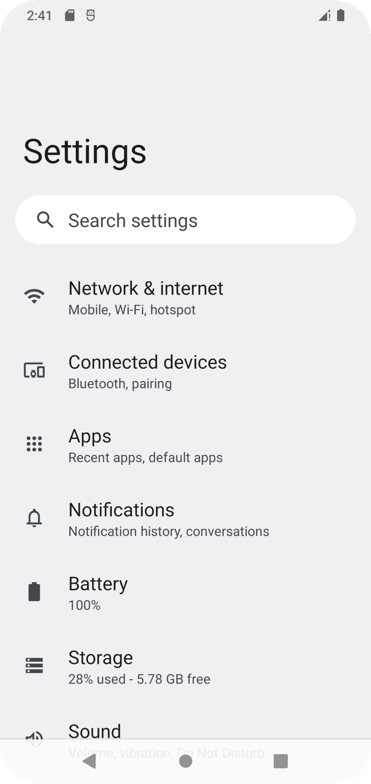
**16. Коллекции**

[1. Введение](https://developer.android.com/codelabs/basic-android-kotlin-compose-collections?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-collections#0)

Во многих приложениях вы наверняка видели данные, отображаемые в виде списка: контакты, настройки, результаты поиска и т. д.



Однако в написанном вами коде вы в основном работали с данными, состоящими из одного значения, например числа или фрагмента текста, отображаемого на экране. Чтобы создавать приложения, использующие произвольные объемы данных, вам необходимо научиться использовать коллекции.

Типы коллекций (иногда называемые структурами данных) позволяют организованно хранить несколько значений, обычно одного и того же типа данных. Коллекция может представлять собой упорядоченный список, группу уникальных значений или сопоставление значений одного типа данных со значениями другого. Возможность эффективного использования коллекций позволяет реализовать общие функции приложений Android, такие как прокрутка списков, а также решать различные реальные задачи программирования, включающие произвольные объемы данных.

В этой лабораторной работе обсуждается, как работать с несколькими значениями в коде, и представлены различные структуры данных, включая массивы, списки, наборы и карты.

Что вы узнаете

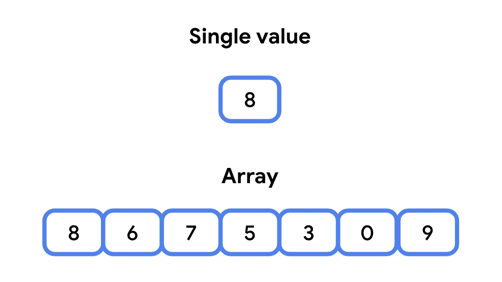
* Как создавать и изменять массивы.
* Как использовать List и MutableList.
* Как использовать Set и MutableSet.
* Как использовать Map и MutableMap.

[2. Массивы в Котлине](https://developer.android.com/codelabs/basic-android-kotlin-compose-collections?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-collections#1)

Что такое массив?

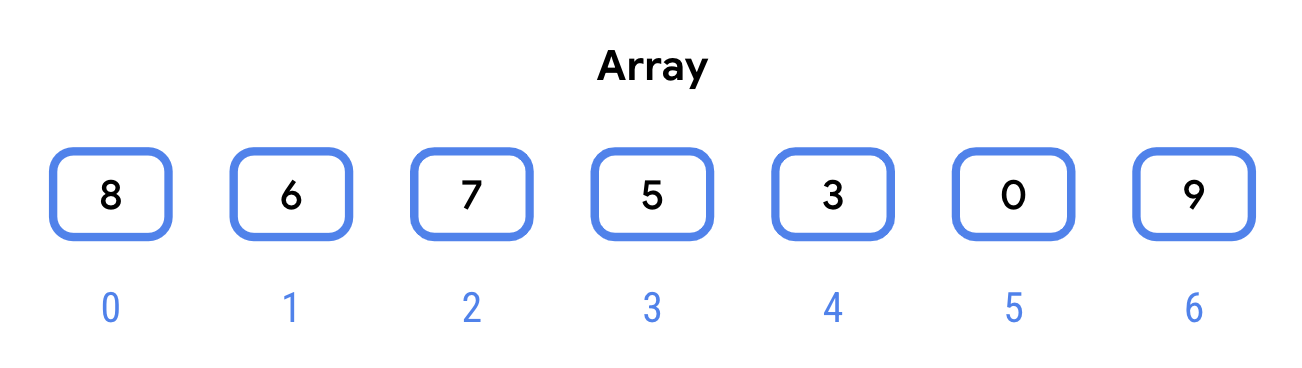
Массив — это самый простой способ сгруппировать произвольное количество значений в ваших программах.

Массив Array представляет собой последовательность значений, имеющих один и тот же тип данных.



* Массив содержит несколько значений, называемых *элементами*.
* Элементы массива упорядочены, и доступ к ним осуществляется по индексу.

Что такое индекс? Индекс — это целое число, соответствующее элементу массива. Индекс сообщает расстояние между элементом и начальным элементом массива. Это называется нулевой индексацией. Первый элемент массива имеет индекс 0, второй элемент имеет индекс 1, поскольку он находится на одну позицию от первого элемента, и так далее.



В памяти устройства элементы массива хранятся рядом друг с другом. Это имеет два важных последствия:

* Доступ к элементу массива по его индексу происходит быстро. Вы можете получить доступ к любому случайному элементу массива по его индексу и ожидать, что доступ к любому другому случайному элементу займет примерно такое же количество времени. Вот почему говорят, что массивы имеют ***произвольный доступ*** .
* Массив имеет фиксированный размер. Это означает, что вы не можете добавлять элементы в массив, превышающие этот размер. Попытка получить доступ к элементу с индексом 100 в массиве из 100 элементов вызовет исключение, поскольку самый высокий индекс равен 99 (помните, что первый индекс равен 0, а не 1). Однако вы можете изменить значения индексов массива.

Чтобы объявить массив в коде, вы используете функцию arrayOf().



Функция arrayOf() принимает элементы массива в качестве параметров и возвращает массив типа, соответствующего переданным параметрам. Это может немного отличаться от других функций, которые вы видели, поскольку arrayOf() имеет различное количество параметров. Если вы передадите два аргумента в arrayOf(), результирующий массив будет содержать два элемента с индексами 0 и 1. Если вы передадите три аргумента, результирующий массив будет содержать 3 элемента с индексами от 0 до 2.

Давайте посмотрим на массивы в действии, проведя небольшое исследование Солнечной системы!

1. Перейдите на [Kotlin Playground](https://developer.android.com/training/kotlinplayground" \t "_blank) .
2. В main() создайте переменную rockPlanets. Вызовем arrayOf(), передавая тип String вместе с четырьмя строками — по одной для каждой каменной планеты Солнечной системы.

val rockPlanets = arrayOf<String>("Mercury", "Venus", "Earth", "Mars")

1. Поскольку Kotlin использует вывод типа, вы можете опустить имя типа при вызове arrayOf(). Под переменной rockPlanets добавьте еще одну переменную gasPlanets, не передавая тип в угловые скобки.

val gasPlanets = arrayOf("Jupiter", "Saturn", "Uranus", "Neptune")

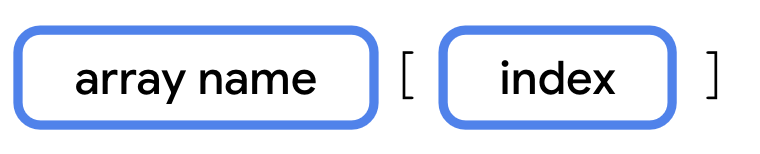
1. С массивами можно делать интересные вещи. Например, как и числовые типы Int или Double, вы можете сложить два массива вместе. Создайте новую переменную с именем solarSystem и присвойте ей значение результата rockPlanets и gasPlanets с помощью оператора плюс (+). Результатом является новый массив, содержащий все элементы массива rockPlanets и элементы массива gasPlanets.

val solarSystem = rockPlanets + gasPlanets

1. Запустите программу, чтобы убедиться, что она работает. Вы пока не должны видеть никаких результатов.

Доступ к элементу массива

Вы можете получить доступ к элементу массива по его индексу.



Это называется синтаксисом индексов. Он состоит из трех частей:

* Имя массива.
* Открывающая ( [) и закрывающая ( ]) квадратные скобки.
* Индекс элемента массива в квадратных скобках.

Давайте получим доступ к элементам массива solarSystem по их индексам.

1. В main() распечатайте каждый элемент массива solarSystem. Обратите внимание, первый индекс 0 и последний индекс 7.

println(solarSystem[0])  
println(solarSystem[1])  
println(solarSystem[2])  
println(solarSystem[3])  
println(solarSystem[4])  
println(solarSystem[5])  
println(solarSystem[6])  
println(solarSystem[7])

1. Запустите вашу программу. Элементы расположены в том же порядке, в котором вы их перечисляли при вызове arrayOf().

Mercury

Venus

Earth

Mars

Jupiter

Saturn

Uranus

Neptune

Вы также можете установить значение элемента массива по его индексу.



Доступ к индексу такой же, как и раньше: имя массива, за которым следуют открывающая и закрывающая квадратные скобки, содержащие индекс. Затем следует оператор присваивания ( =) и новое значение.

Давайте попрактикуемся в изменении значений массива solarSystem.

1. Давайте дадим Марсу новое имя для его будущих поселенцев. Получите доступ к элементу по индексу 3и установите его равным "Little Earth".

solarSystem[3] = "Little Earth"

1. Распечатайте элемент по индексу 3.

println(solarSystem[3])

1. Запустите вашу программу. Обновляется четвертый элемент массива (по индексу ).

...

Little Earth

1. Теперь предположим, что учёные открыли, что за Нептуном существует девятая планета под названием Плутон. Ранее мы упоминали, что вы не можете изменить размер массива. Что бы произошло, если бы вы попытались? Давайте попробуем добавить в массив solarSystem Плутон. Добавьте Плутон по индексу 8, так как это девятый элемент массива.

solarSystem[8] = "Pluto"

1. Запустите свой код. Это вызывает исключение ArrayIndexOutOfBounds. Поскольку в массиве уже было 8 элементов, как и ожидалось, вы не можете просто добавить девятый элемент.

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: **Index 8 out of bounds** for length 8

1. Удалите добавление Плутона в массив.

**Код для удаления**

solarSystem[8] = "Pluto"

1. Если вы хотите сделать массив больше, чем он есть, вам нужно создать новый массив. Определите новую переменную с именем newSolarSystem. Этот массив может хранить девять элементов вместо восьми.

val newSolarSystem = arrayOf("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune", "Pluto")

1. Теперь попробуйте напечатать элемент по индексу 8.

println(newSolarSystem[8])

1. Запустите свой код и убедитесь, что он выполняется без каких-либо исключений.

...

Pluto

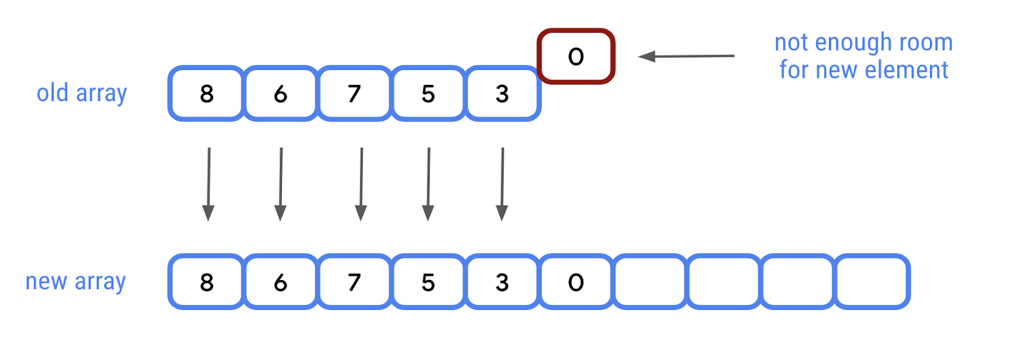
Зная о массивах, вы сможете делать с коллекциями практически все, что угодно.

Хотя массивы являются одним из фундаментальных аспектов программирования, использование массивов для задач, требующих добавления и удаления элементов, уникальности коллекции или сопоставления объектов с другими объектами, не совсем простое и прямое, и код вашего приложения быстро превратится в беспорядок.

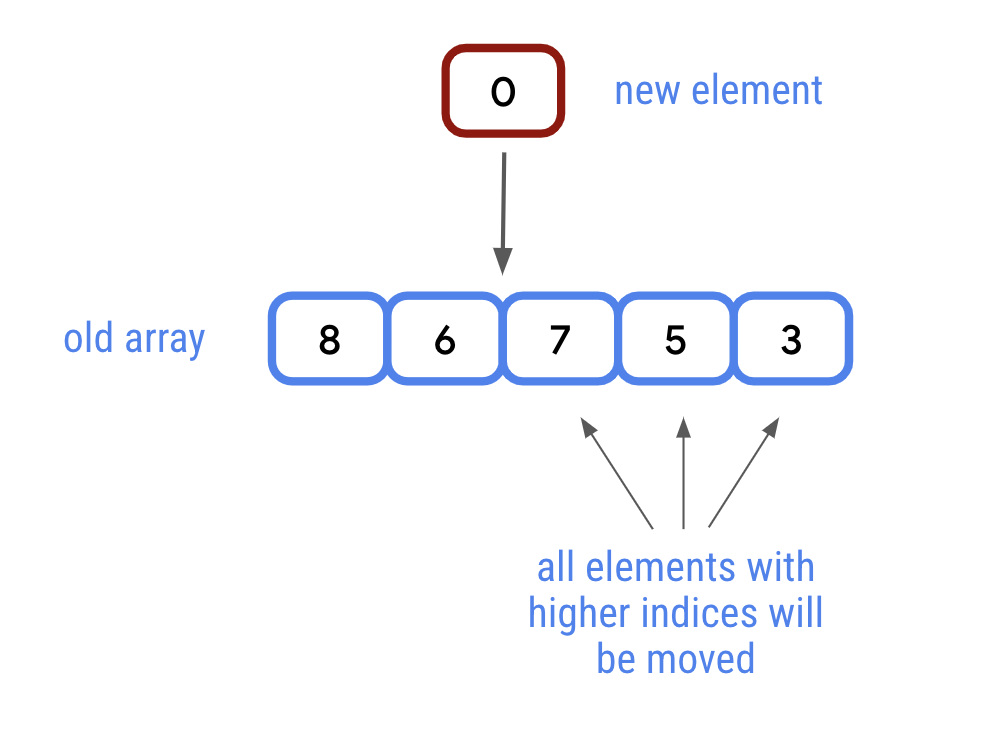
Вот почему большинство языков программирования, включая Kotlin, реализуют специальные типы коллекций для обработки ситуаций, которые часто возникают в реальных приложениях. В следующих разделах вы узнаете о трех распространенных коллекциях: List, Set и Map. Вы также узнаете общие свойства и методы, а также ситуации, в которых можно использовать эти типы коллекций.

## [3. Списки](https://developer.android.com/codelabs/basic-android-kotlin-compose-collections?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-collections#2) [Lists](https://developer.android.com/codelabs/basic-android-kotlin-compose-collections?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-collections#2)

**Список** — это упорядоченная коллекция с изменяемым размером, обычно реализуемая как массив с изменяемым размером. Когда массив заполняется до отказа и вы пытаетесь вставить новый элемент, массив копируется в новый, больший массив.



С помощью списка вы также можете вставлять новые элементы между другими элементами по определенному индексу.



В большинстве случаев добавление любого элемента в список занимает одинаковое количество времени, независимо от количества элементов в списке. Время от времени, если добавление нового элемента приведет к тому, что размер массива превысит определенный размер, элементы массива, возможно, придется переместить, чтобы освободить место для новых элементов. Списки делают все это за вас, но на самом деле это просто массив, который при необходимости заменяется новым массивом.

**List и MutableList**

Типы коллекций, с которыми вы столкнетесь в Kotlin, реализуют один или несколько интерфейсов. Интерфейсы предоставляют стандартный набор свойств и методов для реализации класса. Класс, реализующий интерфейс List, предоставляет реализации для всех свойств и методов интерфейса List. То же самое верно и для MutableList.

В чем отличие  List и MutableList?

* List— это интерфейс, определяющий свойства и методы, относящиеся к упорядоченной коллекции элементов, доступной только для чтения.
* MutableList расширяет интерфейс List, определяя методы изменения списка, например добавление и удаление элементов.

Эти интерфейсы определяют только свойства и методы List или MutableList. Класс, который их расширяет, должен определить, как реализуется каждое свойство и метод. Описанная выше реализация на основе массива — это то, что вы будете использовать чаще всего, но Kotlin позволяет другим классам расширять List и использовать MutableList.

Функция listOf()​

функция listOf()принимает элементы в качестве параметров, но возвращает список вместо массива.

1. Удалите существующий код из main().
2. В main(), создайте List группу планет, называемых solarSystem вызовом listOf().

fun main() {  
    val solarSystem = **listOf**("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")  
}

1. List имеет свойство size для получения количества элементов в списке.

println(solarSystem.size)

1. Запустите свой код. Размер списка должен быть 8.

8

Доступ к элементам из списка

Как и в случае с массивом, вы можете получить доступ к элементу по определенному индексу с помощью индекса. Вы можете сделать то же самое, используя метод get(). Синтаксис индекса и  метод get() принимают Int в качестве параметра и возвращают элемент по этому индексу. Нумерация начинается с 0.

1. Выведите планету по индексу 2, используя синтаксис индексов.

println(solarSystem[2])

1. Распечатайте элемент по индексу 3, вызвав get().

println(solarSystem.get(3))

1. Запустите свой код. Элемент по индексу 2— "Earth"и элемент по индексу 3— "Mars".

...

Earth

Mars

Помимо получения элемента по его индексу, с помощью метода можно также выполнить поиск индекса конкретного элемента indexOf(). Метод indexOf() ищет в списке заданный элемент (переданный в качестве аргумента) и возвращает индекс первого вхождения этого элемента. Если элемент не встречается в списке, он возвращает -1.

1. Выведите результат вызова indexOf()списка solarSystem, передав "Earth".

println(solarSystem.indexOf("Earth"))

1. Вызовите indexOf(), передав "Pluto"и распечатайте результат.

println(solarSystem.indexOf("Pluto"))

1. Запустите свой код. Элемент соответствует"Earth" , поэтому печатается индекс  2. Нет элемента, соответствующего "Pluto", поэтому выводится -1.

...

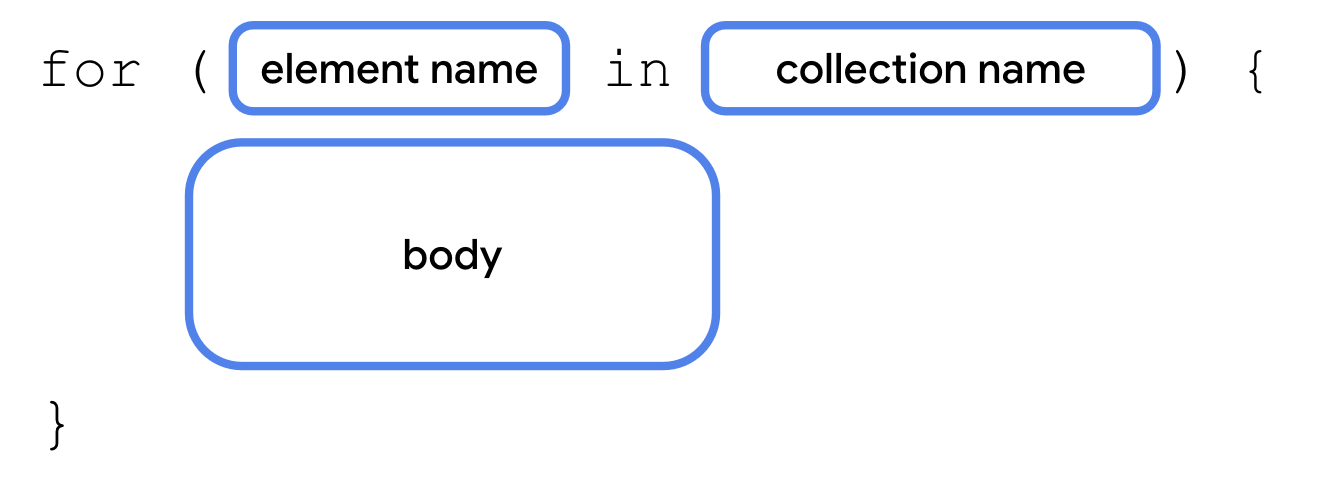
2

-1

Перебирать элементы списка с помощью цикла for

Когда вы узнали о типах функций и лямбда-выражениях, вы увидели, как можно использовать функцию repeat()для многократного выполнения кода.

Распространенной задачей в программировании является выполнение задачи один раз для каждого элемента списка. В Kotlin есть функция, называемая циклом for, позволяющая реализовать это с помощью краткого и читаемого синтаксиса. Вы часто будете видеть, что это называется ***циклом* по списку или *итерацией* по списку**.



Чтобы просмотреть список, используйте ключевое слово for, за которым следует пара открывающих и закрывающих круглых скобок. В скобках укажите имя переменной, затем in ключевое слово и имя коллекции. После закрывающей скобки идет пара открывающих и закрывающих фигурных скобок, в которые вы включаете код, который хотите выполнить для каждого элемента коллекции. Это называется **телом *цикла*** . Каждое выполнение этого кода называется ***итерацией*** .

Переменная перед ключевым словом in не объявлена с помощью val или var — предполагается, что она доступна только для чтения. Вы можете назвать его как угодно. Если списку присвоено имя во множественном числе, например planets, переменной обычно присваивается имя в единственном числе, например planet. Также принято называть переменную item или element.

Она будет использоваться как временная переменная, соответствующая текущему элементу в коллекции — элементу по индексу 0 для первой итерации, элементу по индексу 1 для второй итерации и т. д., и доступ к ней можно получить в фигурных скобках.

Чтобы увидеть это в действии, вы распечатаете название каждой планеты в отдельной строке, используя цикл for.

1. В main(), под самым последним вызовом println(), добавьте цикл for. В круглых скобках назовите переменную planet и пройдите по списку solarSystem.

for (planet in solarSystem) {

}

1. В фигурных скобках выведите значение planet используя println().

for (planet in solarSystem) {

**println(planet)**

}

1. Запустите свой код. Код в теле цикла выполняется для каждого элемента коллекции.

...

Mercury

Venus

Earth

Mars

Jupiter

Saturn

Uranus

Neptune

Добавить элементы в список

Возможность добавлять, удалять и обновлять элементы в коллекции принадлежит только классам, реализующим этот интерфейс MutableList. Если вы отслеживаете недавно открытые планеты, вам, вероятно, понадобится возможность часто добавлять элементы в список. Вам необходимо специально вызвать функцию mutableListOf() вместо listOf(), при создании списка, из которого вы хотите добавлять и удалять элементы.

Существует две версии функции add():

* Первая функция add() имеет единственный параметр типа элемента в списке и добавляет его в конец списка.
* Другая версия add() имеет два параметра. Первый параметр соответствует индексу, по которому должен быть вставлен новый элемент. Второй параметр — это элемент, добавляемый в список.

Давайте посмотрим оба в действии.

1. Измените инициализацию solarSystem. Теперь вы можете вызывать методы, определенные в MutableList.

val solarSystem = **mutableListOf**("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")

1. Опять же, мы могли бы классифицировать Плутон как планету. Вызовите метод add() списка solarSystem, передав в качестве единственного аргумента "Pluto".

solarSystem.add("Pluto")

1. Некоторые ученые предполагают, что планета под названием Тейя существовала до того, как столкнулась с Землей и образовала Луну. Вставить "Theia" по индексу  3 между "Earth" и "Mars".

solarSystem.add(3, "Theia")

Обновить элементы по определенному индексу

Вы можете обновить существующие элементы с помощью синтаксиса индексов:

1. Обновите значение по индексу 3 до "Future Moon".

solarSystem[3] = "Future Moon"

1. Выведите значение по индексам 3, 9 используя синтаксис индексов.

println(solarSystem[3])

println(solarSystem[9])

1. Запустите свой код, чтобы проверить результат.

Future Moon

Pluto

Удалить элементы из списка

Элементы удаляются с помощью метода remove() или removeAt(). Вы можете удалить элемент, передав его в метод remove(), или по его индексу, используя removeAt().

Давайте посмотрим оба метода удаления элемента в действии.

1. Вызовите removeAt() списка solarSystem передав индекс 9. Этот оператор удалит "Pluto" из списка.

solarSystem.removeAt(9)

1. Вызовите remove() списка  solarSystem, передав "Future Moon" в качестве удаляемого элемента. Должен выполниться поиск в списке, и если соответствующий элемент будет найден, он будет удален.

solarSystem.remove("Future Moon")

1. List предоставляет метод contains(), который возвращает Boolean,  если элемент существует в списке. Распечатайте результат contains() вызова "Pluto".

println(solarSystem.contains("Pluto"))

1. Еще более краткий синтаксис — использовать  оператор in. Вы можете проверить, находится ли элемент в списке, используя элемент in. Используйте оператор in, чтобы проверить, solarSystem содержит ли "Future Moon".

println("Future Moon" in solarSystem)

1. Запустите свой код. Оба утверждения должны быть напечатаны false.

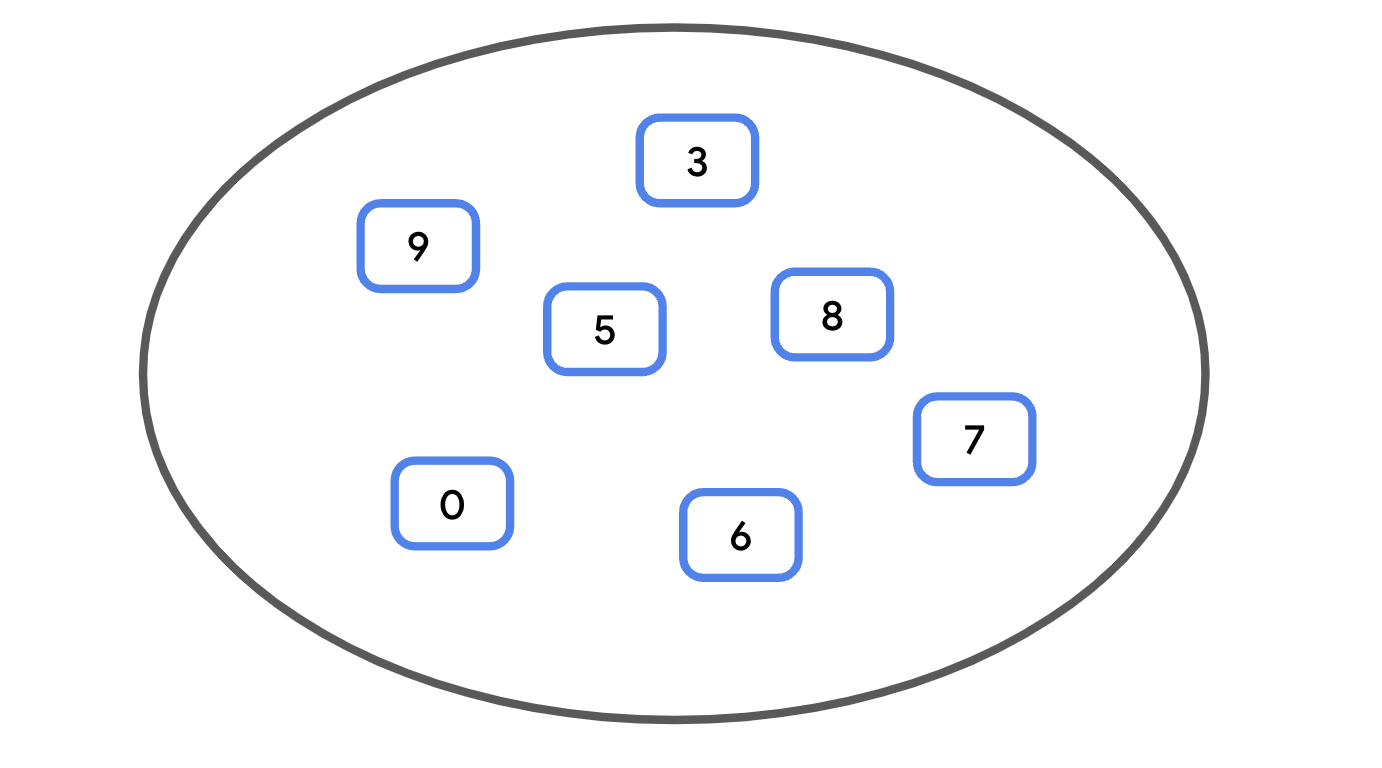
...

false

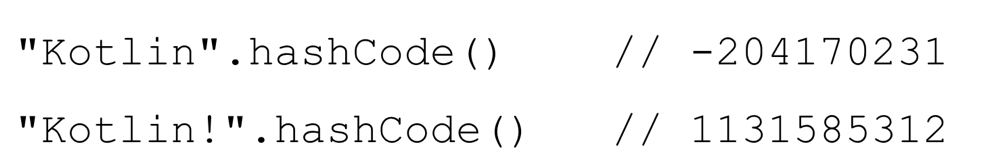
false

## [4. Sets](https://developer.android.com/codelabs/basic-android-kotlin-compose-collections?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-collections#3) – множества

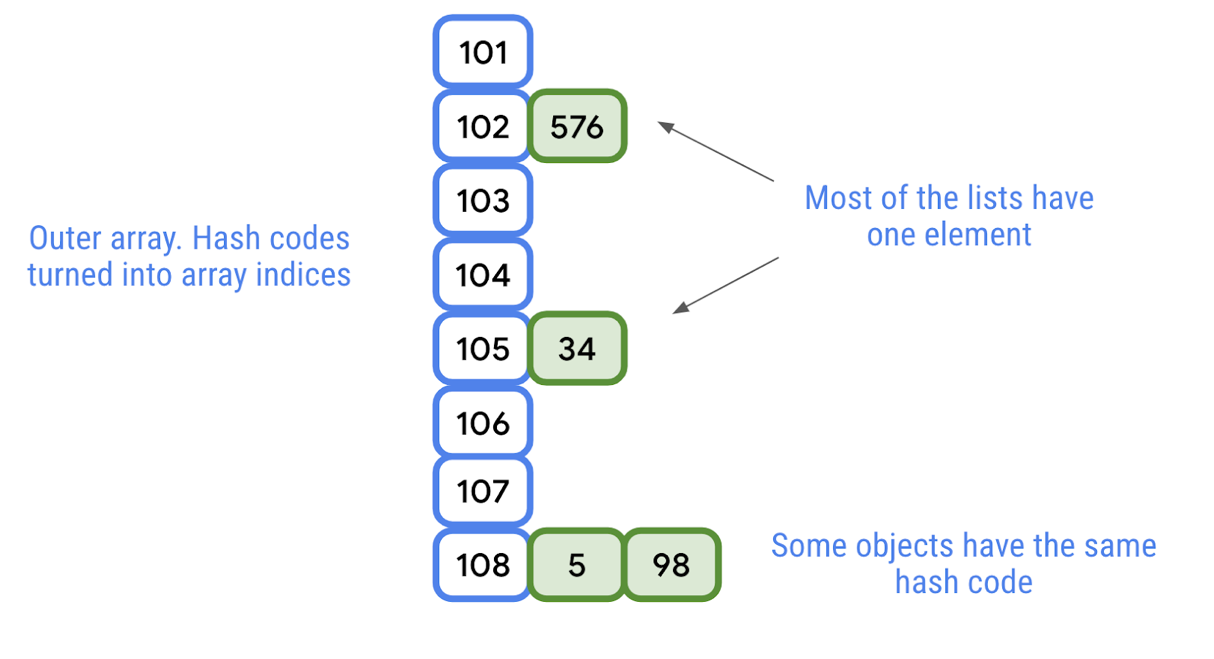
**Множества** — это коллекция, которая не имеет определенного порядка и не допускает дублирования значений.



Как возможна такая коллекция? Секрет — это ***хэш-код*** . Хэш-код создается Int методом hashCode() любого класса Kotlin. Его можно рассматривать как полууникальный идентификатор объекта Kotlin. Небольшое изменение объекта, например добавление одного символа в String, приводит к совершенно другому значению хеш-функции. Хотя два объекта могут иметь один и тот же хеш-код (это называется хеш-коллизией), функция hashCode() обеспечивает некоторую степень уникальности, когда в большинстве случаев каждое из двух разных значений имеет уникальный хеш-код.



**Примечание.** Набор использует хэш-коды в качестве индексов массива. Конечно, различных хэш-кодов может быть около 4 миллиардов, поэтому это множество не просто один гигантский массив. Вместо этого вы можете думать о множестве массива списков.



Каждый внешний массив — числа, обведенные синим цветом слева — соответствует диапазону (также известному как сегмент) возможных хеш-кодов. Каждый внутренний список (затененный зеленым справа) представляет отдельные элементы множества. Поскольку коллизии хешей относительно редки, даже если потенциальные индексы ограничены, внутренние списки в каждом индексе массива будут содержать только один или два элемента каждый, если только не будут добавлены десятки или сотни тысяч элементов.

У множеств есть два важных свойства:

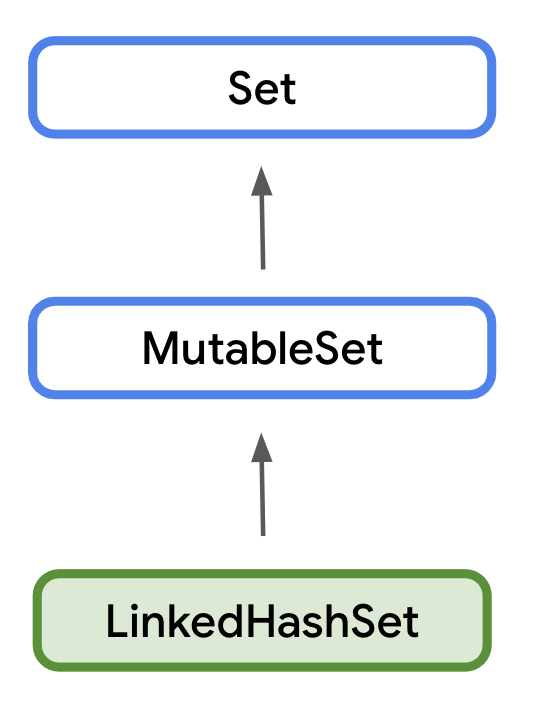
1. Поиск определенного элемента в наборе выполняется быстрее по сравнению со списками, особенно для больших коллекций. Хотя метод indexOf() у List требует проверки каждого элемента с самого начала, пока не будет найдено совпадение, в среднем проверка того, находится ли элемент в наборе, будь то первый элемент или стотысячный, занимает в среднем одинаковое количество времени.
2. Наборы, как правило, используют больше памяти, чем списки, для того же объема данных, поскольку часто требуется больше индексов массива, чем данных в наборе.

**Примечание.** Вопреки распространенному мнению, время, необходимое для проверки наличия элемента в наборе, не фиксировано и фактически зависит от объема данных в наборе. Однако, поскольку коллизий хэшей обычно бывает немного, количество элементов, которые необходимо проверить, все равно на порядки меньше, чем при поиске элемента в списке.

Преимущество наборов заключается в обеспечении уникальности. Если вы пишете программу для отслеживания вновь открытых планет, набор предоставляет простой способ проверить, была ли уже открыта планета. При больших объемах данных это часто предпочтительнее, чем проверка наличия элемента в списке, требующая перебора всех элементов.

Например, List и MutableList, есть и Set и MutableSet.

MutableSet реализации Set, поэтому любая реализация класса MutableSet должна реализовывать и то, и другое.



Используйте MutableSet в Котлине

В этом примере мы будем использовать MutableSet, чтобы продемонстрировать, как добавлять и удалять элементы.

1. Удалите существующий код из main().
2. Создайте  группу планет Set под названием solarSystem используя mutableSetOf(). Это возвращает MutableSet, реализация которого по умолчанию — LinkedHashSet().

val solarSystem = **mutableSetOf**("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")

1. Распечатайте размер набора, используя свойство size.

println(solarSystem.size)

1. Метод add(). Добавьте "Pluto"в  набор solarSystem, используя метод add(). Для добавляемого элемента требуется только один параметр. Элементы в наборах не обязательно имеют порядок, поэтому индекса нет!

solarSystem.add("Pluto")

1. Распечатайте размер size набора после добавления элемента.

println(solarSystem.size)

1. Функция contains() принимает один параметр и проверяет, содержится ли указанный элемент в наборе. Если это так, он возвращает true. В противном случае он возвращает ложь. Вызовите contains(), чтобы узнать, "Pluto" есть ли в solarSystem.

println(solarSystem.contains("Pluto"))

1. Запустите свой код. Размер увеличился и contains()теперь возвращается true.

8

9

true

**Примечание** . В качестве альтернативы вы можете использовать оператор in, чтобы проверить, находится ли элемент в коллекции, например: "Pluto" in solarSystem эквивалентно solarSystem.contains("Pluto").

1. Как упоминалось ранее, наборы не могут содержать дубликаты. Попробуйте добавить "Pluto"еще раз.

solarSystem.add("Pluto")

1. Распечатайте размер набора еще раз.

println(solarSystem.size)

1. Запустите свой код еще раз. "Pluto"не добавляется, так как уже есть в наборе. На этот раз размер не должен увеличиваться.

...

9

Функция remove() принимает один параметр и удаляет указанный элемент из набора.

1. Используйте функцию remove()для удаления "Pluto".

solarSystem.remove("Pluto")

**Примечание.** Помните, что наборы — это неупорядоченная коллекция. Невозможно удалить значение из набора по его индексу, поскольку наборы не имеют индексов.

1. Распечатайте размер коллекции и вызовите contains() еще раз, чтобы проверить, "Pluto" есть ли еще в наборе.

println(solarSystem.size)

println(solarSystem.contains("Pluto"))

1. Запустите свой код. "Pluto" больше нет в наборе и размер теперь 8.

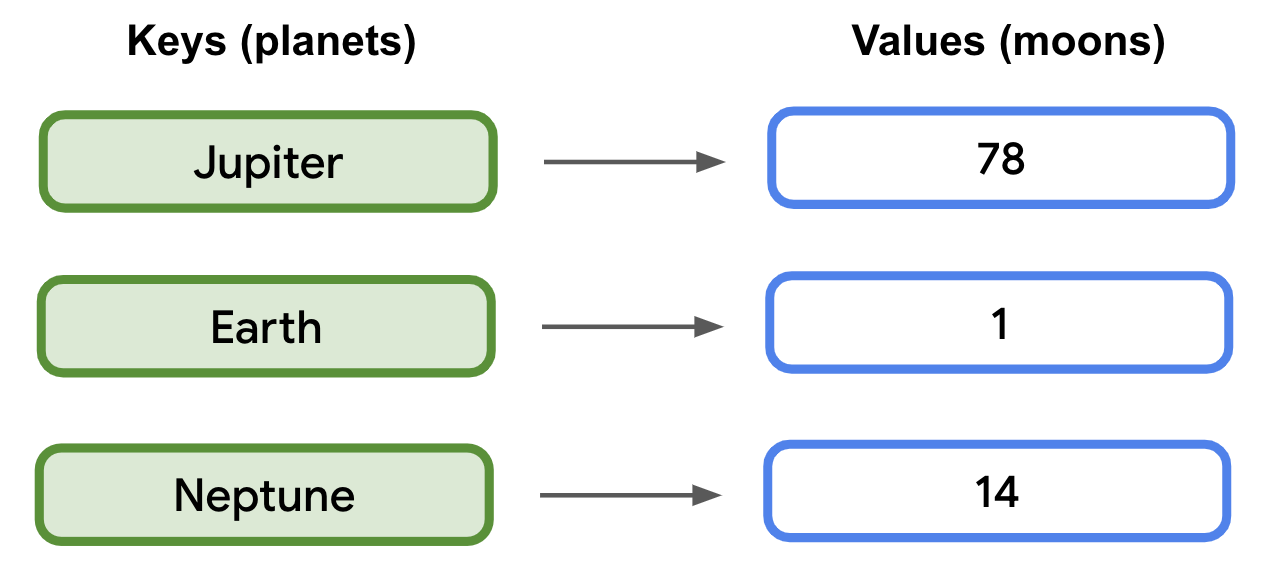
...

8

false

## [5. [Map collection](https://developer.android.com/codelabs/basic-android-kotlin-compose-collections?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-collections#4) - Коллекция карт](https://developer.android.com/codelabs/basic-android-kotlin-compose-collections?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-collections#4)

Map— это коллекция, состоящая из ключей и значений. Это называется картой, потому что уникальные ключи *сопоставляются* с другими значениями. Ключ и сопровождающее его значение часто называются файлом key-value pair.



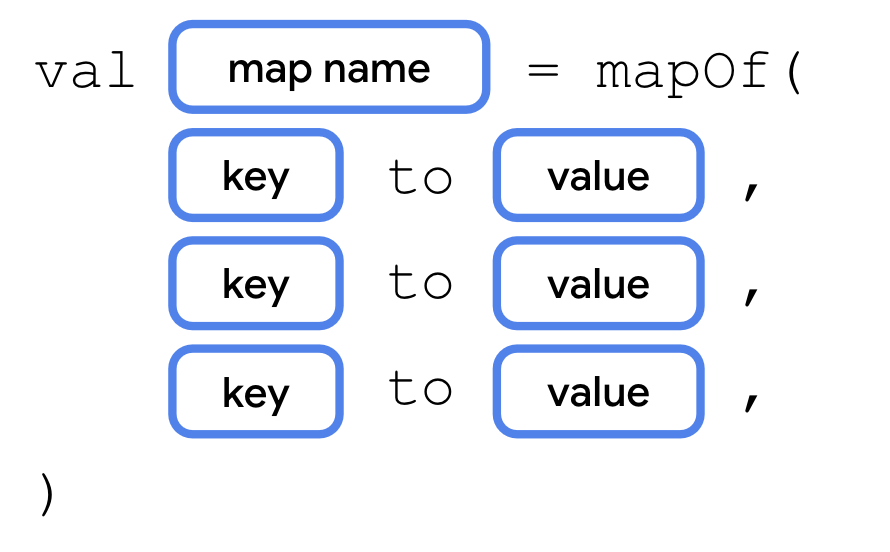
Ключи карты уникальны. Однако значения карты таковыми не являются. Два разных ключа могут соответствовать одному и тому же значению. Например, "Mercury"имеет значение 0 и "Venus" имеет значение 0 .

Доступ к значению на карте по его ключу обычно происходит быстрее, чем поиск по большому списку, например, с помощью indexOf().

Карты можно объявить с помощью функции mapOf() или mutableMapOf(). Карты требуют двух общих типов, разделенных запятой: один для ключей, а другой для значений.



Карта также может использовать выведение типа, если у нее есть начальные значения. Чтобы заполнить карту начальными значениями, каждая пара ключ-значение состоит из ключа, за которым следует оператор to, за которым следует значение. Каждая пара разделяется запятой.



Давайте подробнее рассмотрим, как использовать карты, а также некоторые полезные свойства и методы.

1. Удалите существующий код из main().
2. Создайте карту с именем solarSystem используя mutableMapOf()с начальными значениями, как показано.

val solarSystem = mutableMapOf(  
    "Mercury" to 0,  
    "Venus" to 0,  
    "Earth" to 1,  
    "Mars" to 2,  
    "Jupiter" to 79,  
    "Saturn" to 82,  
    "Uranus" to 27,  
    "Neptune" to 14  
)

1. Подобно спискам и наборам, Map предоставляет свойство size, содержащее количество пар ключ-значение. Распечатайте размер карты solarSystem.

println(solarSystem.size)

1. Вы можете использовать синтаксис индексов для установки дополнительных пар ключ-значение. Установите для ключа "Pluto" значение 5.

solarSystem["Pluto"] = 5

1. Распечатайте размер еще раз после вставки элемента.

println(solarSystem.size)

1. Вы можете использовать синтаксис индексов для получения значения. Выведите количество значений для ключа "Pluto".

println(solarSystem["Pluto"])

1. Вы также можете получить доступ к значениям с помощью метода get(). Независимо от того, используете ли вы синтаксис индексов или вызов get(), возможно, что передаваемый вами ключ отсутствует на карте. Если пары ключ-значение нет, она вернет ноль. Выведите значение для "Theia".

println(solarSystem.get("Theia"))

1. Запустите свой код. Должно быть напечатано количество спутников Плутона. Однако, поскольку Тейи нет на карте, вызов get() возвращает ноль.

8

9

5

null

Метод remove() удаляет пару ключ-значение с указанным ключом. Он также возвращает удаленное значение или null, если указанного ключа нет в карте.

1. Распечатайте результат вызова remove() и передачи "Pluto".

solarSystem.remove("Pluto")

1. Чтобы убедиться, что элемент был удален, распечатайте размер еще раз.

println(solarSystem.size)

1. Запустите свой код. Размер карты после удаления записи равен 8.

...

8

1. Синтаксис индекса или метод put()также может изменять значение уже существующего ключа. Используйте синтаксис индексов, чтобы обновить количество спутников Юпитера до 78 и напечатать новое значение.

solarSystem["Jupiter"] = 78

println(solarSystem["Jupiter"])

1. Запустите свой код. Значение существующего ключа "Jupiter" обновляется.

...

78

[6. Заключение](https://developer.android.com/codelabs/basic-android-kotlin-compose-collections?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-collections#5)

Вы узнали об одном из самых фундаментальных типов данных в программировании — массиве, а также о нескольких удобных типах коллекций, построенных на основе массивов, включая List, Set и Map. Эти типы коллекций позволяют группировать и упорядочивать значения в коде. Массивы и списки обеспечивают быстрый доступ к элементам по их индексу, а наборы и карты используют хэш-коды, чтобы упростить поиск элементов в коллекции. Вы увидите, что эти типы коллекций часто будут использоваться в будущих приложениях, и знание того, как их использовать, принесет вам пользу в вашей будущей карьере программиста.

Краткое содержание

* Массивы хранят упорядоченные данные одного типа и имеют фиксированный размер.
* Массивы используются для реализации многих других типов коллекций.
* Списки представляют собой упорядоченную коллекцию изменяемого размера.
* Наборы представляют собой неупорядоченные коллекции и не могут содержать дубликаты.
* Карты работают аналогично наборам и хранят пары ключей и значений указанного типа.

Дополнительно:

<https://developer.android.com/codelabs/basic-android-kotlin-compose-collections?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-basics-compose-unit-3-pathway-1%23codelab-https%3A%2F%2Fdeveloper.android.com%2Fcodelabs%2Fbasic-android-kotlin-compose-collections#6>