

# 소프트웨어설계및실험

2024 Spring

코틀린 프로그래밍 (기초)

# 이론

## ◆ Kotlin 기초

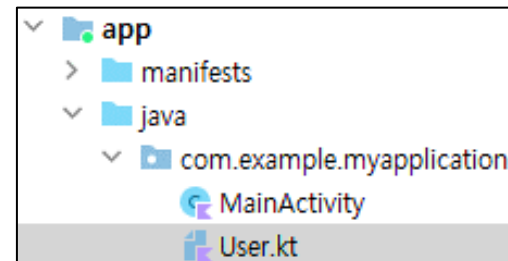
- Kotlin 개요
- 코드 작성 및 실행
- 변수 선언하기
- 함수 선언하기
- 조건문, 반복문
- 형 변환
- 흐름제어
- enum class

## • Kotlin 이란?

- JVM(Java Virtual Machine)에서 동작하는 프로그래밍 언어
- 2017년 구글에서 **Android 공식언어**로 지정
- 객체지향과 함수형 프로그래밍 스타일을 지원

## • Kotlin의 주요 특징: 안정성

- 앱 실행에 대한 안정성이 우수함
- 앱 비정상 종료의 원인이 되는 **Null Pointer Exception**을 완화하기 위한 **Null Safe**를 지원
  - **Null Safe**란? : 객체의 널 상태를 컴파일러가 자동으로 해결하여 안정성을 확보
- 안정성을 기반으로, Android 앱의 **알고리즘 파트**를 구현



*code path*

```
package com.example.myapplication

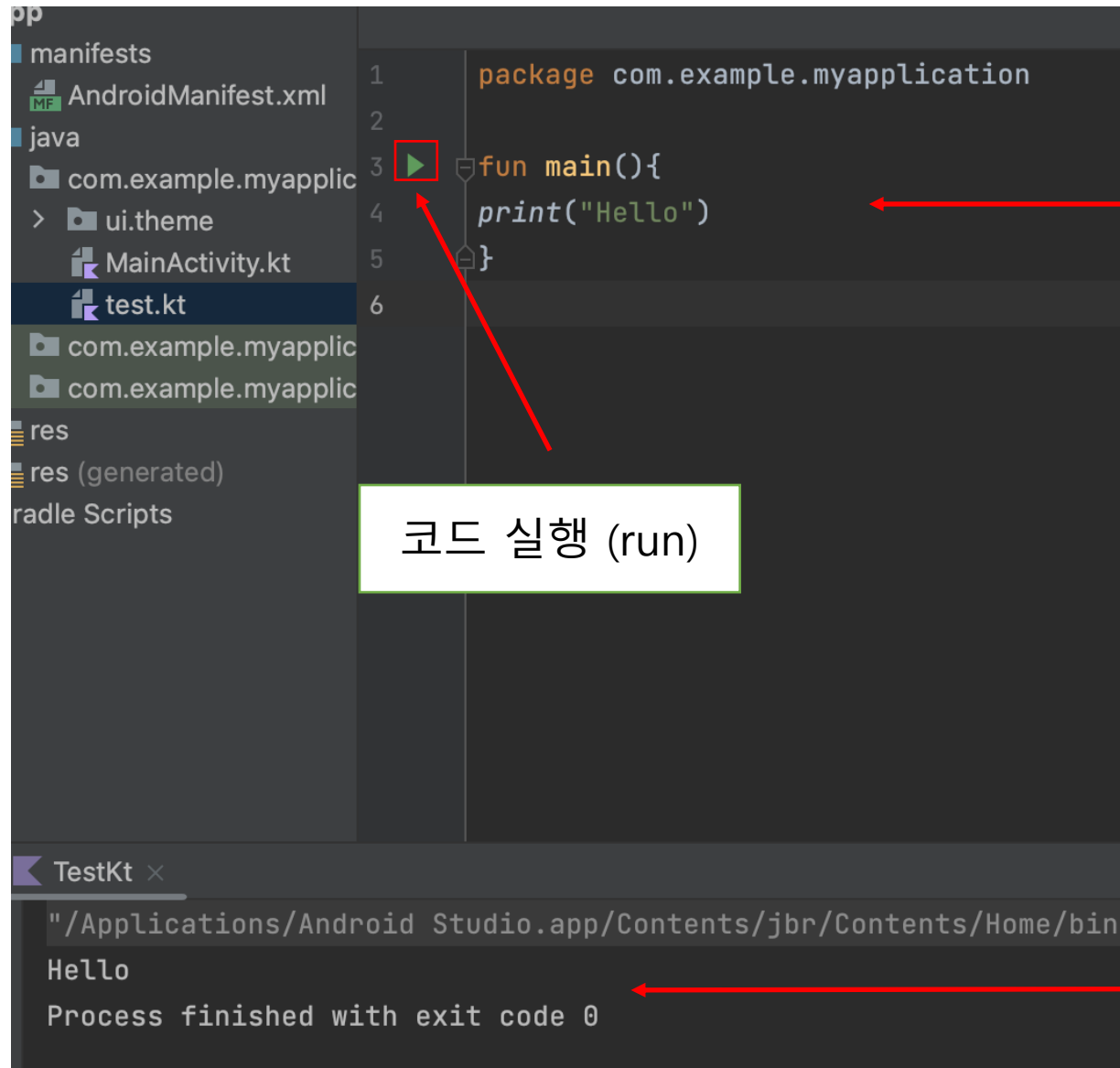
class FirstClass constructor() {
    fun print(){
        println("Hello world")
    }
}

fun main(args : Array<String>){
    var first_object : FirstClass = FirstClass()
    first_object.print()
}
```

```
"C:\Program Files\Android\Android
Hello world

Process finished with exit code 0
```

# 코드 작성 및 실행 (Android Emulator 사용하지 않고, 콘솔창에서만)



코드 작성

코드 실행 (run)

실행 결과

- 변수의 선언

- 변수의 선언 방식은 두 가지가 존재함
  - **Var, Val** 로 변수를 선언 및 초기화

- Var과 Val의 차이점

- **val(value)**은 불변 타입 변수를 지정하며 초기에 값이 할당되면 나중에 값을 변경할 수 없다.
- **var(variable)**은 가변 타입 변수를 지정하며 초기에 값이 할당되어도 나중에 값 변경이 가능하다.

# 변수 선언하기

- Var: 가변 타입의 변수를 지정하고, 변수 값 변경 가능
- Val: 불변 타입의 변수를 지정하고, 변수 값 변경 불가능 -> Java의 final과 비슷

```
1 package com.example.myapplication
2
3 fun main(){
4     val a: Int = 2023
5     println(a)
6
7     var b: Int
8     b = 2024
9     println(b)
10    b = 2025
11    print(b)
12 }
13
```

변수 값 변경 가능

TestKt x

"/Applications/Android Studio.app/Contents/jbr/Contents/Home/bin/java

2023

2024

2025

Process finished with exit code 0

```
1 package com.example.myapplication
2
3 fun main(){
4     val a: Int = 2023
5     println(a)
6     a = 2024
7     print(a)
8 }
9
```

Error  
Val은 변수 값 변경 불가능

Build: Build Output x Build Analyzer x

Build MyApplication: failed At 2023/12/31 5:08 AM with 2 419 ms e: file:///Users/

Download info

app:compileDebugKotlin 1 error 166 ms

test.kt app/src/main/java/com/example/myapplication 1 error

Val cannot be reassigned :6

Compilation error

## • 함수의 선언

- 예약어 **fun** 으로 함수를 선언
- 파라미터 타입과 **리턴 타입**의 설정이 필요함

```
fun plus_number(number1: Int, number2: Int): String{
```

함수 이름

파라미터 설정

리턴 타입

```
return "number1과 number2를 더하면? : "+sum
```

리턴 값

```
package com.example.myapplication

fun plus_number(number1: Int, number2: Int): String{
    var sum = (number1+number2).toString()
    return "number1과 number2를 더하면? : "+sum
}

fun main(){
    var number1: Int = 10
    var number2: Int = 20
    number1 = 20
    number2 = 20
    println(plus_number(number1, number2))
}
```

Modified!

OK!

## ❖ 함수 선언의 간소화

- 파라미터가 존재하지 않는다면, **파라미터 설정 부분 생략이 가능함**
- 리턴 값이 존재하지 않는다면, **리턴 과 리턴 타입 부분 생략이 가능함**

```
"C:\Program Files\Android\Android
number1과 number2를 더하면? : 40

Process finished with exit code 0
```



# 컬렉션 데이터 타입(배열, 리스트)

## • 컬렉션 데이터 타입이란?

- 연관된 데이터를 하나의 변수로 관리하는 방법
- 종류: Array, List, Set, Map
- 세 가지 단계로 활용함
  - 선언과 초기화 -> 사용(값 참조, 출력) -> 수정(추가, 변경, 삭제)

## • Array 타입

- **plus()** 함수를 이용하여 선언한 배열에 값을 추가함
- 선언과 초기화: Array 클래스로 표현

`val(or var) data1: Array<Int> = arrayOf(10, 20, 30)`

배열 이름

정수 배열

초기화

## • List 타입

- 순서가 있는 데이터 집합으로 데이터의 중복을 허용함
- 읽기 전용인 **List** 클래스와 수정가능한 **MutableList** 클래스가 있음
- **MutableList**는 **add()** 함수를 사용하여 값을 추가

선언과 초기화

요소 추가

출력

```
package com.example.myapplication
import java.util.*
fun main(){
    println("===== 배열 =====")
    var list1: Array<Int> = arrayOf(1,2,3,4,5,6,7,8)
    var list2: List<Int> = listOf(1,2,3,4,5,6,7,8)
    var list3: MutableList<Int> = mutableListOf(1,2,3,4,5,6,7,8)

    list1 = list1.plus( element: 9)
    list2 = list2.plus( element: 9)
    list3.add(9)

    println("list1 : ${Arrays.toString(list1)}")
    println("list2 : $list2")
    println("list2 : $list3")

    val filtered_list1 = list1.filter{it%3==0}
    val filtered_list2 = list2.filter{it%3==0}
    val filtered_list3 = list3.filter{it%3==0}

    println("list1의 3의 배수 출력 : $filtered_list1")
    println("list2의 3의 배수 출력 : $filtered_list2")
    println("list3의 3의 배수 출력 : $filtered_list3")
}
```

```
"C:\Program Files\Android\Android S
===== 배열 =====
list1 : [1, 2, 3, 4, 5, 6, 7, 8, 9]
list2 : [1, 2, 3, 4, 5, 6, 7, 8, 9]
list2 : [1, 2, 3, 4, 5, 6, 7, 8, 9]
list1의 3의 배수 출력 : [3, 6, 9]
list2의 3의 배수 출력 : [3, 6, 9]
list3의 3의 배수 출력 : [3, 6, 9]

Process finished with exit code 0
```

## • 필터(filter)

- 조건식을 사용하여 원하는 요소를 추출함
- 필터 결과로 List 타입의 배열을 반환함

```
val filtered_list1 = list1.filter{it%3==0}
```

↓  
필터링  
결과

↓  
필터 입력

↓  
조건식

## • 필터 사용하기

- 조건식의 결과 값에 따라서,  
참이면 결과 배열에 넣고,  
거짓이면 결과 배열에 넣지 않음

배열 필터 적용

필터링 이후  
배열 출력

```
package com.example.myapplication
import java.util.*
fun main(){
    println("===== 배열 =====")
    var list1: Array<Int> = arrayOf(1,2,3,4,5,6,7,8)
    var list2: List<Int> = listOf(1,2,3,4,5,6,7,8)
    var list3: MutableList<Int> = mutableListOf(1,2,3,4,5,6,7,8)

    list1 = list1.plus( element: 9)
    list2 = list2.plus( element: 9)
    list3.add(9)

    println("list1 : ${Arrays.toString(list1)}")
    println("list2 : $list2")
    println("list2 : $list3")

    val filtered_list1 = list1.filter{it%3==0}
    val filtered_list2 = list2.filter{it%3==0}
    val filtered_list3 = list3.filter{it%3==0}

    println("list1의 3의 배수 출력 : $filtered_list1")
    println("list2의 3의 배수 출력 : $filtered_list2")
    println("list3의 3의 배수 출력 : $filtered_list3")
}
```

```
"C:\Program Files\Android\Android St
===== 배열 =====
list1 : [1, 2, 3, 4, 5, 6, 7, 8, 9]
list2 : [1, 2, 3, 4, 5, 6, 7, 8, 9]
list2 : [1, 2, 3, 4, 5, 6, 7, 8, 9]
list1의 3의 배수 출력 : [3, 6, 9]
list2의 3의 배수 출력 : [3, 6, 9]
list3의 3의 배수 출력 : [3, 6, 9]

Process finished with exit code 0
```

## ❖ if - else

- 조건이 참일 때 실행하는 if 구문과, 거짓일 때 실행하는 else 구문으로 구성됨

```
fun main() {
    var grade: Array<String>
    grade = arrayOf("A+", "A", "B+", "B", "C+", "C", "D+", "D", "F")
    var ranking: Int = 1
    var idx: Int = 8

    True! ← if (ranking <= 5) idx = 0
    else if (ranking <= 10) idx = 1
    else if (ranking <= 15) idx = 2
    else if (ranking <= 20) idx = 3
    else if (ranking <= 25) idx = 4
    else if (ranking <= 30) idx = 5
    else if (ranking <= 35) idx = 6
    else if (ranking <= 40) idx = 7
    else if (ranking <= 45) idx = 8
    Pass!

    println("나의 학점은 ? : "+grade[idx])
}
```

## ❖ if-else와 when의 비교

- if-else 문은 조건식으로 판별하고, when 문은 조건값으로 판별 가능함
- 다른 언어의 switch와 달리, when은 조건 값 외에 조건 범위로도 판별 가능함

## ❖ when

- 조건값을 순차적으로 비교하여 일치하는 부분을 찾음

```
fun main() {
    var grade: Array<String>
    grade = arrayOf("A+", "A", "B+", "B", "C+", "C", "D+", "D", "F")
    var ranking: Int = 1
    var idx: Int = 8

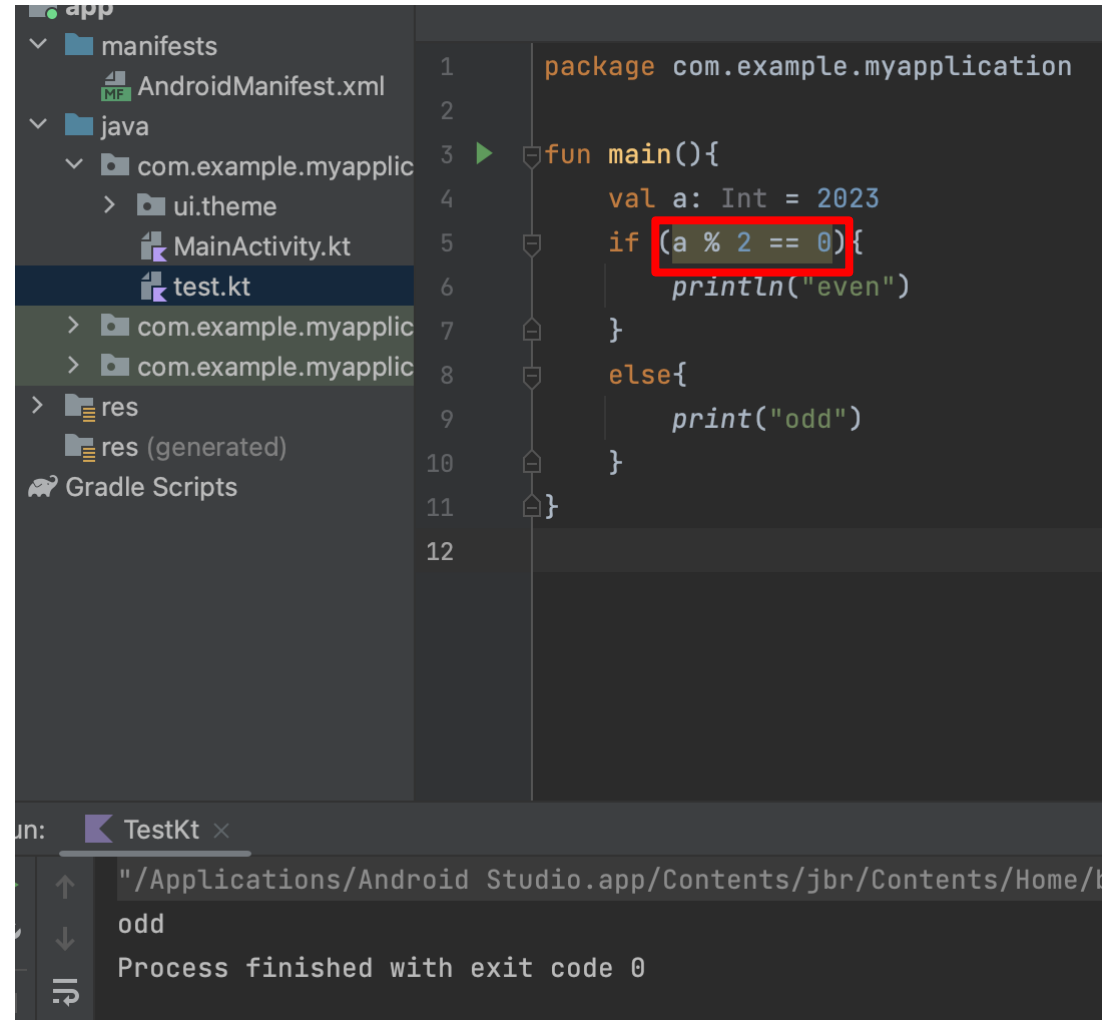
    Style 1
    Match! ← when(ranking / 5) {
        0 -> idx = 0
        1 -> idx = 1
        2 -> idx = 2
        3 -> idx = 3
        4 -> idx = 4
        5 -> idx = 5
        6 -> idx = 6
        7 -> idx = 7
        8 -> idx = 8
    }
    Pass!

    Style 2
    Match! ← when(ranking) {
        in 0 .. 5 -> idx = 0
        in 6 .. 10 -> idx = 1
        in 11 .. 15 -> idx = 2
        in 16 .. 20 -> idx = 3
        in 21 .. 25 -> idx = 4
        in 26 .. 30 -> idx = 5
        in 31 .. 35 -> idx = 6
        in 36 .. 40 -> idx = 7
        else -> idx = 8
    }

    println("나의 학점은 ? : "+grade[idx])
}
```

"C:\Program Files\Android\Android  
나의 학점은 ? : A+

Process finished with exit code 0



```
1 package com.example.myapplication
2
3 fun main(){
4     val a: Int = 2023
5     if (a % 2 == 0){
6         println("even")
7     }
8     else{
9         print("odd")
10    }
11 }
12
```

TestKt ×

"/Applications/Android Studio.app/Contents/jbr/Contents/Home/b  
odd  
Process finished with exit code 0

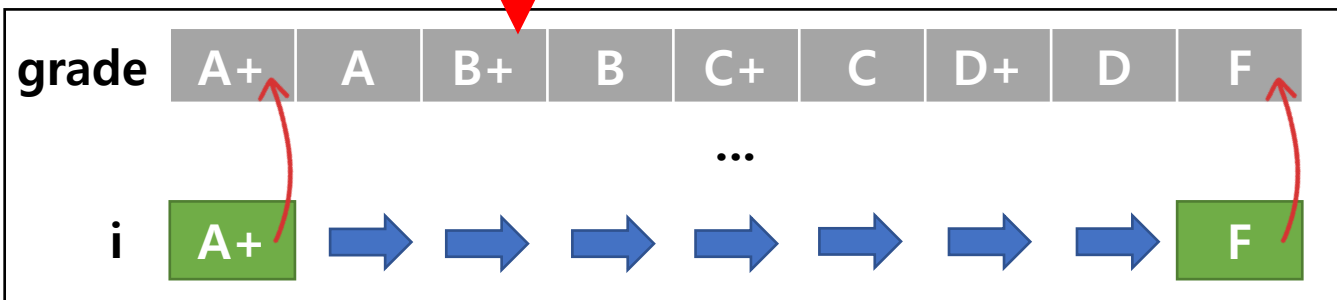
조건문

## ❖ for

- 지정된 범위 내에서 반복을 수행함

```
fun main() {
    var grade: Array<String>
    grade = arrayOf("A+", "A", "B+", "B", "C+", "C", "D+", "D", "F")

    print("학점의 종류는 ? ")
    for (i in grade) print(i + " ")
}
```

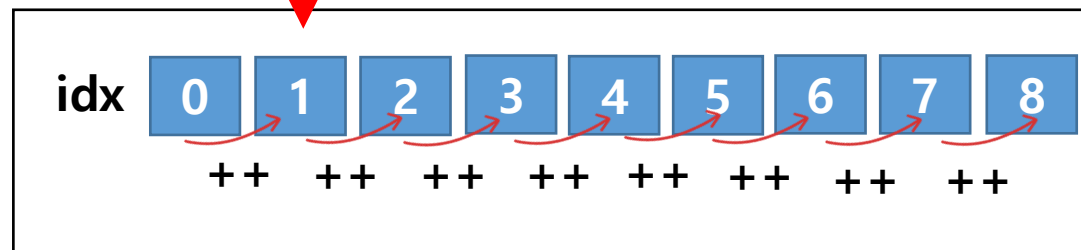


## ❖ While

- 조건식이 일치하는 동안 반복을 수행함

```
fun main() {
    var grade: Array<String>
    grade = arrayOf("A+", "A", "B+", "B", "C+", "C", "D+", "D", "F")
    var idx: Int = 0

    print("학점의 종류는 ? ")
    while (idx < grade.size) print(grade[idx++] + " ")
}
```



## ❖ 반복문의 활용

- 반복문은 코드를 간결하게 만들어 줌
- For문은 in 문과 결합을 통해서 요소에 순차적으로 접근할 수 있어서, 배열을 조작하는 코드를 간결하게 함
- 주로, Android 프로그래밍에서는 UI 위젯 배열에 대한 접근에 사용됨

```
"C:\Program Files\Android\Android S
학점의 종류는 ? A+ A B+ B C+ C D+ D F
Process finished with exit code 0
```

회색 글자 (부등호) 는  
자동 생성되므로  
타이핑 하지 말 것

```
package com.example.myapplication

fun main() {
    for (i: Int in 1 ≤ .. ≤ 5)
        print(i)

    println()
    val len: Int = 5
    for (i in 1 ≤ .. ≤ len)
        print(i)

    println()
    for (i in 1 ≤ until < len)
        print(i)
}
```

for ( i: Int in 1..5 )  
i가 1부터 5까지 증가하며 반복

for ( i in 1..len )  
i가 1부터 len까지 증가하며 반복

for ( i in 1 until len )  
i가 1부터 (len-1)까지 증가하며 반복

TestKt ×

"/Applications/Android Studio.app/Contents/jbr/Contents/Home/bin

12345

12345

1234

Process finished with exit code 0

```
2
3 fun main(){
4     for (i: Int in 1 ≤ .. ≤ 5 step(2))
5         print(i)
6
7     println()
8     val len: Int = 5
9     for (i in 5 ≥ downTo ≥ 1 )
10        print(i)
11
12    println()
13    for (i in 5 ≥ downTo ≥ 1 step(2))
14        print(i)
15 }
16
```

TestKt ×

"/Applications/Android Studio.app/Contents/jbr/Contents/Home/bin

135

54321

531

Process finished with exit code 0

for (i: Int in 1..5 step(2))  
i가 1부터 5까지 2씩 증가하며 반복

for (i in 5 downTo 1)  
i가 5에서 1로 감소하며 반복

for (i in 5 downTo 1 step(2))  
i가 5에서 1까지 2씩 감소하며 반복

# 형 변환 (to변수)

```
java
└─ com.example.myapplication
    └─ ui.theme
        └─ MainActivity.kt
        └─ test.kt
    └─ com.example.myapplication
        └─ com.example.myapplication

res
res (generated)
xml
xml (generated)
Assets
Assets (generated)
Scripts

TestKt x

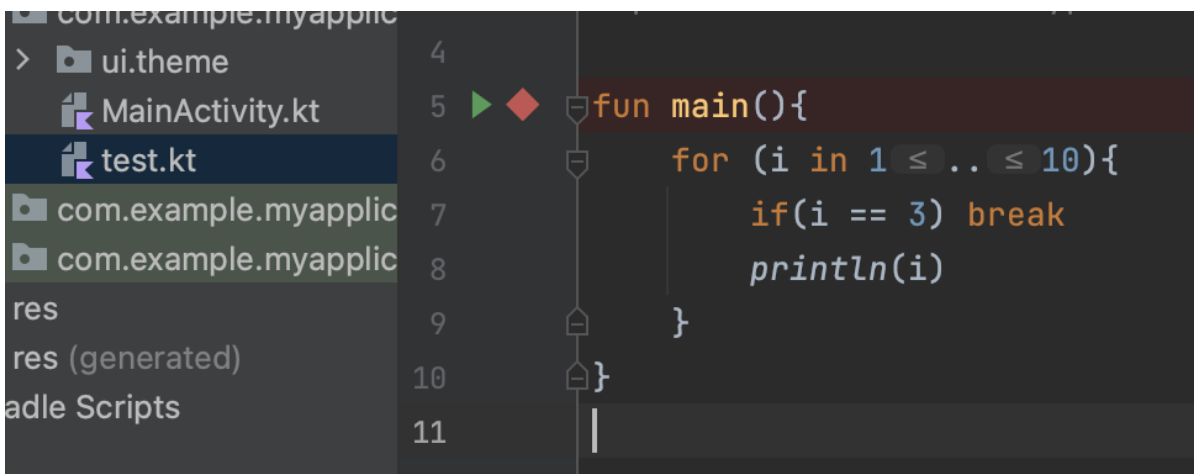
"/Applications/Android Studio.app/Contents/jbr/Contents/Home/bin/
2024
2024
b is String
Process finished with exit code 0
```

```
5 fun main(){
6     var a:Int = 2024
7     println(a)
8     var b: String = a.toString()
9     println(b)
10
11     if(b is String){
12         print("b is String")
13     }
14     else{
15         print("b is not String")
16     }
17 }
18
```

toString() -> Int to String  
데이터타입을 정수에서 문자열로 변환

Type Check  
is는 True이면 1, False이면 0 반환





```
4  
5 fun main(){  
6     for (i in 1..10){  
7         if(i == 3) break  
8         println(i)  
9     }  
10 }  
11
```

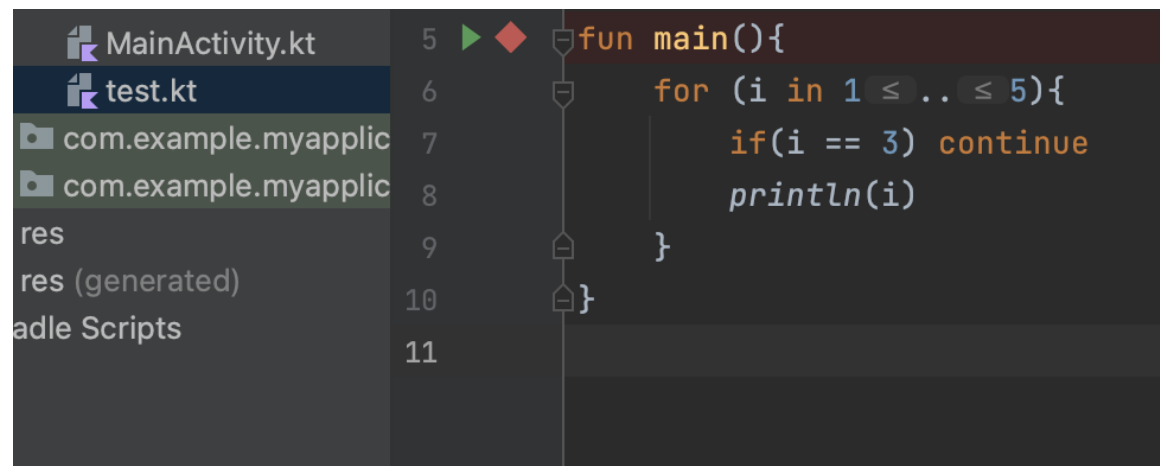
**break**

해당 iteration (반복 차수)를 종료하며, 반복문을 탈출

TestKt ×

"/Applications/Android Studio.app/Contents/jbr/Contents/Hom

1  
2



```
5 fun main(){  
6     for (i in 1..5){  
7         if(i == 3) continue  
8         println(i)  
9     }  
10 }  
11
```

**continue**

해당 iteration를 종료하고, 다음 iteration으로 진입

"/Applications/Android Studio.app/Contents/jbr/Contents/Hom

1  
2  
4  
5

```
1 package com.example.myapplication
2 import java.util.*
3
4 enum class Year{ Freshman, Sophomore, Junior, Senior }
5 class Student(var year:Year, var name: String, var id: Int){
6     fun print_information(){
7         println("YEAR : ${year}, NAME : $name, ID : $id")
8     }
9 }
```

## ◆ enum class : 열거형 클래스

- 동일한 data type의 data를 인스턴스로 하는 class
- 관련된 상수들의 집합을 정의하고, 이를 하나의 타입으로 취급
- 객체이기 때문에 해당 값들을 변수에 할당하거나 함수의 매개변수로 전달

## ◆ enum Class 를 사용하는 이유

- 코드가 단순해지며, 가독성 UP
- 인스턴스 생성과 상속을 방지, 상수값의 타입 안정성 보장

실습

- ◆ 실습

- ◆ 예제 1 홀수 출력
- ◆ 예제 2 점심 메뉴
- ◆ 예제 3 소문자 판별

# 예제 1 - 홀수 출력

- 숫자를 입력, 입력받은 숫자 이하의 홀수 출력

```
com.example.test
├── ui.theme
└── MainActivity.kt
com.example.test (androidTest)
com.example.test (test)
res
res (generated)
gradle Scripts
build.gradle.kts (Project: test)
build.gradle.kts (Module :app)
proguard-rules.pro (ProGuard Rules fo
gradle.properties (Project Properties)
gradle-wrapper.properties (Gradle Ver
local.properties (SDK Location)
settings.gradle.kts (Project Settings)

17
18 fun main(){
19
20     val a = readLine()
21     println("입력 받을 숫자: ${a!!}")
22
23
24     for ( in 1..a){
25         if ( %2 == 1){
26             print(" ")
27         }
28     }
29 }
30
31

MainActivityKt x
"/Applications/Android Studio.app/Contents/jbr/Contents/Home/bin/java" ...
8
입력 받을 숫자: 8
1 3 5 7
Process finished with exit code 0
```

입력 받기

인자 받아오기  
(!! : Null Assertion)  
변수가 Null이 아님을 보장

형변환  
hint : 문자열 -> 정수

## 예제 2 - 점심 메뉴

```
5  [redacted] Food{pizza, burger, chicken}
6
7  class Lunch(var menu: Food, var price: Int){
8      fun choice_lunch(){
9          println("menu : [redacted], price : [redacted] ")
10     }
11 }
12
13 fun main(){
14     [redacted] lunch: MutableList<Lunch> = ArrayList()
15     lunch.add(Lunch(Food.pizza, price: 15000))
16     lunch.add(Lunch(Food.burger, price: 7000))
17     lunch.add(Lunch(Food.chicken, price: 25000))
18
19     var myLunch = lunch.filter{ [redacted] }
20     for([redacted] in myLunch){
21         lunch.choice_lunch()
22     }
23 }
24
```

Run: TestKt x

"/Applications/Android Studio.app/Contents/jbr/Contents/Home/bin/java" ...

menu : burger, price : 7000

Enum class 선언

Class 선언  
및 함수 생성

조건문

배열 필터링으로  
조건문을 통해 메뉴 선택  
(가격이 10000 미만)

## 예제 3 – 윤년 출력하기

### ■ 윤년 출력하기

- 주어진 메인함수를 사용하여 연도가 주어졌을 때, 윤년이면 1, 아니면 0을 반환하는 Year 함수를 작성하시오.

- 윤년
  - 연도가 4로 나누어 떨어지는 해
  - 연도가 400으로 나누어 떨어지는 해
- 평년
  - 연도가 100으로 나누어 떨어지지만 400으로 나누어 떨어지지 않는 해
- 우선순위 :  $400 > 100 > 4$

skeleton code는 레퍼런스. 출력만 동일하면 수정가능.

```
fun Year(number:Int): Int{  
    todo  
}  
  
fun printer(number:Int){  
    if(number == 0) println("윤년이 맞습니다.")  
    else if(number == 1) println("윤년이 아닙니다.")  
}  
  
fun main(){  
    println("2000년은 윤년 일까?")  
    printer(Year( number: 2000))  
  
    println("1900년은 윤년 일까?")  
    printer(Year( number: 1900))  
  
    println("2020년은 윤년 일까?")  
    printer(Year( number: 2020))  
  
    println("2013년은 윤년 일까?")  
    printer(Year( number: 2013))  
}
```