Advanced

Higher-Order Function

Realism Programmer

송치원 (곰튀김)

iamchiwon.github.io

Function as Value

Value

```
42
```

```
let n = 42
```

```
type(of: n)
// Int
```

```
let n: Int = 42
```

```
21 + n
```

Function

```
func function(_ i: Int) -> Int {
    return i + 42
}
```

```
let f = function
```

```
type(of: f)
// (Int) -> Int
```

```
let f: (Int) -> Int = function
```

```
let f: (Int) -> Int = { i in
    return i + 42
}
```

```
f(21)
```

Value

```
func output(_ i: Int) {
    print(i)
}

output(n)
```

```
func getN() -> Int {
    let n = 42
    return n
}
```

Function

```
func output(_ i: Int,
            _ m: (Int) -> Int)
{
    print(m(i))
}

output(21, f)
```

```
func getF() -> (Int) -> Int {
    let f = { $0 + 42 }
    return f
}
```

Higher-Order Functions

```
func fs1(_ f:                                    ) {
}
```

```
func fs2() ->                                    {
}
```

Function stores acts

Sync Action

```swift
@discardableResult
func createView<V : UIView>(_ view: V,
                            _ parent: UIView?,
                            _ setter: (V) -> Void) -> V {
    parent?.addSubview(view)
    setter(view)
    return view
}
```

```swift
createView(UIImageView(), view) { iv in
    iv.contentMode = .scaleAspectFill
    iv.clipsToBounds = true
    iv.layer.cornerRadius = 16
    iv.frame = CGRect(x: 0, y: 0, width: 100, height: 100)
}
```

Async Action

```
func downloadImage(from url: URL,
                   completion: @escaping (UIImage?) -> Void)
{
    DispatchQueue.main.async {
        if let image = try? UIImage(data: Data(contentsOf: url)) {
            completion(image)
        } else {
            completion(nil)
        }
    }
}
```

```
downloadImage(from: URL(string: "image download url")!,
              completion: { img in
                  self.imageView.image = img
              })
```

Closure

```
extension UIView {
    @available(iOS 4.0, *)
    open class func animate(withDuration duration: TimeInterval,
                            animations: @escaping () -> Void,
                            completion: ((Bool) -> Void)? = nil)
}
```

```
{
    let fadeOutAlpha: CGFloat = 0

    UIView.animate(withDuration: 0.3,
                   animations: {
                       self.button.alpha = fadeOutAlpha
                   },
                   completion: { completed in
                       self.button.isHidden = true
                   })
}
```

General Async Function

Generalize

```
func [redacted]([redacted]) {
    DispatchQueue.main.async {
        [redacted]
        [redacted]
    }
}
```

```
func [redacted]([redacted]([redacted]) -> Void) {
    DispatchQueue.main.async {
        [redacted]([redacted])
    }
}
```

General Async Function

```
func gaFunction<T>(task: @escaping ((@escaping (T) -> Void) -> Void),
                   execute: @escaping (T) -> Void)
{
    DispatchQueue.main.async {
        task(execute)
    }
}
```

```
gaFunction(task: { f in
    let url = URL(string: "image download url")!
    let image = try! UIImage(data: Data(contentsOf: url))
    f(image)
}, execute: { img in
    self.imageView.image = img
})
```

General Async Function Class

```
class GAFunction<T> {

    private let task: ((@escaping (T) -> Void) -> Void)

    init(task: @escaping (@escaping (T) -> Void) -> Void) {
        self.task = task
    }

    func execute(_ f: @escaping (T) -> Void) {
        DispatchQueue.main.async {
            self.task(f)
        }
    }
}
```

GAFunction class

```
class GAFunction<T> {

    private let task: ((@escaping (T) -> Void) -> Void)

    init(task job: @escaping (@escaping (T) -> Void) -> Void) {
        task = job
    }

    func execute(_ f: @escaping (T) -> Void) {
        DispatchQueue.main.async {
            self.task(f)
        }
    }
}
```

```
let gaf = GAFunction<UIImage?>(task: { f in
    let url = URL(string: "image download url")!
    let image = try! UIImage(data: Data(contentsOf: url))
    f(image)
})

gaf.execute({ img in
    self.imageView.image = img
})
```

Async Value

```
extension GAFunction {
    convenience init(just t: T) {
        self.init(task: { f in f(t) })
    }

    convenience init(from ts: [T]) {
        self.init(task: { f in ts.forEach(f) })
    }
}
```

```
GAFunction(just: 42)
    .execute({ i in
        print(i)                                    //42
    })

GAFunction(from: [42, 41, 40])
    .execute({ i in                                 //42
        print(i)                                    //41
    })                                              //40
```

Filter

```
extension GAFunction {
    func filter(_ filter: @escaping (T) -> Bool) -> GAFunction<T> {
        return GAFunction<T>(task: { f in
            self.execute({ t in
                if filter(t) {
                    f(t)
                }
            })
        })
    }
}
```

```
GAFunction(from: [42, 41, 40, 39, 38])
    .filter({ $0 % 2 == 0 })
    .execute({ i in                                    //42
        print(i)                                       //40
    })                                                 //38
```

Map

```
extension GAFunction {
    func map<U>(_ mapper: @escaping (T) -> U) -> GAFunction<U> {
        return GAFunction<U>(task: { f in
            self.execute({ t in
                f(mapper(t))
            })
        })
    }
}
```

```
GAFunction(from: [42, 41, 40])
    .map({ $0 * 10 })
    .execute({ i in                                    //420
        print(i)                                       //410
    })                                                 //400
```

FlatMap

```
extension GAFunction {
    public func flatMap<U>(_ mapper: @escaping (T) -> GAFunction<U>)
                                                    -> GAFunction<U>
    {
        return GAFunction<U>(task: { f in
            self.execute({ t in
                mapper(t).execute({ u in f(u) })
            })
        })
    }
}
```

```
GAFunction(from: [42, 41, 40])
    .flatMap({ i -> GAFunction<Float> in
        GAFunction(just: Float(i) / 3)
    })
    .execute({ f in                                    //14.0
        print(f)                                       //13.666667
    })                                                 //13.333333
```

All together

```
func downloadImage(_ url: URL) -> GAFunction<UIImage?> {
    return GAFunction(task: { completion in
        if let image = try? UIImage(data: Data(contentsOf: url)) {
            completion(image)
        } else {
            completion(nil)
        }
    })
}



GAFunction(just: "image url")
    .map({ URL(string: $0) })
    .filter({ $0 != nil })
    .map({ $0! })
    .flatMap(downloadImage)
    .execute({ img in
        self.imageView.image = img
    })
```

All together

```
func downloadImage(_ url: URL) -> GAFunction<UIImage?> {
    return GAFunction(task: { completion in
        if let image = try? UIImage(data: Data(contentsOf: url)) {
            completion(image)
        } else {
            completion(nil)
        }
    })
}



GAFunction(just: "image url")
    .map({ URL(string: $0) })
    .filter({ $0 != nil })
    .map({ $0! })
    .flatMap(downloadImage)
    .execute({ img in
        self.imageView.image = img
    })
```

Summary

1. 함수는 value 처럼
   - 변수에 담을 수 있다
   - 파라미터로 넘길 수 있다
   - 리턴값으로 반환될 수 있다

2. 고차함수(Higher-Order Function)
   - 함수가 파라미터로 쓰이거나 반환값으로 사용되는 경우
   - 전달된 함수가 본체의 수행이 종료된 후에 사용되는 경우 @escaping을 해줘야 한다.
   - optional function 은 @escaping이 기본이다.

3. 함수는 행동을 저장한다
   - 저장한 행동은 전달될 수 있다.
   - 저장된 행동은 나중에 사용될 수 있다.

4. 이러한 특성들로 여러 재밌는 것들을 만들 수 있다.
   - 사례. GAFunction

5. 이미 이러한 성질을 이용해서 범용으로 만들어진 라이브러리가 있다.
   - RxSwift 를 쓰자

광고

programmers