세상의 속도를
따라잡고 싶다면

Do it!

깡샘의 안드로이드 앱 프로그래밍 with 코틀린

이지스퍼블리싱(주)

03

코틀린 시작하기

이지스퍼블리싱(주)

03-1 코틀린 언어 소개

코틀린의 등장 배경

- 코틀린은 젯브레인스(JetBrains)에서 오픈소스 그룹을 만들어 개발한 프로그래밍 언어
- 2017년 구글에서 안드로이드 공식 언어로 지정
- JVM에 기반을 둔 언어

- 코틀린의 이점
  - 최신 언어 기법을 이용하면 훨씬 간결한 구문으로 프로그램을 작성
  - 코틀린은 널 안전성null safety을 지원하는 언어
  - 코틀린은 자바와 100% 호환합니다.
  - 코루틴coroutines이라는 기법을 이용하면 비동기 프로그래밍을 간소화할 수 있습니다.

03-1 코틀린 언어 소개

코틀린 파일 구성

• 코틀린 파일 구성(User.kt)

```
package com.example.test3                          패키지

import java.text.SimpleDateFormat                  임포트
import java.util.*

var data = 10                                      변수

fun formatData(date: Date): String {               함수
    val sdformat = SimpleDateFormat("yyyy-mm-dd")
    return sdformat.format(date)
}

class User {                                       클래스
    var name = "hello"

    fun sayHello() {
        println("name : $name")
    }
}
```

03-1 코틀린 언어 소개

• 패키지 경로를 같게 작성한 예(Test.kt)

```
package com.example.test3

import java.util.*

fun main() {
    data = 20
    formatDate(Date())
    User().sayHello()
}
```

03-1 코틀린 언어 소개

• 패키지 경로를 다르게 작성한 예

```
package ch3

import com.example.test3.User
import com.example.test3.data
import com.example.test3.formatDate
import java.util.*

fun main() {
    data = 20
    formatDate(Date())
    User().sayHello()
}
```

패키지 경로가 다르므로
import 구문으로 불러 와야 함

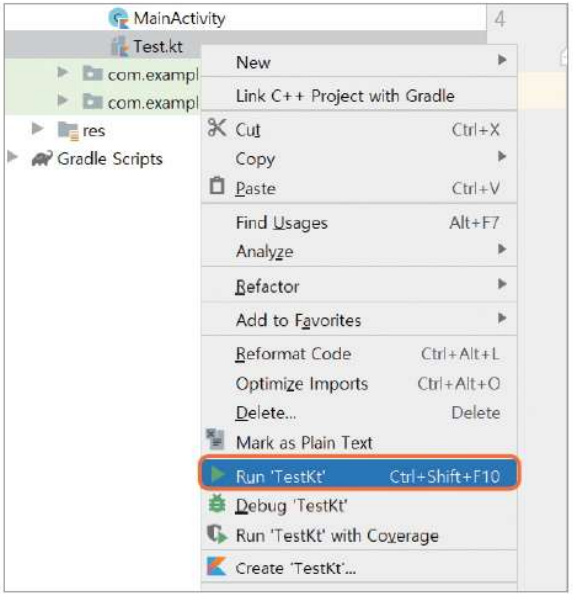
03-1 코틀린 언어 소개

코틀린 소스를 테스트하는 방법

- 테스트할 코틀린 소스 파일에는 main() 함수가 있어야 하며, 실행하면 main() 함수가 자동으로 실행됐다가 끝나면 프로그램이 종료됩니다.

• 코틀린 소스 파일 • Test.kt

```
fun main() {
    println("hello world")
}
```



03-2 변수와 함수

변수 선언하기

- 변수는 val, var 키워드로 선언
- val은 value의 줄임말로 초깃값이 할당되면 바꿀 수 없는 변수를 선언
- var는 variable의 줄임말로 초깃값이 할당된 후에도 값을 바꿀 수 있는 변수를 선언

• 변수 선언 형식

```
val(혹은 var) 변수명: 타입 = 값
```

• val과 var 변수의 차이

```
val data1 = 10
var data2 = 10

fun main() {
    data1 = 20    // 오류!
    data2 = 20    // 성공!
}
```

03-2 변수와 함수

- 타입 지정과 타입 추론
  - 변수명 뒤에는 콜론(:)을 추가해 타입을 명시
  - 대입하는 값에 따라 타입을 유추(타입 추론)할 수 있을 때는 생략

• 변수에 타입 지정과 타입 추론

```
val data1: Int = 10
val data2 = 10
```

03-2 변수와 함수

- 초깃값 할당
  - 최상위에 선언한 변수나 클래스의 멤버 변수는 선언과 동시에 초깃값을 할당해야 하며, 함수 내부에 선언한 변수는 선언과 동시에 초깃값을 할당하지 않아도 됩니다.

• 초깃값 할당

```
val data1: Int    // 오류!
val data2 = 10    // 성공!

fun someFun() {
    val data3: Int
    println("data3 : $data3")    // 오류!
    data3 = 10
    println("Data3 : $data3")    // 성공!
}

class User {
    val data4: Int          // 오류!
    val data5: Int = 10     // 성공!
}
```

03-2 변수와 함수

- 초기화 미루기
  - lateinit 키워드는 이후에 초깃값을 할당할 것임을 명시적으로 선언
  - lateinit은 var 키워드로 선언한 변수에만 사용할 수 있습니다.
  - Int, Long, Short, Double, Float, Boolean, Byte 타입에는 사용할 수 없습니다.

• 초기화 미루기 1 - lateinit 키워드

```
lateinit var data1: Int       // 오류!
lateinit val data2: String    // 오류!
lateinit var data3: String    // 성공!
```

03-2 변수와 함수

- by lazy { } 형식으로 선언하며, 소스에서 변수가 최초로 이용되는 순간 중괄호로 묶은 부분이 자동으로 실행되어 그 결괏값이 변수의 초깃값으로 할당

• 초기화 미루기 2 - lazy 키워드

```
val data4: Int by lazy {
    println("in lazy......")
    10
}

fun main() {
    println("in main......")
    println(data4 + 10)
    println(data4 + 10)
}
```

▶ 실행 결과

```
in main......
in lazy......
20
20
```

03-2 변수와 함수

- 데이터 타입
  - 코틀린의 모든 변수는 객체

• Int 타입에 null 대입과 메서드 이용

```
fun someFun() {
    var data1: Int = 10
    var data2: Int? = null      // null 대입 가능

    data1 = data1 + 10
    data1 = data1.plus(10)      // 객체의 메서드 이용 가능
}
```

03-2 변수와 함수

- Int, Short, Long, Double, Float, Byte, Boolean — 기초 타입 객체
- Char, String — 문자와 문자열
  - String 타입의 데이터는 문자열을 큰따옴표(")나 삼중 따옴표(""")로 감싸서 표현

• 문자 표현

```
val a: Char = 'a'
if (a == 1) {    // 오류!
}
```

• 기초 데이터 타입

```
val a1: Byte = 0b00001011

val a2: Int = 123
val a3: Short = 123
val a4: Long = 10L
val a5: Double = 10.0
val a6: Float = 10.0f

val a7: Boolean = true
```

• 문자열 표현 - 큰따옴표와 삼중 따옴표의 차이

```
fun main() {
    val str1 = "Hello \n World"
    val str2 = """
Hello
World
    """
    println("str1 : $str1")
    println("str2 : $str2")
}
```

▶ 실행 결과

```
str1 : Hello
 World
str2 :
Hello
World
```

03-2 변수와 함수

- 문자열 템플릿 : String 타입의 데이터에 변숫값이나 어떤 연산식의 결괏값을 포함해야 할 때는 $ 기호를 이용

• 문자열 템플릿 사용 예

```
fun main() {
    fun sum(no: Int): Int {
        var sum = 0
        for (i in 1..no) {
            sum += i
        }
        return sum
    }

    val name: String = "kkang"
    println("name : $name, sum : ${sum(10)}, plus : ${10 + 20}")
}
```

▶ 실행 결과

```
name : kkang, sum : 55, plus : 30
```

03-2 변수와 함수

- Any — 모든 타입 가능
- Unit — 반환문이 없는 함수

• Any 타입 사용 예

```
val data1: Any = 10
val data2: Any = "hello"

class User
val data3: Any = User()
```

• Unit 타입 사용 예

```
val data1: Unit = Unit
```

• Unit 타입 사용 예 - 반환문이 없는 함수

```
fun some(): Unit {
    println(10 + 20)
}
```

• 반환 타입을 생략한 예

```
fun some() {
    println(10 + 20)
}
```

03-2 변수와 함수

- Nothing — null이나 예외를 반환하는 함수
- 널 허용과 불허용

• Nothing 사용 예

```
val data1: Nothing? = null
```

• null 반환 함수와 예외를 던지는 함수

```
fun some1(): Nothing? {
    return null
}
fun some2(): Nothing {
    throw Exception()
}
```

• 널 허용과 불허용

```
var data1: Int = 10
data1 = null    // 오류!

var data2: Int? = 10
data2 = null    // 성공!
```

03-2 변수와 함수

함수 선언하기

- 함수를 선언하려면 fun이라는 키워드를 이용
- 반환 타입을 선언할 수 있으며 생략하면 자동으로 Unit 타입이 적용
- 함수의 매개변수에는 var나 val 키워드를 사용할 수 없으며 val이 자동으로 적용

• 함수 선언 형식

```
fun 함수명(매개변수명: 타입): 반환 타입 { ... }
```

• 반환 타입이 있는 함수 선언

```
fun some(data1: Int): Int {
    return data1 * 10
}
```

• 매개변숫값 변경 오류

```
fun some(data1: Int) {
    data1 = 20    // 오류!
}
```

03-2 변수와 함수

- 함수의 매개변수에는 기본값 선언 가능

• 기본값 활용

```
fun main() {
    fun some(data1: Int, data2: Int = 10): Int {
        return data1 * data2
    }
    println(some(10))
    println(some(10, 20))
}
```

▶ 실행 결과

```
100
200
```

03-2 변수와 함수

- 매개변수명을 지정하여 호출하는 것을 명명된 매개변수라고 하며, 이렇게 하면 함수 선언문의 매개변수 순서에 맞춰 호출하지 않아도 됩니다.

• 매개변수명 생략 - 매개변수 순서대로 할당

```
fun some(data1: Int, data2: Int): Int {
    return data1 * data2
}
println(some(10, 20))
```

• 매개변수명을 지정하여 호출

```
some(data2 = 20, data1 = 10)
```

03-2 변수와 함수

컬렉션 타입

- Array — 배열 표현
  - 배열은 Array 클래스로 표현
  - 배열의 데이터에 접근할 때는 대괄호([])를 이용해도 되고 set()이나 get() 함수를 이용할 수도 있습니다.

• 배열의 데이터에 접근하는 예

```
fun main() {
    val data1: Array<Int> = Array(3, { 0 })
    data1[0] = 10
    data1[1] = 20
    data1.set(2, 30)

    println(
        """
    array size : ${data1.size}
    array data : ${data1[0]}, ${data1[1]}, ${data1.get(2)}
    """
    )
}
```

배열에서 2번째 데이터를 30으로 설정

2번째 데이터 가져오기

▶ 실행 결과

```
array size : 3
array data : 10, 20, 30
```

03-2 변수와 함수

- 기초 타입의 배열
  - 기초 타입이라면 Array를 사용하지 않고 BooleanArray, ByteArray, CharArray, DoubleArray, FloatArray, IntArray, LongArray, ShortArray 클래스를 이용할 수도 있습니다.

• 기초 타입 배열 선언

```
val data1: IntArray = IntArray(3, { 0 })
val data2: BooleanArray = BooleanArray(3, { false })
```

  - arrayOf()라는 함수를 이용하면 배열을 선언할 때 값을 할당할 수도 있습니다.
  - 기초 타입을 대상으로 하는 booleanArrayOf(), byteArrayOf(), char ArrayOf(), doubleArrayOf(), floatArrayOf(), intArrayOf(), longArrayOf(), shortArrayOf() 함수를 제공

03-2 변수와 함수

• 기초 타입 arrayOf() 함수

```
val data1 = intArrayOf(10, 20, 30)
val data2 = booleanArrayOf(true, false, true)
```

• 배열 선언과 동시에 값 할당

```
fun main() {
    val data1 = arrayOf<Int>(10, 20, 30)
    println(
        """
    array size : ${data1.size}
    array data : ${data1[0]}, ${data1[1]}, ${data1.get(2)}
    """
    )
}
```

크기가 3인 Int 배열을 선언하고 10, 20, 30으로 할당

▶ 실행 결과

```
array size : 3
array data : 10, 20, 30
```

03-2 변수와 함수

- List, Set, Map
  - List: 순서가 있는 데이터 집합으로 데이터의 중복을 허용합니다.
  - Set: 순서가 없으며 데이터의 중복을 허용하지 않습니다.
  - Map: 키와 값으로 이루어진 데이터 집합으로 순서가 없으며 키의 중복은 허용하지 않습니다.

  - Collection 타입의 클래스는 가변 클래스와 불변 클래스로 나뉩니다.
  - 불변 클래스는 초기에 데이터를 대입하면 더 이상 변경할 수 없는 타입입니다.
  - 가변 클래스는 초깃값을 대입한 이후에도 데이터를 추가하거나 변경할 수 있습니다.

구분	타입	함수	특징
List	List	listOf()	불변
	MutableList	mutableListOf()	가변
Set	Set	setOf()	불변
	MutableSet	mutableSetOf()	가변
Map	Map	mapOf()	불변
	MutableMap	mutableMapOf()	가변

03-2 변수와 함수

• 리스트 사용 예

```
fun main() {
    var list = listOf<Int>(10, 20, 30)
    println(
        """
    list size : ${list.size}
    list data : ${list[0]}, ${list.get(1)}, ${list.get(2)}
    """
    )
}
```

▶ 실행 결과

```
list size : 3
list data : 10, 20, 30
```

• 가변 리스트 사용 예

```
fun main() {
    var mutableList = mutableListOf<Int>(10, 20, 30)
    mutableList.add(3, 40)
    mutableList.set(0, 50)
    println(
        """
    list size : ${mutableList.size}
    list data : ${mutableList[0]}, ${mutableList.get(1)},
                ${mutableList.get(2)}, ${mutableList.get(3)}
    """
    )
}
```

▶ 실행 결과

```
list size : 4
list data : 50, 20, 30, 40
```

03-2 변수와 함수

- Map 객체는 키와 값으로 이루어진 데이터의 집합
- Map 객체의 키와 값은 Pair 객체를 이용할 수도 있고 '키 to 값' 형태로 이용할 수도 있습니다.

• 집합 사용 예

```
fun main() {
    var map = mapOf<String, String>(Pair("one", "hello"), "two" to "world")
    println(
        """
    map size : ${map.size}
    map data : ${map.get("one")}, ${map.get("two")}
    """
    )
}
```

▶실행 결과

```
map size : 2
map data : hello, world
```

03-3 조건문과 반복문

조건문 if~else와 표현식

• if~else 문 사용 예

```
fun main() {
    var data = 10
    if (data > 0) {
        println("data > 0")
    } else {
        println("data <= 0")
    }
}
```

▶실행 결과

```
data > 0
```

• 조건을 여러 개 나열한 예

```
fun main() {
    var data = 10
    if (data > 10) {
        println("data > 10")
    } else if (data > 0 && data <= 10) {
        println("data > 0 && data <= 10")
    } else {
        println("data <= 0")
    }
}
```

▶실행 결과

```
data > 0 && data <= 10
```

03-3 조건문과 반복문

- 코틀린에서 if~else는 표현식으로도 사용할 수 있습니다.
- 표현식이란 결괏값을 반환하는 계산식을 말합니다.

• 표현식으로 사용 예

```
fun main() {
    var data = 10
    val result = if (data > 0) {
        println("data > 0")
        true
    } else {
        println("data <= 0")
        false
    }
    println(result)
}
```

true — 참일 때 반환값

false — 거짓일 때 반환값

▶ 실행 결과

```
data > 0
true
```

03-3 조건문과 반복문

조건문 when

• when 문 사용 예

```
fun main() {
    var data = 10
    when (data) {
        10 -> println("data is 10")
        20 -> println("data is 20")
        else -> {
            println("data is not valid data")
        }
    }
}
```

▶ 실행 결과

```
data is 10
```

03-3 조건문과 반복문

- when 문의 조건으로 정수가 아닌 다른 타입의 데이터를 지정할 수도 있습니다.

• 문자열 타입을 조건으로 사용

```
fun main() {
    var data = "hello"
    when (data) {
        "hello" -> println("data is hello")
        "world" -> println("data is world")
        else -> {
            println("data is not valid data")
        }
    }
}
```

▶ 실행 결과

```
data is hello
```

03-3 조건문과 반복문

- when 문에서는 조건을 데이터 타입, 범위 등으로 다양하게 명시할 수 있습니다.
- is는 타입을 확인하는 연산자이며 in은 범위 지정 연산자입니다.

• 다양한 유형의 조건 제시

```
fun main() {
    var data: Any = 10
    when (data) {
        is String -> println("data is String")      // data가 문자열 타입이면
        20, 30 -> println("data is 20 or 30")       // data가 20 또는 30이면
        in 1..10 -> println("data is 1..10")        // data가 1~10의 값이면
        else -> println("data is not valid")
    }
}
```

▶ 실행 결과

```
data is 1..10
```

03-3 조건문과 반복문

- when은 if 문과 마찬가지로 표현식으로도 사용할 수 있습니다.

• when 문을 표현식으로 사용

```
fun main() {
    var data = 10
    val result = when {
        data <= 0 -> "data is <= 0"
        data > 100 -> "data is > 100"
        else -> "data is valid"
    }
    println(result)
}
```

03-3 조건문과 반복문

반복문 for와 while

- for 문은 제어 변숫값을 증감하면서 특정 조건이 참일 때까지 구문을 반복해서 실행합니다.
- or 문의 조건에는 주로 범위 연산자인 in을 사용합니다.
- for (i in 1..10) { ... } → 1부터 10까지 1씩 증가
- for (i in 1 until 10) { ... } → 1부터 9까지 1씩 증가(10은 미포함)
- for (i in 2..10 step 2) { ... } → 2부터 10까지 2씩 증가
- for (i in 10 downTo 1) { ... } → 10부터 1까지 1씩 감소

• for~in 반복문

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

▶ 실행 결과

```
55
```

03-3 조건문과 반복문

- 컬렉션 타입의 데이터 개수만큼 반복
- indices는 컬렉션 타입의 인덱스값을 의미
- 인덱스와 실제 데이터를 함께 가져오려면 withIndex() 함수를 이용

• 반복 조건에 컬렉션 타입 활용

```
fun main() {
    var data = arrayOf<Int>(10, 20, 30)
    for (i in data.indices) {
        print(data[i])
        if (i !== data.size - 1) print(",")
    }
}
```

▶ 실행 결과

```
10,20,30
```

• 인덱스와 데이터를 가져오는 withIndex() 함수

```
fun main() {
    var data = arrayOf<Int>(10, 20, 30)
    for ((index, value) in data.withIndex()){
        print(value)
        if (index !== data.size - 1) print(",")
    }
}
```

▶ 실행 결과

```
10,20,30
```

03-3 조건문과 반복문

- while 문은 조건이 참이면 중괄호 {}로 지정한 영역을 반복해서 실행

• while 반복문

```
fun main(args: Array<String>) {
    var x = 0
    var sum1 = 0
    while (x < 10) {
        sum1 += ++x
    }
    println(sum1)
}
```

▶ 실행 결과

```
55
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare