Лекция #6. Функции в Kotlin

Функции являются основой многих языков программирования. Проще говоря, функция позволяет определить блок кода, который выполняет определенную задачу. Затем, если приложению требуется выполнить данную задачу, можно вызвать функцию вместо того, чтобы копировать и вставлять одинаковый код.

Создание новой функции в Kotlin

Представьте, что у вас есть приложение, которому часто требуется выводить ваше имя. Для этого можно создать функцию:

```
1  fun printMyName() {
2      println("Меня зовут Алексей Плотников!")
3  }
4
5  fun main(args : Array<String>) {
6      printMyName() // Вызываем функцию
7  }
```

Приведенный выше код известен как объявление функции.

Функция определяется с помощью ключевого слова **fun**. После этого следует название функции и круглые скобки.

Подробнее о назначении этих скобок поговорим позже.

После круглых скобок следует открывающая фигурная скобка, за которой идет код, который нужно запустить в функции, а затем закрывающая скобка. После определения функции ее можно вызвать следующим образом в главной функции **main()**:

```
1  printMyName()
```

Вывод будет следующим:

```
Меня зовут Алексей Плотников!
```

В прошлом мы уже использовали функции. Функция **println** выводит результат в консоль.

Параметры функции в Kotlin

В предыдущем примере, функция просто выводит на экран сообщение. Это здорово, но зачастую для функции требуется настроить параметры, с помощью которых она будет работать по-разному в зависимости от данных, которые в нее передаются.

В качестве примера рассмотрим следующую функцию:

```
fun printMultipleOfFive(value: Int) {
    println("$value * 5 = ${value * 5}")
}

fun main(args : Array<String>) {
    printMultipleOfFive(10)
}
```

Здесь дается определение одного параметра **value** типа **Int** в круглых скобках после названия функции. В любой функции круглые скобки содержат так называемый список параметров. Эти круглые скобки необходимы как при объявлении, так и при вызове функции, даже если список параметров пуст.

Данная функция выведет любое заданное число умноженное на пять. В этом примере вызывается функция с аргументом 10, поэтому вывод будет следующим:

```
10 * 5 = 50
```

На заметку: Не путайте термины «**параметр**» и «**аргумент**». Функция объявляет свои параметры в списке параметров. При вызове функции вы предоставляете значения в качестве аргументов для параметров функции.

Можно написать функцию более обобщенной. С двумя параметрами, функция выведет результат умножение любых двух значений.

```
fun printMultipleOf(multiplier: Int, andValue: Int) {
    println("$multiplier * $andValue = ${multiplier * andValue}")
}

fun main(args : Array<String>) {
    printMultipleOf(4, 2)
}
```

Теперь в скобках после названия функции есть два параметра — **multiplier** и **andValue**, оба типа **Int**.

Иногда при вызове функции будет правильнее использовать именованные аргументы, это упрощает понимание назначения каждого аргумента.

```
printMultipleOf(multiplier = 4, andValue = 2)
```

Теперь при вызове функции очевидно, для чего нужны аргументы. Это особенно полезно, когда у функции несколько параметров.

Вы также можете назначить параметрам значения по умолчанию:

```
fun printMultipleOf(multiplier: Int, value: Int = 1) {
    println("$multiplier * $value = ${multiplier * value}")
}

fun main(args : Array<String>) {
    printMultipleOf(4)
}
```

Мы добавили **= 1** после второго параметра. Это значит, что при отсутствии значения для второго параметра значение по умолчанию будет равно 1.

Таким образом, вывод кода будет следующим:

```
4 * 1 = 4
```

Иметь значение по умолчанию может быть полезно, когда вы ожидаете, что у параметра будет одно конкретное значение большую часть времени, и это упростит код при вызове функции.

Функции и оператор возврата return в Kotlin

Все рассматриваемые до сих пор функции выполняли простую задачу — они что-то выводили в консоль. Функции также могут возвращать значение. При вызове функции можно присвоить возвращаемое значение переменной или константе или использовать его непосредственно в **if** или **when** выражениях в качестве значения для проверки.

Это означает, что функцию можно использовать для управления данными. Данные просто принимаются через параметры, они изменяются внутри функции, а затем возвращаются.

Определить функцию, возвращающую значение, можно следующим образом:

```
fun multiply(number: Int, multiplier: Int): Int {
    return number * multiplier
}

fun main(args : Array<String>) {
    // Присваиваем переменной значение из функции.
    var data = multiply(2, 10)
    println(data)
}
```

Для объявления возвращаемого типа функции, добавляется **:**, за которым следует тип возвращаемого значения после круглых скобках и перед открывающейся фигурной скобкой. В этом примере функция возвращает тип **Int**.

Внутри функции используется оператор **return** для возврата значения. В этом примере возвращается результат от умножения двух параметров.

Также можно вернуть несколько значений с помощью **Pair**:

```
fun multiplyAndDivide(number: Int, factor: Int): Pair<Int, Int> {
    return Pair(number * factor, number / factor)
}

fun main(args : Array<String>) {
    val (product, quotient) = multiplyAndDivide(4, 2)
    println("Результат умножения: $product")
    println("Результат деления: $quotient")
}
```

Данная функция возвращает результат от умножения и деления двух параметров в виде **Pair**, содержащую два **Int** значения.

Если функция состоит только из одного выражения, вы можете присвоить данное выражение функции через использование **=**, в таком случае, у нас будут отсутствовать фигурные скобки, возвращаемый тип и оператор **return**:

```
fun multiplyInferred(number: Int, multiplier: Int) = number * multiplier

fun main() {
    val result = multiplyInferred(2, 4)
    println(result)
}
```

В таком случае тип возвращаемого функцией значения выводится как тип выражения, присвоенного функции. В примере выше возвращаемым типом является **Int**, так как **number** и **multiplier** также принадлежат к типу **Int**.

Параметры в качестве значений в Kotlin

Параметры функции по умолчанию являются константами, а это значит, что их изменить нельзя.

Рассмотрим следующим пример кода:

```
fun incrementAndPrint(value: Int) {
    value += 1
    print(value)
}

fun main() {
    incrementAndPrint(2)
}
```

Результатом выполнения данного кода будет ошибка: **Val cannot be reassigned**

Значение из параметра **value** эквивалентно константе, объявленной с помощью **val**, и поэтому его нельзя переназначить. По этой причине, когда функция пытается увеличить его, компилятор выдает ошибку.

Обычно такое поведение ожидаемо. В идеале функция не меняет свои параметры. Если это так, то вы не можете быть уверены в значениях параметров и можете сделать неверные предположения касательно кода, что приведет к ошибочным данным при их использовании.

Если нужно, чтобы функция изменила параметр и вернула его, вы должны сделать это косвенно, объявив новую переменную следующим образом:

```
fun incrementAndPrint(value: Int): Int {
    val newValue = value + 1
    println(newValue)
    return newValue
}

fun main() {
    incrementAndPrint(2)
}
```

На заметку: Как вы увидите в будущем, при добавлении параметров к основному конструктору при определении класса вы действительно добавляете **var** или **val** к параметрам. Это делается для указания, что параметры являются свойствами класса и что их значение может или не может быть изменено.

Перегрузка функций в Kotlin

Что делать, если вам нужно использовать несколько функций с одним и тем же названием?

```
fun getValue(value: Int): Int {
  return value + 1
}

fun getValue(value: String): String {
  return "Значение равно: $value"
}
```

Это перегрузка, которая дает возможность определить похожие функции, используя одинаковое название для них, НО с разными типами параметров.

Однако, компилятор по-прежнему должен видеть разницу между данными функциями внутри текущей области видимости. Каждый раз при вызове функции должно быть понятно, какая функция будет выполняться.

Обычно это достигается через разницу в списке параметров:

- Разное количество параметров;
- Разные типы у параметров.

На заметку: Одного возвращаемого типа недостаточно для различия двух функций. Т.е. если одна функция возвращает **Int** а другая возвращает строку — этого будет недостаточно, чтобы компилятор понял какую из них вызвать ведь тип у параметров один и тот же.

К примеру, такое определение двух методов приведет к ошибке:

```
fun getValue(value: String): String {
  return "Полученное значение: $value"
}

fun getValue(value: String): Int {
    // Ошибка: конфликт перегрузки функции
    return value.length
}
```

У методов выше одинаковые названия, типы параметров и количество параметров. **Kotlin** не может различить их.

Стоит отметить, что перегрузку нужно использовать осторожно. Используйте перегрузку только для тех функций, поведение которых похоже, но из за разных типов у параметров у них будет разная обработка данных внутри функции.

Функции как переменные в Kotlin

Функции в **Kotlin** являются просто **ещё одним типом данных**. Их можно присваивать переменным и константам как и значения любого другого типа вроде **Int** или **String**.

Рассмотрим следующую функцию:

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

Данная функция принимает два параметра и возвращает сумму их значений.

Вы можете присвоить данную функцию переменной через использование метода создания ссылки **::** следующим образом:

```
var function = ::add
```

Здесь названием переменной является **function** и ее тип выводится как (**Int, Int**) **-> Int** из присвоенной функции add. Переменная **function** состоит из типа функции, которая принимает два параметра **Int** и возвращает **Int**.

Теперь можно использовать переменную **function** так же, как использовалась бы функция **add**:

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

fun main() {
    var function = ::add
    print( function(4, 2) )
}
```

Теперь рассмотрим следующий код:

```
fun subtract(a: Int, b: Int) : Int {
    return a - b
}
```

Здесь объявляется другая функция, которая принимает два параметра **Int** и возвращает **Int**. Вы можете назначить переменную **function**, используемую ранее, на новую функцию **subtract**, потому что список параметров и тип возвращаемого значения **subtract** совместимы с типом переменной **function**.

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

fun subtract(a: Int, b: Int) : Int {
    return a - b
}

fun main() {
    var function = ::add
    function = ::subtract
    print( function(4, 2) )
}
```

На этот раз вызов **function** возвращает 2.

Назначение функций переменным может быть очень удобным, потому что таким образом вы можете передавать функции другим функциям. Далее дан показательный этому пример:

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

fun printResult(function: (Int, Int) -> Int, a: Int, b: Int) {
    val result = function(a, b)
    print(result)
}

fun main() {
    printResult(::add, 4, 2)
}
```

Функция **printResult** принимает три параметра:

- **function** типа функции, которая принимает два параметра **Int** и возвращает **Int**, объявление выглядит следующих образом **(Int, Int) -> Int**;
- **a** с типом **Int**;
- **b** с типом **Int**.

Функция **printResult** вызывает переданную функцию из **function**, передавая в нее параметры **a** и **b**. Затем, полученный результат выводится в консоль.

Возможность передавать функции другим функциям очень полезна. Вы можете не только передавать данные для различных манипуляций с ними, но и функции в качестве параметров. Это также означает, что вы можете гибко выбирать какой именно код будет выполняться.

Присваивание функций переменным и передача функций в качестве аргументов — это один из аспектов **функционального программирования**.

Невозвращаемые функции в Kotlin

Это может показаться запутанным, но мы все-таки рассмотрим пример функции, которая предназначена для сбоя работы приложения. Если приложение собирается работать с поврежденными данными, зачастую лучше завершить работу вместо продолжения работы в неизвестном и потенциально опасном состоянии.

Другим примером невозвращаемой функции является функция, которая обрабатывает цикл событий. Цикл событий лежит в основе каждого современного приложения, которое принимает данные от пользователя и отображает их на экране. Цикл событий обслуживает запросы, поступающие от пользователя, а затем передает эти события в код приложения, который вызывает отображение информации на экране. Затем цикл возвращается назад и обслуживает следующее запросы.

Эти циклы событий часто запускаются в приложении путем вызова функции, которая никогда не возвращается. Если вы начинаете разрабатывать приложения для **Android**, помните об этом, когда столкнетесь с основным потоком, также известным как поток **UI**, или поток пользовательского интерфейса.

У **Kotlin** есть способ сообщить компилятору, что функция никогда не вернет значение. Вы устанавливаете тип возвращаемого значения функции на тип **Nothing**, указывая на то, что функция никогда ничего не возвращает.

Грубая, но честная реализация невозвращаемой функции будет выглядеть следующим образом:

```
1  fun infiniteLoop(): Nothing {
2      while (true) {
3
4      }
5  }
```

Вам может быть интересно, что такого в этом специальном возвращаемом типе. Компилятор зная, что функция никогда не вернет значение, может внести определенную оптимизацию при генерации кода для вызова функции.

По сути, вызывающий функцию код ни о чем не беспокоится. Причина в том, что коду известно, что эта функция никогда не закончится до завершения приложения.

Создание **хороших** функций в Kotlin

Есть много способов решить проблемы при помощи функций. Лучшие (то есть самые простые в использовании и понимании) функции решают **одну простую задачу** и не пытаются сделать что-то еще. Это упрощает их сборку в более сложные модели поведения. У хороших функций есть четкий набор входных данных, которые каждый раз приводят к одному и тому же результату. Это облегчает процесс проверки правильности выполнения кода. Помните об этих практических правилах при создании функций.