Swift

Swift 5 및 iOS 개발 방식에 대한 이해

교재 제작 v3.1

이재호 2019-06-25

목차

1 Swift 소개 및 기본 요소 2
1.1 강의 계획 2
1.2 Swift의 기본 요소 2

1 Swift 소개 및 기본 요소

1.1 강의 계획

본 교재에는 Swift 5의 기본적인 문법을 빠르게 익힌 후, 객체 지향 프로그래밍의 개념에 익숙해지도록 도와줍니다. 나아가 본 수업을 통해 Swift와 Cocoa Touch 프레임워크를 사용해 iOS 모바일 어플리케이션을 개발하는 방법에 대해 배웁니다. 기존에 Python이나 C 언어 등 다른 언어로 어느 정도는 프로그래밍을 해보았다고 가정합니다. 다음은 Swift로 `Hello, World!`를 출력하는 코드입니다:

```
print("Hello, World!")
```

이는 Python 3와 작성한 Hello World 프로그램과 정확히 일치합니다. 이처럼 Swift는 모던 언어이기 때문에 문법적으로 간결하고, 이미 프로그래밍을 해보았다면 하루 정도면 어느 정도 원하는 작업을 수행하는 코드를 작성할 수 있을 것입니다. 그렇지만 본인이 원래 알던 언어의 억양을 강하게 가진 상태로 Swift 코드를 짜는 것은 그리 바람직하지 못한 일이므로, Swift의 기본적인 문법을 알아봅시다.

1.2 Swift의 기본 요소

상수Constant와 변수Variable

Swift에서는 상수와 변수의 구분을 중요하게 여깁니다.

```
var myVariable = 10
myVariable = 20
let myConstant = 7
myConstant = 8  // Cannot assign to value: 'myConstant' is a 'let' constant
```

Swift에서도 Python과 마찬가지로 손쉽게 변수 swapping을 할 수 있습니다. 이 역시 multiple assignment를 사용한 것으로, 아래와 같이 사용할 수 있습니다.

```
var (a, b, c) = (2, 7, 12)
(a, b, c) = (b, c, a)
print(a, b, c)  // 7 12 2
(a, b) = (b % a, a)
print(a, b)  // 5 7
```

Swift에서도 Python과 같이 모든 사칙 연산이 가능하지만, 지수 연산자 **에 해당하는 연산자는 존재하지 않습니다. 대신 `Foundation` 프레임워크에 내장되어 있는 `pow(_:_:)`를 사용할 수 있습니다.

Foundation 프레임워크는 Swift의 표준 라이브러리에 추가적으로 사용할 수 있는 프레임워크입니다. 앞으로 iOS 앱 개발에서 GUI를 다룰 때 항상 코드 맨 앞에 UIKit을 불러올 것인데, 이는 Foundation을 이미 포함하는 것입니다. 따라서 UI 와 무관한 기본적인 기능을 구현할 때에는 보통 Foundation을 불러와 구현합니다. 예를 들어, Swift는 언어 자체에는 URL이라는 자료형이 없지만, Foundation을 사용하면 이를 쉽게 다룰 수 있습니다.

Swift, C, Java 등은 정적 타입 언어이고, Python, JavaScript, Ruby 등은 동적 타입 언어입니다. 무슨 말이냐 하면, Swift와 같은 정적 타입 언어는 컴파일 시 어떤 값의 자료형이 정해지고, Python과 같은 동적 타입 언어는 런타임 시 값의 자료형이 바뀔 수 있습니다. 즉, 아래와 같은 코드는 Swift에서 허용되지 않습니다.

```
var myInt = 4
myInt = 1.2  // Cannot assign value of type 'Double' to type 'Int'
```

실제로 위에서 첫번째 줄은 var myInt: Int = 4를 줄여 쓴 것입니다. C에서처럼 int myInt = 4으로 자료형을 항상 명시할 필요는 없지만, 강조하거나 컴파일러가 자료형을 추측할 수 없을 경우 표시해주어야 합니다. 또한 특정적인 점은 자료형을 변수 뒤에 써준 다는 것입니다. 이는 새로운 것은 아니며 Pascal의 방식을 따른 것입니다.

Swift에서는 문자를 다루는 것이 Python만큼 간결하지는 않지만, 간단한 부분에 있어서는 간편하게 쓸 수 있습니다. Swift에서는 문자열을 큰따옴포(")로만 감쌀 수 있습니다. 작은 따옴표는 허용되지 않습니다. 또한 문자열에서 다른 변수의 값을 넣는 것이 매우 간편합니다:

```
let value = 4
print("value is \(value).")
```

와 같이 \(·)안에 변수를 감싸면 됩니다.

배열을 만드는 것도 Python만큼이나 Swift에서도 매우 간단합니다. 똑같이 [·]으로 감싸고, 인덱스로 접근하는 방식도 같습니다:

```
var myList = ["Swift", "Python", "C", "Objective-C", "Scala", "Haskell"]
print(myList[0])
```

단, Swift가 정적 타입 언어라는 것을 유념해야 합니다. Python처럼 아무 값이나 전부 배열로 만들 수 없으며, 반드시 한 종류의 자료형만을 담고 있어야 합니다. 즉, 위에서 myList는 [String]의 자료형을 가지고 있습니다.

이는 딕셔너리도 마찬가지입니다:

```
var myDictionary = ["Swift": 2014,
                    "Python": 1991,
                    "C": 1972,
```

```
4                     "Objective-C": 1984,
5                     "Scala": 2004,
6                     "Haskell": 1990]
7 myDictionary[1996] = "OCaml"
8 // Cannot subscript a value of type '[String : Int]' with an index of type 'Int'
9 myDictionary["Kotlin"] = "2011"
10 // Cannot assign value of type 'String' to type 'Int?'
```

빈 배열이나 딕셔너리의 경우, Swift가 자료형을 추측할 수 있도록 다음과 같이 정의할 수 있습니다:

```
1 var myList1: [Int] = []
2 var myList2 = [Float]()
3 var myList3 = []  // Empty collection literal requires an explicit type
4
5 var myDictionary1: [String: Bool] = [:]
6 var myDictionary2 = [Double: UInt]()
7 var myDictionary3 = [:]  // Empty collection literal requires an explicit type
```

조건문 if입니다:

```
1 let grades = ["A+", "A0", "B+", "A+", "A-", "F"]
2 if "A+" in grades {
3     print("A+ is in grades!")
4 } else {
5     print("A+ is not in grades!")
6 }
```

Python과 다른 점은 콜론(:) 대신 중괄호({·})를 사용하여 구문을 구분한다는 점입니다.

반복문 for입니다:

```
1 let grades = ["A+", "A0", "B+", "A+", "A-", "F"]
2 for grade in grades {
3     print(grade)
4 }
```

혹은, 다음과 같이 클로져와 함께 forEach(_:) 메소드를 사용할 수 있습니다:

```swift
let grades = ["A+", "A0", "B+", "A+", "A-", "F"]
grades.forEach { grade in
    print(grade)
}
```

혹은 더 짧게:

```swift
let grades = ["A+", "A0", "B+", "A+", "A-", "F"]
grades.forEach { print($0) }
```

클로져에 대해서는 뒤에 함수 부분에서 다시 소개하도록 하겠습니다. 클로져는 매우 중요한 내용입니다.

Python에 없었던 내용 중에는 switch 문이 있습니다. 앱에서 다양한 경우를 고려해야 할 경우, switch를 사용하면 편리합니다.

```swift
let test = "sample"
switch test {
case "wow":
    print("wow case")
case "foo", "bar":
    print("foo-bar case")
case let x where x.hasPrefix("sample"):
    print("sample")
default:
    print("default")
}
// sample
```

C와의 비교했을 때 눈에 띄는 차이점은 break가 없다는 것으로, Swift의 switch 구문은 case에서 다음으로 fall through하지 않습니다. 만약 C처럼 fall through가 필요하다면 fallthrough 키워드를 사용하면 됩니다. 또한 let을 통한 조건 확인을 통해서 매우 강력한 도구가 될 수 있습니다. 나아가 enum의 associated value를 활용한다면 더욱 빛을 발합니다.

while과 repeat-while 구문 설명하기

함수와 클로져 설명하기 (map, Python 람다...)