

## < 랬다함수 >

익명함수, fun 키워드 X, 함수이름 X

표현: { 매개변수 → 함수 본문 }

~~~~~  
함수 본문의 마지막 표현식: 함수의 반환값

매개변수 없는 랬다함수

{ → println(" ") } 또는 { println(" ") } 과 같이 왼쪽을 비워두거나 화살표까지 생략가능

매개변수가 1개인 랬다함수

val some = { no: Int → println(no) }

~~~~~  
Int형 매개변수  
이름: no

"it" 키워드사용  
~~~~~

val some: (Int) → Unit = { println(it) }

~~~~~  
익명함수의  
매개변수 타입  
~~~~~  
↑ 익명함수의  
반환타입명시

해당구문은 println 실행 후 반환데이터 값이 없는...

∴ 반환형이 Unit !!!

~~~~~  
데이터 타입 X, 상황을 의미함!!!

람다함수의 반환

return 사용 X, return value 는 본문의 마지막줄 실행결과

• 랬다 함수에서 return 문 사용 오류

```
val some = {no1: Int, no2: Int -> return no1 * no2} // 오류!
```

람다함수에서는 return 키워드 사용 X 계산식값을 그대로 반환

• 랬다 함수의 반환문

```
fun main() {  
    val some = {no1: Int, no2: Int ->  
        println("in lambda function")  
        no1 * no2  
    }  
    println("result : ${some(10, 20)}")  
}
```

▶ 실행 결과

```
in lambda function  
result : 200
```

## 함수 타입 선언

매개변수의 타입, return type.

```
val some : (Int, Int) -> Int = { no1: Int, no2: Int : Int -> no1 + no2 }
```

함수 타입.

함수 내용.

\* 타입 별칭 - typealias

긴 함수타입을 간결하게 표현하는데 효과적.

ex) typealias MyFunType = (Int, Int) -> Int.

✓ val some : MyFunType = { no1: Int, no2: Int : Int -> no1 + no2 }

타입축약

\* 매개변수 타입 생략

매개변수의 타입을 유추할 수 있다면 생략 가능,

앞에 함수타입 지정내용에 따라 매개변수나 반환형 타입을 생략할 수 있다.

함수타입 지정을 생략했다면 ... 함수 본문에서 매개변수 타입 명시해주기.

• 매개변수 타입을 생략한 함수 선언

```
typealias MyFunType = (Int, Int) -> Boolean
val someFun: MyFunType = {no1, no2 ->
    no1 > no2
}
```

MyFunType 함수 타입 선언 덕분에  
함수 본문 내 매개변수에서 타입 선언을 생략해도 매개변수 타입을 유추할 수 있다.

• 매개변수 타입 선언 생략 예

```
val someFun: (Int, Int) -> Boolean = {no1, no2 ->
    no1 > no2
}
```

• 변수 선언 시 타입 생략

```
val someFun = {no1: Int, no2: Int ->
    no1 > no2
}
```

관계연산자를 이용한 계산식이므로, 반환값 타입을 Boolean형으로 유추 가능

## 고차함수

① 매개변수에 함수를 전달

② 반환하는 것이 함수

### • 고차 함수

```
fun hofFun(arg: (Int) -> Boolean): () -> String {  
    val result = if(arg(10)) {  
        "valid"  
    } else {  
        "invalid"  
    }  
    return {"hofFun result : $result"}  
}  
  
fun main() {  
    val result = hofFun({no -> no > 0})  
    println(result)  
}
```

② 반환형도 함수

매개변수 X, 반환형 String  
"람다함수"

함수도 객체  
(O.O.P)

위에 hofFun() 함수 선언부를 보고  
no 타입이 Int인 것을 유추할 수 있음.

함수 호출 결과인 문자열을 반환

result는 String이 아니라 String을 포함하는 람다함수를 반환받음.  
람다함수의 String 출력은 함수호출을 통해 이루어짐.

① hofFun() 함수는

매개변수가 Int형이고 반환형이 Boolean형인  
함수가 들어와야 한다.

② 함수 외부에서 받아들이는

매개변수 없이 본문에서

다른 함수 결과에 따른 String 형 객체의 값을  
전달한다.  
문자열.

### ▶ 실행 결과

hofFun result : valid

## 널 안전성이란?

- 널(null) : 객체 선언, 그러나 초기화 되지 않음.

↳ 사용시 널 포인터 예외발생 (nullPointer Exception)

- 널 안전성 : 널 포인터 예외가 발생하지 않도록 코드를 작성하는 것.

```
• 널 안전성을 개발자가 고려한 코드

fun main() {
    var data: String? = null <- ? 연산자를 통해 null 값을 허용
    val length = if (data == null) {
        0
    } else {
        data.length
    }
    println("data length : $length")
}
```

▶ 실행 결과

data length : 0

```
• 코틀린이 제공하는 널 안전성 연산자를 이용한 코드

fun main() {
    var data: String? = null
    println("data length : ${data?.length ?: 0}")
}
```

▶ 실행 결과

data length : 0

널 허용 연산자

멤버스 연산자

널 안전성 호출 연산자.

멤버스 연산자를 통해 객체가 null 일때, default 값을 지정해 줄 수 있다.

객체가 될 때 대입해야 하는 값, 실행해야 하는 구문이 있는 경우 이용 (null이면 null 대신) 특정값으로 대입

## 널 안전성 제공

객체가 null 일 때 널 포인터 예외가 발생하지 않도록 연산자를 비롯해 코드 구현하는 것 의미.

널 허용으로 선언한 변수의 멤버에 접근할 때 반드시 ?. 연산자를 이용해야 한다.

## 예외발생 연산자 !!

객체가 null인지 여부를 사전에 고려하지 않고

object < null : NullPointerException 발생  
not null : 정상실행

### • 예외 발생 연산자

```
fun some(data: String?): Int {  
    return data!!.length  
}  
  
fun main() {  
    println(some("kkang"))  
    println(some(null))  
}
```

data의 null 여부 상관없이 실행...

data가 < null이면 NullPointerException  
null 아니면 정상동작.

### ▶ 실행 결과

5

Exception in thread "main" java.lang.NullPointerException