→ Hello, Swift

실전 프로젝트로 배우는 iOS 개발

© Jiwon Park 2022

1. 상수와 변수

© Jiwon Park 2022

상수 (constant)

상수는 `let`을 사용하여 선언한다.
한 번 값을 할당한 상수는 값을 재할당 할 수 없다.

```
// constant
// let [constant name]: data type = value
let a: Int = 100
a = 200 // error!
```

© Jiwon Park 2022

변수 (variable)

변수는 `var` 를 사용하여 선언한다.
값을 여러번 재할당 할 수 있다.

```
// variable
// var [variable name]: data type = value
var b: Int = 200
b = 300
```

© Jiwon Park 2022

2. 데이터 타입

© Jiwon Park 2022

데이터 타입

- `Int` : 64bit 정수형
- `UInt` : 부호가 없는 64bit 정수형
- `Float` : 32bit 부동 소수점
- `Double` : 64bit 부동 소수점
- `Bool` : true, false 값
- `Character` : 문자
- `String` : 문자열
- `Any` : 모든 타입을 지칭하는 키워드

© Jiwon Park 2022

3. 컬렉션 타입

© Jiwon Park 2022

컬렉션 타입이란?

컬렉션 타입은 데이터들의 집합을 의미한다.

© Jiwon Park 2022

Array

데이터 타입의 값들을 **순서대로** 저장하는 리스트

```
// 빈 Array 생성
var numbers: Array<Int> = Array<Int>()
numbers.append(1)
numbers.append(2)
numbers.insert(4, at: 2)
numbers.remove(at: 0)
// 또 다른 선언 방법
var names: [String] = []
```

© Jiwon Park 2022

Dictionary

순서없이 키(key) 와 **값(value)** 한 쌍으로 데이터를 저장하는 컬렉션 타입.

```
//var dic: Dictionary<String, Int> = Dictionary<String, Int>()
var dic: [String: Int] = ["박지원": 1]
dic["김철수"] = 3
dic["김민지"] = 5

dic.removeValue(forKey: "김민지")
```

© Jiwon Park 2022

Set

같은 데이터 타입의 값을 **순서없이** 저장하는 리스트

```
var set: Set = Set<Int>()

set.insert(10)
set.insert(20)
set.insert(30)
set.insert(30)

set.remove(20)
```

© Jiwon Park 2022

4. 함수

© Jiwon Park 2022

함수

함수는 작업의 가장 작은 단위이자 코드의 집합

```
func 함수명(parameter 이름: 데이터 타입) -> 반환 타입 {
    return 반환 값
}
```

© Jiwon Park 2022

기본 파라미터 설정

```
func greeting(friend: String, me: String = "Jiwon") {
    print("Hello \(friend), I'm \(me)")
}
```

© Jiwon Park 2022

전달인자 레이블

아래와 같이 레이블을 함수를 호출할 때 사용하는 레이블과
함수 내에서 사용하는 레이블을 별도로 설정할 수 있다.

```
func sendMessage(from myName: String, to name: String) -> String {
    return "Hello \(name), I'm \(myName)"
}

sendMessage(from: "Jiwon", to: "Jason")
```

© Jiwon Park 2022

wildcard 레이블

wildcard 레이블을 설정하면 함수를 호출할 때 전달인자 레이블을 생략할 수 있다.

```
func sendMessage(_ name: String) -> String {
    return "Hello \(name)"
}

sendMessage("Jiwon")
```

© Jiwon Park 2022

가변 매개변수

여러 개의 전달인자를 받는 함수를 정의할 때 가변 매개변수를 사용할 수 있다.

```
func sendMessage(me: String, friends: String...) -> String {
    return "Hello \(friends), I'm \(me)"
}

sendMessage(me: "Jiwon", friends: "Jason", "Albert", "Stella")
```

© Jiwon Park 2022

5. 조건문

© Jiwon Park 2022

조건문

주어진 조건에 따라서 어플리케이션을 다르게 동작하도록 하는 것

© Jiwon Park 2022

if-else 문

```
if age < 0 {
    print("잘못된 입력입니다.")
} else if age < 19 {
    print("미성년자입니다")
} else {
    print("성년입니다")
}
```

© Jiwon Park 2022

switch 문

```
switch color {
    case "blue":
        print("파란색")
    case "green":
        print("녹색")
    case "yellow":
        print("노란색")
    default:
        print("찾는 색상이 없습니다.")
}
```

© Jiwon Park 2022

아래와 같이 case를 값의 범위로 설정할 수 있다.

```
switch temperature {
    case -20...9:
        print("겨울")
    case 10...14:
        print("가을")
    case 15...25:
        print("봄")
    case 26...35:
        print("여름")
}
```

© Jiwon Park 2022

guard 문

일단 이런게 있다만 알아두고 뒤에서 살펴보도록 하자.

© Jiwon Park 2022

6. 반복문

© Jiwon Park 2022

반복문

반복적으로 코드가 실행되도록 만드는 구문

© Jiwon Park 2022

for-in

```
for 루프상수 in 순회대상 {
    // 실행할 코드
}
```

```
for i in 1...4 {
    print(i)
}
```

© Jiwon Park 2022

```
let array = [1,2,3,4,5]

for i in array {
    print(i)
}
```

© Jiwon Park 2022

while

조건식은 반드시 `Bool` 타입

```
while 조건식 {
    // 실행할 코드
}
```

```
var number = 0
while number < 10 {
    number += 1
}
```

© Jiwon Park 2022

repeat-while

다른 언어의 do-while문과 동일

```
repeat {
    x += 2
} while x < 5
```

© Jiwon Park 2022

7. 옵셔널

© Jiwon Park 2022

옵셔널?

값이 있을수도 있고, 없을 수도 있습니다.

© Jiwon Park 2022

옵셔널 사용법

```
var name: String = "안녕하세요"
name = ""
name = nil // error!
```

```
var name: String? = nil
```

© Jiwon Park 2022

8. 옵셔널 바인딩

© Jiwon Park 2022

명시적 해제

- 강제 해재
- 비강제 해제(옵셔널 바인딩)

묵시적 해제

- 컴파일러에 의한 자동 해제
- 옵셔널의 묵시적 해제

© Jiwon Park 2022

강제해재

→ 위험하므로 권장하지 않음

```
var number: Int? = 3
print(number) // Optional(3)
print(number!) // 3
```

© Jiwon Park 2022

if 문을 이용한 추출

```
var number: Int? = 3
if let result = number {
    print(result)
} else {
  // 값 추출이 실패한 경우
}
```

© Jiwon Park 2022

`guard`를 이용한 추출

```
func test() {
    let number: Int? = 5
    guard let result = number else { return }
    print(result)
}
```

if를 이용하면 해당 옵셔널을 if문 스코프 내에서만 사용할 수 있지만, guard문을 이용하면 guard문 이후에도 해당 함수 스코프 내에서 계속 해당 옵셔널 변수를 사용할 수 있다.

© Jiwon Park 2022

묵시적 해제 - 컴파일러를 통한 해제

```
let value: Int? = 6
print(value == 6) // true
```

© Jiwon Park 2022

옵셔널의 묵시적 해제

```
let string = "12"
var stringToInt: Int! = Int(string)
print(StringToInt + 1) // 13
```

© Jiwon Park 2022

9. 구조체

© Jiwon Park 2022

구조체 사용

```
struct 구조체명 {
    구조체 멤버 // 프로퍼티와 메서드
}
```

© Jiwon Park 2022

```swift
struct User {
    var name: String
    var age: Int

    func information() {
        print("\(nickname) \(age)")
    }
}

var user = User(name: "Jiwon", age: 22)

user.name // "Jiwon"

user.information() // User
```

© Jiwon Park 2022

10. 클래스

© Jiwon Park 2022

클래스 사용

```
class 클래스명 {
    클래스 멤버 // 프로퍼티와 메서드
}
```

© Jiwon Park 2022

```swift
class Dog {
    var name: String = ""
    var age: Int = 0

    init() {
        // 생성자 내용
    }

    func introduce() {
        print("name \(name) age \(age)")
    }
}

var dog = Dog()
dog.name == "Coco"
dog.age == 3
```

© Jiwon Park 2022

11. 초기화 구문 `init`

© Jiwon Park 2022

클래스 구조체 또는 열거형의 인스턴스를 사용하기 위한 준비 과정

```
struct User {
    var nickname: String = "Jiwon"
    var age: Int = 0

    init(nickname: String, age: Int) {
        self.nickname = nickname
        self.age = age
    }
}

let user = User(nickname: "Jiwon", age: 22)
user.nickname
user.age
```

© Jiwon Park 2022

오버라이딩

```
struct User {
    var nickname: String = "Jiwon"
    var age: Int = 0

    init(nickname: String, age: Int) {
        self.nickname = nickname
        self.age = age
    }

    init(age: Int) {
        self.nickname = "default"
        self.age = age
    }
}
```

© Jiwon Park 2022

deinitializer

```
struct User {
    ...
    deinit {
        print("deinit")
    }
}

var user: User? = User(age: 22)
user = nil // deinit 호출
```

© Jiwon Park 2022

12. 프로퍼티

© Jiwon Park 2022

프로퍼티의 종류

클래스, 구조체 또는 열거형 등에 관련된 값을 뜻한다.

- 저장 프로퍼티
  - 인스턴트에 값을 저장하는 것
- 연산 프로퍼티
- 타입 프로퍼티

© Jiwon Park 2022

저장 프로퍼티

```
struct Dog {
    var name: String
    let gender: String
}

var dog = Dog(name: "Jiwon", gender: "Male")
```

구조체는 let으로 선언하면 내부의 저장 프로퍼티들을 변경할 수 없지만 클래스는 참조 타입이기 때문에 let으로 선언해도 저장 프로퍼티들을 변경할 수 있다.

© Jiwon Park 2022

계산 프로퍼티

```
struct Stock {
    var averagePrice: Int
    var quantity: Int
    var purchasePrice: Int {
        get {
            return averagePrice * quantity
        }
        set(newPrice) {
            averagePrice = newPrice / quantity
        }
    }
}

var stock = Stock(averagePrice: 2300, quantity: 3)
```

© Jiwon Park 2022

프로퍼티 옵저버

```
class Account {
    var credit: Int = 0 {
        willSet {
            print("잔액이 \(credit)원에서 \(newValue)원으로 변경될 예정입니다.")
        }

        didSet {
            print("잔액이 \(oldValue)원에서 \(credit)원으로 변경되었습니다.")
        }
    }
}

var account = Account()
account.credit = 1000
```

© Jiwon Park 2022

타입 프로퍼티

```
struct SomeStructure {
    static var storedTypeProperty = "Some value."
    static var computedTypeProperty: Int {
        return 1
    }
}

SomeStructure.computedTypeProperty
SomeStructure.storedTypeProperty
SomeStructure.storedTypeProperty = "hello"
SomeStructure.storedTypeProperty
```

© Jiwon Park 2022

13. 클래스와 구조체의 차이

© Jiwon Park 2022

공통점

- 값을 저장할 **프로퍼티**를 선언할 수 있다
- 함수적 기능을 하는 **메서드**를 선언할 수 있다
- 내부 값에 .을 사용하여 접근할 수 있다
- **생성자**를 사용해 초기 상태를 설정할 수 있다
- extension을 사용하여 기능을 확장할 수 있다
- protocol을 채택하여 기능을 설정할 수 있다

© Jiwon Park 2022

차이점(1)

클래스(class)

- **참조** 타입
- ARC*로 메모리를 관리
- **상속** 가능
- 타입 캐스팅을 통해 런타임에서 클래스 인스턴스의 타입을 확인할 수 있음
- deinit을 사용해 클래스 인스턴스의 메모리 할당을 해제할 수 있음
- 같은 클래스 인스턴스를 여러 개의 변수에 할당한 뒤 값을 변경 시키면 모든 변수에 영향을 줌 (**메모리가 복사됨**)

* 추후 설명할 예정

© Jiwon Park 2022

차이점(2)

구조체(struct)

- **값** 타입
- 구조체 변수를 새로운 변수에 할당할 때마다 새로운 구조체가 할당됨
  → 같은 구조체를 여러 개의 변수에 할당한 뒤 값을 변경시키더라도 다른 변수에 영향을 주지 않음 (**값 자체를 복사**)

© Jiwon Park 2022

참조타입과 값타입의 차이점

```
class SomeClass {
    var count: Int = 0
}

struct SomeStruct {
    var count: Int = 0
}

// 참조타입
var class1 = SomeClass()
var class2 = class1
var class3 = class1

class3.count = 2
class1.count // 2
```

© Jiwon Park 2022

```
var struct1 = SomeStruct()
var struct2 = struct1
var struct3 = struct1

struct2.count = 3
struct3.count = 4

struct1.count // 0
struct2.count // 3
struct3.count // 4
```

© Jiwon Park 2022

14. 상속

© Jiwon Park 2022

Class는 다른 class로부터 내부 멤버들을 물려받을 수 있다.

© Jiwon Park 2022

```
class Vehicle {
    var currentSpeed = 0.0
    var description: String {
        return "traveling at \(currentSpeed) miles per hour"
    }
    func makeNoise() {
        // do nothing - an arbitrary vehicle doesn't necessarily make a noise
    }
}
```

© Jiwon Park 2022

```
class Bicycle: Vehicle {
    var hasBasket = false
}

var bicycle = Bicycle()
bicycle.currentSpeed // 0
bicycle.currentSpeed = 15.0
```

© Jiwon Park 2022

메소드 오버라이딩

```
class Train: Vehicle {
    override func makeNoise() {
        super.makeNoise()
        print("Choo Choo")
    }
}

let train = Train()
train.makeNoise() // "Choo Choo"
```

© Jiwon Park 2022

프로퍼티 오버라이딩

```
class Car: Vehicle {
    var gear = 1
    override var description: String {
        return super.description + " in gear \(gear)"
    }
}

let car = Car()
car.currentSpeed = 30.0
car.gear = 2
print(car.description) // traveling at 30.0 miles per hour in gear 2
```

© Jiwon Park 2022

```
class AutomaticCar: Car {
    override var currentSpeed: Double {
        didSet {
            gear = Int(currentSpeed / 10.0) + 1
        }
    }
}

let automatic = AutomaticCar()
automatic.currentSpeed = 35.0
print(automatic.description) // traveling at 35.0 miles per hour in gear 4
```

© Jiwon Park 2022

Final 키워드를 사용하면 다른 클래스에서 해당 클래스를 상속받을 수 없다.

```
class Vehicle {
    final var currentSpeed = 0.0
    ...
}
class AutomaticCar: Car {
    override var currentSpeed: Double { // error!
        didSet {
            gear = Int(currentSpeed / 10.0) + 1
        }
    }
}
```

© Jiwon Park 2022

15. 타입캐스팅

© Jiwon Park 2022

인스턴스의 타입을 확인하거나 어떠한 클래스의 인스턴스를 해당 클래스 계층 구조의 슈퍼 클래스나 서브 클래스로 취급하는 방법

© Jiwon Park 2022

is 연산자

```
class MediaItem {
    var name: String
    init(name: String) {
        self.name = name
    }
}
```

© Jiwon Park 2022

```
class Movie: MediaItem {
    var director: String
    init(name: String, director: String) {
        self.director = director
        super.init(name: name)
    }
}

class Song: MediaItem {
    var artist: String
    init(name: String, artist: String) {
        self.artist = artist
        super.init(name: name)
    }
}
```

© Jiwon Park 2022

```swift
let library = [
    Movie(name: "Casablanca", director: "Michael Curtiz"),
    Song(name: "Blue Suede Shoes", artist: "Elvis Presley"),
    Movie(name: "Citizen Kane", director: "Orson Welles"),
    Song(name: "The One And Only", artist: "Chesney Hawkes"),
    Song(name: "Never Gonna Give You Up", artist: "Rick Astley")
]

var movieCount = 0
var songCount = 0

for item in library {
    if item is Movie { // is 연산자
        movieCount += 1
    } else if item is Song {
        songCount += 1
    }
}
```

© Jiwon Park 2022

다운캐스팅

```
for item in library {
    if let movie = item as? Movie {
        print("Movie: '\(movie.name)', dir. \(movie.director)")
    } else if let song = item as? Song {
        print("Song: '\(song.name)', by \(song.artist)")
    }
}
```

© Jiwon Park 2022

16. `assert`와 `guard`

© Jiwon Park 2022

`assert` 함수

- 특정 조건을 체크하고, 조건이 성립되지 않으면 메세지를 출력하게 할 수 있는 함수
- `assert` 함수는 **디버깅 모드에서만 동작**하고 주로 디버깅 중 조건의 검증을 위하여 사용한다

```
var value = 0
assert(value == 0)

value = 2
assert(value == 0, "값이 0이 아닙니다") // error occured
```

© Jiwon Park 2022

`guard`문

- 무언가를 검사하여 그 다음에 오는 코드의 실행 여부를 결정
- `guard` 문에 주어진 조건문이 거짓일 때 구문이 실행됨

```
// value가 0일때만 실행
func guardTest(value: Int) {
    guard value == 0 else { return }
    print("안녕하세요")
}
```

© Jiwon Park 2022

`guard` 문을 이용한 옵셔널 바인딩

```
func guardTest(value: Int?) {
    guard let value = value else { return }
    print(value)
}

guardTest(value: 2) // 2
guardTest(value: nil) // (nothing)
```

© Jiwon Park 2022

17. protocol

© Jiwon Park 2022

protocol

특정 역할을 하기 위한 메서드, 프로퍼티, 기타 요구사항 등의 청사진

→ typescript의 interface와 비슷한 개념

```
protocol FullyNames {
    var fullName: String { get set }
    func printFullName()
    init()
}
```

© Jiwon Park 2022

```
struct Person: FullyNames {
    var fullName: String
    func printFullName() {
        print(fullName)
    }
    init() {
        fullName = "unknown"
    }
}

class Company: FullyNames {
    var fullName: String
    func printFullName() {
        print(fullName)
    }
    required init() {
        fullName = "unknown"
    }
}
```

© Jiwon Park 2022

18. extension

© Jiwon Park 2022

extension

기존의 클래스, 구조체, 열거형, 프로토콜에 새로운 기능을 추가하는 기능

© Jiwon Park 2022

extension의 종류

- 연산 타입 프로퍼티 / 연산 인스턴스 프로퍼티
- 타입 메서드 / 인스턴스 메서드
- 이니셜라이저
- 서브스크립트
- 중첩 타입
- 특정 프로토콜을 준수할 수 있도록 기능 추가

© Jiwon Park 2022

```swift
extension Int {
    var isEven: Bool {
        return self % 2 == 0
    }
    var isOdd: Bool {
        return self % 2 == 1
    }
}
```

© Jiwon Park 2022

```
extension String {
    func convertToInt() -> Int? {
        return Int(self)
    }
}
```

© Jiwon Park 2022

19. 열거형

© Jiwon Park 2022

열거형(enum)

연관성이 있는 값을 모아 놓은 것을 말한다

© Jiwon Park 2022

```
enum CompassPoint: String {
    case north = "N"
    case south = "S"
    case east = "E"
    case west = "W"
}

enum Planet {
    case mercury, venus, earth, mars, jupiter, saturn, uranus, neptune
}

var direction = CompassPoint.east // east
direction = .west // west
```

© Jiwon Park 2022

```
switch direction {
    case .north:
        print(direction.rawValue)
    case .south:
        print(direction.rawValue)
}
```

© Jiwon Park 2022

```
let direction2 = CompassPoint(rawValue: "N") // north
```

© Jiwon Park 2022

연관값

```
enum PhoneError {
    case unknown
    case batteryLow(String)
}

let error = PhoneError.batteryLow("충전하세요")

switch error {
    case .batteryLow(let message):
        print(message)
    default:
        print("알 수 없는 오류")
}
```

© Jiwon Park 2022

20. 옵셔널 체이닝

© Jiwon Park 2022

옵셔널 체이닝

옵셔널에 속해있는 `nil` 일지도 모르는 프로퍼티, 메서드, 서브스크립션 등을 가져오거나 호출할 때 사용할 수 있는 일련의 과정

© Jiwon Park 2022

```swift
struct Developer {
    let name: String
}

struct Company {
    let name: String
    var developer: Developer?
}

var developer = Developer(name: "Kim")
var company = Company(name: "Apple", developer: developer)
print(company.developer?.name) // Optional("Kim")
print(company.developer!.name) // "Kim"
```

© Jiwon Park 2022

21. try-catch

© Jiwon Park 2022

try-catch

프로그램 내에서 에러가 발생한 상황에 대해 대응하고 이를 복구하는 과정

이벤트 lifecycle

1. 발생 (throwing)
2. 감지 (catching)
3. 전파 (propagating)
4. 조작 (manipulating)

© Jiwon Park 2022

Error throwing

```
enum PhoneError: Error {
    case unknown
    case batteryLow(batteryLevel: Int)
}

throw PhoneError.batteryLow(batteryLevel: 20) // error occured
```

© Jiwon Park 2022

```swift
func checkPhoneBatteryStatus(batteryLevel: Int) throws -> String{
    guard batteryLevel != -1 else {
        throw PhoneError.unknown
    }
    guard batteryLevel > 20 else {
        throw PhoneError.batteryLow(batteryLevel: 20)
    }
    return "배터리 잔량이 충분합니다"
}
```

© Jiwon Park 2022

do-catch

```
do {
    try checkPhoneBatteryStatus(batteryLevel: 20)
} catch PhoneError.unknown {
    print("알 수 없는 오류")
} catch PhoneError.batteryLow(let batteryLevel) {
    print("배터리 잔량이 \(batteryLevel)% 남았습니다")
} catch {
    print("예기치 못한 오류 : \(error)")
}

let status = try? checkPhoneBatteryStatus(batteryLevel: 30) // Optional("배터리 잔량이 충분합니다")

let status = try! checkPhoneBatteryStatus(batteryLevel: 30) // "배터리 잔량이 충분합니다"
```

© Jiwon Park 2022

22. 클로저

© Jiwon Park 2022

클로저

코드에서 전달 및 사용할 수 있는 독립 기능 블록이며, 일급 객체의 역할을 할 수 있음

전달 인자로 보낼 수 있고, 변수/상수 등으로 저장하거나 전달할 수 있으며, 함수의 반환 값이 될 수도 있다

© Jiwon Park 2022

```
func hello() {
    print("안녕하세요")
}

let hello2 = { print("안녕하세요") }
```

```
{ (매개 변수) -> 리턴 타입 in
    코드
}
```

© Jiwon Park 2022

```
let hello = { () -> () in
    print("Hello")
}

hello() // "Hello"

let hello2 = { (name: String) -> String in
    return "Hello, \(name)"
}

hello2(name: "Kim") // error
hello2("Kim") // "Hello, Kim"
```

© Jiwon Park 2022

```
func doSomething(closure: () -> ()) {
    closure()
}

doSomething(closure: hello)
doSomething() {
    print("Hello")
}
```

© Jiwon Park 2022

23. 고차함수

© Jiwon Park 2022

고차함수

다른 함수를 전달 인자로 받거나 함수 실행의 결과를 함수로 반환하는 함수

- map
- filter
- reduce

© Jiwon Park 2022

```swift
// map
let numbers = [1, 2, 3, 4, 5]
let mapArray = numbers.map { (number) -> Int in
    return number * 2
}

// filter
let intArray = [10, 5, 20, 13, 4]
let filterArray = intArray.filter { $0 > 5 }

// reduce
let someArray = [1, 2, 3, 4, 5]
let reduceResult = someArray.reduce(0) {
    (result: Int, element: Int) -> Int in
    print("\(result) + \(element)")
    return result + element
}
```

© Jiwon Park 2022