Лямбда-выражения в Kotlin

Лямбда-выражение – это функция, записанная в виде выражения, и которую можно передавать как аргумент в другие функции. В свою очередь обычные функции могут возвращать лямбды.

Лямбда-выражения иногда называют анонимными функциями, потому что у них нет имен. Однако в Kotlin анонимные функции есть сами по себе, то есть они и лямбды – несколько разные вещи. Анонимные функции в Kotlin определяются через ключевое слово `fun` как обычные функции, хотя не имеют имени. В то же время лямбда-выражения определяются заключением их в фигурные скобки.

Рассмотрим простейшее лямбда-выражение:

```
fun main() {
    {println(1)}
}
```

Выполнение этого кода ни к чему не приведет, хотя мы могли бы ожидать появления на экране единицы. Заключив выражение `println(1)` в фигурные скобки, мы превратили его в тело функции. Пусть эта функция и лямбда, она все-таки функция, и без вызова ее тело не выполняется. В случае обычной функции, чтобы ее вызвать, надо обратиться к ней по имени и после имени поставить круглые скобки, даже если мы не передаем в функцию никаких аргументов. Аналогично делается и в случае лямбда-функции. Из-за отсутствия имени вызывать ее будем сразу после определения:

```
fun main() {
    {println(1)}()
}
```

Данный код выведет на экран единицу.

Усовершенствуем функцию так, чтобы она выводила на экран переданный ей аргумент. Лямбда-выражения, как и обычные функции, могут иметь параметры.

Они указываются в начале и отделяются от тела "стрелкой" – знаком минуса и угловой скобкой, то есть `->`.

```
fun main() {
    {n: Int -> println(n)}(10)
}
```

Лямбда-функции не только принимают аргументы, но и возвращают значения. Лямбда возвращает результат выполнения последнего выражения ее тела. В нашем случае тело состоит из одной строки. Функция `println()` возвращает в место своего вызова объект `Unit`. Этот объект лямбда передаст в место своего вызова. Мы можем присвоить объект переменной и увидеть его.

```
fun main() {

    // выведет: 10
    val a = {n: Int -> println(n)}(10)

    // выведет: kotlin.Unit
    println(a)
}
```

Тело лямбда-функции может быть сложнее, а параметров быть больше.

```
fun main() {
    val a = {n: Int, m: Int ->
        println(n + m)
        n + m
    }(10, 20)

    println(a)
}
```

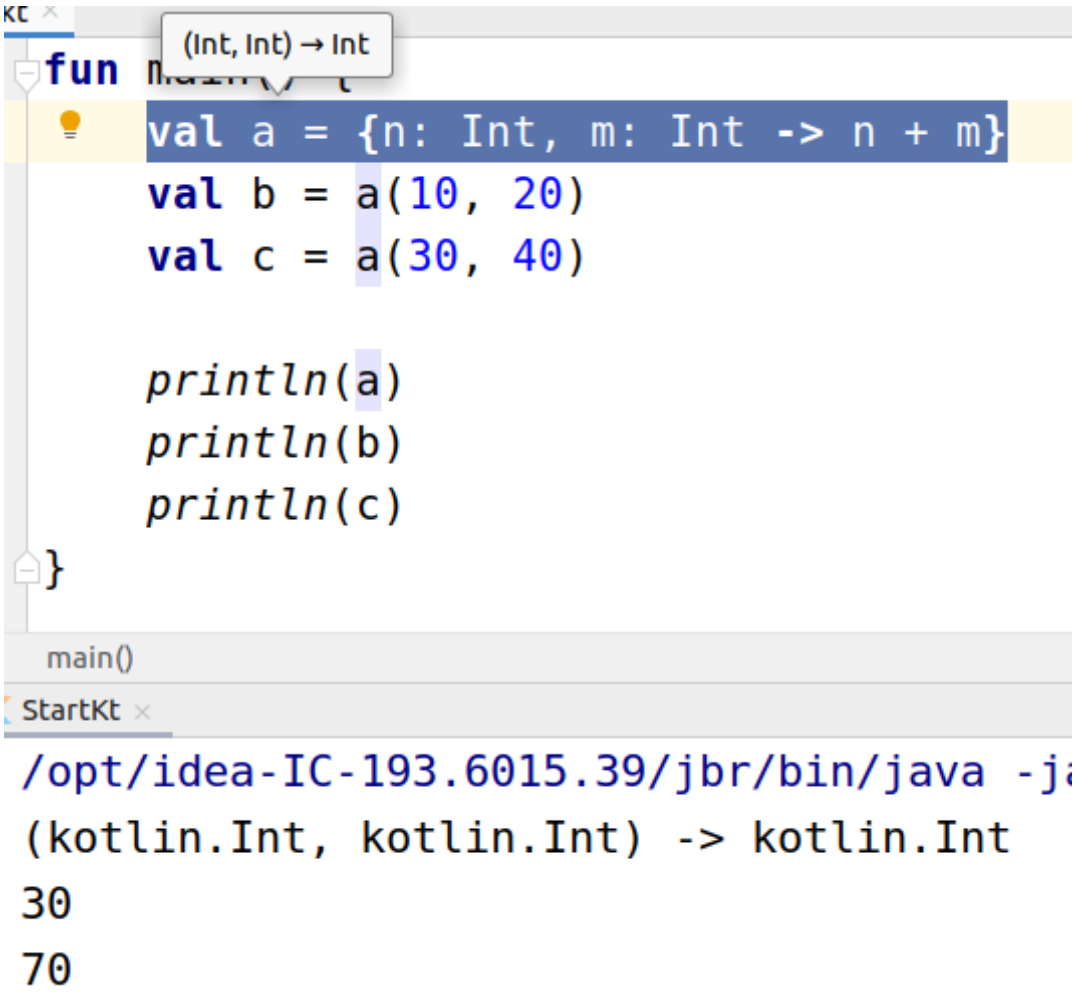
Код выше два раза выведет 30. Первое число будет результатом выполнения выражения `println(n + m)` в теле лямбды. Второе – будет присвоено переменной *a*, потому что выражение `n + m` последнее в теле лямбды и его

результат присваивается переменной *a*, значение которой потом выводится на экран.

В данном случае если мы захотим повторно использовать лямбда-функцию, но уже с другими аргументами, придется ее снова переписывать. Однако ничего не мешает присваивать переменным сами лямбда-выражения. Тогда у такой функции появляется имя, с помощью которого ее можно вызывать как обычную функцию:

```
fun main() {
    val a = {n: Int, m: Int -> n + m}
    val b = a(10, 20)
    val c = a(30, 40)

    println(a)
    println(b)
    println(c)
}
```



Переменные *b* и *c* будут иметь целочисленный тип, потому что им присваивается результат выполнения функции, то есть то, что из нее было возвращено, а не сама функция. Переменной *a* присваивается функция. Такая переменная имеет функциональный тип – особую разновидность объектов. В данном случае тип функциональной переменной определен как `(Int, Int) -> Int`. Это значит, что она принимает два целочисленных аргумента и возвращает число.

При описании конкретного функционального типа сначала в скобках указывается тип параметров, после стрелки – тип возвращаемого значения. Как и любой другой тип, компилятор Котлина может вывести функциональный тип сам из определения функции. Однако мы можем указать его вручную. При этом записывать типы в самом определении лямбды необязательно.

```
val a: (Int, Int) -> Int = {n, m -> n + m}
```

В то же время подобное имеет смысл, если мы хотим "документировать" функцию, сделать понятными для программиста назначение параметров.

Вернемся к варианту лямбда-функции с одним параметром:

```
fun main() {
    val a = {n: Any -> println(n)}
    a(10)
    a(30)
}
```

Если параметр только один, в Котлин он по-умолчанию присваивается встроенному идентификатору `it`. Используя его, можно не объявлять параметр. Уже само наличие в теле `it` будет говорить, что у лямбды один параметр. При этом, чтобы было понятно, какого типа этот параметр, функциональный тип переменной придется указать явно.

```
val a: (Any) -> Unit = {println(it)}
```

В других ситуациях Kotlin способен сам вывести тип, что называется, из контекста. Подобное происходит при вызове методов коллекций, принимающих в качестве

одного из аргументов лямбда-выражения. Там без явных указаний `it` становится типом элементов коллекций.

PDF-версия курса с дополнительными уроками