iOS 프로그래밍

스위프트 기초에서 응용까지

코드스쿼드

강사소개

IP 기반 멀티미디어 프로토콜 설계 및 프레임워크 개발

- Windows, Linux, Solaris, Mac OS X
- Multi-platform Core Engine 개발

아이폰 개발 12년차

osxdev.org 운영진, Let’Swift 컨퍼런스 오거나이저

iPhone Hacking (2010), Xcode4(2011), Cocoa Internals(2017) 저자 그외 다수 번역

맥/아이폰 기술 컨설팅, 개발, 강의

전 NHN NEXT 모바일 전공 교수

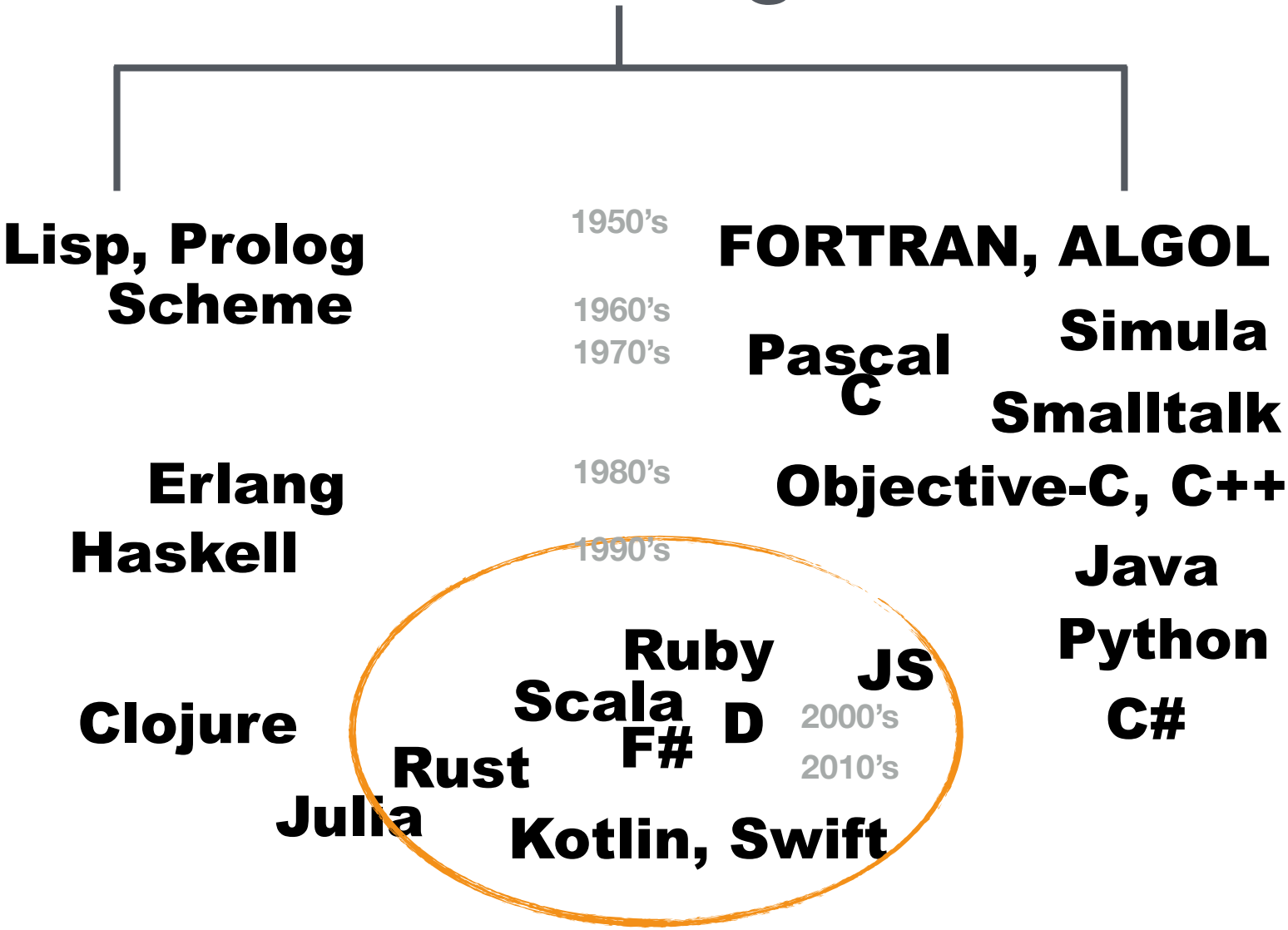
현 코드스쿼드 대표 / 모바일 마스터

학습 목표

* 스위프트 기본 문법 다지기
* 다른 언어와 차이점 이해하기
* 모르는 건 직접 써보기
* 스위프트 객체 중심 프로그래밍 개념 이해하기


START

Two Paradigms



Class & Structures

Closures

Generics

Operators

Namespace

.map.reduce

Currying

Initialization

Enumerations

inferred type

Bridging

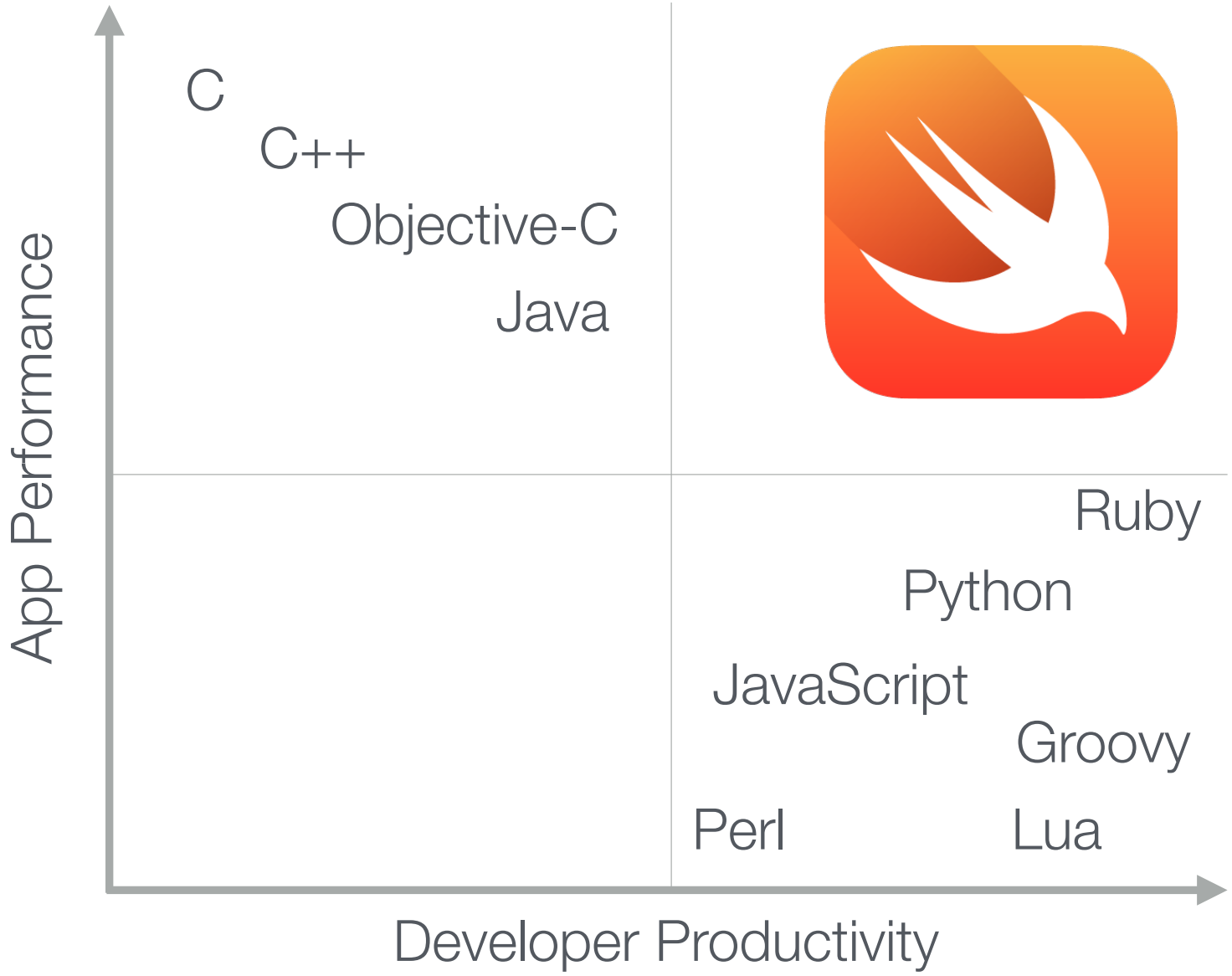
Strings and Char

ARC

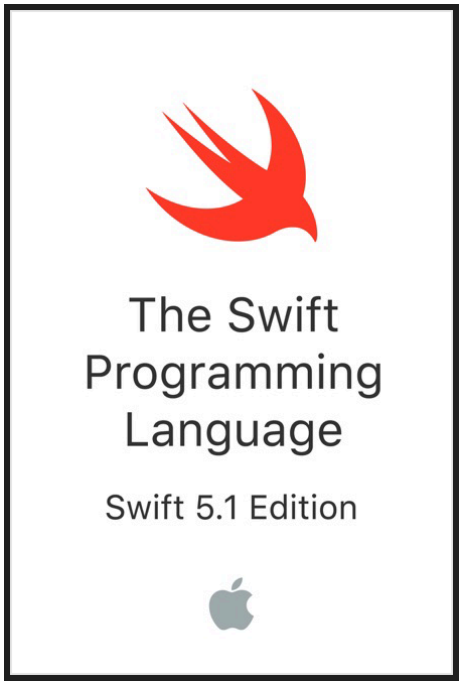
Protocol

Optional

Objective-C


C
C++
Objective-C
Java
Ruby
Python
JavaScript
Groovy
Perl
Lua
App Performance
Developer Productivity


Harry Potter
LF-BLOOD
RINCE
HEYDAY FILMS
Harry Potter Publishing Rights © J.K.R.
Harry Potter characters, names and related indicia are trademarks of and © Warner Bros. Ent. All Rights Reserved.
WARNER BROS. PICTURES

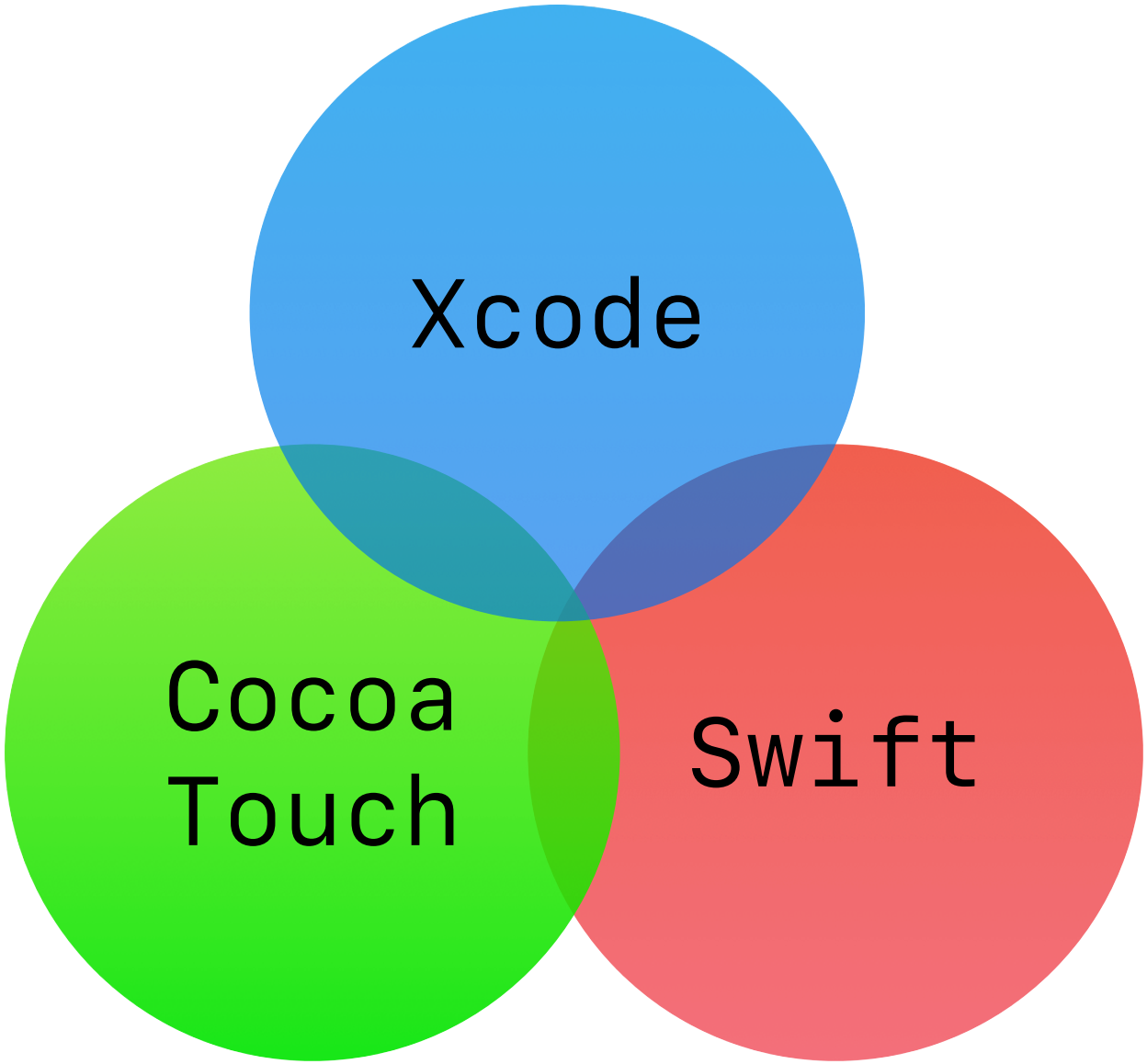


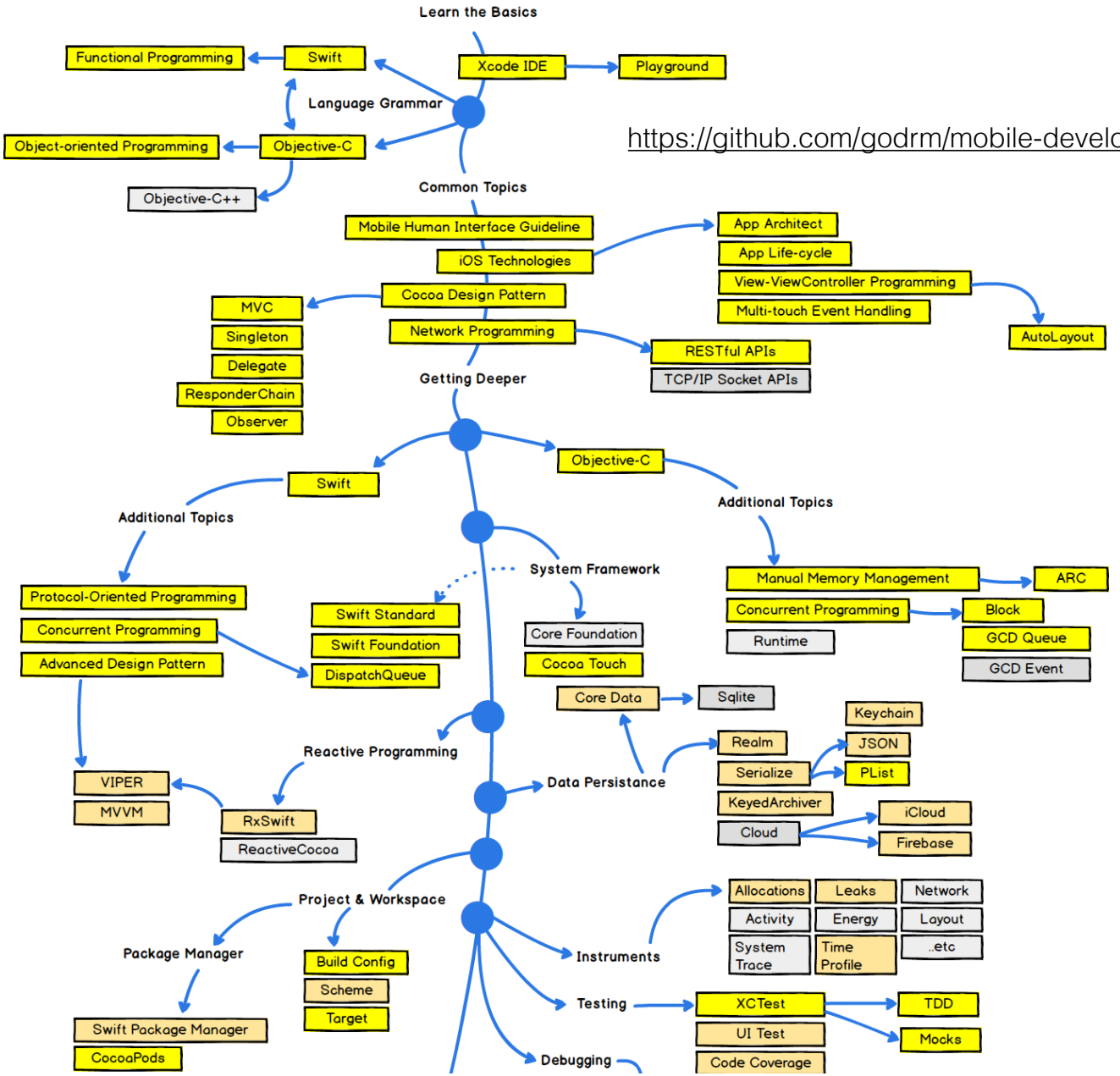
https://itunes.apple.com/kr/book/app-development-with-swift/id1219117996?l=en&mt=11

학습 참고 사이트

- 동영상 강의
  - https://www.inflearn.com/course/스위프트-기본-문법/
  - http://tryhelloworld.co.kr/courses/ios-swift입문
  - https://www.inflearn.com/course/stanford-ios-한글자막-강의/
- udacity 강의
  - https://classroom.udacity.com/courses/ud1022
  - https://www.udacity.com/course/intro-to-ios-app-development-with-swift--ud585
- https://www.raywenderlich.com/category/swift
- Swift Korea 페이스북 그룹


Cocoa
Internals
오브젝티브-C와 스위프트, 멀티 패러다임의 시작
김정 지음
프로그래밍인사이트


Xcode
Cocoa
Touch
Swift

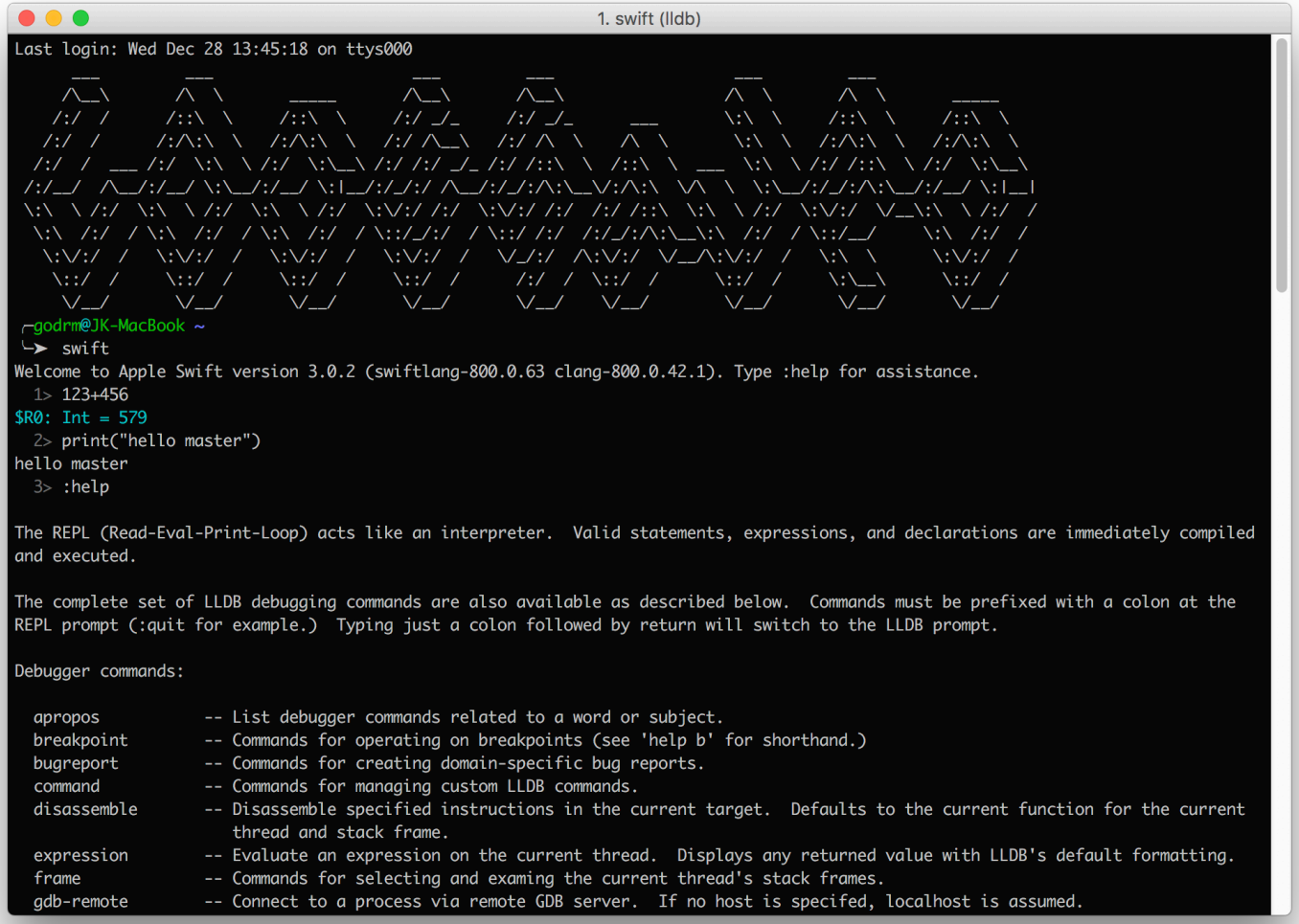


https://github.com/godrm/mobile-developer-roadmap

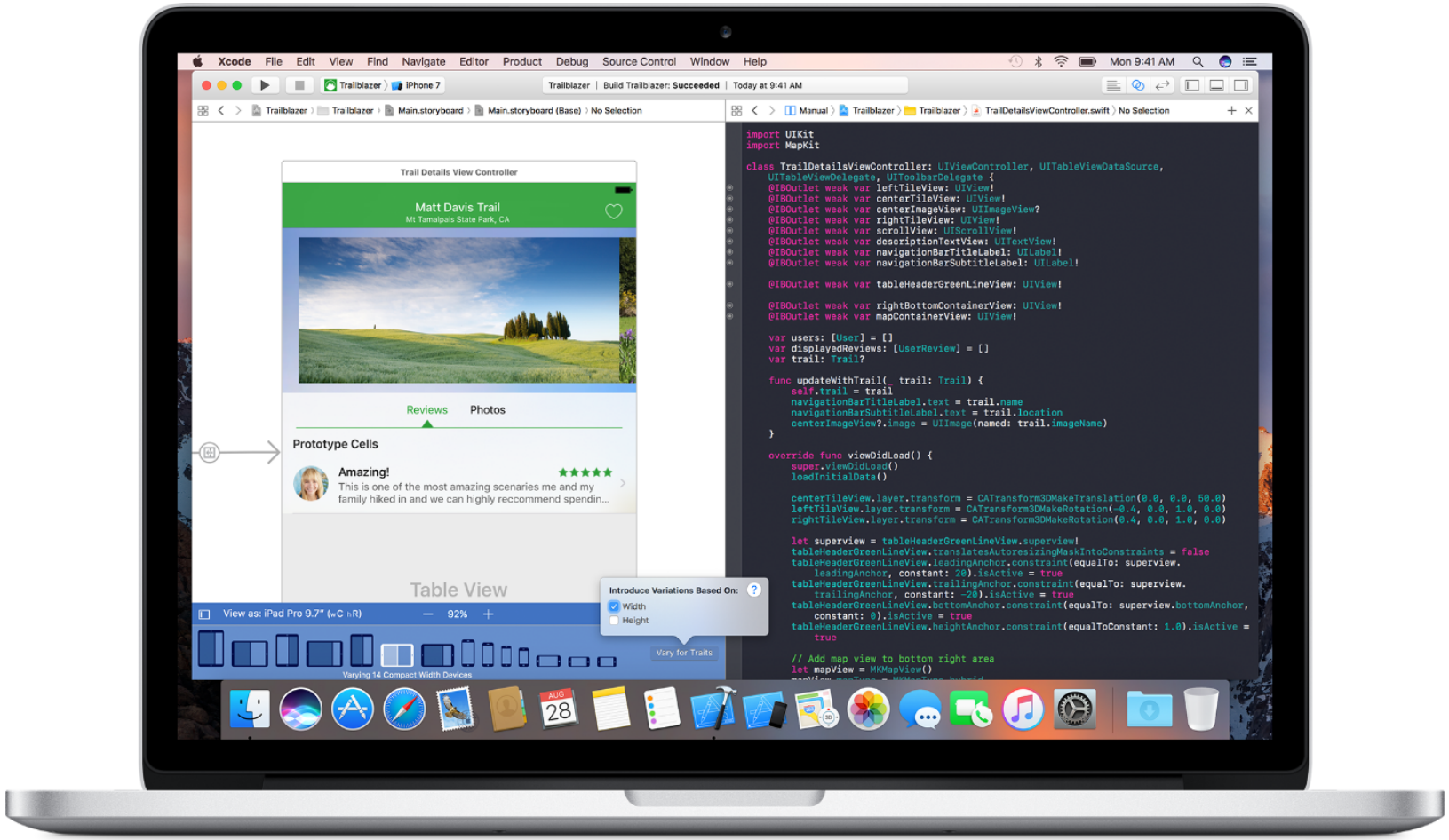
스위프트 개발 환경

REPL, Xcode, Playground

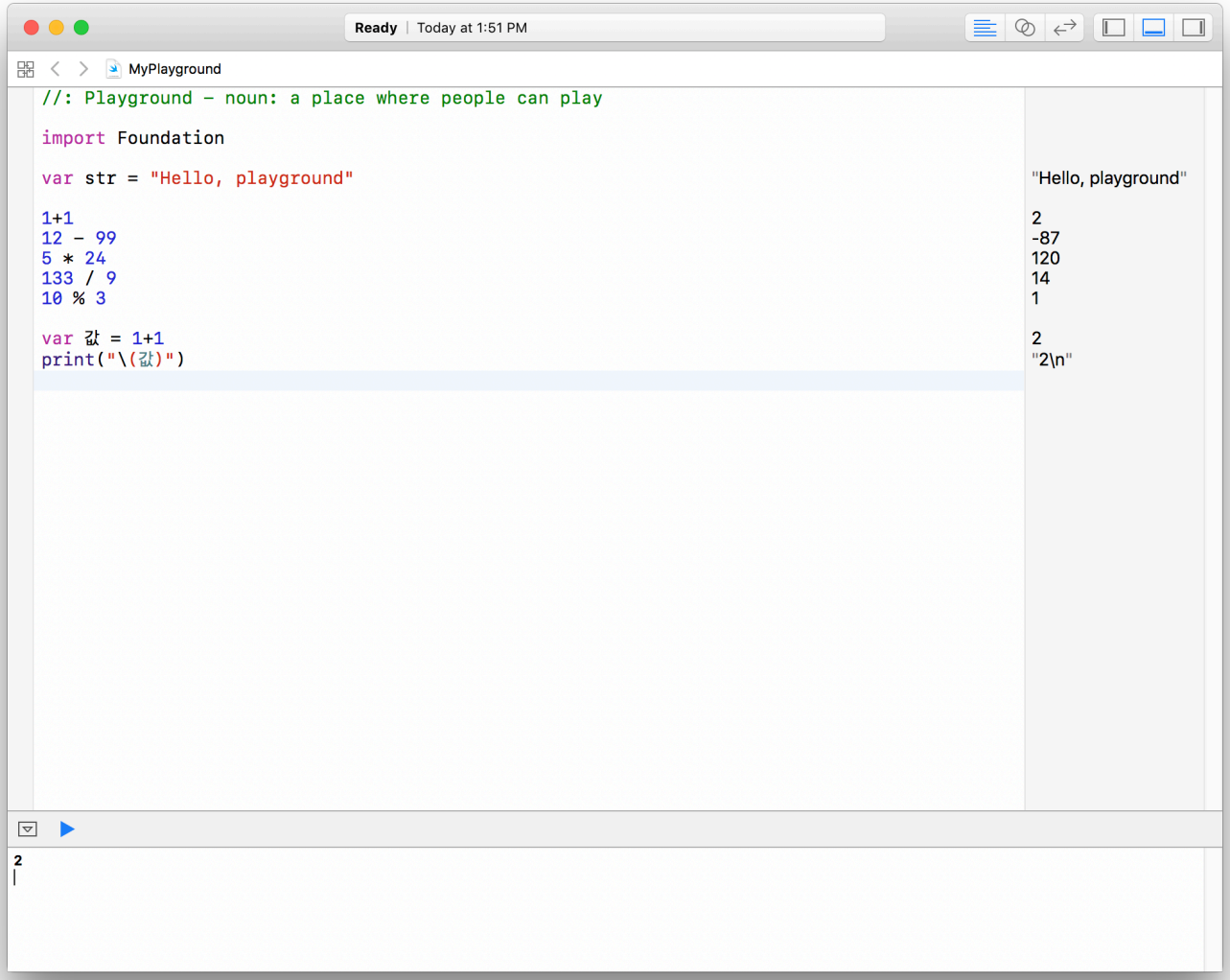
swift 명령어(REPL)



Xcode


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Mon 9:41 AM
Trailblazer | Build Trailblazer: Succeeded | Today at 9:41 AM
Trail Details View Controller
Matt Davis Trail
Mt Tamalpais State Park, CA
Reviews
Photos
Prototype Cells
Amazing!
This is one of the most amazing scenaries me and my family hiked in and we can highly reccommend spendin...
Table View
Introduce Variations Based On:
Width
Height
Vary for Traits
Varying 14 Compact Width Devices
import UIKit
import MapKit
class TrailDetailsViewController: UIViewController, UITableViewDataSource,
UITableViewDelegate, UIToolbarDelegate {
@IBOutlet weak var leftTileView: UIView!
@IBOutlet weak var centerTileView: UIView!
@IBOutlet weak var centerImageView: UIImageView?
@IBOutlet weak var rightTileView: UIView!
@IBOutlet weak var scrollView: UIScrollView!
@IBOutlet weak var descriptionTextView: UITextView!
@IBOutlet weak var navigationBarTitleLabel: UILabel!
@IBOutlet weak var navigationBarSubtitleLabel: UILabel!
@IBOutlet weak var tableHeaderGreenLineView: UIView!
@IBOutlet weak var rightBottomContainerView: UIView!
@IBOutlet weak var mapContainerView: UIView!
var users: [User] = []
var displayedReviews: [UserReview] = []
var trail: Trail?
func updateWithTrail(_ trail: Trail) {
self.trail = trail
navigationBarTitleLabel.text = trail.name
navigationBarSubtitleLabel.text = trail.location
centerImageView?.image = UIImage(named: trail.imageName)
}
override func viewDidLoad() {
super.viewDidLoad()
loadInitialData()
centerTileView.layer.transform = CATransform3DMakeTranslation(0.0, 0.0, 50.0)
leftTileView.layer.transform = CATransform3DMakeRotation(-0.4, 0.0, 1.0, 0.0)
rightTileView.layer.transform = CATransform3DMakeRotation(0.4, 0.0, 1.0, 0.0)
let superview = tableHeaderGreenLineView.superview!
tableHeaderGreenLineView.translatesAutoresizingMaskIntoConstraints = false
tableHeaderGreenLineView.leadingAnchor.constraint(equalTo: superview.
leadingAnchor, constant: 20).isActive = true
tableHeaderGreenLineView.trailingAnchor.constraint(equalTo: superview.
trailingAnchor, constant: -20).isActive = true
tableHeaderGreenLineView.bottomAnchor.constraint(equalTo: superview.bottomAnchor,
constant: 0).isActive = true
tableHeaderGreenLineView.heightAnchor.constraint(equalToConstant: 1.0).isActive =
true
// Add map view to bottom right area
let mapView = MKMapView()

Playground


Ready | Today at 1:51 PM
MyPlayground
//: Playground - noun: a place where people can play
import Foundation
var str = "Hello, playground"
"Hello, playground"
1+1
2
12 - 99
-87
5 * 24
120
133 / 9
14
10 % 3
1
var 값 = 1+1
2
print("\(값)")
"2\n"
2

Using Loops

Goal: Use a for loop to repeat a sequence of commands.

To break down coding tasks, you wrote functions for repeated patterns. Now you'll call one function multiple times using a loop. With a loop, you write your code once and enter the number of times to repeat it.

In this puzzle, there's a gem in the same position in each row. You will collect the gems by following the same pattern multiple times. This is the perfect place for a loop!

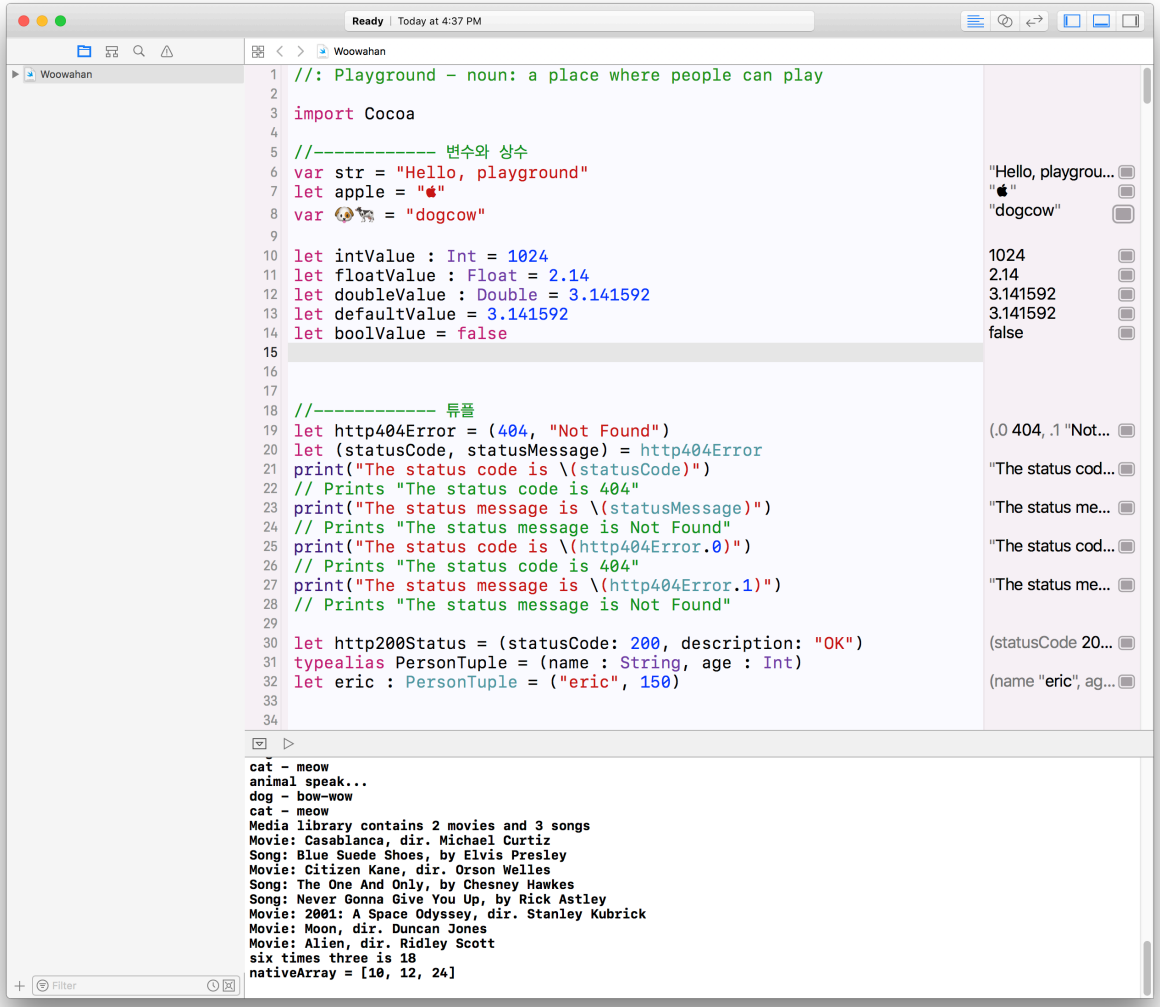
1 Enter the solution for one row inside the curly braces.

2 Decide how many times to repeat the loop.

3 Tap the number placeholder and specify the number of repetitions.

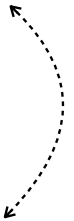
```
for i in 1 ... 5 {
    moveForward()
    moveForward()
    collectGem()
    moveForward()
}
```

http://bit.ly/2kqoQxa



정수 두 개를 더하는 함수 선언

```
func add(number1 : Int, number2 : Int) -> Int {
    let sum = number1 + number2
    return sum
}
```



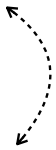
범위scope

예약어 (reserved keyword)

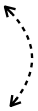
func, class, enum, struct, import, init, deinit, protocol, extension,
var, let, _, break, case, continue, default, defer,
if, in, repeat, return, switch, where, while
as, is, self, try, catch, throw, true, false, true, false,
optional, mutating, lazy, final, dynamic …

함수 호출(사용)하기

```swift
func add(number1 : Int, number2 : Int) -> Int {
    let sum = number1 + number2
    return sum
}

var result1 = add(number1: 123, number2: 234)
var result2 = add(number1: 10, number2: result1)
```



선언부분



호출부분

```swift
func add(_ number1 : Int, num2 number2 : Int) -> Int {
    let sum = number1 + number2
    return sum
}

var result3 = add(123, num2: 234)
```

이미 만들어놓은 함수들

* 자주 쓰는 표준 함수들

abs
assert
assertionFailure
debugPrint
dump
fatalError
getVaList
isKnownUniquelyReferenced
max
min
numericCast
precondition
preconditionFailure
print
readLine
repeatElement
sequence
stride
swap
transcode
unsafeBitCast
unsafeDowncast
withExtendedLifetime
withUnsafeMutablePointer
withUnsafePointer
withVaList
zip

https://developer.apple.com/reference/swift/1693602-swift_standard_library_functions

func print()

* 주어진 값을 콘솔 화면에 출력하는 함수

  print(_:separator:terminator:)

```
@inline(never) func print(_ items: Any..., separator: String = default,
   terminator: String = default)
```

```
print(12,99,3.14)
print(12,99,3.14, separator: ",", terminator: "---")
print(max(99,12,31.4),123)
```

```
"12 99 3.14\n"
"12,99,3.14---"
"99.0 123\n"
```

```
12 99 3.14
12,99,3.14---99.0 123
```

https://developer.apple.com/reference/swift/1541053-print

요약 다른 언어와 차이점

Static Typing + Duck-type System

cf. Dynamic Type

Type Inference

Bi-directional, based HM Type system

Safety

Type Safety, No Pointers, Optionals, Boundary Check

Functional Programming

Closures, Currying, Tailing closure, Monad …

데이터 타입 Data Type, 자료형

데이터 값value

컴퓨터가 처리하는(계산하는) 데이터 단위와 종류

* 정수: Int
* 실수 (부동 소수): Float / Double
* 문자열: String
* 논리: Bool

```
MemoryLayout<Int>.size // 8
MemoryLayout<Bool>.size // 1
MemoryLayout<String>.size // 24
MemoryLayout<Float>.size // 4
MemoryLayout<Double>.size // 8
```

변수와 상수

* 변수variable (mutable)
  * 값을 보관하면서 계속해서 바꿀 수 있는 공간
  * 표현식

```
var <변수 이름> = <값>
var num = 20
```

* 상수constant (Immutable)
  * 처음 값만 넣고 다시 바꿀 수 없는 공간
  * 표현식

```
let <상수 이름> = <값>
let quarter = 4
```

변수(상수) 이름

길이 제한 없음

첫 문자는 문자(유니코드, 알파벳), $, _ 만 가능.

```
var 🐶🐄 = "dogcow"
```

숫자로 시작하면 안됨

두 번째 문자부터 숫자, 문자, $, _ 모두 가능

읽어서 명확한 이름을 지어주자. 처음 단어는 소문자로, 다음 단어 시작만 대문자로

basicSwift, twoNumber2

부동 소숫점

Float

- 부호(1자리) + 지수(8자리) + 가수(23자리) = 32비트
- single-precision 32-bit IEEE 754 floating point

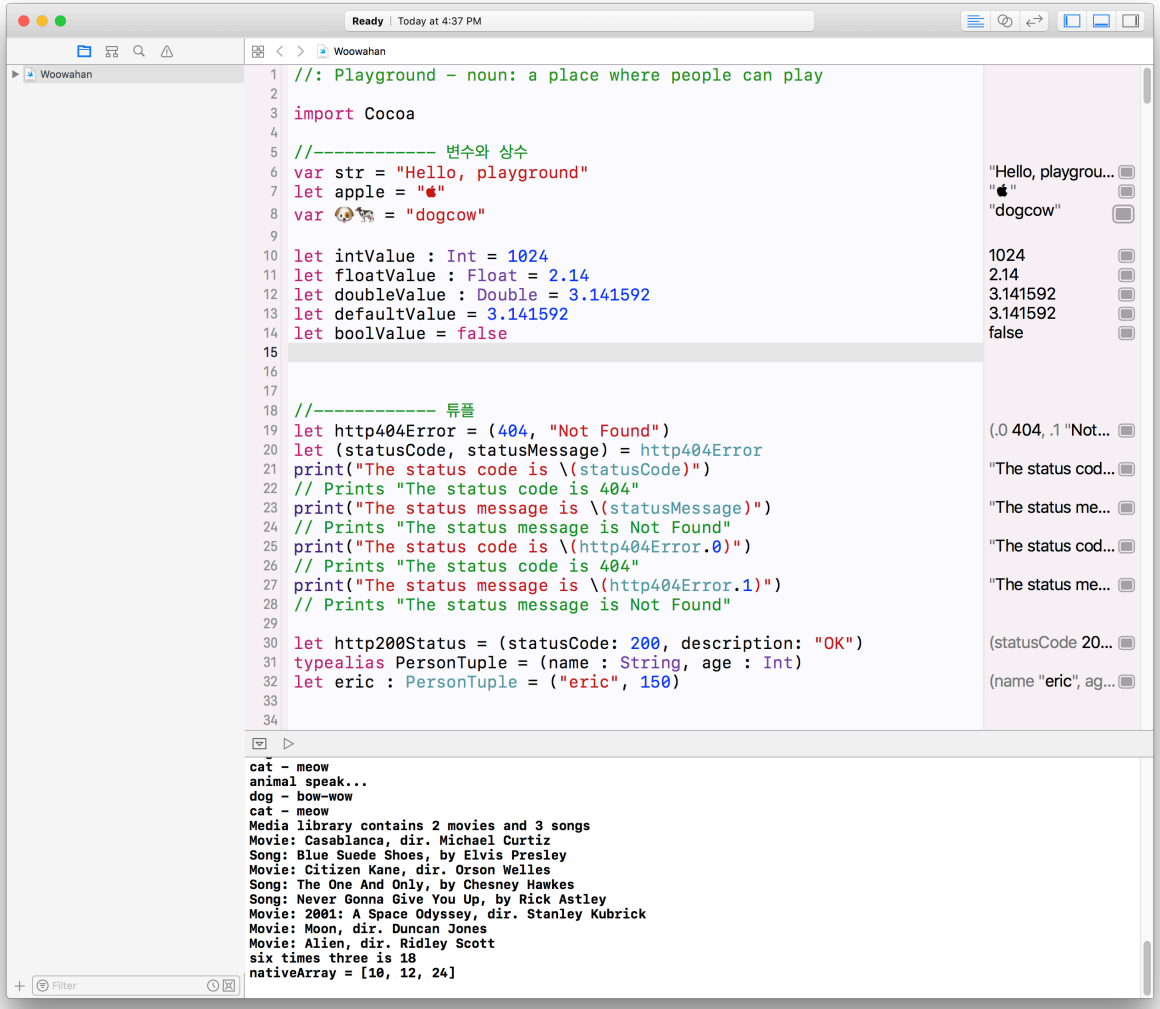
Double

- 부호(1자리) + 지수(11자리) + 가수(52자리) = 64비트
- double-precision 64-bit IEEE 754 floating point

```
MemoryLayout<Float>.size // 4
MemoryLayout<Double>.size // 8
```

타입	최소	최대
Float	$1.401298464324817E-45f\ (2^{-149})$	$3.4028234663852886E38f\ ((2-2^{-23}) \cdot 2^{127})$
Double	$4.9E-324\ (2^{-1074})$	$1.7976931348623157E308\ ((2-2^{-52}) \cdot 2^{1023})$

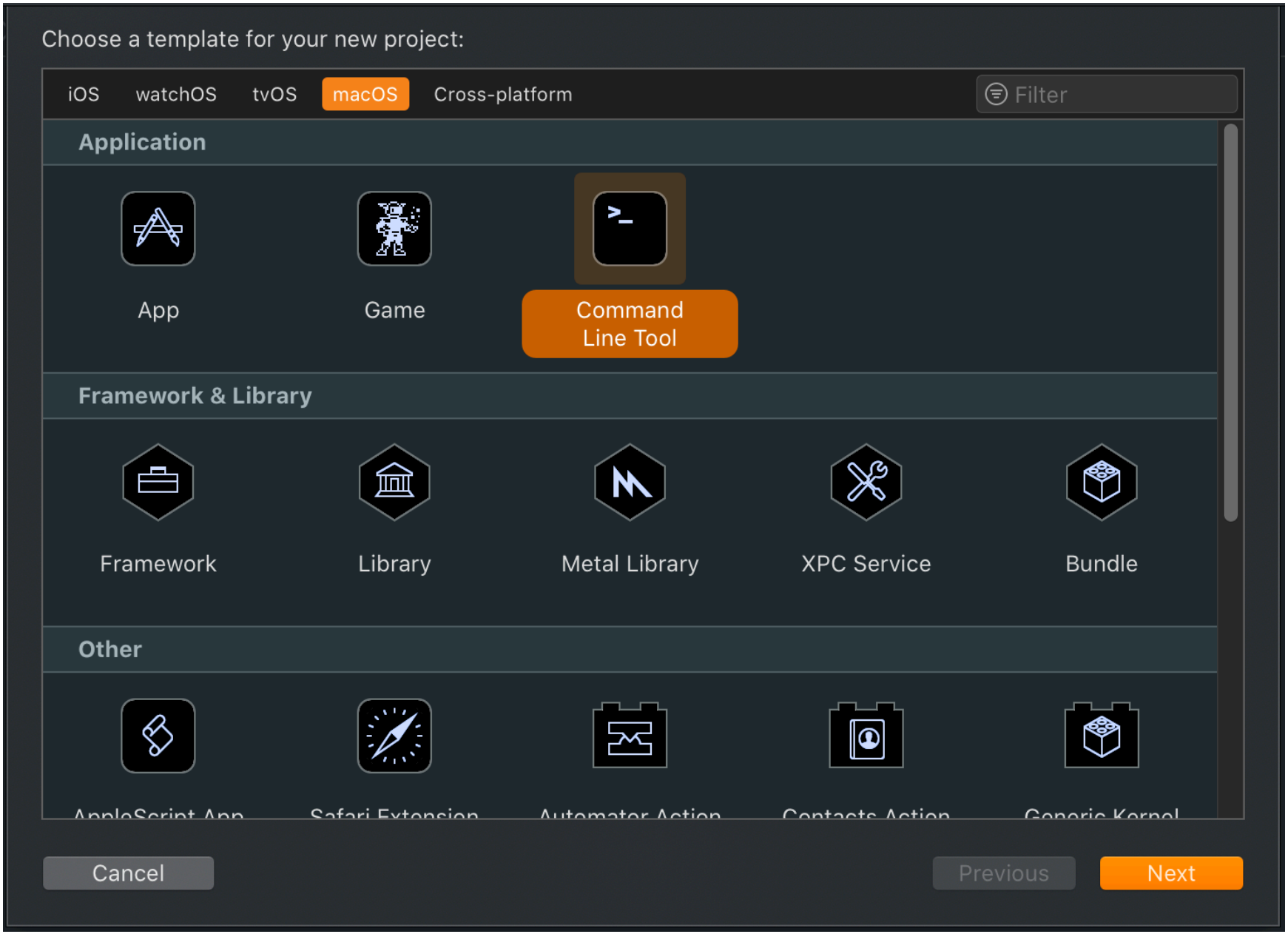
https://goo.gl/L1UwRD



워밍업 코딩미션#1

Swift Programming

코드스쿼드


Choose a template for your new project:
iOS
watchOS
tvOS
macOS
Cross-platform
Filter
Application
App
Game
Command Line Tool
Framework & Library
Framework
Library
Metal Library
XPC Service
Bundle
Other
Cancel
Previous
Next

같이 코딩해봐요

1) 구구단 출력 함수

```
func printGugu(dan: Int) {
    //
}
```

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
```

직접 코딩해봐요

2) 구구단 전체를 출력 함수

```
func printAllGugu() {
    //
}
```

직접 코딩해봐요

점A(x,y)와 점B(x,y) 좌표를 입력받아서
두 점 사이의 거리를 실수로 계산하는 함수

```
func distance(ax : Int, ay : Int,
              bx : Int, by : Int) -> Float {

}
```

위 함수를 만들고 A(1,1)와 B(5,6) 점 사이의 거리를 구하세요

복합 데이터 구조

배열과 사전, 집합과 튜플, 열거형

여러 값을 한꺼번에 tuples

* 여러 값을 한꺼번에 묶어서 사용하는 타입

```
let http404Error = (404, "Not Found")
// http404Error is of type (Int, String), and equals (404, "Not Found")


let (statusCode, statusMessage) = http404Error
print("The status code is \(statusCode)")
// Prints "The status code is 404"
print("The status message is \(statusMessage)")
// Prints "The status message is Not Found"

print("The status code is \(http404Error.0)")
// Prints "The status code is 404"
print("The status message is \(http404Error.1)")
// Prints "The status message is Not Found"

let http200Status = (statusCode: 200, description: "OK")

typealias PersonTuple = (name : String, age : Int)
let eric : PersonTuple = (“eric", 150)
```

연관된 항목을 중에 하나 enum

* 정의한 항목 값 중에 선택하는 타입

제한된 선택지를 주고 싶을 때
정해진 값 외에는 입력받고 싶지 않을 때
예상된 입력 값이 한정되어 있을 때

<기본 방식>

```
enum CompassPoint {
  case north
  case south
  case east
  case west
}
```

<원시값 raw-value>

```
enum ASCIIControlCharacter: Character {
    case tab = "\t"
    case lineFeed = "\n"
    case carriageReturn = "\r"
}
```

<연관값 associated value>

```
var productBarcode = Barcode.upc(8, 85909, 51226, 3)
productBarcode = .qrCode("ABCDEFGHIJKLMNOP")

switch productBarcode {
case .upc(let numberSystem, let manufacturer, let product, let check):
    print("UPC: \(numberSystem), \(manufacturer), \(product), \(check).")
case .qrCode(let productCode):
    print("QR code: \(productCode).")
}
// 결과값 "QR code: ABCDEFGHIJKLMNOP."
```

한꺼번에 많은 값을 담아보자

* 배열array

  동일한 데이터 타입을 연속해서 담아놓고 순서대로 접근하는 콜렉션

```
var ageArray = [10, 20, 30, 40, 50]
print(ageArray[0])
```

* 사전dictionary

  동일한 데이터 타입을 키값과 함께 담아놓고 키값으로 접근하는 콜렉션

```
var gradeDic = ["a" : 90, "b" : 80, "c" : 70, "d" : 60]
print(gradeDic["a"] ?? 0)
```

* 집합set

  동일한 데이터 타입을 순서없이 집합에 담아놓고 포함되어 있는지 확인하는 콜렉션

```
var aSet: Set = [11, 12, 13]
aSet.contains(12)
```

워밍업 코딩미션#2

타입 변환 + 복합 타입

코드스쿼드

직접 코딩해봐요

점A(x,y)와 점B(x,y) 좌표를 튜플로 입력받아서
두 점 사이의 거리를 실수로 계산하는 함수

```
func distance(a : Point, b : Point) -> Double {

}
```

위 함수를 만들고 A(1,1)와 B(5,6) 점 사이의 거리를 구하세요

```
distance(a: (1,1), b: (5,6))
```

연산자 operator

+, -, *, /, &&, ||, ??

연산자 Operator

기본 자료형을 계산하기 위해 사용

특별히 String 클래스는 + 연산자가 이어붙이는 기능.

boolean 제외

대입 연산자 = (overloading안됨)

```
intValue1 = 5
intValue2 = 10
```

덧셈(+), 뺄셈(-) 연산자

```
result = intValue1 + intValue2
```

곱하기(*), 나누기(/) 연산자

```
result = intValue1 * intValue2
```

연산자 Operator

나머지(%) 연산자

```
result = intValue1 % intValue2
```

배수 확인할 때 사용. (intValue1 % 3) 결과가 0인지 확인

계산하는 대입 연산자

```
intValue1 += 5;
intValue2 /= 2;
```

```
intValue2 -= 2
intValue2 %= 2
```

단항 연산자

+ 와 -

```
var intValue = -10
result = +intValue
minus = -intValue
```

+ 는 "변수 * (1)"을 의미하고

- 는 "변수 * (-1)"을 의미한다.

단항 연산자 Complement

!

Bool 타입에 대해 결과값이 반대로 됨

```
var flag = true
print(flag)
print(!flag)
```

연산자 계산 순서

대부분의 계산은 왼쪽에서 오른쪽으로 진행됨

1+2*3

(1+2)*3

구분	연산자	우선 순위
단항 연산자	+, -, !, ~	1
산술 연산자	*, /, %	2
	+, -	3

비교 연산자

== : 같음

!= : 같지 않음

> : (왼쪽 값이) 큼

>= : (왼쪽 값이) 같거나 큼

< : (왼쪽 값이) 작음

<= : (왼쪽 값이) 같거나 작음

논리 연산자

&& : AND 결합 Conditional AND

|| : OR 결합 Conditional OR

값		결과	
x	y	x && y	x || y
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

삼항 연산자

Conditional operator

```
doBlindDateFlag = (point>=80) ? true : false
```

변수 = (boolean조건식) ? 참일때값 : 거짓일때값

Nil-coalescing operator

```
z = a ?? b
```

```
z = (a != nil) ? a! : b
```

범위 연산자

닫힌 범위 연산자

`A...B` A부터 시작해서 B(포함)까지 범위

`1...5` 1,2,3,4,5

반닫힌 범위 연산자

`A..<B` A부터 시작해서 B(제외)까지 범위

`1..<5` 1,2,3,4

고급 연산자

비트 연산자

`~(not) , &(and) , |(or) , ^(xor)`

쉬프트

`<< (left shift) , >> (right shift)`

오버플로우 연산자

`&+ , &- , &*`

흐름 제어하기

if, switch, for, while

조건을 따져요 if

```
var temperature = 10
if temperature < 15 {
    print("아우 추워. 겉옷을 챙기세요")
}
// Prints "아우 추워. 겉옷을 챙기세요"

temperature = 40
if temperature > 30 {
    print("더운 날씨에요. 티셔츠만 입어도 될꺼에요")
}
else if temperature > 20 {
    print(“날씨가 활동하기 좋겠어요")
}
else {
    print("쌀쌀할 수 있어요")
}
// Prints "더운 날씨에요. 티셔츠만 입어도 될꺼에요"
```

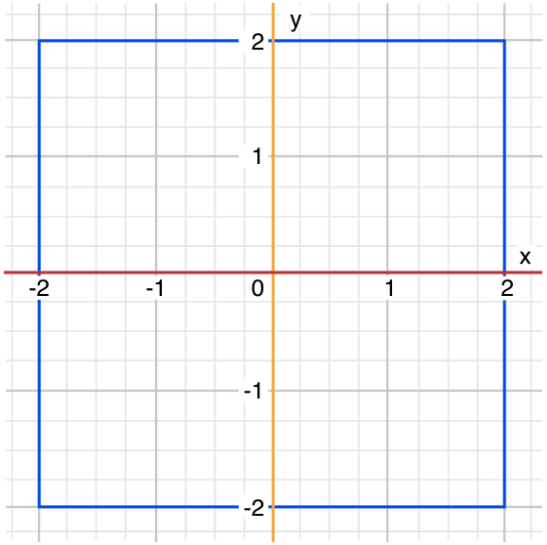
복잡한 조건 비교는 switch-case

```
let someCharacter: Character = "z"
switch someCharacter {
case "a":
    print("The first letter of the alphabet")
case "z":
    print("The last letter of the alphabet")
default:
    print("Some other character")
}
// Prints "The last letter of the alphabet"
```

```
let approximateCount = 62
let countedThings = "moons orbiting Saturn"
var naturalCount: String
switch approximateCount {
case 0:
    naturalCount = "no"
case 1..<5:
    naturalCount = "a few"
case 5..<12:
    naturalCount = "several"
case 12..<100:
    naturalCount = "dozens of"
case 100..<1000:
    naturalCount = "hundreds of"
default:
    naturalCount = "many"
}
print("There are \(naturalCount) \(countedThings).")
// Prints "There are dozens of moons orbiting Saturn."
```

복잡한 조건 비교는 switch-case

```swift
let somePoint = (1, 1)
switch somePoint {
case (0, 0):
    print("(0, 0) is at the origin")
case (_, 0):
    print("(\(somePoint.0), 0) is on the x-axis")
case (0, _):
    print("(0, \(somePoint.1)) is on the y-axis")
case (-2...2, -2...2):
    print("(\(somePoint.0), \(somePoint.1)) is inside the box")
default:
    print("(\(somePoint.0), \(somePoint.1)) is outside of the box")
}
// Prints "(1, 1) is inside the box"
```



반복 작업 for

```
for index in 1...5 {
    print("\(index) 곱하기 5 는 \(index * 5)")
}

for index in stride(from: 5, through: 1, by: -1) {
    print("\(index) 곱하기 5 는 \(index * 5)")
}
```

```
let base = 3
let power = 10
var answer = 1
for _ in 1...power {
    answer *= base
}
print("\(base) to the power of \(power) is \(answer)")
// Prints "3 to the power of 10 is 59049"
```

```
let names = ["Honux", "JK", "Crong", "Anonymous"]
for name in names {
    print("Hello, master \(name)!")
}
//Hello, master Honux!
//Hello, master JK!
//Hello, master Crong!
//Hello, master Anonymous!
```

조건에 맞는 동안 반복하기 while

```
let finalSquare = 25
var board = [Int](repeating: 0, count: finalSquare + 1)

board[03] = +08; board[06] = +11; board[09] = +09; board[10] = +02
board[14] = -10; board[19] = -11; board[22] = -02; board[24] = -08

var square = 0
var diceRoll = 0
while square < finalSquare {
    // roll the dice
    diceRoll += 1
    if diceRoll == 7 { diceRoll = 1 }
    // move by the rolled amount
    square += diceRoll
    if square < board.count {
        // if we're still on the board, move up or down for a snake or a ladder
        square += board[square]
    }
}
print("Game over!")
```

코딩미션#3

사다리 게임

직접 코딩해봐요

1) 가로/세로 칸을 입력값으로 넘기면 가로*세로 크기의 사다리를 만드는 makeLadder()함수를 작성하세요.

사다리는 Bool 타입 2차원 배열로 표현하세요.

사다리가 있으면 true , 없으면 false

2) 사다리 배열 데이터를 기준으로 사다리를 콘솔 화면에 출력해 주는 printLadder() 함수를 작성하세요.

사다리 값이 true면 가로로 |-| 를 출력하고

사다리 값이 false면 가로는 생략하고 | | 만 출력하세요.

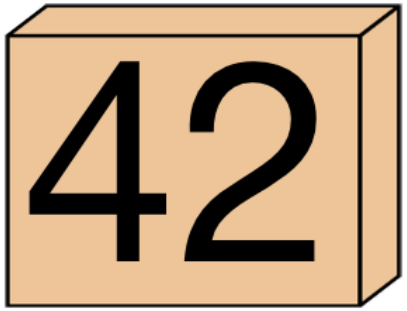
전체 사다리 모양이 잘 보이도록 그려보세요.

옵셔널 optional

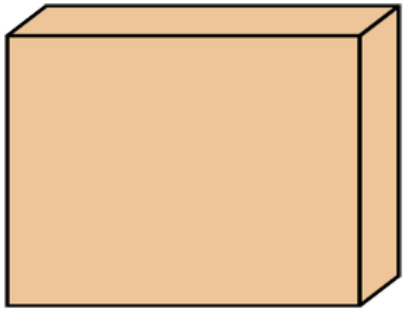
값이 있느냐 없느냐 그것이 문제로다!

42

Int



Int?



Int?

Optional

```
let possibleNumber = "123"
let convertedNumber = Int(possibleNumber)
```

Forced unwrapping


```
var serverResponseCode: Int? = 404
serverResponseCode = nil

if convertedNumber != nil {
    print("convertedNumber has an integer value of \(convertedNumber!).")
}
```

Optional Binding

```
if let actualNumber = Int(possibleNumber) {
    print("\"\(possibleNumber)\" has an integer value of \(actualNumber)")
} else {
    print("\"\(possibleNumber)\" could not be converted to an integer")
}
```

Implicitly Unwrapped Optionals

```
let possibleString: String? = "An optional string."
let forcedString: String = possibleString! // requires an exclamation mark

let assumedString: String! = "An implicitly unwrapped optional string."
let implicitString: String = assumedString // no need for an exclamation mark
```

Error Handling

```
func makeASandwich() throws {
            // ...
}

do {
    try makeASandwich()
        eatASandwich()
} catch SandwichError.outOfCleanDishes {
    washDishes()
} catch SandwichError.missingIngredients(let ingredients) {
    buyGroceries(ingredients)
}


func buyASandwich() -> Sandwich? throws {
    var mySandwich : Sandwich? = nil

    if (…) throw SandwichError.outOfCleanDishes
    return mySandwich
}

let resultOptional = try? buyASandwich()
let resultValue    = try! buyASandwich()
```

do { try } catch가 필요함

클로저 closure

모든 함수는 클로저. 클로저는 함수이거나 아닐수도 있다

함수 중심 프로그래밍

Functional Programming

함수가 1등 시민 first-class citizen

- 함수를 **타입**으로 지정하거나,
- **인자값**으로 넘기거나,
- **리턴값**으로 받을 수 있다

클로저 형태

함수

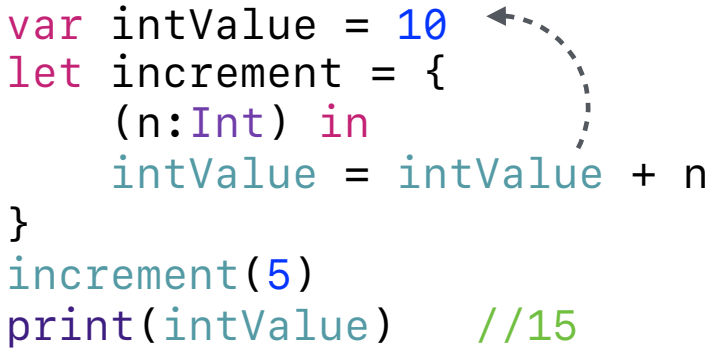
```
func squared(n : Int) -> Int { return n * n }
```

클로저

```
let closure1 : (Int) -> Int = { n in return n * n }
let closure2 = { (n:Int) -> Int in return n * n }
```

* 클로저 closure는 람다계산식(lamda Calculus) 구현체
* 이름 없는 함수(anonymous function)로 작성가능
* 선언된 범위(scope)의 변수를 캡처해서 저장하고 닫힘
* 스위프트 클로저는 캡처한 변수를 참조(reference)

클로저 선언과 값 참조

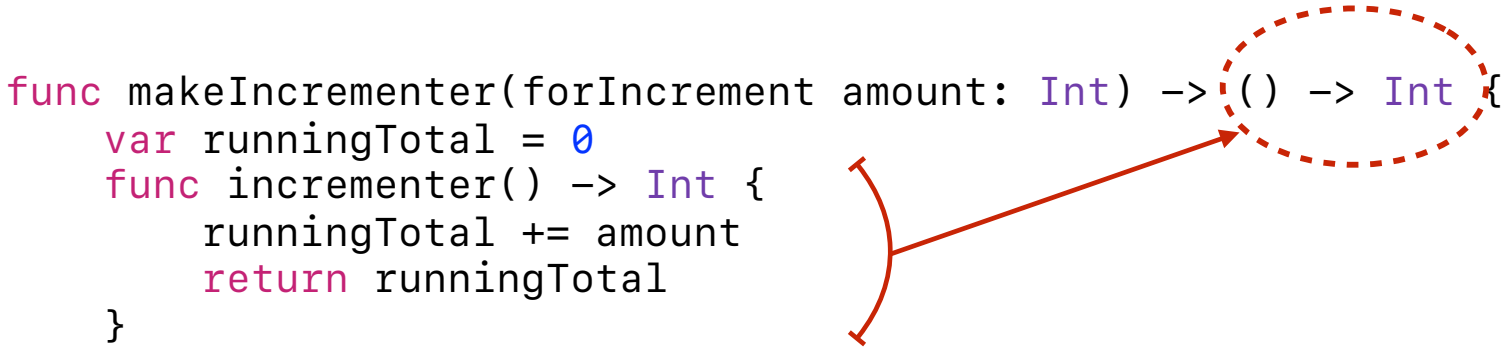


```
var intValue = 10
let increment = {
    (n:Int) in
    intValue = intValue + n
}
increment(5)
print(intValue)   //15


let c10 = increment
intValue = 100
c10(10)
print(intValue)   //110
```

클로저 선언과 캡처목록

```
var intValue = 10
let increment = {
    [intValue] (n:Int) in
     print(intValue + n)
}
intValue = 100
increment(5) //15
print(intValue)   //여전히 100
```

클로저를 리턴하는 함수



```
func makeIncrementer(forIncrement amount: Int) -> () -> Int {
    var runningTotal = 0
    func incrementer() -> Int {
        runningTotal += amount
        return runningTotal
    }
    return incrementer
}

let incrementByTen = makeIncrementer(forIncrement: 10)

print(incrementByTen()) //10
print(incrementByTen()) //20

let incrementBy7 = makeIncrementer(forIncrement: 7)
print(incrementBy7()) //7
print(incrementByTen()) //30
```

고차 함수 high-order function

* 함수(또는 클로저)를 인자값 또는 리턴값으로 사용하는 함수

```
let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]
let reversedNames = names.sorted(by: {
    (s1: String, s2: String) -> Bool in
        return s1 > s2
    })
```

```
let reversedNames = names.sorted { s1, s2 in s1 > s2 }
//["Ewa", "Daniella", "Chris", "Barry", "Alex"]
```

```
let numbers = [1, 2, 3]
let strings = numbers.map( { element in String(element) } )
print(strings) //["1", "2", "3"]
```

```
[ x1, x2, ... , xn].map(f) -> [f(x1), f(x2), ... , f(xn)]
```

클로저 다양한 표현식

```
let numberArray = [2, 8, 1, 3, 5]
let resultArray = numberArray.map(squared) // 결과 값은 [4, 64, 1, 9, 25]

let result1 = numberArray.map({ (n : Int) -> Int in return n*n })
let result2 = numberArray.map({ (n : Int) -> Int in n*n })
let result3 = numberArray.map({ n in return n*n })
let result4 = numberArray.map({ n in n*n })
let result5 = numberArray.map({ $0 * $0 })
let result6 = numberArray.map() { $0 * $0 }
let result7 = numberArray.map { $0 * $0 }
```

- 클로저는 함수로 선언하지 않아도 되고, 인자값 대신에 그 자리에 바로 선언 할 수 있다.
- 한 줄 표현 클로저에서는 **return** 구문을 생략해도 된다.
- 클로저에서 사용하는 변수 타입은 생략할 수 있고, 배열에 있는 변수 타입으로 추정이 가능하다. **in** 지시어를 사용해서 변수를 명시할 수 있다.
- 위의 경우와 마찬가지로 **return** 구문은 생략이 가능하다.
- 클로저 내부에서는 축약 변수를 인자 값 순서에 따라서 **$0**부터 사용할 수 있다.
- 함수 인자 중에서 마지막 인자 값이 클로저인 경우는 후행(trailing) 클로저로 판단하고 함수 괄호 다음에 빼서 선언할 수 있다.
- 함수에 대한 인자값이 없는 경우 괄호는 생략할 수 있다.

고차 함수와 옵셔널

```swift
let numbers = [1, 2, 3, 4, 5, 10, 11, 12]

let filtered = numbers.filter({ $0 > 10 })
print(filtered) // [11, 12]
```

```swift
let reduced = numbers.reduce(0, { $0 + $1 })
print(reduced) // 48
```

```swift
let nilNumbers : [Int?] = [1, 2, nil, 4, 5]
let mapNumber = nilNumbers.map({ $0 })
print(mapNumber)
//[Optional(1), Optional(2), nil, Optional(4), Optional(5)]

let mapNumberOnly = nilNumbers.flatMap({ $0 })
print(mapNumberOnly) //[1, 2, 4, 5]
```

코딩미션#4

응용 문제해결

코드스쿼드

직접 코딩해봐요

정수 숫자 배열 Array를 입력으로 받아서
배열중에 있는 2 또는 3의 배수를 골라서
골라진 숫자들에 각각 5를 곱하고
모든 숫자들의 합을 구하는 함수

```
func complex(from array: [Int]) -> Int {
}
```

플러스 미션

```
func fizzbuzz(lines : Int)->Array<String>
```

3의 배수는 “fizz”
5의 배수는 “buzz”
15의 배수는 “fizzbuzz”
나머지는 숫자값

테스트 코드

```
print(fizzbuzz(lines:15))
```

참고

```
var result : Array<String> = []
```

입력 n=15
출력

```
[
    "1",
    "2",
    "Fizz",
    "4",
    "Buzz",
    "Fizz",
    "7",
    "8",
    "Fizz",
    "Buzz",
    "11",
    "Fizz",
    "13",
    "14",
    "FizzBuzz"
]
```

https://leetcode.com/problems/fizz-buzz/

구조체 struct

데이터 추상화하기

여러 데이터를 한꺼번에

값을 저장하기 위한 **프로퍼티**property 선언

기능을 제공하기 위한 **메서드**method 선언

서브스크립트로 접근할 수 있는 문법 지원

초기 상태를 위한 초기화 메서드 제공

```
struct SomeStructure {
    // 정의하는 부분
}

struct Resolution {
    var width = 0
    var height = 0
}

let someResolution = Resolution()
someResolution.width

let vga = Resolution(width: 640, height: 480)
```


서울우유
Blueberry
Milk
서울우유
Banana
Milk
서울우유
Energy Choco
Milk
일반화 + 추상화
Milk
스위프트 코드
struct Milk

우유 객체 구조체

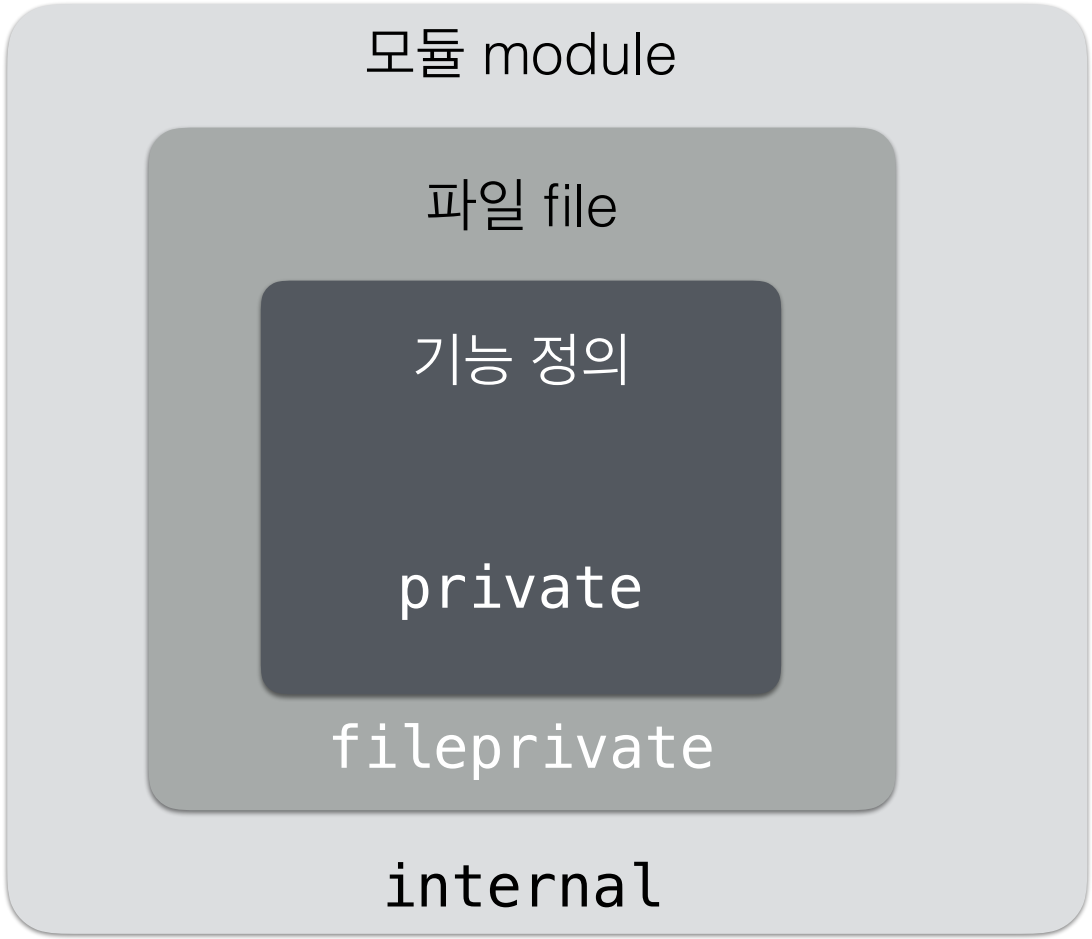
```
enum MilkType {
    case Blueberry
    case Banana
    case EnergyChoco
}

struct Milk {
    let brand : String
    let amount : Int
    let title : String
    let type : MilkType
}
```

우유 객체 인스턴스

```
let berryMilk1 = Milk(brand: "서울우유", amount: 150,
       title: "블루베리우유", type: .Blueberry)
let bananaMilk1 = Milk(brand: "서울우유", amount: 150,
       title: "바나나우유", type: .Banana)
let chocoMilk1 = Milk(brand: "서울우유", amount: 150,
       title: "에너지초코우유", type: .EnergyChoco)
```

접근제어

open, public, internal, fileprivate, private


모듈 module
파일 file
기능 정의
private
fileprivate
internal
public, open

범위와 상속

	범위	상속
open* (클래스)	모듈 외부	O
public	모듈 외부	O
internal	모듈 내부	△ (범위내)
~~**fileprivate**~~	~~파일 내부~~	~~△ (범위내)~~ swift4
Private	기능 내부	X

객체 중심 프로그래밍

struct 대신 class

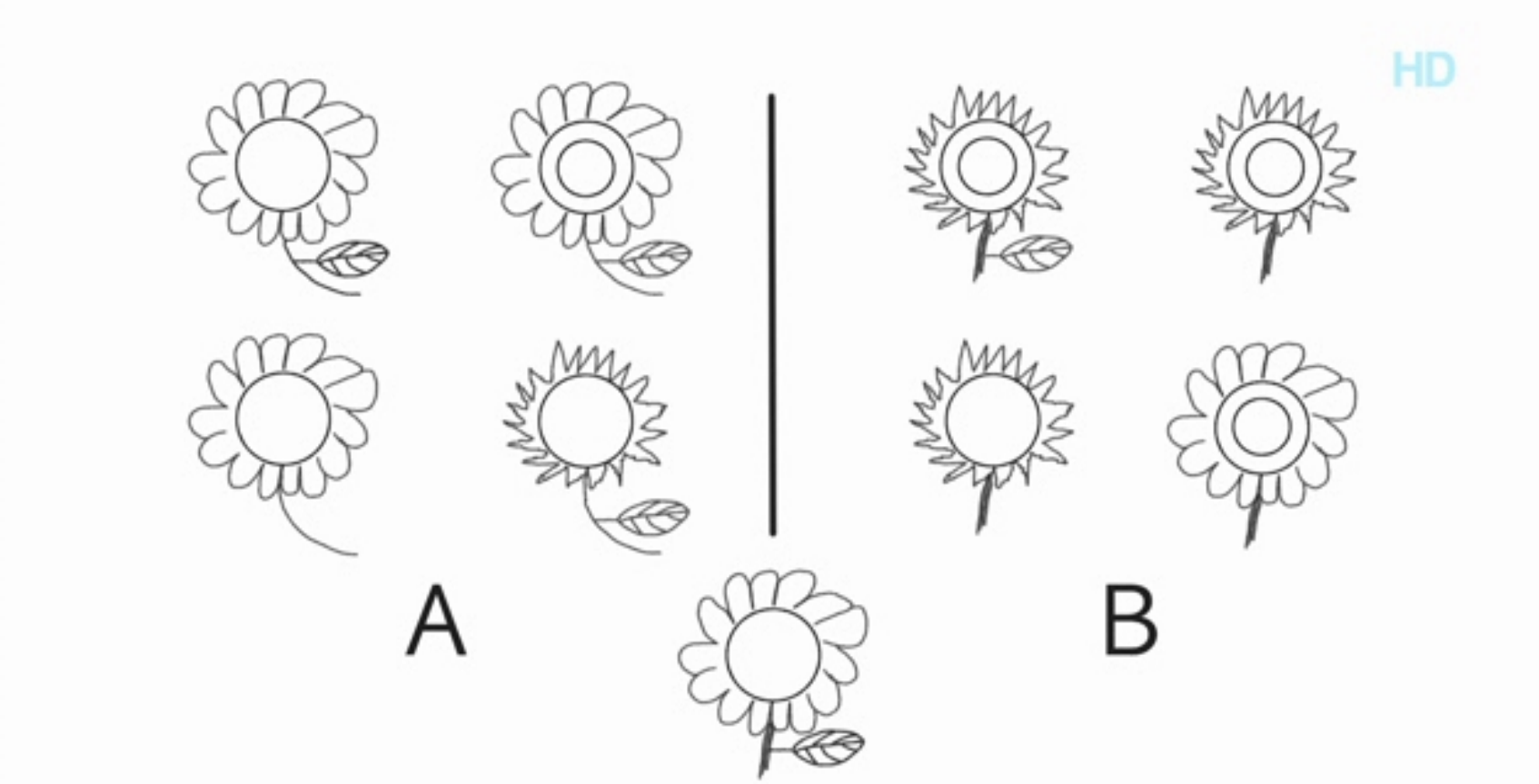
구조체 vs 클래스

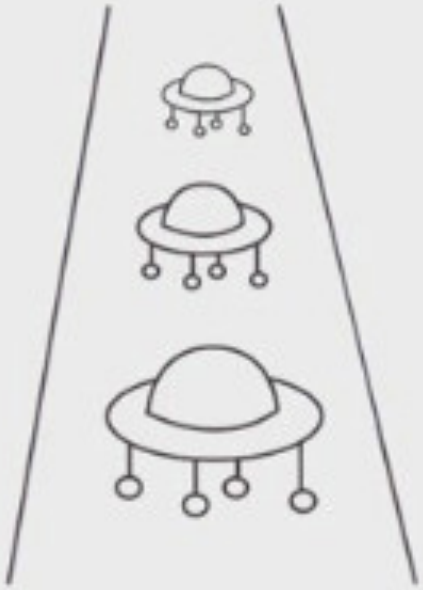
struct	class
프로퍼티에 값을 저장할 수 있다 함수로 원하는 기능을 제공할 수 있다 서브스크립션으로 값에 접근할 수 있다 초기(상태)값을 위해 생성함수를 정의할 수 있다	
상속 불가	상속 가능 상위/하위 클래스 타입으로 형변환
-	소멸함수에서 불필요한 리소스를 해제
의미있는 **값** **Value** semantic (Direct)	의미있는 **레퍼런스** **Reference** semantic (Indirect)
-	인스턴스별 참조 개수 관리 필요

과연 객체란 무엇인가?

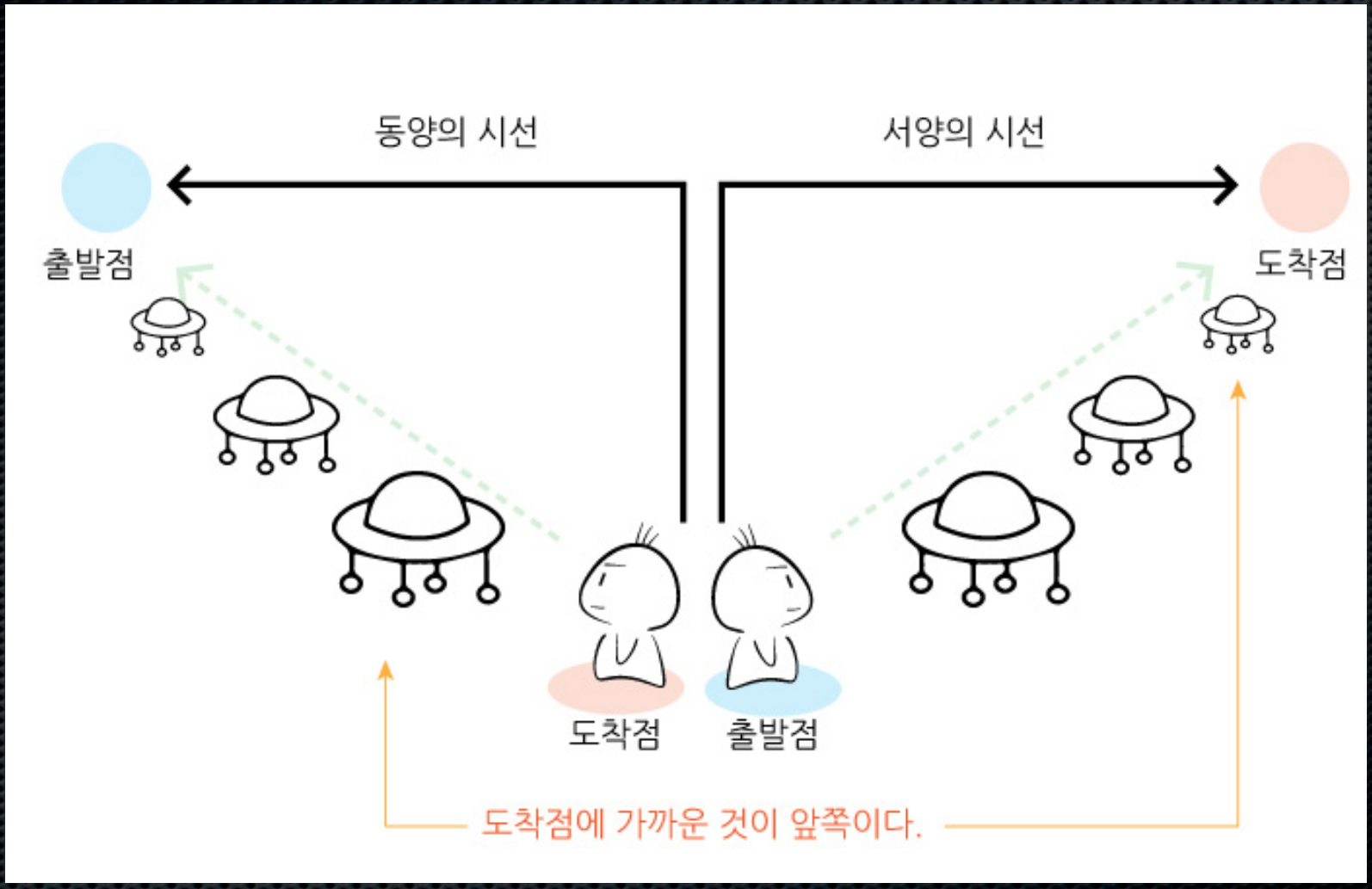
What is the meaning of OBJECT?

EBS 타큐멘터리 동과 서


A
B

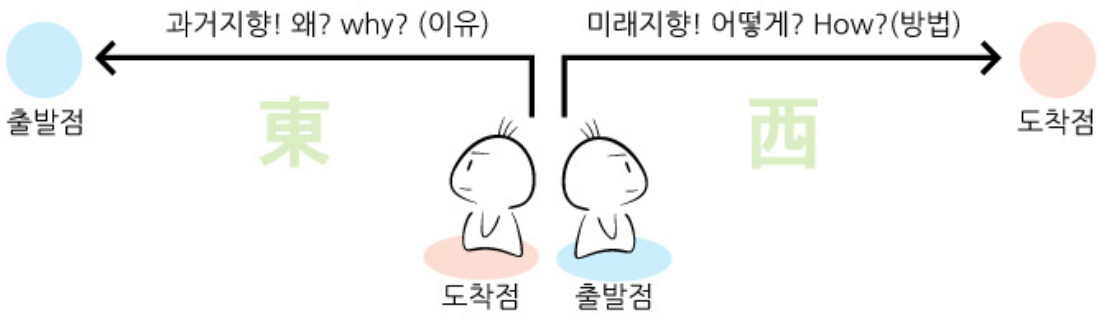


어떤 것이 앞쪽에 있을까?



EBS 타큐멘터리 동과 서

언어와 생각하는 방식

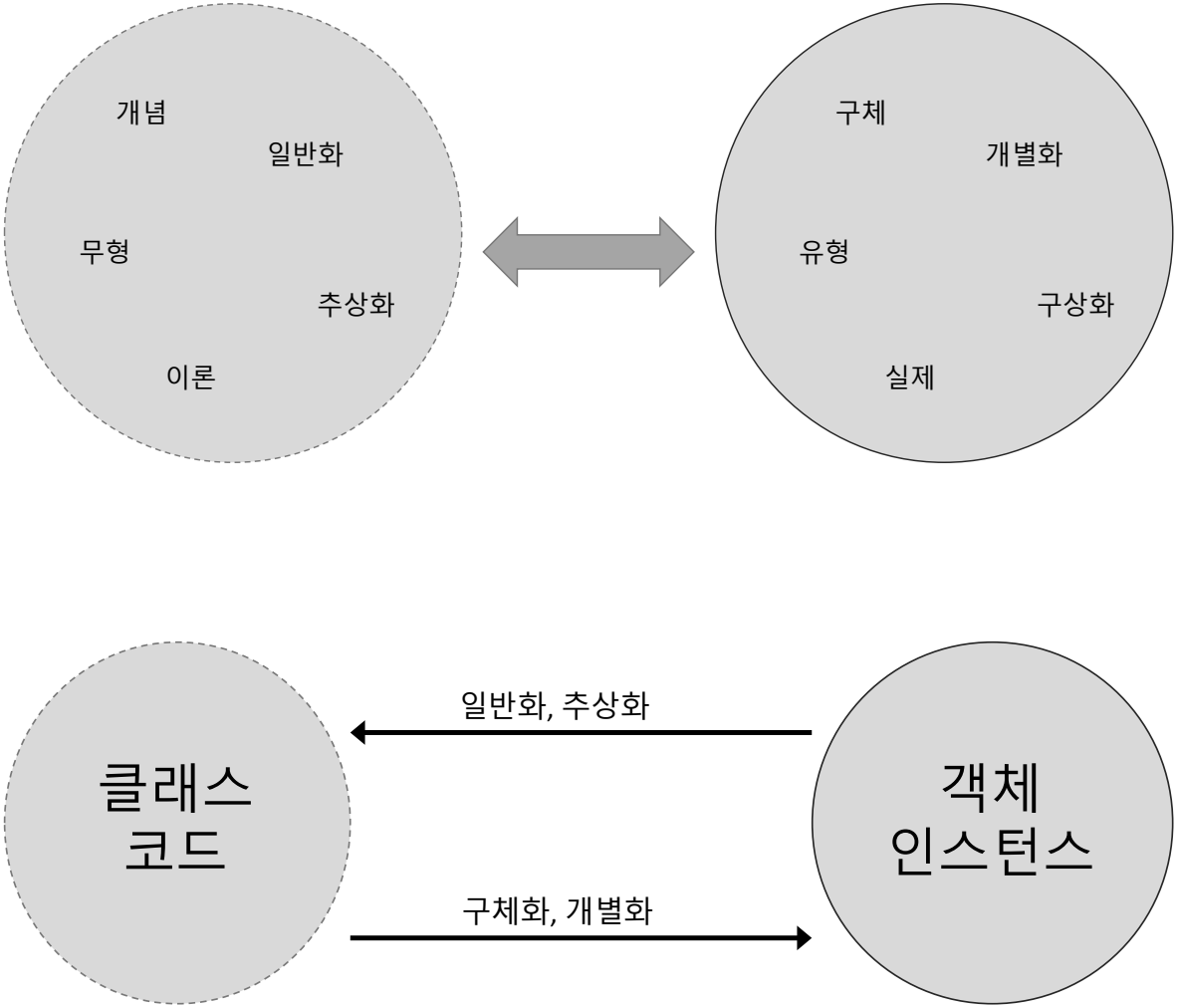


* **명사로 세상을 보는 사람**

  (Would you like to have) more tea?

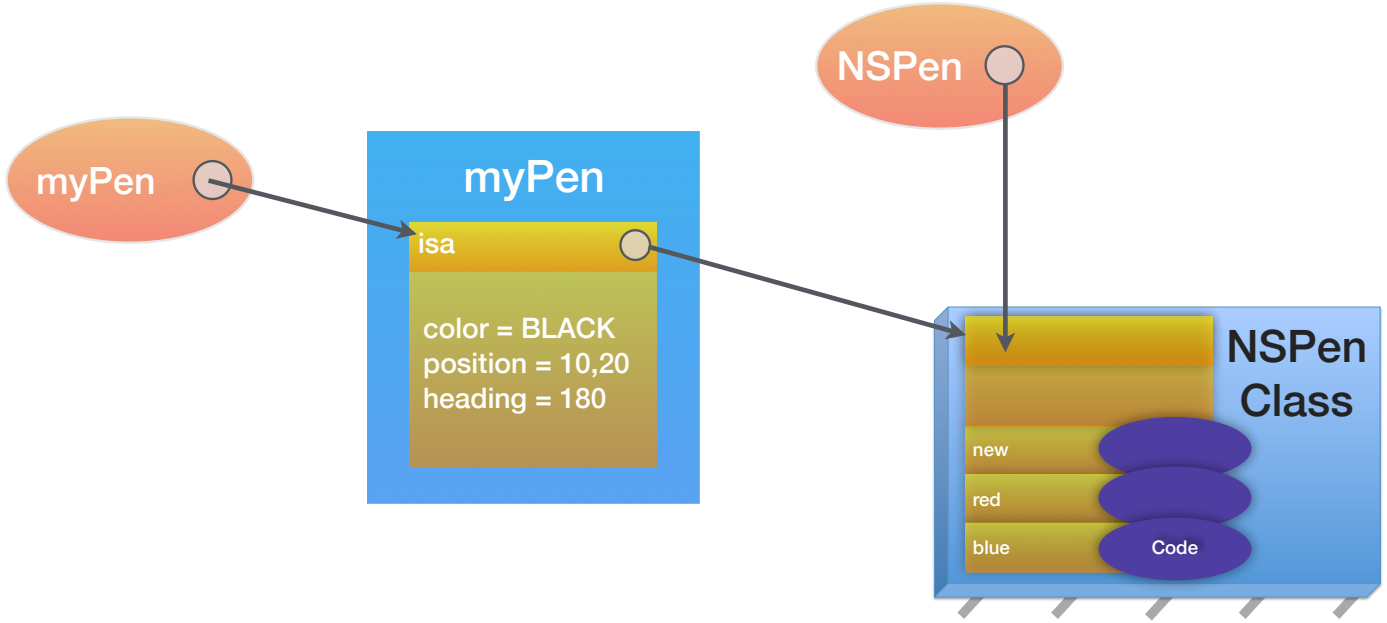
* **동사로 세상을 보는 사람**

  (차) 더 마실래?


개념
일반화
무형
추상화
이론
구체
개별화
유형
구상화
실제
클래스
코드
일반화, 추상화
구체화, 개별화
객체
인스턴스

클래스와 객체 인스턴스

```
var myPen = NSPen()
```



> 우유 객체 클래스

```
class Milk {
    var brand : String
    var amount : Int
    var title : String
    var type : MilkType

    init() {
        brand = ""
        amount = 0
        title = ""
        type = .unknown
    }
}
```

```
class ChocoMilk : Milk {
    override init() {
        super.init()
        type = .energyChoco
    }
}

class BananaMilk : Milk {
    override init() {
        super.init()
        type = .banana
    }
}
```

> 우유 객체 인스턴스

```
let bananaMilk1 = BananaMilk()
let chocoMilk1 = ChocoMilk()
```

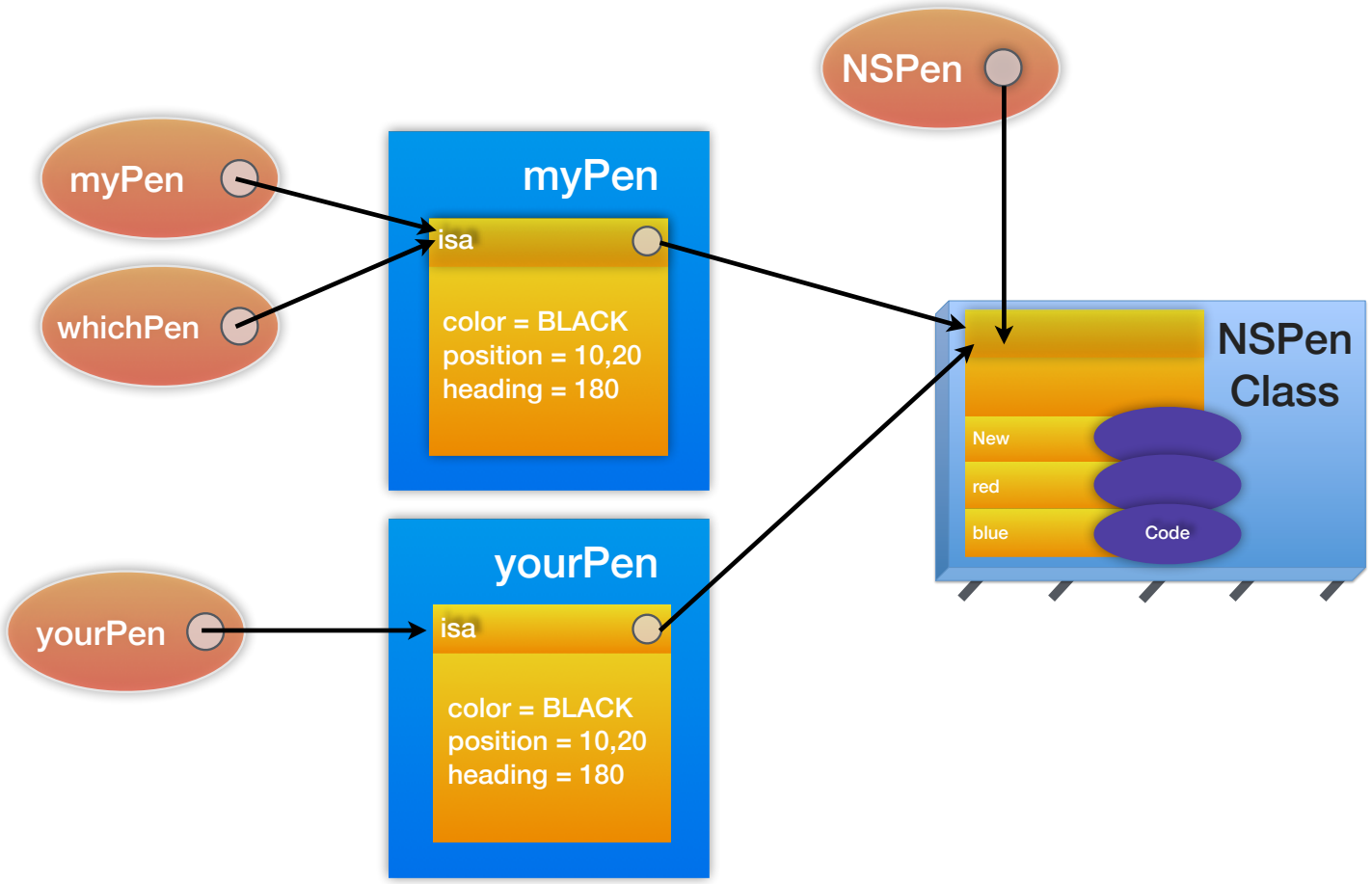

만드는 앱
코코아 프레임워크 (SDK)
스위프트 표준 라이브러리

객체 중심 프로그래밍

- 프로퍼티 property와 메소드 method
- 캡슐화 encapsulation
- 상속 inheritance
- 다형성 polymorphism
- 클래스 객체 class와 객체 인스턴스 instance
- 객체 디자인 패턴 design pattern

Identity vs. Equality

```
var myPen = NSPen()
var yourPen = NSPen()
var whichPen = myPen
```



객체(인스턴스)를 비교한다는 것은?

* 같은 클래스인가?
* identity 가 같은가?

  === 비교 연산자
* equality 객체인가?

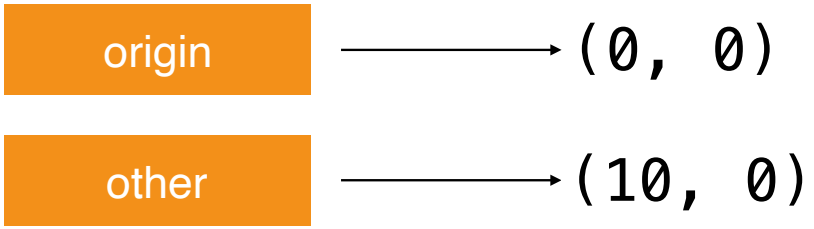
  == 비교 연산자 또는 Equatable 프로토콜

Value vs. Reference

Value Types

```
let origin = CGPoint(x: 0, y: 0)
var other = origin
other.x += 10
```

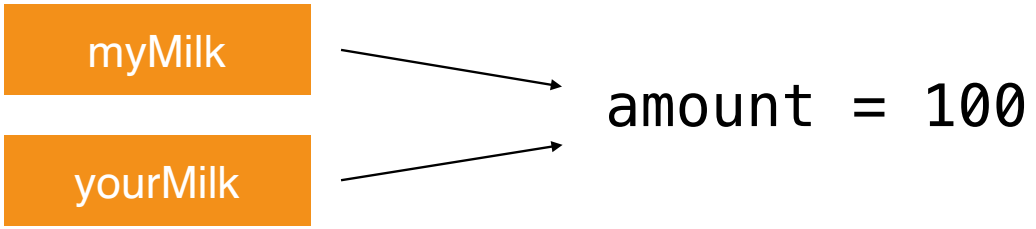
Call by value



Reference Types

```
var myMilk = ChocoMilk()
myMilk.amount = 300
var yourMilk = myMilk
yourMilk.amount = 100
print(myMilk.amount)
```

Call by reference



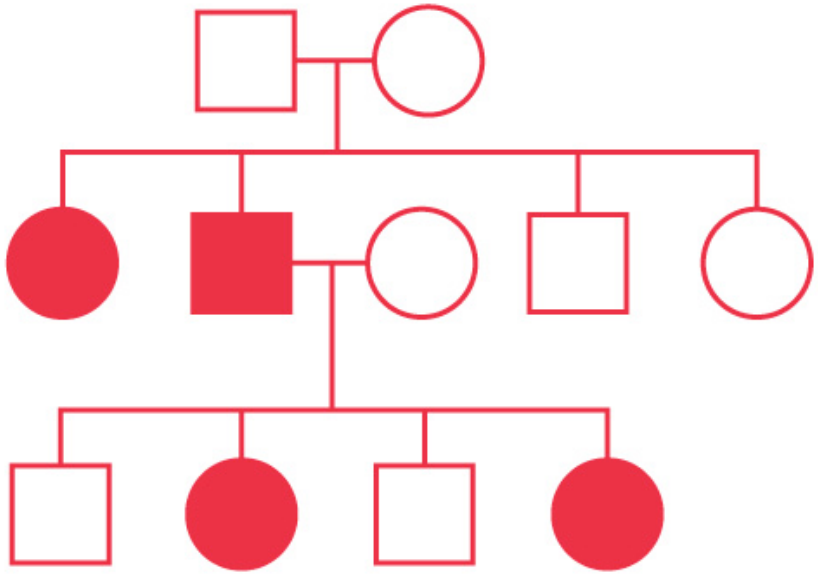
Value Types Reference

```
let origin = CGPoint(x: 0, y: 0)
var other = origin
other.x += 10
var another = origin
another.y += 5

func swapPoint(pointA : inout CGPoint, pointB : inout CGPoint) {
    let temp = pointA
    pointA = pointB
    pointB = temp
}

swapPoint(pointA: &other, pointB: &another)
print(other, another)
// (0.0, 5.0) (10.0, 0.0)
```

Call by reference

Class-based

구현 상속 inheritance



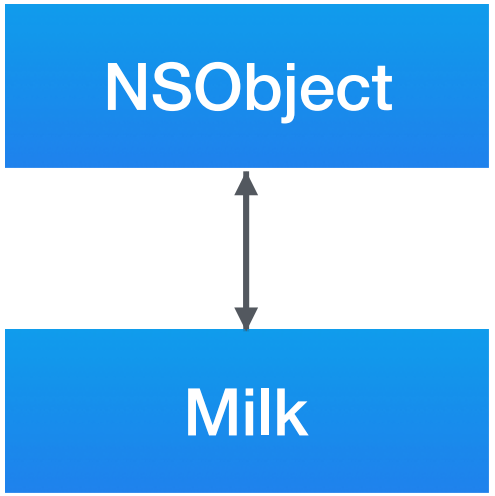
모든 클래스의 뿌리

* NSObject : Root Class
* 부모 클래스 (Parent class)

  슈퍼 클래스 (Super-class)
* 자식 클래스 (Child class)

  서브 클래스 (Sub-class)

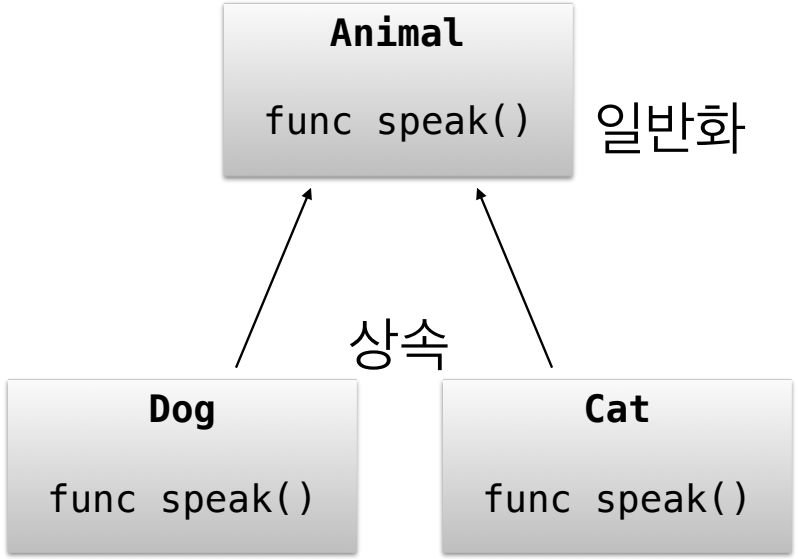
* 클래스 다중 상속 지원안함



다형성

Polymorphism

```swift
class Animal {
    func speak() {
        print("animal speak...")
    }
}

var animal = Animal()
animal.speak()

class Dog : Animal {
    override func speak() {
        print("dog - bow-wow")
    }
}

class Cat : Animal {
    override func speak() {
        print("cat - meow")
    }
}

var dog = Dog()
dog.speak()

var cat = Cat()
cat.speak()

var animalArray : [Animal] = [animal, dog, cat]
for x in animalArray {
    x.speak()
}
```



생성자 상속과 재정의

Designated initializer, Convenience Initializer

Rule 1

A designated initializer must call a designated initializer from its immediate superclass.

Rule 2

A convenience initializer must call another initializer from the *same* class.

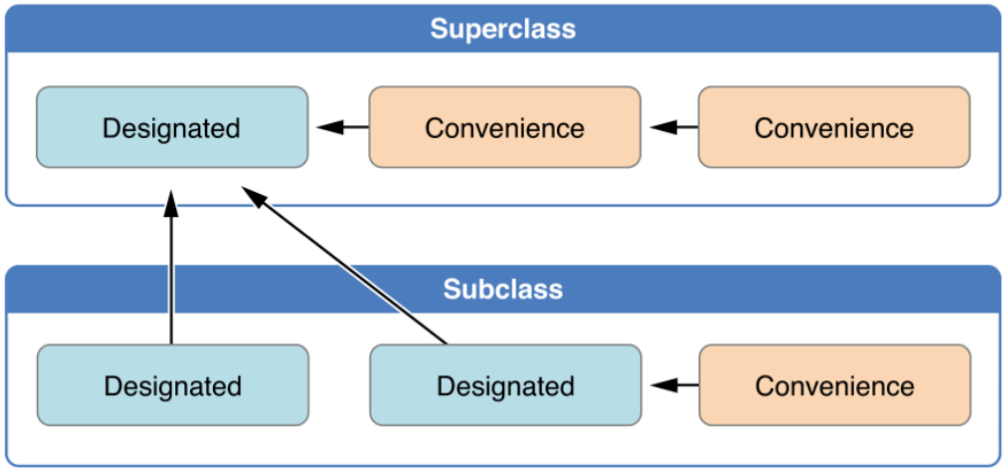
Rule 3

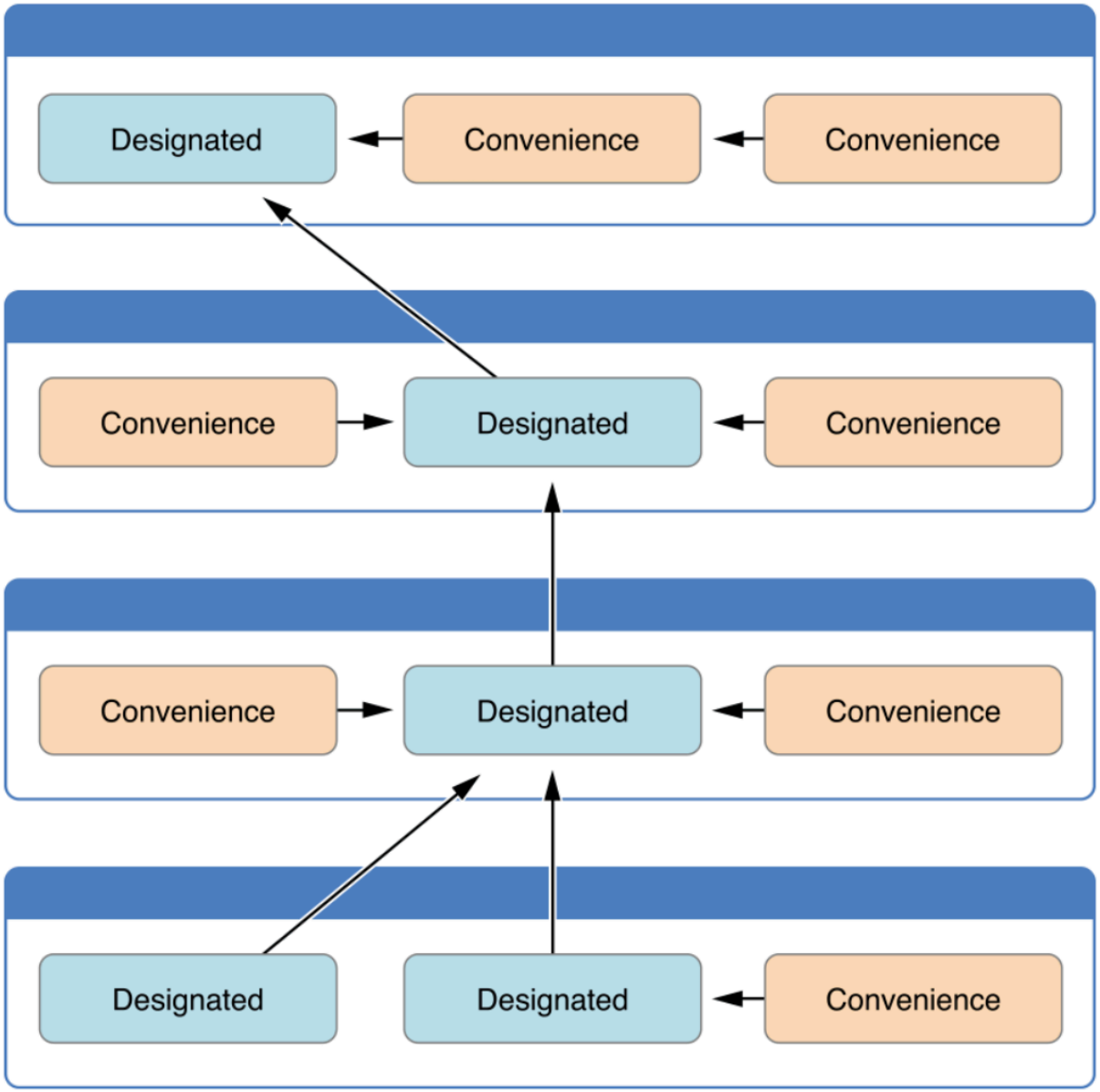
A convenience initializer must ultimately call a designated initializer.

A simple way to remember this is:

- Designated initializers must always delegate *up*.
- Convenience initializers must always delegate *across*.

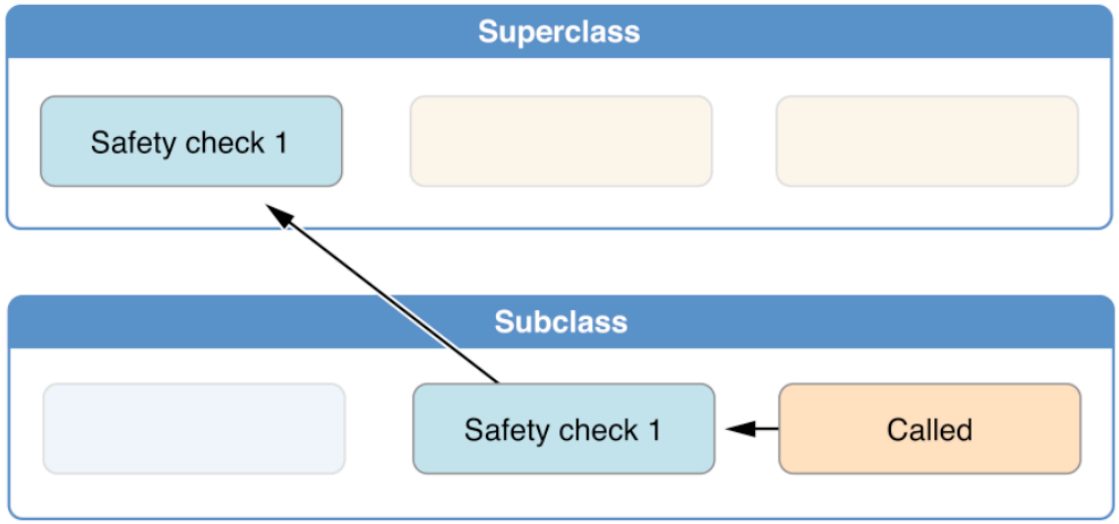
These rules are illustrated in the figure below:




Designated
Convenience
Convenience
Convenience
Designated
Convenience
Convenience
Designated
Convenience
Designated
Designated
Convenience

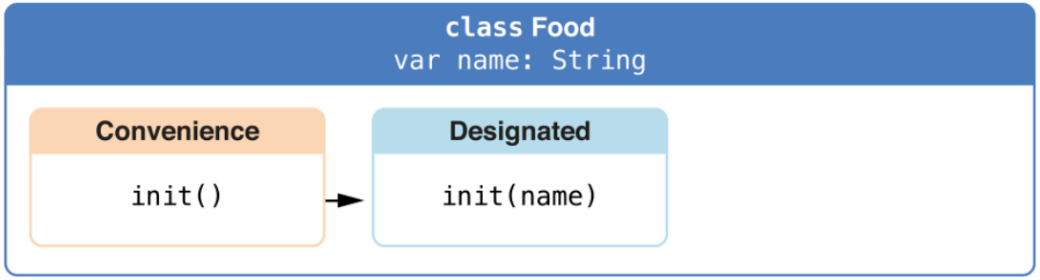
두 단계 초기화

1. 서브클래스의 지정 생성자가 수퍼클래스의 생성자를 호출하기 전에 자신의 프로퍼티를 모두 초기화했는지 확인한다.
2. 서브클래스의 지정 생성자는 상속받은 프로퍼티에 값을 할당하기 전에 반드시 수퍼클래스의 생성자를 호출해야 한다.
3. 편의 생성자는 자신의 클래스에 정의된 프로퍼티를 포함해서 그 어떤 프로퍼티 값을 할당하기 전에 다른 생성자를 호출해야 한다.
4. 초기화 1단계를 마치기 전까지 생성자는 인스턴스 메서드를 호출할 수 없다. self로 접근할 수도 없다.



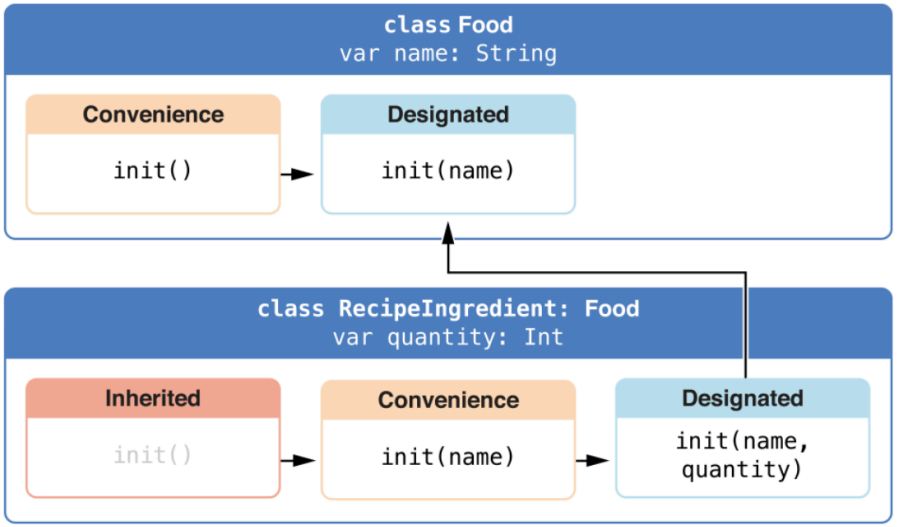
Convenience

```
class Food {
    var name: String
    init(name: String) {
        self.name = name
    }
    convenience init() {
        self.init(name: "[Unnamed]")
    }
}
```



Convenience 상속

```
class RecipeIngredient: Food {
    var quantity: Int
    init(name: String, quantity: Int) {
        self.quantity = quantity
        super.init(name: name)
    }
    override convenience init(name: String) {
        self.init(name: name, quantity: 1)
    }
}
```



타입 변환 type casting

Upcasting 와 Downcasting

타입을 바꾸자!

<Objective-C>
```
double value = 3.14
int convertedValue = (int)value
convertedValue = 1.234 //명시하지 않았지만 내부적으로 int 타입을 바꿔버림
```

<Swift>
```
var value: Double = 3.14
var convertedValue: Int = Int(value)
convertedValue = 1.234 //에러!
```

To check the type of an Instance

is-a

I am a boy.

To treat that Instance as a different superclass of subclass

as-a

He was dressed as a policeman.

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

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)
    }
}

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")
]
```

is-a 관계

```
var movieCount = 0
var songCount = 0

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

print("Media library contains \(movieCount) movies and \(songCount) songs")
// Prints "Media library contains 2 movies and 3 songs"
```

as-a 관계

```
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)")
    }
}

// Movie: Casablanca, dir. Michael Curtiz
// Song: Blue Suede Shoes, by Elvis Presley
// Movie: Citizen Kane, dir. Orson Welles
// Song: The One And Only, by Chesney Hawkes
// Song: Never Gonna Give You Up, by Rick Astley
```

as-a 관계

```
let someObjects: [AnyObject] = [
    Movie(name: "2001: A Space Odyssey", director: "Stanley Kubrick"),
    Movie(name: "Moon", director: "Duncan Jones"),
    Movie(name: "Alien", director: "Ridley Scott")
]

for object in someObjects {
    let movie = object as! Movie
    print("Movie: \(movie.name), dir. \(movie.director)")
}
```

```
for movie in someObjects as! [Movie] {
    print("Movie: \(movie.name), dir. \(movie.director)")
}
// Movie: 2001: A Space Odyssey, dir. Stanley Kubrick
// Movie: Moon, dir. Duncan Jones
// Movie: Alien, dir. Ridley Scott
```

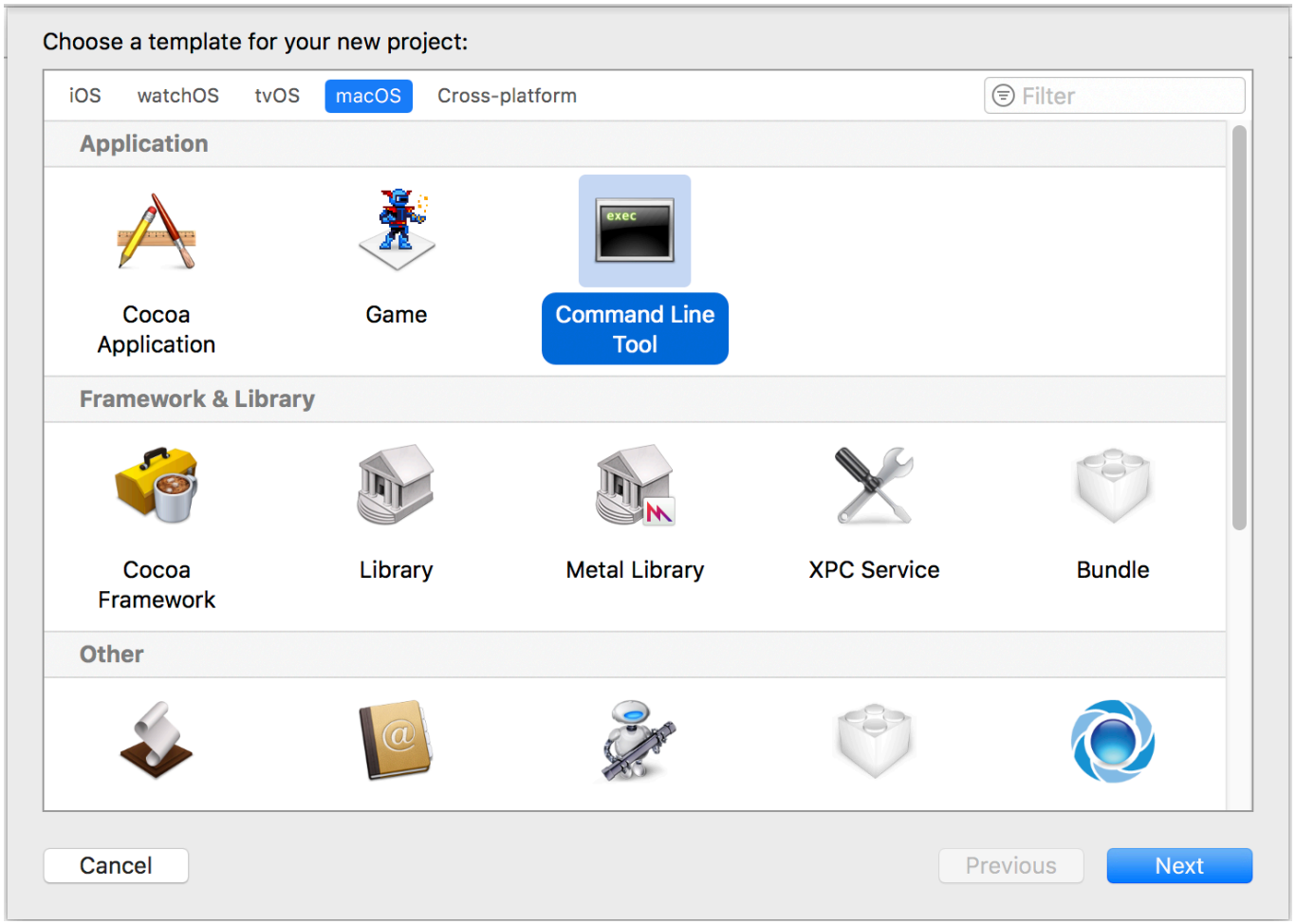
코딩미션#5

초간단 자판기 콘솔 앱

코드스쿼드

File > New > Project

macOS > Command Line Tool



간단한 자판기

미션: 딱 2가지 상품만 판매하는 자판기 struct를 만드세요

원하는 상품을 두 가지 struct로 만드세요. (예. 음료수, 옷, 식기…)

다음과 같은 동작을 위한 함수를 작성하세요.

main에서 테스트하기 위한 조건에 따라 함수를 호출하세요.

- 자판기 금액을 5000원 단위로 올리는 함수
- 상품 재고를 추가하는 함수
- 현재 금액으로 구매가능한 상품 목록을 리턴하는 함수

확장

subscript, protocol, extension, generic

서브스크립트

내부 요소에 접근하는 단축 문법

Subscript Syntax

```
subscript(index: Int) -> Int {
    get {
        //
    }
    set(newValue) {
        //
    }
}
```

```
struct TimesTable {
    let multiplier: Int
    subscript(index: Int) -> Int {
        return multiplier * index
    }
}
let threeTimesTable = TimesTable(multiplier: 3)
print("six times three is \(threeTimesTable[6])")
```

프로토콜 : 지켜야할 규칙

protocol

Communication Contracts

* 타입이 구현해야하는 함수 목록

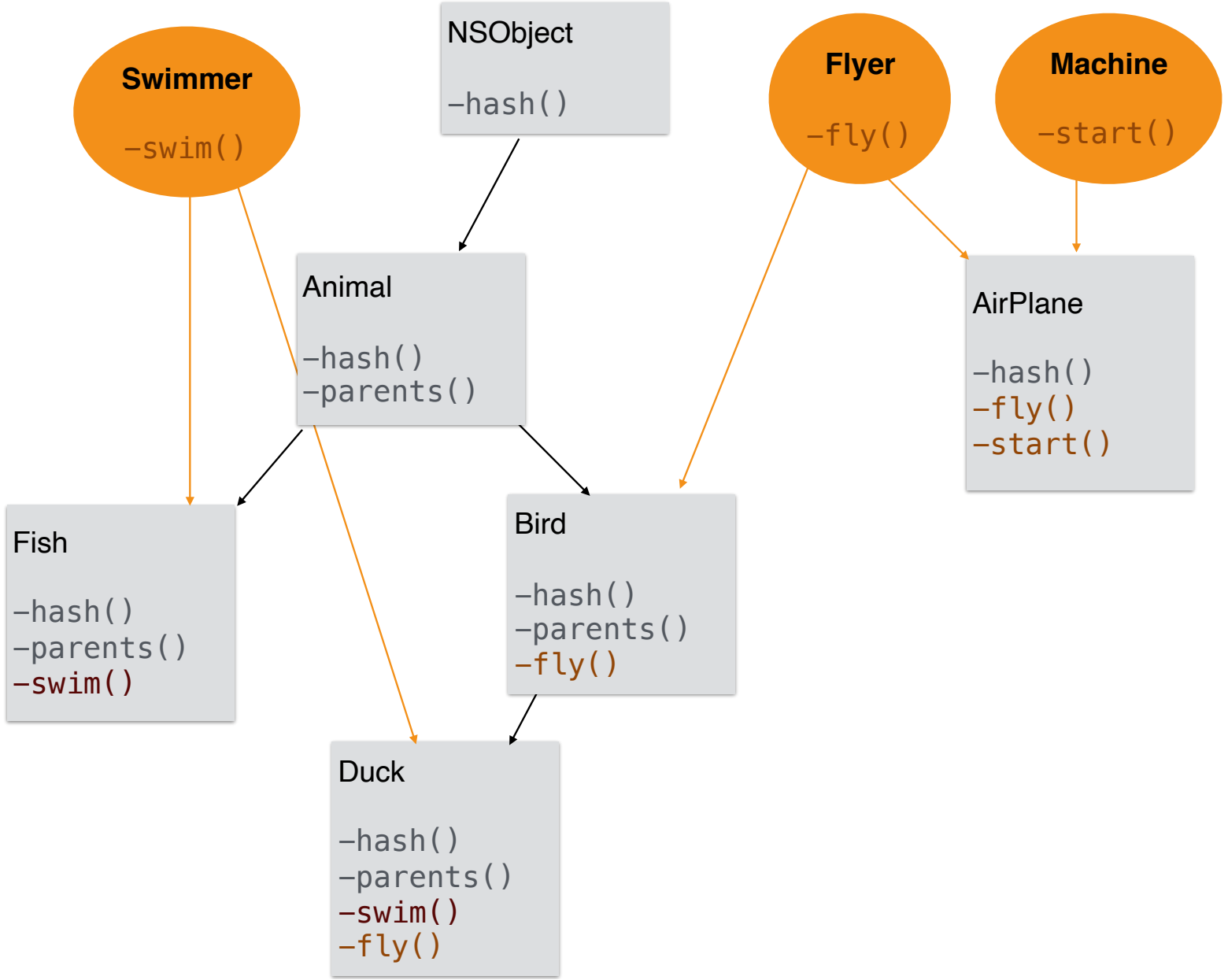
```
protocol Receivable {
    func received(data: Any, from: Sendable)
}

protocol Sendable {
    func send(data: Any)
}
```

* 타입이 구현해야하는 프로퍼티 목록

```
protocol SomeProtocol {
    var settableProperty: String { get set }
    var readOnlyProperty: String { get }

    static var someTypeProperty: Int { get set }
    static var otherTypeProperty: Int { get }
}
```

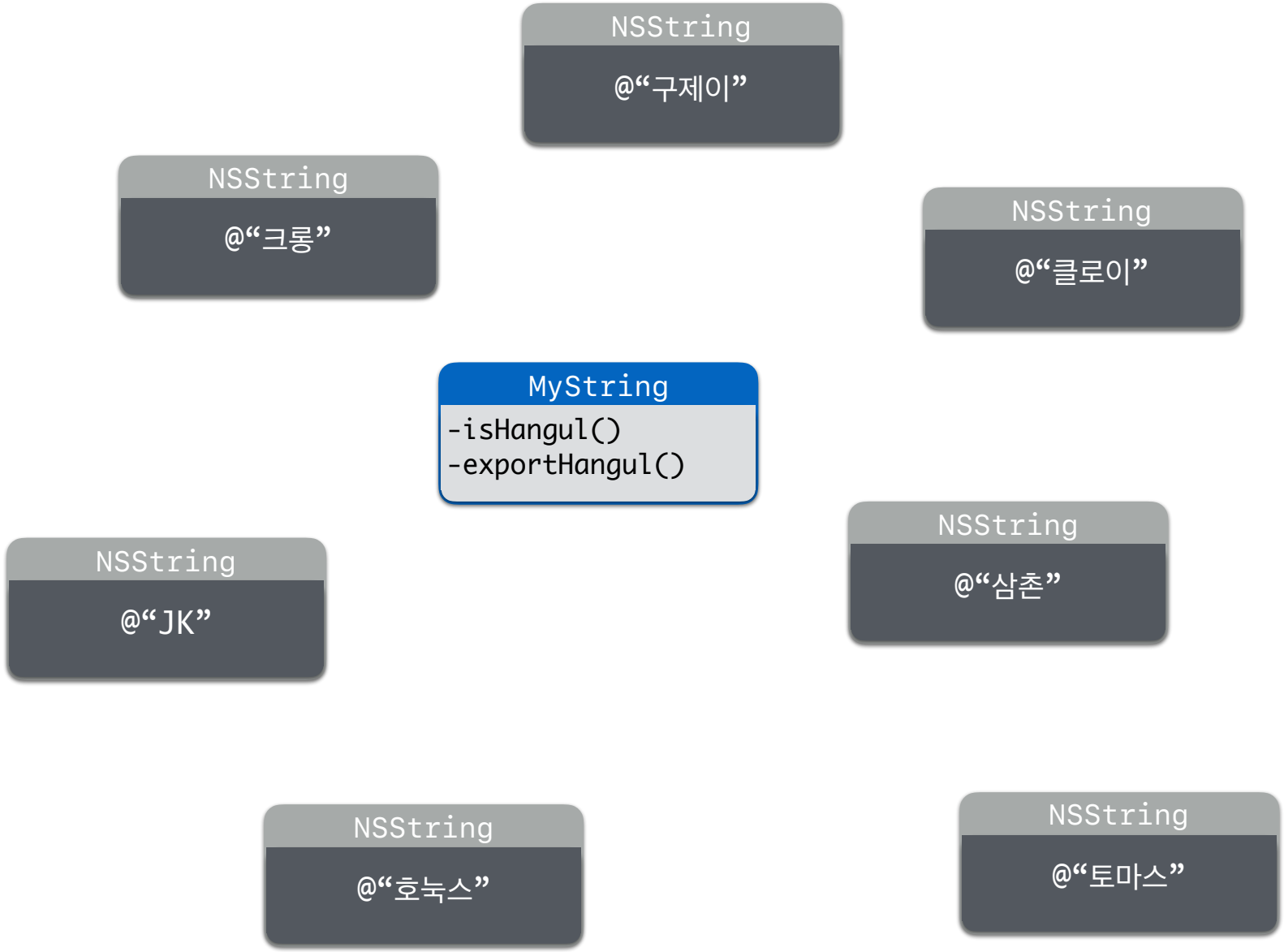
Confirming to a protocol

```
public protocol CustomStringConvertible {
    public var description: String { get }
}


class WorkManager : CustomStringConvertible {
    var nativeArray = [10,12,24]
    var stringArray = ["Ebcd", "Bcd", "Acc", "Dedd"]

    var description : String {
        return "nativeArray = \(nativeArray)"
    }
}

var instance = WorkManager()
print(instance)
//nativeArray = [10, 12, 24]
```


Swimmer
-swim()
NSObject
-hash()
Flyer
-fly()
Machine
-start()
Animal
-hash()
-parents()
AirPlane
-hash()
-fly()
-start()
Fish
-hash()
-parents()
-swim()
Bird
-hash()
-parents()
-fly()
Duck
-hash()
-parents()
-swim()
-fly()

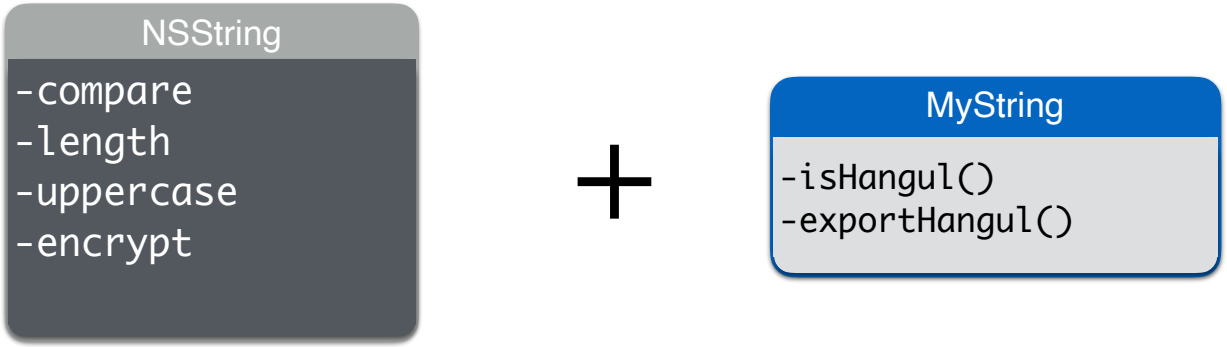
다이나믹 동작

Extension vs Category @objc


NSString
@“구제이”
NSString
@“크롱”
NSString
@“클로이”
MyString
-isHangul()
-exportHangul()
NSString
@“JK”
NSString
@“삼촌”
NSString
@“호눅스”
NSString
@“토마스”

익스텐션

나만의 기능을 추가하기



연산 타입/인스턴스 프로퍼티

타입 메서드 / 인스턴스 메서드

생성자

서브스크립트

중첩 타입

특정 프로토콜을 준수할 수 있도록 구현

익스텐션

나만의 기능을 추가하기

새로운 기능을 추가할 수 있지만, 기존의 기능을 재정의할 수는 없음!

	상속	익스텐션
확장	수직 확장	수평 확장
사용	클래스 타입만	클래스, 구조체, 프로토콜 제네릭 등 모든 타입
재정의	재정의 가능	불가능

익스텐션

기본 문법

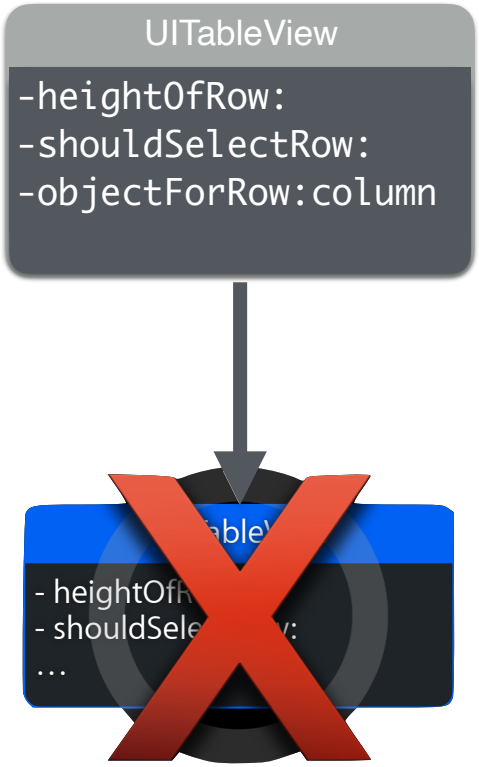
```
extension ExtensionTypeName {
    // 확장해서 구현할 내용
}
```

```
extension ExtensionTypeName : ProtocolName1, ProtocolName2 {
    // 프로토콜 구현 내용
}
```

```
extension Drink : CustomStringConvertible {
    var description: String {
        return "\(drinkName)"
    }
}
extension Drink : CustomDebugStringConvertible {
    var debugDescription: String {
        return "debug-\(drinkName)"
    }
}
```

익스텐션

구현 상속보다는 인터페이스 상속



제네릭

<Generics> solve

Generic Types

```
struct IntStack {
    var items = [Int]()
    mutating func push(_ item: Int) {
        items.append(item)
    }
    mutating func pop() -> Int {
        return items.removeLast()
    }
}


struct Stack<Element> {
    var items = [Element]()
    mutating func push(_ item: Element) {
        items.append(item)
    }
    mutating func pop() -> Element {
        return items.removeLast()
    }
}

var stackOfStrings = Stack<String>()
stackOfStrings.push("uno")
stackOfStrings.push("dos")
stackOfStrings.push("tres")
stackOfStrings.push("cuatro")
```