1.1. Введение в язык Kotlin

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 1.1. Введение в язык Kotlin

Напечатано:: Павел Степанов
Дата: понедельник, 9 октября 2023, 11:04

Оглавление

1.1.1. Язык Kotlin

Kotlin (Ко́тлин) - статически типизированный язык программирования, работающий поверх Java Virtual Machine и разрабатываемый компанией *JetBrains*. Программы на Котлин компилируются в JavaScript и в исполняемый код ряда платформ через инфраструктуру LLVM (Low Level Virtual Machine). Компания JetBrains начала разработку языка в 2010 году и назвала его в честь острова Котлин в Финском заливе, на котором расположен город Кронштадт. Официальный релиз языка был 17 февраля 2016 года.

Авторы ставили целью создать язык более лаконичный и типобезопасный, чем Java, и более простой, чем Scala. Согласно утверждениям разработчиков, они хотели сделать практичный язык для максимально широкой аудитории.

Следствием упрощения по сравнению со Scala стали более быстрая компиляция и лучшая поддержка языка в IDE. Котлин полностью совместим с Java, что позволяет java-разработчикам постепенно перейти к его использованию; в частности, в Android язык встраивается с помощью Gradle, что позволяет для существующего android-приложения внедрять новые функции на Kotlin без переписывания приложения целиком.

Кроме того, в семантику языка Котлин заложены принципы, которые позволяют избежать некоторого числа распространенных ошибок, например null-безопасность, о которой рассказано в дальнейших разделах учебника.

1.1.2. Первая программа на Котлин. Вывод сообщений

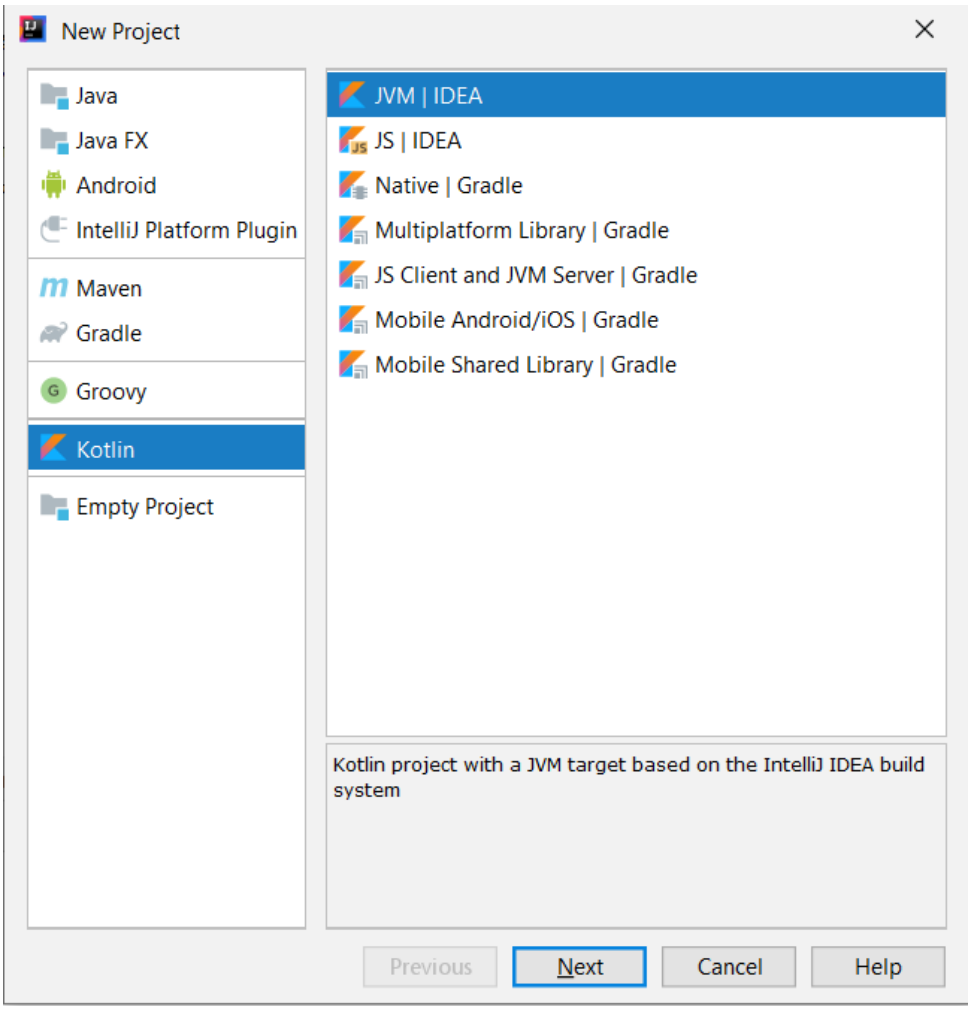
Средой разработки для языка Котлин в нашем курсе выбрана IntelliJ Idea. Первый шаг - скачать эту среду с сайта разработчиков языка Котлин JetBrains и установить ее на свой компьютер.

Подробно прочитать про установку и настройку среду IntelliJ Idea можно здесь.

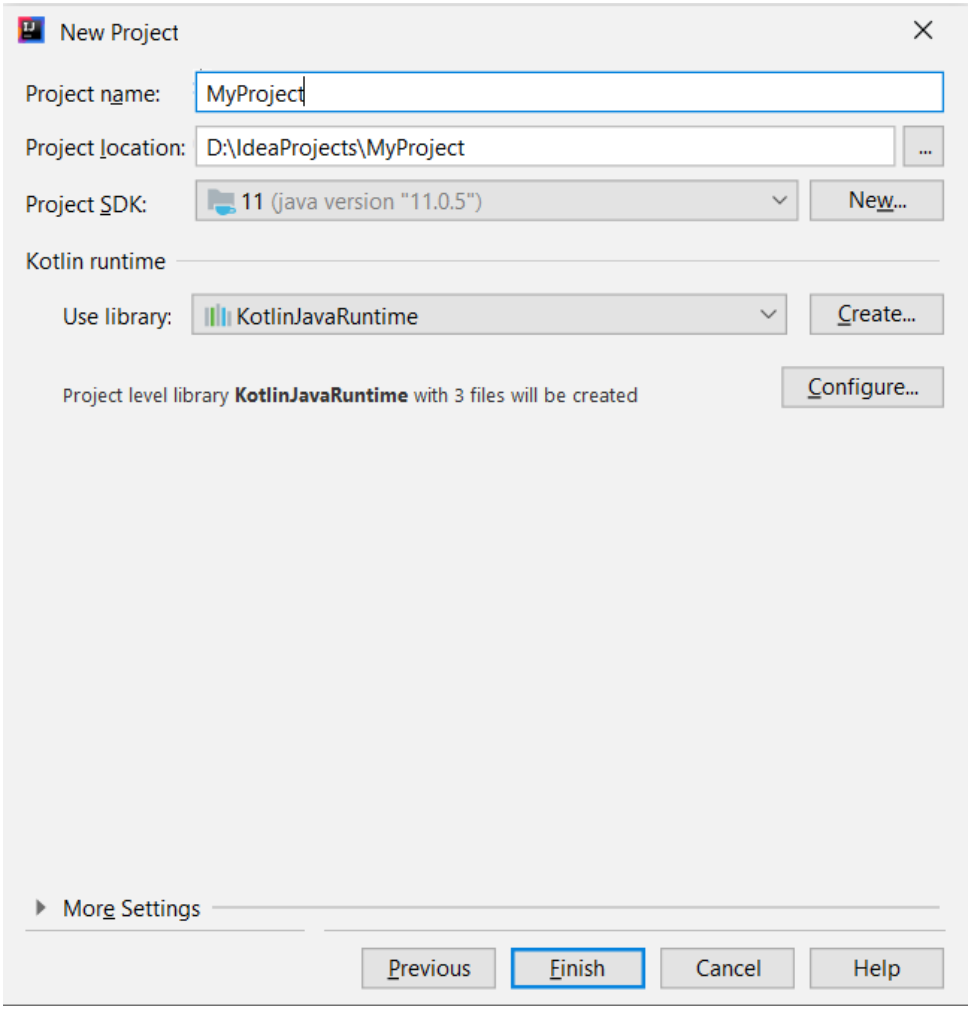
Итак, считаем, что среда разработки скачана и установлена. Создадим первый проект, выведем на экран традиционное приветствие и представимся сами.

Шаг 1. Создание проекта

Если раньше Intellij Idea не использовалась, то вы увидите небольшой экран приветствия, на котором следует выбрать пункт "Create new project". Если же среда уже использовалась, то при запуске будет открыт последний проект и создать новый можно через панель основного меню File -> New -> Project. Попадаем в окно мастера создания нового проекта (рис. 1)



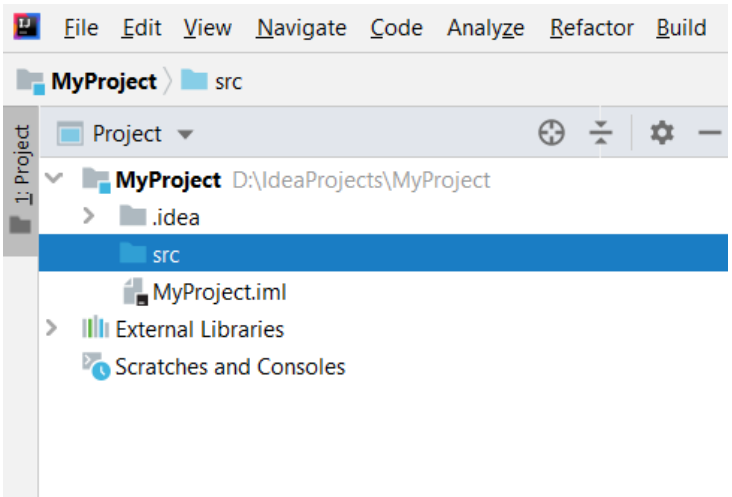
В этом окне мы вибираем Котлин среди пунктов меню слева и справа выбираем JVM (Java Virtual Machine) как основу для компиляции программы. Переходим к следующему окну создания проекта, которое предполагает ввод имени проекта (рис 2).



Кроме имени проекта в этом окне можно определить местоположение проекта в системе, версию java, на которой базируется Котлин а также библиотеку KotlinJavaRunTime, которую рекомендуется оставить как предлагает среда. После нажатия кнопки "Finish" Intellij создаст пустой проект, не содержащий файлов.

Шаг 2. Создание файла/класса Котлин

Окно проекта содержит две части: слева - структура проекта (рис. 3), справа (гораздо больше по размеру) окно для написания кода. Сразу после файлов в проекте нет.

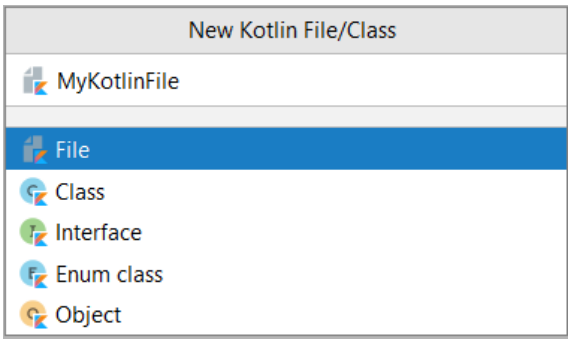


В структуре проекта есть:

- папка src - для будущих файлов исходного кода;
- внешние библиотеки, необходимые для обеспечения функциональности программы;

- различные настроечные и временные файлы.

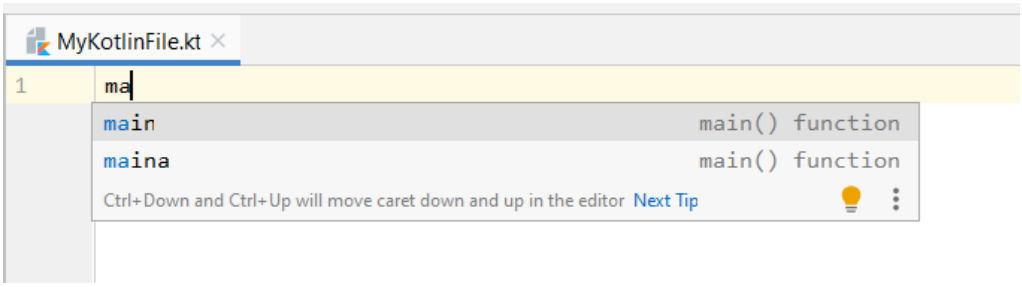
Для добавления в папку src файла исходного кода нужно в контекстном меню src выбрать new -> Kotlin File/Class или сделать то же самое через верхнее меню. Среда предложит дать имя файлу (рис. 4).



При выборе имени файла не обязательно писать расширение. Расширение файлов с исходным кодом Котлин - *.kt - среда подставит автоматически.

Шаг 3. Пишем простую программу

Новый файл не содержит никаких шаблонов. Чтобы начать писать программу нужно создать функцию main. с которой и начинается выполнение программы на JVM. Для создания функции можно воспользоваться контекстной подсказкой. Начнем набирать имя функции и среда предложит два варианта (рис. 5)



При выборе первого варианта функция main создается без параметров

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

Open in Playground → Target: JVM Running on v.1.9.10

При выборе второго варианта контекстной подсказки функция получит параметром строковый массив.

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

Open in Playground → Target: JVM Running on v.1.9.10

Напишем внутри функции традиционное приветствие "Hello, World!!!". Для вывода на экран сообщения, а в последующем и значений переменных используется функция `println()`:

```
fun main() {
    println("Hello, World!!!")
}
```

Open in Playground → Target: JVM Running on v.1.9.10

Обратите внимание, что точка с запятой как оператор завершения инструкции не используется. Выводимая строка заключается в двойные кавычки.

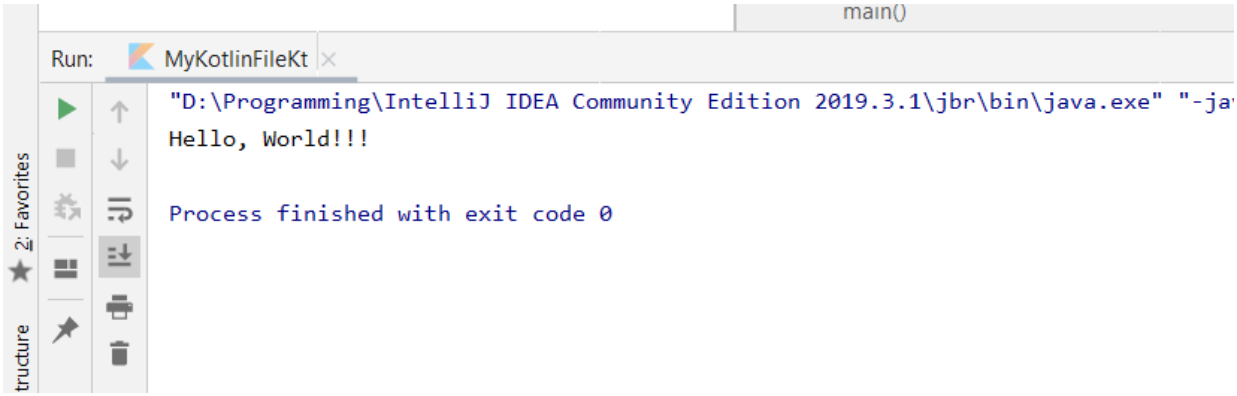
Шаг 4. Запуск программы

Для запуска программы впервые можно использовать два способа:

1. Через основное меню: Run -> run
2. Через контекстное меню, вызываемое кликом правой кнопки мыши на имени файла. В контекстном меню также выбираем Run.

После первого запуска активируется кнопка быстрого запуска (зеленая стрелка в правом верхнем углу окна кода программы) и в дальнейшем можно использовать эту кнопку.

Результат работы программы можно увидеть в консоли, которая появляется в нижней части окна (рис. 6). В этой же консоли выполняется ввод данных.



Наша первая программа завершена.

1.1.3. Базовые типы данных. Переменные. Символьные литералы.

В языке Котлин, в отличие от Java, нет понятия примитивных типов. Все типы представляют собой объекты со свойствами и методами. Однако, как можно заметить из предыдущего раздела, Котлин разрешает создавать функции и писать простые программы не используя классов явным образом.

Числа

Kotlin обрабатывает численные типы примерно так же, как и Java, хотя некоторые различия всё же присутствуют. Например, отсутствует неявное расширяющее преобразование для чисел, а литералы в некоторых случаях немного отличаются.

Для представления чисел в Kotlin используются следующие встроенные типы (подобные типам в Java):

Тип	Количество бит
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

Обратите внимание, что символы (characters) в языке Kotlin не являются числами (в отличие от Java).

Объявление переменных

В Java мы сначала указываем тип переменной, а потом её имя. В Kotlin немного не так. Сначала вы указываете ключевое слово val или var, затем имя переменной и по желанию можете указать тип переменной.

Если вы не инициализируете переменную, то тип указать нужно обязательно.

```
val age: Int
age = 7
```

Open in Playground → Target: JVM Running on v.1.9.10

Ключевое слово val (от слова value) ссылается на неизменяемую переменную, что соответствует ключевому слову final в Java.

Для обычных изменяемых переменных используется ключевое слово var (от слова variable).

Рекомендуется всегда использовать val, если это позволяет логика программы.

При этом нужно помнить, что хотя ссылка val неизменяема, сам объект может быть изменяемым.

Символьные литералы

В языке Kotlin присутствуют следующие виды символьных постоянных (констант) для целых значений:

Десятичные числа: 123
Тип Long обозначается заглавной L: 123L
Шестнадцатеричные числа: 0x0F
Двоичные числа: 0b00001011
ВНИМАНИЕ: Восьмеричные литералы не поддерживаются.

Также Kotlin поддерживает числа с плавающей запятой:

Тип Double по умолчанию: 123.5, 123.5e10
Тип Float обозначается с помощью f или F: 123.5f
Нижние подчеркивания в числовых литералах (начиная с версии 1.1)
Вы можете использовать нижние подчеркивания, чтобы сделать числовые константы более читаемыми:

```
val oneMillion = 1_000_000
val creditCardNumber = 1234_5678_9012_3456L
val socialSecurityNumber = 999_99_9999L
val hexBytes = 0xFF_EC_DE_5E
```

```
val bytes = 0b11010010_01101001_10010100_10010010
```

Open in Playground → Target: JVM Running on v.1.9.10

Символы

Символы в Kotlin представлены типом Char. Напрямую они не могут рассматриваться в качестве чисел

```
fun check(c: Char) {
  if (c == 1) { // ОШИБКА: несовместимый тип
    // ...
  }
}
```

Open in Playground → Target: JVM Running on v.1.9.10

Символьные литералы записываются в одинарных кавычках: '1', '\n', '\uFF00'. Мы можем явно привести символ в число типа Int

```
fun decimalDigitValue(c: Char): Int {
  if (c !in '0'..'9')
    throw IllegalArgumentException("Вне диапазона")
  return c.toInt() - '0'.toInt() // Явные преобразования в число
}
```

Open in Playground → Target: JVM Running on v.1.9.10

Подобно числам, символы оборачиваются при необходимости использования nullable ссылки. При использовании обёрток тождественность (равенство по ссылке) не сохраняется.

Логический тип

Тип Boolean представляет логический тип данных и принимает два значения: true и false.

При необходимости использования nullable ссылок логические переменные оборачиваются.

Встроенные действия над логическими переменными включают

|| – ленивое логическое ИЛИ
&& – ленивое логическое И
! - отрицание

1.1.4. Ветвления

Условное выражение if

В языке Kotlin ключевое слово if является выражением, т.е. оно возвращает значение. Это позволяет отказаться от тернарного оператора (условие ? условие истинно : условие ложно), поскольку выражению if вполне по силам его заменить.

```
// обычное использование
var max = a
if (a < b)
  max = b

// с блоком else
var max: Int
if (a > b)
  max = a
else
  max = b

// в виде выражения
val max = if (a > b) a else b
```

Open in Playground → Target: JVM Running on v.1.9.10

"Ветви" выражения if могут содержать несколько строк кода, при этом последнее выражение является значением блока:

```
val max = if (a > b) {
    print("возвращаем a")
    a
  }
  else {
    print("возвращаем b")
    b
  }
```

Open in Playground → Target: JVM Running on v.1.9.10

Если вы используете конструкцию if в качестве выражения (например, возвращая его значение или присваивая его переменной), то использование ветки else является обязательным.

См. использование if.

Условное выражение when

Ключевое слово when призвано заменить оператор switch, присутствующий в C-подобных языках. В простейшем виде его использование выглядит так:

```
when (x) {
  1 -> print("x == 1")
  2 -> print("x == 2")
  else -> { // обратите внимание на блок
    print("x is neither 1 nor 2")
  }
}
```

Open in Playground → Target: JVM Running on v.1.9.10

Выражение when последовательно сравнивает аргумент со всеми указанными значениями до удовлетворения одного из условий. when можно использовать и как выражение, и как оператор. При использовании в виде выражения значение ветки, удовлетворяющей условию, становится значением всего выражения. При использовании в виде оператора значения отдельных веток отбрасываются. (В точности как if: каждая ветвь может быть блоком и её значением является значение последнего выражения блока.)

Значение ветки else вычисляется в том случае, когда ни одно из условий в других ветках не удовлетворено. Если when используется как выражение, то ветка else является обязательной, за исключением случаев, в которых компилятор может убедиться, что ветки покрывают все возможные значения.

Если для нескольких значений выполняется одно и то же действие, то условия можно перечислять в одной ветке через запятую:

```
when (x) {
  0, 1 -> print("x == 0 or x == 1")
  else -> print("otherwise")
}
```

Open in Playground → Target: JVM Running on v.1.9.10

Помимо констант в ветках можно использовать произвольные выражения:

```
when (x) {
  parseInt(s) -> print("s encodes x")
  else -> print("s does not encode x")
}
```

Open in Playground → Target: JVM Running on v.1.9.10

Также можно проверять вхождение аргумента в интервал in или !in или его наличие в коллекции:

```
when (x) {
  in 1..10 -> print("x is in the range")
  in validNumbers -> print("x is valid")
  !in 10..20 -> print("x is outside the range")
  else -> print("none of the above")
}
```

Open in Playground → Target: JVM Running on v.1.9.10

Помимо этого Kotlin позволяет с помощью is или !is проверить тип аргумента. Обратите внимание, что благодаря умным приведениям вы можете получить доступ к методам и свойствам типа без дополнительной проверки:

```
val hasPrefix = when(x) {
  is String -> x.startsWith("prefix")
  else -> false
}
```

Open in Playground → Target: JVM Running on v.1.9.10

Оператор `when` также может служить аналогом switch/case:

```
val hourOfDay = 12
val timeOfDay: String
timeOfDay = when (hourOfDay) {
  0, 1, 2, 3, 4, 5 -> "Early morning"
  6, 7, 8, 9, 10, 11 -> "Morning"
  12, 13, 14, 15, 16 -> "Afternoon"
  17, 18, 19 -> "Evening"
  20, 21, 22, 23 -> "Late evening"
  else -> "Invalid"
}
```

Open in Playground → Target: JVM Running on v.1.9.10

when удобно использовать вместо цепочки условий вида if-else if. При отсутствии аргумента, условия работают как простые логические выражения, а тело ветки выполняется при его истинности:

```
when {
  x.isOdd() -> print("x is odd")
  x.isEven() -> print("x is even")
  else -> print("x is funny")
}
```

Open in Playground → Target: JVM Running on v.1.9.10

1.1.5. Циклические конструкции

Циклы while

Ключевые слова while и do..while работают как обычно:

```
while (x > 0) {
  x--
}

do {
  val y = retrieveData()
} while (y != null) // y здесь доступно!
```

Open in Playground → Target: JVM Running on v.1.9.10

См. использование while.

Ключевые слова break и continue в циклах

Kotlin поддерживает обычные операторы break и continue в циклах.

Циклы for

Цикл for обеспечивает перебор всех значений, поставляемых итератором. Для этого используется следующий синтаксис:

```
for (item in collection)
  print(item)
```

Open in Playground → Target: JVM Running on v.1.9.10

Телом цикла может быть блок кода:

```
for (item: Int in ints) {
  // ...
}
```

Open in Playground → Target: JVM Running on v.1.9.10

Как отмечено выше, цикл for позволяет проходить по всем элементам объекта, имеющего итератор, например:
обладающего внутренней или внешней функцией iterator(), возвращаемый тип которой
обладает внутренней или внешней функцией next(), и
обладает внутренней или внешней функцией hasNext(), возвращающей Boolean.

Все три указанные функции должны быть объявлены как operator.

Если при проходе по массиву или списку необходим порядковый номер элемента, используйте следующий подход:

```
for (i in array.indices)
  print(array[i])
```

Open in Playground → Target: JVM Running on v.1.9.10

Обратите внимание, что данная "итерация по ряду" компилируется в более производительный код без создания дополнительных объектов.

Также вы можете использовать библиотечную функцию withIndex:

```
for ((index, value) in array.withIndex()) {
    println("the element at $index is $value")
}
```

Open in Playground → Target: JVM Running on v.1.9.10

1.1.6. Строковые шаблоны

Строки

Строки в Kotlin представлены типом String

Строки в Kotlin представлены типом String. Строки являются неизменяемыми. Строки состоят из символов, которые могут быть получены по порядковому номеру: s[i]. Проход по строке выполняется циклом for:

```
for (c in str) {
  println(c)
}
```

Open in Playground → Target: JVM Running on v.1.9.10

Строковые литералы

В Kotlin представлены два типа строковых литералов: строки с экранированными символами и обычные строки, которые могут содержать символы новой строки и произвольный текст. Экранированная строка очень похожа на строку в Java:

```
val s = "Hello, world!\n"
```

Open in Playground → Target: JVM Running on v.1.9.10

Экранирование выполняется общепринятым способом, а именно с помощью обратной косой черты.

Обычная строка выделена тройной кавычкой ("""), не содержит экранированных символов, но может содержать символы новой строки и любые другие символы:

```
val text = """
  for (c in "foo")
    print(c)
"""
```

Open in Playground → Target: JVM Running on v.1.9.10

Строковые шаблоны

Строки могут содержать шаблонные выражения, т.е. участки кода, которые выполняются, а полученный результат встраивается в строку. Шаблон начинается со знака доллара ($) и состоит либо из простого имени (например, переменной):

```
val i = 10
val s = "i = $i" // evaluates to "i = 10"
```

Open in Playground → Target: JVM Running on v.1.9.10

либо из произвольного выражения в фигурных скобках:

```
val s = "abc"
val str = "$s.length is ${s.length}" // evaluates to "abc.length is 3"
```

Open in Playground → Target: JVM Running on v.1.9.10

Шаблоны поддерживаются как в обычных, так и в экранированных строках. При необходимости символ $ может быть представлен с помощью следующего синтаксиса:

```
val price = "${'$'}9.99"
```

Open in Playground → Target: JVM Running on v.1.9.10

1.1.7. Массивы

Массивы в Котлин также как и все остальные типы данных представляют собой классы. При создании массива нужно определить. для обработки каких данных предназначен массив. В зависимости от ответа на этот вопрос массивы можно разделить на:

- обобщенный класс Array<Тип элементов массива>;
- специализированные классы для обработки числовых и символьных данных: IntArray, DoubleArray, CharArray, BoobleanArray и т.д. Массивы этой категории оптимизированы для работы с соответствующими типами. Данные классы не наследуют класс Array, хотя и обладают тем же набором методов и свойств.

Примеры объявления переменных - массивов.

```
//массив вещественных чисел
var arrDouble : DoubleArray
//массив целых чисел
var arrIntArray: Array<Int>
//массив строк с разрешенным null значением
var arrStr: Array<String?>
```

Open in Playground → Target: JVM Running on v.1.9.10

Как и в Java, при объявлении массив еще не существует как последовательность значений. Массив нужно создать. Для создания массива в Котлин есть две возможности

- использовать библиотечную функцию arrayOf(), которой в качестве аргумента передаются элементы массива, т.е. выполнение arrayOf(1, 2, 3) создаёт массив [1, 2, 3]. Для оптимизированных массивов эта функция принимает вид: intArrayOf(), doubleArrayOf() и т.п.
- использовать конструктор класса, который принимает размер массива и функцию, возвращающую начальное значение каждого элемента по его индексу:

Рассмотрим примеры создания массивов.

```
//создаем массив из трех целых чисел, заполненный нулями
var array: IntArray = intArrayOf(0, 0, 0)
//создадим массив целых чисел из 10 элементов, заполенный нулями, при помощи конструктора
var arrayConstr = IntArray(10, {0})
// создадим массив типа Array со значениями ["0", "1", "4", "9", "16"]
val asc = Array(5, { i -> (i * i).toString() })
```

Open in Playground → Target: JVM Running on v.1.9.10

Обратите внимание на синтаксическое оформление второго аргумента конструктора - фигурные скобки. Второй аргумент - это функция, заполняющая массив значениями. Подробнее о такого рода функциях будет рассказано в следующих темах.

Обращение к элементам массива выглядит стандартно, например: `arrayConstr[3] = 12`, т.е. в квадратных скобках указывается индекс элемента. Нумерация элементов начинается с нуля. Размерность массива хранится в свойстве size, т.е. чтобы определить сколько элементов в массиве arrayConstr нужно использовать следующее обращение: `arrayConstr.size`

Обратите внимание: в отличие от Java массивы в Kotlin являются инвариантными. Это значит, что Kotlin запрещает нам присваивать массив Array<String> переменной типа Array<Any>, предотвращая таким образом возможный отказ во время исполнения.

Рассмотрим небольшой пример. Найдем сумму четных элементов массива. Массив сформируем случайным образом. Генератор случайных чисел взят из языка java для демонстрации тесной связи языков.

```
import java.util.*
fun main() {
    var rand = Random()
    val intAr = IntArray(10, {rand.nextInt(300)})
    var sum = 0
    for (i in intAr){
        print(i.toString()+", ")
        if (i % 2 == 0)
            sum += i
    }
    println(sum)
}
```

Open in Playground → Target: JVM Running on v.1.9.10

Двумерные массивы

В Котлин, как и в прочих языках программирования, существует возможность создавать многомерные массивы. Серьезное отличие в том, что при создании многомерного, в частности, двумерного массива необходимо явно написать, что элементами массива являются другие массивы. Иными словами, при создании двумерного массива нужно в конструкторе в качестве второго аргумента, определяющего формирование элементов строки указать еще один конструктор.

Например: `val intAr = Array(5, {IntArray(6, {0})})`. В этом примере создан массив из пяти строк, каждая из которых является целочисленным массивом.

Обработка элементов массива выполняется аналогично java и C/C++. Рассмотрим пример обработки двумерного массива. Сформируем массив случайным образом и найдем максимальное значение сумм строк. Обратите внимание на переменные циклов for. Их значения в данном примере не индексы элементов, а сами элементы массива.

```
import java.util.*
fun main() {
    var rand = Random()
    val intAr2 = Array(5, { IntArray(6, { rand.nextInt(100) }) })
    var max = Int.MIN_VALUE
    for (strArray in intAr2){
        var sum = 0
        for (num in strArray){
            sum += num
        }
        if (max < sum)
            max = sum
    }
    println(max)
}
```

Open in Playground → Target: JVM Running on v.1.9.10

Реализуем пример выше иначе, для ситуации, в которой индексы элементов важны.

```
import java.util.*
fun main() {
    var rand = Random()
    val intAr2 = Array(5, { IntArray(6, { rand.nextInt(100) }) })
    var max = Int.MIN_VALUE
    for (i in intAr2.indices){ //организуем цикл по индексам строк
        var sum = 0
        for (j in intAr2[i].indices){ // организуем цикл по индексам столбцов
            sum += intAr2[i][j] // обращение и элементу массива по индексам строки и столбца
        }
        if (max < sum)
            max = sum
    }
    println(max)
}
```

Open in Playground → Target: JVM Running on v.1.9.10

> Двумерные массивы указанным в примерах способом можно создать только для класса Array. Оптимизированные для работы с числовыми данными и символами массивы не допускают подобную конструкцию.