i
        }
    }
    return nil
}
```

```swift
if let i = find(array, 123) {
    array[i] *= 2
}
```
struct Person {
    mother: Person?
}

extension Person {
    var grandMother: Person? {
        if let mom = self.mother {
            return mom.mother
        }
    }
}

struct Person {
    mother: Person?
}

extension Person {
    var grandMother: Person? {
        return self.mother?.mother
    }
}
Idiomatic Swift: optionals and flatMap

```swift
struct Person {
    mother: Person?
}

func askParent(person: Person)

if let mom = adam.mother {
    askParent(mom)
}

adam.mother.flatMap { m in askParent(m) }
adam.mother.flatMap { askParent($0) }
```
func findMoms(people: [Person]) -> [Person] {
    var moms = [Person]()
    for person in people {
        if let mom = person.mother {
            moms.append(mom)
        }
    }
    return moms
}

func findMoms(people: [Person]) -> [Person] {
    return people.compactMap { $0.mother }
}
func find(array: [Int], target: Int) -> Int? {
    for (i, elem) in array.enumerated {
        if elem == target {
            return i
        }
    }
    return nil
}

if let i = find(array, 123) {
    array[i] *= 2
}
Idiomatic Swift: enums to represent state

```swift
func login(json: JSON) -> (String, String)

enum HTTPError {
    case noUserFound
    case wrongPassword
    case networkDown
}

func login(json: JSON) -> (HTTPError?, String)
```
Idiomatic Swift: enums to represent state

```swift
enum HTTPError {
    case noUserFound
    case wrongPassword
    case networkDown
}

func login(json: JSON) -> (HTTPError?, String)

enum HTTPResponse {
    case noUserFound
    case wrongPassword
    case networkFail(Int)
    case ok(UserAccount)
}

func login(json: JSON) -> HTTPResponse
```
Idiomatic Swift: enums to represent state

class CouponView: UITableView {
    var hasLoaded: Bool
    var hasDisplayed: Bool
    var networkDataLoaded: Bool
    var networkData: [Coupon]?
}

class CouponView: UITableView {
    enum State {
        case loaded
        case displayed
        case waitingForNetwork
        case networkLoaded([Coupon])
    }
    var state: State
}

Thank you!

Adam Chalmers

@adam_chal
github.com/adamchalmers
achalmers@rmn.com