**스위프트 UI 1강(CS193p 2023)**

**구술적(narrative) 방식 vs 삽화적(vignette) 방식**

구술적 방식은 맥락을 스토리로 만들어 가며 배우는 방식, 삽화적 방식은 한 가지 기능을 구현하기 위해 웹에서 한 사례(현재의 맥락과는 거리가 있는)를 가져와 배우고 익히는 방식, 두 가지 방법을 같이 진행하는 게 이상적

객체지향은 클래스를 통해 데이터를 캡슐화하고 객체를 정의하는 패러다임이라면 함수형 프로그래밍은 구조체를 통해 행위를 정의하고 어떠한 행위를 하는 구조체인지 정의하는 패러다임이다. 스위프트의 구조체는 객체지향의 클래스처럼 변수와 함수를 갖지만 상속은 하지 않는다. 또 객체지향이 데이터 캡슐화를 기반으로 하는 것과 달리 함수형은 행위의 모듈화를 기반으로 한다.

일반 변수와 연산되는 변수

Some 변수

View 조립, 분해

Struct는 블록의 단위

시스템 네임 파라미터

디폴트 파라미터

중괄호 표현 함수가 하나의 블록 단위가 되고 이를 조립, 해체해가며 함수의 파라미터로 활용하는 모습이 함수형 프로그래밍의 일반적 풍경이다.

중괄호는 하나의 뷰를 반환한다. 뷰를 쌓아 뷰에 넣어 새로운 뷰를 만든다.

H, V, Z

Rotor 레고의 비유, 그리드,

**1차 유인물(+ 스위프트 5.9.2 공식문서)**

1차 유인물의 목적은 스위프트의 기본 요소들(변수, 제어흐름, 메소드, 프로퍼티, Arrays and Strings)을 이해하면서 스위프트가 강점으로 가지는(라벨 인자, 패턴 매칭 스위치문, 일급 객체로서의 함수, 객체지향과 함수형 프로그래밍의 혼합…) 부분을 익숙하게 하는 것이다.(4강 듣기 전에 1차 유인물 다 보기)

**스위프트란…**

스위프트는 현대적인 프로그래밍 패턴을 채택해 다른 프로그래밍 언어들에서 흔히 발생하곤 하는 문제들을 피한다.

1. 변수는 항상 사용하기 전에 선언되어야 한다.
2. 인덱스는 경계값 오류를 발생시키기 전에 미리 확인한다.
3. 정수값은 오버플로우를 발생시키기 전에 확인한다.
4. 옵셔널 타입은 Nil 값을 명시적으로 다루도록 보장한다.
5. 메모리는 자동으로 관리된다.
6. 예기치 못한 실패 시 에러 핸들링이 통제된 회복을 허용한다.

스위프트는 현대적 하드웨어 위에서 빠르고(단순한 문법), 안정적(강력한 타입추론, 패턴 매칭)이고, 표현력 있는 소프트웨어를 정확하게 구현하기 위해 만들어졌다. 스위프트는 쓰기 쉽고 읽기도 쉬운 코드를 지향한다.

**스위프트 둘러보기**

스위프트는 여타의 언어와 같이 Main()으로 시작하지 않고 문장을 세미콜론으로 마무리 할 필요가 없다. 또한 기본 출력 함수인 print(“”)를 사용하기 위해 라이브러리를 불러올 필요도 없다. 메인 함수가 없으므로 전역으로 선언된 코드는 엔트리 포인트가 된다.

**간단한 변수**

let으로 선언된 변수는 상수, var로 선언된 변수는 변수가 된다. 상수는 컴파일 시점에 알려질 필요가 없지만 반드시 한 가지 값으로만 선언되어야 한다. 타입을 선언하면 반드시 할당되는 값과 타입이 일치해야 하지만 명시적 선언이 강제되지는 않는다. 컴파일러가 값을 통해 자동으로 타입을 추론하기 때문이다. 다만 초기에 할당되는 값으로 타입을 추론하기 어렵다면(ex. 원하는 타입은 double인데 초기값이 정수인 경우) let explicitDouble: Double = 70과 같이 변수명 뒤에 콜론을 붙이고 뒤에 타입을 표기한다.

**명시적 형변환**

묵시적 형변환은 지원하지 않는다. 타입이 다른 값을 연산하려면 반드시 명시적 형변환을 해야 한다. Ex. “The number is ” + String(number)

문자열 안에서 다른 변수를 연산하려면 해당 변수들을 ₩()로 감싸면 된다. 또 문자열의 줄바꿈을 ₩n을 통해 표기하지 않고 화면 그대로 표현하고 싶다면 “””를 통해 문자열을 여닫으면 된다.

**배열과 딕셔너리**

인자를 대괄호[]로 감싸고 컴마로 구분해 선언하면 된다. 딕셔너리는 대괄호로 감싸고 콜론을 통해 키-밸류를 구분한다. 배열의 경우 .append()를 통해 값을 삽입하면 자동으로 길이가 늘어난다.

fruits = []

occupations = [:]

위는 빈 배열, 아래는 빈 딕셔너리, 만약 타입 추론을 하기 힘든 상황이라면

let emptyDictionary: [String: Float] = [:]와 같이 선언해준다.

**제어흐름**

If나 switch의 조건절에 소괄호는 선택사항이지만 대괄호로 이행 사항을 감싸는 것은 필수. 또한 If조건의 조건절은 반드시 불리언 값이어야 한다. 특징적으로 변수에 조건문을 할당하는 것이 가능하다.

let scoreDecoration = if teamScore > 10 {

"🎉"

} else {

""

}

print("Score:", teamScore, scoreDecoration)

// Prints "Score: 11 🎉"

옵셔널 변수는 타입 뒤에 물음표를 붙여 해당 타입이 nil일 수도 있음을 명시한다.

var optionalString: String? = "Hello"

print(optionalString == nil)

// Prints "false"

var optionalName: String? = "John Appleseed"

var greeting = "Hello!"

if let name = optionalName {

greeting = "Hello, \(name)"

}

If let이 함께 쓰인 경우 if가 optionalName이 값을 가지고 있는지 먼저 체크한다. 만약 값을 가지고 있다면 이 값은 name에 할당되며 이는 다시 if의 범위 안에 들어가게 된다. If 범위 안에서 {}내의 greeting = “Hello, \(name)”이 실행되게 된다.

여기서 if를 변수 선언 앞에 둬 optionalName이 nil인 경우에 발생할 수 있는 런타임 오류를 막고 있다.

let nickname: String? = nil

let fullName: String = "John Appleseed"

let informalGreeting = "Hi \(nickname ?? fullName)"

여기서는 ?? 기호를 사용해 nickname이 nil일 경우 디폴트 값인 fullName을 할당하도록 한다.