< 람다함수 >

- 익명함수 , fun 키워드 X , 함수이름 X
- 표현 : { 매개변수 → 함수 본문 }

  함수 본문의 마지막 표현식 : 함수의 반환값

- 매개변수 없는 람다함수

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

- 매개변수가 1개인 람다함수

```
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

• 람다 함수에서 return 문 사용 오류

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

람다함수에서는 return 키워드 사용 X 계산식값을 그대로 반환

• 람다 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 }
```

(Int, Int) -> Int : 함수 타입.

{ no1 : Int, no2 : Int : Int -> no1 + no2 } : 함수 내용.

* 타입별칭 - typealias

긴 함수타입을 간결하게 표현하는데 효과적.

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

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

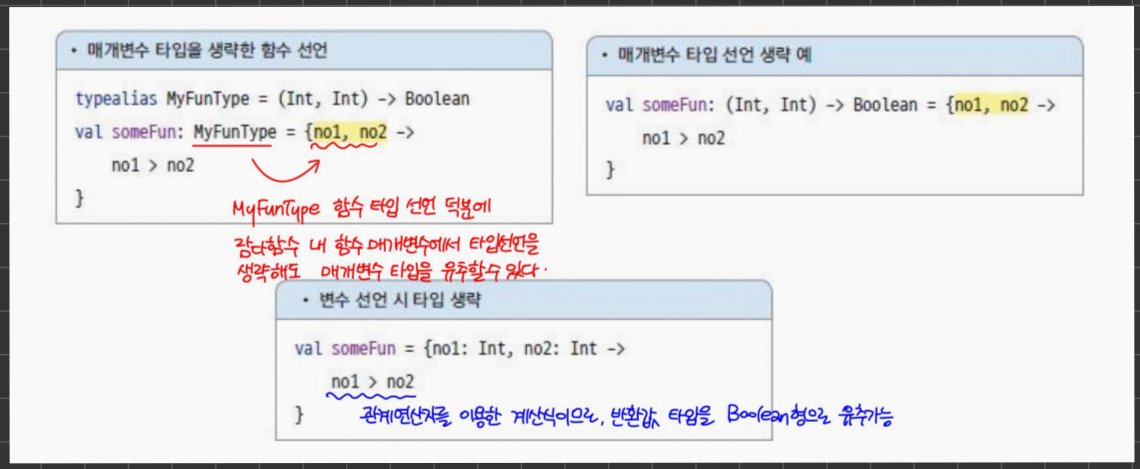
MyFunType : 타입축약

* 매개변수 타입 생략

매개변수의 타입을 유추할수 있다면 생략가능,

앞에 함수타입 지정내용에 따라 매개변수나 반환형 타입을 생략할 수 있다.

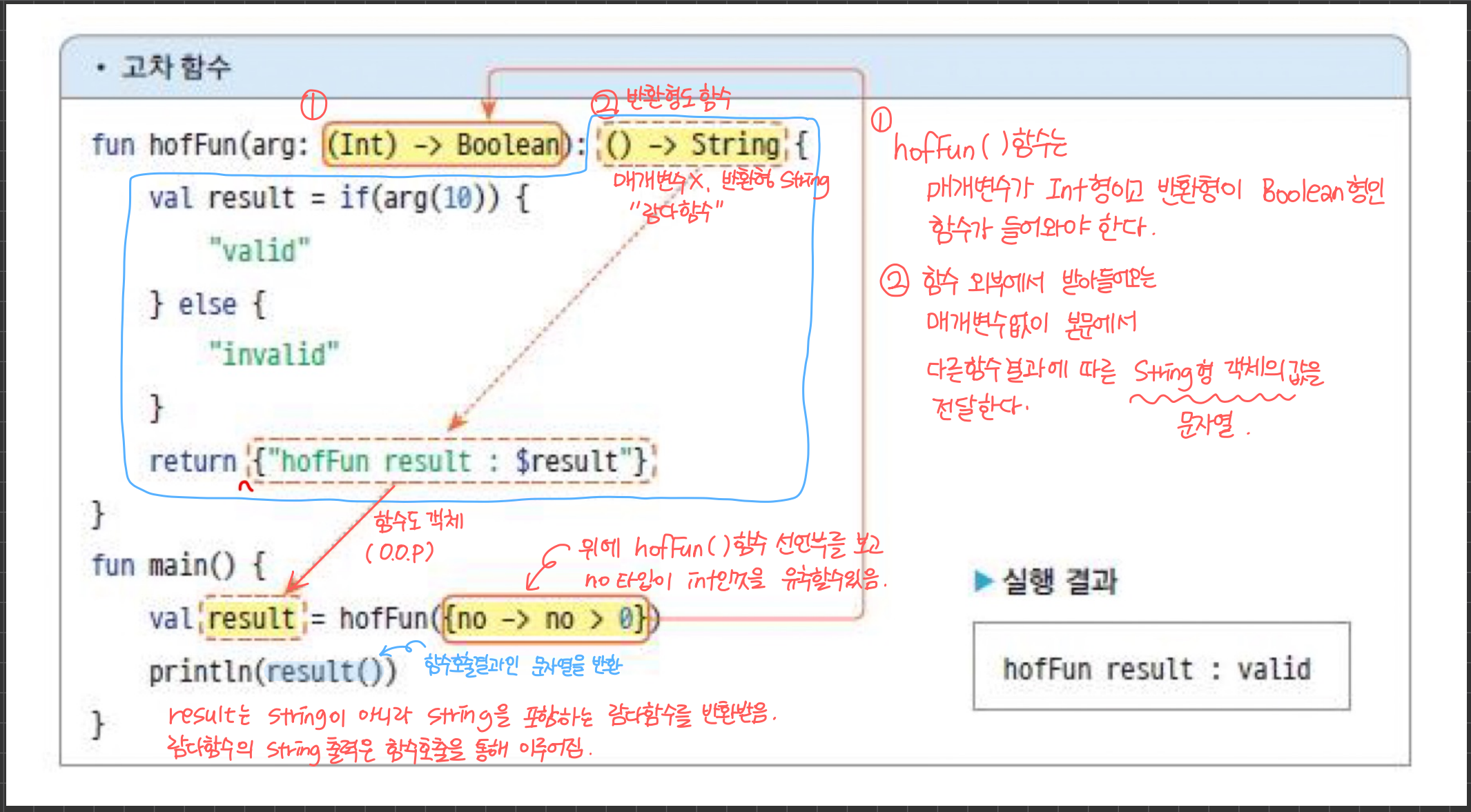
함수타입 지정을 생략했다면 ... 함수본문에서 매개변수타입 명시해주기.



고차함수

① 매개변수에 함수를 전달

② 반환하는 것이 함수



널 안전성이란?

- 널(null) : 객체선언, 그러나 초기화 되진 않음.
  - ↳ 사용시 널 포인터 예외발생 (nullPointerException)
- 널안정성 : 널 포인터 예외가 발생하지 않도록 코드를 작성하는것.

• 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}")
}
```

널허용 연산자 (?)

엘비스 연산자 (?:)

?. 널 안정성 호출 연산자.

▶ 실행 결과

```
data length : 0
```

객체가 널 일때 대입해야할값, 실행해야하는 구문이 있는 경우 이용 (null이면 null 대신) 특정값으로 대입

엘비스 연산자를 통해 객체가 null 일때, default 값을 지정해줄수있다.

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

널 안정성 제공

객체가 null일때 널 포인터 예외가 발생하지 않도록 연산자를 비롯해 코드 구현하는것의 의미.

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 읽고서...

data에가 < null이면 NullPointerException
null 아니면 정상출력.

▶ 실행 결과

```
5
Exception in thread "main" java.lang.NullPointerException
```