Closure

강사 주영민

Fast campus

일급 객체

컴퓨터 프로그래밍 언어 디자인에서, 특정 언어의 **일급 객체** (first-class citizens, 일급 값, 일급 엔티티, 혹은 일급 시민)이라 함은 일반적으로 다른 객체들에 적용 가능한 연산을 모두 지원하는 객체를 가리킨다. 함수에 매개변수로 넘기기, 변수에 대입하기와 같은 연산들이 여기서 말하는 일반적인 연산의 예에 해당한다.

다음과 같은 조건을 만족할 때 일급 객체라고 말할 수 있다.

- 변수나 데이터 구조안에 담을 수 있다.
- 파라미터로 전달 할 수 있다.
- 반환값(return value)으로 사용할 수 있다.
- 할당에 사용된 이름과 관계없이 고유한 구별이 가능하다.
- 동적으로 프로퍼티 할당이 가능하다.

- wikipedia 에서 -

**스위프트의 함수는 일급 객체이다

Fast campus

중첩함수

- 함수 내부에서 함수를 정의해서 사용 되는 함수
- 외부에는 숨겨져 있으며 선언된 함수 내부에서만 호출 가능하다.

```
func chooseStepFunction(backward: Bool) -> (Int) -> Int {
    func stepForward(input: Int) -> Int { return input + 1 }
    func stepBackward(input: Int) -> Int { return input - 1 }

    return backward ? stepBackward : stepForward
}
```

중첩함수

```
func chooseStepFunction(backward: Bool) -> (Int) -> Int {
    func stepForward(input: Int) -> Int { return input + 1 }
    func stepBackward(input: Int) -> Int { return input - 1 }

    return backward ? stepBackward : stepForward
}
```

```
var currentValue = -4
let moveNearerToZero = chooseStepFunction(backward: currentValue > 0)
// moveNearerToZero now refers to the nested stepForward() function
while currentValue != 0 {
    print("\(currentValue)... ")
    currentValue = moveNearerToZero(currentValue)
}
```

클로저

Closures are self-contained **blocks of functionality** that can be passed around and used in your code. 클로저 는 코드에서 전달되고 사용할 수있는 독립적 인 기능 블록입니다.

Closures can **capture** and store references to any constants and variables from the context in which they are defined. 클로저는 정의 된 컨텍스트에서 모든 상수 및 변수에 대한 참조를 캡처하고 저장할 수 있습니다.

Global and nested functions, as introduced in Functions , are actually special cases of closures. 글로벌 및 중첩 함수는 함수라고 설명했었으나 , 실제로는 클로져의 특별한 종류이다.

<클로저의 유형>

- Global function는 이름이 있고 값을 캡처하지 않는 클로저입니다.
- nested function는 이름을 가진 클로저로 내부 함수에서 값을 캡처 할 수 있습니다.
- Closure expressions은 주변 컨텍스트에서 값을 캡처 할 수있는 간단한 구문으로 작성된 이름없는 클로저입니다.

Closure Expressions

```
{( parameters ) -> return type in
    statements
}
```

Sorted(by:) 메소드 사용

```
func sorted(by areInIncreasingOrder: (Element, Element) -> Bool) -> [Element]
```

```
override func viewDidLoad() {
  super.viewDidLoad()

  let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]

  func backward(_ s1: String, _ s2: String) -> Bool {
       return s1 > s2
  }

  let sorted = names.sorted(by: backward)

  print(sorted) //[“Ewa”, "Daniella", "Chris", "Barry", "Alex"]
}
```

Inline Closure

```
override func viewDidLoad() {
  super.viewDidLoad()

  let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]

  //아래 예제 backward(_:_:)는 이전 버전의 함수를 클로저식으로 나타낸 것입니다.
  let sorted = names.sorted(by:
     {( s1: String, s2: String) -> Bool in
        return s1 > s2
     }
  )

  print(sorted) //[“Ewa”, "Daniella", "Chris", "Barry", "Alex"]
}
```

클로저 간소화

- Inferring parameter and return value types from context
- Implicit returns from single-expression closures
- Shorthand argument names

Inferring Type From Context

```
override func viewDidLoad() {
  super.viewDidLoad()

  let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]

  //아래 예제 backward(_:_:)는 이전 버전의 함수를 클로저식으로 나타낸 것입니다.
  let sorted = names.sorted(by:{ s1, s2 in return s1 > s2 })

  print(sorted) //[“Ewa”, "Daniella", "Chris", "Barry", "Alex"]
}
```

* s1과 s2는 배열(names)에서 가져온 인자인기 때문에 타입을 유추할수 있다. 또 리턴 결과값으로 인해 리턴의 타입도 유추 가능하다. **즉 클로저를 함수 또는 메소드에 인라인 클로저 표현식으로 전달할 때 항상 매개 변수 유형 및 리턴 유형을 유추 할 수 있습니다.**

Implicit Returns from Single-Expression Closures

```
override func viewDidLoad() {
  super.viewDidLoad()

  let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]

 //아래 예제 backward(_:_:)는 이전 버전의 함수를 클로저식으로 나타낸 것입니다.
  let sorted = names.sorted(by:{ s1, s2 in s1 > s2 })

  print(sorted) //[“Ewa”, "Daniella", "Chris", "Barry", "Alex"]
}
```

*** 클로져 내부 구현부에 리턴내용이 한줄로 단순하게 표시되어 있다면, return 키워드를 생략 가능하다.**

Fast campus

Shorthand Argument Names

Swift는 인라인 클로저에 자동으로 Shorthand Argument Names을 제공하며, 클로저 인수의 값을 이름 `$0`, `$1` 등 으로 참조하는 데 사용할 수 있습니다.

```
override func viewDidLoad() {
  super.viewDidLoad()

  let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]

 //아래 예제 backward(_:_:)는 이전 버전의 함수를 클로저식으로 나타낸 것입니다.
  let sorted = names.sorted(by:{ $0 > $1 })

  print(sorted) //[“Ewa”, "Daniella", "Chris", "Barry", "Alex"]
}
```

Operator Methods

Swift의 greater-than 메소드는 (>) 의 정의를 보면 `String` type의 두 매개 변수가 있고, Bool type값을 반환합니다. 이것은 `sorted(by:)`메소드에 필요한 메소드 유형과 정확히 일치합니다. 따라서 greater-than 연산자를 전달하기 만하면 Swift는 문자열 관련 구현을 사용하려고한다고 추론합니다.

```
override func viewDidLoad() {
  super.viewDidLoad()

  let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]


  let sorted = names.sorted(by: > )
  print(sorted) //[“Ewa”, "Daniella", "Chris", "Barry", "Alex"]
}
```

Trailing Closures

함수에 클로저 표현식을 함수의 최종 인수로 전달해야하고 클로저 표현식이 길면 후행 클로저 로 작성하는 것이 유용 할 수 있습니다 . Trailing Closures을 사용하면 함수 호출의 일부로 클로저의 인수 레이블을 쓰지 않습니다.

```
func someFunctionThatTakesAClosure(closure: () -> Void) {
    // function body goes here
}

// Here's how you call this function without using a trailing
closure:

func someFunctionThatTakesAClosure(closure: {
    // closure's body goes here
})

// Here's how you call this function with a trailing closure
instead:

func someFunctionThatTakesAClosure() {
    // trailing closure's body goes here
}
```

Trailing Closures

클로저 표현식이 함수 또는 메소드의 유일한 인수로 제공되고 그 표현식을 후행 클로저로 제공하면 함수를 호출 할 때 함수 또는 메소드의 이름 뒤에 한 쌍의 괄호를 쓸 필요가 없습니다 .

```
override func viewDidLoad() {
  super.viewDidLoad()

  let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]


  let sorted = names.sorted{ $0 > $1 }

  print(sorted) //[“Ewa”, "Daniella", "Chris", "Barry", "Alex"]
}
```

변수 할당

```
let closureValue = {(name:String) in print(name) }
```

```
closureValue("joo")
```

캡쳐

- 클로져 안의 모든 상수와 변수에 대한 참조를 캡쳐 해서 관리 한다.
- Swift는 캡쳐를 위한 모든 메모리를 관리한다.

캡쳐

```
func makeIncrementer(forIncrement amount: Int) -> () -> Int {
    var runningTotal = 0
    func incrementer() -> Int {
        runningTotal += amount
        return runningTotal
    }
    return incrementer
}
```

캡쳐

```swift
func makeIncrementer(forIncrement amount: Int) -> () -> Int {
    var runningTotal = 0
    func incrementer() -> Int {
        runningTotal += amount
        return runningTotal
    }
    return incrementer
}
```

```swift
let incrementByTen = makeIncrementer(forIncrement: 10)

incrementByTen()
// returns a value of 10
incrementByTen()
// returns a value of 20
incrementByTen()
// returns a value of 30
```

Closure 예

```
self.present(nextVC, animated:true, completion:( () -> void )? )
```

```
let alert = UIAlertAction(title: "알럿", style: .default,
                          handler: ((UIAlertAction) -> Void)? )
```

```
UIView.animate(withDuration: 0.3,
               animations: () -> Void,
               completion: ((Bool) -> Void)?)
```