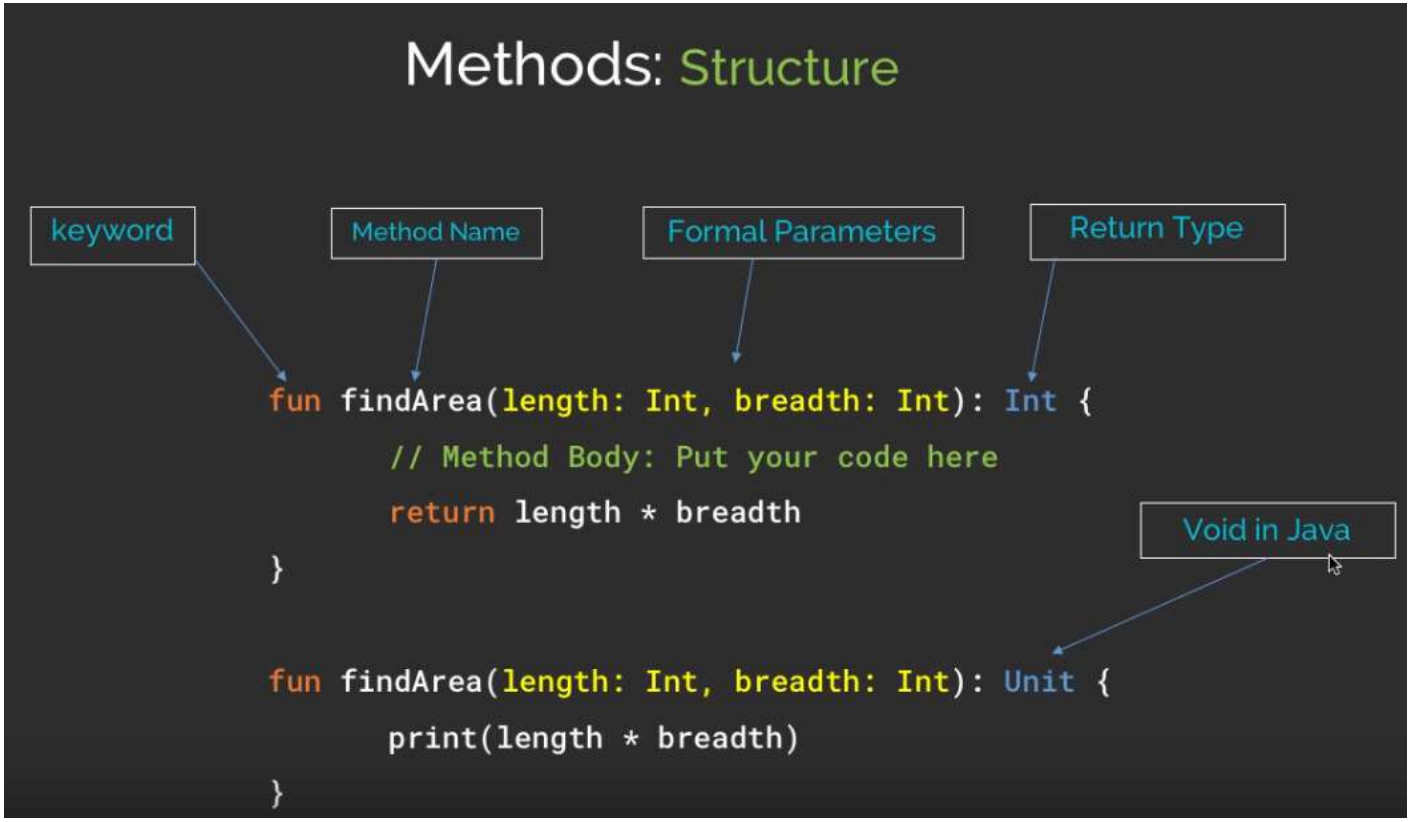
Kotlin : 함수

Mobile Software
2019 Fall
인천대학교 컴퓨터공학부
홍 윤식 교수

Basics and Syntax (1/2)



Unit 은 생략가능

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
}
```

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

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

Kotlin Functions as Expressions

```
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) a else b
```

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

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

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

마지막에 있는 값을 return
여기서는 b 대신 98을 return

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.
그러나 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() {
    findVolume(length=2, width=3)
    findVolume(length=2, width=3, height=30)
    findVolume(height=30, length=2, width=3)
}

fun findVolume(length: Int, width: Int, height: Int = 10) {
    var volume = length * width * height
    println("Volume is $volume")
}
```

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
}
```

Lambda Expression

- It is just a **function with no name**.
  - 변수에 lambda expression을 할당

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

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

Which one is prefer?

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

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

High order Functions (2/2)

```
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)
}
```

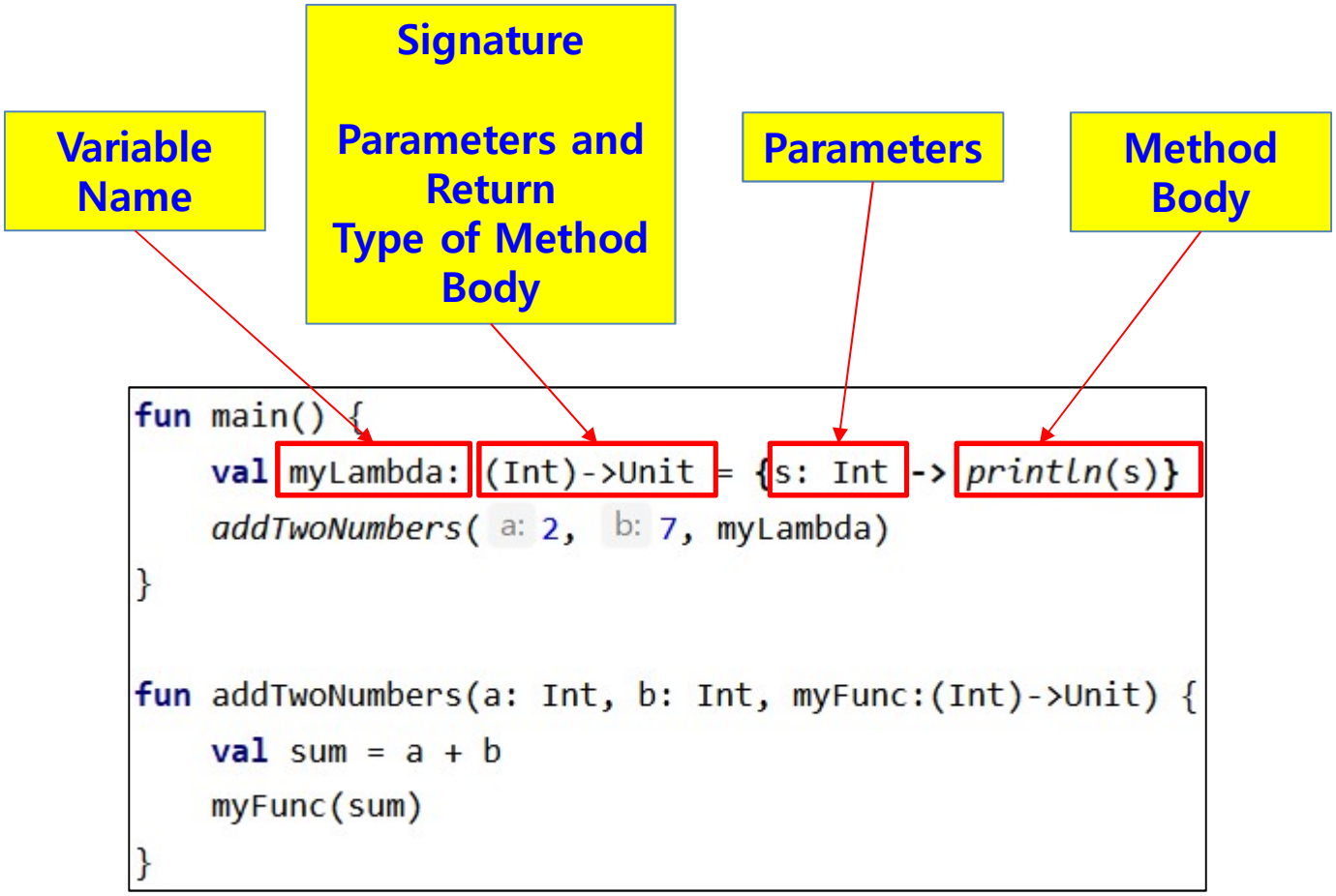
함수를 return 값으로 사용

High order Functions with Lambda expressions (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)
}
```

High order Functions with Lambda expressions (2/3)



High order Functions with Lambda expressions (3/3)

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

fun addTwoNumbers(a: Int, b: Int, myFunc:(Int, Int)-> Int) {
    val result = myFunc(a, b)
    println(result)
}
```

Anonymous functions (익명 함수)

- 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))
}
```

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

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

Inline functions

- Inline function을 호출하는 곳에 함수 본문 내용을 그대로 복사.
- Inline function은 lambda expression과 같은 형태의 매개변수를 사용해야 함
- 일반 함수 실행 방식보다 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
```

```
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);
}
```

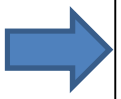
Extension Functions

- 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)

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

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



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

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

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

Extension Functions 예(2/2)

```
fun main() {
    var str1: String = "Hello, "
    var str2: String = "Kotlin!"

    var tmp = str1.add(str2)
    println(tmp)
}

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

Infix Functions

- 중위 표기법을 지원하는 함수

  $add\,(x, \mathrm{y})$ ➔ $\mathrm{x}\ add\ \mathrm{y}$

- 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 Functions 예

```
fun main() {
    var currentValue = newFunction( left: 20,  right: 20)

    currentValue increaseBy 30
    println("${currentValue.left}, ${currentValue.right}")

    currentValue decreaseBy 20
    println("${currentValue.left}, ${currentValue.right}")
}

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

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

```
50, 50
30, 30
```