Создание и использование функций в Kotlin

Перед началом работы

В одном из предыдущих уроках вы видели простую программу, которая печатала «Hello, world!». В программах, которые вы написали до сих пор, вы видели две функции:

- функцию `main()`, которая необходима в каждой программе на Kotlin. Она является точкой входа, или стартовой точкой, программы. функцию `println()`, которую вы вызывали из `main()` для вывода текста.

В этом уроке вы узнаете больше о функциях.

Функции позволяют разбить код на многократно используемые части, а не включать все в `main()`. Функции - это важный строительный блок приложений для Android, и изучение их определения и использования - это важный шаг на пути к становлению разработчика Android.

Необходимые условия

- Знание основ программирования на Kotlin, включая переменные, функции println() и main().

Что вы узнаете

- Как определять и вызывать собственные функции.
- Как возвращать из функции значения, которые можно хранить в переменной.
- Как определять и вызывать функции с несколькими параметрами.
- Как вызывать функции с именованными аргументами.
- Как устанавливать значения по умолчанию для параметров функции.

Что вам понадобится

Компьютер с доступом к Idea

Определение и вызов функции

Прежде чем приступить к углубленному изучению функций, давайте рассмотрим базовую терминологию.

Объявление (или определение) функции использует ключевое слово **fun** и включает в себя код в фигурных скобках, который содержит инструкции, необходимые для выполнения задачи. Вызов функции приводит к выполнению всего кода, содержащегося в этой функции. До сих пор вы писали весь свой код в функции `main()`. На самом деле функция `main()` не вызывается нигде в вашем коде; компилятор Kotlin использует ее в качестве отправной точки. Функция `main()` предназначена только для включения в нее другого кода, который вы хотите выполнить, например, вызовов функции `println()`.

Функция `println()` является частью языка Kotlin. Однако вы можете определять свои собственные функции. Это позволяет повторно использовать код, если вам нужно вызвать его несколько раз. Возьмем для примера следующую программу.

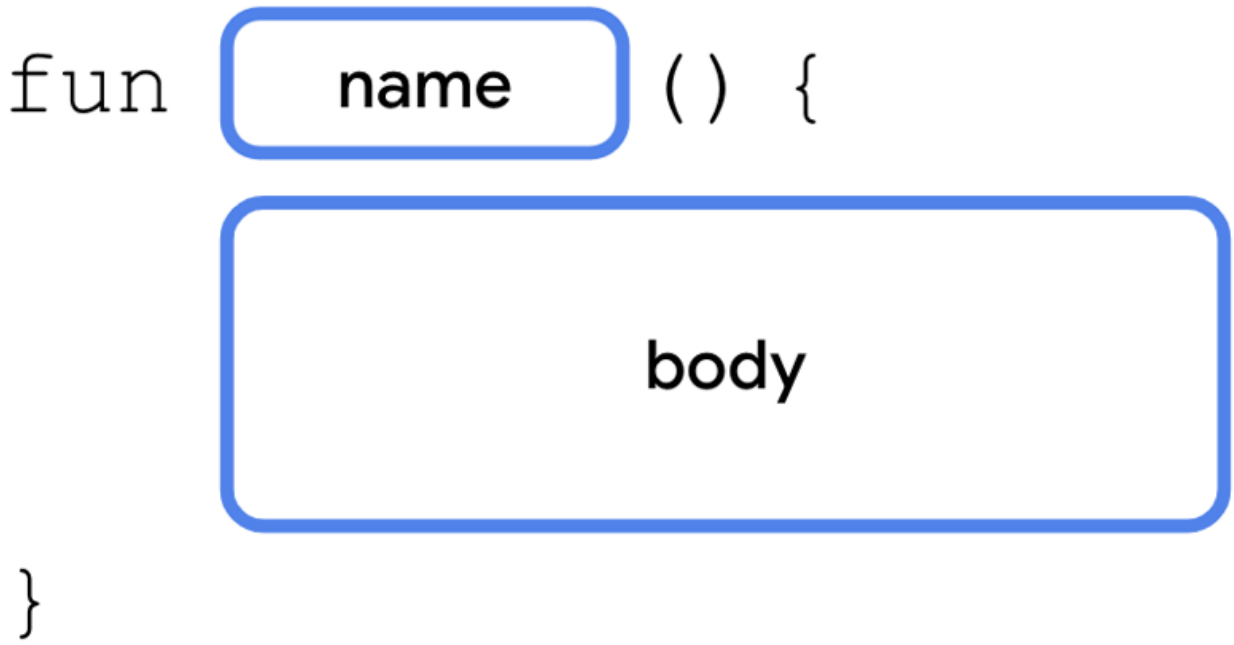
```
fun main() {
    println(«С днем рождения, Ровер!»)
    println(«Тебе уже 5 лет!»)
}
```

Функция `main()` состоит из двух операторов `println()` - один поздравляет Ровера с днем рождения, а другой сообщает возраст Ровера.

Хотя Kotlin позволяет поместить весь код в функцию `main()`, вы не всегда можете этого захотеть. Например, если вы хотите, чтобы ваша программа содержала также новогоднее поздравление, в функцию main придется включить и эти вызовы println(). Или, возможно, вы хотите поприветствовать Ровера несколько раз. Вы можете просто скопировать и вставить код, а можете создать отдельную функцию для поздравления с днем рождения. Вы сделаете последнее. Создание отдельных функций для конкретных задач имеет ряд преимуществ.

Многократно используемый код: Вместо того чтобы копировать и вставлять код, который нужно использовать несколько раз, вы можете просто вызвать функцию там, где это необходимо. Удобство чтения: Если функции выполняют одну и только одну конкретную задачу, это помогает другим разработчикам и членам команды, а также вам самим в будущем точно знать, что делает тот или иной фрагмент кода.

Синтаксис определения функции показан на следующей схеме.



Определение функции начинается с ключевого слова **fun**, за которым следует имя функции, набор открывающих и закрывающих круглых скобок, а также набор открывающих и закрывающих фигурных скобок. В фигурных скобках содержится код, который будет выполняться при вызове функции.

Вы создадите новую функцию, чтобы перенести два оператора `println()` из функции `main()`.

- откройте Idea и в новом проекте замените его содержимое на следующий код.

```
fun main() {
    println(«С днем рождения, Ровер!»)
    println(«Тебе уже 5 лет!»)
}
```

После функции `main()` определите новую функцию с именем `birthdayGreeting()`. Эта функция объявляется с тем же синтаксисом, что и функция `main()`.

```
fun main() {
    println(«С днем рождения, Ровер!»)
    println(«Тебе уже 5 лет!»)
}

fun birthdayGreeting() {

}
```

- Переместите два оператора `println()` из `main()` в фигурные скобки функции `birthdayGreeting()`.

```
fun main() {

}

fun birthdayGreeting() {
    println(«С днем рождения, Ровер!»)
    println(«Тебе уже 5 лет!»)
}
```

- В функции `main()` вызовите функцию `birthdayGreeting()`. Ваш готовый код должен выглядеть следующим образом:

```
fun main() {
    birthdayGreeting()
}

fun birthdayGreeting() {
    println(«С днем рождения, Ровер!»)
    println(«Тебе уже 5 лет!»)
}
```

- Запустите ваш код. Вы должны увидеть следующий результат:

```
С днем рождения, Ровер!
Тебе исполнилось 5 лет!
```

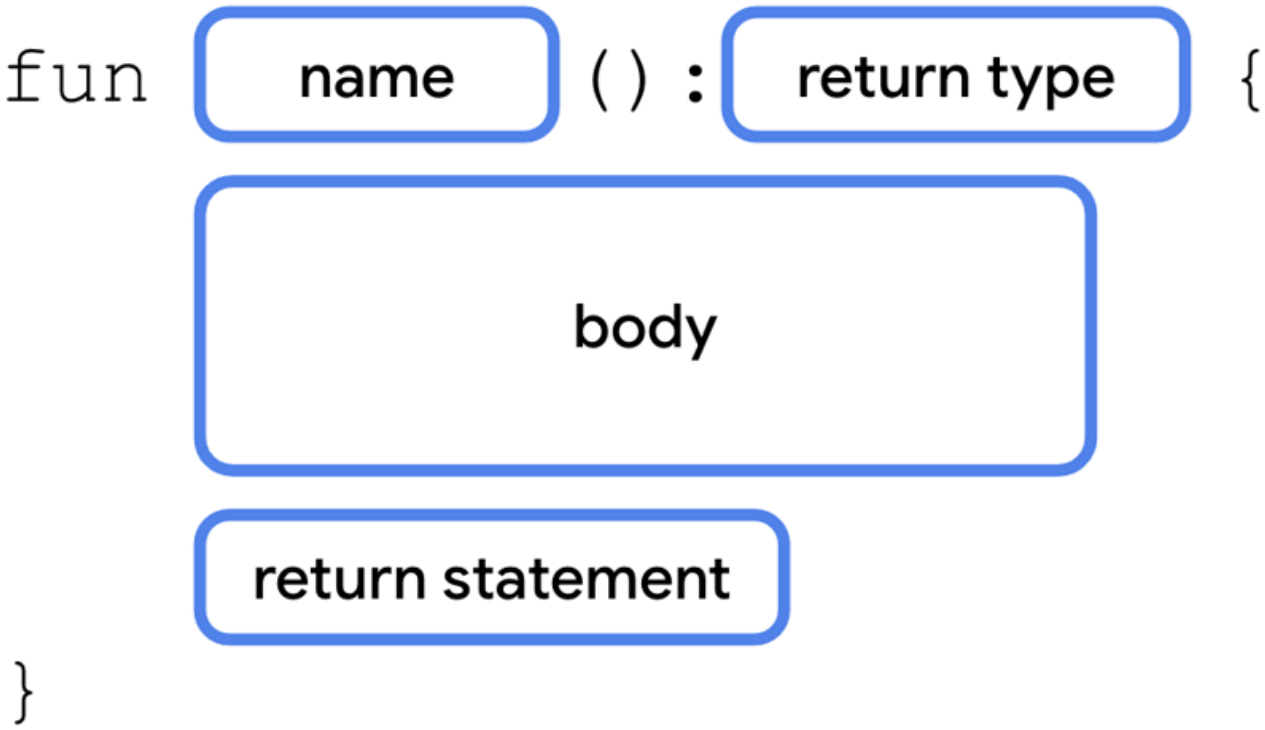
Возврат значения из функции

В более сложных приложениях функции делают больше, чем просто выводят текст.

Функции Kotlin также могут генерировать данные, называемые возвращаемым значением, которые хранятся в переменной, которую вы можете использовать в других местах вашего кода.

При определении функции вы можете указать тип данных значения, которое она должна возвращать. Тип возвращаемого значения задается двоеточием (😃 после круглых скобок, одним пробелом, а затем именем типа (Int, String и т. д.). Затем между возвращаемым типом и открывающей фигурной скобкой ставится одинарный пробел. В теле функции после всех операторов используется оператор return для указания значения, которое должна вернуть функция. Оператор return состоит из ключевого слова return, за которым следует значение (например, переменная), которое функция должна вернуть в качестве выхода.

Синтаксис объявления функции с типом return выглядит следующим образом.



Тип Unit

По умолчанию, если вы не указываете тип возврата, по умолчанию используется тип **Unit**. Unit означает, что функция не возвращает значение. Unit эквивалентен типу возврата void в других языках

(void в Java и C; Void/пустой кортеж () в Swift; None в Python и т. д.). Любая функция, которая не возвращает значение, неявно возвращает Unit. Вы можете убедиться в этом, изменив свой код так, чтобы он возвращал Unit.

В объявлении функции `birthdayGreeting()` добавьте двоеточие после закрывающей скобки и укажите возвращаемый тип как Unit.

```
fun main() {
    birthdayGreeting()
}

fun birthdayGreeting(): Unit {
    println(«С днем рождения, Ровер!»)
    println(«Тебе уже 5 лет!»)
}
```

- Запустите код и убедитесь, что все по-прежнему работает.

```
С днем рождения, Ровер!
Тебе исполнилось 5 лет!
```

В Kotlin необязательно указывать тип возврата Unit. Для функций, которые ничего не возвращают или возвращают Unit, оператор return не нужен.

> Примечание: Вы снова встретитесь с типом Unit, когда будете знакомиться с функцией Kotlin под названием lambdas в одном из следующих уроках.

Возврат строки из функции birthdayGreeting()

Чтобы продемонстрировать, как функция может возвращать значение, вы измените функцию birthdayGreeting(), чтобы она возвращала строку, а не просто выводила результат.

Замените тип возвращаемого значения Unit на String.

```
fun birthdayGreeting(): String {
    println(«С днем рождения, Ровер!»)
    println(«Тебе уже 5 лет!»)
}
```

- Запустите ваш код. Вы получите ошибку. Если вы объявляете возвращаемый тип для функции (например, String), то эта функция должна содержать оператор return.

```
Выражение 'return' требуется в функции с телом блока ('{...}')
```

- Из функции можно вернуть только одну строку, а не две. Замените операторы println() двумя переменными, nameGreeting и ageGreeting, используя ключевое слово val. Поскольку вы удалили вызов println() из функции birthdayGreeting(), вызов birthdayGreeting() ничего не выведет.

```
fun birthdayGreeting(): String {
    val nameGreeting = «С днем рождения, Ровер!»
    val ageGreeting = «Тебе уже 5 лет!»
}
```

Используя синтаксис форматирования строк, который вы изучили в предыдущем codelab, добавьте оператор return, чтобы вернуть из функции строку, состоящую из обоих поздравлений.

Чтобы отформатировать приветствия в отдельной строке, вам также нужно использовать символ \n. Он похож на символ \», о котором вы узнали в предыдущем уроке. Символ \n заменяется на новую строку, чтобы два приветствия располагались в отдельной строке.

```
fun birthdayGreeting(): String {
    val nameGreeting = «С днем рождения, Ровер!»
    val ageGreeting = «Тебе уже 5 лет!»
    return «$nameGreeting\n$ageGreeting»
}
```

В main(), поскольку функция birthdayGreeting() возвращает значение, вы можете сохранить результат в строковой переменной. Объявите переменную greeting с помощью val для хранения результата вызова функции birthdayGreeting().

```
fun main() {
    val greeting = birthdayGreeting()
}
```

В функции main() вызовите println(), чтобы вывести строку поздравления. Теперь функция main() должна выглядеть следующим образом.

```
fun main() {
    val greeting = birthdayGreeting()
    println(greeting)
}
```

Запустите свой код и посмотрите, что результат будет таким же, как и раньше. Возврат значения позволяет хранить результат в переменной, но как вы думаете, что произойдет, если вызвать функцию birthdayGreeting() внутри функции println()?

```

```

```
С днем рождения, Ровер!
Тебе исполнилось 5 лет
```

- Удалите переменную, а затем передайте результат вызова функции birthdayGreeting() в функцию println():

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

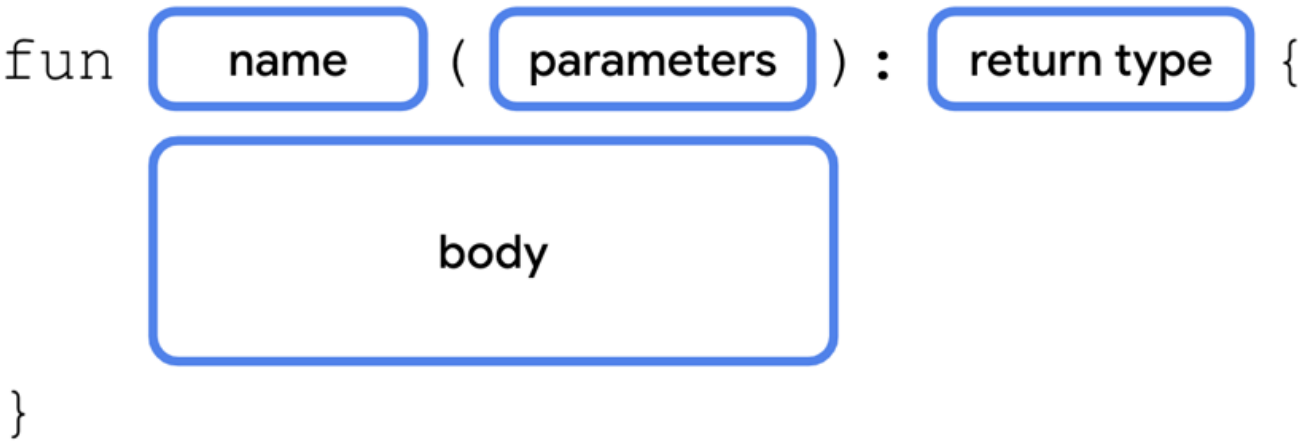
- Запустите код и посмотрите на вывод. Возвращаемое значение вызова функции birthdayGreeting() передается непосредственно в println().

```
С днем рождения, Ровер!
Тебе исполнилось 5 лет!
```

Добавление параметра в функцию birthdayGreeting()

Как вы видели, при вызове функции println() вы можете заключить строку в круглые скобки или передать значение в функцию. То же самое можно сделать и с функцией birthdayGreeting(). Однако сначала вам нужно добавить параметр в функцию birthdayGreeting().

Параметр - это имя переменной и тип данных, которые вы можете передать в функцию в качестве данных для доступа к ним внутри функции. Параметры объявляются в круглых скобках после имени функции.



Каждый параметр состоит из имени переменной и типа данных, разделенных двоеточием и пробелом. Несколько параметров разделяются запятой.

Сейчас функция birthdayGreeting() может использоваться только для приветствия Rover. Вы добавите параметр в функцию birthdayGreeting(), чтобы можно было приветствовать любое имя, которое вы передадите в функцию.

- В круглых скобках функции birthdayGreeting() добавьте параметр name типа String, используя синтаксис name: String.

```
fun birthdayGreeting(name: String): String {
    val nameGreeting = «С днем рождения, Ровер!»
    val ageGreeting = «Тебе уже 5 лет!»
    return «$nameGreeting\n$ageGreeting»
}
```

Параметр, определенный в предыдущем шаге, работает как переменная, объявленная с помощью ключевого слова val. Его значение может быть использовано в любом месте функции birthdayGreeting(). В одном из предыдущих кодовых заданий вы узнали о том, как можно вставить значение переменной в строку.

- Замените Rover в строке nameGreeting символом $, за которым следует параметр name.

```
fun birthdayGreeting(name: String): String {
    val nameGreeting = «С днем рождения, $name!»
    val ageGreeting = «Вам уже 5 лет!»
    return «$nameGreeting\n$ageGreeting»
}
```

- Запустите код и понаблюдайте за ошибкой. Теперь, когда вы объявили параметр name, вам нужно передать String при вызове функции birthdayGreeting(). Когда вы вызываете функцию, принимающую параметр, вы передаете ей аргумент. Аргумент - это значение, которое вы передаете, например «Rover».

```
Не передано значение для параметра 'name'
```

- Передайте «Rover» в вызов birthdayGreeting() в main().

```
fun main() {
    println(birthdayGreeting(«Rover»))
}
```

- Запустите код и посмотрите на результат. Имя Rover получено из параметра name.

```
С днем рождения, Ровер!
Тебе исполнилось 5 лет!
```

- Поскольку функция birthdayGreeting() принимает параметр, вы можете вызвать ее с именем, отличным от Rover. Добавьте еще один вызов birthdayGreeting() внутри вызова println(), передав в качестве аргумента «Rex».

```
println(birthdayGreeting(«Rover»))
println(birthdayGreeting(«Rex»))
```

- Запустите код еще раз и обратите внимание, что вывод отличается в зависимости от аргумента, переданного в birthdayGreeting().

```
С днем рождения, Ровер!
Тебе исполнилось 5 лет!
С днем рождения, Рекс!
Тебе исполнилось 5 лет!
```

> Примечание: Несмотря на то, что эти понятия часто используются как взаимозаменяемые, параметр и аргумент - это не одно и то же. Когда вы определяете функцию, вы определяете параметры, которые должны быть переданы ей при вызове. Когда вы вызываете функцию, вы передаете аргументы для параметров. Параметры - это переменные, доступные функции, например переменная имени, а аргументы - это фактические значения, которые вы передаете, например строка «Rover».

> Предупреждение: В отличие от некоторых языков, например Java, где функция может изменить значение, переданное в параметр, параметры в Kotlin неизменяемы. Вы не можете переназначить значение параметра из тела функции.

Функции с несколькими параметрами

Ранее вы добавили параметр для изменения приветствия на основе имени. Однако для функции можно задать более одного параметра, даже параметры разных типов данных. В этом разделе вы измените приветствие так, чтобы оно также менялось в зависимости от возраста собаки.

Определения параметров разделяются запятыми. Аналогично, когда вы вызываете функцию с несколькими параметрами, передаваемые аргументы также разделяются запятыми. Давайте посмотрим на это в действии.

После параметра name добавьте в функцию birthdayGreeting() параметр age типа Int. В новом объявлении функции оба параметра, name и age, должны быть разделены запятой:

```
fun birthdayGreeting(name: String, age: Int): String {
    val nameGreeting = «С днем рождения, $name!»
    val ageGreeting = «Тебе уже 5 лет!»
    return «$nameGreeting\n$ageGreeting»
}
```

- Новая строка поздравления должна использовать параметр age. Обновите функцию birthdayGreeting(), чтобы использовать значение параметра age в строке ageGreeting.

```
fun birthdayGreeting(name: String, age: Int): String {
    val nameGreeting = "Happy Birthday, $name!"
    val ageGreeting = "You are now $age years old!"
    return "$nameGreeting\n$ageGreeting"
}
```

- Запустите функцию и обратите внимание на ошибки в выводе:

```
Не передано значение для параметра 'age'
Не передано значение для параметра 'age'
```

- Измените два вызова функции birthdayGreeting() в main(), чтобы передать разный возраст для каждой собаки. Передайте 5 для возраста Ровера и 2 для возраста Рекса.

```
fun main() {
    println(birthdayGreeting(«Rover», 5))
    println(birthdayGreeting(«Rex», 2))
}
```

- Запустите ваш код. Теперь, когда вы передали значения для обоих параметров, при вызове функции в выводе должны отражаться имя и возраст каждой собаки.

```
Happy Birthday, Rover!
You are now 5 years old!
Happy Birthday, Rex!
You are now 2 years old!
```

Сигнатура функции

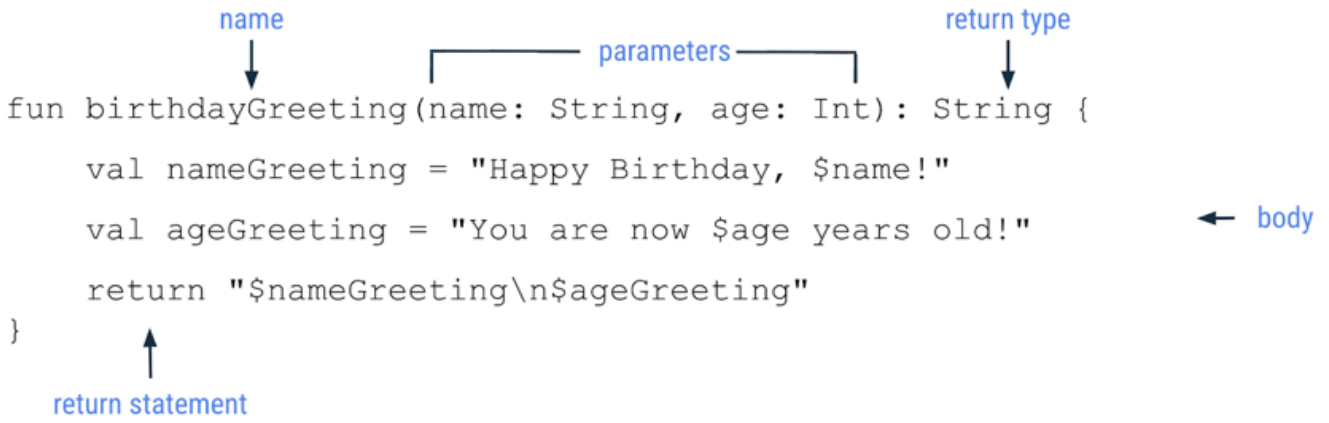
До сих пор вы видели, как определить имя функции, входы (параметры) и выходы. Имя функции и ее входы (параметры) вместе называются сигнатурой функции. Сигнатура функции состоит из всего, что находится перед типом return, и показана в следующем фрагменте кода.

```
fun birthdayGreeting(name: String, age: Int)
```

Параметры, разделенные запятыми, иногда называют списком параметров.

Эти термины часто встречаются в документации к коду, написанному другими разработчиками. Сигнатура функции говорит вам о том, как называется функция и какие типы данных могут быть переданы в нее.

Вы узнали много нового о синтаксисе, связанном с определением функций. Посмотрите на следующую диаграмму, чтобы получить представление о синтаксисе функций.



Именованные аргументы

В предыдущих примерах вам не нужно было указывать имена параметров, name или age, когда вы вызывали функцию. Однако вы можете сделать это по своему усмотрению. Например, вы можете вызвать функцию с большим количеством параметров или передать аргументы в другом порядке, например, поместить параметр age перед параметром name. Если при вызове функции вы указываете имя параметра, это называется именованным аргументом. Попробуйте использовать именованный аргумент в функции birthdayGreeting().

- Измените вызов функции Rex, чтобы использовать именованные аргументы, как показано в этом фрагменте кода. Для этого нужно указать имя параметра, за которым следует знак равенства, а затем значение (например, name = «Rex»).

```
println(birthdayGreeting(name = «Rex», age = 2))
```

- Запустите код и посмотрите, что вывод не изменился:

```
С днем рождения, Ровер!
Тебе исполнилось 5 лет!
```

```
С днем рождения, Рекс!
Тебе исполнилось 2 года!
```

- Перестройте именованные аргументы. Например, поместите именованный аргумент age перед именованным аргументом name.

```
println(birthdayGreeting(age = 2, name = «Rex»))
```

- Запустите код и обратите внимание, что вывод остался прежним. Несмотря на то, что вы изменили порядок аргументов, для тех же параметров передаются те же значения.

```
Happy Birthday, Rover!
You are now 5 years old!
Happy Birthday, Rex!
You are now 2 years old!
```

Аргументы по умолчанию

В параметрах функции также могут быть указаны аргументы по умолчанию. Возможно, Rover - ваша любимая собака, или вы ожидаете, что в большинстве случаев функция будет вызываться с определенными аргументами. Когда вы вызываете функцию, вы можете опустить аргументы, для которых есть значение по умолчанию, и в этом случае будет использоваться значение по умолчанию.

Чтобы добавить аргумент по умолчанию, добавьте оператор присваивания (=) после типа данных параметра и установите его равным значению. Измените свой код, чтобы использовать аргумент по умолчанию.

- В функции birthdayGreeting() установите для параметра name значение по умолчанию «Rover».

```
fun birthdayGreeting(name: String = «Rover», age: Int): String {
    return «С днем рождения, $name! Тебе исполнилось $age лет!»
}
```

- В первом вызове функции birthdayGreeting() для Rover в main() установите именованный аргумент age равным 5. Поскольку параметр age определен после имени, необходимо использовать именованный аргумент age. Без именованных аргументов Kotlin предполагает, что порядок аргументов совпадает с порядком определения параметров. Именованный аргумент используется для того, чтобы убедиться, что Kotlin ожидает Int для параметра age.

```
println(birthdayGreeting(age = 5))
println(birthdayGreeting(«Rex», 2))
```

- Запустите свой код. Первый вызов функции birthdayGreeting() печатает «Rover» в качестве имени, потому что вы не указали имя. Второй вызов функции birthdayGreeting() по-прежнему использует значение Rex, которое вы передали в качестве имени.

```
С днем рождения, Ровер! Тебе исполнилось 5 лет!
С днем рождения, Рекс! Тебе исполнилось 2 года!
```

- Удалите имя из второго вызова функции birthdayGreeting(). Опять же, поскольку имя опущено, необходимо использовать именованный аргумент для возраста.

```
println(birthdayGreeting(age = 5))
println(birthdayGreeting(age = 2))
```

- Запустите свой код и обратите внимание, что теперь оба вызова birthdayGreeting() выводят в качестве имени «Rover», поскольку не передан аргумент имени.

```
С днем рождения, Ровер! Тебе исполнилось 5 лет!
С днем рождения, Ровер! Тебе исполнилось 2 года!
```

Заключение

- Функции определяются с помощью ключевого слова **fun** и содержат многократно используемые фрагменты кода.
- Функции помогают упростить сопровождение больших программ и предотвратить ненужное повторение кода.
- Функции могут возвращать значение, которое вы можете сохранить в переменной для последующего использования.
- Функции могут принимать параметры - переменные, доступные внутри тела функции.
- Аргументы - это значения, которые вы передаете при вызове функции.
- При вызове функции аргументы можно именовать. При использовании именованных аргументов можно изменять порядок аргументов, не влияя на вывод.
- Можно указать аргумент по умолчанию, который позволяет опустить аргумент при вызове функции.