04

04-5. 컬렉션 타입의 함수

Functional Programming

집합 연산 함수

forEach(), forEachIndexed()

- Collection 타입의 데이터 개수만큼 반복 특정 구문을 반복 실행

```
val resultList2=list2.filter { it > 10 }
for(i in resultList2){
    println(i)
}
```

```
list2.filter { it > 10 }
        .forEach { println(it)}
```

- forEachIndexed() 함수는 forEach와 동일하며 람다함수에 index 값까지 전달

```
listOf(1, 2, 3).forEachIndexed { index, value -> println("index : $index, value : $value") }
```

집합 연산 함수

---

all(), any()

- all() 은 Collection 타입의 데이터가 특정 조건에 모두 만족하는지에 대한 판단
- any()는 특정 조건에 만족하는 데이터가 있는지에 대한 판단

```
class User(val name: String, val age: Int)
var list= listOf(User("kkang", 33), User("lee", 28))

println("all test : ${list.all { it.age > 30 }}")
println("any test : ${list.any { it.age > 30 }}")
```

실행결과

```
all test : false
any test : true
```

집합 연산 함수

count(), find()

- count 함수는 람다식으로 대입시킨 조건에 만족하는 데이터 개수를 리턴
- find 함수는 조건에 만족하는 가장 첫번째 데이터를 리턴

```
class User(val name: String, val age: Int)
var list= listOf(User("kkang", 33), User("lee", 28))

println("count test : ${ list.count { it.age > 25 }}")
val user = list.find { it.age > 25 }
println("find test : ${user?.name} ${user?.age}")
```

실행결과

```
count test : 2
find test : kkang 33
```

집합 연산 함수

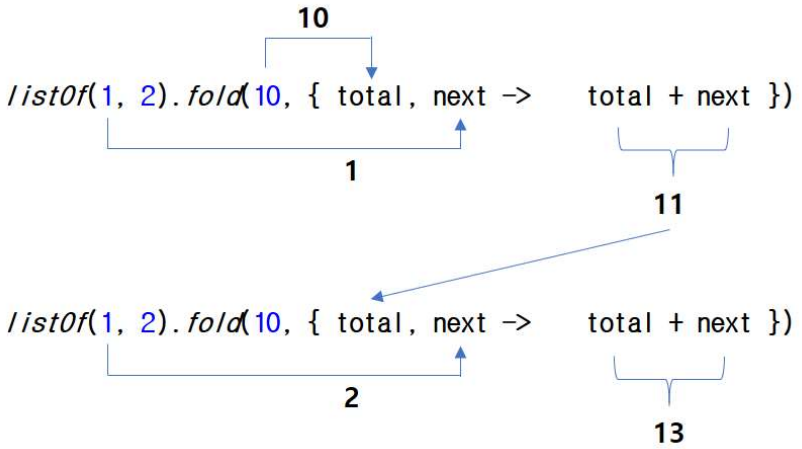
reduce(), reduceRight() fold(), foldRight()

- reduce(), fold() 는 람다함수를 지정하면 Collection 타입의 데이터를 람다함수에 순차적으로 전달
- 람다함수에서 리턴한 값을 기억하고 있다가 그다음 데이터에 의해 호출될 때 이전에 리턴했던 값을 같이 전달
- fold() 는 초기값 지정이 가능하며 reduce() 는 초기값 지정 불가

```
var result=listOf(1, 2).fold(10, { total, next ->
    println("$total ... $next")
    total + next
})
println("fold test : $result")
```

실행결과

```
10 ... 1
11 ... 2
fold test : 13
```



집합 연산 함수

reduce(), reduceRight() fold(), foldRight()

- max 값을 구하는 알고리즘

```
result=listOf(1, 11, 5).fold(10) { max, next ->
    if (next > max) next else max
}
println("fold test : $result")
```

실행결과

```
fold test : 11
```

집합 연산 함수

reduce(), reduceRight() fold(), foldRight()

- foldRight() 함수는 fold와 동일한데 전달되는 데이터가 마지막부터 거꾸로 전달되며 람다함수의 매개변수도 첫번째가 전달되는 데이터, 두번째가 이전 결과값
- reduce는 fold 와 동일한데 단지 초기값 설정 부분이 없다

```
result=listOf(1, 2, 3).foldRight(4) { next, total ->
    println("$total ... $next")
    total + next }
println("foldRight test : $result")
```

실행결과

```
4 ... 3
7 ... 2
9 ... 1
foldRight test : 10
```

집합 연산 함수

max(), maxBy(), min(), minBy()

- max() 는 Collection 타입의 데이터중 가장 큰 값을 리턴시켜 주는 함수
- maxBy() 에는 람다함수를 매개변수로 지정하여 로직에 의한 계산 결과중 가장 큰 수를 리턴
- min() 과 minBy() 는 대입된 데이터중 최소값을 리턴 시켜주는 함수

```
var result1: Int? = listOf(1, 11, 5).max()
println("max test : $result1")
```

실행결과

max test : 11

```
result1=listOf(1, 11, 5).maxBy { it % 10 }
println("max test : $result1")
```

실행결과

max test : 5

집합 연산 함수

none(), sumBy()

- none 함수는 람다함수 조건에 맞는 데이터가 없는 경우 true, 있는 경우 false를 리턴
- sumBy 함수는 람다함수를 거쳐 리턴된 모든 값을 더하는 함수

```
val result2 =listOf(1, 11, 5).none { it % 10 == 0 }
println("none test : $result2")
```

```
실행결과
none test : true
```

```
var result3=listOf(1, 2, 3).sumBy { it * 10 }
println("sumBy test : $result3")
```

```
실행결과
sumBy test : 60
```

필터링 함수

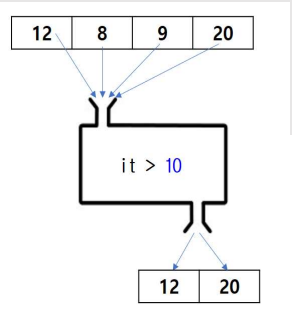
filter()

- 특정 조건에 맞는 데이터만 추출하고자 할 때 이용

```
val list2= listOf<Int>(12, 8, 9, 20)

val resultList=ArrayList<Int>()
for(i in list2){
    if(i>10) resultList.add(i)
}
for(i in resultList){
    println(i)
}
```

```
val list2= listOf<Int>(12, 8, 9, 20)
val resultList2=list2.filter { it > 10 }
for(i in resultList2){
    println(i)
}
```



필터링 함수

---

filter()

- Map 데이터는 키와 값으로 구분됨으로 { }에 전달되는 데이터는 Map.Entry
- entry.key, entry.value

```
val map2= mapOf<String, Int>("one" to 15, "two" to 5)
val resultMap2=map2.filter { entry -> entry.value > 10 }
for(i in resultMap2){
    println(i)
}
```

filterNot(), filterNotNull()

- filterNot() 은 조건에 맞지 않는 데이터만 추출하기 위한 함수
- filterNotNull()은 null이 아닌 데이터만 추출하기 위한 함수

필터링 함수

drop(), dropWhile(), dropLastWhile()

- Collection 데이터중 일부분을 제외하고 나머지를 취하기 위해 사용되는 함수
- drop 함수는 매개변수로 숫자 대입하면 앞 부분부터 대입한 숫자 개수만큼을 제외

```
listOf(1, 2, 3, 4).drop(2)
        .forEach { println(it) }
```

실행결과

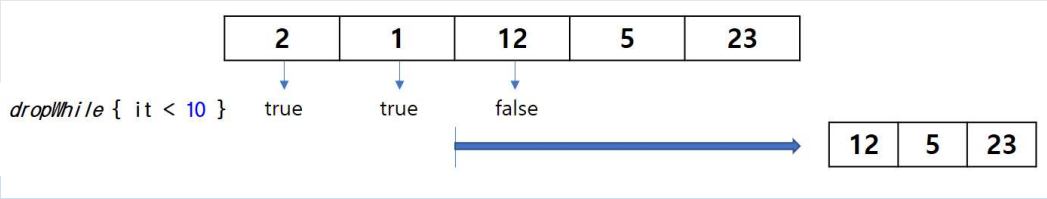
```
3
4
```

- dropWhile 은 매개변수로 람다함수를 지정하며 앞부분부터 조건에 만족하지 않는 데이터가 나올나올 때 데이터를 제외. 그 이후 나머지 데이터를 추출

```
listOf(2, 1, 12, 5, 23).dropWhile { it < 10 }
        .forEach { println(it) }
```

실행결과

```
12
5
23
```



필터링 함수

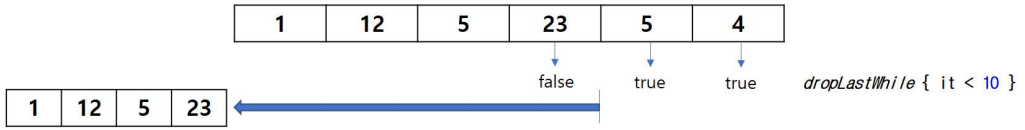
drop(), dropWhile(), dropLastWhile()

- dropLastWhile 함수는 마지막부터 거꾸로 조건을 판단

```
listOf(1, 12, 5, 23, 5, 4).dropLastWhile { it < 10 }
        .forEach { println(it) }
```

실행결과

```
1
12
5
23
```



필터링 함수

slice(), take(), takeLast(), takeWhile()

- 특정 위치에 있는 데이터를 취하기 위한 함수들
- slide는 범위 혹은 숫자 값을 여러 개 대입하고 그 위치에 있는 데이터만을 취하기 위한 함수

```
listOf(1, 12, 5, 23, 5, 4).slice(1 .. 3)
        .forEach { println(it) }
```

실행결과

```
12
5
23
```

```
listOf(12, 5, 23, 5, 4).slice(listOf(0, 2, 4))
        .forEach { println(it) }
```

실행결과

```
12
23
4
```

필터링 함수

slice(), take(), takeLast(), takeWhile()

- take 함수는 숫자 값을 매개변수로 지정하면 앞에서부터 숫자 개수의 데이터 획득
- takeLast는 take의 반대이며 마지막 데이터부터 몇 개를 취하기 위한 함수

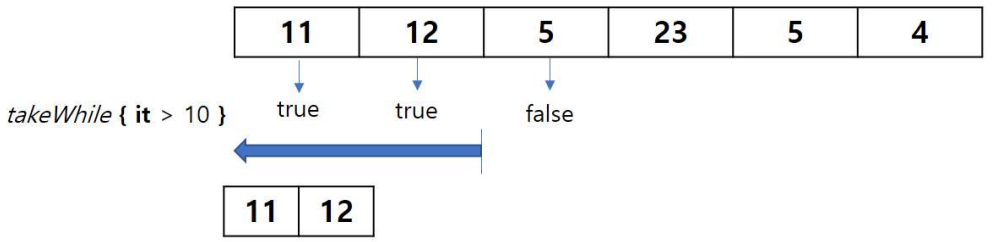
```
listOf(1, 12, 5, 23, 5, 4).take(3)
        .forEach { println(it) }
```

실행결과

```
1
12
5
```

- takeWhile() 은 람다함수로 조건을 명시하고 조건에 맞지 않는 데이터가 나오기 전까지의 데이터를 취하는 함수

```
listOf(11, 12, 5, 23, 5, 4).takeWhile { it > 10 }
        .forEach { println(it) }
```



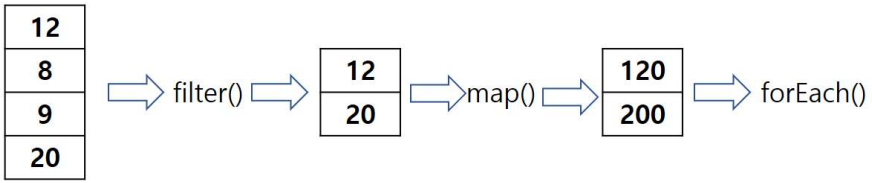
매핑 함수

map(), mapIndexed()

- 집합객체의 데이터 개수만큼 반복 실행
- 반복적 실행 결과를 리턴
- mapIndexed 함수는 map 함수와 동일한데 람다함수에 index 까지 전달

```
val list2= listOf<Int>(12, 8, 9, 20)

list2.filter{ it > 10 }
        .map { it * 10 }
        .forEach { println(it) }
```



매핑 함수

groupBy()

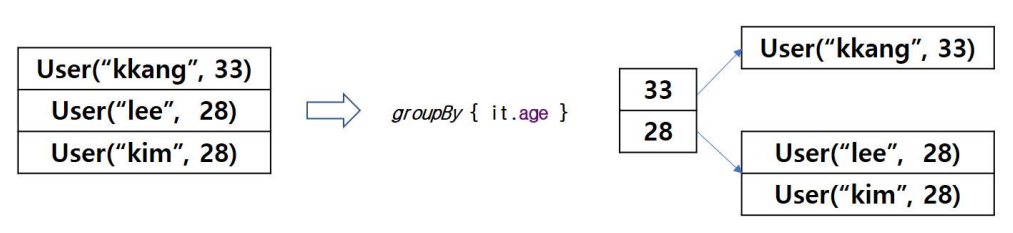
```
inline fun <T, K> Array<out T>.groupBy(
    keySelector: (T) -> K
): Map<K, List<T>>
```

```
class User(val name: String, val age: Int)

list= listOf<User>(User("kkang", 33), User("lee", 28), User("kim", 28))
list.groupBy { it.age }
        .forEach {
            println("key : ${it.key} ... count : ${it.value.count()}")
            it.value.forEach { println("${it.name} .. ${it.age}") }
        }
```

실행결과

```
key : 33 ... count : 1
kkang .. 33
key : 28 ... count : 2
lee .. 28
kim .. 28
```



요소 함수

contains()

- 특정 데이터가 있는지를 판단

```
val result=listOf(2, 5, 10, 8).contains(7)
println("contains test : $result")
```

실행결과

contains test : false

요소 함수

elementAt(), elementAtOrElse(), elementAtOrNull()

- elementAt() 함수는 특정 위치의 데이터를 획득, index 값이 데이터 범위를 벗어난 위치를 지칭하게 되면 IndexOutOfBoundsException 이 발생
- elementAtOrElse 함수는 elementAt 함수와 동일, index 값이 데이터 범위를 벗어나서 지정되는 람다함수가 실행
- elementAtOrNull 함수는 elementAt과 동일, 데이터 범위를 벗어난 index 값이 지정이 되면 Null을 리턴

```
val result1=listOf(2, 5, 10, 8).elementAt(2)
println("elementAt test : $result1")
```

```
result1=listOf(2, 5, 10, 8).elementAtOrElse(5, { 0 })
println("elementAtOrElse test : $result1")
```

요소 함수

first(), firstOrNull(), last(), lastOrNull()

- first 함수는 람다함수로 조건을 주고 그 조건에 맞는 가장 첫 데이터를 획득, 조건에 맞는 데이터가 하나도 없으면 NoSuchElementException 이발생
- last 함수는 람다함수로 조건을 주고 그 조건에 만족하는 가장 마지막 데이터를 획득

```
result1=listOf(2, 5, 10, 8).last { it % 5 == 0 }
println("last test : $result1") //10
```

요소 함수

indexOf(), indexOfFirst(), indexOfLast()

- 데이터가 위치하는 첫번째 index 값을 반환, 데이터가 없다면 -1이 반환
- indexOfFirst() 는 람다함수를 대입
- indexOfLast() 는 마지막 위치를 반환

```
result1=listOf(2, 5, 10, 2).indexOf(2)
println("indexOf test : $result1") //0
```

```
result1=listOf(2, 5, 10, 2).indexOfFirst { it % 2 == 0 }
println("indexOfFirst test : $result1") //0
```

정렬 함수

reversed()

- 데이터의 순서를 거꾸로 바꾸기 위해서 사용

```
listOf(1, 5, 2).reversed()
        .forEach { println(it) }
```

sorted(), sortedBy(), sortedDescending(), sortedDescendingBy()

- sorted 함수는 정렬
- sortedBy 함수는 람다함수를 대입
- sortedDescending, sortedDescendingBy 함수는 역순

```
listOf(1, 5, 2).sorted()
        .forEach { println(it) }//1, 2, 5
```

```
listOf("lee","kkang","park").sorted()
        .forEach { println(it) } //kkang, lee, park
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare