ANYTHING YOU CAN DO
I CAN DO BETTER.

OPEN SOURCED

HTTPS://GITHUB.COM/KICKSTARTER/IOS-OSS
HTTPS://GITHUB.COM/KICKSTARTER/ANDROID-OSS

KOTLIN & SWIFT

KOTLIN

- JVM LANGUAGE
- BUILT BY JETBRAINS
- 100% INTEROP WITH JAVA
- OOP WITH A BIT OF FP
- VERY EXPRESSIVE

OPTIONALS

```
1
2 let xs = [1, 2, 3, 4]
3
4 xs.first + 1
5
6
7
```

OPTIONALS

```
10
11      val x: Int? = null
12      val y: Int = null
13
14      val xs = listOf(1, 2, 3, 4, 5)
15
16      xs.firstOrNull() + 1
17
18
```

STRUCTS AND ENUMS

```
1
2  struct User {
3    let bio: String
4    let id: Int
5    let name: String
6  }
7
8  enum Either<A, B> {
9    case left(A)
10   case right(B)
11 }
12
13 let user = User(bio: "Hello world", id: 1, name: "Blob")
14 let intOrString = Either<Int, String>.left(2)
15
```

DATA CLASSES AND SEALED CLASSES

```
1
2   data class User(
3     val bio: String,
4     val id: Int,
5     val name: String
6   )
7
8   sealed class Either<out A, out B> {
9     data class Left<out A>(internal val left: A) : Either<A, Nothing>()
10    data class Right<out B>(internal val right: B) : Either<Nothing, B>()
11  }
12
13  val user = User(bio = "Hello world", id = 1, name = "Blob")
14  val intOrString = Either.Left(1)
15
```

EXTENSIONS, CLOSURES AND DESTRUCTURING

```
12
13 extension Either {
14   func map<C>(f: (B) -> C) -> Either<A, C> {
15     switch self {
16     case let .left(value):  return .left(value)
17     case let .right(value): return .right(f(value))
18     }
19   }
20 }
21
22 Either<String, Int>.right(2).map { $0 * $0 }
23
```

EXTENSIONS, CLOSURES AND DESTRUCTURING

```
5
6   fun <A, B, C> Either<A, B>.map(f: (B) -> C): Either<A, C> {
7       return when(this) {
8           is Either.Left -> Either.Left(this.left)
9           is Either.Right -> Either.Right(f(this.right))
10      }
11  }
12
13  Either.Right(2).map { it * it }
14
```

EXTENSIONS. CLOSURES AND DESTRUCTURING

EVEN BETTER...

```
7
8   fun <A, B, C> Either<A, B>.map(f: (B) -> C): Either<A, C> = when(this) {
9     is Either.Left -> Either.Left(this.left)
10    is Either.Right -> Either.Right(f(this.right))
11  }
12
```

OPERATORS

```
2
3  infix operator >>>
4
5  func >>> <A, B, C> (f: @escaping (A) -> B,
6                      g: @escaping (B) -> C) -> (A) -> C {
7    return { g(f($0)) }
8  }
9
10 func incr(_ x: Int) -> Int { return x + 1 }
11 func square(_ x: Int) -> Int { return x * x }
12
13 let f = incr >>> square >>> incr
14
15 Array(1...10).map(f)
16
```

OPERATORS

```
2   infix fun <A, B, C> ((A) -> B).andThen(g: (B) -> C): (A) -> C {
3     return { g(this(it)) }
4   }
5
6   fun Int.incr(): Int {
7     return this + 1
8   }
9
10  fun Int.square(): Int {
11    return this * this
12  }
13
14  val f = Int::incr andThen Int::square andThen Int::incr
15
16  1.rangeTo(10).map(f)
```

TAIL RECURSION

KOTLIN

```
tailrec fun sum(xs: List<Int>, total: Int = 0): Int {
  val first = xs.firstOrNull()
  return if(first == null) total else sum(xs.drop(1), first + total)
}
```

TAIL RECURSION

SWIFT 😭

```
1
2 func sum(_ xs: ArraySlice<Int>, total: Int = 0) -> Int {
3   guard let first = xs.first else { return total }
4   return sum(xs.dropFirst(), total: first + total)
5 }
6
```

FUNCTIONAL PROGRAMMING

FUNCTIONAL PROGRAMMING

```
return SignalProducer.concat(
  recommendedProjects,
  similarToProjects,
  staffPickProjects
  )
  .filter { $0.id != project.id }
  .uniqueValues { $0.id }
  .take(first: 3)
  .collect()
```

```
return Observable.concat(
   recommendedProjects,
   similarToProjects,
   staffPickProjects
   )
   .filter { it !== project }
   .distinct()
   .take(3)
   .toList()
```

ANYTHING YOU CAN DO
WE CAN DO TOGETHER

@MBRANDONW

@LUOSER