02. Swift 2.0으로 iOS 코딩하기 - Swift Tour (2)

두번째 시간입니다. Swift 에서 기본 제공하는 Collection type 들에 대해 알아보면서 시작하도록 하겠습니다.

Array

어느 언어나 Array 를 지원합니다. C 에서는 연속된 메모리로 지원하는데 이건 (편리하지만) 꽤 위험한 접근이죠. 다른 고급 언어들은 이러한 방법을 지양하고 있습니다. 데이터구조론적 측면에서 봤을때는 array 란 index 와 value 의 짝일 뿐이니까요. 많은 언어들에서 array 를 이용하기 위해 square bracket 을 subscript 와 함께 사용하는 문법을 제공하는데, Swift 에서도 마찬가지입니다. 다음과 같이 사용합니다.

<a>

```
5  let names = [ "Kim", "Lee", "Park" ]
6  let kim = names[0]
7  let count = names.count
8
```

```
["Kim", "Lee", "Park"]
"Kim"
3
```

위 코드<a> 에서 names 의 type 은 [String] 입니다. 즉 `let names:[String] = [ ... ]` 이렇게 쓸 수 있겠죠. 하지만 지난 시간에 말씀드린 것과 같이 Swift 는 type-inference 라는 기능을 이용해 컴파일러가 무슨 타입인지 알아낼 수 있다면 타입의 지정을 생략해도 됩니다.

일반적으로 여러 언어들에서 array 에 제공해 주는 operation 들은 어떤 것이 있을까요? 갯수를 알아 오는 것 (count) 과 특정 index 의 value 를 알아 오는 것이 가장 기본이 되는 operation 입니다. 후자는 위에 이미 배운대로 sqare bracket ([]) 를 사용하여 숫자를 지정하는 방식을 사용합니다. 갯수는 array 의 property 인 count 를 사용합니다. 이 외에도 Swift 의 Array 는 isEmpty, first, last 등의 property 를 제공하기도 하고 sort() 와 같은 함수를 통해 정렬된 새로운 Array 를 리턴하는 함수를 제공하기도 합니다.

다른 언어의 array 가 또 어떤 함수들을 제공하나요? 그것은 바로 element 추가/삭제하기, 내용 변경하기, 정렬(sortInPlace)하기 등입니다. 하지만 위 코드에서 names 에는 이런 함수들이 전혀 사용할 수 없습니다. 왜 array 가 그런 기본적인 함수조차 제공하지 않는 것일까요?

<b>

```
5  let names = [ "Kim", "Lee", "Park" ]
6  names.append("Choi")
   Cannot use mutating member on immutable value: 'names' is a 'let' constant
7
8
```

```
["Kim", "Lee", "Park"]
["Kim", "Lee", "Park", "Choi"]
```

그것은 바로 append() 와 같은 함수들이 mutating 으로 선언되어 있고, 이렇게 mutating 으로 선언된 함수들은 let 으로 선언된 값 (constant) 에게는 사용할 수 없기 때문입니다. 값을 변경하기 위해서는 var 로 선언해야만 합니다. Objective-C 를 이용할 때도 NSArray 와 NSMutableArray 라는 두 개의 클래스가 각각 immutable 객체와 mutable 객체를 구분지었는데, Swift 는 아예 이런 구분을 언어 차원에서 할 수 있도록 하고 있습니다. 위에서 언급한 sort() 가 정렬된 새로운 배열을 리턴한다면 배열 내부를 새롭게 정리하는 함수인 sortInPlace() 는 mutating 함수입니다.

<c>

```
4  var nums = [ 1,2,3,4,5 ]          [1, 2, 3, 4, 5]
5  nums.append(67)                   [1, 2, 3, 4, 5, 67]
6  nums[3] = 100                     100
7
8  nums                              [1, 2, 3, 100, 5, 67]
9  nums.sortInPlace()                [1, 2, 3, 5, 67, 100]
10 nums                              [1, 2, 3, 5, 67, 100]
11
```

값을 변경하는 것은 위 <c> 코드의 6 라인처럼 합니다. 매우 직관적이죠. 배열의 값을 변경시키는 함수 및 방법들은 꽤나 다양하지만 차차 알아나기로 합니다. 이번엔 다음 코드 <d> 를 보죠.

<d>

```
12 let a = [ 1, 2, 3 ]          [1, 2, 3]
13 var b = a                    [1, 2, 3]
14 var c = b                    [1, 2, 3]
15 b[1] = 100                   100
16
17 a                            [1, 2, 3]
18 b                            [1, 100, 3]
19 c                            [1, 2, 3]
```

배열을 선언한 뒤 b 와 c 에 옮겨 담았습니다. 그런데, Swift 가 배열을 다룰 때 특이한 점은 Copy-on-write 라는 기법을 사용한다는 점입니다. 13 라인과 14 라인을 통해 이루어지는 것은 무엇일까요? 하나의 배열 객체를 세 개의 변수 및 상수가 공유하는 것 처럼 보입니다만, 사실은 그렇기도 하고 아니기도 합니다. 14라인까지 실행되었을 때 원소 세 개를 가지는 배열을 가리키는 변수/상수 는 모두 a, b, c 세 개이지만, 이 배열을 유지하기 위해 사용되는 메모리는 하나뿐입니다. 15 라인에 이르러서야, b 의 내용을 변경하는 순간 실제 copy 가 일어납니다. 그래서 a 와 c 는 동일 메모리에 유지되고, b 만 다른 메모리로 복사되어 index 1 에 100 의 값을 써넣게 됩니다. c 는 b 를 대입받았지만, 복사의 개념이므로 (실제 복사는 나중에 일어나지만) b 의 원소를 변경하더라도 c 에는 영향을 미치지 않게 되는 것입니다.

Swift 의 type inference 를 이용하면 각 변수/상수 의 type 을 적어 주지 않아도 된다고 했지만, 가끔 명시적으로 적어 주어야 하는 경우도 있습니다.

<e>

```
5  let nums = [ 1, 2 ]                          [1, 2]
6  nums.dynamicType                             Array<Int>.Type
7
8  let doubles = [ 1.0, 2 ]                     [1, 2]
9  doubles.dynamicType                          Array<Double>.Type
10
11 let x = 1, y = 2
12 let xy = [ x, y]                             [1, 2]
13 xy.dynamicType                               Array<Int>.Type
14
15 let xy2 : [Double] = [ x, y ]
16    Cannot convert value of type 'Int' to expected element type 'Double'
17 xy2.dynamicType
```

위 <e> 코드에서 nums 의 경우 타입은 [Int] 입니다. doubles 의 경우 Double literal 인 1.0 과 Int literal 인 2 가 사용되었지만 둘 다 Double 로 취급될 수 있으므로 [Double] 이 됩니다. 하지만 타입이 이미 Int 로 지정되어 있는 x 와 y 를 원소로 하는 xy 의 경우 [Int] 로만 선언이 가능합니다. [Double] 로 선언하기 위해 15라인과 같이 쓰면 에러가 되며 이때는 Double(x) 와 같은 형태로 형변환을 해 주어야 합니다. dynamicType 은 예제에서 타입이 무엇인지 보여드리기 위해 임시로 사용한 코드이니, 실제 코드에서는 지양하는 것이 좋겠습니다. type 에 따라 다른 동작을 하기를 원한다면 is 라는 operator 를 쓰거나 함수를 override 하는 등의 방법을 쓰는 것이 바람직합니다.

<f>

```
5  let what = [ "Swift", 2.0 ]                  ["Swift", 2]
6  what                                         ["Swift", 2]
   V [NSObject] what
```

자동변환 (conversion) 이 가능하지 않은 type 들을 섞어 쓰는 Array 도 가능합니다. 위 <f> 코드의 경우 what 의 type 은 [NSObject] 가 됩니다. 이런 Array 를 사용할 때는 각 element 의 타입이 NSObject 이므로 실제 객체의 class 로 형변환을 해야 해당 객체로서 동작할 수 있게 됩니다.

빈 array 는 [] 로 표현합니다.

<g>

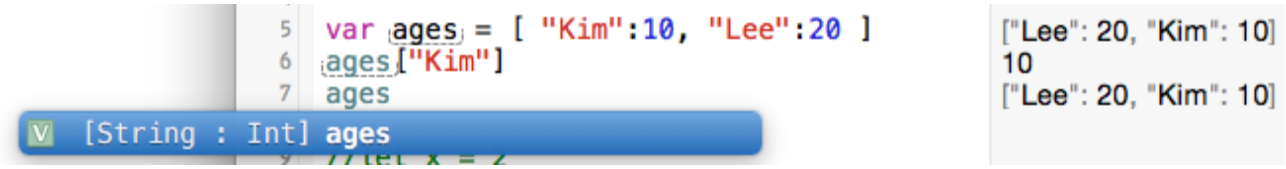
```
5  var names : [String] = []                    []
6  names.append("Kim")                          ["Kim"]
7  names                                        ["Kim"]
8
9  var what = []                                (
10 what.append("Hello")                         (
11 what
      Value of type 'NSArray' has no member 'append'
   V NSArray what
```

단 그냥 [] 로 쓰면 컴파일러 입장에서는 type inference 를 적용할 방법이 없기 때문에 그냥 Objective-C NSArray 타입인 것으로 인지합니다. 따라서 비어 있는 것을 초기값으로 하는 array 의 type 은 명시해 주는 것이 대개의 목적에 맞습니다. 8 라인처럼 type 을 명시하지 않은 빈 array 에 append() 함수를 호출하려고 하면 위와 같은 에러가 발생합니다. 위 <b> 코드의 에러와 조금 다르죠. element 를 추가하는 함수는 NSArray 에는 없고 NSMutableArray 에만 있는 함수이기 때문인데, (게다가 함수 이름도 append() 가 아니고 addObject()) 지금 수준에서는 일단 에러다 라고만 생각하기로 하죠.

Dictionary

Array 와 함께 매우 많이 사용되는 Collection type 이 바로 Dictionary 입니다. Associative Array 라는 표현을 쓰기도 하고 Perl 이나 Javascript 등의 언어에서 많이 지원하는 개념인데, 사전적인 정의는 Key 와 Value 의 짝을 모아 둔 것이죠. (Java 에서는 Hashtable 혹은 HashMap) Type 을 적을 때는 Key 의 type 과 Value 의 type 을 : 으로 구분하여 함께 적습니다. 예를 들어 String 이 Key 이고 Int 가 Value 인 Dictionary 는 [String:Int] 로 표현합니다. 아래 코드의 ages 가 이 형태의 Dictionary 입니다.

<h>



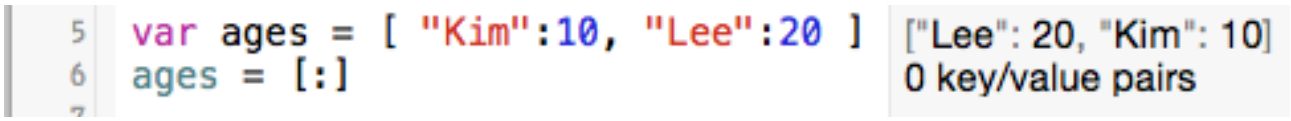
Array 와 마찬가지로 Dictionary 도 let 으로 선언한 것은 값을 추가/변경/삭제 할 수 없는 immutable dictionary 이고 var 로 선언하면 변경 가능한 mutable dictionary 가 됩니다.

Key 로 사용할 수 있는 type 은 무엇일까요? (아직은) 어려운 말로 하면 `Hashable` protocol 을 conform 하는 type 이라고 할 수 있지만, 여러분들이 알아들을 수 있는 말은 아닌듯 합니다. 일단은 String 을 사용할 수 있고, Int, Double 등의 Number 타입들 을 Key 로 사용할 수 있다 정도로 생각하시면 당분간은 적당할 듯합니다.

위 <h> 의 오른쪽에 출력된 것을 보면 "Kim" 과 "Lee" 의 순서가 반대로 되어 있는 것을 볼 수 있습니다. 왜 그럴까요? 일반적으로 dictionary 자료 구조는 주어진 key 로 빠르게 (hash 값을 사용하여) value 를 찾는 데에 촛점이 맞추어져 있기 때문에, 각 element 의 저장 순서는 보장해 주지 않습니다. 앞의 말을 다시 정확히 표현하면 순서가 반대로 되어 있는 것이 아니라 순서는 그때그때 다를 수 있다 라는 것입니다.

Array 와 마찬가지로, Dictionary 도 빈 객체를 의미하는 표현법이 있는데, 바로 [:] 입니다.

<i>



Array 와 Dictionary 는 존재하지 않는 입력값에 대응하는 방법이 다릅니다. Array 는 존재하지 않는 index 를 주면 Runtime Error 를 발생시킵니다. 따라서 반드시 접근 전에 count 를 보고 존재하는 값인지 확인해야 합니다. 다른 방법으로는 멤버 함수인 contains() 를 사용하여 해당 인덱스가 존재하는지 확인해 주어야 합니다.

<j>

```
5  var x = [ 1, 2, 3 ]
6  x.contains(2)
7  x.contains(10)
8  var count = x.count
9
```

```
[1, 2, 3]
true
false
3
```

반면, Dictionary 는 [] 를 거치고 난 값이 Value 타입이 아니고 Value? 타입 (물음표 맞습니다. Optional Value 타입) 입니다. 문제는 아직 Optional 이 뭔지 안 배웠다는 것이죠. Optional 은 Swift 의 아주 강력한 기능이자 다른 언어에서 경험해보지 못한 개념일 수 있습니다. 반드시 이해야야 하는 개념이지만, 일단 Optional 을 배우기 전까지는 아래처럼 ! 를 붙여서 쓴다 라고만 생각해 두셔도 좋겠습니다.

<k>

```
5  var ages = ["Kim":10, "Lee":20]
6  let kim = ages["Kim"]
7  print(kim)
8
9  let kim2 = ages["Kim"]!
10 print(kim2)
11
```

```
["Lee": 20, "Kim": 10]
10
"Optional(10)\n"

10
"10\n"
```

Set

Array, Dictionary 말고도 Set 라는 Collection type 도 존재하는데, 우리말로 번역하면 집합 입니다. Dictionary 에서 Value 가 빠진 것이라 생각할 수도 있고, Array 에서 중복 및 순서가 없는 것이다 라고 생각할 수도 있습니다. 존재여부, 합집합, 교집합 등의 operation 을 제공하는데, 많이 사용되지는 않습니다. Literal 을 따로 제공하지는 않기 때문에 Set<Int> 나 Set<String> 과 같은 형태로 type 을 지정하여 사용합니다.

마침

두번째 시간을 이것으로 마칩니다. 다음 시간에 만나요.