

Kotlin 기초 01



목차

 코틀린 개요

 코틀린 프로그램의 기본 구조

 변수, 배열 및 Collection

 조건문의 활용

 반복문의 활용

*Test t = new ...
t.sell();*
객체 생성

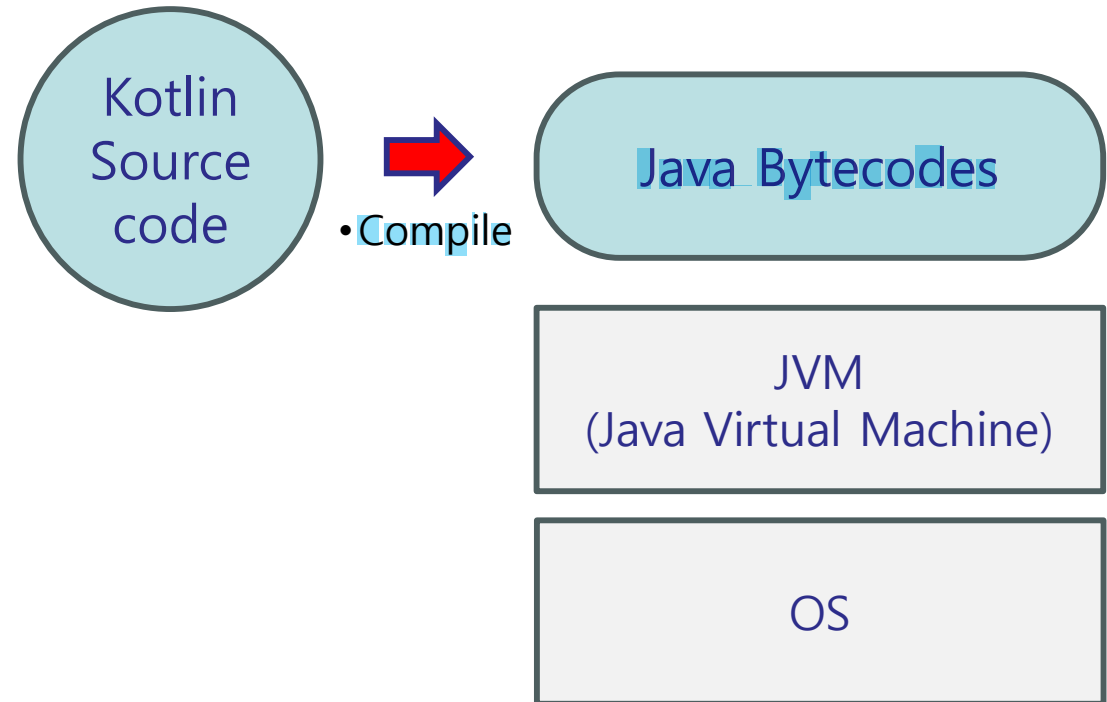
JetBrains 에서 개발한 프로그래밍 언어

- ◆ 2017년 안드로이드 개발 언어로 공식 지정
- ◆ JVM 기반 언어 → Java 와 호환 (Java Bytecode 로 변환)

특징

- ◆ 간결한 프로그램 구문
- ◆ Null Safety 지원
- ◆ Java 와 호환
- ◆ 비동기 프로그래밍 용이
 - Coroutine

→ 동시에 여러가지 작업 가능

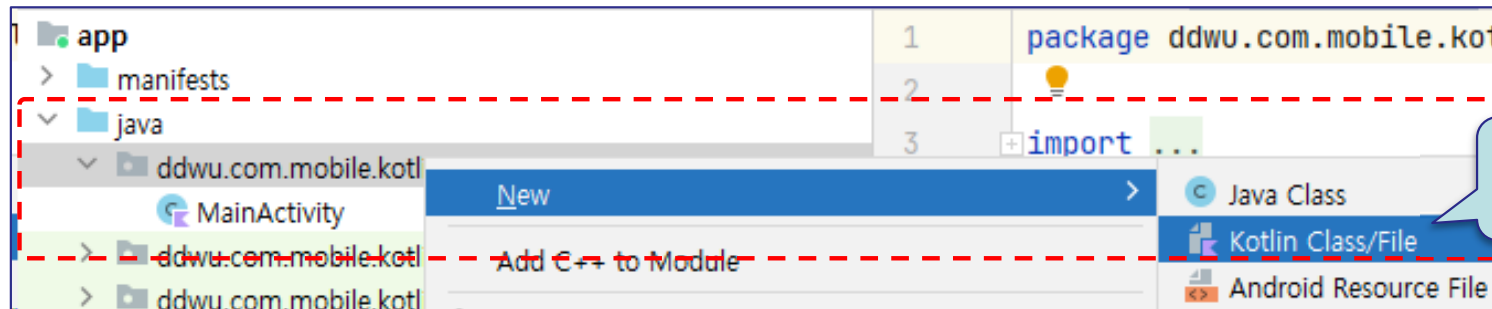


Kotlin 코딩 준비

Android Studio 에서 테스트 가능

소스코드 파일(*.kt) 생성

◆ Android 프로젝트(Kotlin test) 생성 후 진행



Kotlin Program

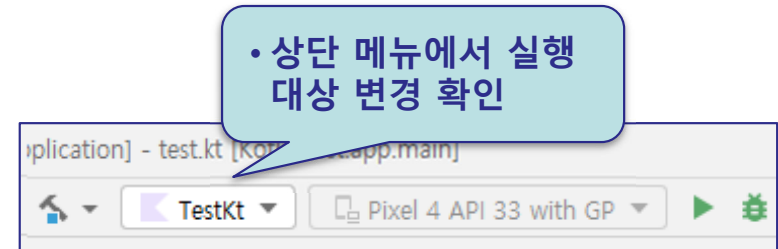
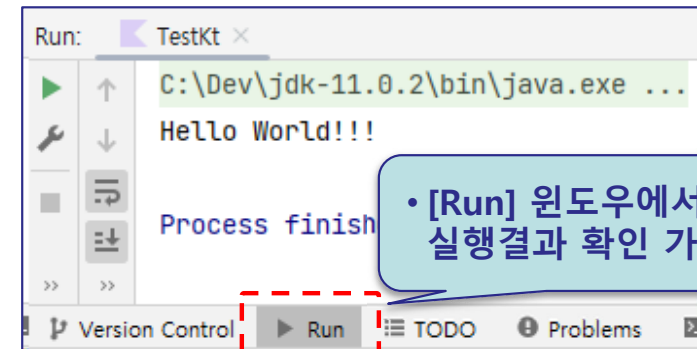
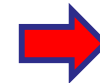
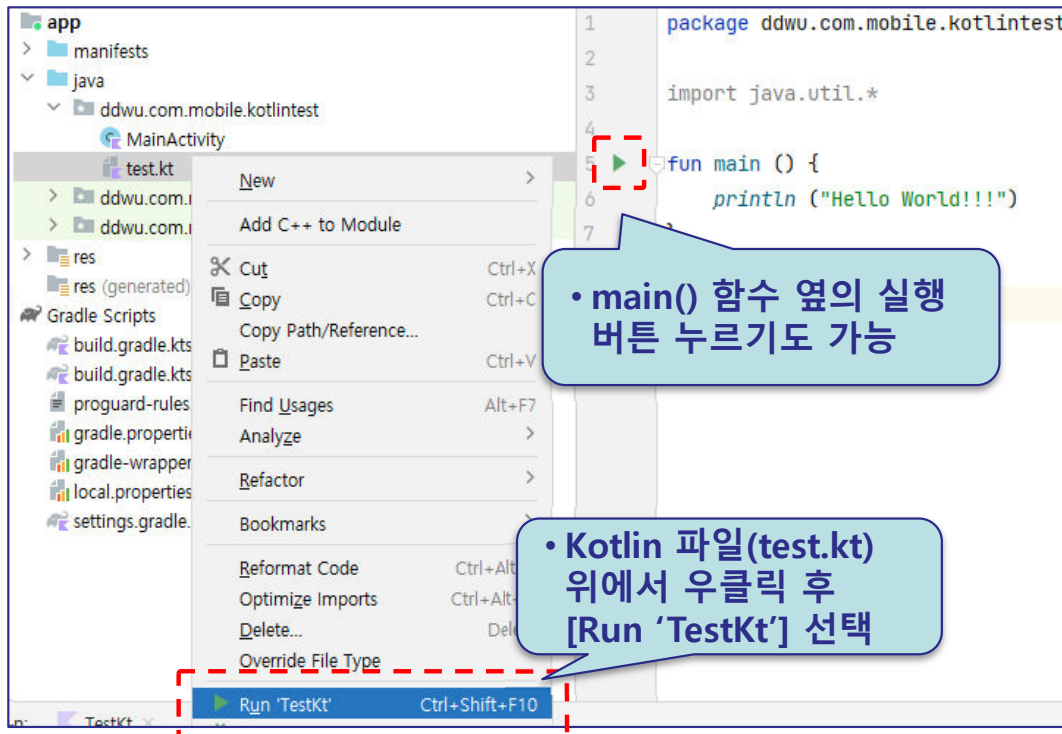
기본 구조

```
package ddwu.com.mobile.kotlintest

import java.util.*

fun main () {
    println ("Hello World!!!")
}
```

실행



변수의 사용 01

☞ 변수의 선언

`val (또는 var) 변수명 [: 자료형] = 값`

- ◆ `val`: 변수 선언 시의 초기화 값으로 값 고정
 - 함수의 지역변수로 선언 시에는 생성 이후에 지정하는 것도 가능
- ◆ `var`: 변수 선언 후 필요에 따라 값 변경
- ◆ 자료형을 생략했을 경우 초기화 값의 자료형으로 결정

`val myValue = 10 // Integer 형`

```
val data01 : Int = 1234
var data02 = 5
```

• 전역변수 또는 클래스의 멤버 변수는 반드시 초기화 수행

```
fun main () {
    val data01: Int
    var data03: String
    data01 = 4321
    data03 = "SomSom"
    println ("Hello World!!! $data01 $data03")
}
```

• 지역변수로 선언할 경우 초기화를 수행하지 않는 것도 가능

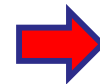
변수의 사용 02

초기화 지연

- ◆ 필요에 따라 변수의 선언 이후에 초기화를 진행하여야 할 경우 적용
- ◆ `lateinit` var 변수 : 자료형
 - var 로 선언한 변수의 초기화 지연
 - 기본 자료형에는 적용 불가
- ◆ val 변수 [: 자료형] `by lazy { }`
 - val 변수가 최초로 사용되는 순간 `by lazy` 블록이 실행되며, 초기 작업 및 초기화 수행

```
lateinit var data01 : String
val data02 : Int by lazy {
    println ("val init")
    1234 ^lazy
}
fun main () {
    data01 = "Mobile"
    println ("data01: $data01 data02: $data02")
}
```

초기화 지연



```
C:\Dev\jdk-11.0.2\bin\java.exe ...
val init
data01: Mobile data02: 1234
```

변수의 자료형 01

객체 자료형

- ◆ 모든 자료형은 객체 (Kotlin Int == Java Integer)
- ◆ 기본: Int, Short, Long, Double, Byte, Boolean, Char
- ◆ 기본적으로 null 대입 불가
 - non-null type
 - 변수 선언 시 ? 를 기록하여 null 대입 가능

```
var data01 : Int = null (x)
var data02 : Int? = null (o)
```

문자열 String

- ◆ "일반문자열" 또는 """"띄어쓰기/탭/줄바꿈이 그대로 표현"""

Any, Unit, Nothing 자료형

- ◆ Any: 모든 객체의 최상위 타입 (Java Object)
- ◆ Unit: 반환타입이 없음을 표시 (Java void와 유사)
- ◆ Nothing: null 또는 Exception 반환 표시 (null 의 경우 ? 사용)

Unit 생략

```
var data01 : Int = 10      // 변수 선언
var data02 : Int? = null   // null 대입 가능
var data03 : Any = "20"   // Any 를 사용하여 문자열 저장

fun main() {
    var data04 = data01.toFloat() // 객체이므로 멤버함수 사용 가능
    println ("${data01}, ${data02}, ${data03} ${data04}")
}

fun myFunction01 () : Unit {    // 반환타입 없음
    println ("myFunction01")
}

fun myFunction02 () : Nothing? { // null 반환
    return null
}

fun myFunction03 () : Nothing { // Exception throw
    throw Exception()
}
```

• 현재 자료형을 Float 로 변환

• null을 반환하므로 ? 사용

함수의 사용

함수 선언

```
fun 함수명 ( 매개변수명: 타입 [, ...] ) : 반환타입 { 함수본체 }
```

- ◆ [반환타입]이 없을 경우 Unit 사용 또는 생략
- ◆ 매개변수는 val/var 키워드를 사용하지 않으며 **val**로 취급

```
fun myPower01 (value : Int) : Int { // 함수 기본
    return value * value
}

fun myAdd (value : Int) : Unit { // 매개변수 변경 금지
    // value = 10 → 오류
}

fun myDefault (value : Int = 1) : Int { // 매개변수 디폴트값
    return value * value
}

fun mySelection (value01: Int, value02: Int) : Unit { // 매개변수 지정
    print ("add == ${value01 + value02}")
}

fun main() {
    println("${myDefault()} ${myDefault(10)}")
    println("${mySelection(1, 2)} ${mySelection(value02=1, value01=2)}")
}
```

↓
정식받은 값 그대로 사용

• \${ } 를 사용하여 계산식 등의 결과를 문자열로 출력

배열과 Collection 01

배열 Array

```
val myArray : Array<Int> = Array(3, {0})

fun main() {
    println (" ${myArray[0]}, ${myArray.get(0)}")
    myArray[1] = 10
    myArray.set(2, 20)
    println (" ${myArray[1]}, ${myArray.get(2)}")
}
```

Size ↑ 초기값 ↑

- 3개의 값을 저장할 수 있는 배열 선언
- 초기화 값은 0



```
C:\Dev\jdk-11.0.2\bin\java.exe
1, 1 0, 0
10, 20
```

- get/set 사용 가능
- [] 사용 권장

기본 자료형 별 배열 선언

- ◆ ~of: 배열 생성 함수
- ◆ Int 배열의 예

```
val intArray01 : Array<Int> = Array(3, {0})
val intArray02 = IntArray(3, {0})
val intArray03 = arrayOf<Int>(1, 2, 3)
val intArray04 = intArrayOf(4, 5, 6)

fun main() {
    println ("array01 size: ${ intArray01.size }")
}
```

배열과 Collection 02

기본 Collection : List, Set, Map

- ◆ Collection : 저장공간 크기의 실행 중 변경이 가능
 - List: 순서 있는 컬렉션, 값 중복 허용
 - Set: 순서 없는 컬렉션, 값 중복 불가
 - Map: key와 value의 집합으로 구성, key로 value 접근
- ◆ 가변타입과 불변타입(mutable~)으로 구분

```
val myList01 : List<Int> = List(3, {0})
val myList02 = listOf(1, 2, 3)
val myMuList01 : MutableList<Int> = MutableList(3, {0})
val myMuList02 = mutableListof<Int>(1, 2, 3)

fun main() {
    //    myList01[0] = 10    // 오류 발생
    myMuList01[0] = 10      • 새로운 항목 추가
    myMuList01.add(4)
    myMuList02.set(0, 10)   • 기존 항목 변경
    println ("${myMuList01}  ${myMuList02}")
}
```



```
C:\Dev\jdk-11.0.2\bin\java.exe
[10, 0, 0, 4] [10, 2, 3]
```

조건문의 활용 01

if, if-else

- ◆ Java 의 if문과 동일
- ◆ if-else 문 블록의 가장 마지막 값을 결과로 전달(표현식) 가능

```
fun main() {
    val value = 10
    if (value > 10) {
        println ("Up")
    } else {
        println ("Down")
    }
    val result : (Boolean) = if (value > 5) {
        true // 표현식 - 값 반환
    } else {
        false
    }
    println ("result: $result")
}
```

• 블록의 마지막에 쓰여진 값으로 결과 표현



```
C:\Dev\jdk-11.0.2\bin\java.exe
Down
result: true
```

when

◆Java의 switch-case 문 역할

```
fun main() {  
    val data : Any = 10  
    val result = when (data) { // 조건에 String도 사용 가능  
        → is Int -> {  
            println ("Integer!")  
            true  
        }  
        → is String -> false  
        → 10 -> true  
        → in 5..20 -> true  
        → else -> false  
    }  
    println ("result : $result")  
}
```

• 표현식 사용 가능

• 자료형 확인

is

A..B

• 범위 확인 가능
• $5 \leq \text{data} \leq 20$



```
C:\Dev\jdk-11.0.2\bin\java.exe  
Integer!  
result : true
```

반복문의 활용 01

while

- ◆반복 횟수가 정해지지 않은 경우에 적합
- ◆Java의 while과 동일

```
fun main() {
    while (true) {
        print("Input a number (0 for exit): ")
        val inputData = readLine()?.toInt()
        if (inputData != 0) continue
        else break
    }
}
```

입력 → 그냥 Enter만 쳤을 때

• 키보드에서 값을 입력하여 Int로 변환



```
C:\Dev\jdk-11.0.2\bin\java.exe
Input a number (0 for exit): 1
Input a number (0 for exit): 2
Input a number (0 for exit): 0
```

for

- ◆ 특정 횟수만큼 반복을 수행
- ◆ 범위 연산자 in 사용

```
fun main() {  
    for (i in 1..10) print( "$i ")      // 1 <= i <= 10  
    println()  
    for (i in 1 until 10) print( "$i ")  // 1 <= i < 10  
    println()  
    for (i in 1..10 step 3) print( "$i ") // 1부터 3씩 증가  
    println()  
    for (i in 10 downTo 2) print( "$i ") // 10 >= i >= 2  
}
```



```
C:\Dev\jdk-11.0.2\bin\java.exe  
1 2 3 4 5 6 7 8 9 10  
1 2 3 4 5 6 7 8 9  
1 4 7 10  
10 9 8 7 6 5 4 3 2
```


for 와 Collection

```
fun main() {  
    val data = intArrayOf(1, 2, 3, 4, 5)  
  
    for (value in data) print("$value ")  
    println()  
    for (index in data.indices) {  
        println("index: $index value: ${data[index]}")  
    }  
    for ((index, value) in data.withIndex()) {  
        println("[ $index ] : $value")  
    }  
}
```

• indices: Collection의
index 집합

• withIndex(): index와
value를 함께 반환

```
C:\Dev\jdk-11.0.2\bin\java.exe  
12345  
index: 0 value: 1  
index: 1 value: 2  
index: 2 value: 3  
index: 3 value: 4  
index: 4 value: 5  
[0] : 1  
[1] : 2  
[2] : 3  
[3] : 4  
[4] : 5
```

키보드 입력 & 실습 00

키보드 입력

```

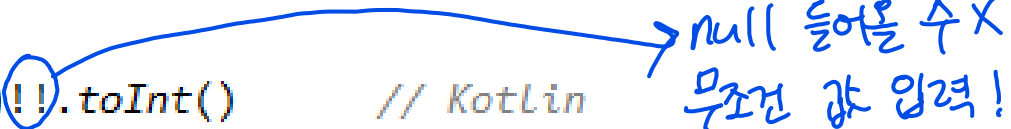
val keyboard : Scanner = Scanner(System.`in`)           // Java
val javaValue = keyboard.nextInt()

val kotlinValue = readLine()!!.toInt()                 // Kotlin

val arrayValue = Array<Int>(size: 3) { readLine()!!.toInt() } // Array init.

println("자바형식: $javaValue   코틀린형식: $kotlinValue 배열[2]: ${arrayValue[2]}")

```


 null 들어올 수 X
무조건 값 입력!

\$ 변수 \$ { 배열 }

실습 00

- ◆ 5개의 숫자를 키보드에서 입력하여 총합과 평균을 출력하는 프로그램 구현

실습 01

☞ 키와 몸무게를 입력하여 BMI 계산 후 비만 정도를 출력하는 프로그램을 구현

- ◆ BMI 계산식 : 체중 (kg) / 키 (m)²
키를 cm 단위로 입력할 경우 m 단위로 변환하도록 구현

BMI	18.5	23	25
저체중	정상	과체중	비만

- ◆ 입력 예
 - 키를 입력하시오. (cm 단위): # 키 입력
 - 몸무게를 입력하시오. (kg 단위) : # 몸무게 입력
- ◆ 출력 예
 - 키 XXX cm, 몸무게 XX kg의 BMI 지수는 XX 이며 XX 입니다.
- ◆ 참고 사이트 :
<https://health.seoulmc.or.kr/healthCareInfo/myBMIPopup.do>

실습 02

☞ 다음 조건에 해당하는 로또 생성기를 구현

- ◆ 1 ~ 45 까지의 수 중 서로 다른 숫자 6개 생성

☞ Hint

- ◆ Random 함수 사용

- import java.util.Random
- 0~44 사이의 랜덤값을 사용할 때

```
val random = Random()
var num = random.nextInt( bound: 45)
```

- ◆ Collection 객체에 값 저장 → 값을 변경할 수 있는 객체로 생성 → MutableSet