# swift 문법 1

# 과제

- 필수
  - 실행 가능한 모든 소스를 변형해서 실행해보고 소스만 제출하세요.
  - 주석은 많을수록 좋아요!
- 선택
  - 실행 결과를 주석 처리하세요.

# 절대 사소한 실수가 아니다

- Ios IOS ios object-c xcode cocoapod swift Swift UI

iOS


CocoaPods

COCOAPODS

Xcode

SwiftUI

Objective-C

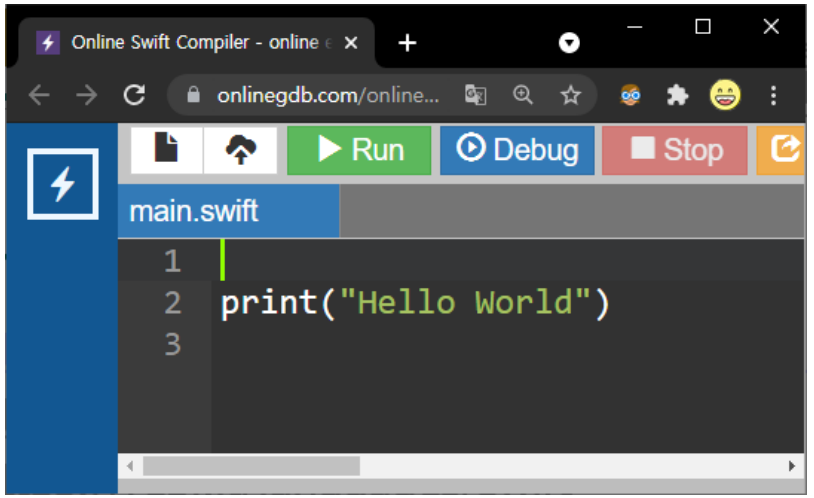
Swift

# Swift 추천 사이트

- https://docs.swift.org/swift-book/index.html
- https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html
- Swift 변경 사항 정리
  - https://docs.swift.org/swift-book/RevisionHistory/RevisionHistory.html
  - https://github.com/apple/swift-evolution
- raywenderlich.com의 Swift 스타일가이드
  - https://github.com/swift-kr/swift-style-guide-raywenderlich/blob/master/ko_style_guide.md

# Swift 문법 실습

- online Swift compiler
- Swift 5.x
  - https://www.onlinegdb.com/online_swift_compiler
  - http://online.swiftplayground.run/
  - https://repl.it/languages/swift
  - https://www.jdoodle.com/execute-swift-online/




- Swift 4.0(비추)
  - https://www.tutorialspoint.com/compile_swift_online.php

# Swift

| | |
|---|---|
| Paradigm | Multi-paradigm: protocol-oriented, object-oriented, functional, imperative, block structured, declarative, concurrent |
| Designed by | Chris Lattner, Doug Gregor, John McCall, Ted Kremenek, Joe Groff, and Apple Inc.[1] |
| Developer | Apple Inc. and open-source contributors |
| First appeared | June 2, 2014; 8 years ago[2] |
| | |
| Stable release | 5.6.2[3] / 15 June 2022; 2 months ago |
| Preview release | 5.7 branch (5.8 and 6 coming next) |
| Typing discipline | Static, strong, inferred |
| OS | Apple's operating systems (Darwin, iOS, iPadOS, macOS, tvOS, watchOS), Linux, Windows 10, Android |
| License | Apache License 2.0 (Swift 2.2 and later)<br>Proprietary (up to Swift 2.2)[4][5] |
| Filename extensions | .swift, .SWIFT |
| Website | •www.swift.org<br>•developer.apple.com/swift/ |
| Influenced by | |
| Objective-C,[6] Rust, Haslkell, Ruby, Python, C#, CLU,[7] D[8] | |
| Influenced | |
| Rust[9] | |

https://en.wikipedia.org/wiki/Swift_(programming_language)

# 데이터 타입(자료형)
# 상수(let), 변수(var)

# 데이터 타입(자료형, data type)

- 정수형 숫자를 저장하는 변수
  - var myNumber = 10
  - myNumber라는 이름의 변수를 생성했으며, 숫자 10을 할당
  - var myNumber : Int = 10  // int x = 10;  // C/C++
  - 위와 같이 초깃값이 있을 경우에는 컴파일러가 타입 추론(type inference)을 하므로 데이터 타입을 명시할 필요 없음
- Bool, Character, Int, Float, Double, String, Void
- var x : Int
- x= 10
  - //주의 error '=' must have consistent whitespace on both sides
  - '=' 양쪽에 일관된 공백이 있어야 함
- print(x) //실행결과 주석처리 방법
  - https://www.onlinegdb.com/online_swift_compiler
  - https://developer.apple.com/documentation/swift/1541053-print

# 자료형의 종류와 크기가 궁금해요

```swift
var x = 10
print(type(of:x))  //Int
let s = MemoryLayout.size(ofValue: x)  //8
let t = MemoryLayout<Int>.size
print(s, t)
```

## print(_:separator:terminator:)

주어진 항목의 텍스트 표현을 표준 출력에 씁니다.

### 선언

```
func print(_ items: Any..., separator: String = " ", terminator: String = "\n")
```

### 매개변수

**items**
인쇄할 항목이 0개 이상입니다.

**separator**
각 항목 사이에 인쇄할 문자열입니다. 기본값은 단일 공백( " ")입니다.

**terminator**
모든 항목이 인쇄된 후 인쇄할 문자열입니다. 기본값은 개행( "\n")입니다.

# 일반적으로 초깃값을 주지 않을 경우에만 자료형을 씀

- https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html#//apple_ref/doc/uid/TP40014097-CH5-ID309

- `var welcomeMessage: String   //초깃값이 없으므로 자료형을 직접 표기함`
- It is rare that you need to write type annotations in practice.
- If you provide an initial value for a constant or variable at the point that it is defined, Swift can almost always infer the type to be used for that constant or variable.

- `let myAge = 25                    // myAge가 Int형으로 타입 추론됨`
- `let pi = 3.14159                  // pi는 Double형으로 추론됨`
- `let anotherPi = 3 + 0.14159       // anotherPi는 Double형으로 추론됨`

# 정수 데이터 타입 : Int

- 정수(소수점이 없는 수)를 저장하는 데 사용
  - 양수, 음수, 영(0) 값을 담을 수 있는 부호 있는(signed) 정수
  - 영과 양수만 담을 수 있는 부호 없는(unsigned) 정수
- 8비트, 16비트, 32비트, 64비트 정수를 지원
  - Int8, Int16, Int32, Int64 타입
- 부호 없는 정수
  - UInt8, UInt16, UInt32, UInt64 타입
- 애플은 특정 크기의 데이터 타입 보다 Int 데이터 타입을 권장
  - Int 데이터 타입은 해당 코드가 실행되는 플랫폼에 맞는 정수 크기를 사용
  - `var myAge : Int = 20   // 초깃값 20이 있으므로 : Int는 일반적으로 생략함`
- 32비트 부호 있는 정수 데이터 타입에 대한 최솟값과 최댓값을 출력
- \(출력하고 싶은 변수나 상수)
  - `print("Int32 Min = \(Int32.min) Int32 Max = \(Int32.max)")`
  - `Int32 Min = -2147483648 Int32 Max = 2147483647`

# 부동 소수점 데이터 타입: Double

- 소수점이 있는 숫자
  - 4353.1223
- Float와 Double 타입을 제공
- Double 타입
  - 64비트로 부동 소수점 수를 저장, 소수점 15자리 정확도
- Float 데이터 타입
  - 32비트로 부동 소수점 수를 저장, 소수점 6자리 정확도
- Double형이 기본
- `var myWeight : Double`
- `var myWeight : Double =  58.5`
  - 초깃값 58.5이 있으므로 : Double은 일반적으로 생략함

# 부울 데이터 타입 : Bool

- 참 또는 거짓(1 또는 0) 조건을 처리할 데이터 타입
- Boolean 데이터 타입을 처리하기 위하여 두 개의 불리언 상수 값(true/false) 사용
  - `var orangesAreOrange : Bool`
  - `var orangesAreOrange : Bool = true`
  - 초깃값 true가 있으므로 : Bool은 일반적으로 생략함

# 문자 데이터 타입 : Character

- 문자, 숫자, 문장 부호, 심볼 같은 유니코드(Unicode) 문자 하나를 저장
  - 스위프트에서의 문자들은 문자소 묶음(grapheme cluster)의 형태로 저장
    - 문자소 묶음은 하나의 문자를 표현하기 위하여 유니코드 코드 값들로 이루어짐
- var 변수명: Character ="초깃값"
- 주의 : 초깃값은 작은 따옴표가 아니고 큰 따옴표
  - `var myChar1  : Character`
  - `var myChar2 : Character = ":"`
  - `var myChar3 : Character = "X"  //:Character 생략불가,생략하면 String형임`
    - `print(type(of: myChar3))`
  - 유니코드를 이용하여 변수에 문자 'X'를 할당
    - `var myChar4 = "\u{0058}"`

# 문자열 데이터 타입 : String

- 단어나 문장을 구성하는 일련의 문자
- 저장, 검색, 비교, 문자열 연결, 수정 등의 기능을 포함
- 문자열 보간(string interpolation)을 사용하여 문자열과 변수, 상수, 표현식, 함수 호출의 조합으로 만들 수도 있음

```
var userName : String = "Kim"  // : String 생략하는 것이 일반적임
var age = 20
var message = "\(userName)의 나이는 \(age)입니다."
print(message) // Kim의 나이는 20입니다.
```

- https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/StringsAndCharacters.html#//apple_ref/doc/uid/TP40014097-CH7-ID285

# 특수 문자/이스케이프 시퀀스

- 표준 문자 세트뿐만 아니라 문자열에 개행, 탭, 또는 유니코드 값과 같은 항목을 지정할 수 있는 여러 특수 문자도 있음
- 이스케이프 시퀀스(escape sequence)
- 특수 문자들은 역슬래시를 접두어로 하여 구별
- `var newline = "\n"  // newline`
- 역슬래시로 시작되는 모든 문자는 특수 문자로 간주
- 역슬래시 문자 자체
  - `var backslash = "\\"`
- 일반적으로 많이 사용되는 특수 문자
  - \n – 개행
  - \r – 캐리지 리턴(carriage return)
  - \t – 수평 탭
  - \\ – 역슬래시
  - \" – 큰따옴표(문자열 선언부에 큰따옴표를 쓰고 싶을 경우에 사용됨)
  - \' – 작은따옴표(문자열 선언부에 작은따옴표를 쓰고 싶을 경우에 사용됨)
  - \u{nn} – nn 위치에 유니코드 문자를 표현하는 두 개의 16진수가 배치되는 1바이트 유니코드 스칼라
  - \u{nnnn} – nnnn 위치에 유니코드 문자를 표현하는 네 개의 16진수가 배치되는 2바이트 유니코드 스칼라
  - \U{nnnnnnnn} – nnnnnnnn 위치에 유니코드 문자를 표현하는 네 개의 16진수가 배치되는 4바이트 유니코드 스칼라

# 변수 : var

- 기본적으로 변수(variable)는 프로그램에서 사용될 데이터를 저장하기 위한 메모리 공간
- 변수에 할당된 값은 변경 가능
- `var myVariable = 10   // :Int`
- `var x = 0.0, y = 0.0, z = 0.0`

# 상수 : let

- 상수(constant)는 데이터 값을 저장하기 위하여 메모리 내의 명명된 공간을 제공한다는 점에서 변수와 비슷
- 어떤 값이 한번 할당되면 이후에 변경될 수 없음
- 상수는 코드 내에서 반복적으로 사용되는 값이 있을 경우에 유용
- 코드 내에서 반복적으로 사용되는 특정 값을 매번 사용하는 것보다, 그 값을 상수에 할당한 다음 코드 내에서 참조하면 코드 읽기가 더 쉬워짐
  - `let maximumNumber = 10`
  - `let π = 3.14159`
  - `let 🐶🐮 = "dogcow"  //[Edit]-[Emoji & Symbols]`
- 변수나 상수 명은 영문자, 숫자, Unicode(이모티콘, 중국어, 한글….)도 가능

# 상수와 변수 선언하기

- 변수는 var 키워드를 이용하여 선언되며, 변수를 생성할 때에 값을 가지고 초기화할 수도 있음
  - `var userCount = 10`
  - `var userCount : Int?`
- 상수는 let 키워드를 사용하여 선언
  - 선언하는 시점에서 상수에 값이 할당되어 초기화되고, 할당된 값을 수정할 수 없음
  - `let maxUserCount = 20`
- 애플은 코드의 효율성과 실행 성능을 높이기 위해서 변수(var)보다는 상수(let)를 사용하라고 권장

# 타입 어노테이션과 타입 추론

- 스위프트는 타입 안전(type safe) 프로그래밍 언어
  - 변수의 데이터 타입이 식별되면 그 변수는 다른 타입의 데이터를 저장하는 데 사용될 수 없음
  - 변수가 선언된 후에도 다른 데이터 타입을 저장할 수 있는 느슨한 타입(loosely typed)의 언어와 대조적
- 상수와 변수의 타입을 식별하는 방법은 두 가지
- 첫 번째 방법은 변수 또는 상수가 코드 내에서 선언되는 시점에 타입 어노테이션(type annotation)을 사용하는 것
  - 변수 또는 상수 이름 다음에 타입 선언을 두면 됨
  - Int 타입의 userCount 라는 이름의 변수를 선언
  - `var userCount : Int = 10  // : Int가 type annotation`
- 선언부에 타입 어노테이션이 없으면 스위프트 컴파일러는 상수 또는 변수의 타입을 식별하기 위하여 타입 추론(type inference) 사용
- 해당 상수 또는 변수에 값이 할당되는 시점에서 그 값의 타입을 확인하고 그와 같은 타입처럼 사용
  - `var signalStrength = 2.231     // var signalStrength : Double = 2.231`
  - `let companyName = "My Company"`
  - signalStrength라는 변수를 Double 타입(스위프트의 타입 추론에서 모든 부동 소수점 수는 Double 타입)
  - companyName이라는 상수는 String 타입으로 간주

# 타입 어노테이션과 타입 추론

- 상수를 선언할 때도 타입 어노테이션을 사용하면 나중에 코드에서 값을 할당할 수 있다.

```
let bookTitle: String
var book = true
if book {
    bookTitle = "iOS"
} else {
    bookTitle = "Android"
}
print(bookTitle)
```

- 상수에는 값을 한 번만 할당할 수 있다.
- 이미 값이 할당된 상수에 다시 값을 할당하려고 시도한다면 구문 에러(syntax error)가 발생

# 튜플(Tuple)

- 튜플은 스위프트 프로그래밍 언어에서 가장 강력한 기능 중 하나
- 여러 값을 하나의 개체에 일시적으로 묶는 방법
- 튜플에 저장되는 항목들은 어떠한 타입도 될 수 있으며, 저장된 값들이 모두 동일한 타입이어야 한다는 제약도 없음
- `let myTuple = (10, 12.1, "Hi")`
- 튜플의 요소들은 여러 다른 방법들을 사용하여 접근할 수 있음
- 특정 튜플 값은 인덱스 위치를 참조하면 간단하게 접근
  - 맨 첫 번째 값은 인덱스 0
- 인덱스 2 위치를 추출하고 그 값을 새로운 문자열 변수에 할당
  - ```
    let myTuple = (10, 12.1, "Hi")
    var myString = myTuple.2
    print(myString)   //출력되는 값은?
    ```
    ```

# 튜플(Tuple)

- 단 한 줄의 코드로 튜플의 모든 값을 추출하여 변수 또는 상수에 할당
  - `let myTuple = (10, 12.1, "Hi")  //과제 : myTuple의 자료형`
  - `let (myInt, myFloat, myString) = myTuple`
- 튜플의 값을 선택적으로 추출하는 데 사용될 수 있으며, 무시하고 싶은 값에 밑줄을 사용하면 그 값은 무시
  - `var (myInt, _, myString) = myTuple  //부동 소수점 수는 무시`
- 튜플을 생성할 때 각 값에 이름을 할당할 수도 있음
  - `let myTuple = (count: 10, length: 12.1, message: "Hi") //과제 : myTuple의 자료형`
- 튜플에 저장된 값에 할당된 이름은 각 값을 참조하는 데 사용
- myTuple 인스턴스의 message 값을 출력하는 코드
  - `print(myTuple.message)   //?`
- 튜플의 가장 강력한 점은 함수에서 여러 값들을 한 번에 반환하는 것