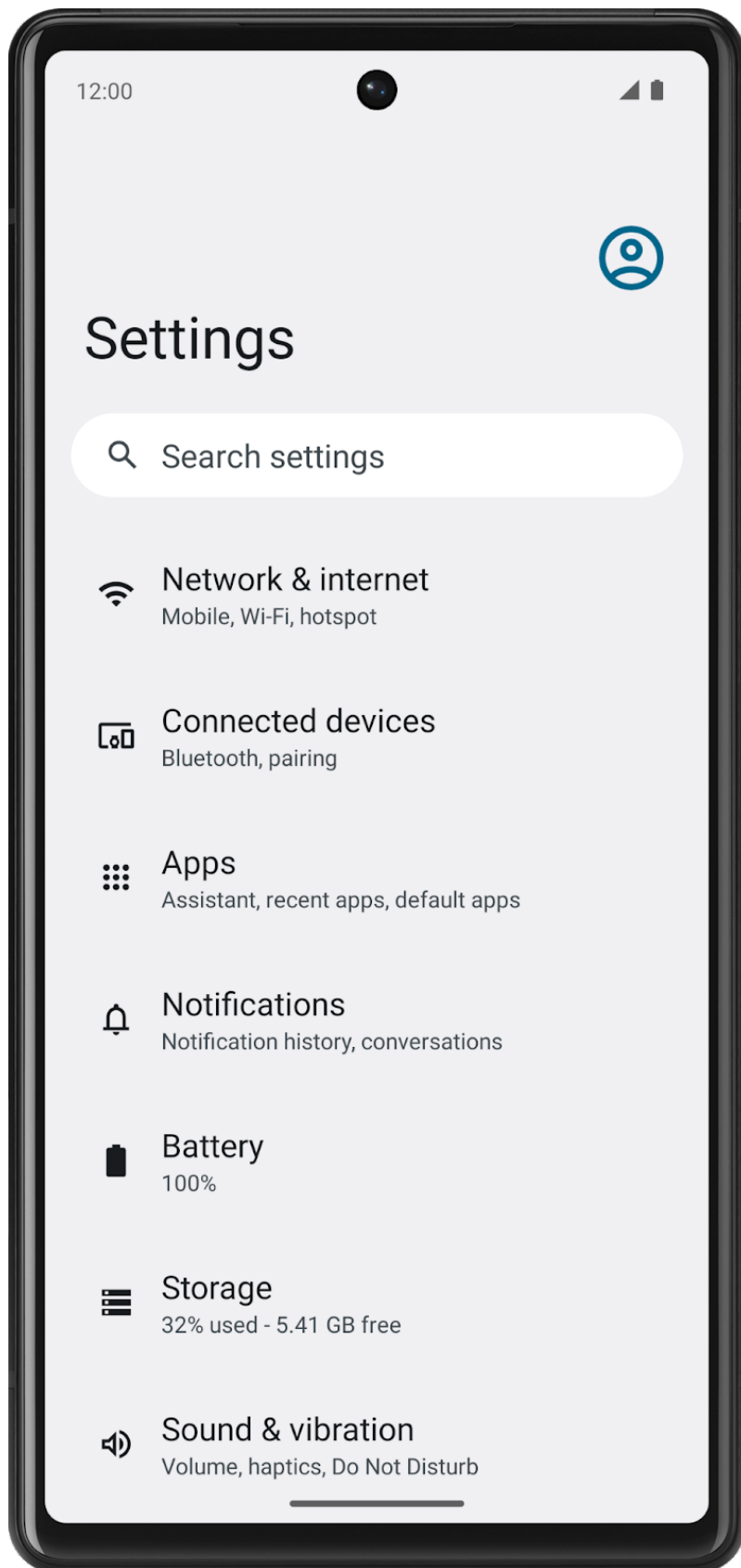


Тема: Создание и использование переменных в Kotlin

В приложениях, которые вы используете на своем телефоне, обратите внимание, что некоторые части приложения остаются неизменными, а другие меняются (или являются переменными).

Например, названия категорий в приложении «Настройки» остаются неизменными - «Сеть и интернет», «Подключенные устройства», «Приложения» и другие.



С другой стороны, если вы посмотрите на новостное приложение, статьи будут часто меняться. Меняются название статьи, источник, время публикации и изображения.

Как написать код так, чтобы контент менялся со временем? Вы не можете переписывать код в своем приложении каждый раз, когда появляются новые статьи, а это происходит каждый день, каждый час и каждую минуту!

На этом занятии вы узнаете, как написать код с использованием переменных, чтобы определенные части вашей программы могли меняться без необходимости писать новый набор инструкций.

Что вы будете создавать

- Короткие программы на Kotlin, использующие переменные.

Что вы узнаете

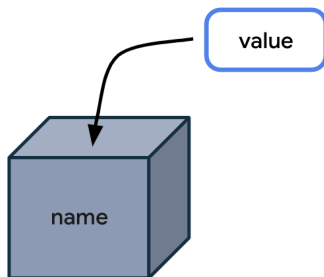
- Как определить переменную и обновить ее значение.
- Как выбрать подходящий тип данных для переменной из основных типов данных в Kotlin.
- Как добавлять комментарии к своему коду.

Что вам понадобится

- Компьютер с доступом в Интернет и веб-браузер.

Переменные и типы данных

В компьютерном программировании существует понятие переменной, которая представляет собой контейнер для одной части данных. Вы можете представить ее в виде ячейки, содержащей значение. У ячейки есть метка, которая является именем переменной. Обращаясь к ящику по его имени, вы получаете доступ к значению, которое в нем хранится.



Переменные и типы данных

В компьютерном программировании существует понятие переменной, которая представляет собой контейнер для одной части данных. Вы можете представить ее в виде ячейки, содержащей значение. У ячейки есть метка, которая является именем переменной. Обращаясь к ячейке по ее имени, вы получаете ее значение.

Зачем хранить значение в ячейке и обращаться к ней по ее имени, если можно просто использовать значение напрямую? Проблема в том, что если в вашем коде во всех инструкциях используются значения напрямую, то ваша программа будет работать только в этом конкретном случае.

Вот аналогия, которая поможет понять, почему переменные полезны. Ниже приведено письмо для человека, с которым вы недавно познакомились.

Дорогая Лорен,

Было приятно встретить тебя сегодня в офисе. С нетерпением жду встречи с тобой в пятницу.

Хорошего дня!

Это письмо замечательное, но оно подходит только для вашей конкретной ситуации с Лорен. А что, если вам приходится писать одно и то же письмо много раз, но с небольшими вариациями для разных людей? Эффективнее было бы создать единый шаблон письма, оставив пустые места для тех частей, которые могут меняться.

Дорогой _____ ,

Было приятно встретиться с вами сегодня на _____. С нетерпением жду встречи с вами на _____ .

Хорошего дня!

Вы также можете указать тип информации, которая будет размещена в каждом пустом месте. Это гарантирует, что шаблон письма будет использован так, как вы ожидали.

Уважаемый { имя } ,

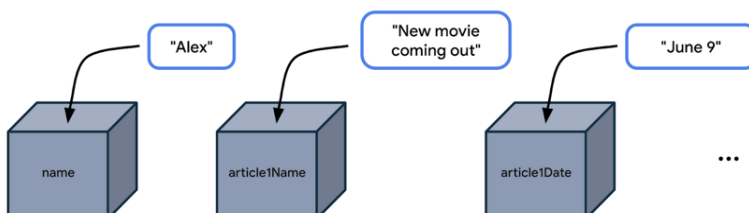
Было приятно встретиться с вами сегодня в { место } . С нетерпением жду встречи с вами в { дата } .

Хорошего дня!

Концептуально создание приложения похоже. У вас есть держатели для некоторых данных, в то время как другие части приложения остаются неизменными.



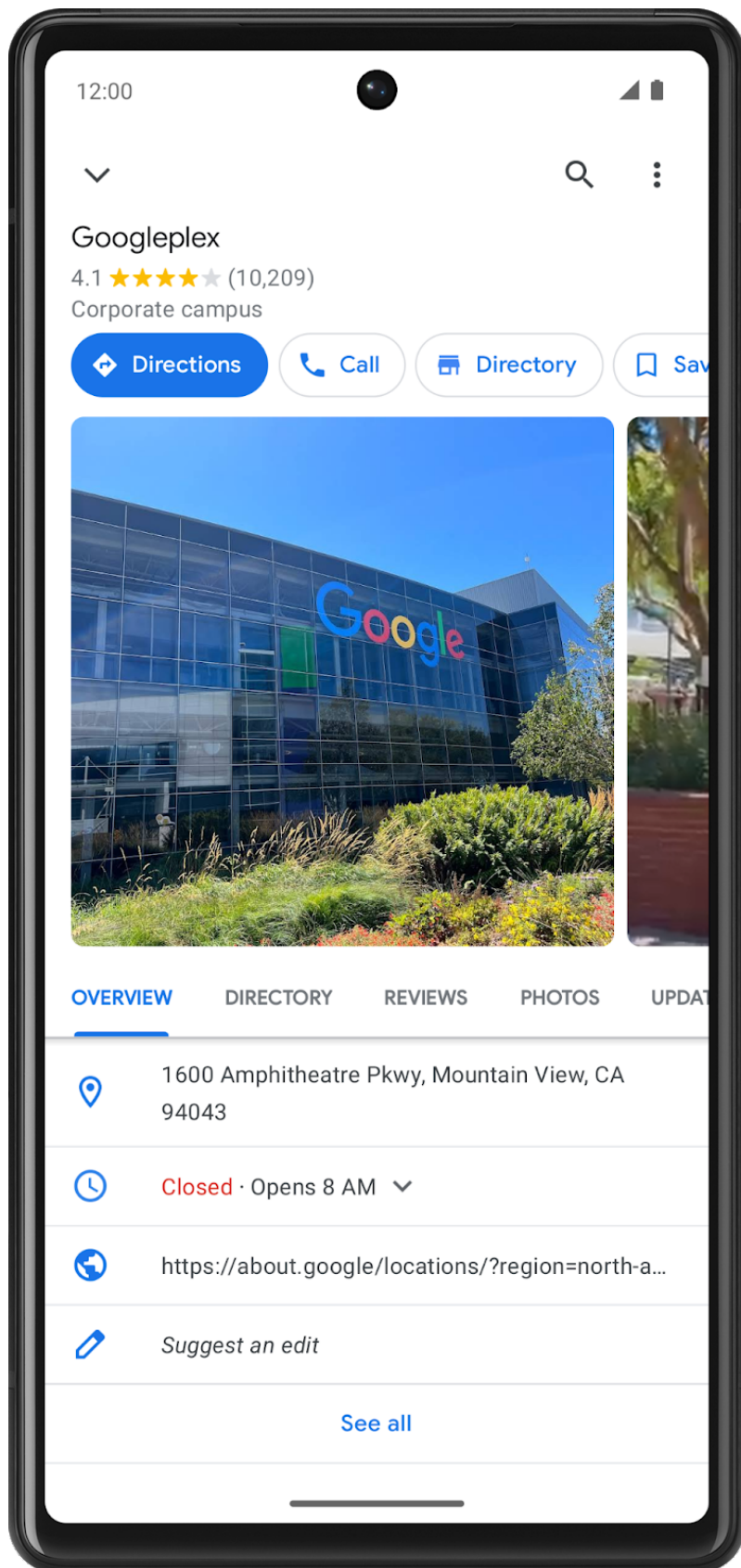
На приведенной выше иллюстрации новостного приложения текст «Добро пожаловать», заголовок «Последние новости для вас» и текст кнопки «Просмотреть больше статей» всегда остаются неизменными. Напротив, имя пользователя и содержание каждой статьи будут меняться, так что это отличная возможность использовать переменные для хранения каждой части информации.



Вы же не хотите, чтобы код (или инструкции) вашего новостного приложения работал только для пользователя по имени Алекс или для новостной статьи, которая всегда имеет один и тот же заголовок и дату публикации. Вместо этого вы хотите создать более гибкое приложение, поэтому вам следует писать код, ссылаясь на такие имена переменных, как `name`, `article1Name`, `article1Date` и так далее. Тогда ваш код станет достаточно общим, чтобы работать во многих различных случаях, когда имя пользователя может быть другим, а детали статьи - другими.

Пример приложения с переменными

Давайте рассмотрим пример приложения, чтобы увидеть, где в нем могут использоваться переменные.



В приложении карт вы можете найти экран с подробной информацией о каждом месте, например о ресторане или предприятии. На приведенном выше скриншоте из приложения **Google Maps** показана информация о штаб-квартире компании Google, которая называется Googleplex. Как вы думаете, какие части данных хранятся в приложении в виде переменных?

- Название места
- Звездный рейтинг места
- Количество отзывов о месте
- Сохранил ли пользователь это место (или сделал закладку)
- Адрес места

Измените данные, хранящиеся в этих переменных, и вы получите приложение для карт, достаточно гибкое, чтобы отображать информацию и о других местах.

Типы данных

Когда вы решаете, какие аспекты вашего приложения могут быть переменными, важно указать, какой тип данных может храниться в этих переменных. В Kotlin есть несколько общих базовых типов данных. В таблице ниже в каждой строке указан свой тип данных. Для каждого типа данных есть описание того, какие данные он может хранить, и примеры значений.

| Тип данных Kotlin | Какие данные он может содержать | Примеры буквенных значений |
|-------------------|---|------------------------------------|
| String | Text | "Add contact", "Search", "Sign in" |
| Int | Integer number | 32, 1293490, -59281 |
| Double | Decimal number | 2.0, 501.0292, -31723.99999 |
| Float | Decimal number (that is less precise than a Double). Has an f or F at the end of the number. | 5.0f, -1630.209f, 1.2940278F |
| Boolean | true or false. Use this data type when there are only two possible values. Note that true and false are keywords in Kotlin. | true, false |

Теперь, когда вы знаете о некоторых распространенных типах данных Kotlin, какой тип данных подойдет для каждой из переменных, указанных на странице детализации местоположения, которую вы видели ранее?



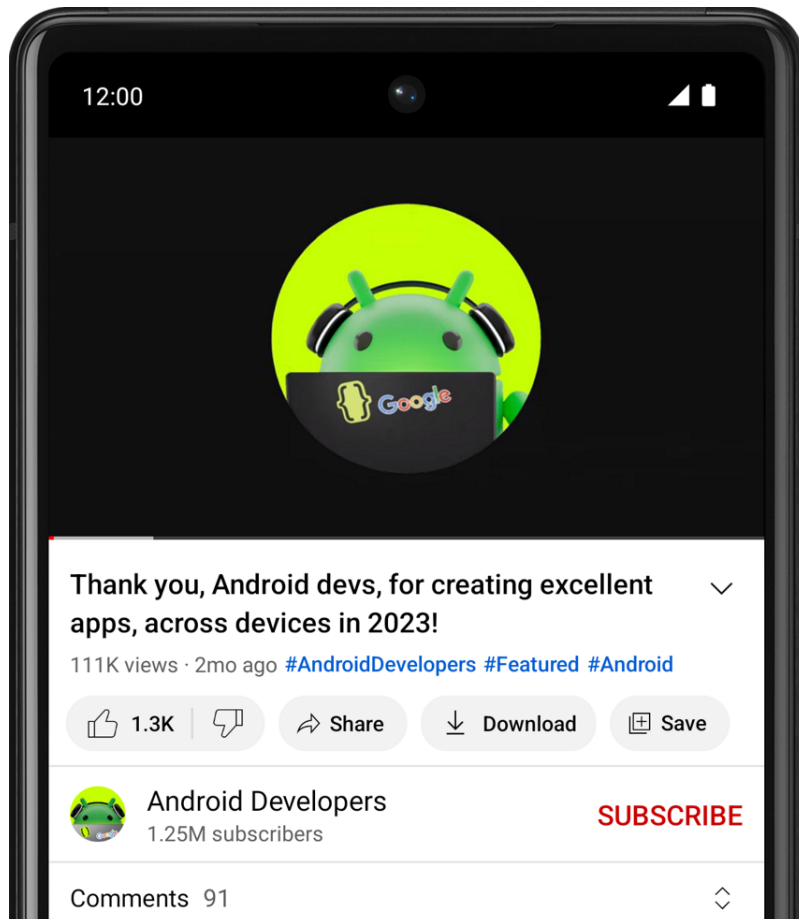
- Название места - это текст, поэтому его можно хранить в переменной, тип данных которой - String.
- Звездный рейтинг места - десятичное число (например, 4,2 звезды), поэтому его можно хранить как Double.
- Количество отзывов о локации - целое число, поэтому его следует хранить как Int.
- Сохранил ли пользователь это место, имеет только два возможных значения (сохранено или не сохранено), поэтому хранится как Boolean, где true и false могут представлять каждое из этих

состояний.

- Адрес местоположения - это текст, поэтому он должен быть строкой.

Потренируйтесь на двух других сценариях ниже. Определите, как используются переменные и их типы данных в следующих приложениях.

В приложении для просмотра видео, например в приложении YouTube, есть экран подробностей видео. Где могут использоваться переменные? Каков тип данных этих переменных?



Единого правильного ответа нет, но в приложении для просмотра видео переменные могут использоваться для следующих данных:

- Название видео (String)
- Название канала (String)
- Количество просмотров видео (Int)
- Количество лайков на видео (Int)
- Количество комментариев к видео (Int)

В таком приложении, как Messages, на экране отображается список последних полученных текстовых сообщений. Где могут использоваться переменные? Каков тип данных этих переменных?



И здесь нет единственно верного ответа. В приложении для обмена текстовыми сообщениями переменные могут использоваться для следующих данных:

Номер телефона отправителя (Строка) Временная метка сообщения (Строка) Предварительный просмотр содержимого сообщения (Строка) Является ли сообщение прочитанным (булево).

Определение и использование переменных

Прежде чем использовать переменную, ее нужно сначала определить в коде.

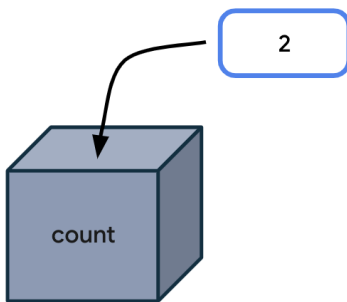
Когда вы определяете переменную, вы присваиваете ей имя, чтобы однозначно идентифицировать ее. Вы также решаете, какой тип данных она может хранить, указывая тип данных. Наконец, вы можете указать начальное значение, которое будет храниться в переменной, но это необязательно.

Примечание: Вы можете услышать альтернативную фразу «объявить переменную». Слова `declare` и `define` могут использоваться как взаимозаменяемые и имеют одинаковое значение. Вы также можете услышать термин «определение переменной» или «объявление переменной», которые относятся к коду, определяющему переменную. В других языках `declare` и `define` имеют разные значения.

Определив переменную, вы можете использовать ее в своей программе. Чтобы использовать переменную, напечатайте ее имя в коде, что скажет компилятору Kotlin, что вы хотите использовать значение переменной в данной точке кода.

Например, определите переменную для количества прочитанных сообщений в папке входящих сообщений пользователя. Переменная может иметь имя `count`. Сохраните в переменной значение, например, число 2, обозначающее 2 прочитанных сообщения в папке «Входящие» пользователя. (Вы

можете выбрать другое число для хранения в переменной, но для целей этого примера используйте число 2).



Каждый раз, когда вашему коду нужно получить доступ к количеству непрочитанных сообщений, введите в код `count`. При выполнении ваших инструкций компилятор Kotlin видит имя переменной в вашем коде и использует вместо него значение переменной.

Технически для описания этого процесса существуют более конкретные словарные слова:

Выражение - это небольшая единица кода, которая оценивается в значение. Выражение может состоять из переменных, вызовов функций и многого другого. В следующем примере выражение состоит из одной переменной: переменной `count`. Это выражение оценивается в 2.

expression

value

`count`

2

Evaluate означает определение значения выражения. В данном случае выражение оценивается в 2. Компилятор оценивает выражения в коде и использует эти значения при выполнении инструкций в программе.



Чтобы понаблюдать за этим поведением в Idea, запустите программу из следующего раздела.

Пример

- Откройте Idea.
- Замените существующий код в Kotlin Playground следующей программой.

Эта программа создает переменную `count` с начальным значением 2 и использует ее, распечатывая значение переменной `count` на выходе. Не волнуйтесь, если вы пока не понимаете всех аспектов синтаксиса кода. Более подробно он будет рассмотрен в последующих разделах.

```
fun main() {  
    val count: Int = 2  
    println(count)  
}
```

- Запустите программу, и на выходе должно получиться следующее:

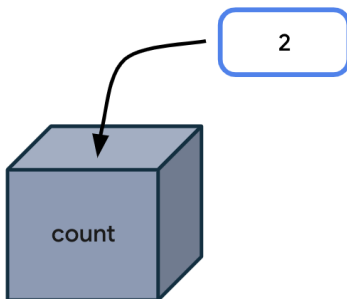
2

Объявление переменной

В программе, которую вы запустили, вторая строка кода гласит:

```
val count: Int = 2
```

Этот оператор создает целочисленную переменную count, которая содержит число 2.



Знакомство с синтаксисом (или форматом) объявления переменной в Kotlin может занять некоторое время. На следующей диаграмме показано, где должна располагаться каждая деталь переменной, а также расположение пробелов и символов.

`val` **name** `:` **data type** `=` **initial value**

В контексте примера с переменной count вы видите, что объявление переменной начинается со слова val. Имя переменной - count. Тип данных - Int, а начальное значение - 2.

name data type initial value

↓ ↓ ↙

```
val count: Int = 2
```

Ниже приводится более подробное описание каждой части объявления переменной.

Ключевое слово для определения новой переменной

Чтобы определить новую переменную, начните с ключевого слова **val** (что означает value) в Kotlin. Тогда компилятор Kotlin поймет, что в этом выражении находится объявление переменной.

Имя переменной

Точно так же, как вы называете функцию, вы называете и переменную. В объявлении переменной имя переменной следует за ключевым словом `val`.

```
val    name :    data type = initial value
```

Вы можете выбрать любое имя переменной, но в качестве лучшей практики избегайте использования ключевых слов Kotlin в качестве имени переменной.

Лучше всего выбрать имя, описывающее данные, которые хранит переменная, чтобы ваш код было легче читать.

Имена переменных должны следовать соглашению о верблюжьем регистре. Первое слово в имени переменной пишется в нижнем регистре. Если в имени несколько слов, пробелы между ними не ставятся, а все остальные слова должны начинаться с прописной буквы.

Примеры имен переменных:

- numberOfEmails
- cityName
- bookPublicationDate

В примере кода, показанном ранее, `count` - это имя переменной.

```
val count: Int = 2
```

Тип данных переменной

После имени переменной ставится двоеточие, пробел, а затем указывается тип данных переменной. Как упоминалось ранее, String, Int, Double, Float и Boolean - это основные типы данных Kotlin. Другие типы данных вы узнаете позже в этом курсе. Не забывайте писать типы данных именно так, как показано на рисунке, и начинать каждый из них с заглавной буквы.

val **name** : **data type** = **initial value**

В примере с переменной count типом данных переменной является Int.

```
val count: Int = 2
```

Оператор присваивания

В объявлении переменной после типа данных следует символ знака равенства (=). Символ знака равенства называется **оператором присваивания**. Оператор присваивания присваивает переменной значение. Другими словами, значение, стоящее в правой части знака равенства, сохраняется в переменной, стоящей в левой части знака равенства.

val count: Int = 2

Начальное значение переменной

Значение переменной - это фактические данные, которые хранятся в переменной.

val **name** : **data type** = **initial value**

В примере с переменной count число 2 является начальным значением переменной.

```
val count: Int = 2
```

Вы также можете услышать фразу: «Переменная count инициализирована на 2». Это означает, что 2 - это первое значение, которое сохраняется в переменной при ее объявлении.

Начальное значение будет отличаться в зависимости от типа данных, объявленного для переменной.

Обратитесь к следующей таблице 1, которую вы можете узнать из предыдущего раздела. В третьем столбце приведены примеры значений, которые можно хранить в переменной каждого соответствующего типа. Эти значения называются литералами, поскольку они являются фиксированными или постоянными (значение постоянно одно и то же). Например, целое число 32 всегда будет иметь значение 32. В отличие от этого, переменная не является литералом, поскольку ее значение может меняться. Вы можете услышать, как литеральные значения называются в зависимости от их типа: строковый литерал, целочисленный литерал, булевский литерал и т. д.

Важно указать подходящее и правильное значение в соответствии с типом данных переменной. Например, вы не можете хранить строковый литерал типа «Hello» в переменной типа `Int`, потому что компилятор Kotlin выдаст ошибку.

Использование переменной

Ниже приведена исходная программа, которую вы запускали в Idea. Вы узнали, что вторая строка кода создает новую целочисленную переменную `count` со значением 2.

```
fun main() {  
    val count: Int = 2  
    println(count)  
}
```

Теперь посмотрите на третью строку кода. Вы выводите на экран переменную `count`:

```
println(count)
```

Обратите внимание, что вокруг слова `count` нет кавычек. Это имя переменной, а не строковый литерал. (При запуске программы компилятор Kotlin оценивает выражение в круглых скобках, которое является `count`, для инструкции `println()`. Поскольку выражение оценивается в 2, то вызывается метод `println()` с 2 в качестве входных данных: `println(2)`).

Таким образом, результатом работы программы будет:

```
2
```

Само по себе число в выводе не очень полезно. Было бы полезнее, если бы в выводе было выведено более подробное сообщение, объясняющее, что представляет собой цифра 2.

Шаблон строки

Более полезным сообщением для вывода будет следующее:

```
You have 2 unread messages.
```

Выполните следующие шаги, чтобы программа выводила более полезное сообщение.

Обновите свою программу в Idea, добавив следующий код. Для вызова `println()` передайте строковый литерал, содержащий имя переменной `count`. Не забудьте окружить текст кавычками. Обратите внимание, что это не даст вам ожидаемых результатов. Вы решите эту проблему на следующем этапе.

```
fun main() {  
    val count: Int = 2  
    println("You have count unread messages.")  
}
```

- Запустите программу, и на экране должен появиться результат:

```
You have count unread messages.
```

Это предложение не имеет смысла! Вы хотите, чтобы в сообщении отображалось значение переменной `count`, а не ее имя.

Чтобы исправить вывод, вам понадобится строковый шаблон. Это строковый шаблон, потому что он содержит выражение шаблона, которое представляет собой знак доллара (\$), за которым следует имя переменной. Шаблонное выражение оценивается, и его значение подставляется в строку.

"string contents \$ **variable name** rest of string"

Добавьте знак доллара \$ перед переменной `count`. В этом случае шаблонное выражение `$count` оценивается в 2, и 2 подставляется в строку, в которой находилось выражение.

```
fun main() {  
    val count: Int = 2  
    println("You have $count unread messages.")  
}
```

Когда вы запускаете программу, результат соответствует желаемой цели:

```
You have 2 unread messages.
```

Это предложение имеет гораздо больше смысла для пользователя!

Теперь измените начальное значение переменной `count` на другой целочисленный литерал. Например, вы можете выбрать число 10. Остальной код программы оставьте без изменений.

```
fun main() {  
    val count: Int = 10  
    println("You have $count unread messages.")  
}
```

Запустите программу. Обратите внимание, что вывод изменился соответствующим образом, и вам даже не пришлось менять оператор `println()` в своей программе.

```
You have 10 unread messages
```

Вы видите, насколько полезным может быть строковый шаблон. Вы написали шаблон строки только один раз в своем коде («У вас \$count непрочитанных сообщений.»). Если вы измените начальное значение переменной `count`, оператор `println()` по-прежнему будет работать. Теперь ваш код стал более гибким!

Чтобы еще раз подчеркнуть этот момент, сравните две следующие программы. В первой программе используется строковый литерал с точным количеством непрочитанных сообщений непосредственно в строке. Эта программа работает только в том случае, если у пользователя 10 непрочитанных сообщений.

```
fun main() {  
    println("You have 10 unread messages.")  
}
```

Благодаря использованию переменной и шаблона строки во второй программе, ваш код может адаптироваться к большому количеству сценариев. Эта вторая программа более гибкая!

```
fun main() {  
    val count: Int = 10  
    println("You have $count unread messages.")  
}
```

Вывод типа Вот совет, который позволит вам писать меньше кода при объявлении переменных.

Вывод типа - это когда компилятор Kotlin может сделать вывод (или определить), какой тип данных должен быть у переменной, без явного указания типа в коде. Это означает, что вы можете не указывать тип данных в объявлении переменной, если задаете для нее начальное значение. Компилятор Kotlin посмотрит на тип данных начального значения и предположит, что вы хотите, чтобы переменная содержала данные этого типа.

Ниже приведен синтаксис объявления переменной, в котором используется вывод типа:

`val` **name** = **initial value**

Возвращаясь к примеру с подсчетом, скажу, что изначально программа содержала такую строку кода:

```
val count: Int = 2
```

Однако эту строку кода можно записать и так. Обратите внимание, что символ двоеточия (😊) и тип данных `Int` опущены. В обновленном синтаксисе меньше слов, и достигается тот же результат - создание переменной `Int` под названием `count` со значением `2`.

```
val count = 2
```

Компилятор Kotlin знает, что вы хотите сохранить `2` (целое число) в переменной `count`, поэтому он может сделать вывод, что переменная `count` имеет тип `Int`. Удобно, правда? Это один из примеров того, как писать код на Kotlin становится более лаконичным!

Примечание: Если вы не указываете начальное значение при объявлении переменной, вы должны указать ее тип. В этой строке кода начальное значение не указано, поэтому необходимо указать тип данных: `val count: Int` В этой строке кода указывается назначенное значение, поэтому тип данных можно не указывать: `val count = 2`

Несмотря на то, что в этом примере рассматривается только переменная типа `Int`, концепция вывода типа применима ко всем типам данных в Kotlin.

Основные математические операции с целыми числами В чем разница между переменной `Int` со значением `2` и переменной `String` со значением «`2`»? Когда они обе выводятся на экран, они выглядят одинаково.

Преимущество хранения целых чисел в виде `Int` (в отличие от `String`) заключается в том, что вы можете выполнять математические операции с переменными `Int`, такие как сложение, вычитание, деление и умножение (см. другие операторы). Например, две целочисленные переменные можно сложить, чтобы получить их сумму. Конечно, есть случаи, когда целесообразно хранить целые числа в виде строк, но цель этого раздела - показать, что можно делать с переменными `Int`.

- Вернитесь в Kotlin Playground и удалите весь код в редакторе кода.
- Создайте новую программу, в которой определите целочисленную переменную для количества неп прочитанных писем в папке «Входящие» и инициализируйте ее значением, например `5`. При желании вы можете выбрать другое число. Определите вторую целочисленную переменную для количества прочитанных писем в папке «Входящие». Инициализируйте ее значением, например

100. При желании вы можете выбрать другое число. Затем выведите общее количество сообщений в папке «Входящие» путем сложения двух целых чисел.

```
fun main() {  
    val unreadCount = 5  
    val readCount = 100  
    println("You have ${unreadCount + readCount} total messages in your inbox.")  
}
```

- Запустите программу, и она должна отобразить общее количество сообщений в папке «Входящие»:

```
You have 105 total messages in your inbox.
```

Для строкового шаблона вы узнали, что перед именем одной переменной можно поставить символ \$. Однако если у вас есть более сложное выражение, вы должны заключить его в фигурные скобки с символом \$ перед фигурными скобками: \${unreadCount + readCount}. Выражение в фигурных скобках, unreadCount + readCount, оценивается в 105. Затем значение 105 подставляется в строковый литерал.

expression

value

unreadCount + readCount

105

Предупреждение: Если вы забудете фигурные скобки вокруг шаблонного выражения, то получите неожиданные результаты. Вы можете проверить это в Kotlin Playground, изменив оператор println() на println(«У вас в папке входящих сообщений \$unreadCount + readCount.») и наблюдая за выводом.

Для дальнейшего изучения этой темы создайте переменные с разными именами и разными начальными значениями и используйте шаблонные выражения для печати сообщений на выходе. Например, измените свою программу так, чтобы она выводила следующее:

```
100 photos  
10 photos deleted  
90 photos left
```

Вот один из способов написания программы, хотя есть и другие правильные способы!

```
fun main() {  
    val numberOfPhotos = 100  
    val photosDeleted = 10
```

```
println("$numberOfPhotos photos")
println("$photosDeleted photos deleted")
println("${numberOfPhotos - photosDeleted} photos left")
}
```

Обновление переменных

Во время работы приложения может потребоваться обновить значение переменной. Например, в приложении для покупок, когда пользователь добавляет товары в корзину, общая сумма корзины увеличивается.

Давайте упростим пример с покупками до простой программы. Логика написана ниже человеческим языком, а не на Kotlin. Это называется псевдокодом, потому что он описывает ключевые моменты того, как будет написан код, но не содержит всех деталей кода.

Примечание: Псевдокод не является рабочим кодом, который можно скомпилировать, поэтому он и называется псевдокодом.

В функции `main` программы:

Создайте целочисленную переменную `cartTotal`, которая начинается со значения 0. Пользователь добавляет в корзину свитер стоимостью 20 долларов. Обновите переменную `cartTotal` до значения 20, что является текущей стоимостью товаров в корзине. Выведите на экран общую стоимость товаров в корзине, которая является переменной `cartTotal`. Чтобы еще больше упростить код, вам не нужно писать код, когда пользователь добавляет товары в корзину. (Вы еще не узнали о том, как программа может реагировать на ввод пользователя. Об этом мы расскажем в следующем разделе.) Поэтому сосредоточьтесь на тех частях, где вы создаете, обновляете и выводите на печать переменную `cartTotal`.

Замените существующий код в Kotlin Playground на приведенную ниже программу. В строке 2 программы вы инициализируете переменную `cartTotal` значением 0. Поскольку вы указываете начальное значение, нет необходимости указывать тип данных `Int` из-за вывода типа. В строке 3 программы вы пытаетесь обновить переменную `cartTotal` до значения 20 с помощью оператора присваивания (`=`). В строке 4 программы вы выводите переменную `cartTotal` с помощью шаблона строки.

```
fun main() {
    val cartTotal = 0
    cartTotal = 20
    println("Total: $cartTotal")
}
```

Запустите программу, и вы получите ошибку компиляции. Обратите внимание, что в ошибке говорится о невозможности переназначения переменной `val`. Ошибка находится в третьей строке программы, которая пытается изменить значение переменной `cartTotal` на 20. Переменная `val cartTotal` не может быть переназначена на другое значение (20) после того, как ей было присвоено начальное значение (0).

```
Val cannot be reassigned
```

Если вам нужно обновить значение переменной, объявите ее с помощью ключевого слова `var` в Kotlin, а не `val`.

Ключевое слово `val` - используется, когда вы ожидаете, что значение переменной не изменится.

Ключевое слово `var` - используется, когда предполагается, что значение переменной может измениться. При использовании `val` переменная доступна только для чтения, что означает, что вы можете читать или получать доступ только к значению переменной. Как только значение установлено, вы не можете редактировать или изменять его. При использовании `var` переменная является мутабельной, то есть ее значение может быть изменено или модифицировано. Значение может быть изменено.

Чтобы запомнить разницу, думайте о `val` как о фиксированном значении, а о `var` как о переменной. В Kotlin рекомендуется использовать ключевое слово `val` вместо ключевого слова `var`, когда это возможно.


Обновите объявление переменной `cartTotal` в строке 2 программы, чтобы использовать `var` вместо `val`. Вот как должен выглядеть код:

```
fun main() {  
    var cartTotal = 0  
    cartTotal = 20  
    println("Total: $cartTotal")  
}
```

Обратите внимание на синтаксис кода в строке 3 программы, который обновляет переменную.

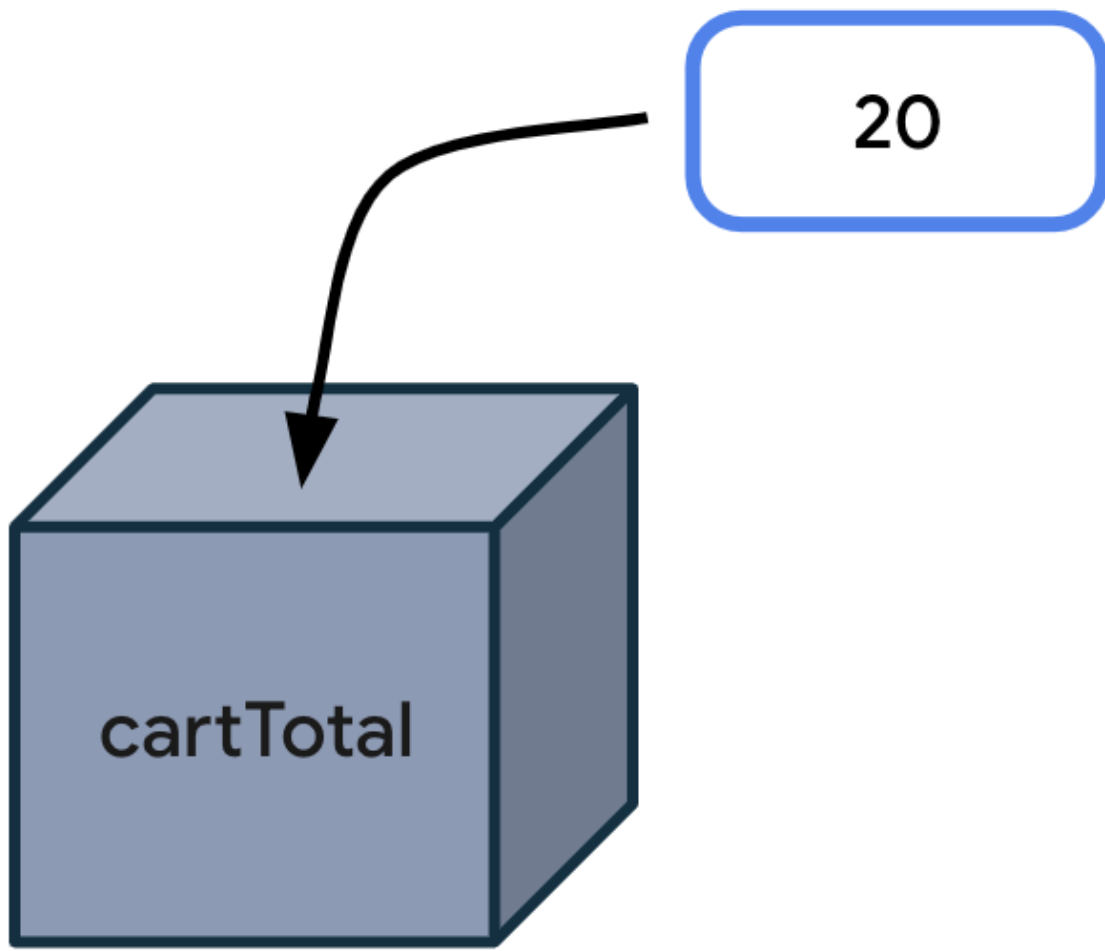
```
cartTotal = 20
```

Используйте оператор присваивания (`=`), чтобы присвоить новое значение (20) существующей переменной (`cartTotal`). Вам не нужно снова использовать ключевое слово `var`, потому что переменная уже определена.



`cartTotal = 20`

Используя аналогию с коробкой, представьте, что значение 20 хранится в ячейке с надписью `cartTotal`.



Вот схема общего синтаксиса обновления переменной, которая уже была объявлена в предыдущей строке кода. Начните оператор с имени переменной, которую вы хотите обновить. Добавьте пробел, знак равенства, а затем еще один пробел. Затем напишите обновленное значение переменной.



Запустите вашу программу, и код должен успешно скомпилироваться. Он должен вывести следующее сообщение:

```
Total: 20
```

Чтобы увидеть, как изменяется значение переменной во время работы программы, выведите переменную `cartTotal` на экран после ее первоначального объявления. Смотрите изменения в коде ниже. В строке 3 появился новый оператор `println()`. В строке 4 кода также добавлена пустая строка.

Пустые строки никак не влияют на то, как компилятор понимает код. Добавьте пустую строку там, где это облегчит чтение кода, отделив связанные блоки кода.

```
fun main() {  
    var cartTotal = 0  
    println("Total: $cartTotal")  
  
    cartTotal = 20  
    println("Total: $cartTotal")  
}
```

Запустите программу снова, и результат должен быть следующим:

```
Total: 0  
Total: 20
```

Вы видите, что первоначально сумма в корзине равна 0. Затем она обновляется до 20. Вы успешно обновили переменную! Это стало возможным благодаря тому, что вы превратили `cartTotal` из переменной, доступной только для чтения (с помощью `val`), в изменяемую переменную (с помощью `var`).

Помните, что использовать `var` для объявления переменной следует только в том случае, если вы ожидаете, что ее значение будет меняться. В противном случае следует по умолчанию использовать `val` для объявления переменной. Такая практика делает ваш код более безопасным. Использование `val` гарантирует, что переменные не будут обновляться в вашей программе, если вы этого не ожидаете. Если переменной `val` присвоено значение, оно всегда остается таким.

Примечание: Если вы знакомы с другими языками программирования, объявление `val` похоже на объявление постоянного значения, поскольку это переменная, доступная только для чтения. При объявлении констант в Kotlin существуют дополнительные соглашения, которые слишком сложны для этого коделаба, но вы можете найти их в разделе `Constants` руководства по стилю.

Операторы инкремента и декремента Теперь вы знаете, что переменная должна быть объявлена как `var`, чтобы обновить ее значение. Примените эти знания к приведенному ниже примеру с почтовым сообщением, который должен показаться вам знакомым.

Замените код в Kotlin Playground на эту программу:

```
fun main() {  
    val count: Int = 10  
    println("У вас $count непрочитанных сообщений.")  
}
```

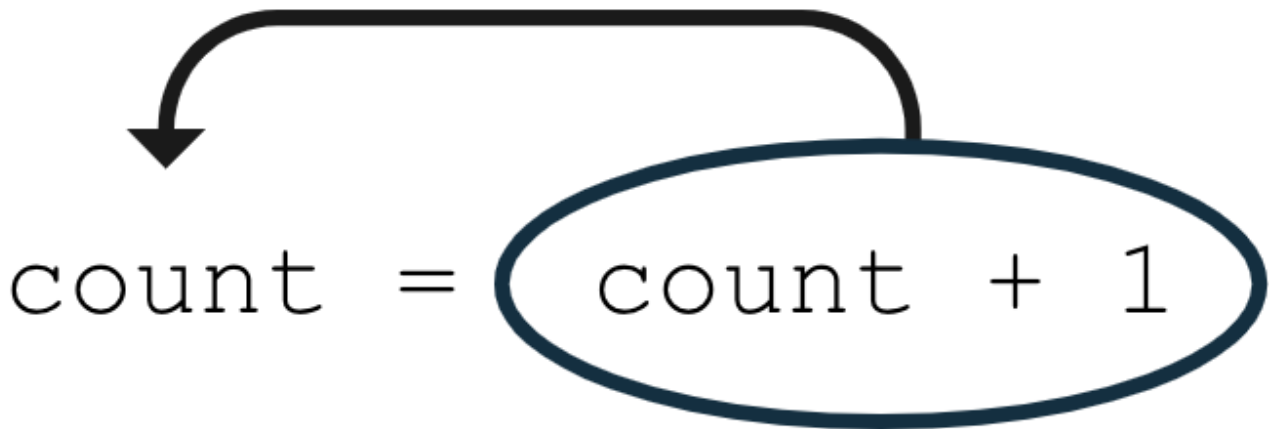
Запустите программу. Она должна вывести:

У вас есть 10 непрочитанных сообщений.

Однако теперь вы можете обновить счетчик до другого значения. Например, когда в папку входящих сообщений пользователя приходит одно новое письмо, вы можете увеличить счетчик на 1. (Вам не нужно писать код для получения письма. Получение данных из интернета - это более сложная тема для более позднего раздела). Пока же сосредоточьтесь на том, чтобы переменная `count` увеличивалась на 1 с помощью этой строки кода:

```
count = count + 1
```

Выражение справа от знака равенства равно `count + 1` и оценивается в 11. Это потому, что текущее значение `count` равно 10 (которое находится в строке 2 программы), а `10 + 1` равно 11. Затем с помощью оператора присваивания значение 11 присваивается или сохраняется в переменной `count`.



Добавьте эту строку кода в свою программу в нижней части функции `main()`. Ваш код должен выглядеть следующим образом:

```
fun main() {  
    var count = 10  
    println(«У вас $count непрочитанных сообщений.»)  
    count = count + 1  
}
```

Если вы запустите программу сейчас, вывод будет таким же, как и раньше, потому что вы не добавили никакого кода для использования переменной `count` после ее обновления.

Добавьте еще один оператор `print`, который выведет количество непрочитанных сообщений после обновления переменной.


```
fun main() {  
    var count = 10  
    println("You have $count unread messages.")  
    count = count + 1  
    println("You have $count unread messages.")  
}
```

Запустите программу. Во втором сообщении должно появиться обновленное число 11 сообщений.

```
У вас 10 непрочитанных сообщений.  
У вас 11 непрочитанных сообщений.
```

Для сокращения, если вы хотите увеличить переменную на 1, вы можете использовать оператор инкремента (++), состоящий из двух символов плюс. Используя эти символы непосредственно после имени переменной, вы сообщаете компилятору, что хотите прибавить 1 к текущему значению переменной, а затем сохранить новое значение в переменной. Следующие две строки кода эквивалентны, но использование оператора инкремента ++ требует меньшего количества символов.

```
count = count + 1
```

```
count++
```

Внесите это изменение в ваш код и запустите программу. Между именем переменной и оператором инкремента не должно быть пробелов.

```
fun main() {  
    var count = 10  
    println(«У вас $count непрочитанных сообщений.»)  
    count++  
    println(«У вас $count непрочитанных сообщений.»)  
}
```

Запустите программу. Результат тот же, но теперь вы узнали о новом операторе!

```
У вас есть 10 непрочитанных сообщений.  
У вас 11 непрочитанных сообщений.
```

Теперь измените строку 4 вашей программы, чтобы использовать оператор декремента (--) после имени переменной count. Оператор декремента состоит из двух символов минус. Помещая оператор

декремента после имени переменной, вы сообщаете компилятору, что хотите уменьшить значение переменной на 1 и сохранить новое значение в переменной.

```
fun main() {  
    var count = 10  
    println(«У вас $count непрочитанных сообщений.»)  
    count--  
    println(«У вас $count непрочитанных сообщений.»)  
}
```

Запустите программу. Она должна вывести следующее сообщение:

```
У вас 10 непрочитанных сообщений.  
У вас 9 непрочитанных сообщений.
```

В этом разделе вы узнали, как обновлять изменяемую переменную с помощью оператора инкремента (++) и оператора декремента (--). Более конкретно, count++ - это то же самое, что count = count + 1, а count-- - это то же самое, что count = count - 1.

Изучите другие типы данных

Ранее в codelab вы познакомились с некоторыми распространенными базовыми типами данных: String, Int, Double и Boolean. Вы только что использовали тип данных Int, теперь вы изучите другие типы данных.

| Kotlin data type | What kind of data it can contain |
|------------------|--|
| String | Text |
| Int | Integer number |
| Double | Decimal number |
| Boolean | true or false (only two possible values) |

Попробуйте эти программы в Kotlin Playground и посмотрите, что получится на выходе.

Double

Когда вам нужна переменная с десятичным значением, используйте переменную Double. Чтобы узнать о ее допустимом диапазоне, обратитесь к этой таблице и посмотрите, какие десятичные цифры она может хранить, например.

Примечание: Название типа данных Double происходит от того, что этот тип данных имеет двойную точность по сравнению с типом данных Float, который имеет одинарную точность. Точность - это количество десятичных цифр, которые они могут хранить. Таким образом, переменная Double может хранить более точное значение. Если вам интересно, в этой таблице

приведены более подробные сведения о конкретных различиях между типами Double и Float. Этот раздел коделаба посвящен использованию Double для работы с десятичными числами.

Представьте, что вы направляетесь в пункт назначения, и ваше путешествие разделено на три отдельные части, потому что вам нужно делать остановки по пути. Эта программа отображает общее расстояние, оставшееся до пункта назначения.

- Введите этот код в Kotlin Playground. Можете ли вы понять, что происходит в каждой строке кода?

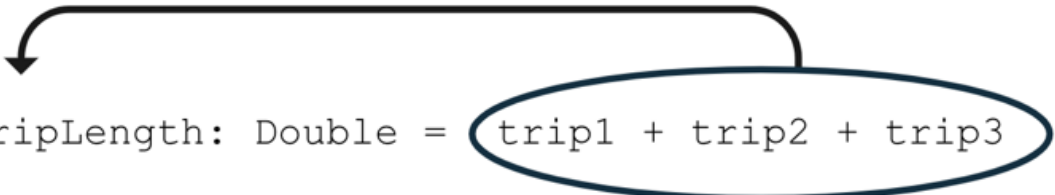
```
fun main() {  
    val trip1: Double = 3.20  
    val trip2: Double = 4.10  
    val trip3: Double = 1.72  
    val totalTripLength: Double = 0.0  
    println("$totalTripLength miles left to destination")  
}
```

Три переменные под названием trip1, trip2 и trip3 объявлены для представления расстояния каждой части поездки. Все они являются переменными Double, поскольку хранят десятичные значения. Используйте val для объявления каждой переменной, потому что их значения не меняются в процессе работы программы. Программа также создает четвертую переменную TotalTripLength, которая в данный момент инициализирована значением 0.0. В последней строке программы печатается сообщение со значением переменной totalTripLength.

- Исправьте код так, чтобы переменная totalTripLength была суммой всех трех длин поездок.

```
val totalTripLength: Double = trip1 + trip2 + trip3
```

Выражение справа от знака равенства оценивается в 9.02, потому что $3.20 + 4.10 + 1.72$ равно 9.02. Значение 9,02 сохраняется в переменной totalTripLength.



The diagram shows the code `val totalTripLength: Double = trip1 + trip2 + trip3`. A hand-drawn oval encircles the expression `trip1 + trip2 + trip3`. A curved arrow points from this oval to the equals sign in the assignment statement, indicating that the sum of the three trip distances is assigned to the variable totalTripLength.

Вся ваша программа должна выглядеть так, как показано ниже:

```
fun main() {  
    val trip1: Double = 3.20  
    val trip2: Double = 4.10  
    val trip3: Double = 1.72  
    val totalTripLength: Double = trip1 + trip2 + trip3
```

```
println("$totalTriplength miles left to destination")
}
```

- Запустите программу. Она должна вывести следующее:

```
9.02 miles left to destination
```

Обновите код, чтобы удалить ненужный тип данных `Double` из объявлений переменных из-за вывода типа. Компилятор Kotlin может сделать вывод о том, что эти переменные относятся к типу данных `Double`, на основании десятичных чисел, предоставленных в качестве начальных значений.

```
fun main() {
    val trip1 = 3.20
    val trip2 = 4.10
    val trip3 = 1.72
    val totalTriplength = trip1 + trip2 + trip3
    println("$totalTriplength miles left to destination")
}
```

Запустите код еще раз, чтобы убедиться, что он по-прежнему компилируется. Результат должен быть таким же, но теперь ваш код стал проще!

String

Когда вам нужна переменная, которая может хранить текст, используйте переменную `String`. Не забывайте использовать кавычки вокруг литеральных значений `String`, таких как «Hello Kotlin», в то время как литеральные значения `Int` и `Double` не заключаются в кавычки.

- Скопируйте и вставьте эту программу в Kotlin Playground.

```
fun main() {
    val nextMeeting = "Next meeting:"
    val date = "January 1"
    val reminder = nextMeeting + date
    println(reminder)
}
```

Обратите внимание, что объявлены две строковые переменные, переменная `nextMeeting` и переменная `date`. Затем объявляется третья строковая переменная `reminder`, которая устанавливается равной переменной `nextMeeting` плюс переменная `date`.

С помощью символа `+` можно сложить две строки вместе, что называется конкатенацией. Две строки объединяются одна за другой. Результатом выражения `nextMeeting + date` будет «Следующая встреча:1 января», как показано на диаграмме ниже.



Значение «Следующая встреча:1 января» сохраняется в переменной reminder с помощью оператора присваивания в строке 4 программы.

- Запустите свою программу. Она должна вывести следующее:

```
Следующая встреча:1 января
```

При конкатенации двух строк между ними не добавляются пробелы. Если вам нужен пробел после двоеточия в результирующей строке, вам нужно добавить пробел в одну или другую строку.

- Обновите переменную nextMeeting, чтобы в конце строки перед закрывающей кавычкой появился дополнительный пробел. (В качестве альтернативы можно было бы добавить дополнительный пробел в начало переменной date). Ваша программа должна выглядеть следующим образом:

```
fun main() {  
    val nextMeeting = "Next meeting: "  
    val date = "January 1"  
    val reminder = nextMeeting + date  
    println(reminder)  
}
```

- Запустите программу снова, и теперь в выходном сообщении после двоеточия должен быть пробел.

```
Следующая встреча: 1 января
```

- Измените код так, чтобы конкатенировать - или добавить - еще один фрагмент текста к выражению, которое сохраняется в переменной напоминания.

Используйте символ +, чтобы добавить строковый литерал « на работе» в конец строки напоминания.

- Запустите программу.

Она должна вывести следующее сообщение:

```
Следующая встреча: 1 января на работе
```

В приведенном ниже коде показан один из способов реализации этого поведения.

```
fun main() {  
    val nextMeeting = "Next meeting: "  
    val date = "January 1"  
    val reminder = nextMeeting + date + " at work"  
    println(reminder)  
}
```

Обратите внимание, что вокруг nextMeeting и date нет кавычек, потому что это имена существующих строковых переменных (где их соответствующие значения - это текст с кавычками вокруг них). И наоборот, литерал « at work» ранее не был определен ни в одной переменной, поэтому используйте кавычки вокруг этого текста, чтобы компилятор знал, что это строка, которая должна быть конкатенирована с другими строками.

Технически, вы можете получить тот же результат, объявив одну переменную String с полным текстом вместо использования отдельных переменных. Однако цель этого упражнения - продемонстрировать, как можно объявлять переменные String и работать с ними, особенно как конкатенировать отдельные строки.

При чтении кода, содержащего строки, вы можете столкнуться с экранирующими последовательностями. Эскейп-последовательности - это символы, которым предшествует символ обратной косой черты (), который также называется экранирующей обратной косой чертой. Примером может служить символ \» внутри строкового литерала, как в примере ниже. Скопируйте и вставьте этот код в Kotlin Playground.

```
fun main() {  
    println(«Скажи \»привет\"")  
}
```

Ранее вы научились использовать двойные кавычки вокруг строкового литерала. Но что, если вы хотите использовать символ « в строке? Тогда вам нужно добавить символ обратной косой черты перед двойными кавычками как \» в вашей строке. Помните, что вокруг всей строки должны оставаться двойные кавычки.

Запустите программу, чтобы увидеть результат. В результате должно получиться:

```
Say "hello"
```

В выводе отображаются кавычки вокруг hello, потому что мы добавили \» перед и после hello в операторе println().

О других управляющих последовательностях, поддерживаемых в Kotlin, читайте на странице документации по управляющим последовательностям. Например, если вам нужна новая строка в строке, используйте символ \ перед символом n, как в \n.

Теперь вы узнали о конкатенации строк, а также об экранирующих последовательностях внутри строк. Переходим к последнему типу данных, который рассматривается в этой кодовой лаборатории.

Boolean

Тип данных Boolean полезен, когда переменная имеет только два возможных значения - true или false.

Примером может служить переменная, которая показывает, включен или выключен авиарежим на устройстве, или включены или выключены уведомления в приложении.

Введите этот код в Kotlin Playground. В строке 2 этой программы вы объявляете булеву переменную `notificationsEnabled` и инициализируете ее значением `true`. Технически, вы можете опустить `: Boolean` в объявлении, так что вы можете удалить его, если хотите. В строке 3 программы вы выводите значение переменной `notificationsEnabled`.

```
fun main() {  
    val notificationsEnabled: Boolean = true  
    println(notificationsEnabled)  
}
```

Запустите программу, и она должна вывести следующее:

```
true
```

Измените начальное значение булевой функции на `false` в строке 2 программы.

```
fun main() {  
    val notificationsEnabled: Boolean = false  
    println(notificationsEnabled)  
}
```

Запустите программу, и она должна вывести следующее:

```
false
```

- К строкам можно конкатенировать и другие типы данных. Например, к строкам можно присоединять булевы числа. Используйте символ `+` для конкатенации (или добавления) значения булевой переменной `notificationsEnabled` в конец строки «Включены ли уведомления? «.

```
fun main() {  
    val notificationsEnabled: Boolean = false
```

```
println("Are notifications enabled? " + notificationsEnabled)
}
```

- Запустите программу, чтобы увидеть результат конкатенации. Программа должна вывести следующее сообщение:

```
Уведомления включены? false
```

Вы видите, что можно установить булеву переменную в значение true или false. Булевы переменные позволяют создавать более интересные сценарии, в которых вы выполняете некоторый набор инструкций, когда булева переменная имеет истинное значение. Или, если булева переменная имеет ложное значение, вы пропускаете эти инструкции. Подробнее о булевых переменных вы узнаете в одном из следующих кодовых уроков.

Соглашения по кодированию

Вы познакомились с руководством по стилю Kotlin для написания кода для Android в соответствии с рекомендациями Google и рекомендациями других профессиональных разработчиков.

Вот несколько других соглашений по форматированию и кодированию, которые вы должны соблюдать, исходя из новых тем, которые вы узнали:

Имена переменных должны быть в верблюжьем регистре и начинаться со строчной буквы. В объявлении переменной после двоеточия должен стоять пробел, когда вы указываете тип данных.


space



```
val discount: Double = .20
```


Перед и после оператора должен стоять пробел, как, например, перед операторами присваивания (=), сложения (+), вычитания (-), умножения (*), деления (/) и другими.

space



```
var pet = "bird"
```

space



```
val sum = 1 + 2
```

При написании более сложных программ рекомендуется ограничение в 100 символов на строку. Это гарантирует, что вы сможете легко прочитать весь код программы на экране компьютера, без необходимости горизонтальной прокрутки при чтении кода.

7. Комментирование кода

При написании кода еще одной хорошей практикой является добавление комментариев, которые описывают, для чего предназначен код. Комментарии могут помочь людям, читающим ваш код, легче его понять. Два символа прямой косой черты, или `//`, указывают на то, что текст после них в остальной части строки считается комментарием, поэтому он не интерпретируется как код. Обычно после двух символов прямой косой черты ставится пробел.

```
// Это комментарий.
```

Комментарий также может начинаться в середине строки кода. В этом примере `height = 1` является обычным кодовым утверждением. `// Предположим, что начало height = 1` интерпретируется как комментарий и не считается частью кода.

```
height = 1 // Предположим, что высота равна 1 для начала
```

Если вы хотите описать код более подробно с помощью длинного комментария, превышающего 100 символов в строке, используйте многострочный комментарий. Начните многострочный комментарий с прямой косой черты (/) и символа звездочки (), как /. Добавьте звездочку в начале каждой новой строки комментария. И, наконец, завершите комментарий звездочкой и символом прямой косой черты */.

```
/*
 * This is a very long comment that can
 * take up multiple lines.
 */
```

Эта программа содержит однострочные и многострочные комментарии, которые описывают происходящее:

```
/*
 * Эта программа отображает количество сообщений
 * в папке входящих сообщений пользователя.
 */
fun main() {
    // Создаем переменную для количества непрочитанных сообщений.
    var count = 10
    println(«У вас $count непрочитанных сообщений.»)

    // Уменьшаем количество сообщений на 1.
    count--
    println(«У вас $count непрочитанных сообщений.»)
}
```

Как уже говорилось ранее, вы можете добавить в код пустые строки, чтобы сгруппировать связанные утверждения и сделать код более удобным для чтения.

- Добавьте несколько комментариев к фрагменту кода, который вы использовали ранее.
- Запустите программу, чтобы убедиться, что поведение не изменилось, поскольку комментарии не должны влиять на вывод.

Заключение

Отличная работа! Вы узнали о переменных в Kotlin, о том, почему переменные полезны в программировании и как их создавать, обновлять и использовать. Вы экспериментировали с различными базовыми типами данных в Kotlin, включая типы данных Int, Double, String и Boolean. Вы также узнали о разнице между ключевыми словами val и var.

Все эти понятия являются важнейшими строительными блоками на вашем пути к становлению разработчика.

Резюме

- Переменная - это контейнер для одной части данных.
- Прежде чем использовать переменную, ее нужно сначала объявить.
- Используйте ключевое слово `val`, чтобы определить переменную, которая доступна только для чтения, и ее значение не может измениться после присвоения.
- Ключевое слово `var` используется для определения переменной, которая является мутабельной или изменяемой.
- В Kotlin предпочтительнее использовать `val`, а не `var`, когда это возможно.
- Чтобы объявить переменную, начните с ключевого слова `val` или `var`. Затем укажите имя переменной, тип данных и начальное значение. Например: `val count: Int = 2`.
- При выводе типов можно опустить тип данных в объявлении переменной, если указано начальное значение. Некоторые распространенные базовые типы данных Kotlin включают: `Int`, `String`, `Boolean`, `Float` и `Double`.
- Используйте оператор присваивания (`=`), чтобы присвоить переменной значение либо при объявлении переменной, либо при ее обновлении. Обновлять можно только ту переменную, которая была объявлена как изменяемая (с помощью `var`).
- Используйте оператор инкремента (`++`) или декремента (`--`), чтобы увеличить или уменьшить значение целочисленной переменной на 1, соответственно. Используйте символ `+` для объединения строк. Вы также можете объединять в строки переменные других типов данных, например `Int` и `Boolean`.