함수와 함수형 프로그래밍

코틀린 기초강좌

강사: 배정만

학습요약

- 학습목표: 함수 선언 및 구현 학습
- 핵심키워드: 함수, 함수형 프로그래밍, Higher-order, Anonymous
- 학습내용: 함수 선언과 인수, 매개변수, 함수형 프로그래밍 기초에 대해서 배웁니다.

함수

```
fun double(x: Int): Int {
    return 2 * x
}
```

```
val result = double(2)
```

```
val result = Sample().foo()
```

매개변수(Parameters)

```
fun powerOf(number: Int, exponent: Int) { ... }
```

```
fun read(b: Array<Byte>, off: Int = 0) { ... }
```

이름있는 인수

```
fun reformat(str: String,
             normalizeCase: Boolean = true,
             upperCaseFirstLetter: Boolean = true,
             divideByCamelHumps: Boolean = false,
             wordSeparator: Char = ' ') {
...
}
```

```
reformat(str, true, true, false, '_')
```

```
reformat(str,
    normalizeCase = true,
    upperCaseFirstLetter = true,
    divideByCamelHumps = false,
    wordSeparator = '_'
)
```

Unit-returning functions

```
fun printHello(name: String?): Unit {
    if (name != null)
        println("Hello ${name}")
    else
        println("Hi there!")
    // `return Unit` or `return` is optional
}
```

Single-Expression functions

```
fun double(x: Int): Int {
  return x * 2
}

fun double(x: Int): Int = x * 2

fun double(x: Int) = x * 2
```

vararg (가변 인수)

```
fun <T> asList(vararg ts: T): List<T> {
    val result = ArrayList<T>()
    for (t in ts) // ts is an Array
        result.add(t)
    return result
}
```

```
val list = asList(1)
```

```
val list = asList(1, 2, 3)
```

함수 범위

- Local Functions

```
fun dfs(graph: Graph) {
    fun dfs(current: Vertex, visited: Set<Vertex>) {
        if (!visited.add(current)) return
        for (v in current.neighbors)
            dfs(v, visited)
    }

    dfs(graph.vertices[0], HashSet())
}
```

- Member Functions

```
class Sample() {
    fun foo() { print("Foo") }
}
```

제네릭 함수

```
fun <T> singletonList(item: T): List<T> {
    ...
}
```

Extensions

```
fun String.lastChar() = this.get(this.length - 1)

“abc".lastChar()
```

함수형 프로그래밍

Higher-order function

```
fun higherOrder(value: String, op: (String) -> String):
String {
    println("Executing the operation $op")
    return op(value)
}

fun lowerCase(value: String) = value.toLowerCase()
```

```
fun main(args: Array<String>) {
    println(higherOrder("HELLO", ::lowerCase))
}
```

Executing the operation fun lowerCase(kotlin.String): kotlin.String
hello

Lambda

```
max(strings, { a, b -> a.length < b.length})
```

```
val sum = { x: Int, y: Int -> x + y }
```

```
val product = items.fold(1) { acc, e -> acc * e }
```

```
run { println("...") }
```

Anonymous functions

```
fun(x: Int, y: Int): Int = x + y

fun(x: Int, y: Int): Int {
    return x + y
}
```