2016. 08. 18 ~ 8. 31

Swift BASIC

---

let과 var

- let으로 상수 선언하기
- var로 변수 선언하기
- 선언의 원형 & 생략한 타입 추론'
- let을 많이 사용하고 var를 줄이는게 좋다. 컴파일러도 그렇게 권장한다.

String

: 문자열 데이터를 다루는 작업은 우리가 앱을 제작할 때 많은 비중을 차지한다. 앱이 다루는 많은 부분이 문자열로 이루어져 있을 뿐만 아니라 파일의 경로나 웹 주소 등도 문자열이기 때문이다.

- 빈 문자열 만들고 문자열 추가하기
- Character 타입의 배열로 변환하기. 글자 수 카운트하기
- "\( )" 이용해서 문자열 구성하기

Numbers

- int는 정수(Int 32, Int 64), UInt는 양의 정수
- Float는 36bit 소수, Double은 64bit 소수, 그냥 만들면 Double

Tuple

- 코마로 구분된 값의 리스트
- 간단하게 만들어 쓰고 읽을 수 있다.
- element 번호로 접근하거나 만들 때 지정한 이름으로 접근 가능

typealias

- 간단한 타입 지정
- 예를 들어, Int 타입으로 시간 간격을 나타내고 싶다면, 사실은 Int인 TimeInterval 값을 만들 수 있다.

Collections: Array

- Array<Type>( ) 또는 [Type]
- Array도 let으로 만들면 수정 불가
- Array 안에는 동일한 타입의 인스턴스가 들어가야 함

Collections: Dictionary

- 키와 값의 쌍으로 이루어진 컬렉션
- 문자열 키에 정수 값을 가지는 Dictionary의 타입 선언은 [String:Int]
- keys와 values로 키, 값 배열 추출하기

Set

- 순서를 가지지 않은 컬렉션
- 주로 집한 연산이 필요한 경우 사용한다.

    Intersect, Subtract, Union, ExclusiveOR

Control Flow

: 흐름 제어문에는 조건 분기문과 반복문이 있다. if와 switch는 자주 사용하는 조건 분기문이며 for는 대표적인 반복문이다.

- if문

    if condition {…1} else {…2}

    condition이 참이면 …1 실행

    condition이 거짓이면 …2 실행

- for문

    조건에 따라 일정 구문을 반복해서 실행

    Array 컬렉션의 for문

    Dictionary 컬렉션의 for문

- switch문

    하나의 값에 다양한 상태에 대한 매칭

    switch 값

        case 조건

    모든 가능성 있는 조건에 대응하는 구문이 없으면 default로 처리

Optional

: 옵셔널은 '값이 없는' 상태를 나타낸다.

- nil

    '값이 없음'을 나타냄

    모든 타입의 변수는 nil을 가질 수 있다. (Int, 구조체 인스턴스, 오브젝트)

    nil에 접근 시 프로그램이 크래시를 일으킨다.

- 옵셔널 만들기

  변수 타입 뒤에 ?를 붙이면 옵셔널이 된다.

  옵셔널로 선언된 변수는 접근 시에 특별한 문법이 필요하다.

- 옵셔널 변수 안에 nil 아닌 값이 존재한다고 확신할 때 사용하는 옵셔널 접근 방법

  하지만 개발자의 확신은 경험적으로 수많은 에러를 만들어 냄

  옵셔널을 무력화하는 시도임

  !가 많은 코드는 나쁜 코드

- Optional Binding

  if let으로 옵셔널이 아닌 새로운 상수를 만들어 사용

  새롭게 만들어진 상수는 옵셔널이 아니므로 편하게 사용

- Implicitly Unwrapped Optional

  옵셔널 사용을 편하게 하기 위한 편의장치

  어쩔 수 없이 옵셔널이지만 실행 중 항상 값을 가지는 게 거의 확실

  선언시에 !를 사용하면 옵셔널이지만 옵셔널이 아닌 것처럼 사용

Function

: 함수는 입력되는 값에 특정한 작용을 해서 출력값을 만들어 주는 대응 관계이다.

-Swift 함수의 선언

parameterName을 통해 함수로의 입력값이 들어가고, parameter로 사용할 수 있는 타입이 함수 선언 시에 결정된다.

returnTyple은 이 함수의 실행결과로 돌려줄 값의 타입이며 이 역시 함수 선언 시에 결정된다.

매개변수 : history라는 이름의 Double 타입 배열

리턴타입 : 3개의 더블값으로 구성된 튜플

Structure

: 구조체는 Class와 함께 데이터 모델의 중요한 구성요소 중 하나.

- 내부에 변수나 상수 또는 함수를 선언한 뒤 인스턴스(Instance)를 만들어서 사용
- 주로 좌표나 크기처럼 간단한 값을 표현하는데 많이 사용되어 왔다.
- Swift에서 Class의 대안으로 그 역할이 커지고 있다.

  구조체는 Value Type

  Value 타입의 인스턴스

    Int나 Double과 같이 직접 값을 가지는 것

  Reference 타입의 인스턴스

인스턴스가 있는 메모리 번지를 참조하는 것

Value 타입은 할당시에 인스턴스가 복사되지만

Reference 타입은 할당시 참조하는 메모리 번지만 전달

Class

: 데이터 모델의 중요한 구성요소로써 Object를 만든다.

클래스는 객체지향 프로그래밍(OOP, Object Oriented Programming)의 바탕을 이룬다.

- 클래스로 만든 오브젝트 Reference Type으로 동작하며, 할당시 복사되지 않는다.

  하나의 인스턴스에 대한 레퍼런스를 공유한다.

  인스턴스를 할당한 뒤 수정하면 원본과 할당된 곳 모두 수정됨

  let으로 정의해도 프라퍼티가 var이면 수정 가능

Enumerations

: 연관성 있는 값들의 그룹을 만들어 Type-Safe하게 사용하는 것이다.

- 여러 다른 언어에서 일련의 값에 일대일 대응되는 Enum을 정의해서 사용

  Task의 상태 값을 나타내기 위해 0, 1, 2등의 정수값을 사용

  보다 직관적으로 0은 READY, 1은 COUNTING, 2는 PAUSED, 3은 DONE과 같이 Enum으로 정의해서 사용하는 방식

- Swift에서 Enumeration은 더욱 강력한 기능을 가진다.

  1st class type

  매개변수나 리턴타입으로 사용

  메소드를 가진다거나 프로토콜을 준수

Initialize

: 초기화 작업은 인스턴스가 가지고 있는 모든 스토어드 프라퍼티(Stored Property)들의 최초 값을 설정하는 것

- 스토어드 프라퍼티

  저장소를 가지고 있는 프라퍼티

- 컴퓨티드 프라퍼티

  저장소 없이 계산에 의해 값을 리턴하는 프라퍼티

- 구조체의 초기화

  여러 개의 init 메소드 허용

  상속을 허락하지 않으므로 Class에 비해 상대적으로 단순

- 클래스의 초기화

    하나의 지정초기화 메소드

    여러 개의 편의 초기화 메소드 허용

Method

- 메소드란 인스턴스 안에 종속된 함수
- 인스턴스에서 필요한 기능들 함수로 만들어 넣은 것
- 클래스, 구조체, 이너머레이션 모두 인스턴스 메소드를 가질 수 있다.
- self는 그 자신을 가리킨다.

Enum Associsted Value

- TaskType이라는 이너머레이션에 Associated Value를 설정
- Task 구조체 안에 선언한 doBasicTask( )라는 메소드

    Task의 타입 값에 따라 switch문을 통해 각각 다른 작업을 수행

        .Call이면 전화번호를 number 상수로

        .Report이면 receiver와 time이라는 값

        .Meet과 .Support의 경우 적당한 상수값