02

02-3. 함수

Basic Syntax

함수 사용법

함수 선언

- fun 함수명(매개변수명 : 타입) : 리턴타입 { }

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

- 매개변수 에는 var, val을 선언할 수 없다. 매개변수는 기본으로 val이 적용
- 의미있는 반환값이 없을 때는 Unit으로 명시
- Unit은 생략할 수 있으며 함수의 반환 타입이 선언되지 않았다면 기본으로 Unit이 적용
-

```
fun sum(a: Int, b: Int): Unit {
    //.....
}
```

```
fun sum(a: Int, b: Int) {
    //.....
}
```

함수 사용법

함수 선언

- 함수내에 함수선언 가능

```
fun sum(a: Int, b: Int): Int {
    var sum=0
    fun calSum(){
        for(i in a..b){
            sum += i
        }
    }
    calSum()
    return sum
}
```

함수 사용법

함수 선언

- Single expression function
- 함수 선언 때 단일 구문으로 값을 반환하는 함수라면 중괄호 { }을 생략하고, 등호 =을 이용해 쉽게
- 정의

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

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

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

함수 사용법

함수 오버로딩

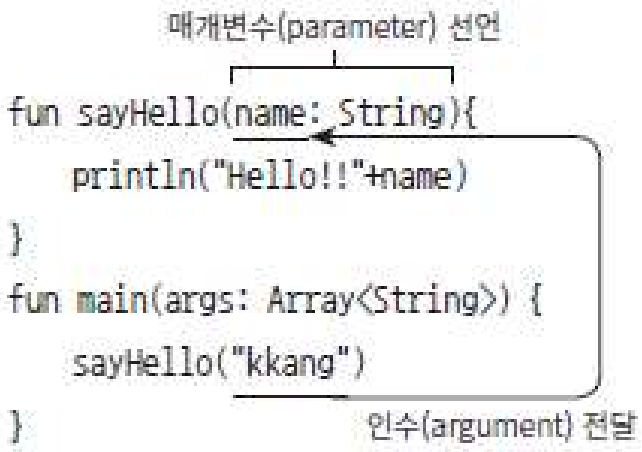
- 같은 이름의 함수를 매개변수 부분을 다르게 하여 여러 개 정의

```
fun some(a: String){
    println("some(a: String) call....")
}
fun some(a: Int){
    println("some(a: Int) call....")
}
fun some(a: Int, b: String){
    println("some(a: Int, b: String) call....")
}
```

함수 사용법

기본 인수와 명명된 인수

- 함수를 호출하면서 전달하는 값을 인수(argument)라고 부르며, 이 인수 값을 전달받는 함수의 변수를 매개변수(parameter)라고 부릅니다.



- 함수에 매개변수(parameter)가 있으면 호출하는 곳(호출자)에서는 호출받는 함수(피호출자)의 매개변수 타입과 개수에 맞추어 호출해야 합니다.

함수 사용법

기본 인수와 명명된 인수

- 호출자가 인수를 명시하지 않아도 피호출자가 알아서 기본값을 적용하게 할 수 있습니다. 이를 '기본 인수(default argument)'라고 합니다.

```
fun sayHello(name: String){
    println("Hello!!"+name)
}
```

```
fun sayHello(name: String = "kkang"){
    println("Hello!!"+name)
}
```

함수 사용법

기본 인수와 명명된 인수

- 호출자에서 전달할 값을 대입할 매개변수명을 명시하는 방법이 명명된 인수입니다

```
fun sayHello(name: String = "kkang", no: Int){
    println("Hello!!"+name)
}
fun main(args: Array<String>) {
//    sayHello(10)//error
    sayHello("lee", 20)
    sayHello(no=10)
    sayHello(name="kim", no=10)
}
```

함수 사용법

중위표현식

- infix(중위 표현식) 이란 연산자를 피 연산자의 중간에 위치시킨 다는 개념
- 'A + B'라는 구문에서 A와 B라는 피연산자 사이에 +라는 연산자를 놓을 때 "중위 표현식을 사용했다"고 표현
- 중위 표현식을 함수 호출에도 사용이 가능
- 함수를 중위 표현식으로 이용하려면 infix라는 예약어로 선언해야 합니다.

- 클래스의 맴버 함수로 선언되거나 혹은 클래스의 extension 함수인 경우
- 하나의 매개변수를 가지는 함수의 경우

함수 사용법

중위표현식

```
infix fun Int.myFun(x: Int): Int {
    return x * x
}
class FunClass {
    infix fun infixFun(a: Int){
        println("infixFun call....")
    }
}
fun main(args: Array<String>) {
    val obj=FunClass()
    obj.infixFun(10)
    //중위 표현식
    obj infixFun 10

    println(10 myFun 10)
    println(10.myFun(10))
}
```

함수 사용법

가변인수

- 함수의 매개변수 선언 부분에 vararg를 이용하여 가변 인수를 지정
- 함수당 하나의 vararg 파라미터만 허용
- 배열을 전달할 때는 스프레드 연산자(*) 필요
- 일반 파라미터와 함께 사용 가능
- vararg 파라미터는 마지막 파라미터로 오는 것이 권장됨

```
fun <T> varargsFun(a1: Int, vararg array: T){
    for( a in array){
        println(a)
    }
}

fun main(args: Array<String>) {
    varargsFun(10, "hello", "world")
    varargsFun(10, 20, false)
}
```

함수 사용법

재귀함수

- 재귀함수란 함수 내에서 자신의 함수를 다시 호출하는 것

```
fun loopPrint(no: Int = 1){
    var count=1
    while(true){
        println("loopPrint..")
        if(no == count) return
        else count++
    }
}
```

```
fun recPrint(no: Int = 1, count: Int = 1){
    println("recPrint...")
    return if(no==count) return else recPrint(no -1, count)
}
```

함수 사용법

재귀함수

- 꼬리 재귀 함수
- 재귀 호출이 함수의 마지막 연산인 경우
- 컴파일러가 이를 반복문으로 최적화할 수 있음
- 스택 오버플로우를 방지할 수 있음

```
tailrec fun tailrecPrint(no: Int = 1, count: Int = 1){
    println("tailrecPrint...")
    return if(no==count) return else tailrecPrint(no -1, count)
}
```

함수 사용법

재귀함수

- tailrec 사용 조건
- 마지막 연산이 재귀 호출이어야 함
- try/catch/finally 블록 내에서 사용할 수 없음

```
tailrec fun sum(n: Int): Int {
    if (n <= 0) return n
    else return n + sum(n - 1)
}
tailrec fun sum2(n: Int, result: Int = 0): Int {
    if (n <= 0) return result
    else return sum2(n - 1, n + result)
}
```

```
public static final int sum(int n) {
   return n <= 0 ? n : n + sum(n - 1);
}

public static final int sum2(int n, int result) {
   while(n > 0) {
      int var10000 = n - 1;
      result += n;
      n = var10000;
   }

   return result;
}
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare