Лекция. Функции высшего порядка с коллекциями

Введение

В практичекой работе «Использование типов функций и лямбда-выражений в Kotlin» вы узнали о ф**ункциях высшего порядка** - функциях, которые принимают другие функции в качестве параметров и/или возвращают функцию, например `repeat()`. Функции высшего порядка особенно важны для коллекций, поскольку они помогают выполнять такие распространенные задачи, как сортировка или фильтрация, с меньшим количеством кода. Теперь, когда у вас есть прочная основа работы с коллекциями, пришло время вернуться к функциям высшего порядка.

В этом уроке вы узнаете о различных функциях, которые можно использовать для типов коллекций, включая `forEach()`, `map()`, `filter()`, `groupBy()`, `fold()` и `sortedBy()`. В процессе вы получите дополнительную практику работы с лямбда-выражениями.

Необходимые условия

- Знакомство с типами функций и лямбда-выражениями.
- Знакомство с синтаксисом лямбда-выражений, например, с функцией repeat().
- Знание различных типов коллекций в Kotlin, таких как List.

Что вы узнаете

- Как встраивать лямбда-выражения в строки.
- Как использовать различные функции высшего порядка с коллекцией List, включая forEach(), map(), filter(), groupBy(), fold() и sortedBy().

forEach() и шаблоны строк с лямбдами

Стартовый код В следующих примерах вы возьмете List, представляющий меню печенья в пекарне, и с помощью функций высшего порядка отформатируете это меню разными способами.

Начните с подготовки начального кода.

Над функцией main() добавьте класс Cookie. Каждый экземпляр Cookie представляет собой пункт меню с названием, ценой и другой информацией о нем.

```
class Cookie(
    val name: String,
    val softBaked: Boolean,
    val hasFilling: Boolean,
    val price: Double
)

fun main() {

}
```

В классе Cookie, вне main(), создайте список cookie, как показано на рисунке. Тип предполагается как `List<Cookie>`.

```
class Cookie(
    val name: String,
    val softBaked: Boolean,
    val hasFilling: Boolean,
    val price: Double
)

val cookies = listOf(
    Cookie(
        name = "Chocolate Chip",
        softBaked = false,
        hasFilling = false,
        price = 1.69
    ),
    Cookie(
        name = "Banana Walnut",
        softBaked = true,
        hasFilling = false,
        price = 1.49
    ),
    Cookie(
        name = "Vanilla Creme",
        softBaked = false,
        hasFilling = true,
        price = 1.59
    ),
    Cookie(
        name = "Chocolate Peanut Butter",
        softBaked = false,
        hasFilling = true,
        price = 1.49
    ),
    Cookie(
        name = "Snickerdoodle",
        softBaked = true,
        hasFilling = false,
        price = 1.39
    ),
    Cookie(
        name = "Blueberry Tart",
        softBaked = true,
        hasFilling = true,
        price = 1.79
    ),
    Cookie(
        name = "Sugar and Sprinkles",
        softBaked = false,
        hasFilling = false,
        price = 1.39
```

```
    )
)

fun main() {

}
```

Перебор списка с помощью forEach()

Первая функция высшего порядка, о которой вы узнаете, - это функция `forEach()`. Функция forEach() выполняет переданную в качестве параметра функцию один раз для каждого элемента коллекции. Это работает аналогично функции repeat() или циклу for. Лямбда выполняется для первого элемента, затем для второго и так далее, пока не будет выполнена для каждого элемента коллекции. Подпись метода выглядит следующим образом:

```
forEach(action: (T) -> Unit)
```

forEach() принимает единственный параметр действия - функцию типа `(T) -> Unit`.

`T` соответствует любому типу данных, который содержит коллекция. Поскольку лямбда принимает единственный параметр, вы можете опустить имя и ссылаться на параметр с его помощью.

Используйте функцию forEach() для печати элементов в списке cookies.

- В main() вызовите функцию forEach() для списка cookies, используя синтаксис лямбды с запятой. Поскольку скользящая лямбда является единственным аргументом, при вызове функции можно опустить круглые скобки.

```
fun main() {
    cookies.forEach {

    }
}
```

- В тело лямбды добавьте оператор println(), который выведет его на печать.

```
fun main() {
    cookies.forEach {
        println("Menu item: $it")
    }
}
```

- Запустите свой код и понаблюдайте за выводом. Все, что выводится, - это имя типа (Cookie) и уникальный идентификатор объекта, но не его содержимое.

```
Menu item: Cookie@5a10411
Menu item: Cookie@68de145
Menu item: Cookie@27fa135a
Menu item: Cookie@46f7f36a
Menu item: Cookie@421faab1
Menu item: Cookie@2b71fc7e
Menu item: Cookie@5ce65a89
```

Вставка выражений в строки

Когда вы только знакомились с шаблонами строк, вы видели, как символ доллара (`$`) можно использовать с именем переменной, чтобы вставить ее в строку. Однако в сочетании с оператором точки (.) для доступа к свойствам это работает не так, как ожидалось.

- В вызове forEach() измените тело лямбды, чтобы вставить `$it.name` в строку.

```
cookies.forEach {
    println("Menu item: $it.name")
}
```

- Запустите свой код. Обратите внимание, что здесь вставлено имя класса, Cookie, и уникальный идентификатор объекта, за которым следует `.name`. К значению свойства `name` доступа нет.

```
Menu item: Cookie@5a10411.name
Menu item: Cookie@68de145.name
Menu item: Cookie@27fa135a.name
Menu item: Cookie@46f7f36a.name
Menu item: Cookie@421faab1.name
Menu item: Cookie@2b71fc7e.name
Menu item: Cookie@5ce65a89.name
```

Чтобы получить доступ к свойствам и вставить их в строку, вам нужно выражение. Вы можете сделать выражение частью шаблона строки, окружив его фигурными скобками.

Лямбда-выражение помещается между открывающей и закрывающей фигурными скобками. Вы можете обращаться к свойствам, выполнять математические операции, вызывать функции и т. д., а возвращаемое значение лямбды вставляется в строку.

Давайте изменим код так, чтобы в строку вставлялось имя.

- Окружим it.name фигурными скобками, чтобы сделать его лямбда-выражением.

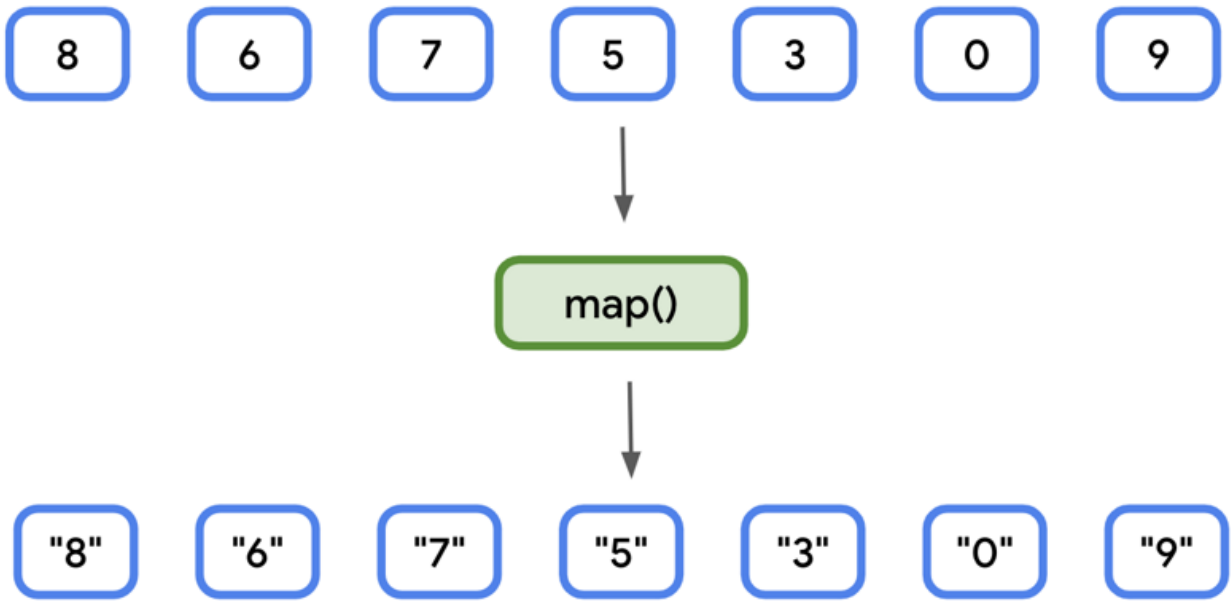
```
cookies.forEach {
    println("Menu item: ${it.name}")
}
```

- Запустите ваш код. В выводе будет указано имя каждого Cookie.

```
Menu item: Chocolate Chip
Menu item: Banana Walnut
Menu item: Vanilla Creme
Menu item: Chocolate Peanut Butter
Menu item: Snickerdoodle
Menu item: Blueberry Tart
Menu item: Sugar and Sprinkles
```

map()

Функция map() позволяет преобразовать коллекцию в новую коллекцию с тем же количеством элементов. Например, map() может преобразовать `List<Cookie>` в `List<String>`, содержащий только имя cookie, если вы укажете функции map(), как создать String из каждого элемента Cookie.



Допустим, вы пишете приложение, которое отображает интерактивное меню для пекарни. Когда пользователь переходит к экрану, на котором отображается меню печенья, он может захотеть увидеть данные, представленные в логической форме, например, название, за которым следует цена. Вы можете создать список строк, отформатированных с соответствующими данными (название и цена), с помощью функции map().

- Удалите весь предыдущий код из функции main(). Создайте новую переменную fullMenu и установите ее равной результату вызова функции map() для списка cookies.

```
val fullMenu = cookies.map {

}
```

- В теле лямбды добавьте строку, отформатированную таким образом, чтобы в ней были указаны название и цена.

```
val fullMenu = cookies.map {
    "${it.name} - $${it.price}"
}
```

> Примечание: Перед выражением используется второй $. Первый воспринимается как символ знака доллара ($), поскольку за ним не следует имя переменной или лямбда-выражение.

- Выведите содержимое FullMenu. Это можно сделать с помощью функции `forEach()`. Коллекция fullMenu, возвращаемая из map(), имеет тип `List<String>`, а не `List<Cookie>`. Каждому Cookie в cookies соответствует строка в fullMenu.

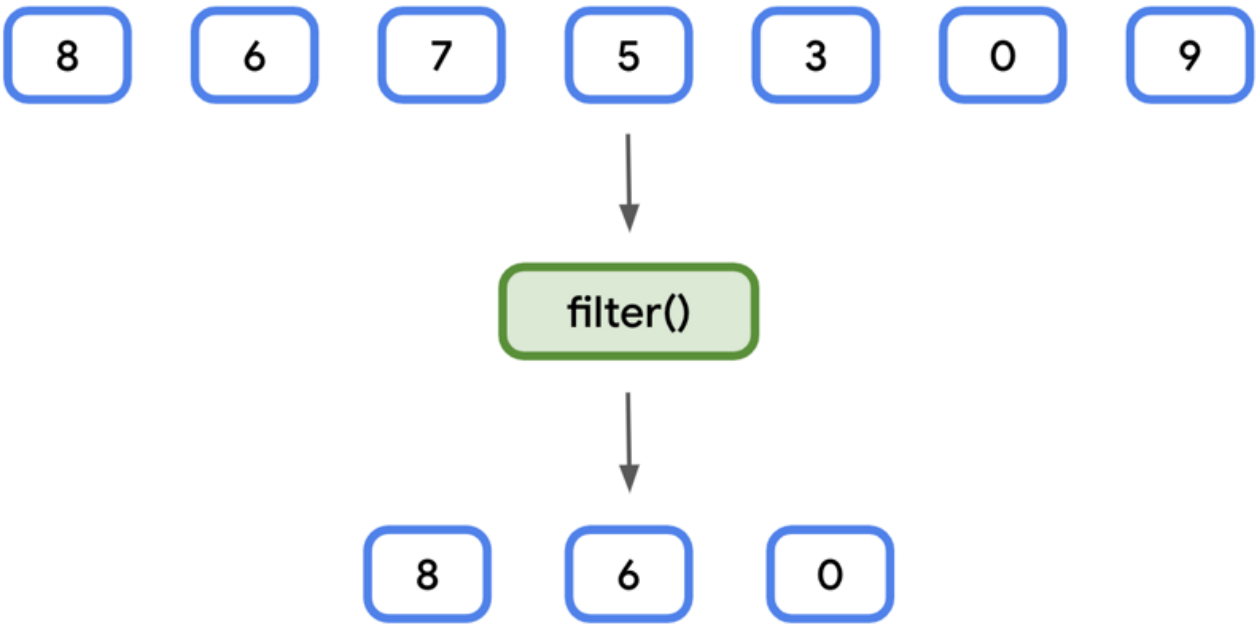
```
println("Full menu:")
fullMenu.forEach {
    println(it)
}
```

- Запустите свой код. Вывод совпадает с содержимым списка FullMenu.

```
Full menu:
Chocolate Chip - $1.69
Banana Walnut - $1.49
Vanilla Creme - $1.59
Chocolate Peanut Butter - $1.49
Snickerdoodle - $1.39
Blueberry Tart - $1.79
Sugar and Sprinkles - $1.39
```

filter()

Функция filter() позволяет создать подмножество коллекции. Например, если у вас есть список чисел, вы можете использовать filter(), чтобы создать новый список, содержащий только числа, кратные 2.



Если результат функции map() всегда дает коллекцию одинакового размера, то filter() дает коллекцию того же размера или меньше, чем исходная коллекция. В отличие от map(), результирующая коллекция также имеет тот же тип данных, поэтому фильтрация `List<Cookie>` приведет к созданию другого `List<Cookie>`.

Как и map() и forEach(), filter() принимает в качестве параметра одно лямбда-выражение. Лямбда имеет один параметр, представляющий каждый элемент коллекции, и возвращает булево значение.

Для каждого элемента коллекции:

- Если результат лямбда-выражения равен true, то элемент включается в новую коллекцию.
- Если результат равен false, то элемент не будет включен в новую коллекцию.

Это полезно, если вы хотите получить подмножество данных в вашем приложении. Например, допустим, пекарня хочет выделить свое сдобное печенье в отдельный раздел меню. Вы можете сначала filter() список печенья, а затем вывести его элементы.

В main() создайте новую переменную softBakedMenu и установите ее в результат вызова функции filter() для списка печенья.

```
val softBakedMenu = cookies.filter {
}
```

В теле лямбды добавьте булево выражение, чтобы проверить, равно ли свойство softBaked печенья true. Поскольку softBaked - это булево выражение, тело лямбды должно содержать только it.softBaked.

```
val softBakedMenu = cookies.filter {
    it.softBaked
}
```

- Выведите содержимое меню softBakedMenu с помощью функции forEach().

```
println("Soft cookies:")
softBakedMenu.forEach {
    println("${it.name} - $${it.price}")
}
```

- Запустите свой код. Меню печатается, как и раньше, но включает только печенье с мягкой выпечкой.

```
...
Soft cookies:
Banana Walnut - $1.49
Snickerdoodle - $1.39
Blueberry Tart - $1.79
```

groupBy()

Функция `groupBy()` может использоваться для преобразования списка в карту на основе функции. Каждое уникальное возвращаемое значение функции становится ключом в результирующей карте. Значениями для каждого ключа являются все элементы коллекции, которые дали это уникальное возвращаемое значение.

Тип данных ключей совпадает с типом возвращаемой функции, переданной в groupBy(). Тип данных значений - это список элементов из исходного списка.

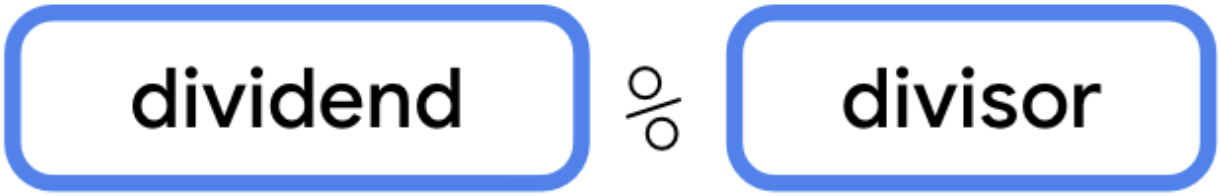
> Примечание: значение не обязательно должно быть того же типа, что и список. Существует другая версия groupBy(), которая может преобразовывать значения в другой тип. Однако эта версия здесь не рассматривается.

Это может быть трудно понять, поэтому давайте начнем с простого примера.

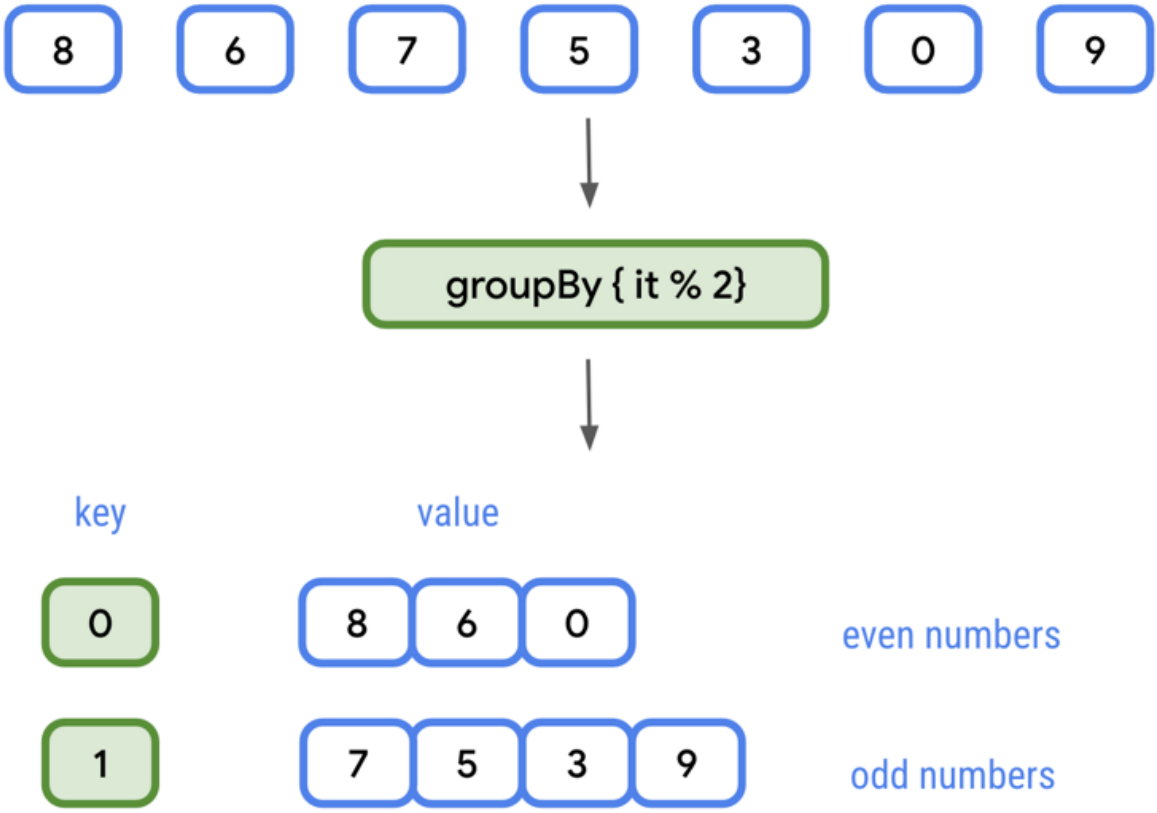
- Дайте тот же список чисел, что и раньше, и сгруппируйте их как четные или нечетные.

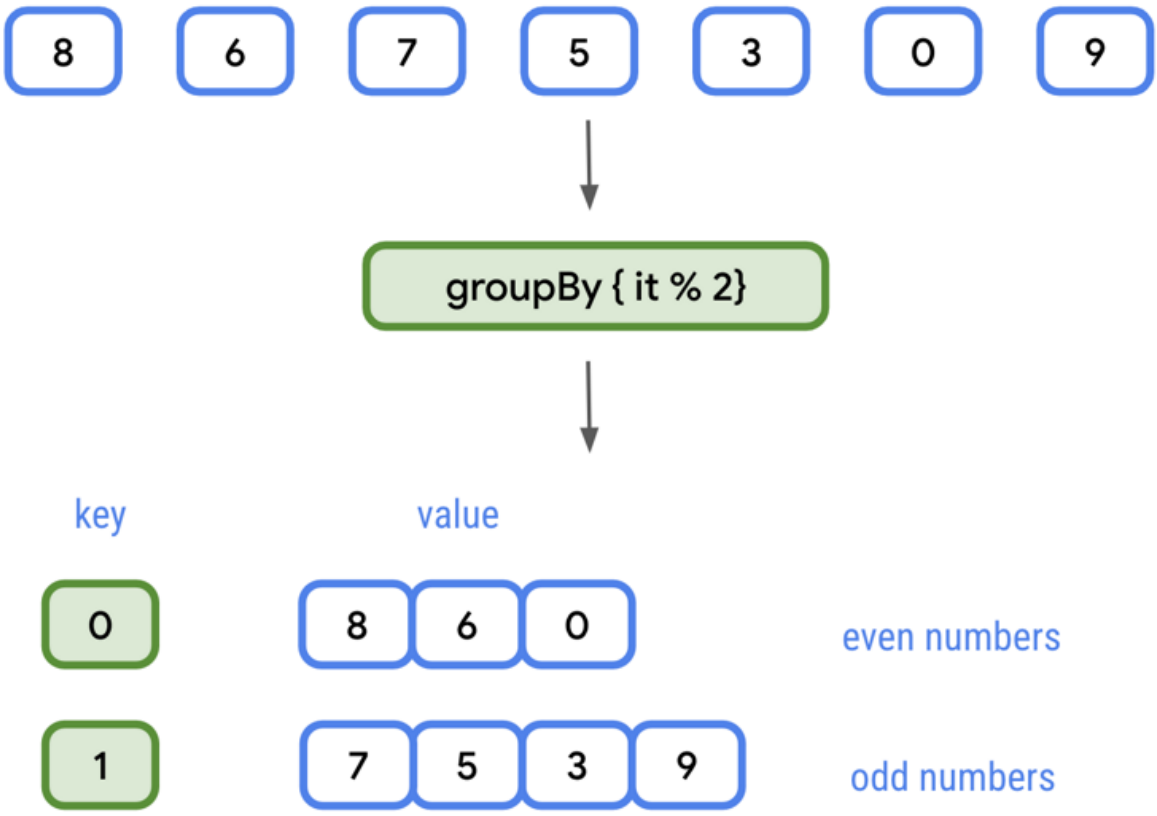
Вы можете проверить, является ли число четным или нечетным, разделив его на 2 и проверив, равен ли остаток 0 или 1. Если остаток равен 0, то число четное. В противном случае, если остаток равен 1, число нечетное.

Этого можно добиться с помощью оператора `modulo` (%). Оператор `modulo` делит делитель в левой части выражения на делитель в правой.



Вместо того чтобы возвращать результат деления, как оператор деления (/), оператор modulo возвращает остаток. Это делает его полезным для проверки того, является ли число четным или нечетным.





Функция groupBy() вызывается со следующим лямбда-выражением: { it % 2 }.

Полученная карта имеет два ключа: 0 и 1. Каждый ключ имеет значение типа `List<Int>`. Список для ключа 0 содержит все четные числа, а список для ключа 1 - все нечетные числа.

Реальным примером может быть приложение для фотографий, которое группирует снимки по объекту или месту, где они были сделаны. Для нашего меню пекарни давайте сгруппируем меню по тому, является ли печенье мягким или нет.

Используйте groupBy(), чтобы сгруппировать меню на основе свойства softBaked.

- Удалите вызов filter() из предыдущего шага.

Код для удаления

```
val softBakedMenu = cookies.filter {
    it.softBaked
}
println("Soft cookies:")
softBakedMenu.forEach {
    println("${it.name} - $${it.price}")
}
```

- Вызовите groupBy() для списка cookies, сохранив результат в переменной groupedMenu.

```
val groupedMenu = cookies.groupBy {}
```

- Передайте лямбда-выражение, которое возвращает it.softBaked. Возвращаемый тип будет `Map<Boolean, List<Cookie>>`.

```
val groupedMenu = cookies.groupBy { it.softBaked }
```

- Создайте переменную softBakedMenu, содержащую значение groupedMenu[true], и переменную crunchyMenu, содержащую значение groupedMenu[false]. Поскольку результат подписки Map является нулевым, вы можете использовать оператор Элвиса (?😃, чтобы вернуть пустой список.

```
val softBakedMenu = groupedMenu[true] ?: listOf()
val crunchyMenu = groupedMenu[false] ?: listOf()
```

> Примечание: В качестве альтернативы emptyList() создает пустой список и может быть более читабельным.

- Добавьте код для печати меню для мягкого печенья, а затем меню для хрустящего печенья.

```
println("Soft cookies:")
softBakedMenu.forEach {
    println("${it.name} - $${it.price}")
}
println("Crunchy cookies:")
crunchyMenu.forEach {
    println("${it.name} - $${it.price}")
}
```

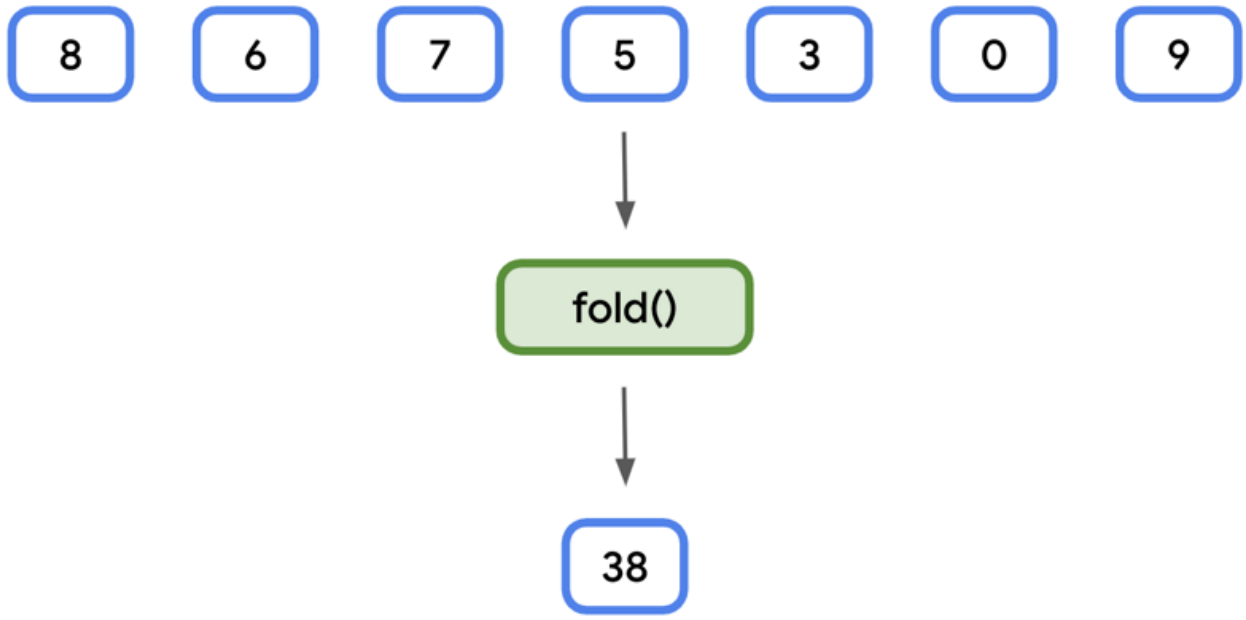
- Запустите свой код. С помощью функции groupBy() вы разделите список на две части, основываясь на значении одного из свойств.

```
...
Soft cookies:
Banana Walnut - $1.49
Snickerdoodle - $1.39
Blueberry Tart - $1.79
Crunchy cookies:
Chocolate Chip - $1.69
Vanilla Creme - $1.59
Chocolate Peanut Butter - $1.49
Sugar and Sprinkles - $1.39
```

> Примечание: Если вам нужно только разделить список на две части, альтернативой может быть функция `partition()`.

fold()

Функция `fold()` используется для получения одного значения из коллекции. Чаще всего она используется для таких задач, как подсчет общего количества цен или суммирование всех элементов списка для нахождения среднего значения.



Функция fold() принимает два параметра:

- Начальное значение. Тип данных определяется при вызове функции (то есть начальное значение 0 считается Int).
- Лямбда-выражение, возвращающее значение того же типа, что и начальное значение.

Лямбда-выражение дополнительно имеет два параметра:

- Первый называется аккумулятором. Он имеет тот же тип данных, что и начальное значение. Считайте, что это - промежуточный итог. Каждый раз, когда вызывается лямбда-выражение, аккумулятор равен возвращаемому значению, полученному в предыдущий раз, когда была вызвана лямбда.
- Второй элемент имеет тот же тип, что и каждый элемент коллекции. Как и в других функциях, которые вы уже видели, лямбда-выражение вызывается для каждого элемента коллекции, поэтому вы можете использовать `fold()` как лаконичный способ суммирования всех элементов.

Давайте воспользуемся функцией fold(), чтобы вычислить общую стоимость всех печений.

- В main() создайте новую переменную totalPrice и установите ее равной результату вызова fold() для списка печенья. В качестве начального значения передайте 0,0. Предполагается, что ее тип - Double.

```
val totalPrice = cookies.fold(0.0) {
}
```

Для лямбда-выражения необходимо указать оба параметра. Для аккумулятора используйте total, а для элемента коллекции - cookie. Используйте стрелку (->) после списка параметров.

```
val totalPrice = cookies.fold(0.0) {total, cookie ->
}
```

- В теле лямбды вычислите сумму total и cookie.price. Это значение считается возвращаемым и при следующем вызове лямбды передается в качестве total.

```
val totalPrice = cookies.fold(0.0) {total, cookie ->
    total + cookie.price
}
```

- Выведите значение totalPrice, отформатированное в виде строки для удобства чтения.

```
println("Total price: $${totalPrice}")
```

- Запустите свой код. Результат должен быть равен сумме цен в списке печенья.

```
...
Total price: $10.83
```

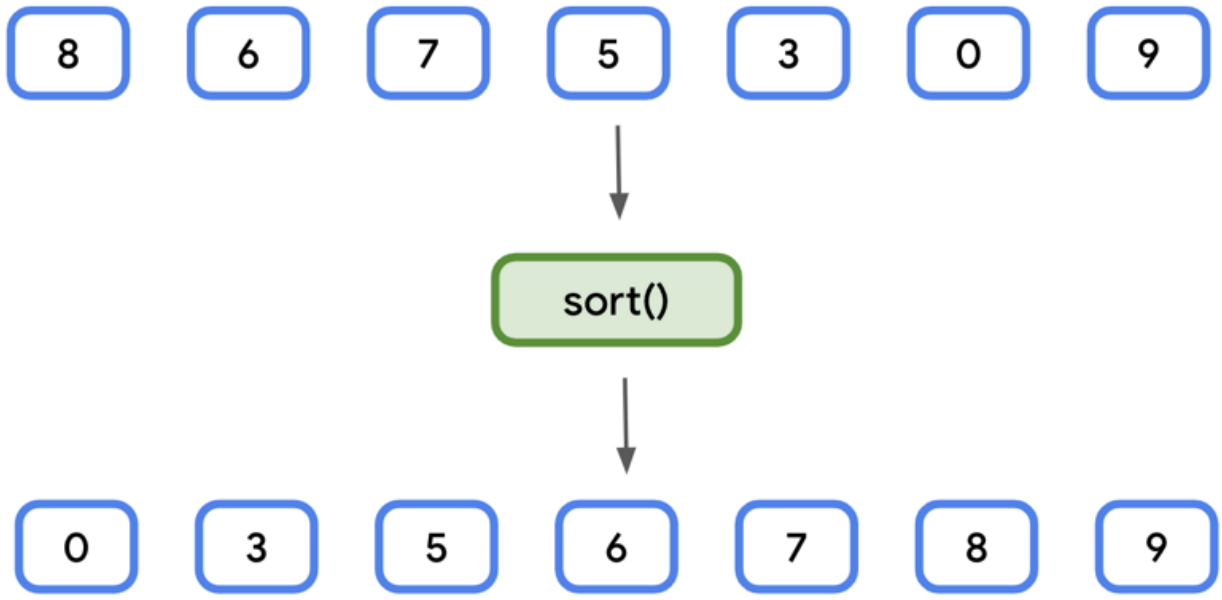
> Примечание: функцию fold() иногда называют reduce(). Функция fold() в Kotlin работает так же, как и функция reduce() в JavaScript, Swift, Python и т. д. Обратите внимание, что в Kotlin также есть своя функция `reduce()`, в которой накопитель начинается с первого элемента коллекции, а не с начального значения, передаваемого в качестве аргумента.

> Примечание: В коллекциях Kotlin также есть функция `sum()` для числовых типов, а также функция более высокого порядка `sumOf()`.

sortedBy()

Когда вы только знакомились с коллекциями, вы узнали, что для сортировки элементов можно использовать функцию `sort()`. Однако это не сработает для коллекции объектов Cookie. Класс Cookie имеет несколько свойств, и Kotlin не будет знать, по каким свойствам (имя, цена и т. д.) вы хотите отсортировать.

Для таких случаев в коллекциях Kotlin предусмотрена функция `sortedBy()`. sortedBy() позволяет указать лямбду, которая возвращает свойство, по которому вы хотите отсортировать. Например, если вы хотите отсортировать по цене, лямбда вернет it.price. Если тип данных значения имеет естественный порядок сортировки - строки сортируются по алфавиту, числовые значения сортируются по возрастанию - оно будет отсортировано так же, как и коллекция этого типа.



Вы будете использовать sortedBy() для сортировки списка cookies по алфавиту.

- В main() после существующего кода добавьте новую переменную под названием alphabeticalMenu и установите ее равной вызову sortedBy() для списка cookies.

```
val alphabeticalMenu = cookies.sortedBy {
}
```

- В лямбда-выражении верните it.name. Результирующий список по-прежнему будет иметь тип `List<Cookie>`, но отсортирован по имени.

```
val alphabeticalMenu = cookies.sortedBy {
    it.name
}
```

- Выведите названия файлов cookie в меню в алфавитном порядке. Вы можете использовать forEach(), чтобы вывести каждое имя в новой строке.

```
println("Alphabetical menu:")
alphabeticalMenu.forEach {
    println(it.name)
}
```

- Запустите свой код. Имена cookie выводятся в алфавитном порядке.

```
...
Alphabetical menu:
```

```
Banana Walnut
Blueberry Tart
Chocolate Chip
Chocolate Peanut Butter
Snickerdoodle
Sugar and Sprinkles
Vanilla Creme
```

> Примечание: Коллекции Kotlin также имеют функцию ```sort()``, если тип данных имеет естественный порядок сортировки.

Заключение

Вы только что увидели несколько примеров использования функций высшего порядка с коллекциями. Такие распространенные операции, как сортировка и фильтрация, могут быть выполнены в одной строке кода, что делает ваши программы более лаконичными и выразительными.

Резюме

- Вы можете перебирать каждый элемент коллекции с помощью функции forEach().
- В строки можно вставлять выражения.
- map() используется для форматирования элементов коллекции, часто в виде коллекции другого типа данных.
- filter() позволяет создать подмножество коллекции.
- groupBy() разделяет коллекцию на основе возвращаемого значения функции.
- fold() превращает коллекцию в одно значение.
- sortedBy() используется для сортировки коллекции по заданному свойству.