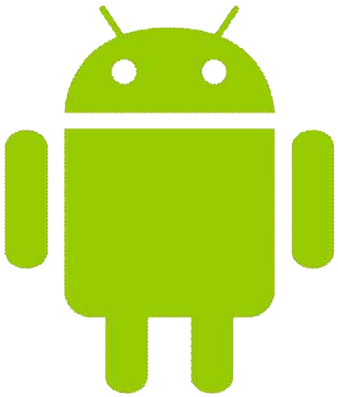
KOTLIN

Education



MADE IN HONG

INDEX

1 왜 코틀린을 배우는가?

앱 개발 3가지

웹 뷰를 감싸고 있는 웹 사이트!

네이티브 앱에 대한 지식 X

But,,모든 UI를 직접...

폰의 강력한 기능을 사용하지 못한다.. ㅠ

앱 개발 3가지

네이티브 코드가 아닌걸로 코딩을 한 후, 나중에 android가 이해할 수 있는 코드로 변환!

코드 한번 작성하고 2개의 플래폼으로?!

지금도 발전중,,! 단점은 네이티브가 아니라 퍼포먼스 차이가 있다..

react native > java script > run

앱 개발 3가지

https://gdg.community.dev/events/details/google-gdg-seoul-presents-flutter-engage-extended-seoul/#/

앱 개발 3가지

말그대로 iOS, android 코드를 사용!

iOS -> swift , android -> java, kotlin

폰이 가지고 있는 최대 퍼포먼스를 사용가능!

중간 단계가 없으니까!

단점을 둘 다 배워야함 ㅠㅠ

2 코틀린 문법

코틀린은 자바처럼 정적 타입의 언어입니다!

다시말해 모든 표현식의 타입이 컴파일 시점에 알 수 있다!

자바와 동일하게 객체지향 프로그래밍을 지원!

Null 값으로 인해 발생할 수 있는 NulPointerException

예외가 생기지 않도록 언어 자체에서 배려하고 있어서 안전하다

1. 함수

```
//1. 함수

//  1. 함수

//Unit 은 아무것도 리턴형이 없을때
//최재 글씨는 안써도 무방하기는 함
fun helloWorld() : Unit {
    println("Hello World")
}

//변수 타입 순서임
//리턴 타입이 있으면 반드시 써줘야함
fun add (a : Int , b : Int) : Int {
    return a+b
}
```

2. 변수

```
//2. val = value  변하지 않는값
//var = 변할 수 있는값

fun hi(){
    val a  : Int = 10
    var b : Int = 9
    var c = 99 // 뒤에 자료형을 쓰지 않아도 된다.
    var name : String = "Tom"
    //a = 100 값을 변경하려해서 오류가 나온다.
    b= 100

    // 변수 타입을 선언만 하고 싶을때는??
    // var game (x)
    var game : String //(o) 타입명까지 기재
}
```

타입 추론 개꿀

변수 (Cont'd)

var랑 val를 왜 구분하고? 그리고 언제 주로 사용하나여?

상수 val 기본적으로는 변경할 수 없는 값을 임시로 저장하기 위해
코틀린에서는 사라진 자바의 final 을 대응합니다
주로 절대로 바뀌면 안되는 수식, 또는 변수에 사용됨

```
val a  : Int = 10
var b : Int = 9
var c = 99 // 뒤에 자료형을 쓰지 않아도 된다.
var name : String = "Tom"
a = 100 //값을 변경하려해서 오류가 나올다.
```

If)만약 값이 바뀐다면 컴파일 에러가 나옵니다:)

3. String Template

```
//3. Strimg Template
val name = "Tom"
val lastName = "Hong"
println("my name is ${name + lastName}i'm 23")
println("this is \$1000")
var phone = "010-5011-7265-전화하면-밥사드려여"
var split_test = phone.split( ...delimiters: "-")
println(split_test)
println(split_test.joinToString ( separator: "-" ))
// tip. 주석 작성은 드래그 후에 ctrl + /
// 그외에 파이썬에서도 사용가능한 replace, substring 등이 있습니다.
```

```
my name is TomHongi'm 23
this is $1000
[010, 5011, 7265, 전화하면, 밥사드려여]
010-5011-7265-전화하면-밥사드려여

Process finished with exit code 0
```

4. 조건식

```
fun checkNum(score  : Int) {
    when (score) {
        0-> println("this isn 0")
        1,2 -> println("this is 1 or 2")
        0-> println("this isn 0")
    }

    var b : Int = when(score){
        1 -> 1
        2 -> 2
        else -> 3 //반드시 else가 필요하다 b가 score가 아닐수도 있기 때문에
    }

    println("b : ${b}")

    when(score){
        in 90..100-> println("You are genius")
        in 0..80 -> println("not bad")
        else -> println("error")
    }
}
```

```
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
this is 1 or 2
b : 1
not bad

Process finished with exit code 0
```

4. 조건식을 간단하게

```
// 4. 조건식
fun maxBy(a : Int , b : Int) : Int {
    if(a > b){
        return a
    }else {
        return b
    }
}

//하지만 실전에서는 더 짧게
fun maxBy2(a : Int , b : Int) = if(a>b) a else b

fun scoreTest(score : Int){
    var grade : Char
    if(score >= 80.0 && score<=89.9) grade = 'B'

    if(score in 80.0 .. 89.9) grade = 'B'
    // in을 이용해서 더 짧게 가능
}
```

Two comparisons should be converted to a range check

Convert to a range check Alt+Shift+Enter More actions... Alt+Enter

5. Array and List

```
//Array
//list  1. list(읽기전용) 2, Mutablelist

fun array(){
    val array :Array<Int> = arrayOf(1,2,3)
    val list :List<Int> =  listOf(1,2,3)

    val array2 :Array<Any> = arrayOf(1,"d",3)
    val list2 :List<Any> =  listOf(1,"어질어질하다 그쵸?",0)

    array[0] = 3

    //list[0] = 3 리스트 원소 변경은 불가능
    //리스트는 어레이랑은 다르게 인터페이스!
    var result :Int = list.get(0)
    //가져올 수는 있지만 직접적인 변경은 어렵다.
    var arrayList : ArrayList<Int> = arrayListOf<Int>()
    val arrayList2 : ArrayList<Int> = arrayListOf<Int>()
    //val의 참조값은 변하지 않는다.
    arrayList.add(10)
    arrayList.add(20)

}
```

List에는 2가지가 있다
일반 list와 mutable
list가 있다!

List는 인터페이스이다!

6. For / while

```
fun forAndWhile(){
    val students = arrayListOf("나영이","나정이","태구")
    for (name in students){
        println ("${name}")
    }

    var sum : Int = 0
    for (i:Int in 1..10) {
        sum += i
    }

    for (i:Int in 1..10 step 2) {
        // 1 3 5 7..
        sum += i
    }

    for (i:Int in 10 downTo 1) {
        // 10 9 8 7..
        sum += i
    }
```

```
for (i:Int in 10 downTo 1) {
    // 10 9 8 7..
    sum += i
}
print(sum)
var index = 0
while(index < 10){
    println("current index : ${index}")
    index++
}

for((index, name) in students.withIndex()){
    println("${index+1}번째 학생 : ${name}")
}
```

7. Nullable / NonNull

```
// 7. Nullable / NonNull

fun nullcheck(){
    //NPE : Null Pointer Exception
    //자바에서 컴파일 시점에서는 잡을 수 없고 런타임 시점에만 알 수 있다.
    //코틀린 :  "오케이! 그럼 내가 잡아줄게!"

    var name : String = "Joyee"
    //var nullName : String = null 에러발생!
    var nullName : String? = null
    //?를 타입 뒤에 붙여주면 Nullable 타입으로 변한다.
    //그냥 변수를 선언하면 기본적으로 NonNull타입

    var nameInUpperCase : String = name.toUpperCase()

    //var nullNameInUpperCase = nullName.toUpperCase() 오류발생
    // 다음 슬라이스에서 자세히

}
```

7. Nullable / NonNull (Cont'd)

```
var nullNameInUpperCase = nullName?.toUpperCase() // '?'가 추론을 해준다!
//만약에 null이 아니면 upperCase를 반환, null이면 바로 null을 반환한다.

//default값을 주고 싶을때는 어떻게 ??
// ?: 엘비스 연산자!

val lastName : String? = null
val lastName2 = "MinSeok"
val fullName  = name + (lastName?: "No lastName")
val fullName2  = name + (lastName2?: "No lastName")
    //lastName이 null인지 체크!
println(fullName)
println(fullName2)

 val elvisCheck : String = lastName ?: return //이렇게 함수 자체 리턴도 가능

 val elvisCheck2: String =
    lastName ?: throw NullPointerException()
// 현재 null 이기때문에 결과적으로 NullPointerException 예외가 발생!
```

```
Sample1Kt
"C:\Program Files\Android\Android Studio\
KimNo lastName
KimMinSeok

Process finished with exit code 0
```

```
fun ignoreNulls(str : String?){
    val NotNull : String = str
    //왜 오류가 나을까여??
}
```

str이 null일 수도 있기 때문에 오류가 발생!!!!!!!!!!!!!!!!!!!!

```
fun ignoreNulls(str : String?){
    val NotNull : String = str!!
    //왜 오류가 나을까여??
}
```

str은 진짜 죽었다가 깨어나도 null이
아니니까 그냥 쓸 수 있게 해줘!!

7. Nullable / NonNull (Cont'd)

```
fun ignoreNulls(str : String?){
    //val NotNull : String = str
    //str은 현재 nullable이기 때문에 오류가 발생합니다.
    val mNotNull : String = str!!
    //컴파일러야! str은 내가 null이 아니라고 내가 보장할게!!
    val upper : String  = mNotNull.toUpperCase()
    //!!는 진짜 확실하게 null이 아닌 값에만 사용해야한다. 되도록이면 지양하는게 좋음

    val email : String? = "kei3824@naver.com"
    //이메일이 null이 아니면 {~}를 해라
    email?.let { it: String
        println("My email is ${email}")
    }
    //자신의 히시버 객체를 람다식 내무로 옮겨준다.
}
```

if (variable != null)

```
"C:\Program Files\Android\Android Studio\jre\
My email is kei3824@naver.com

Process finished with exit code 0
```

8. class

자바랑 무엇이 다른가?

```
class Car (var speed : Int = 0){
    constructor(speed : Int, Car_type : String) : this(speed){
        println("이 ${Car_type}의 최대 속력은 ${speed} 입니다.")
    }
    //java
    /*
    class Person {
    public Person(String name){
    }
    public Person(String name, int age){
    this(name);
    }
   */
}
```

8. class

```
public class HandSome09d {
    String pikachu;
    HandSome09d(String pikachu){
        this.pikachu = pikachu;
    }
}
```

자바의 경우 생성자로부터 받아온 변수를 해당 클래스에서 사용할 때 생성자 내에서 [this.클래스변수 = 매개변수] 같은 형식으로 받아올 수 있다. 하지만 코틀린은 생성자 영역이 없어서 이렇게 할 수가 없다.

두 파라미터의 name을 다르게 하면 상관없긴하다.
하지만 배우는 입장이니까!

```
class HandSome09b(pikachu : String){
    var pikachu : String = ""
    //....??
}
```

8. class

초기화 영역 init

자바에서는 생성자에서 바로 필드들을 초기화할 수 있다. 하지만 코틀린을 별도의 생성자 영역이 없기 때문에 init 영역에서 초기화해주어야 한다!

```
class HandSome09b(pikachu : String){
    var pikachu : String = ""
    init {
        this.pikachu = pikachu
    }
}
```

init 안에서는 자바처럼 [this.클래스변수= 매개변수]를 얼마든지 할 수 있다!!

그런데 이렇게 항상 init을 만들어줘야 할까?!

8. class

인자=변수 생성자

인자=변수 생성자가 무엇인지 설명하기 전에 우선 코드부터 보고 가자.

```
class HandSome09e(var pikachu: String, var num : Int){
    fun pikachuInfo(){
        println("$pikachu number is $num")
    }
}
```

그렇다! init을 쓰지 않아도 생성자의 인자를 바로 클래스 내에서 쓸 수 있는 것이다. 즉 생성자 인자임과 동시에 클래스 변수이기 때문에 init 영역의 초기화가 없어도 외부에서 받아온 값을 그대로 쓸 수 있다!

8. class

코틀린 주 생성자 만드는 법 정리!

```
// 1번
class User1 constructor(nickname: String){
    val nickname: String
    init {
        this.nickname = nickname
    }
}

// 2번
class User2(_nickname: String){
    val nickname = _nickname
}

// 3번
class User3(val nickname: String)
```

8. Constructor

음..근데 생성자를 꼭 클래스 이름 옆에 생성할 수 밖에 없을까? 그러면 여러 개 만들 때는 어떡하지??

```
class HandSome09f{
    constructor(){
        //여기가 생성자 영역
    }
}
```

8. Constructor

부수 생성자 constructor

constructor라는 키워드를 사용하면 된다 :)

```
class HandSome09f{
    var pikachu : String = ""
    constructor(pikachu: String){
        this.pikachu = pikachu
    }
}
```

constructor는 여러개 사용할 수 있다. 즉 기타 언어들과 마찬가지로 생성자 여러개 생성하는 것이 가능

8. Constructor

```
class HandSome09f(){
    var pikachu : String = ""
    constructor(pikachu: String){
        this.pikachu = pikachu
    }
}
```

무엇이 문제일까?!

```
class HandSome09f(){
    var pikachu : String = ""
    constructor(pikachu: String) : this(){
        this.pikachu = pikachu
    }
}
```

8. Constructor

```
class HandSome09f(){
    var pikachu : String = ""
    constructor(pikachu: String){
        this.pikachu = pikachu
    }
}
```

무엇이 문제일까?!

```
class HandSome09f(){
    var pikachu : String = ""
    constructor(pikachu: String) : this(){
        this.pikachu = pikachu
    }
}
```

8. Constructor

```
class HandSome09f(num : Int){
    var pikachu : String = ""
    constructor(pikachu: String) : this(num){
        this.pikachu = pikachu
    }
}
```

무엇이 문제일까?!

```
class HandSome09f(num : Int){
    var pikachu : String = ""
    constructor(num : Int, pikachu: String) : this(num){
        this.pikachu = pikachu
    }
}
```

8. Constructor

```
class HandSome09f(num : Int){
    var pikachu : String = ""
    constructor(num : Int, pikachu: String) : this(num){
        this.pikachu = pikachu
    }
    constructor(num : Int, pikachu: String, type : String) : this(num){
        this.pikachu = pikachu
    }
}
```

이렇게 여러개 만들 수도 있다!

8. Constructor

```
class HandSome09{
    constructor(name : String){

    }
    constructor(name : String, num : Int){

    }
    constructor(name : String, type : String){

    }
}
```

기본생성자가 없다는 것은 곧 constructor가 상속해야 할 생성자가 없다는 것이다. 즉 이런 클래스에서는 constructor를 매우 자유롭게 만들어줄 수 있다!

8. Constructor

```
class HandSome09e(var pikachu: String, var num : Int){
    fun pikachuInfo(){
        println("$pikachu number is $num")
    }
}
```

```
class HandSome09{
    constructor(var name : String){

    }
    constructor(var name : String, val num : Int){

    }
    constructor(var name : String, val type : String){

    }
}
```

constructor사용에 있어 주의사항이 있다. constructor로 생성한 생성자는 기본생성자가 아니다. 그! 래! 서!

[인자 = 변수 생성자]

같은 형태로 쓸 수는 없다

8. class

```
//코틀린은 파일 이름과 class이름과 같지 않아도 된다.
//여러 클래스를 한 파일안에 넣을 수 있다.

//자바와 생성자를 만드는법은 조금 다르다.
class Human constructor(age : Int) {
    //주 생성자
    init{ //init은 주 생성자의 일부이다.
        println("New human has been born!!")
    }
    val name = "Alvaro Morte"
    val age = age
    fun eatingCake() {
        println("This is YUMMMMMMYYYYY~~~~")
    }
}
//age가 형광색으로 나오는 이유는 축약해서 아래와 같이 쓰라고
//추천하기 때문이다.
class Human2 constructor(val age : Int = 99){}

fun main(){
    val human = Human( age: 46)
    val human2 = Human2( age: 46)
```

질문과 답변

감사합니다