스위프트 프로그래밍 문법 읽기 과제

스위프트 문법에 익숙해지기 위한 **스위프트 프로그래밍 문법(Swift Programming Language)** 문서 읽기 과제[1]입니다.

모든 문서를 한번에 다 읽는 것이 아니라, 아래에 적힌 순서대로 해당 챕터를 읽으면 됩니다.

효율적인 시간 활용과 각 개념의 중요도 표시를 위해 아래와 같이 네가지로 분류했습니다. 개념의 중요도/난이도와 본인의 학습 진도를 고려하여 읽으시기 바랍니다.

빨간색 매우 중요한 개념으로 유심히 읽어야함. 이해하는데 조금 어려울 수 있음
노란색 중요한 개념. 이해하는데 크게 어렵지 않음
초록색 중요한 개념. 기초적이고 기본적인 내용
회　색 당장 필요한 개념은 아니지만 읽어두면 추후 도움이 되는 내용으로, 후반에 다시 읽어야함

그리고 하나 더, 스위프트 API 디자인 가이드 문서

스위프트 API 디자인 가이드(Swift API Design Guidelines)도 반드시 함께 읽으시기 바랍니다.

- 스위프트에 익숙하지 않다면, 가이드 내용을 완전히 이해하기 어려울 수 있습니다. 하지만 좋은 스위프트 코드 작성을 위해 가이드 문서에 익숙해지는 것은 매우 중요합니다.

- 아래 각 과제를 마친 뒤, 문서를 반복적으로 읽으며 가이드에 익숙해지십시오.

* Swift API Design Guidelines 원문 : https://swift.org/documentation/api-design-guidelines/
* Swift API Design Guidelines 번역문[2] : https://github.com/connect-boostcamp/SwiftAPIDesignGuidelines

* 문서 정리 도움 : **전미정** 님

---

[1] 본 과제는 Stanford CS193(Developing iOS 10 Apps with Swift) 강의의 Reading Assignment를 참고하여 번역 및 재구성하였습니다.

[2] 스위프트 API 디자인 가이드 원문을 번역해 놓았습니다. 원문과 비교하며 읽으면 더욱 좋습니다.

첫 번째 읽어오기 과제

The Basics

Constants and Variables
Comments
Semicolons
Integers
Floating-Point Numbers
Type Safety and Type Inference
Numeric Literals
Numeric Type Conversion
Type Aliases
Booleans
Tuples
Optionals
Error Handling
Assertions

Basic Operators

Terminology
Assignment Operator
Arithmetic Operators
Compound Assignment Operators
Comparison Operators
Ternary Conditional Operator
Nill Coalescing Operator
Range Operators
Logical Operators

Strings and Characters

String Literals
Initializing an Empty String
String Mutability
Strings Are Value Types
Working with Characters
Concatenating Strings and Characters
String Interpolation
Unicode
Counting Characters
Accessing and Modifying a String
Comparing Strings
Unicode Representations of Strings

Collection Types

Mutability of Collections
Arrays
Sets
Performing Set Operations
Dictionaries

- 만약 배열에 들어있는 원소의 개수보다 더 높은 인덱스에 접근하려고하면 프로그램이 강제종료(crash)됩니다.
- 배열(Array)에는 마지막 원소를 꺼내오는 *last*라는 메서드가 있지만 이 메서드에서 반환하는 값은 *Optional*입니다. (배열이 비어있을 경우 nil이 반환 될 수 있기 때문입니다.)
- 배열에 사용되는 += 연산자는 배열 자체를 하나의 원소로 더하지 않고, 더해지는 배열의 원소들을 오른쪽에 추가합니다. (ex. var arr1 = [1, 2, 3], var arr2 = [4, 5, 6], arr1 += arr2 // [1,2, 3, 4, 5, 6])

Control Flow

For loops
While loops
Conditional Statements(switch)(Tuples, Value Binding, Where)
Control Transfer Statements(Labeled Statements)
Early Exit
Checking API Availability

- 스위프트에서의 *switch* 구문은 C 언어나 다른 언어에서 보다 훨씬 중요합니다.

Functions

Defining and Calling Functions
Function Parameters and Return Values(Functions with Multiple Return Values, Optional Tuple Return Types)
Function Argument Labels and Parameter Names(Variadic, In-Out Parameters)
Function Types
Nested Functions

Closures

Closure Expressions
Trailing Closures
Capturing Values
Closures are Reference Types
Non-escaping Closure
Autoclosures

- 스위프트에서 함수(전달인자와 반환 값 등을 가지는 종류의 것들)가 일급객체(혹은 일급시민)으로, 타입으로 사용될 수 있다는 것을 이해하는 것은 매우 중요합니다.
- 많은 iOS의 API가 클로저를 전달인자로 사용합니다.

Enumerations

Enumerations Sytax
Matching Enumeration Values with a Swift Statement
Associated Values
Raw Values
Recursive Enumerations

Classes and Structures(클래스와 구조체)

Comparing Classes and Structures
Structures and Enumerations are Value Types
Classes are Reference Types
Choosing Between Classes and Structures
Assignment and Copy Behavior for Strings, Arrays, and Dictionaries

- 스위프트에서 클래스와 구조체는 *initializers, functions*, *properties* 부분에서 매우 유사합니다. 하지만 이 둘은 **값 타입 vs 참조 타입**이라는 중요한 차이를 지니고 있으며, 해당 챕터에서 강조하고있으니 유심히 읽어보시기 바랍니다.

Properties

Stored Properties(Lazy Stored Properties, Stored Properties and Instance Variables)
Computed Properties
Property Observers
Global and Local Variables
Type Properties

두 번째 읽어오기 과제

Basic Operators

Nill Coalescing Operator

Strings and Characters

Unicode
Accessing and Modifying a String
Unicode Representations of Strings

- 이번 과제에서 *unicode*에 대해 특별히 알아둬야 할 내용은 없지만, *unicode*는 알아두면 두루두루 좋습니다.

Collection Type

Sets(노트 박스는 무시 가능)
Performing Set Operations

- 배열(Array)의 초기화(initializers)에 대해서도 읽어두시기 바랍니다.

Controll Flow

Conditional Statements
Control Transfer Statements
Early Exit

- Conditional Statements 부분의 Tuples, Value Bindings 그리고 Where 부분을 함께 읽어두시기 바랍니다.

Functions

Function and Return Values 부분의 Functions with Multiple Return Values, Optional Tuple Return Types
Function Argument Labels and Parameter Names 부분의 Variadic과 In-Out Parameters

Closure

Capturing Values
Closures are Reference Types
Nonescaping Closures
Auto Closures

- 클로저는 굉장히 중요한 개념이기 때문에, 나중에 다시 깊게 살펴봐야합니다. 이번 과제에서 너무 진을 빼진 않길 바랍니다.

Properties

Stored Properties 부분의 Lazy Stored Properties, Stored Properties and Instance Variables

Property Observers
Global and Local Variables
Type Properties

Methods

Instance Methods(Parameter Names)
Type Methods

- *Instance method*와 *Type method*의 차이점을 확실히 이해해야합니다.

Subscripts

Subscript Syntax (읽는건 선택사항이지만 읽어두면 정말 좋아요!)
Subscript Usage
Subscript Options (역시, 선택사항이지만 읽으면 최고!!)

Inheritance

Defining a Base Class
Subclassing
Overriding
Preventing Overrides

Initialization

Setting Initial Values for Stored Properties
Customizing Initialization
Default Initializers
Initializer Delegation for Value Types
Class Inheritance and Initialization
Failable Initializers
Required Initializers
Setting a Default Property Value with a Closure or Function

- 이번 챕터를 모두 다 읽지 않고 *initialization*을 이해하기란 굉장히 힘든 일입니다. 기본적인 내용에 먼저 집중하는 것이 좋습니다.

- UIViewController의 initialization은 꽤 복잡하기 때문에 관련 내용을 추가해 두었습니다. (당장 필요한 내용은 아니기 때문에 다음 파트로 넘어가도 좋지만, UIViewController에 *initializer*가 필요하다면 최소한 아래 내용은 알아둬야 합니다.)

- UIViewController는 아래 두 가지 *initializer*를 지니고 있으며 *subclass*에서 둘다 구현하거나 둘다 구현하지 않아야합니다.

```
    override init(nibName nibNameOrNil: String?, bundle nibBundleOrNil: Bundle?) {
        super.init(nibName: nibNameOrNil, bundle: nibBundleOrNil)
        // 초기화 내용
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        // 초기화 내용
    }
```

- *override*와 *required* 키워드를 절대 잊지마세요.
- 가능하다면(항상 가능하겠지만) **위 두가지 메소드는 구현하지 않는 것이 좋습니다.**

- UIViewController의 초기화는 ViewController의 생명주기 중 하나인 아래 메소드에서 주로 작업합니다.

```
    override viewdidLoad() {
        super.viewDidLoad()
        // 초기화 내용
    }
```

- 위 메소드가 호출 될때 모든 outlets은 연결 되어있지만 아직 화면에는 나타나지 않은 상태로, 초기화 작업을 하기 매우 좋은 시점입니다.
- 프로퍼티를 암시적 추출 옵셔널으로 생성해 위 메소드에서 초기화하는 전략을 사용하는 것이 좋습니다.

Optioanal Chaining

챕터 모든 부분

- 이 부분은 모두 읽어서 명확하게 해두는 것이 좋지만, 아직 Optional 개념을 완전히 이해하지 못했다면 지금 당장 이 챕터를 읽는 것이 어려울 수 있습니다.

Type Casting

챕터 모든 부분

- 스위프트는 극도의 타입 안정성(type safe)을 추구하는 반면, iOS 자체는 그리 엄격하지 않은 방향으로 발전해 왔습니다. iOS API를 사용하다보면 어떤 객체의 클래스를 알고 싶거나 지정해줘야 할 경우가 종종 발생합니다. 스위프는 그런 경우를 안전하게 다룰 수 있는 방식을 제공하고 있습니다.

Nested Types

챕터 모든 부분

Generics

- 지금 알아야 할 부분은 Generic을 어떻게 사용하는가 입니다.
- 사용법은 매우 간단한데 예를 들어 Array<Double> 나 Dictionary<String, int>와 같이 사용할 수 있습니다.
- 이와 관련된 자세한 내용은 이번 챕터에 설명되어 있으며 읽는 것은 선택사항입니다.

Access Control

- 이미 읽어야할 내용이 충분히 많으므로, 지금 당장 챕터 모든 부분를 읽으라고 하지는 않겠습니다. 본 챕터의 내용을 간단히 요약하면 다음과 같습니다.
- (특별히 예외 상황이 아니라면) 우리가 만드는 모든 API 앞에 private 또는 fileprivate 키워드를 붙여야 합니다.

- 보통 우리는 개발을 할때 여러명의 프로그래머와 함께 개발하게 되는데, 다른 프로그래머들이 당신이 만든 코드를 사용하려는 경우가 발생할 수 있습니다. 이럴때 method나 property의 접근 수준 정도를 나타내는 것이 access control 입니다.

Advanced Operators

Overflow Operators

- 산술 오버 플로우 에러 확인을 위해 위 연산자를 사용할 수 있습니다.

세 번째 읽어오기 과제

* 세번째 과제에서는 앞의 두 과제에서 회색으로 표시해두었던 내용 또는 다시 읽어보면 좋을 내용을 살펴봅니다.
* **빨간색**은 매우 중요함, **노란색**은 그다지 중요하지 않음을 의미합니다.

The Basics

Error Handling

Subscripts

Subscript Syntax
Subscript Options

Initialization

Class Inheritance and Initialization
Failable Initializers
Required Initializers

Deinitialization

이 부분은 실제로 활용하는 일은 드물지만 읽어두면 차후에 도움이 될 것입니다.

Automatic Reference Counting(ARC)

ARC에 대해 너무 겁먹지 않으시기 바랍니다. 순환참조(reference cycles)은 흔히 발생하는 일이 아닙니다. 하지만 클로저에서 강한 참조로 생성된 무언가를 다시 또 강한 참조 하지 않게하는 일은 매우 중요하므로…

Strong Reference Cycles for Closures

Optional Chaining

챕터 모든 부분

Error Handling

챕터 모든 부분

Type Casting

챕터 모든 부분

Nested Types

챕터 모든 부분

Extensions

- 익스텐션은 정말 굉장하고 유용한 기능이지만 코드의 가독성을 저해할 가능성이 있습니다.
- 코드의 가독성은 굉장히 중요한 것이기 때문에 너무 많은 익스텐션이나 이상한 익스텐션을 구현해서 사용하여 가독성을 떨어트리지 않도록 유의하십시오.
- 그렇다고 익스텐션을 기피할 이유는 없습니다. 적재적소에 사용할 수 있도록 많은 고민이 필요한 부분입니다.

Protocols

챕터 모든 부분

- 매우 중요한 개념입니다. 유심히 읽으시기 바랍니다.

Generics

챕터 모든 부분

Access Control

챕터 모든 부분

- framework를 직접 만드는 경우가 아니라면, public 키워드를 사용하는 일은 절대 없습니다.
- 적절하게 private(혹은 fileprivate) 키워드를 사용하는 것은 매우 중요합니다.

Advanced Operators

챕터 모든 부분