Passing functions to function arguments

Yoshikuni Kato

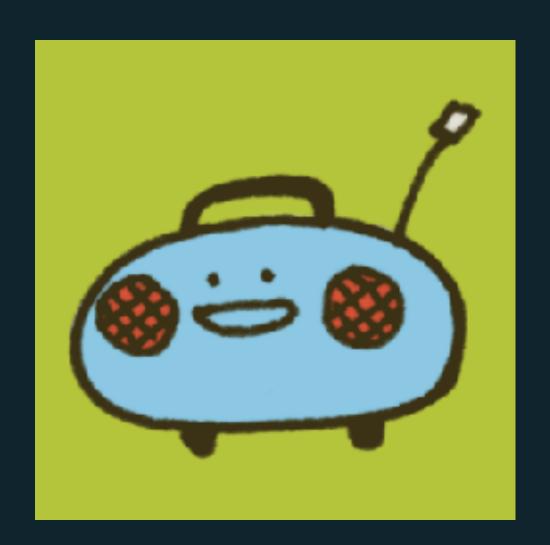
Notice

This is the English version of "関数を引数として渡す書き方のポイント" which was presented at iOSDC 2017 (2017/09/17)

— https://speakerdeck.com/yoching/guan-shu-woyin-shu-tositedu-sushu-kifang-falsepointo

Who am I?

- Yoshikuni Kato (加藤由訓) @yoshikuni_kato
- iOS Engineer (3 years)
- Yahoo! Japan -> Ohako, inc.
- "Radi-Hey" →
- Interests: Class Design / FRP / Coordinator Pattern / UI implementation



Wave of writing functionally

- More opportunity to write functionally
 - FRP (RxSwift / ReactiveSwift)
 - map / filter / reduce
- reducing if and for
- more declarative way

What I talk?

- how to write functions as arguments of other functions
 - -> a little change, much declarative way
- example: map of array

How to write #1: Write closures directly

```
let array: [Int] = [1, 2, 3]
array.map { number -> Int in
    return number * 2
}
```

Definition of map of array

```
func map<T>(_ transform: (Element) throws -> T) rethrows -> [T]
```

- arguments of map: closure that takes Element and return T
- functions are like "named closures"
- possible to pass function directly

How to write #2: Pass function

```
// declare function first
func twoTimes(of number: Int) -> Int {
  return number * 2
}
let array: [Int] = [1, 2, 3]
array.map(twoTimes) // pass the function
```

In case of multiple parameters

```
func someFunc(a: Int, b: Int) -> String {
    return "a = \backslash(a), b = \backslash(b)"
let array: [Int] = [1, 2, 3]
array
    .map { number -> (a: Int, b: Int) in
        return (a: number, b: number) // make a tuple
    .map(someFunc)
```

In case of initializer

```
struct Sample {
    let number: Int
    init(number: Int) {
        self.number = number
    }
}
```

How to write #1: Write closure directly

```
let array: [Int] = [1, 2, 3]
array.map { number -> Sample in
    return Sample(number: number)
}
```

How to write #2: Pass function

```
let array: [Int] = [1, 2, 3]
array.map(Sample.init)
```

— initializer(.init) = function which returns the object

Comparison 1

```
array.map { number -> Sample in
    return Sample(number: number)
}
array.map(Sample.init)
```

Comparison 2

```
array
  .map { number -> Int in
      return number * 2
  .map { number -> Sample in
    return Sample(number: number)
  .map { sample -> Foo in
    return Foo(sample: sample)
array
  .map(twoTimes)
  .map(Sample.init)
  .map(Foo.init)
```

Wrap up

- a little change, much declarative way
- example: convert Model to ViewModel

```
model.map(ViewModel.init)
```

- extracting logics as methods, as a result
- feeling of passing function (different from imperative programming style)

References

- Connecting View Controllers, Swift Talk¹
- From Runtime Programming to Functions, Swift Talk²

¹ https://talk.objc.io/episodes/S01E05-connecting-view-controllers

² https://talk.objc.io/episodes/S01E19-from-runtime-programming-to-functions

One more thing

```
func someFunc(a: Int, b: Int) -> String {
    return "a: \(a), b: \(b)"
// cannot pass tuples to function itself
let parameters = (a: 0, b: 0)
someFunc(parameters) // 🎕 (swift3~)
// can pass tuples when using map
let array: [(Int, Int)] = [(0, 0)]
array.map(someFunc) // 🔞 (even in swift3)
```

— Please tell me if you know how this is possible