04

04-5. 고차함수

다양한 기법

고차함수

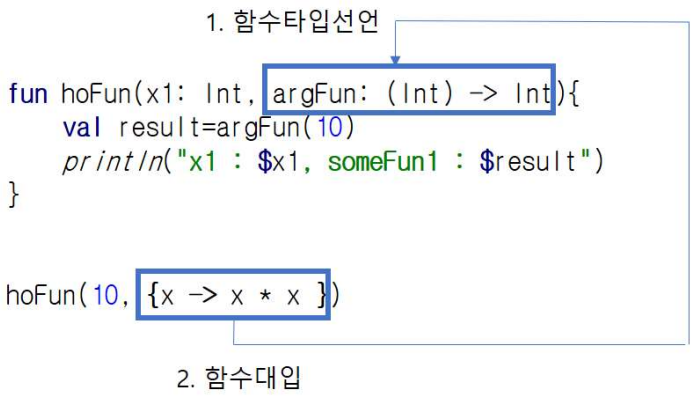
고차함수란?

- 고차함수(High-Order Function, 고계함수)
- 함수의 매개변수로 함수를 전달 받거나 함수를 리턴시킬수 있는 함수를 지칭

```
fun normalFun(x1: Int, x2: Int): Int{
    return x1 + x2
}
```

```
fun hoFun(x1: Int, argFun: (Int) -> Int){
    val result=argFun(10)
    println("x1 : $x1, someFun1 : $result")
}

hoFun(10, {x -> x * x })
```



고차함수

함수 타입의 매개변수

- 일반적으로 함수를 호출할 때는 함수명 뒤에 ( )를 붙이고 ( ) 안에 인수를 작성합니다.
- 그런데 고차 함수의 매개변수가 함수 타입이면 함수 호출 때 ( )을 생략할 수 있습니다.

```
fun hoFun1(argFun: (Int) -> Int){
    val result=argFun(10)
    println("result : $result")
}
hoFun1({x -> x * x})
hoFun1 {x -> x* x }
```

```
val array= arrayOf(10, 20, 15, 22, 8)
array.filter{ x -> x > 10 }
        .forEach{ x -> println(x) }
```

```
inline fun IntArray.filter(
    predicate: (Int) -> Boolean
): List<Int>
```

고차함수

함수 타입의 매개변수

- ( ) 밖에 작성할 수 있는 함수 타입은 맨 마지막 인수만 가능합니다.

```
fun hoFun_1(no: Int, argFun1: (Int)->Int, argFun2: (Int)->Boolean){

}
hoFun_1(10, {it * it}, {it > 10})
hoFun_1(10, {it * it}) {it > 10}
hoFun_1(10){it * it} {it > 10}//error
```

고차함수

함수타입 디폴트값 이용

- 함수의 매개변수에 기본값을 선언하는 것은 함수 타입의 고차 함수에서도 가능

```
fun some(x1:Int = 10){
    println(x1)
}
some()
```

```
fun hoFun2(
        x1: Int,
        argFun1: (Int) -> Int,
        argFun2: (Int) -> Boolean = { x: : Int -> x > 10}
){
    val result = argFun1(x1)
    println("result : ${argFun2(result)}")
}

hoFun2(2, { x: Int -> x * x }, {x: Int -> x > 20})
hoFun2(4, { x: Int -> x * x })
```

고차함수

고차함수와 함수반환

```
fun hoFun5(str: String): (x1: Int, x2: Int) -> Int {
    when (str){
        "-" -> return { x1, x2 -> x1 - x2 }
        "*" -> return { x1, x2 -> x1 * x2 }
        "/" -> return { x1, x2 -> x1 / x2 }
        else -> return { x1, x2 -> x1 + x2 }
    }
}

val resultFun=hoFun5("*")
println("result * : ${resultFun(10, 5)}")
```

고차함수

함수 참조와 익명 함수 이용

- 함수 참조를 이용한 함수 전달

```
fun hoFun6(argFun: (x: Int) -> Int){
    println("${argFun(10)}")
}

hoFun6 { it * 5 }
```

```
fun hoFun6(argFun: (x: Int) -> Int){
    println("${argFun(10)}")
}

fun nameFun(x: Int): Int {
    return x * 5
}
hoFun6(::nameFun)
```

고차함수

익명함수를 이용한 함수 전달

- 람다함수는 이름이 없는 함수
- 람다함수는 return 예약어를 이용하여 리턴 값을 명시할수 없다.

```
val lambdasFun={ x: Int ->
    println("i am lambdas function")
    return x * 10//error
}
```

- 익명함수는 단어뜻 그대로 이름이 없는 함수.

```
val anonyFun1 = fun(x: Int): Int = x * 10

val anonyFun2 = fun(x: Int): Int {
    println("i am anonymous function")
    return x * 10
}
```

```
fun hoFun7(argFun: (Int)->Int){
   println("${argFun(10)}")
}
hoFun7(fun(x: Int): Int = x * 10)
```

고차함수

익명함수를 이용한 함수 전달

- **일반함수**

```
fun myFun(x: Int): Int {
    return x * 10
}
```

```
fun 함수명(매개변수): 리턴타입 {
    함수내용
}
```

- **Anonymous Function**

```
fun(x: Int): Int {
    return x * 10
}
```

```
fun 함수명(매개변수): 리턴타입 {
    함수내용
}
```

⇨

```
fun(매개변수): 리턴타입 {
    함수내용
}
```

- **Lambdas Function**

```
{ x: Int -> x * 10}
```

```
fun 함수명(매개변수): 리턴타입 {
    함수내용
}
```

⇨ { 매개변수 -> 함수내용 }

코틀린 API의 유용한 고차함수

run()

- 단순 람다함수를 실행시키고 그 결과 값을 획득

```
inline fun <R> run(block: () -> R): R
```

```
val result= run {
    println("lambdas function call...")
    10 + 20
}
println("result : $result")
```

코틀린 API의 유용한 고차함수

run()
- 객체의 맴버에 접근

```
inline fun <T, R> T.run(block: T.() -> R): R
```

```
class User() {
    var name: String?=null
    var age: Int?=null

    fun sayHello(){
        println("hello $name")
    }
    fun sayInfo(){
        println("i am $name, $age years old")
    }
}

val user = User()
user.name="kkang"
user.age=33
user.sayHello()
user.sayInfo()
```

```
val runResult=user.run {
    name="kim"
    age=28
    sayHello()
    sayInfo()
    10 + 20
}

println("run result : $runResult")
```

코틀린 **API**의 유용한 고차함수

---

apply()

- apply() 함수는 run() 함수와 사용 목적은 동일한데 리턴되는 값에 차이

- run() 함수의 리턴 값은 대입된 람다함수의 리턴값이 그대로 run() 함수의 리턴값
- apply() 함수는 apply() 함수를 적용한 객체가 리턴

```
inline fun <T> T.apply(block: T.() -> Unit): T
```

```
val user3=user.apply {
    name="park"
    sayHello()
    sayInfo()
}
println("user name : ${user.name}, user3 name : ${user3.name}")//park, park
user.name="aaa"
user3.name="bbb"
println("user name : ${user.name}, user3 name : ${user3.name}")//bbb, bbb
```

코틀린 **API**의 유용한 고차함수

let()

- let 을 이용하는 객체를 let 의 매개변수로 지정한 람다함수에 매개변수로 전달

```
inline fun <T, R> T.let(block: (T) -> R): R
```

```
class User() {

    var name: String?=null
    var age: Int?=null

    constructor(name: String, age: Int) : this() {
        this.name=name
        this.age=age
    }
}
fun letTestFun(user: User){
    println("letTestFun() : ${user.name} .. ${user.age}")
}
val user4=User("kkang", 33)
letTestFun(user4)
```

```
User("kim", 28).let { user ->
    letTestFun(user)
}
```

코틀린 **API**의 유용한 고차함수

---

with()

- with() 함수는 run() 함수와 사용 목적이 유사
- 객체의 맴버들을 반복적으로 접근할 때 객체명을 일일이 명시하지 않고 멤버들을 바로 이용하기 위한 용도
- run() 함수는 run() 함수를 이용한 객체가 람다함수에서 바로 이용
- with() 함수는 with() 함수의 매개변수로 지정한 객체를 람다함수에서 이용

```
inline fun <T, R> with(receiver: T, block: T.() -> R): R
```

```
user.run {
    name="kkang"
    sayHello()
}

with(user){
    name="kkang"
    sayHello()
}
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare