типы функций и лямбда-выражения в Котлине

8. <https://developer.android.com/courses/pathways/android-basics-compose-unit-2-pathway-3>

В Котлине функции считаются конструкциями первого класса. Это означает, что функции можно рассматривать как тип данных. Вы можете хранить функции в переменных, передавать их другим функциям в качестве аргументов и возвращать их из других функций.

Как и другие типы данных, которые можно выразить с помощью литеральных значений (например, тип Int значения 10 и тип String значения "Hello"), вы также можете объявлять литералы функций, которые называются **лямбда-выражениями**.

## [3. Сохраните функцию в переменной](https://developer.android.com/codelabs/basic-android-kotlin-compose-function-types-and-lambda?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-function-types-and-lambda#2)

До сих пор вы научились объявлять функции с помощью ключевого слова fun. Можно вызвать функцию, объявленную с помощью ключевого слова fun, что приведет к выполнению кода в теле функции.

Как конструкции первого класса, функции также являются типами данных, поэтому вы можете хранить функции в переменных, передавать их функциям и возвращать их из функций. Возможно, вам нужна возможность изменять поведение части вашего приложения во время выполнения или вкладывать компонуемые функции для создания макетов. Все это стало возможным благодаря лямбда-выражениям.

Сохраните функцию в переменной:

1. Перейдите на [Kotlin Playground](https://developer.android.com/training/kotlinplayground" \t "_blank) .
2. После функции main()определите  функцию без параметров trick() и возвращаемого значения, которая печатает "No treats!". Синтаксис такой же, как и у других функций, которые вы видели ранее.

fun main() {  
      
}  
  
fun trick() {  
    println("No treats!")  
}

1. В теле функции main() создайте переменную с именем trickFunction и присвойте ей значение trick. Вы не включаете круглые скобки после trick,  потому что хотите сохранить функцию в переменной, а не вызывать ее.

fun main() {  
    val trickFunction = trick  
}  
  
fun trick() {  
    println("No treats!")  
}

1. Запустите свой код. Это выдает ошибку, поскольку компилятор Kotlin распознает  имя функции trick, но ожидает, что вы вызовете функцию trick(), а не присвоите ее переменной.

Function invocation 'trick()' expected

Вы пытались сохранить trick в  переменной trickFunction. Однако, чтобы ссылаться на функцию как на значение, вам необходимо использовать оператор ссылки на функцию ( ::). Синтаксис проиллюстрирован на этом изображении:



1. Чтобы обратиться к функции как к значению, переназначьте trickFunction ее  ::trick.

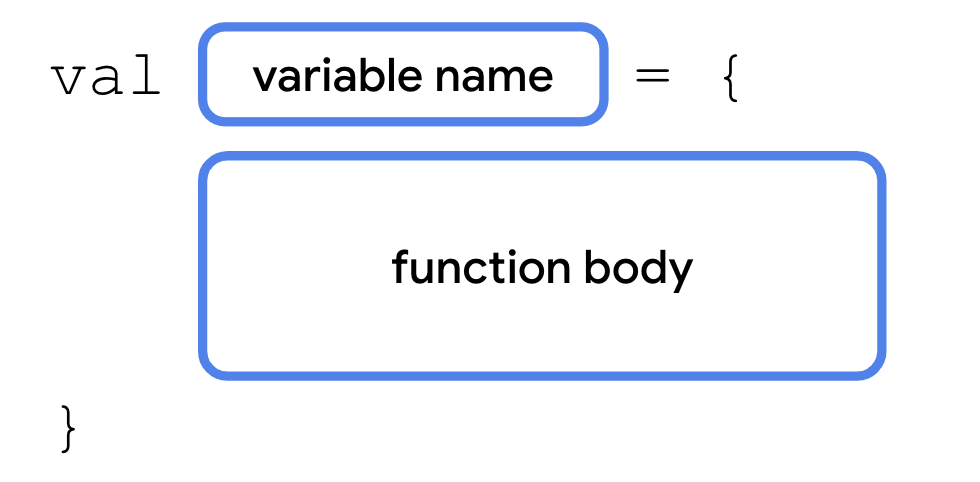
fun main() {  
    val trickFunction = ::trick  
}  
  
fun trick() {  
    println("No treats!")  
}

1. Запустите код, чтобы убедиться, что ошибок больше нет. Вы видите предупреждение, о том, что trickFunction никогда не используется, но оно исправлено дальше.

## Переопределите функцию с помощью лямбда-выражения

Лямбда-выражения предоставляют краткий синтаксис для определения функции без  ключевого слова fun. Вы можете сохранить лямбда-выражение непосредственно в переменной без ссылки на другую функцию.

Перед оператором присваивания ( =) вы добавляете ключевое слово valили var, за которым следует имя переменной, которую вы используете при вызове функции. После оператора присваивания ( =) следует лямбда-выражение, состоящее из пары фигурных скобок, образующих тело функции. Синтаксис проиллюстрирован на этом изображении:



Когда вы определяете функцию с помощью лямбда-выражения, у вас есть переменная, которая ссылается на функцию. Вы также можете присвоить его значение другим переменным, как и любому другому типу, и вызвать функцию с именем новой переменной.

Обновите код, чтобы использовать лямбда-выражение:

1. Перепишите  функцию trick(), используя лямбда-выражение. Имя trick теперь относится к имени переменной. Тело функции в фигурных скобках теперь является лямбда-выражением.

fun main() {  
    val trickFunction = ::trick  
}  
  
val trick = {  
    println("No treats!")  
}

1. В main()функции удалите оператор ссылки на функцию ( ::), поскольку trickтеперь он относится к переменной, а не к имени функции.

fun main() {  
    val trickFunction = trick  
}  
  
val trick = {  
    println("No treats!")  
}

1. Запустите свой код. Ошибок нет, и вы можете обращаться к trick()функции без оператора ссылки на функцию ( ::). Вывода нет, потому что вы еще не вызвали функцию.
2. В  функции main() вызовите  функцию trick(), но на этот раз включите круглые скобки, как если бы вы вызывали любую другую функцию.

fun main() {  
    val trickFunction = trick  
    trick()  
}  
  
val trick = {  
    println("No treats!")  
}

1. Запустите свой код. Тело лямбда-выражения выполняется.

No treats!

1. В  функции main() вызывайте  переменную trickFunction, как если бы это была функция.

fun main() {  
    val trickFunction = trick  
    trick()  
    trickFunction()  
}  
  
val trick = {  
    println("No treats!")  
}

1. Запустите свой код. Функция вызывается дважды: один раз для trick()вызова функции и второй раз для trickFunction()вызова функции.

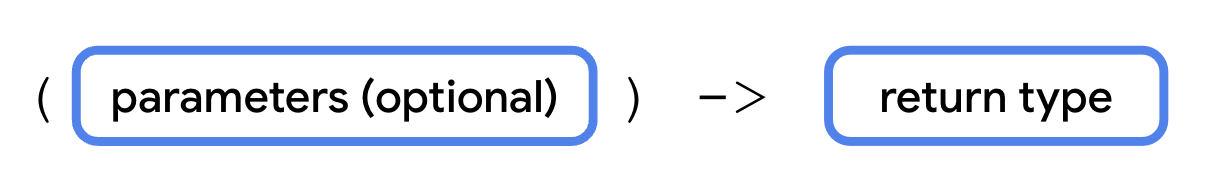
No treats!

No treats!

С помощью лямбда-выражений вы можете создавать переменные, в которых хранятся функции, вызывать эти переменные как функции и сохранять их в других переменных, которые можно вызывать как функции.

## [4. Используйте функции в качестве типа данных](https://developer.android.com/codelabs/basic-android-kotlin-compose-function-types-and-lambda?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-function-types-and-lambda#3)

На предыдущей лабораторной работе вы узнали, что в Kotlin есть вывод типов. Когда вы объявляете переменную, вам часто не нужно явно указывать тип. В предыдущем примере компилятор Kotlin смог сделать вывод, что значение trick является функцией. Однако если вы хотите указать тип параметра функции или тип возвращаемого значения, вам необходимо знать синтаксис выражения типов функций. Типы функций состоят из набора круглых скобок, содержащих список необязательных параметров, символ ->и тип возвращаемого значения. Синтаксис проиллюстрирован на этом изображении:



Тип данных  объявленной ранее переменной trick будет () -> Unit. Круглые скобки пусты, поскольку функция не имеет параметров. Тип возвращаемого значения Unit обусловлен  тем, что функция ничего не возвращает. Если бы у вас была функция, которая принимала два  параметра Int и возвращала Int, ее тип данных был бы (Int, Int) -> Int.

Объявите другую функцию с лямбда-выражением, которое явно указывает тип функции:

1. После  переменной trick объявите переменную с именем treat равной лямбда-выражению с телом, которое печатает "Have a treat!".

val trick = {  
    println("No treats!")  
}  
  
val treat = {  
    println("Have a treat!")  
}

1. Укажите  тип данных переменной treat как () -> Unit.

val treat: () -> Unit = {  
    println("Have a treat!")  
}

1. В  функции main() вызовите  функцию treat().

fun main() {  
    val trickFunction = trick  
    trick()  
    trickFunction()  
    treat()  
}

1. Запустите код. Функция treat() ведет себя как  функция trick(). Обе переменные имеют один и тот же тип данных, хотя только  переменная treat объявляет его явно.

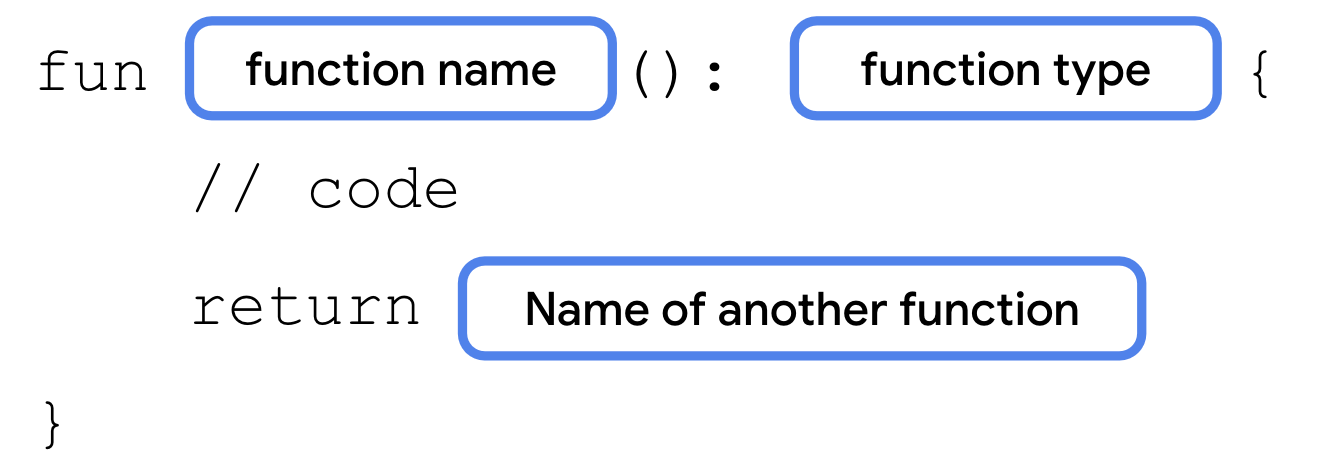
No treats!

No treats!

Have a treat!

## Используйте функцию в качестве возвращаемого типа

Функция — это тип данных, поэтому ее можно использовать как любой другой тип данных. Вы даже можете возвращать функции из других функций. Синтаксис проиллюстрирован на этом изображении:



Создайте функцию, которая возвращает функцию.

1. Удалите код из  функции main().

fun main() {  
      
}

1. После  функции main() определите  функцию trickOrTreat(), которая принимает параметр isTrick типа Boolean.

fun main() {  
      
}  
  
fun trickOrTreat(isTrick: Boolean): () -> Unit {  
}  
  
val trick = {  
    println("No treats!")  
}  
  
val treat = {  
    println("Have a treat!")  
}

1. В тело функции trickOrTreat() добавьте  оператор if, который возвращает  функцию trick(), если isTrick =  true, и возвращает  функцию treat(), если isTrick ложна.

fun trickOrTreat(isTrick: Boolean): () -> Unit {  
    if (isTrick) {  
        return trick  
    } else {  
        return treat  
    }  
}

1. В  функции main() создайте переменную с именем treatFunction и присвойте ей результат вызова trickOrTreat(), передав в качестве параметра isTrick значение false. Затем создайте вторую переменную с именем trickFunction и присвойте ей результат вызова trickOrTreat(), на этот раз передав его в качестве параметра true.

fun main() {  
    val treatFunction = trickOrTreat(false)  
    val trickFunction = trickOrTreat(true)  
}

1. Вызовите treatFunction(), а затем позвоните trickFunction()на следующей линии.

fun main() {  
    val treatFunction = trickOrTreat(false)  
    val trickFunction = trickOrTreat(true)  
    treatFunction()  
    trickFunction()  
}

1. Запустите свой код. Вы должны увидеть вывод для каждой функции. Даже если вы не вызывали функции trick()или treat()напрямую, вы все равно можете вызывать их, поскольку вы сохраняете возвращаемые значения каждый раз, когда вы вызываете функцию trickOrTreat(), и вызываете функции с переменными trickFunctionи treatFunction.

Have a treat!

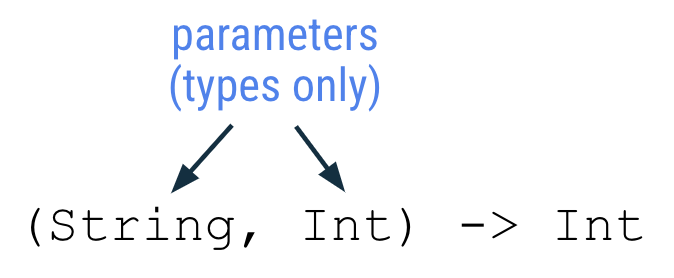
No treats!

Теперь вы знаете, как функции могут возвращать другие функции. Вы также можете передать функцию в качестве аргумента другой функции. Возможно, вы хотите предоставить функции какое-то собственное поведение, trickOrTreat()чтобы она делала что-то кроме возврата одной из двух строк. Функция, которая принимает другую функцию в качестве аргумента, позволяет передавать другую функцию каждый раз, когда она вызывается.

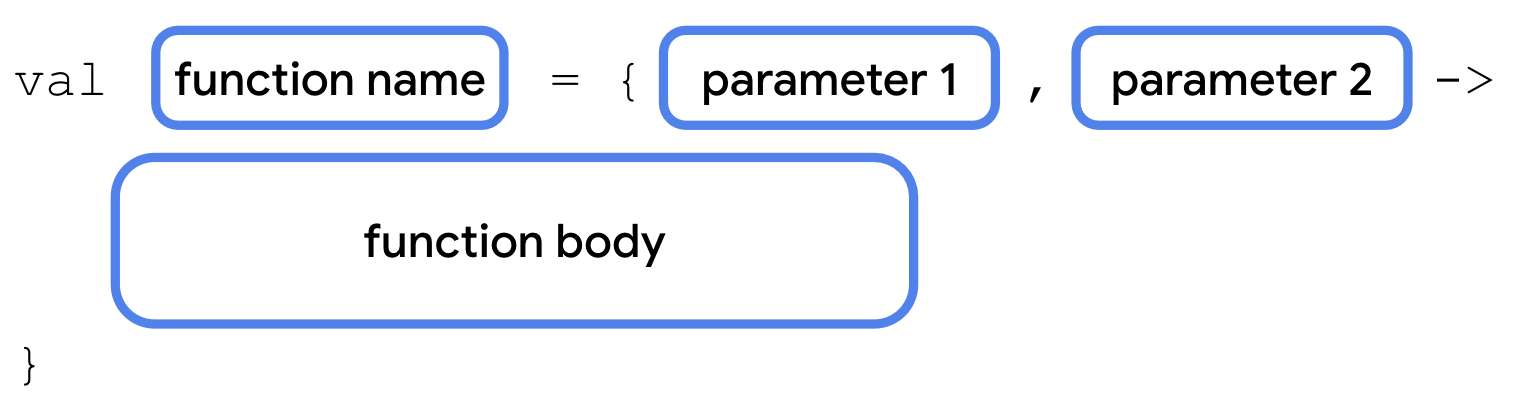
## ****Передайте функцию другой функции в качестве аргумента****

Вы измените свою функцию trickOrTreat(), чтобы разрешить предоставление в качестве аргумента дополнительного удовольствия, представленного функцией.

Функция, которая  использует в качестве параметра trickOrTreat(), также должна принимать собственный параметр. При объявлении типов функций параметры не помечаются. Вам нужно только указать типы данных каждого параметра через запятую. Синтаксис проиллюстрирован на этом изображении:



Когда вы пишете лямбда-выражение для функции, принимающей параметр, параметрам присваиваются имена в том порядке, в котором они встречаются. Имена параметров перечисляются после открывающих фигурных скобок, и каждое имя отделяется запятой. Стрелка ( ->) отделяет имена параметров от тела функции. Синтаксис проиллюстрирован на этом изображении:



Обновите trickOrTreat()функцию, чтобы она принимала функцию в качестве параметра:

1. После isTrickпараметра добавьте extraTreatпараметр типа (Int) -> String.

fun trickOrTreat(isTrick: Boolean, extraTreat: (Int) -> String): () -> Unit {

1. В elseблоке перед returnоператором call println(), передавая вызов функции extraTreat(). Перейдите 5к вызову extraTreat().

fun trickOrTreat(isTrick: Boolean, extraTreat: (Int) -> String): () -> Unit {  
    if (isTrick) {  
        return trick  
    } else {  
        println(extraTreat(5))  
        return treat  
    }  
}

1. Теперь, когда вы вызываете trickOrTreat()функцию, вам нужно определить функцию с лямбда-выражением и передать его в качестве extraTreatпараметра. В main()функции перед вызовом функции trickOrTreat()добавьте coins()функцию. Функция coins()присваивает Intпараметру имя quantityи возвращает String. Вы можете заметить отсутствие ключевого слова return, которое нельзя использовать в лямбда-выражениях. Вместо этого результатом последнего выражения в функции становится возвращаемое значение.

fun main() {  
    val coins: (Int) -> String = { quantity ->  
        "$quantity quarters"  
    }  
      
    val treatFunction = trickOrTreat(false)  
    val trickFunction = trickOrTreat(true)  
    treatFunction()  
    trickFunction()  
}

1. После coins()функции добавьте cupcake()функцию, как показано. Назовите Intпараметр quantityи отделите его от тела функции с помощью ->оператора. Теперь вы можете передать функцию coins()или cupcake()в trickOrTreat()функцию.

fun trickOrTreat(isTrick: Boolean, extraTreat: (Int) -> String): () -> Unit {  
    if (isTrick) {  
        return trick  
    } else {  
        println(extraTreat(5))  
        return treat  
    }  
}

fun main() {  
    val coins: (Int) -> String = { quantity ->  
        "$quantity quarters"  
    }  
  
    val cupcake: (Int) -> String = { quantity ->  
        "Have a cupcake!"  
    }  
  
    val treatFunction = trickOrTreat(false)  
    val trickFunction = trickOrTreat(true)  
    treatFunction()  
    trickFunction()  
}

1. В cupcake()функции удалите quantityпараметр и ->символ. Они не используются, поэтому их можно опустить.

val cupcake: (Int) -> String = {  
    "Have a cupcake!"  
}

**Примечание.** В coins()функции Intпараметр имеет имя quantity. Однако ему можно назвать что угодно, лишь бы имя параметра и имя переменной в строке были одинаковыми.

1. Обновите вызовы функции trickOrTreat(). Для первого вызова, когда isTrickis false, передайте функцию coins(). Для второго вызова, когда isTrickis true, передайте функцию cupcake().

fun main() {  
    val coins: (Int) -> String = { quantity ->  
        "$quantity quarters"  
    }  
  
    val cupcake: (Int) -> String = {  
        "Have a cupcake!"  
    }  
  
    val treatFunction = trickOrTreat(false, coins)  
    val trickFunction = trickOrTreat(true, cupcake)  
    treatFunction()  
    trickFunction()  
}

1. Запустите свой код. Функция extraTreat()вызывается только в том случае, если isTrickпараметру присвоено значение falseаргумента, поэтому выходные данные включают 5 четвертаков, но не кексы.

5 quarters

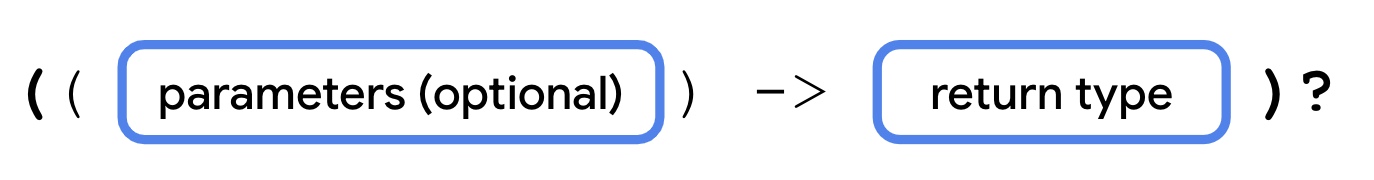
Have a treat!

No treats!

## Типы функций, допускающие значение NULL

Как и другие типы данных, типы функций могут быть объявлены как допускающие значение NULL. В этих случаях переменная может содержать функцию или быть null.

Чтобы объявить функцию допускающей значение NULL, заключите тип функции в круглые скобки, за которыми следует символ ?вне конечной скобки. Например, если вы хотите сделать () -> Stringтип допускающим значение NULL, объявите его как (() -> String)?тип. Синтаксис проиллюстрирован на этом изображении:



Сделайте extraTreatпараметр обнуляемым, чтобы вам не приходилось предоставлять extraTreat()функцию каждый раз, когда вы ее вызываете trickOrTreat():

1. Измените extraTreatтип параметра на (() -> String)?.

fun trickOrTreat(isTrick: Boolean, extraTreat: ((Int) -> String)?): () -> Unit {

1. Измените вызов функции extraTreat(), чтобы использовать ifоператор для вызова функции только в том случае, если она не равна нулю. Теперь функция trickOrTreat()должна выглядеть следующим образом:

fun trickOrTreat(isTrick: Boolean, extraTreat: ((Int) -> String)?): () -> Unit {  
    if (isTrick) {  
        return trick  
    } else {  
        if (extraTreat != null) {  
            println(extraTreat(5))  
        }  
        return treat  
    }  
}

1. Удалите cupcake()функцию, а затем замените cupcakeаргумент на nullво втором вызове функции trickOrTreat().

fun main() {  
    val coins: (Int) -> String = { quantity ->  
        "$quantity quarters"  
    }  
  
    val treatFunction = trickOrTreat(false, coins)  
    val trickFunction = trickOrTreat(true, null)  
    treatFunction()  
    trickFunction()  
}

1. Запустите свой код. Вывод должен остаться неизменным. Теперь, когда вы можете объявлять типы функций как допускающие значение NULL, вам больше не нужно передавать функцию для extraTreatпараметра.

5 quarters

Have a treat!

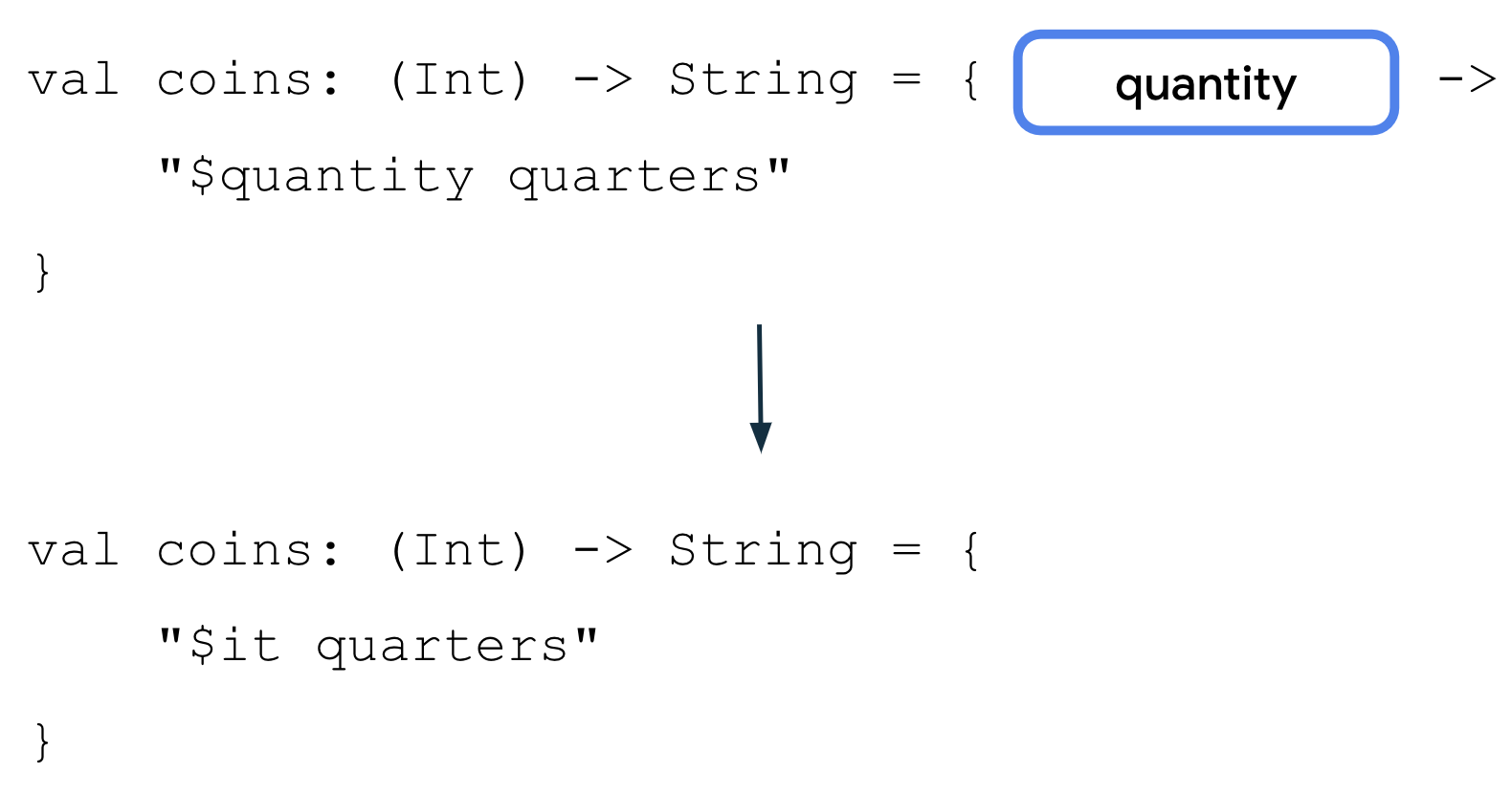
No treats!

## [5. Напишите лямбда-выражения с сокращенным синтаксисом.](https://developer.android.com/codelabs/basic-android-kotlin-compose-function-types-and-lambda?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-function-types-and-lambda#4)

Лямбда-выражения предоставляют множество способов сделать ваш код более кратким. В этом разделе вы изучите некоторые из них, поскольку большинство лямбда-выражений, с которыми вы сталкиваетесь и пишете, написаны с использованием сокращенного синтаксиса.

## Опустить имя параметра

Когда вы писали coins()функцию, вы явно указали имя параметра quantityфункции Int. Однако, как вы видели на примере этой cupcake()функции, вы можете полностью опустить имя параметра. Если функция имеет единственный параметр и вы не указываете имя, Kotlin неявно присваивает ей имя it, поэтому вы можете опустить имя и ->символ параметра, что делает ваши лямбда-выражения более краткими. Синтаксис проиллюстрирован на этом изображении:



Обновите coins()функцию, чтобы использовать сокращенный синтаксис для параметров:

1. В coins()функции удалите quantityимя и ->символ параметра.

val coins: (Int) -> String = {  
    "$quantity quarters"  
}

1. Измените "$quantity quarters"шаблон строки, чтобы он ссылался на один параметр, используя $it.

val coins: (Int) -> String = {  
    "$it quarters"  
}

1. Запустите свой код. Котлин распознает itимя параметра Intи по-прежнему печатает количество четвертей.

5 quarters

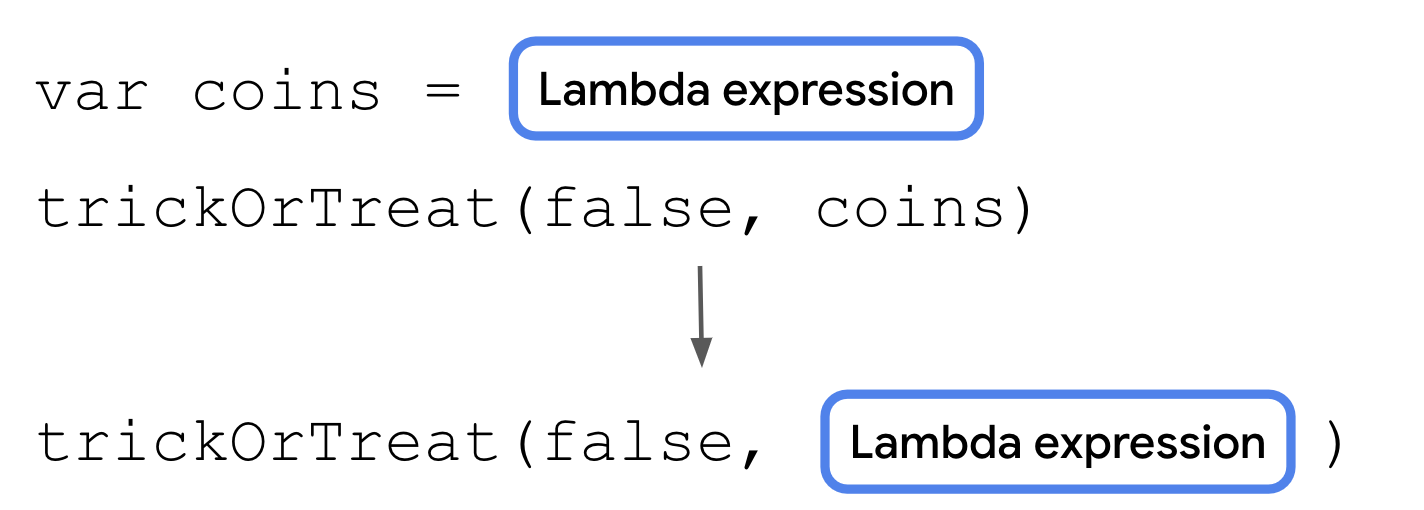
Have a treat!

No treats!

## Передайте лямбда-выражение непосредственно в функцию

В настоящее время функция coins()используется только в одном месте. Что, если бы вы могли просто передать лямбда-выражение непосредственно в trickOrTreat()функцию без необходимости предварительного создания переменной?

Лямбда-выражения — это просто функциональные литералы, точно так же, 0как целочисленный литерал или "Hello"строковый литерал. Вы можете передать лямбда-выражение непосредственно в вызов функции. Синтаксис проиллюстрирован на этом изображении:



Измените код, чтобы можно было удалить coinsпеременную:

1. Переместите лямбда-выражение так, чтобы оно передавалось непосредственно в вызов функции trickOrTreat(). Вы также можете сократить лямбда-выражение до одной строки.

fun main() {  
    val coins: (Int) -> String = {  
        "$it quarters"  
    }  
    val treatFunction = trickOrTreat(false, { "$it quarters" })  
    val trickFunction = trickOrTreat(true, null)  
    treatFunction()  
    trickFunction()  
}

1. Удалите coinsпеременную, поскольку она больше не используется.

fun main() {  
    val treatFunction = trickOrTreat(false, { "$it quarters" })  
    val trickFunction = trickOrTreat(true, null)  
    treatFunction()  
    trickFunction()  
}

1. Запустите код. Он по-прежнему компилируется и работает как положено.

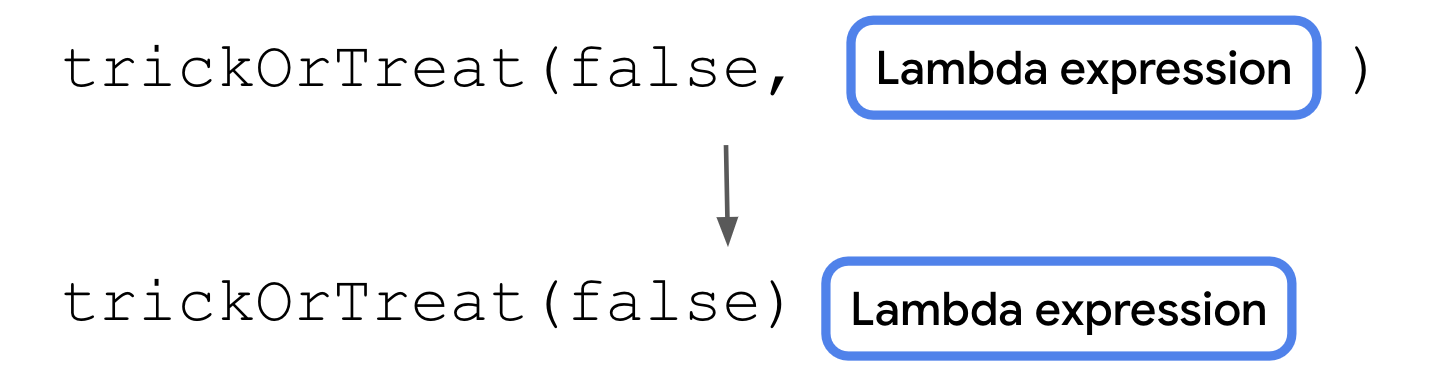
5 quarters

Have a treat!

No treats!

## ****Используйте конечный синтаксис лямбда****

Вы можете использовать другой вариант сокращения для записи лямбда-выражений, когда тип функции является последним параметром функции. Если это так, вы можете поместить лямбда-выражение после закрывающей скобки, чтобы вызвать функцию. Синтаксис проиллюстрирован на этом изображении:



Это делает ваш код более читабельным, поскольку отделяет лямбда-выражение от других параметров, но не меняет того, что делает код.

Обновите код, чтобы использовать синтаксис завершающего лямбда-выражения:

1. В treatFunctionпеременной переместите лямбда-выражение {"$it quarters"}после закрывающей скобки при вызове trickOrTreat().

val treatFunction = trickOrTreat(false) { "$it quarters" }

1. Запустите свой код. Все по-прежнему работает!

5 quarters

Have a treat!

No treats!

**Примечание.** Составные функции, которые вы использовали для объявления пользовательского интерфейса, принимают функции в качестве параметров и обычно вызываются с использованием завершающего синтаксиса лямбда.

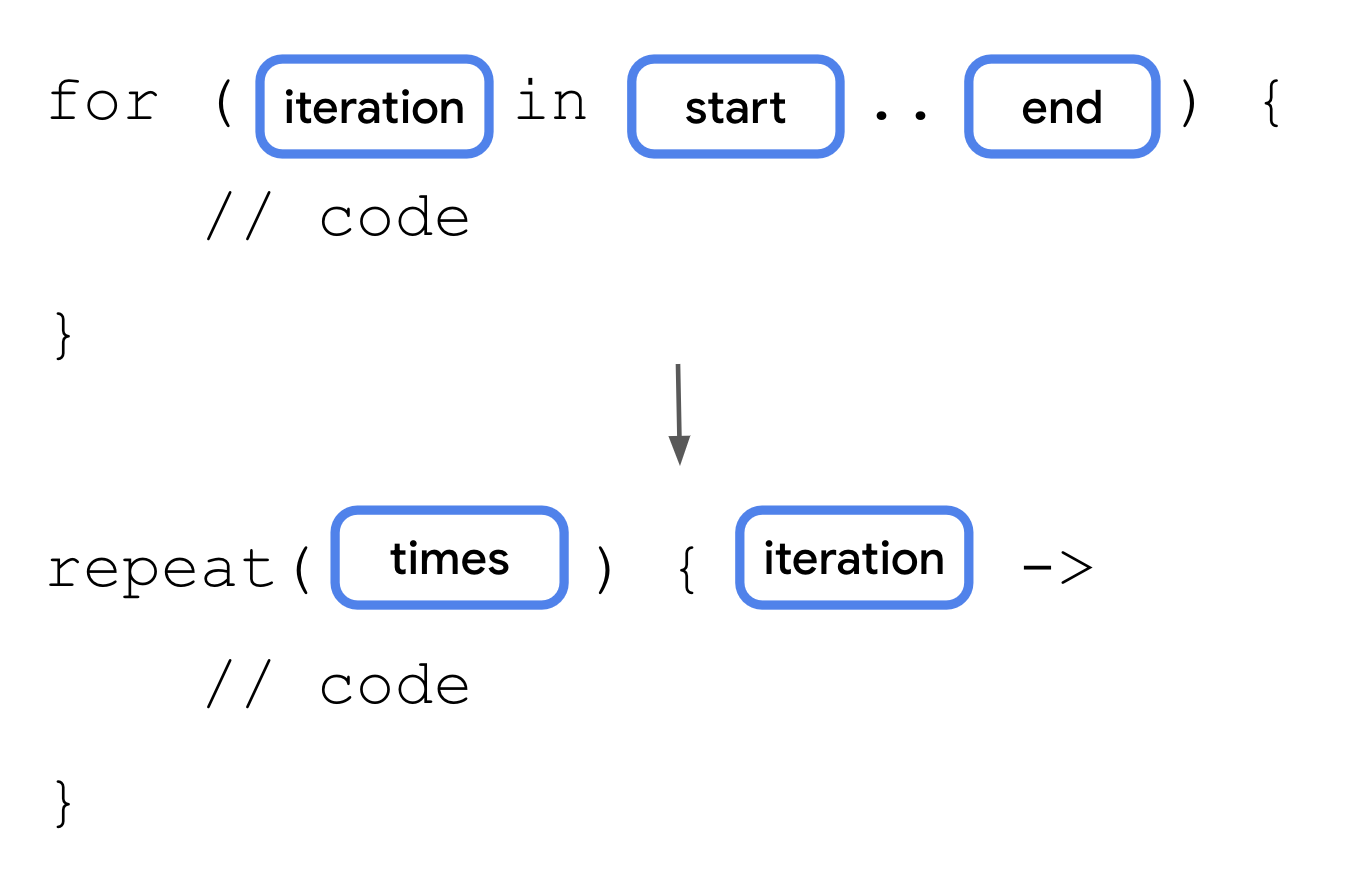
## [6. Используйте функцию повтора().](https://developer.android.com/codelabs/basic-android-kotlin-compose-function-types-and-lambda?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-2-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-function-types-and-lambda#5)

Когда функция возвращает функцию или принимает функцию в качестве аргумента, она называется функцией высшего порядка. Функция trickOrTreat()является примером функции высшего порядка, поскольку она принимает функцию ((Int) -> String)?типа в качестве параметра и возвращает функцию () -> Unitтипа. Kotlin предоставляет несколько полезных функций высшего порядка, которыми вы можете воспользоваться благодаря своим новым знаниям о лямбда-выражениях.

Функция [repeat()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/repeat.html" \t "_blank)является одной из таких функций высшего порядка. Функция repeat()— это краткий способ выразить forцикл с помощью функций. Вы часто используете эту и другие функции более высокого порядка в более поздних модулях. Функция repeat()имеет следующую сигнатуру:

repeat(times: Int, action: (Int) -> Unit)

Параметр times— это количество раз, которое должно произойти действие. Параметр action— это функция, которая принимает один Intпараметр и возвращает Unitтип. Параметр actionфункции Int— это количество раз, которое действие выполнялось на данный момент, например аргумент 0для первой итерации или 1аргумент для второй итерации. Вы можете использовать эту repeat()функцию для повторения кода указанное количество раз, аналогично циклу for. Синтаксис проиллюстрирован на этом изображении:



Вместо того, чтобы вызывать trickFunction()функцию только один раз, вы можете вызывать ее с помощью repeat()функции несколько раз.

Обновите свой код «кошелек или жизнь», чтобы увидеть функцию repeat()в действии:

1. В main()функции вызовите функцию repeat()между вызовами treatFunction()и trickFunction(). Передайте 4параметр timesи используйте конечный лямбда-синтаксис для actionфункции. Вам не нужно указывать имя для Intпараметра лямбда-выражения.

fun main() {  
    val treatFunction = trickOrTreat(false) { "$it quarters" }  
    val trickFunction = trickOrTreat(true, null)  
    treatFunction()  
    trickFunction()  
    repeat(4) {  
          
    }  
}

1. Переместите вызов treatFunction()функции в repeat()лямбда-выражение функции.

fun main() {  
    val treatFunction = trickOrTreat(false) { "$it quarters" }  
    val trickFunction = trickOrTreat(true, null)  
    repeat(4) {  
        treatFunction()  
    }  
    trickFunction()  
}

1. Запустите свой код. Строка "Have a treat"должна быть напечатана четыре раза.

5 quarters

Have a treat!

Have a treat!

Have a treat!

Have a treat!

No treats!

**Краткое содержание**

* Функции в Kotlin — это конструкции первого класса, и их можно рассматривать как типы данных.
* Лямбда-выражения предоставляют сокращенный синтаксис для записи функций.
* Вы можете передавать типы функций в другие функции.
* Вы можете вернуть тип функции из другой функции.
* Лямбда-выражение возвращает значение последнего выражения.
* Если метка параметра опущена в лямбда-выражении с одним параметром, на него ссылается идентификатор it.
* Лямбды можно писать встроенными без имени переменной.
* Если последний параметр функции является типом функции, вы можете использовать синтаксис завершающего лямбда-выражения, чтобы переместить лямбда-выражение после последней скобки при вызове функции.
* Функции высшего порядка — это функции, которые принимают другие функции в качестве параметров или возвращают функцию.
* Функция repeat()представляет собой функцию более высокого порядка, работающую аналогично циклу for.