Функции в Kotlin

В Kotlin функциями называют не только обычные функции, определенные за пределами класса, но и те, что находятся внутри класса, и которые в других языках принято называть методами. В Kotlin функции, определенные внутри классов, называют функциями-членами. Кроме них есть функции-расширения, которые как и функции-члены относятся к классам, но определены за их пределами как и обычные функции. Вместе функции-члены и функции-расширения можно считать методами класса.

Обычные функции и функции-расширения определяются за пределами класса. Их отличие заключается в наличии имени класса в нотации через точку перед именем функции-расширения. Например, `MyClass.itsFunc() {}`. Подробнее о методах рассказано в уроке об объектно-ориентированном программировании.

Все функции по-умолчанию, то есть без модификатора видимости, являются **`public`**. Их можно использовать отовсюду. Если функция верхнего уровня (не находится внутри класса) объявлена с модификатором **`private`**, ее можно использовать только в данном файле.

В Kotlin функции определяются через ключевое слово **`fun`**. После имени функции в круглых скобках через запятую перечисляются параметры. Каждый параметр состоит из имени переменной и через двоеточие ее типа. Параметры являются val-переменными. Ключевое слово **`val`** опускается. После круглых скобок с параметрами ставится двоеточие и указывается тип возвращаемого функцией значения. Тело функции заключается в фигурные скобки.

```
fun main() {
    val a = 8
    val b = 13

    val d = avr(a, b)

    println(d) // 10.5
}

fun avr(n1: Int, n2: Int): Float {
```

```
    return (n1 + n2).toFloat() / 2
}
```

Если тело функции состоит из одного выражения, допустимо опускать указание возвращаемого типа и оператор `return`, а вместо фигурных скобок использовать знак равенства:

```
fun avr(n1: Int, n2: Int) =
    (n1 + n2).toFloat() / 2
```

В этом случае компилятор Котлина сам выведет из выражения тип возвращаемого значения. Другой пример:

```
fun main() {
    val a = 3
    val b = 5
    val c = abNumber(a, b)
    println(c)
}

fun abNumber(n1: Int, n2: Int) =
    if (n1 > n2) {
        n1 / n2
    }
    else {
        n2 / n1
    }
```

Если функция имеет обычное тело-блок (в фигурных скобках), а не тело-выражение, и при этом ничего не возвращает, то есть в ней нет оператора `return`, то она по-умолчанию возвращает объект `Unit`, о котором не обязательно сообщать в заголовке функции.

```
fun main() {
    val a = hello()
    println(a) // kotlin.Unit
}
```

```
fun hello() {
    println("hi") // hi
}
```

Выполнение:

```
hi
kotlin.Unit
```

В Kotlin можно перегружать функции, то есть определять функции с одним и тем же именем, но разными параметрами. Однако, поскольку в Kotlin параметрам можно назначать значения по-умолчанию, необходимость в перегрузке функций во многих случаях отпадает.

```
fun main() {
    repeatStr("Hi", 3) // HiHiHi
    println()
    repeatStr("Hello") // HelloHello
}

fun repeatStr(s: String, n: Int = 2) {
    for(i in 1..n)
        print(s)
}
```

Параметры со значениями по умолчанию лучше указывать последними. Иначе, при вызове функции следует использовать так называемые именованные аргументы – обращение к параметрам функции по именам их переменных:

```
fun main() {
    repeatStr(3, "Hi") // HiHiHi
    println()
    repeatStr(s = "Hello") // HelloHello
}

fun repeatStr(n: Int = 2, s: String) {
```

```
    for(i in 1..n)
        print(s)
}
```

Вообще, именованные аргументы можно использовать в любых случаях. При этом совпадение с последовательностью параметров не обязательно:

```
fun main() {
    repeatStr(s = "Hi", n = 3) // HiHiHi
    println()
    repeatStr(n = 2, s = "Hello") // HelloHello
}

fun repeatStr(s: String, n: Int) {
    for(i in 1..n)
        print(s)
}
```

В Kotlin есть возможность объявлять функции, принимающие произвольное количество аргументов. Делается это через параметр, который объявляется с ключевым словом `vararg`.

```
fun main() {
    println(avr(0, 1, 8)) // 3.0
    println(avr(-5, 1)) // -2.0
}

fun avr(vararg nums: Int): Float {
    val sum = nums.sum()
    val l = nums.size
    return sum.toFloat() / l
}
```

Параметр `vararg` – это массив. То есть аргументы, переданные в этот параметр, упаковываются в массив элементов указанного типа. С другой стороны, если имеется уже готовый массив, а параметр функции объявлен как `vararg`, то есть

ожидает произвольное количество отдельных аргументов, мы можем распаковать массив через звездочку:

```
fun main() {
    val a: IntArray = intArrayOf(0, 1, 8)
    println(avr(*a)) // 3.0
}

fun avr(vararg nums: Int): Float {
    val sum = nums.sum()
    val l = nums.size
    return sum.toFloat() / l
}
```

Если у функции есть другие аргументы, `vararg` лучше указывать последним. В иных случаях придется использовать именованные аргументы.

PDF-версия курса с дополнительными уроками