

# Kotlin : 함수

Mobile Software  
2021 Fall

All rights reserved, 2021, Copyright by Youn-Sik Hong (편집, 배포 불허)

# What to do next?

- Kotlin 함수
- 람다 식과 고차 함수
- 확장 함수와 중위 함수
- Kotlin 강의 노트에서는
  - 소스 코드를 별도로 제공하지 않음
  - 실행 결과를 포함하지 않음.
  - 직접 코드를 입력하고, 강의 노트 설명을 확인해야 함.
  - Required assignments
    - 강의 노트에 해당하는 강의를 종료되면, 해당 주 일요일 23:59:59까지 kotlin 실습 파일 제출 → 평소 점수에 반영

# Basics and Syntax (1/2)

Methods: Structure

The diagram illustrates the structure of a Kotlin method. It shows two examples of method declarations with labels pointing to their components:

- keyword**: Points to the `fun` keyword in both examples.
- Method Name**: Points to `findArea` in both examples.
- Formal Parameters**: Points to the parameter list `(length: Int, breadth: Int)` in both examples.
- Return Type**: Points to the return type `Int` in the first example and `Unit` in the second example.
- Void in Java**: A label pointing to `Unit` in the second example, indicating that `Unit` is used for void-like return types in Kotlin.

```
fun findArea(length: Int, breadth: Int): Int {  
    // Method Body: Put your code here  
    return length * breadth  
}  
  
fun findArea(length: Int, breadth: Int): Unit {  
    print(length * breadth)  
}
```

**Unit** 은  
생략 가능

# Basics and Syntax (2/2)

```
fun main() {  
    val result1 = add( a: 2, b: 4)  
    val result2 = add( a: 6, b: 3)  
    println(result1)  
    println(result2)  
}
```

```
fun add(a: Int, b: Int) :Int {  
    return a+b  
}
```



코드가 한 줄이면 중괄호와  
return 문 없이 **할당문**으로 표현

```
fun add(a: Int, b: Int): Int = a+b
```



return 타입을 쉽게  
예측할 수 있기 때문에 생략

```
fun add(a: Int, b: Int) = a+b
```

# Kotlin Functions as Expressions (1/2)

```
fun main() {  
    var largerValue = max( a: 4, b: 6)  
    println("The greater value is $largerValue")  
}
```

```
fun max(a: Int, b: Int) :Int {  
    if (a > b)  
        return a  
    else  
        return b  
}
```

함수는 수식이다.

함수의 연산 결과인 `return` 값을 대입하는 할당문이다.

`fun max(a: Int, b: Int) :Int = if (a > b) a else b`

if 문의 조건식 결과에 따라 a 또는 b를 반환

`fun max(a: Int, b: Int) = if (a > b) a else b`

# Kotlin Functions as Expressions (2/2)

```
fun main() {  
    var largerValue = max( a: 4, b: 6)  
    println("The greater value is $largerValue")  
}
```

```
fun max(a: Int, b: Int) :Int {  
    if (a > b)  
        return a  
    else  
        return b  
}
```

```
fun max(a: Int, b: Int) : Int  
= if (a > b) {  
    println("$a is greater")  
    a  
} else {  
    println("$b is greater")  
    b  
}
```

if 문의 조건식 결과에 따라 실행되는 블록 문에서  
맨 마지막에 있는 변수를 반환

```
fun max(a: Int, b: Int) : Int  
= if (a > b) {  
    println("$a is greater")  
    a  
} else {  
    println("$b is greater")  
    b  
    98  
}
```

<Quiz>  
max (4, 6) 일 때  
return 되는 값은?

# Functions with no return values

```
fun main() {  
    max(a: 4, b: 6)  
}
```

```
fun max(a: Int, b: Int) : Unit {  
    println("sum of $a and $b is ${a+b}")  
}
```

Java의 void 와 유사.  
차이점은 Unit 은 data type.



```
fun max(a: Int, b: Int) = println("sum of $a and $b is ${a+b}")
```

Unit 은 생략할 수 있음.

# Named Parameters

```
fun main() {  
    println(findVolume(2, 3))  
    println(findVolume(2, 3, 30))  
    println(findVolume(width=20, length=10))  
    println(findVolume(height=30, length=10, width=10))  
}  
  
fun findVolume(length:Int, width:Int, height:Int=10)  
    = length * width * height
```

실 매개변수(actual parameters)

실 매개변수의 순서를 바꿈

형식 매개변수(formal parameters)

디폴트(default) 값을 지정.  
함수를 호출할 때 이 parameter를  
생략하면 default 값(=10)을 할당.



# Functions with various arguments

```
fun main() {  
    println(add(1, 2, 3))  
    println(add(1, 2, 3, 4))  
    println(add(1, 2, 3, 4, 5))  
}  
  
fun add(vararg values: Int): Int {  
    var sum = 0  
    for (num in values) {  
        sum += num  
    }  
    return sum  
}
```

**가변 인자** (variable arguments)  
인자의 개수가 정해지지 않음

# What to do next?

- Kotlin 함수
- 람다 식과 고차 함수
- 확장 함수와 중위 함수

# Functional Programming (FP)

- **함수형 프로그래밍(FP):** 거의 모든 코드를 순수 함수로 작성하여 프로그램의 부작용(side effect)을 해결하려는 프로그래밍 기법
  - 순수 함수를 사용
  - 란다 식(일급 함수)을 사용
  - 고차 함수를 사용
  - Kotlin은 함수형 프로그래밍 언어
- **순수 함수(pure function)**

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

  - 같은 parameter에 대하여 항상 같은 값을 return한다.
  - 함수 바깥의 어떤 상태도 바꾸지 않는다.
- **일급 함수는 일급 객체(first class citizen)**
  - 란다 식(lambda expression)은 이름이 없는 일급 함수
  - 일급 객체는 함수의 parameter 로 전달할 수 있다
  - 일급 객체는 함수의 return 값에 사용할 수 있다
  - 일급 객체는 변수에 저장할 수 있다.
- **고차 함수(high order function)**
  - 다른 함수를 parameter로 사용하거나 함수를 return 값으로 반환하는 함수

# 람다 식(Lambda( $\lambda$ ) Expression)

- 람다 식: **function with no name**
  - 사용 방법: 변수에 람다 식을 할당

```
fun main() {  
  val add: (Int, Int) -> Int = {x: Int, y: Int -> x + y}  
  println(add(1, 2))  
}
```

Diagram labels for the code above:

- 변수 (Variable): points to `val`
- 변수 타입 (Variable Type): points to `(Int, Int) -> Int`
- 람다 식 (Lambda Expression): points to `{x: Int, y: Int -> x + y}`

람다 식의  
형식 인자

람다 식의  
본문(1문장)

`val add: (Int, Int) -> Int = {x, y -> x + y}`

람다 식에서 타입 생략

Which one is prefer?

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

변수 타입 생략

# 고차 함수(High order Functions) (1/2)

- Can accept **functions** as **parameters**.
- Can **return a function**.
- Or can do both.

```
fun main() {  
    val result1 = add( a: 2, b: 3)  
    val result2 = multiply(add( a: 4, b: 5), b: 6)  
  
    println(result1)  
    println(result2)  
}  
  
fun add(a: Int, b: Int) = a + b  
fun multiply(a: Int, b: Int) = a * b
```

함수 인자로 함수를 사용

## 고차 함수 (2/2)

- Can accept **functions** as **parameters**.
- Can **return a function**.
- Or can do both.

```
fun main() {  
    val result = highOrderFunc( a: 2, b: 3)  
    println(result)  
}  
  
fun add(a: Int, b: Int) = a + b  
fun highOrderFunc(a: Int, b: Int): Int {  
    return add(a, b)  
}
```

함수를 return 값으로 사용

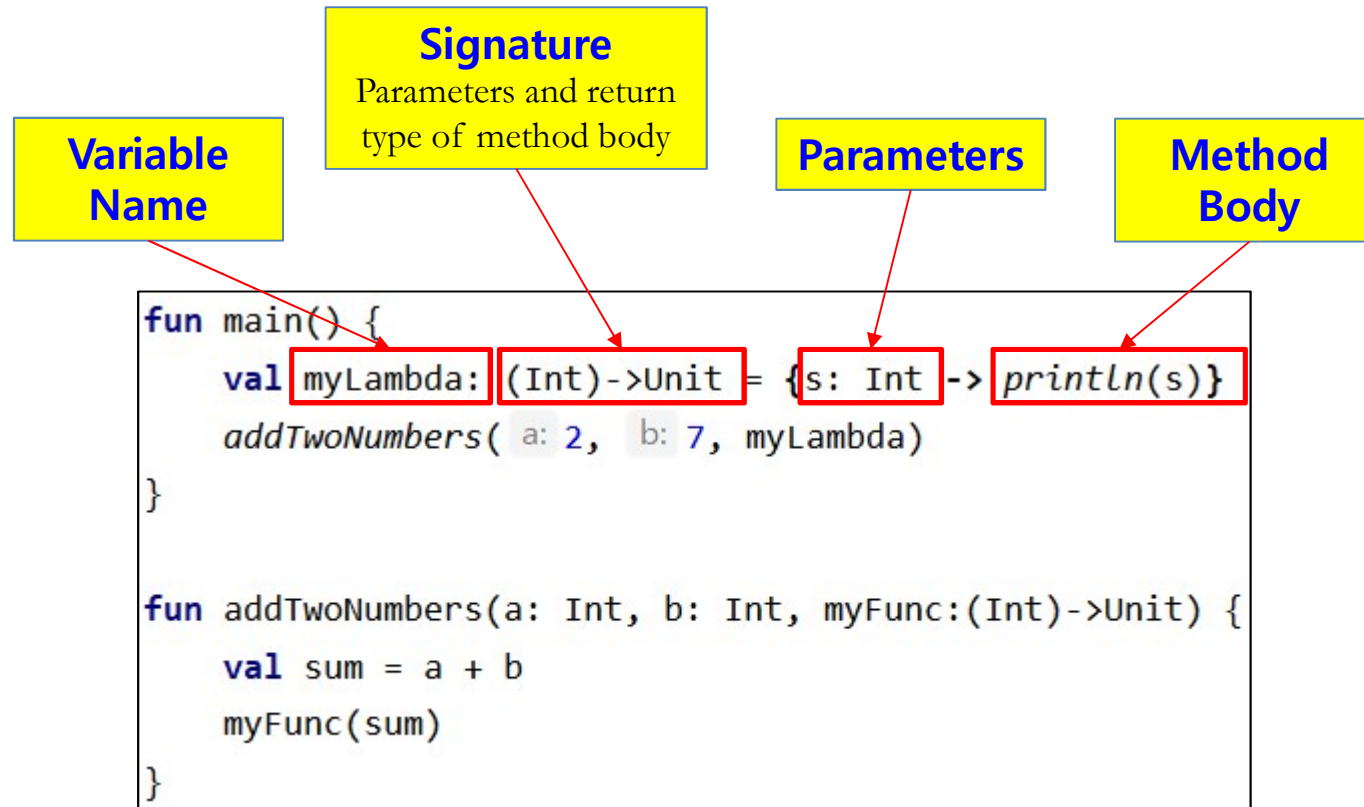
# 람다 식을 갖는 고차 함수 (1/3)

```
fun main() {  
    val myLambda: (Int)->Unit = {s: Int -> println(s)}  
    addTwoNumbers(a: 2, b: 7, myLambda)  
}  
  
fun addTwoNumbers(a: Int, b: Int, myFunc: (Int)->Unit) {  
    val sum = a + b  
    myFunc(sum)  
}
```

return 값이  
없음

람다 식이  
형식 인자로 사용됨

## 람다 식을 갖는 고차 함수 (2/3)





# 람다 식을 갖는 고차 함수 (3/3)

```
fun main() {  
    val myLambda: (Int, Int) -> Int = { x, y -> x + y }  
    addTwoNumbers(2, 7, myLambda)  
    addTwoNumbers(2, 7, { x, y -> x + y })  
}  
  
fun addTwoNumbers(a:Int, b:Int, myFunc:(Int, Int) -> Int) {  
    val result = myFunc(a, b)  
    println(result)  
}
```

람다 식 signature



```
fun main() {  
    val myLambda = { x:Int, y:Int -> x + y }  
    addTwoNumbers(2, 7, myLambda)  
    addTwoNumbers(2, 7){ x, y -> x + y }  
}  
  
fun addTwoNumbers(a:Int, b:Int, myFunc:(Int, Int) -> Int) {  
    println(myFunc(a, b))  
}
```

람다 식을 괄호  
밖으로 이동

# 익명 함수 (Anonymous functions)

- Anonymous function 역시 이름이 없음.
- Lambda expression과 비슷하지만, 일반 함수이기 때문에 제어 문장 (return, break, continue)을 사용할 수 있음.

```
fun main() {  
    val add: (Int, Int) -> Int = fun(a, b) = a + b  
    println(add(10, 2))  
}
```

변수 타입

익명 함수

변수 타입 생략

```
val add = fun(a: Int, b: Int): Int = a + b
```

중괄호로 표현

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

# Inline functions : 코드를 복사하여 함수를 구현

- Inline function을 호출하는 곳에 함수 본문 내용을 그대로 복사.
  - lambda expression과 같은 형태의 매개변수를 사용
  - 일반 함수보다 빨리 처리되기 때문에 성능 개선에 도움.
- Inline function의 본문은 짧아야 하며, 많이 사용하지 않아야 함.

```
fun main() {  
    println(add( a: 10, b: 2))  
    println(add( a: 3, b: 4))  
}  
  
inline fun add(a: Int, b: Int): Int = a + b
```

Tools > Kotlin  
> Show Kotlin Bytecode  
> Decompile



```
public static final void main() {  
    byte a$iv = 10;  
    int b$iv = 2;  
    int $i$f$add = false;  
    int var3 = a$iv + b$iv;  
    boolean var4 = false;  
    System.out.println(var3);  
  
    a$iv = 3;  
    b$iv = 4;  
    $i$f$add = false;  
    var3 = a$iv + b$iv;  
    var4 = false;  
    System.out.println(var3);  
}
```

# What to do next?

- Kotlin 함수
- 람다 식과 고차 함수
- 확장 함수와 중위 함수

# 확장 함수 (Extension Functions)

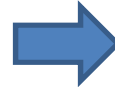
- Adds new functions to the classes
  - Can “**add**” functions to a class without declaring it.
  - The new functions added behaves like **static**.
- **Few properties**
  - They can become **part of your own class**.
    - <e.g.> Student, Employees, etc.
  - They can become **part of predefined classes**.
    - <e.g.> String, Int, Array, etc.
- **Benefits**
  - Reduces code
  - Code is much cleaner and easy to read

# Extension Functions 예(1/2)

```
class Student {  
    fun hasPassed(score: Int): Boolean {  
        return score > 60  
    }  
}
```

Student: 사용자  
정의 class

```
fun main() {  
    var student = Student()  
    println("Pass status: "  
        + student.hasPassed(78))  
}
```



```
class Student {  
    fun hasPassed(score: Int): Boolean {  
        return score > 60  
    }  
}
```

확장 함수: isScholar ()메소드 추가

```
fun Student.isScholar(score: Int): Boolean {  
    return score > 95  
}
```

```
fun main() {  
    var student = Student()  
    println("Pass status: "  
        + student.hasPassed(78))  
    println("Scholarship status: "  
        + student.isScholar(78))  
}
```

확장 함수로 추가한  
isScholar ()메소드 호출

# Extension Functions 예(2/2)

Kotlin에서  
정의한  
String 클래스

```
fun main() {  
    var str1: String = "Hello, "  
    var str2: String = "Kotlin!"  
  
    var tmp = str1.add(str2)  
    println(tmp)  
}
```

확장 함수:  
add ()메소드 추가

```
fun String.add(s1: String): String {  
    return this + s1  
}
```

# 중위 함수 (Infix Functions)

- 중위 표기법을 지원하는 함수

$add(x, y) \rightarrow x \text{ *add* } y$

- Infix functions can be a **Member** function or **Extension** function.

- All *Infix* functions are *Extension* function.
- But all *Extension* functions are not *Infix*.

- They have the **SINGLE** parameter.

- They have prefix of "**infix**".

```
infix fun Double.divide(x:Double) Double = this / x

fun main() {
    println(30.0 divide 7.3)
}
```



# Infix Functions 예

```
class newFunc(var left:Int, var right:Int) {  
    infix fun increaseBy(amount:Int) {  
        this.left += amount  
        this.right += amount  
    }  
}
```

중위 함수(infix function)  
increaseBy ()메소드 추가

```
infix fun newFunc.decreaseBy(amount:Int) {  
    this.left -= amount  
    this.right -= amount  
}
```

확장 함수:  
중위 함수(infix function)  
decreaseBy ()메소드 추가

```
fun main() {  
    var currentValue = newFunc(20, 20)  
    currentValue increaseBy 30  
    println("${currentValue.left}, ${currentValue.right}")  
  
    currentValue decreaseBy 20  
    println("${currentValue.left}, ${currentValue.right}")  
}
```