

Немного продолжения про Git

Создание репозитория.

Создать одиночный репозиторий, через git init.

Создать папку с проектом, зайти туда через консоль и понеслась:

```
git init -> git add . -> git commit -m "git init" -> git push origin master
```

Рассказать, как в один приватный репозиторий добавить несколько товарищей по команде.

Введение в язык Kotlin

Что такое Kotlin. Первая программа

Kotlin представляет современный, статически типизированный и один из самых быстроразвивающихся языков программирования, созданный и развиваемый компанией JetBrains. Kotlin можно использовать для создания самых разных приложений. Это и приложения для мобильных устройств - Android, iOS. Причем Kotlin позволяет писать кроссплатформенный код, который будет применяться на всех платформах. Это и веб-приложения, причем как серверные приложения, которые обрабатывают на стороне сервера - бекэнда, так и браузерные клиентские приложения - фронтенд. Kotlin также можно применять для создания десктопных приложений, для Data Science и так далее.

Таким образом, круг платформ, для которых можно создавать приложения на Kotlin, чрезвычайно широк - Windows, Linux, Mac OS, iOS, Android.

Самым популярным направлением, где применяется Kotlin, является прежде всего разработка под ОС Android. Причем настолько популярным, что компания Google на конференции Google I/O 2017 провозгласила Kotlin одним из официальных языков для разработки под Android (наряду с Java и C++), а инструменты по работе с данным языком были по умолчанию включены в функционал среды разработки Android Studio начиная с версии 3.0.

Официальный сайт языка - <https://kotlinlang.org/>, где можно найти самую последнюю и самую подробную информацию по языку.

Kotlin испытал влияние многих языков: Java, Scala, Groovy, C#, JavaScript, Swift и позволяет писать программы как в объектно-ориентированном, так и в функциональном стиле. Он имеет ясный и понятный синтаксис и довольно легок для обучения.

Также стоит отметить, что Kotlin развивается как opensource, исходный код проекта можно посмотреть в репозитории на github по адресу <https://github.com/JetBrains/kotlin/>.

Примеры приложений написанных на kotlin:

- Pinterest. Популярное приложение для обмена фотографиями Pinterest является одним из самых больших имен, которые использовали Kotlin для разработки приложений для Android.
- Uber. Ведущая служба такси.
- Evernote. Приложение для ведения заметок и организации задач.

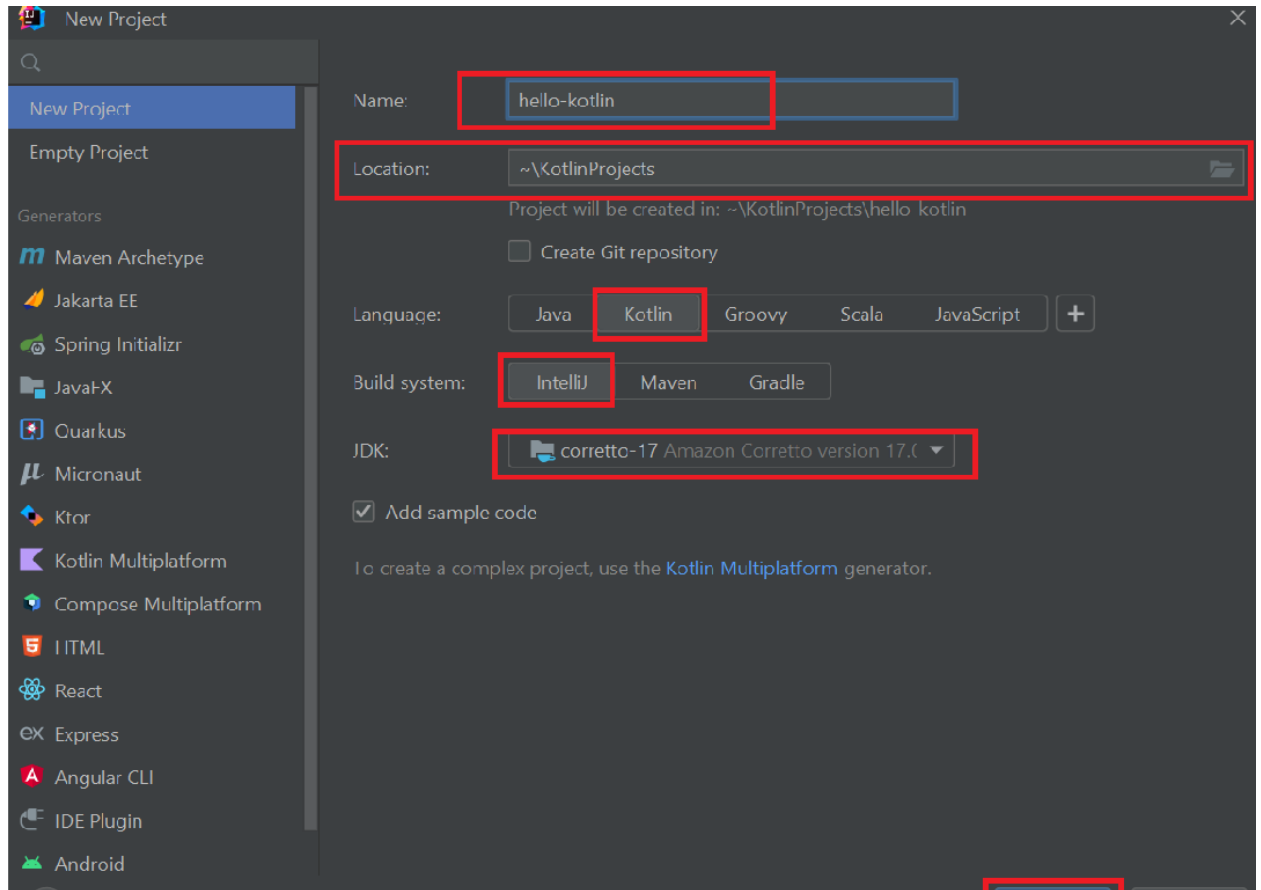
Установка IntelliJ IDEA

<https://www.jetbrains.com/ru-ru/idea/download/#section=windows>

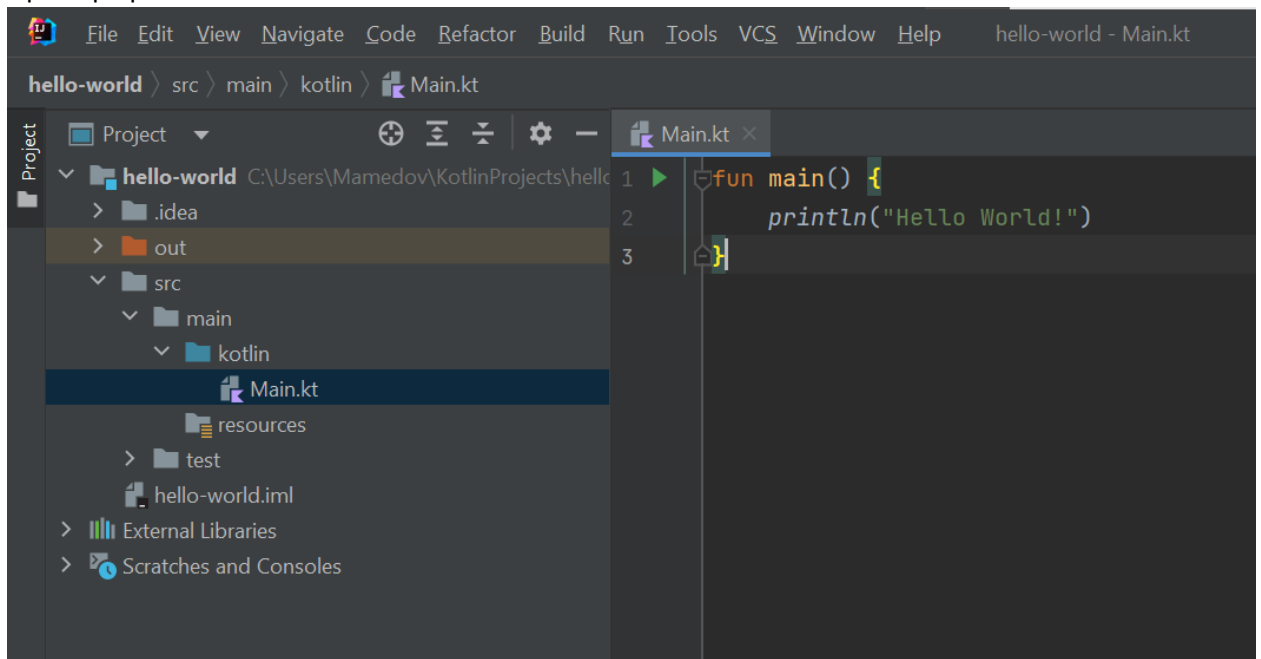
Скачивать Community версию.

Первая программа на Kotlin

Создание проекта:



Пример проекта:



Точкой входа в программу на Kotlin является функция `main`. Для определения функции применяется ключевое слово `fun`, после которого идет название функции - то есть `main`. Данная функция не принимает никаких параметров, поэтому после названия функции указываются пустые скобки.

Далее в фигурных скобках определяются собственно те действия, которые выполняет функция `main`. В данном случае внутри функции `main` выполняется другая функция - `println()`, которая выводит некоторое сообщение на консоль.

Вывод программы:



```
Run: MainKt x
C:\Users\Mamedov\.jdk\corretto-17.0.4.1\bin\java.exe
Hello, World!
Process finished with exit code 0
```

Основы языка Kotlin

Функция `main`

Точкой входа в программу на языке Kotlin является функция `main`. Именно с этой функции начинается выполнение программы на Kotlin, поэтому эта функция должна быть в любой программе на языке Kotlin.



```
Main.kt x .gitignore x
mamedov *
1 fun main() {
2     println("Hello, World!")
3 }
```

Определение функции `main()` (в принципе как и других функций в Kotlin) начинается с ключевого слова `fun`. По сути, оно указывает, что дальше идет определение функции. После `fun` указывается имя функции. В данном случае это `main`.

После имени функции в скобках идет список параметров функции. Здесь функция `main` не принимает никаких параметров, поэтому после имени функции идут пустые скобки.

Все действия, которые выполняет функция, заключаются в фигурные скобки. В данном случае единственное, что делает функция `main`, - вывод на консоль некоторого сообщения с помощью другой встроенной функции `println()`.

Стоит отметить, что до версии 1.3 в Kotlin функция `main` должна была принимать параметры:



```
1 fun main(args: Array<String>) {  
2     println("Hello, World!")  
3 }
```

Параметр `args: Array<String>` представляет массив строк, через который в программу можно передать различные данные.

Начиная с версии 1.3 использовать это определение функции с параметрами необязательно. Хотя мы можем его использовать.

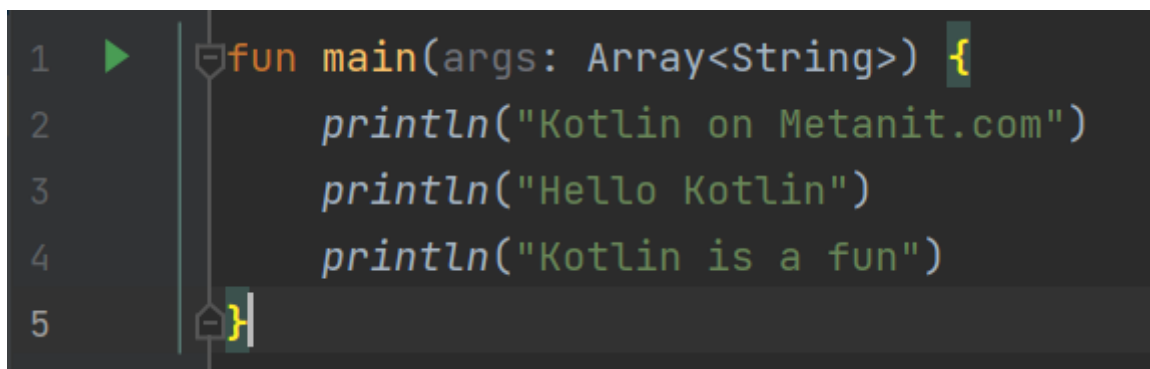
Инструкции и блоки кода

Основным строительным блоком программы на языке Kotlin являются инструкции (statement). Каждая инструкция выполняет некоторое действие, например, вызовы функций, объявление переменных и присвоение им значений. Например:

```
println("Hello, World!")
```

Данная строка представляет встроенной функции `println()`, которая выводит на консоль, некоторое сообщение (в данном случае строку "Hello Kotlin!").

Стоит отметить, что в отличие от других похожих языков программирования, например, Java, в Kotlin не обязательно ставить после инструкции точку с запятой. Каждая инструкция просто размещается на новой строке:



```
1 fun main(args: Array<String>) {  
2     println("Kotlin on Metanit.com")  
3     println("Hello Kotlin")  
4     println("Kotlin is a fun")  
5 }
```

Тем не менее, если инструкции располагаются на одной строке, то чтобы их отделить друг от друга, надо указывать после инструкции точку с запятой:



```
1 fun main(args: Array<String>) {  
2     println("Kotlin on Metanit.com");println("Hello Kotlin");println("Kotlin is a fun")  
3 }
```

Комментарии

Код программы может содержать комментарии. Комментарии позволяют понять смысл программы, что делают те или иные ее части. При компиляции комментарии игнорируются и не оказывают никакого влияния на работу приложения и на его размер.

В Kotlin есть два типа комментариев: однострочный и многострочный. Однострочный комментарий размещается на одной строке после двойного слеша `//`. А многострочный комментарий заключается между символами `/* текст комментария */`. Он может размещаться на нескольких строках. Например:

```
1  /*
2      многострочный комментарий
3      Функция main -
4      точка входа в программу
5  */
6  new *
7  fun main() {           // начало функции main
8      println("Hello Kotlin") // вывод строки на консоль
9  }
```

Переменные

Для хранения данных в программе в Kotlin, как и в других языках программирования, применяются переменные. Переменная представляет именованный участок памяти, который хранит некоторое значение.

Каждая переменная характеризуется определенным именем, типом данных и значением. Имя переменной представляет произвольный идентификатор, который может содержать алфавитно-цифровые символы или символ подчеркивания и должен начинаться либо с алфавитного символа, либо со знака подчеркивания. Для определения переменной можно использовать либо ключевое слово `val`, либо ключевое слово `var`.

Формальное определение переменной:

```
9  /*
10     val|var имя_переменной: тип_переменной
11  */
```

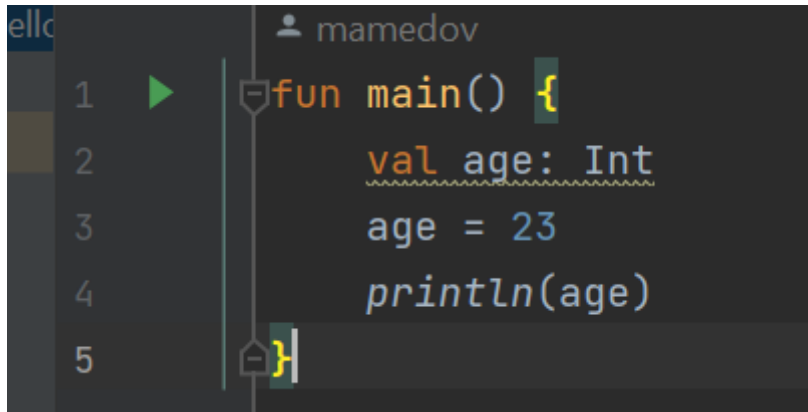
Вначале идет слово `val` или `var`, затем имя переменной и через двоеточие тип переменной.

Например, определим переменную `age`:

```
8  val age: Int
```

То есть в данном случае объявлена переменная `age`, которая имеет тип `Int`. Тип `Int` говорит о том, что переменная будет содержать целочисленные значения.

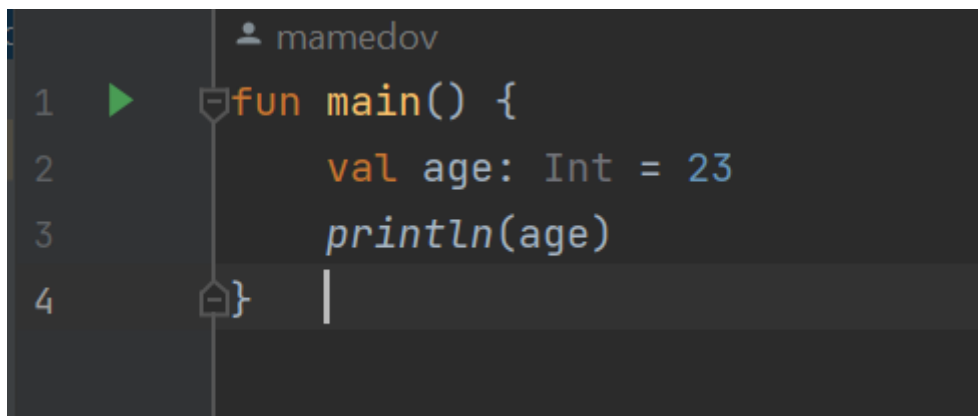
После определения переменной ей можно присвоить значение:



```
1 fun main() {  
2     val age: Int  
3     age = 23  
4     println(age)  
5 }
```

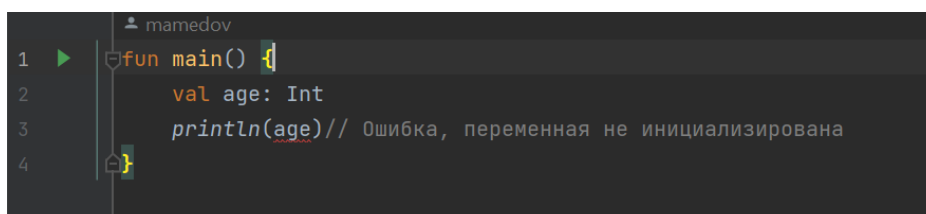
Для присвоения значения переменной используется знак равно. Затем мы можем производить с переменной различные операции. Например, в данном случае с помощью функции `println()` значение переменной выводится на консоль. И при запуске этой программы на консоль будет выведено число 23.

Присвоение значения переменной должно производиться только после ее объявления. И также мы можем сразу присвоить переменной начальное значение при ее объявлении. Такой прием называется инициализацией:



```
1 fun main() {  
2     val age: Int = 23  
3     println(age)  
4 }
```

Однако обязательно надо присвоить переменной некоторое значение до ее использования:



```
1 fun main() {  
2     val age: Int  
3     println(age) // Ошибка, переменная не инициализирована  
4 }
```

Изменяемые и неизменяемые переменные

Выше было сказано, что переменные могут объявляться как с помощью слова `val`, так и с помощью слова `var`. В чем же разница между двумя этими способами?

С помощью ключевого слова `val` определяется неизменяемая переменная (*immutable variable*). То есть мы можем присвоить значение такой переменной только один раз, но изменить его после первого присвоения мы уже не сможем. Например, в следующем случае мы получим ошибку:

```

1  fun main() {
2      val age: Int
3      age = 23      // здесь норм - первое присвоение
4      age = 56      // здесь ошибка - переопределить значение переменной нельзя
5      println(age)
6  }

```

А у переменной, которая определена с помощью ключевого слова `var` мы можем многократно менять значения (mutable variable):

```

1  fun main() {
2      var age: Int
3      age = 23
4      println(age)
5      age = 56
6      println(age)
7  }

```

Поэтому если не планируется изменять значение переменной в программе, то лучше определять ее с ключевым словом `val`.

Типы данных

В Kotlin все компоненты программы, в том числе переменные, представляют объекты, которые имеют определенный тип данных. Тип данных определяет, какой размер памяти может занимать объект данного типа и какие операции с ним можно производить. В Kotlin есть несколько базовых типов данных: числа, символы, строки, логический тип и массивы.

Целочисленные типы

- **Byte**: хранит целое число от -128 до 127 и занимает 1 байт
- **Short**: хранит целое число от -32 768 до 32 767 и занимает 2 байта
- **Int**: хранит целое число от -2 147 483 648 (-2^{31}) до 2 147 483 647 ($2^{31} - 1$) и занимает 4 байта
- **Long**: хранит целое число от -9 223 372 036 854 775 808 (-2^{63}) до 9 223 372 036 854 775 807 ($2^{63}-1$) и занимает 8 байт

В последней версии Kotlin также добавлена поддержка для целочисленных типов без знака:

- **UByte**: хранит целое число от 0 до 255 и занимает 1 байт
- **UShort**: хранит целое число от 0 до 65 535 и занимает 2 байта
- **UInt**: хранит целое число от 0 до $2^{32} - 1$ и занимает 4 байта
- **ULong**: хранит целое число от 0 до $2^{64}-1$ и занимает 8 байт

Объекты целочисленных типов хранят целые числа:

```
mamedov
1  fun main() {
2
3      val a: Byte = -10
4      val b: Short = 45
5      val c: Int = -250
6      val d: Long = 30000
7      println(a) // -10
8      println(b) // 45
9      println(c) // -250
10     println(d) // 30000
11 }
```

Для передачи значений объектам, которые представляют беззнаковые целочисленные типы данных, после числа указывается суффикс U:

```
mamedov
1  fun main() {
2
3      val a: UByte = 10U
4      val b: UShort = 45U
5      val c: UInt = 250U
6      val d: ULong = 30000U
7      println(a) // 10
8      println(b) // 45
9      println(c) // 250
10     println(d) // 30000
11 }
```

Кроме чисел в десятичной системе мы можем определять числа в двоичной и шестнадцатеричной системах.

Шестнадцатеричная запись числа начинается с 0x, затем идет набор символов от 0 до F, которые представляют число:

```
mamedov
1  fun main() {
2      val address: Int = 0x0A1 // 161
3      println(address) // 161
4  }
```

No lines changed: content is already properly formatted

Двоичная запись числа предваряется символами 0b, после которых идет последовательность из нулей и единиц:


```

1  fun main() {
2      val a: Int = 0b0101    // 5
3      val b: Int = 0b1011    // 11
4      println(a)             // 5
5      println(b)             // 11
6  }

```

Числа с плавающей точкой

Кроме целочисленных типов в Kotlin есть два типа для чисел с плавающей точкой, которые позволяют хранить дробные числа:

- **Float**: хранит число с плавающей точкой от $-3.4 \cdot 10^{38}$ до $3.4 \cdot 10^{38}$ и занимает 4 байта
- **Double**: хранит число с плавающей точкой от $\pm 5.0 \cdot 10^{-324}$ до $\pm 1.7 \cdot 10^{308}$ и занимает 8 байта.

В качестве разделителя целой и дробной части применяется точка:

```

1  fun main() {
2      val height: Double = 1.78
3      val pi: Float = 3.14F
4      println(height)      // 1.78
5      println(pi)          // 3.14
6  }

```

Чтобы присвоить число объекту типа Float после числа указывается суффикс f или F.

Также тип Double поддерживает экспоненциальную запись:

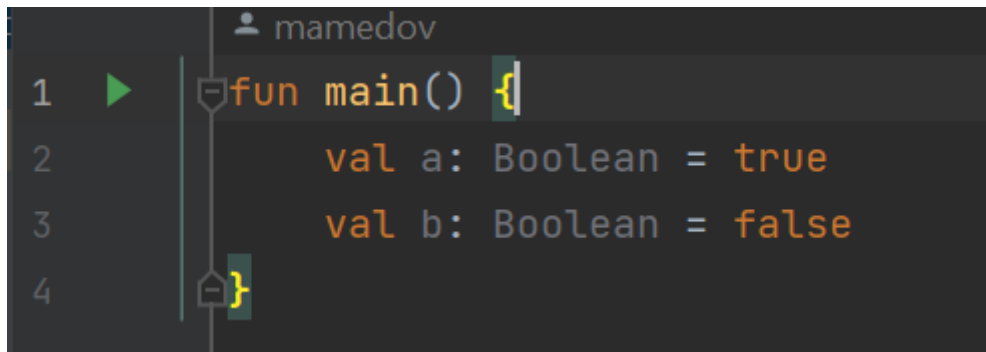
```

1  fun main() {
2      val d: Double = 23e3
3      println(d)      // 23 000
4      val g: Double = 23e-3
5      println(g)      // 0.023
6  }

```

Логический тип Boolean

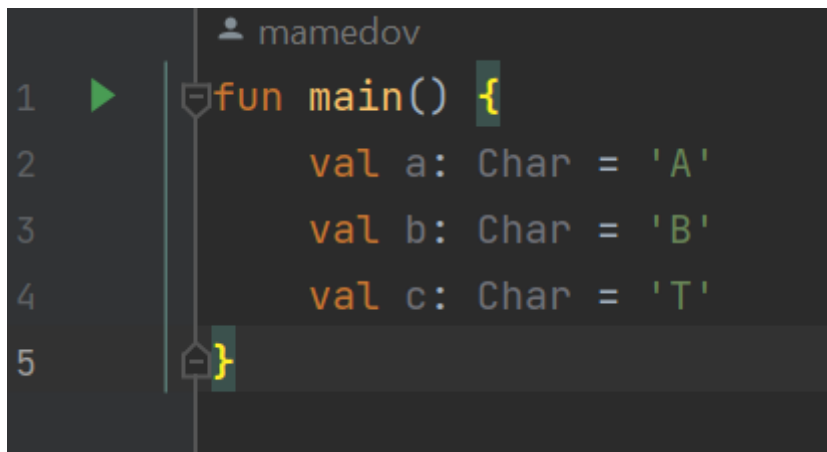
Тип Boolean может хранить одно из двух значений: true (истина) или false (ложь).

A screenshot of a code editor showing Kotlin code. The code defines a function `main()` with two `val` variables: `a` of type `Boolean` set to `true`, and `b` of type `Boolean` set to `false`. The code is as follows:

```
1 fun main() {  
2     val a: Boolean = true  
3     val b: Boolean = false  
4 }
```

Символы

Символьные данные представлены типом `Char`. Он представляет отдельный символ, который заключается в одинарные кавычки.

A screenshot of a code editor showing Kotlin code. The code defines a function `main()` with three `val` variables: `a` of type `Char` set to `'A'`, `b` of type `Char` set to `'B'`, and `c` of type `Char` set to `'T'`. The code is as follows:

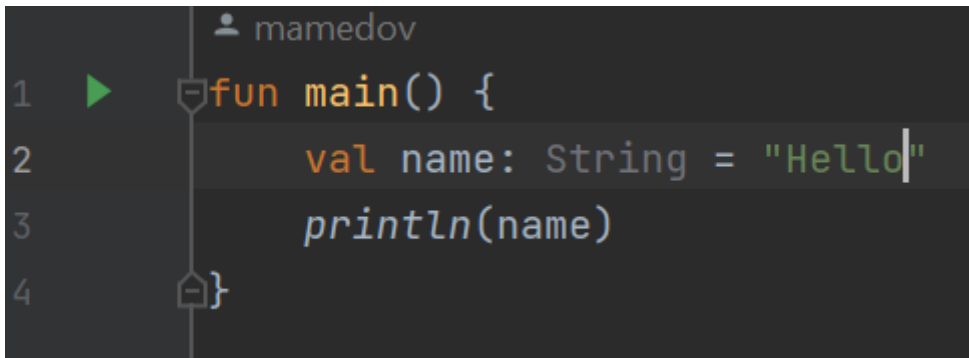
```
1 fun main() {  
2     val a: Char = 'A'  
3     val b: Char = 'B'  
4     val c: Char = 'T'  
5 }
```

Также тип `Char` может представлять специальные последовательности, которые интерпретируются особым образом:

- `\t`: табуляция
- `\n`: перевод строки
- `\r`: возврат каретки
- `\'`: одинарная кавычка
- `\"`: двойная кавычка
- `\\`: обратный слеш

Строки

Строки представлены типом `String`. Строка представляет последовательность символов, заключенную в двойные кавычки, либо в тройные двойные кавычки.

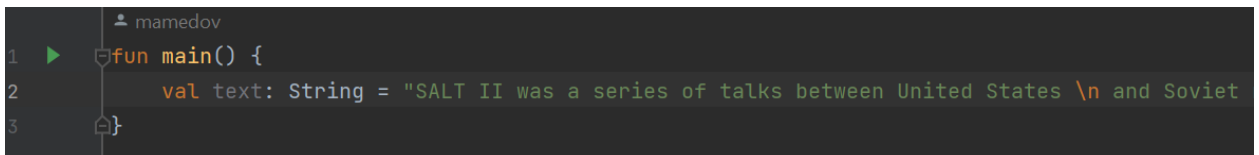


```

1  fun main() {
2      val name: String = "Hello"
3      println(name)
4  }

```

Строка может содержать специальные символы или эскейп-последовательности. Например, если необходимо вставить в текст перевод на другую строку, можно использовать эскейп-последовательность `\n`:

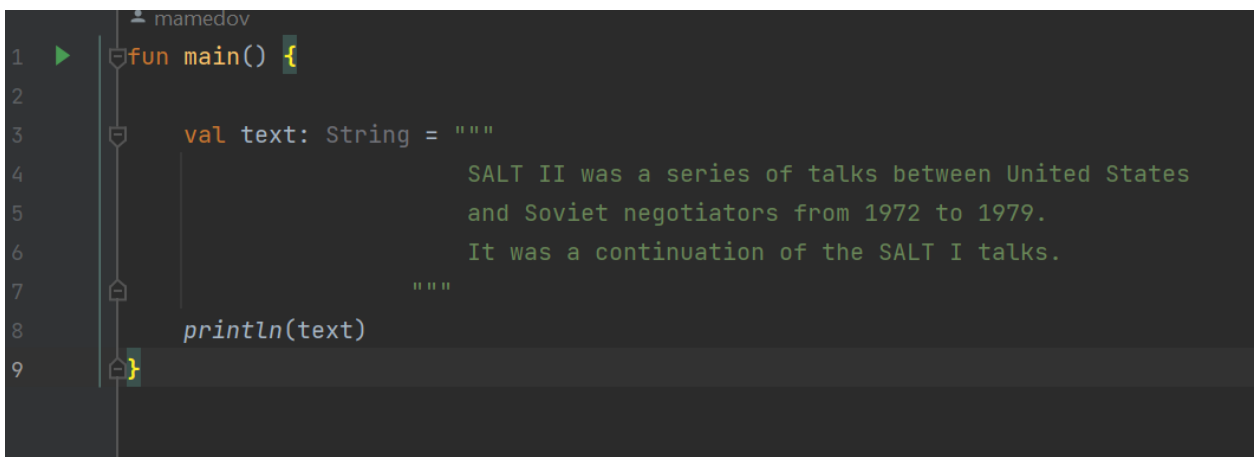


```

1  fun main() {
2      val text: String = "SALT II was a series of talks between United States \n and Soviet
3  }

```

Для большего удобства при создании многострочного текста можно использовать тройные двойные кавычки:



```

1  fun main() {
2
3      val text: String = """
4          SALT II was a series of talks between United States
5          and Soviet negotiators from 1972 to 1979.
6          It was a continuation of the SALT I talks.
7          """
8      println(text)
9  }

```

Шаблоны строк



```

1  fun main() {
2      val firstName = "Tom"
3      val lastName = "Smith"
4      val welcome = "Hello, $firstName $lastName"
5      println(welcome) // Hello, Tom Smith
6  }

```

В данном случае вместо `$firstName` и `$lastName` будут вставляться значения этих переменных. При этом переменные необязательно должны представлять строковый тип:

```

1  fun main() {
2      val name = "Tom"
3      val age = 22
4      val userInfo = "Your name: $name  Your age: $age"
5  }

```

Выведение типа

Kotlin позволяет выводить тип переменной на основании данных, которыми переменная инициализируется. Поэтому при инициализации переменной тип можно опустить:

```

1  fun main() {
2      val age = 5
3      val name = "Tom"
4      val sum = 45L
5      val sumU = 45U
6      val height = 1.78
7      val weight = 1.78F
8  }

```

В данном случае компилятор увидит, что переменной присваивается значение типа `Int`, поэтому переменная `age` будет представлять тип `Int`.

Соответственно если мы присваиваем переменной строку, то такая переменная будет иметь тип `String`.

Любые целые числа, воспринимаются как данные типа `Int`.

Если же мы хотим явно указать, что число представляет значение типа `Long`, то следует использовать суффикс `L`:

Если надо указать, что объект представляет беззнаковый тип, то применяется суффикс `u` или `U`:

Однако нельзя сначала объявить переменную без указания типа, а потом где-то в программе присвоить ей какое-то значение:

```

1  fun main() {
2      val age      // Ошибка, переменная не инициализирована
3      age = 5
4  }

```

Статическая типизация и тип Any

Тип данных ограничивает набор значений, которые мы можем присвоить переменной. Например, мы не можем присвоить переменной типа Double строку:

```
1  fun main() {
2      val height: Double = "1.78"
3  }
```

И после того, как тип переменной установлен, он не может быть изменен:

```
mamedov
1  fun main() {
2      var height: String = "1.78"
3      height = 1.81      // !Ошибка - переменная height хранит только строки
4      println(height)
5  }
```

Однако в Kotlin также есть тип Any, который позволяет присвоить переменной данного типа любое значение:

```
mamedov
1  fun main() {
2      var name: Any = "Tom"
3      println(name)    // Tom
4      name = 6758
5      println(name)    // 6758
6  }
```

Консольный ввод и вывод

Вывод на консоль

Для вывода информации на консоль в Kotlin есть две встроенные функции:

- print()
- println()

Обе эти функции принимают некоторый объект, который надо вывести на консоль, обычно это строка. Различие между ними состоит в том, что функция println() при выводе на консоль добавляет перевод на новую строку:

```

1  fun main() {
2      print("Hello ")
3      print("Kotlin ")
4      println()
5      println("Kotlin is a fun")
6  }

```

Причем функция `println()` необязательно должна принимать некоторое значения. Так, здесь применяется пустой вызов функции, который просто перевод консольный вывод на новую строку:

Консольный вывод программы:

```

Run: MainKt x
C:\Users\Mamedov\.jdk\corretto-17.0
Hello Kotlin
Kotlin is a fun
Process finished with exit code 0

```

Ввод с консоли

Для ввода с консоли применяется встроенная функция `readLine()`. Она возвращает введенную строку. Стоит отметить, что результат этой функции всегда представляет объект типа `String`. Соответственно введенную строку мы можем передать в переменную типа `String`:

```

1  fun main() {
2      print("Введите имя: ")
3      val name = readLine()
4      println("Ваше имя: $name")
5  }

```

Здесь сначала выводится приглашение к вводу данных. Далее введенное значение передается в переменную `name`. Результат работы программы:

```
Run: MainKt x
C:\Users\Mamedov\.jdk\corretto-17.0.4.1\
Введите имя: Vaga
Ваше имя: Vaga
Process finished with exit code 0
```

Подобным образом можно вводить разные данные:

```
1 fun main() {
2
3     print("Введите имя: ")
4     val name = readLine()
5     print("Введите email: ")
6     val email = readLine()
7     print("Введите адрес: ")
8     val address = readLine()
9
10    println("Ваше имя: $name")
11    println("Ваш email: $email")
12    println("Ваш адрес: $address")
13 }
```

Пример работы программы:

```
Run: MainKt x
C:\Users\Mamedov\.jdk\corretto-17.0.4.1\bin\java.exe
Введите имя: Вагиф
Введите email: vagif_m@bk.ru
Введите адрес: Фридриха Энгельса ...
Ваше имя: Вагиф
Ваш email: vagif_m@bk.ru
Ваш адрес: Фридриха Энгельса ...
Process finished with exit code 0
```

Операции с числами

Арифметические операции

Kotlin поддерживает базовые арифметические операции:

- **+** (сложение): возвращает сумму двух чисел.

```
1  fun main() {  
2      val x = 5  
3      val y = 6  
4      val z = x + y  
5      println(z)    // z = 11  
6  }
```

- **-** (вычитание): возвращает разность двух чисел.

```
1  fun main() {  
2      val x = 5  
3      val y = 6  
4      val z = x - y  
5      println(z)    // z = -1  
6  }
```

- ***** (умножение): возвращает произведение двух чисел.

```
1  fun main() {  
2      val x = 5  
3      val y = 6  
4      val z = x * y  
5      println(z)    // z = 30  
6  }
```

- **/** (деление): возвращает частное двух чисел.

```
1  fun main() {  
2      val x = 60  
3      val y = 6  
4      val z = x / y  
5      println(z)    // z = 10  
6  }
```

При этом если в операции деления оба операнда представляют целые числа, то результатом тоже будет целое число, а если в процессе деления образовалась дробная часть, то она отбрасывается:


```

1  fun main() {
2      val x = 11
3      val y = 5
4      val z = x / y // z = 2
5      println(z) // 2
6  }

```

Так в данном случае, хотя если согласно стандартной математике разделить 11 на 5, то получится 2.2. Однако поскольку оба операнда представляют целочисленный тип, а именно тип `Int`, то дробная часть - 0.2 отбрасывается, поэтому результатом будет число 2, а переменная `z` будет представлять тип `Int`.

```

1  fun main() {
2      val x = 11
3      val y = 5.0
4      val z = x / y // z = 2.2
5      println(z) // 2.2
6  }

```

В данном случае переменная `y` представляет тип `Double`, поэтому результатом деления будет число 2.2, а переменная `z` также будет представлять тип `Double`

- `%`: возвращает остаток от целочисленного деления двух чисел.

```

1  fun main() {
2      val x = 65
3      val y = 10
4      val z = x % y // z = 5
5  }

```

- `++` (инкремент): увеличивает значение на единицу.

Префиксный инкремент возвращает увеличенное значение:

```

1  fun main() {
2      var x = 5
3      val y = ++x
4      println(x) // x = 6
5      println(y) // y = 6
6  }

```

Постфиксный инкремент возвращает значение до увеличения на единицу:

```
1 fun main() {
2     var x = 5
3     val y = x++
4     println(x)    // x = 6
5     println(y)    // y = 5
6 }
```

- -- (декремент): уменьшает значение на единицу.

Префиксный декремент возвращает уменьшенное значение:

```
1 fun main() {
2     var x = 5
3     val y = --x
4     println(x)    // x = 4
5     println(y)    // y = 4
6 }
```

Постфиксный декремент возвращает значение до уменьшения на единицу:

```
1 fun main() {
2     var x = 5
3     val y = x--
4     println(x)    // x = 4
5     println(y)    // y = 5
6 }
```

No lines changed: content is already properly formatted

Также есть ряд операций присвоения, которые сочетают арифметические операции и присвоение:

- +=: присваивание после сложения. Присваивает левому операнду сумму левого и правого операндов: $A += B$ эквивалентно $A = A + B$
- -=: присваивание после вычитания. Присваивает левому операнду разность левого и правого операндов: $A -= B$ эквивалентно $A = A - B$
- *=: присваивание после умножения. Присваивает левому операнду произведение левого и правого операндов: $A *= B$ эквивалентно $A = A * B$
- /=: присваивание после деления. Присваивает левому операнду частное левого и правого операндов: $A /= B$ эквивалентно $A = A / B$
- %=: присваивание после деления по модулю. Присваивает левому операнду остаток от целочисленного деления левого операнда на правый: $A \% = B$ эквивалентно $A = A \% B$

ДЛЯ ОБЩЕГО РАЗВИТИЯ (НЕ ОБЯЗАТЕЛЬНО!)

Поразрядные операции

Ряд операций выполняется над двоичными разрядами числа. Здесь важно понимать, как выглядит двоичное представление тех или иных чисел. В частности, число 4 в двоичном виде - 100, а число 15 - 1111.

Есть следующие поразрядные операторы (они применяются только к данным типов Int и Long):

- shl: сдвиг битов числа со знаком влево

```
mamedov
1 fun main() {
2     val z = 3 shl 2    // z = 11 << 2 = 1100
3     println(z)        // z = 12
4     val d = 0b11 shl 2
5     println(d)        // d = 12
6 }
```

В данном случае число сдвигается на два разряда влево, поэтому справа число в двоичном виде дополняется двумя нулями. То есть в двоичном виде 3 представляет 11. Сдвигаем на два разряда влево (дополняем справа двумя нулями) и получаем 1100, то есть в десятичной системе число 12.

- shr: сдвиг битов числа со знаком вправо

```
mamedov
1 fun main() {
2     val z = 12 shr 2    // z = 1100 >> 2 = 11
3     println(z)         // z = 3
4     val d = 0b1100 shr 2
5     println(d)         // d = 3
6 }
```

Число 12 сдвигается на два разряда вправо, то есть два числа справа фактически отбрасываем и получаем число 11, то есть 3 в десятичной системе.

- ushr: сдвиг битов беззнакового числа вправо

```
mamedov
1 fun main() {
2     val z = 12 ushr 2    // z = 1100 >> 2 = 11
3     println(z)         // z = 3
4 }
```

- and: побитовая операция AND (логическое умножение или конъюнкция). Эта операция сравнивает соответствующие разряды двух чисел и возвращает единицу, если эти разряды обоих чисел равны 1. Иначе возвращает 0.

```
mamedov
1 fun main() {
2     val x = 5    // 101
3     val y = 6    // 110
4     val z = x and y    // z = 101 & 110 = 100
5     println(z)        // z = 4
6
7     val d = 0b101 and 0b110
8     println(d)        // d = 4
9 }
```

- or: побитовая операция OR (логическое сложение или дизъюнкция). Эта операция сравнивает два соответствующих разряда обоих чисел и возвращает 1, если хотя бы один разряд равен 1. Если оба разряда равны 0, то возвращается 0.

```
mamedov
1  ▶ fun main() {
2      val x = 5    // 101
3      val y = 6    // 110
4      val z = x or y    // z = 101 | 110 = 111
5      println(z)        // z = 7
6
7      val d = 0b101 or 0b110
8      println(d)        // d = 7
9  }
```

- xor: побитовая операция XOR. Сравнивает два разряда и возвращает 1, если один из разрядов равен 1, а другой равен 0. Если оба разряда равны, то возвращается 0.

```
mamedov
1  ▶ fun main() {
2      val x = 5    // 101
3      val y = 6    // 110
4      val z = x xor y    // z = 101 ^ 110 = 011
5      println(z)        // z = 3
6
7      val d = 0b101 xor 0b110
8      println(d)        // d = 3
9  }
```

- inv: логическое отрицание или инверсия - инвертирует биты числа

```
mamedov
1  ▶ fun main() {
2      val b = 11    // 1011
3      val c = b.inv()
4      println(c)    // -12
5  }
```