**Практическая работа 3**

**Работа с массивами**

[Изучение Kotlin | #5 - Массивы данных - поиск Яндекса по видео (yandex.ru)](https://yandex.ru/video/preview/13453874087491374683)

**Теоретический материал**

Самым простым способом создания нового массива является использование [функции](https://kotlins.org/fun) arrayOf() из стандартной библиотеки Kotlin. Таким образом можно быстро указать какие значения должен хранить массив.

|  |  |
| --- | --- |
| 1  2  3 | fun main() {      val evenNumbers = arrayOf(2, 4, 6, 8)  } |

Поскольку данный массив содержит только [целые числа](https://kotlins.org/number-types), Kotlin воспринимает evenNumbers как массив значений типа Int. Этот тип записывается как Array<Int>. Тип внутри угловых скобок определяет тип значений, которые может хранить массив. Именно тип компилятор будет использовать при добавлении элементов в массив.

Если вы попытаетесь добавить строку в массиве, в котором только числа, то компилятор вернет ошибку, и код не скомпилируется. Такой синтаксис для типа массива является примером аргумента типа или дженерики, о котором вы узнаете подробнее в следующем уроке.

Также, возможно создать массив, у которого все значения будут значениями по умолчанию:

*// int массив, размером 5 со значениями [0, 0, 0, 0, 0]*

**val** arr = IntArray(5)

*// инициализация элементов массива константой*

*// int массив, размером 5 со значениями [42, 42, 42, 42, 42]*

**val** arr = IntArray(5) { 42 }

*// инициализация элементов массива лямбда-выражением*

*// int массив, размером 5 со значениями [0, 1, 2, 3, 4] (элементы инициализированы своим индексом)*

**var** arr = IntArray(5) { it \* 1 }

val people = arrayOf("Tom", "Sam", "Kate", "Bob", "Alice")

Создадим массив и получим значение третьего элемента.

val myArray = arrayOf(1, 2, 3, 4, 5)

println(myArray[2])

Узнать длину массива можно при помощи свойства **size**.

println(myArray.size) // 5

А что случится, если мы добавим в массив строки?

val myArray = arrayOf(1, 2, 3, 4, 5, "зайчик", "вышел", "погулять")

println(myArray[5])

Если мы хотим строгого поведения и не хотим смешивать разные типы, то используем обобщения.

val myArray = arrayOf<Int>(1, 2, 3, 4, 5) // только числа Integer

Существует также синонимы метода, когда уже в имени содержится подсказка: **intArrayOf(),booleanArrayOf(), longArrayOf(), shortArrayOf(), byteArrayOf()**.

Перепишем пример.

val myArray = intArrayOf(1, 2, 3, 4, 5)

Определение двухмерных массивов менее интуитивно понятно и может вызывать сложности. **Например, двухмерный массив чисел:**

|  |  |
| --- | --- |
|  | **val table: Array<Array<Int>> = Array(3, { Array(5, {0}) })** |
|  |  |

**В данном случае двухмерный массив будет иметь три элемента - три строки. Каждая строка будет иметь по пять элементов, каждый из которых равен 0.**

Пройтись по элементам массива и узнать значение индекса можно с помощью метода **withIndex()**:

val numbersArray = intArrayOf(1, 2, 3, 4, 5)

for ((index, value) in numbersArray.withIndex()) {

println("Значение индекса $index равно $value")

}

Свойство indices

У массива есть свойство **indices** и мы можем переписать пример по другому.

val numbers = intArrayOf(1, 2, 3, 4, 5)

for (index in numbers.indices) {

println("Значение индекса $index равно ${numbers[index]}")

}

Свойство возвращает интервал Range, который содержит все индексы массива. Это позволяет не выйти за пределы массива и избежать ошибки **ArrayIndexOutOfBoundsException**.

Но у свойства есть очень интересная особенность. Взгляните на код:

val numbers = intArrayOf(1, 2, 3, 4, 5)

for(index in numbers.indices - 2) {

println(numbers[index])

}

// 1 2 4 5

Из интервала индексов массива мы убрали третий элемент (отсчёт от 0). И теперь при выводе элементов массива мы не увидим числа 3.

Можно сложить два массива.

val numbers = intArrayOf(1, 2, 3)

val numbers3 = intArrayOf(4, 5, 6)

val foo2 = numbers3 + numbers

println(foo2[5]) // 3

Для создания массива с заполненными значениями **null** можно использовать отдельную функцию **arrayOfNulls()**.

Создадим массив с тремя элементами.

val array = arrayOfNulls(3) // [null, null, null]

// равносильно

// arrayOf(null, null, null)

Присвоим значения пустым элементам.

var arr2 = arrayOfNulls<String>(2)

arr2.set(0, "1")

arr2.set(1, "2")

// или

arr2[0] = "1"

arr2[1] = "2"

arr2[1] = readLine().toString.toInt

// получить значения

println(arr2[0]) // или arr2.get(0)

println(arr2[1])

emptyArray()

**Создадим пустой массив и заполним его данными.**

var arr = emptyArray<String>()

arr += "1"

arr += "2"

arr += "3"

arr += "4"

arr += "5"

val vs var

Нужно уяснить разницу между **var** и **val** при работе с массивами.

// Создали новый массив

var myArray = arrayOf(1, 2, 3)

// Это совершенно новый массив

myArray = arrayOf(4, 5)

Фактически мы уничтожили первый массив и создали вместо него второй массив.

Если мы попытаем написать такой же код с использованием **val**, то компилятор запретит такое действие.

// Создали новый массив

val myArray = arrayOf(1, 2, 3)

// Нельзя. Компилятор не пропустит

myArray = arrayOf(4, 5)

Но при этом вы можете менять значения элементов массива, созданного через **val**.

val myArray = arrayOf(1, 2)

myArray[0] = 3 // меняем первый элемент массива

myArray[1] = 4 // меняем второй элемент массива

При использовании конструктора нужно указать размер массива в первом параметре и лямбда-выражение во втором.

val myArray = Array(5, { i -> i \* 2 })

println(myArray[3])

Мы задали пять элементов и каждый элемент в цикле умножаем на 2. В итоге получим массив чисел 0, 2, 4, 6, 8.

**Создадим массив строк от "A" до "Z"**

val letters = Array<String>(26) { i -> ('A' + i).toString() }

println(letters.joinToString(""))

Лямбда-выражение принимает индекс элемента массива и возвращает значение, которое будет помещено в массив с этим индексом. Значение вычисляется путём сложения индекса с кодом символа и преобразованием результата в строку.

Можно опустить тип массива и написать **Array(26)**, компилятор самостоятельно определит нужный тип.

Есть отдельные классы для каждого примитивного типа - **IntArray**, **ByteArray**, **CharArray** и т.д.

val zeros = IntArray(3) // первый способ

val zeros = intArrayOf(0, 0, 0) // второй способ при помощи фабричного метода

println(zeros.joinToString())

Можно использовать лямбда-выражение.

val intArray = IntArray(4){i -> i + i}

println(intArray.joinToString())

**Индекс элемента из списка или массива**

Наиболее удобным способом доступа к элементам массива или списка является использование синтаксис индексирования. Этот синтаксис позволяет получить прямой доступ к любому значению, используя его индекс внутри квадратных скобок:

Java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | fun main() {      val players = mutableListOf("Alice", "Bob", "Cindy", "Dan")      val firstPlayer = players[0]        println("Первый игрок это $firstPlayer")      // Результат: Первый игрок это Alice  } |

Поскольку массивы и списки имеют нулевую индексацию, вы используете индекс 0 для получения значения первого объекта из списка.

Синтаксис индексации эквивалентен вызову метода get() для массива или списка и передачи индекса в качестве аргумента.

|  |  |
| --- | --- |
| 1 | val secondPlayer = players.get(1) |

Вы можете использовать индексы 1, 2, 3 для получения значений следующих элементов массива или списка, но если вы попытаетесь **получить значение индекса который находится за пределами размера массива или списка**, вы получите ошибку и программа прекратит свою работу:

|  |  |
| --- | --- |
| 1 | val player = players[4] // Результат: ошибка IndexOutOfBoundsException |

Такая ошибка возникла, потому что список players содержит только четыре строки. Индекс 4 запрашивает значение пятого элемента из списка, но в списке нет пятого элемента вообще.

Перебор двумерного массива

fun main() {

     val table: Array<Array<Int>> = Array(3, { Array(3, {0}) })

    table[0] = arrayOf(1, 2, 3)

    table[1] = arrayOf(4, 5, 6)

    table[2] = arrayOf(7, 8, 9)

    for(row in table){

         for(cell in row){

            print("$cell \t")

        }

        println()

    }

**Проверка существования элемента в списке**

Вы можете выполнить данную проверку с помощью оператора in который возвращает true, если элемент найден в списке, и false если такого значения в списке нет.

Можно использовать эту стратегию для **создания функции**, которая проверяет, участвует ли определенный игрок в игре или нет:

|  |  |
| --- | --- |
| 1  2  3  5  6  7  8  9 | fun isEliminated(player: String, players: MutableList<String>): Boolean {      return player !in players  }    fun main() {      val players = mutableListOf("Alice", "Bob", "Cindy", "Dan")      println(isEliminated("Bob", players))      // Результат: false  } |

Оператор ! используется, чтобы инвертировать результат из проверки оператором in, т.е. из true превращает в false и обратно из false в true.

Функция спрашивает «**Игрок исключен (isEliminated)?**» затем проверяется если такой игрок находится в списке, если он есть в списке игроков тогда функция возвращает false так как он не исключен из игры.

Оператор in является эквивалентом методу contains(). Вы можете проверить наличие элемента в определенном срезе, используя вместе методы slice() и contains():

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | fun main() {      val players = mutableListOf("Alice", "Bob", "Cindy", "Dan")        if (players.slice(0..3).contains("Alice")) {          println("Игрок найден в срезе!")      } else {          println("В текущем срезе такого игрока нет...")      }  } |

**Задания:**

1. Создать массив из 10 элементов. Вывести на экран значения 5 и 7 элемента
2. Создать массив строк от А до Я. Вывести его на экран
3. Создать массив и вывести на экран его размерность.
4. Сформировать массив, состоящий из дней недели. Ввести массив на экран.
5. Создать массив строк, состоящий из дней недели. Определить является ли “понедельник” элементом массива.
6. Вывести сумму элементов массива. Размерность массива и его содержание определить путем ввода числа с клавиатуры.