01. Swift 로 iOS 코딩하기 - Swift Tour

Swift 로 iOS 코딩을 하려면 우선 XCode 를 설치해야 합니다.

https://itunes.apple.com/us/app/xcode/id497799835?ls=1&mt=12

링크를 열면 AppStore 가 열리고 XCode 를 다운로드 받을 수 있습니다. Mac AppStore 에서 XCode 를 검색해서 찾아도 되겠네요.

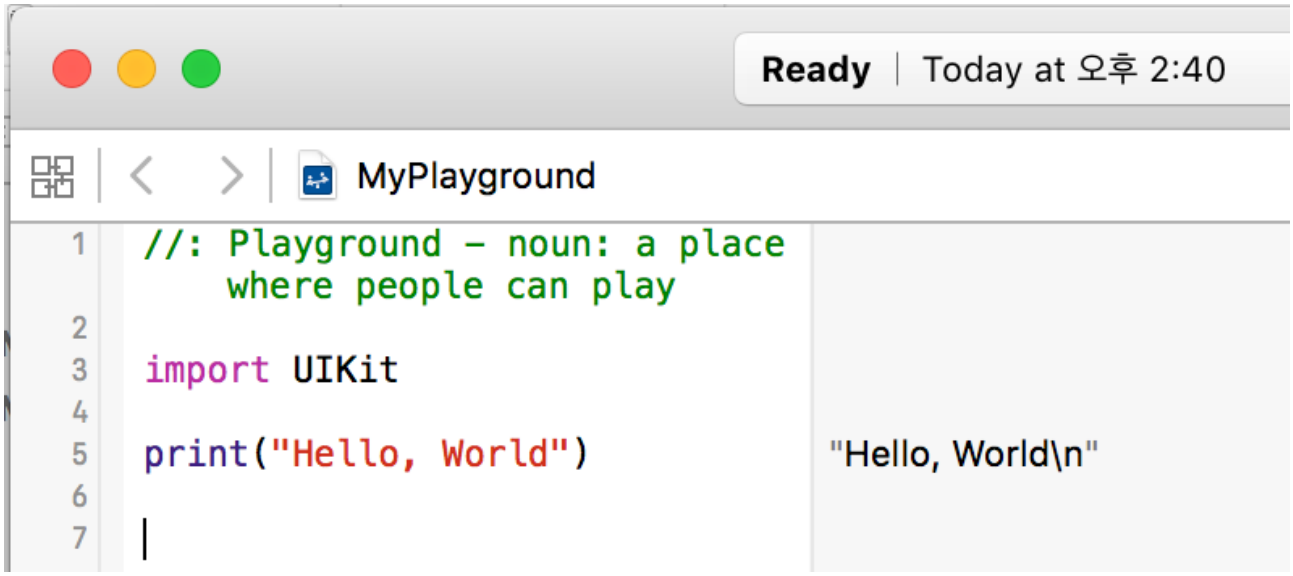
다운로드를 마치고 처음 실행하면 해 볼 수 있는 것중 하나가 Playground 입니다. XCode 는 프로젝트를 열지 않고도 Playground 를 통해 코드를 실행하는 것이 가능하기 때문에, New Playground 를 선택하여 여러 코드들을 입력해 볼 수 있습니다. Playground 별로 저장하고 불러오는 것도 가능하니 강좌에서는 초반 여러 코드를 Playground 를 이용하여 실행해 볼 것입니다.

먼저 XCode 의 메뉴 File->New->Playground 를 통해 Playground 를 엽니다. 이름 및 저장 위치를 지정해 주면 나중에 다시 불러올 수도 있습니다. 여기에 Hello World 부터 해보죠

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

라고 입력합니다. 우측 결과창에 Hello, World 라는 문자열이 출력됩니다.

<a>



Playground 는 이와 같이 실행 결과를 바로 바로 볼 수 있는 장점이 있어서 간단한 코드를 테스트해 볼 때 유용하게 사용할 수 있습니다.

다음은 변수에 대해 알아보겠습니다.

Swift 에서는 변수 라는 말이 조금 모호하긴 합니다. 다른 언어에서라면 값을 변경할 수 있으면 변수, 변경할 수 없으면 상수 라는 표현을 쓰는데, Swift 에서는 다른 언어라면 변수로 선언해서 쓸 만한 값들도 상수로 선언해서 쓰기 때문입니다. 저는 이 구분을 Mutable 변수와 Immutable 변수 로 구분하고 싶기도 한데요, 어쨌든 Swift book 에서는 상수(constant) 라는 표현을 사용하고 있으니 이를 따르기로 합니다.

변수 및 상수 선언을 위해서는 `var` 와 `let` 이라는 키워드를 사용합니다. 다음과 같이 합니다.

<b>

```
7   var name = "GyonG"                          "GyonG"
8   var age = 20                                20
9
10  let bornIn = 1990                           1990
11
12  age++
13  bornIn++
14    Cannot pass immutable value to mutating operator: 'bornIn' is a 'let' constant
```

위의 값들 중 name 과 age 는 변수 (variable) 이고 bornIn 은 상수 (constant) 입니다. 사용하는 방법은 거의 같은데, 변수는 그 값이 변화될 수 있고 (mutable) 상수는 그 값이 변하지 않는 (immutable) 경우 사용합니다. 보통 다른 언어에서의 상수는 컴파일 시점에 값이 결정되는 것을 상수라고 하는 반면 Swift 에서는 값 선언 시점에 결정되어 이후로 변하지 않는 것을 상수라고 하는 것이 다릅니다. 위 예제에서 name 이나 age 는 다른 값을 가리키게 될 수도 있습니다. age++ 에서와 같이 그 값을 1 증가시키는 코드가 실행될 수 있기도 합니다. 하지만 bornIn 은 `let` 으로 선언된 상수이므로 값을 변경시키는 것이 불가능합니다. 라인 넘버 왼쪽의 빨간 동그라미를 눌러 보면 XCode 가 가끔 자동고침 Fix-it 을 보여주기도 하는데, 아래 <b-1> 에서는 ++ 을 할거면 `let` 을 쓰지 말고 `var` 를 쓰는 것이 어뜨냐고 제안을 하는 군요. 라인 10 의 `let` 도 `var` 로 살짝 바뀌어 있습니다.

<b-1>

```
7   var name = "GyonG"                          "GyonG"
8   var age = 20                                20
9
10  var bornIn = 1990                           1990
      Cannot pass immutable value to mutating operator: 'bornIn' is a 'let' constant
      Fix-it  Change 'let' to 'var' to make it mutable
14  age++                                       20
15  bornIn++                                    1990
16    Cannot pass immutable value to mutating operator: 'bornIn' is a 'let' constant
```

Objective-C 로 iOS 코딩을 해 본 사람은 Foundation 에서 제공하는 클래스인 NSString 과 NSMutableString, NSArray 와 NSMutableArray, NSDictionary 와 NSMutableDictionary 등의 구분에 대해 익숙할 것입니다. iOS/OSX 에서는 이와 같은 구별을 오래 전부터 해 왔고, 이것이 프로그램의 안정성이나 Multi-thread (혹은 multi-core) 처리에 유리하다는 것을 경험해 왔습니다. Swift 가 제안되면서 이와 같은 구분이 모든 type 으로 확대되었다고 보면 될 것 같습니다.

C 나 java 같이 type strict 한 언어들은 변수/상수를 선언할 때 반드시 type 과 함께 적도록 합니다. Javascript 같이 그렇지 않은 언어들은 type 을 적지 않습니다. 그러면 Swift 는 type strict 하지 않은 언어일까요? 그렇지 않습니다. Swift 는 어떤 언어보다도 type strict 합니다. 다음 예제를 보죠

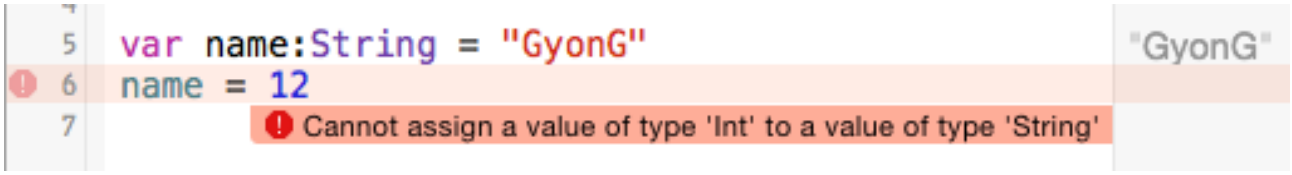
<c-1>

```
5   var name = "GyonG"                          "GyonG"
6   name = 12
7           Cannot assign a value of type 'Int' to a value of type 'String'
```

이게 Javascript 였다면 아무런 문제가 없을 코드입니다. 하지만 Swift 컴파일러는 에러를 냅니다. name 이라는 녀석은 string type 으로 선언된 변수이기 때문에 Int 값인 12 를 담을 수 없어서 에러가 나는 것입니다. 아주 "type-strict" 하다고 말할 수 있죠. 그런데 어찌 type 을 쓰지 않을까요?

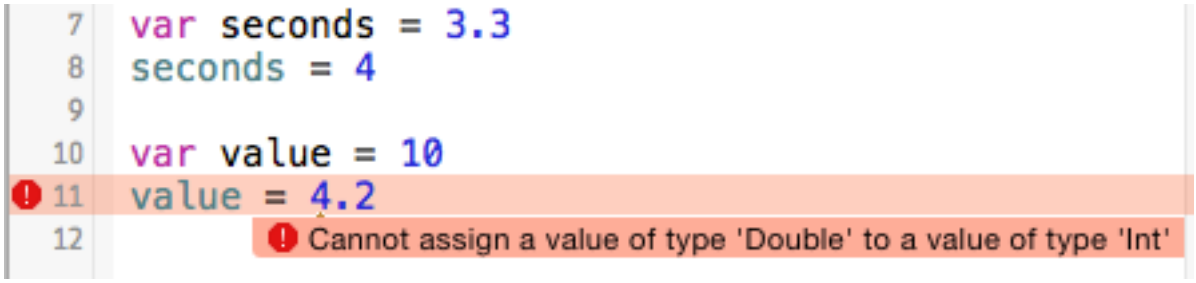
사실은 위의 코드는 아래처럼 쓰는 것과 완전히 동일한 코드를 생성합니다.

<c-2>



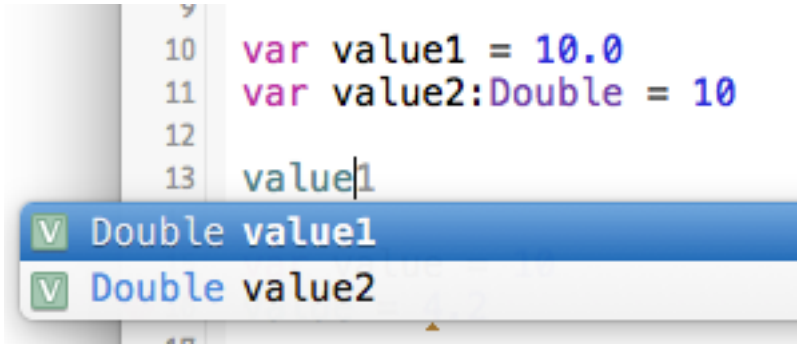
name 이라는 변수를 String 타입으로 선언하는 것으로, : 과 함께 type 을 적습니다. 이것은 변수 선언의 `var` 와 상수 선언의 `let` 이 동일합니다. 그런데 Swift 에는 Type-Inference 라는 것이 있어서, 컴파일러가 무슨 타입인지 알아낼 수 있으면 생략해도 되는 특성이 있습니다. 위 <c-1> 코드에서 name 에 "GyonG" 을 대입했기 때문에, 컴파일러는 이것이 문자열이라는 사실을 알아챌 수 있었습니다. 따라서 이것의 타입을 생략해 주어도 컴파일러가 판단하여 결정합니다. Type inference 를 이용하여 type 을 생략할 때는 프로그래머와 컴파일러가 모두 혼란을 겪지 않도록 명확히 해 주는 것이 중요합니다. 다음 코드를 보죠

<d>



seconds 는 3.3 을 대입한 것으로 보아 컴파일러는 이것이 Double 타입인 것으로 추측하여 선언합니다. 여기에 다시 4 라는 Int 값을 넣는 것은 변환이 가능하기 때문에 (conversion) 문제가 없습니다. 하지만 value 로 선언된 변수는 10 을 대입한 것으로 컴파일러는 이것이 Int 인 것으로 판단합니다. 따라서 다음 11 라인의 4.2 즉 Double 값을 대입하는 것은 에러로 처리합니다. 만일 value 의 초기값이 10 인 Double 타입이기를 원하면 10 라인의 코드를 `var value = 10.0` 이라고 하거나 `var value:Double = 10` 이라고 해 주어야 합니다. 그러면 다음의 <e> 코드처럼 value1 과 value2 가 모두 Double 로 선언된 것을 알 수 있습니다.

<e>



변수/상수 혹은 mutable/immutable 구분이 처음이신 분들은 언제 var 를 써야 하고 언제 `let` 을 써야 하는지가 헷갈릴 수가 있습니다. 만일 그렇다면, **모든 값의 선언을 let 으로 하시면 됩니다.** 그러다가 컴파일러가 다음과 같은 에러를 내면 그 선언부를 `let` 에서 `var` 로 변경해 주면 됩니다.

<e-1>

```
14      override func viewDidLoad() {
15          super.viewDidLoad()
16          // Do any additional setup after loading the view, typically from a
                nib.
17          var value = "Hello";
18          label.text = value;   Variable 'value' was never mutated; consider changing to 'let' constant
19
20      }
```

<e-2>

```
14      override func viewDidLoad() {
15          super.viewDidLoad()
16          // Do any additional setup after loading the view,
17          let value = "Hello";
18          label.text = value;
```

Variable 'value' was never mutated; consider changing to 'let' constant

Fix-it Replace “var” with “let”

<e-1> 에서는 원래 `var` 로 선언했지만 <e-2> 의 Fix-it 이 `let` 을 추천하고 있습니다.

Swift 2.0 부터는 `var` 로 선언해 놓고 값을 변경하지 않으면 위와 같이 warning 으로 처리하고 있으니, 확실히 값을 변경하는 것이 아니면 `let` 으로 시작해야 합니다.

<f>

```
5   let pi = 3.14                                                    3.14
6   pi = 3.14159                  Cannot assign to 'let' value 'pi'
7
```

Swift 에서는 값을 출력하기 위한 함수인 print 계열 함수들을 제공합니다. 보통은 new-line 을 포함하는 print() 함수 형태로 사용합니다. Console 에 출력을 하는 것이므로 iOS App 을 만들 때는 log 를 확인하는 정도로 사용하겠지만, 다른 플랫폼에서라면 standard-output 으로 출력하는 것으로 볼 수 있습니다. print 계열 함수들은 그 argument 가 다양한 type 이 될 수 있지만, 보통은 문자열 형태로 바꾸어서 출력하는 것이 보기에 편리할 때가 있습니다.

<g>

```
5   let pi = 3.14                                    3.14
6   let msg = "value of π is \(pi)"                  "value of π is 3.14"
7   print(msg)                                       "value of π is 3.14\n"
8
```

위와 같이 문자열 중간에 escape 를 해서 변수 값을 넣을 수 있는 기법을 Interpolation 이라 칭하기도 하는데, Perl 같은 언어에서 이런 interpolation 을 지원하곤 하죠. Swift 에서도 \(와) 를 문자열 중간에

사용하는 방식으로 지원합니다. Playground 에서는 6라인과 7라인의 우측에 모두 그 문자열이 출력되었는데, 이는 Playground 는 print 계열 함수가 아니더라도 값이 대입되기만 해도 그 값을 출력해주는 기능 때문입니다. 이 interpolation 이 사용될 때 \(와) 사이에는 아무 expression 이나 올 수 있기 때문에 다음과 같이 계산식이나 함수 호출 등이 사용되어도 무방합니다.

<h>

```
1  let apples = 3
2  let oranges = 5
3  let appleSummary = "I have \(apples) apples."
4  let fruitSummary = "I have \(apples + oranges) pieces of
           fruit."
```

첫시간은 이정도로 마쳐 보기로 할까요? 각자 위에 보여드린 예제 외에 여러 가지들을 연습해 보시면 좋을 것 같습니다.