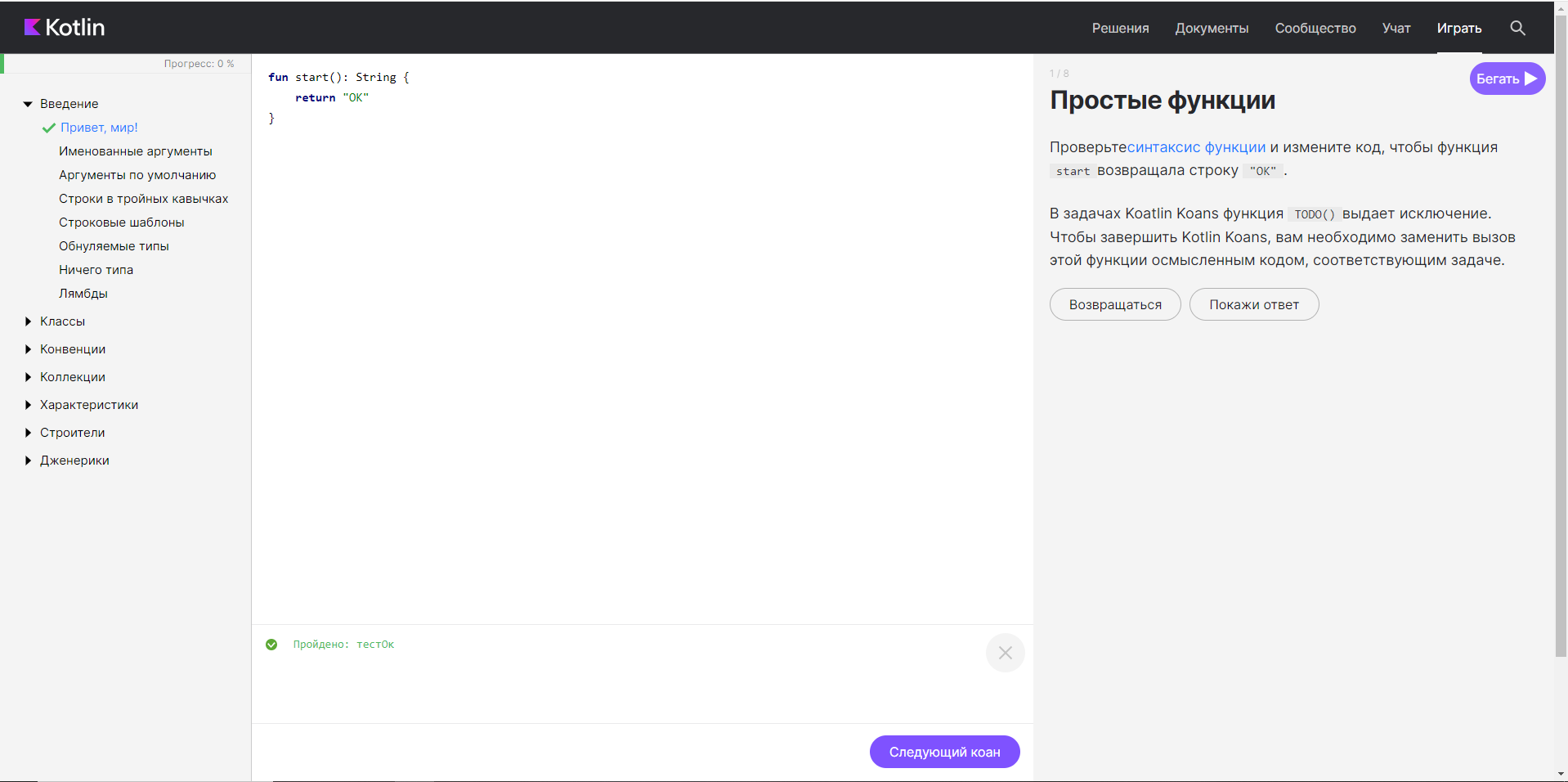
Еловко Дарья0395

Задания в [Kotlin Koans](https://play.kotlinlang.org/koans/overview)

**ВВЕДЕНИЕ**

1. Задание «Привет, мир!»

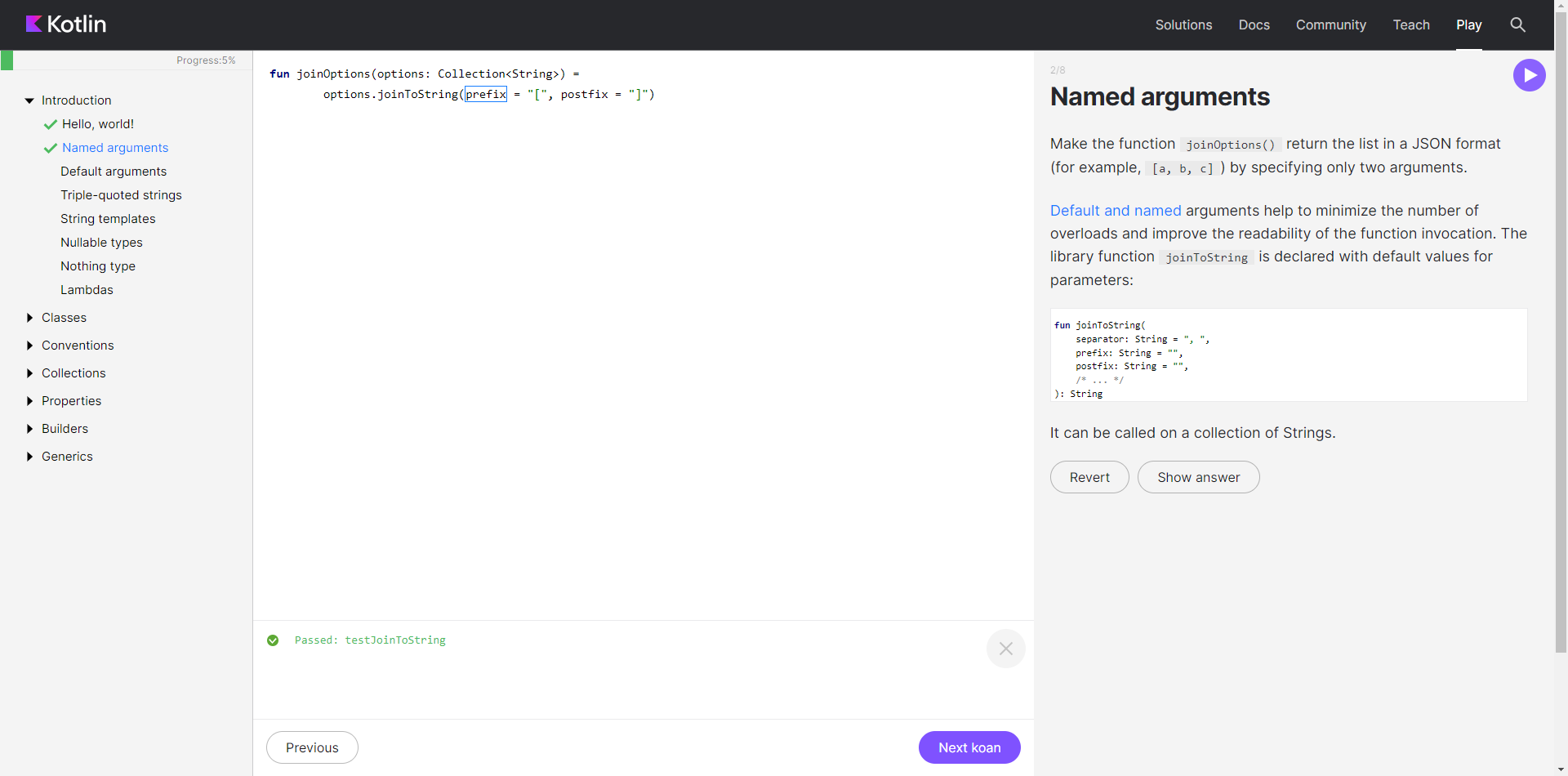


fun start(): String {

return "OK"

}

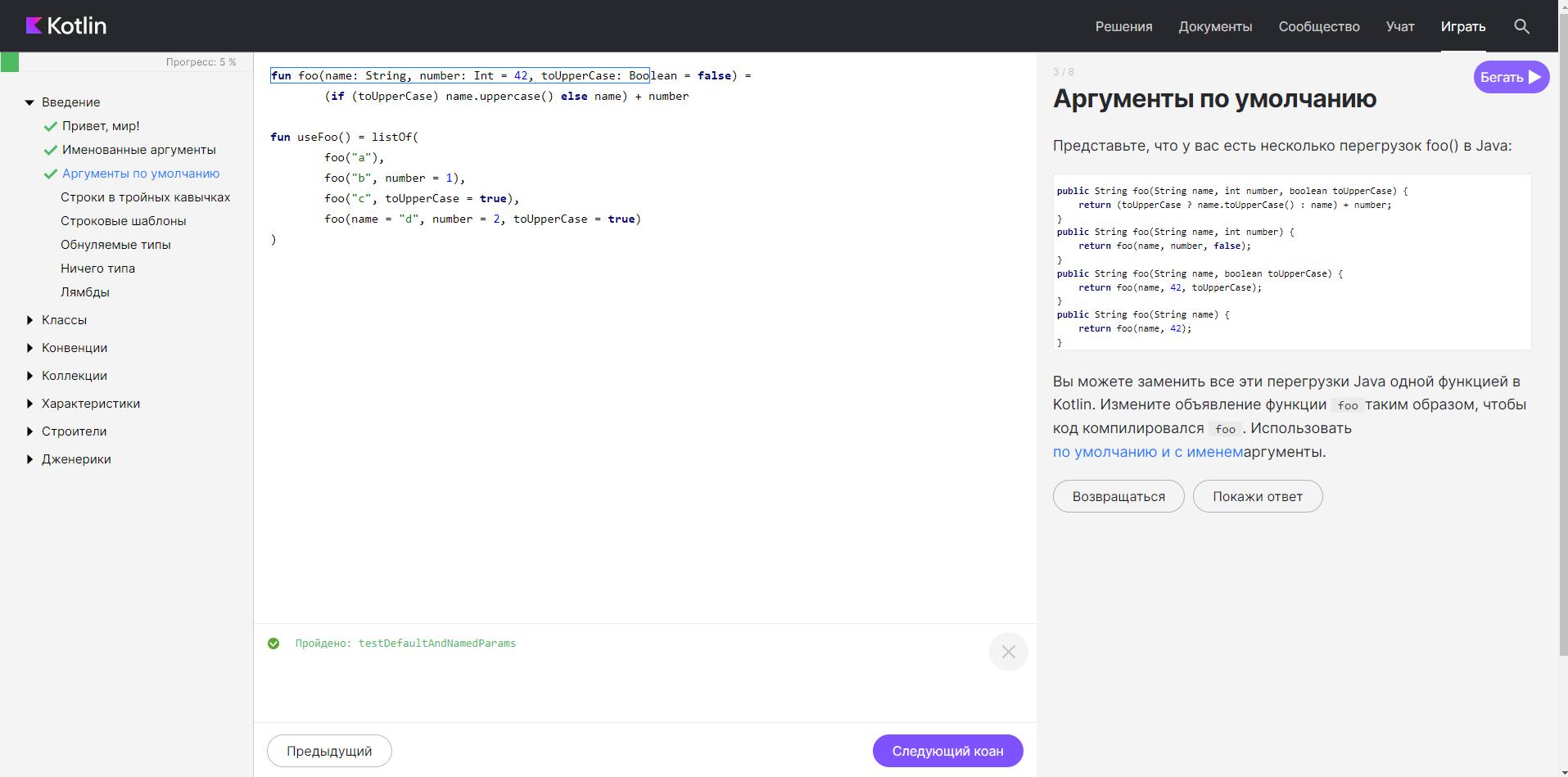
1. Задание «Именованные аргументы»



fun joinOptions(options: Collection<String>) =

options.joinToString(prefix = "[", postfix = "]")

1. Задание «Аргументы по умолчанию»



fun foo(name: String, number: Int = 42, toUpperCase: Boolean = false) =

(if (toUpperCase) name.uppercase() else name) + number

fun useFoo() = listOf(

foo("a"),

foo("b", number = 1),

foo("c", toUpperCase = true),

foo(name = "d", number = 2, toUpperCase = true)

)

1. Задание «Строки в тройных кавычках»



const val question = "life, the universe, and everything"

const val answer = 42

val tripleQuotedString = """

#question = "$question"

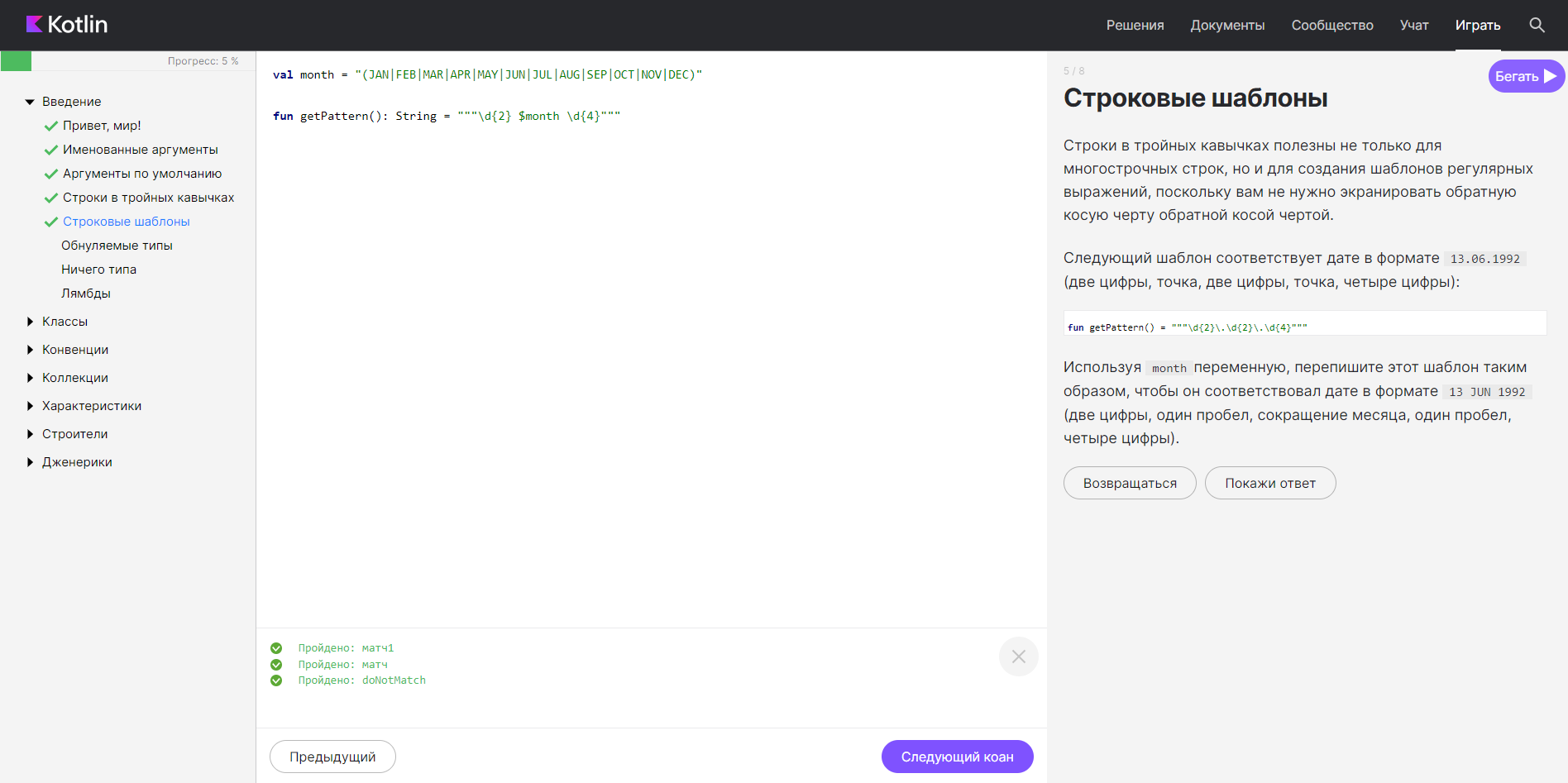
#answer = $answer""".trimMargin("#")

fun main() {

println(tripleQuotedString)

}

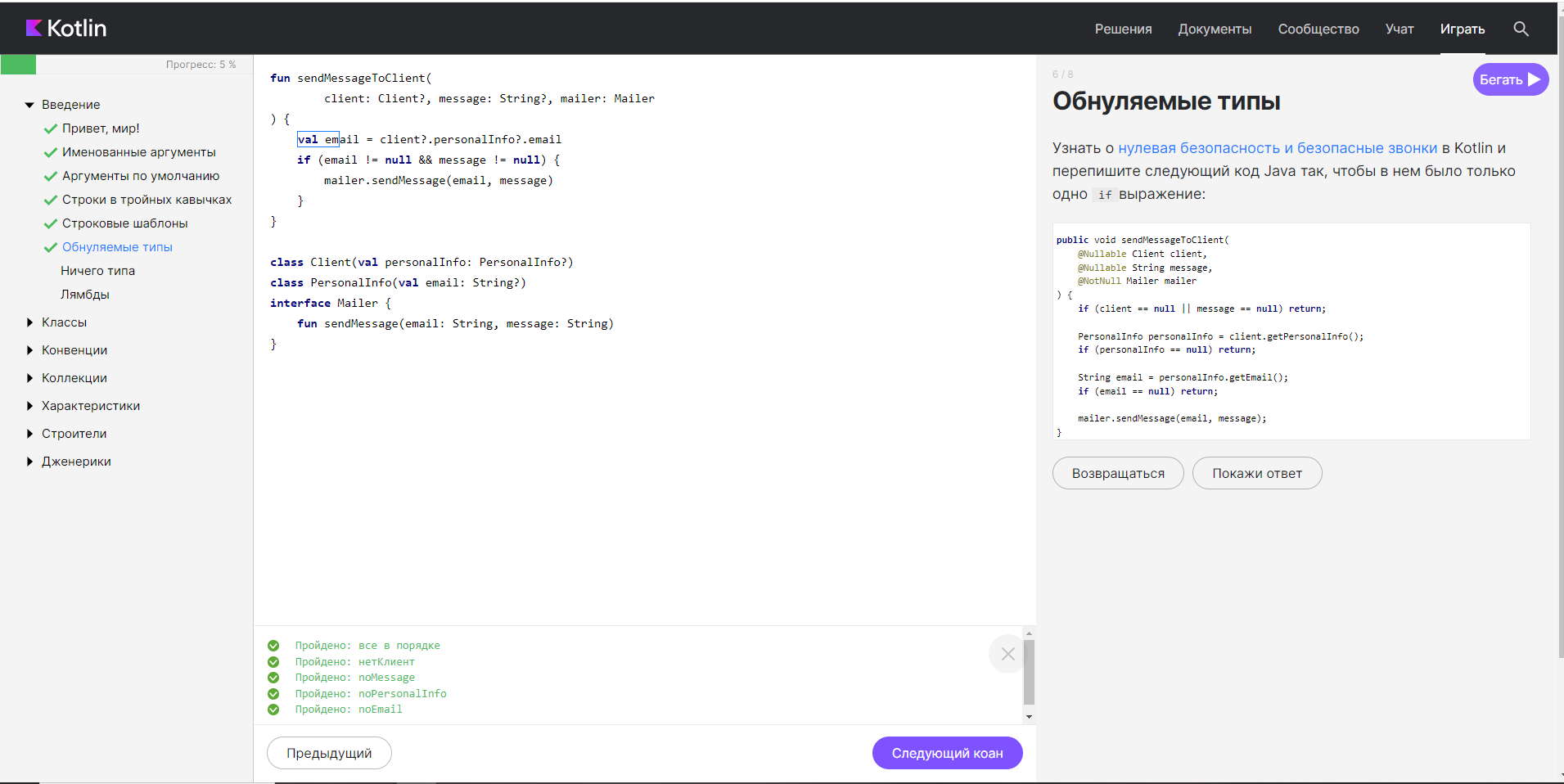
1. Задание «Строковые шаблоны»



val month = "(JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC)"

fun getPattern(): String = """\d{2} $month \d{4}"""

1. Задание «Обнуляем типы»



fun sendMessageToClient(

client: Client?, message: String?, mailer: Mailer

) {

val email = client?.personalInfo?.email

if (email != null && message != null) {

mailer.sendMessage(email, message)

}

}

class Client(val personalInfo: PersonalInfo?)

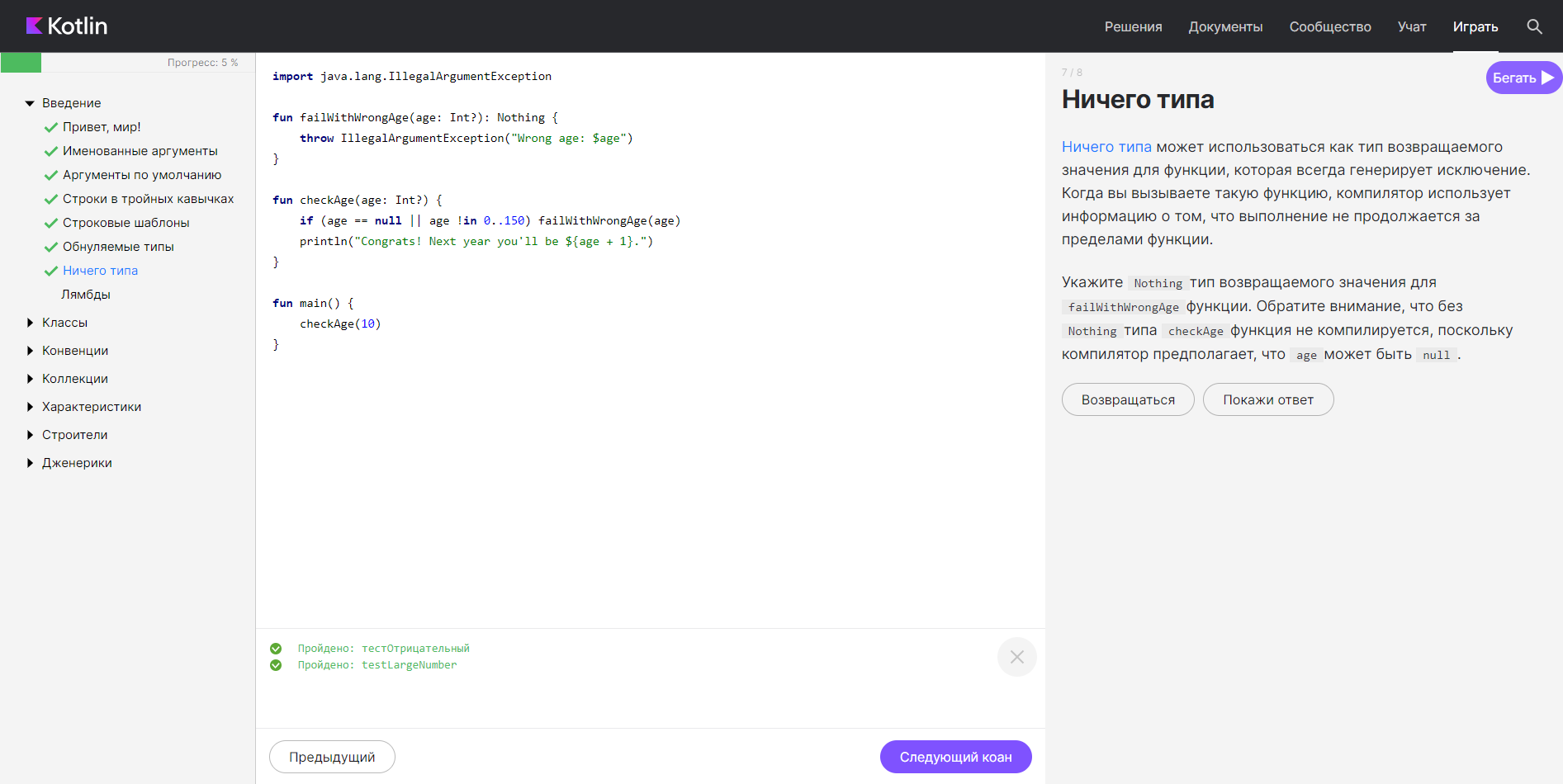
class PersonalInfo(val email: String?)

interface Mailer {

fun sendMessage(email: String, message: String)

}

1. Задание «Ничего типа»



import java.lang.IllegalArgumentException

fun failWithWrongAge(age: Int?): Nothing {

throw IllegalArgumentException("Wrong age: $age")

}

fun checkAge(age: Int?) {

if (age == null || age !in 0..150) failWithWrongAge(age)

println("Congrats! Next year you'll be ${age + 1}.")

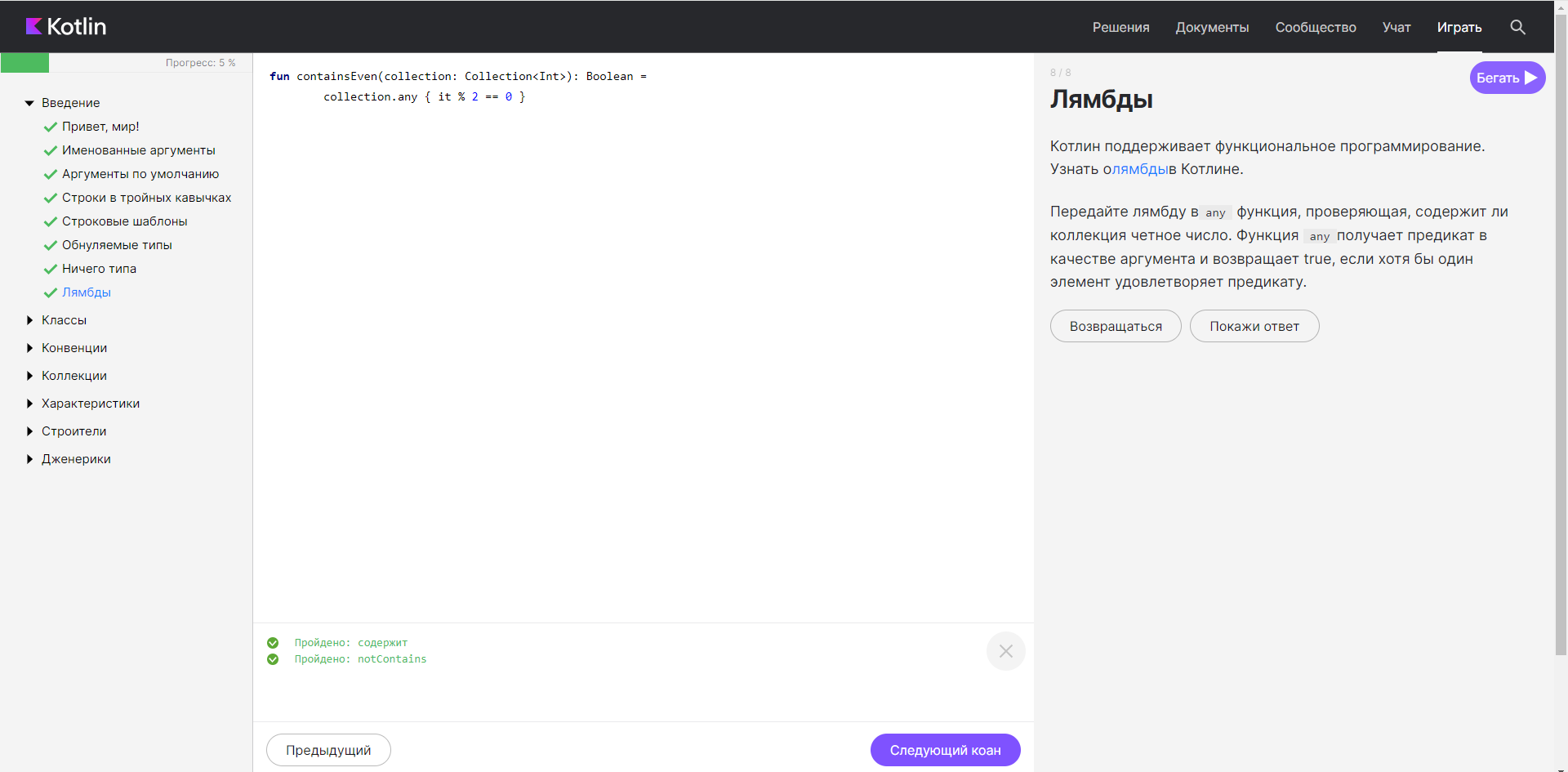
}

fun main() {

checkAge(10)

}

1. Задание «Лямбды»

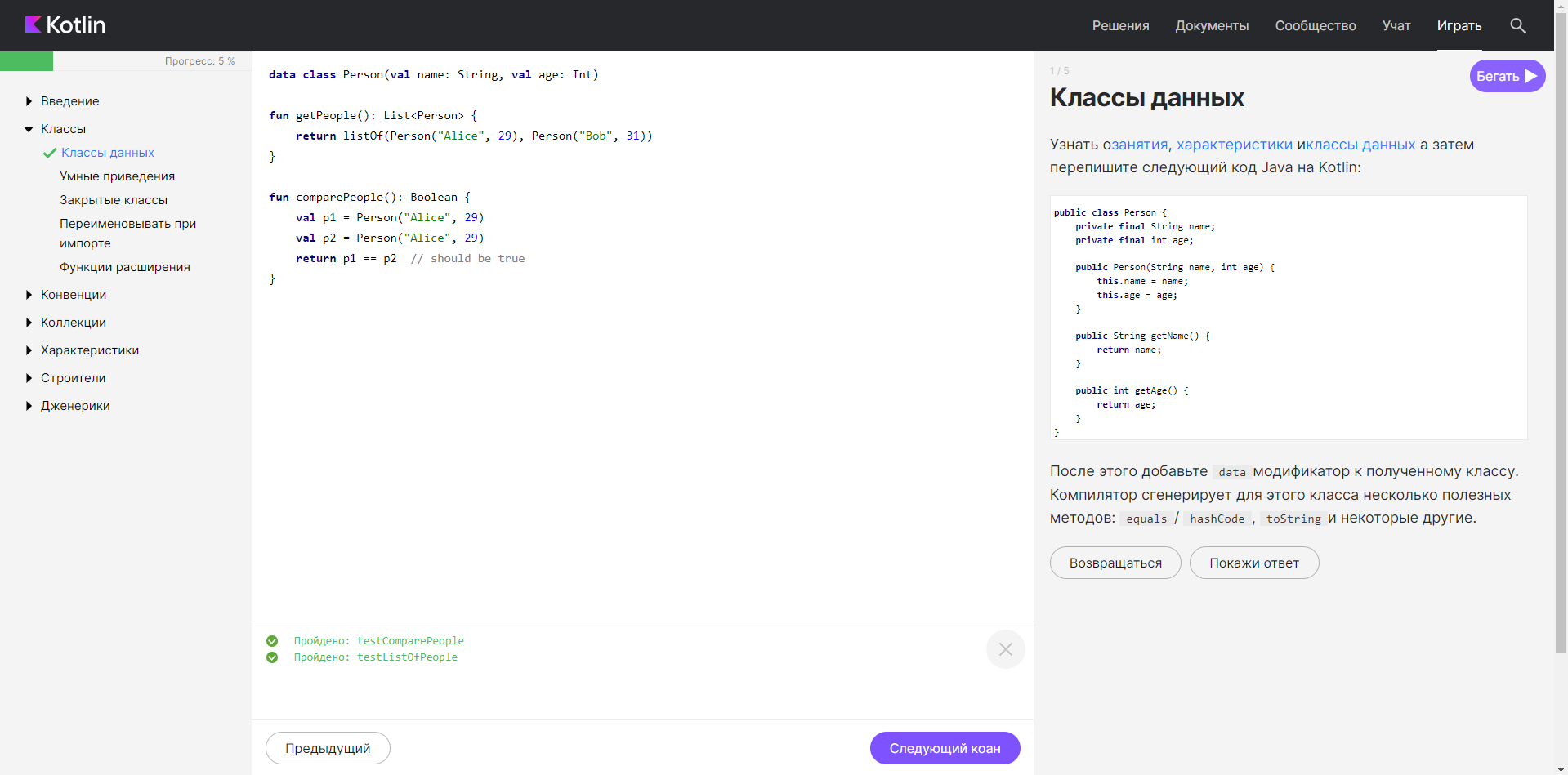


fun containsEven(collection: Collection<Int>): Boolean =

collection.any { it % 2 == 0 }

**КЛАССЫ**

1. Задание «Классы данных»



data class Person(val name: String, val age: Int)

fun getPeople(): List<Person> {

return listOf(Person("Alice", 29), Person("Bob", 31))

}

fun comparePeople(): Boolean {

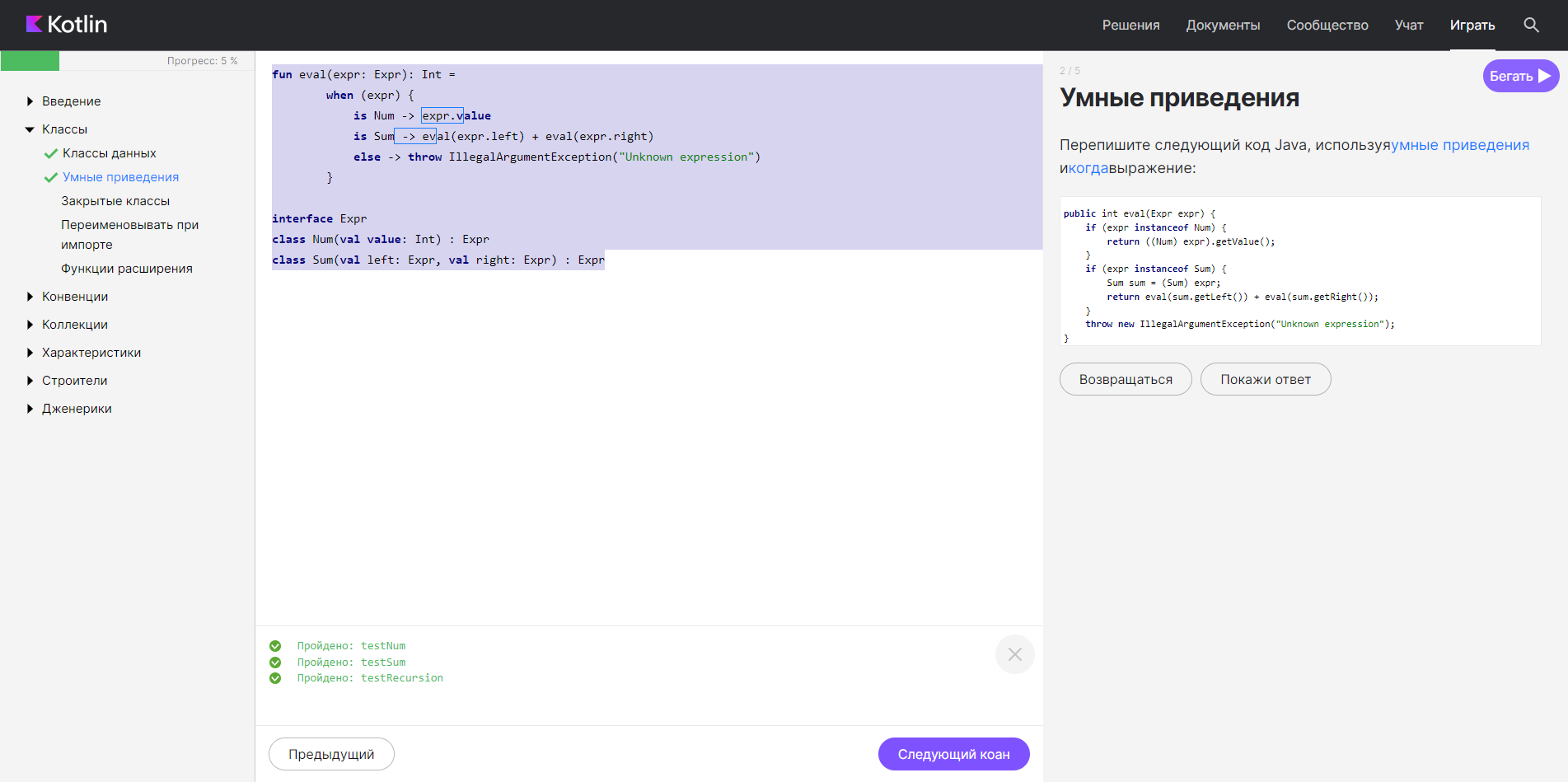
val p1 = Person("Alice", 29)

val p2 = Person("Alice", 29)

return p1 == p2 // should be true

}

1. Задание «Умные приведения»



fun eval(expr: Expr): Int =

when (expr) {

is Num -> expr.value

is Sum -> eval(expr.left) + eval(expr.right)

else -> throw IllegalArgumentException("Unknown expression")

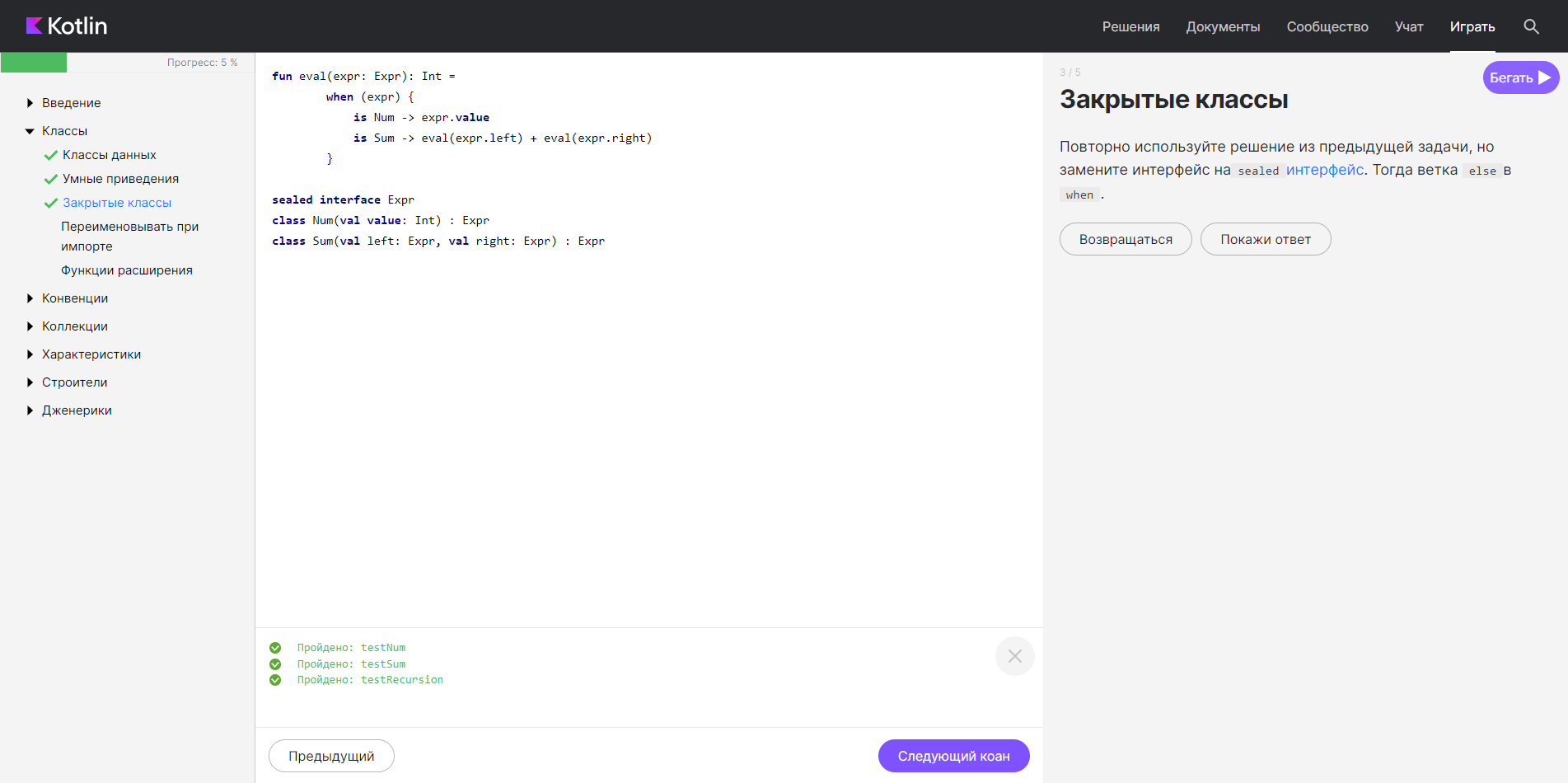
}

interface Expr

class Num(val value: Int) : Expr

class Sum(val left: Expr, val right: Expr) : Expr

1. Задание «Закрытые классы»



fun eval(expr: Expr): Int =

when (expr) {

is Num -> expr.value

is Sum -> eval(expr.left) + eval(expr.right)

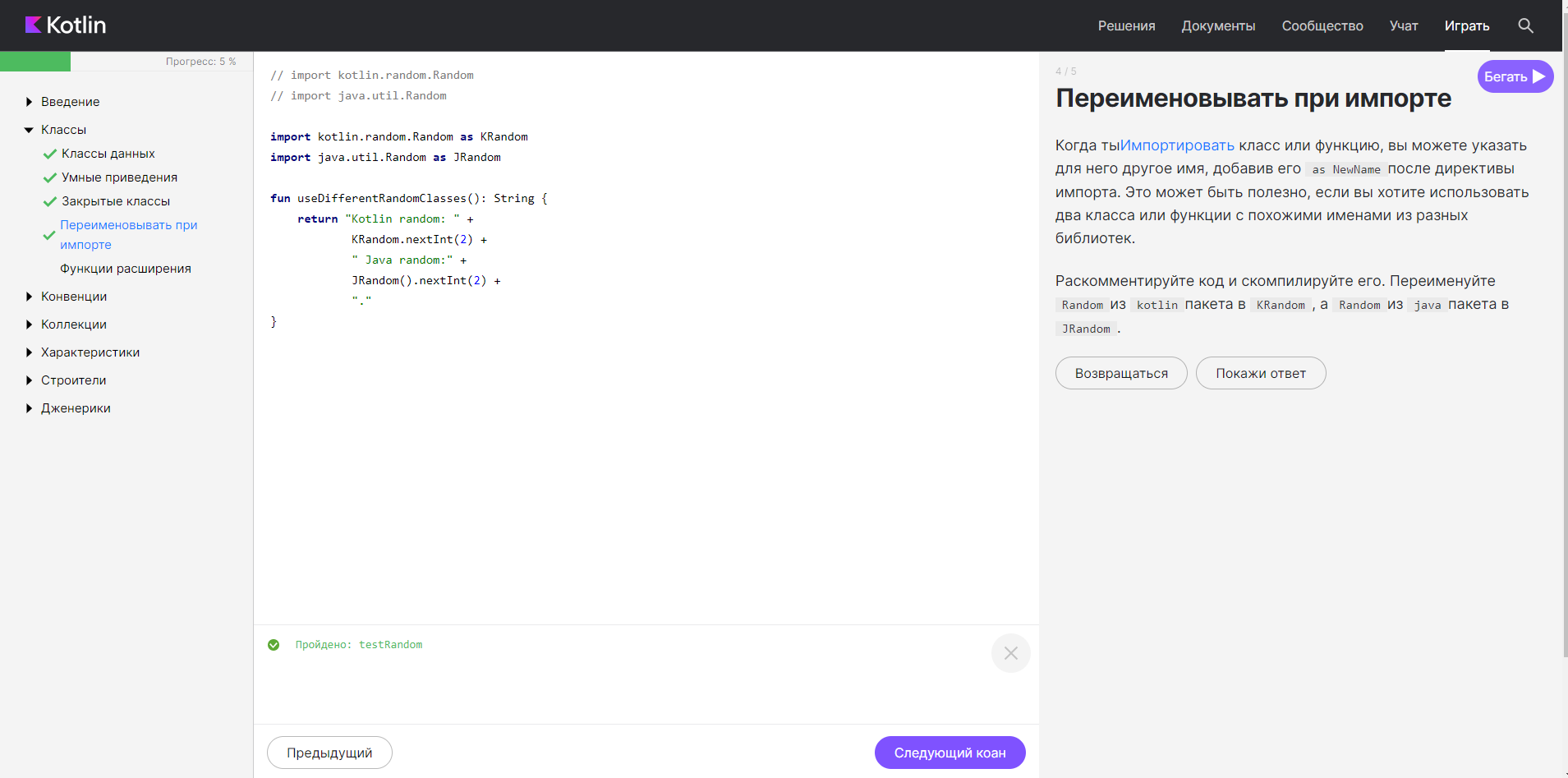
}

sealed interface Expr

class Num(val value: Int) : Expr

class Sum(val left: Expr, val right: Expr) : Expr

1. Задание «Переименовывать при импорте»



import kotlin.random.Random as KRandom

import java.util.Random as JRandom

fun useDifferentRandomClasses(): String {

return "Kotlin random: " +

KRandom.nextInt(2) +

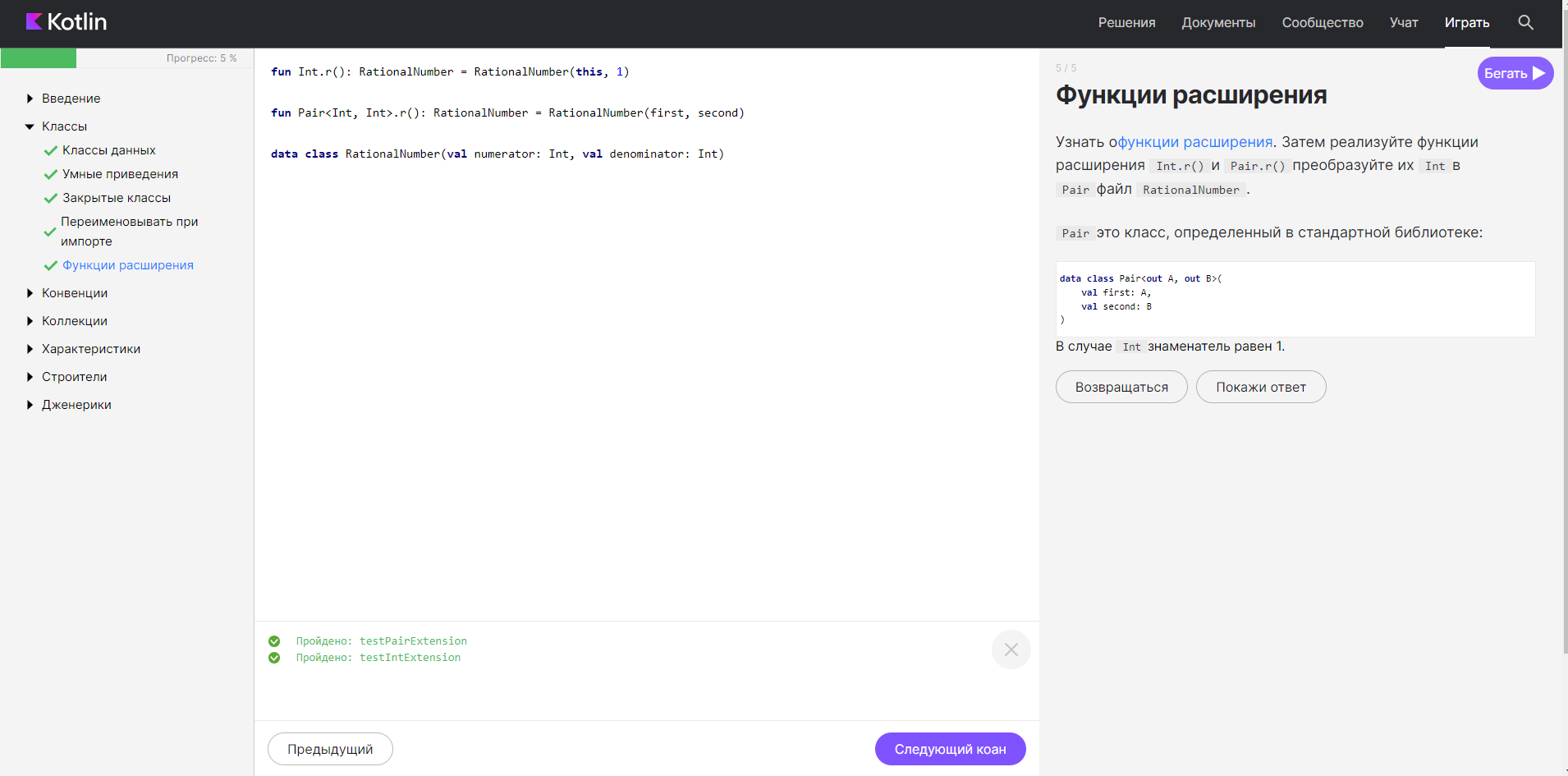
" Java random:" +

JRandom().nextInt(2) +

"."

}

1. Задание «Функции расширения»



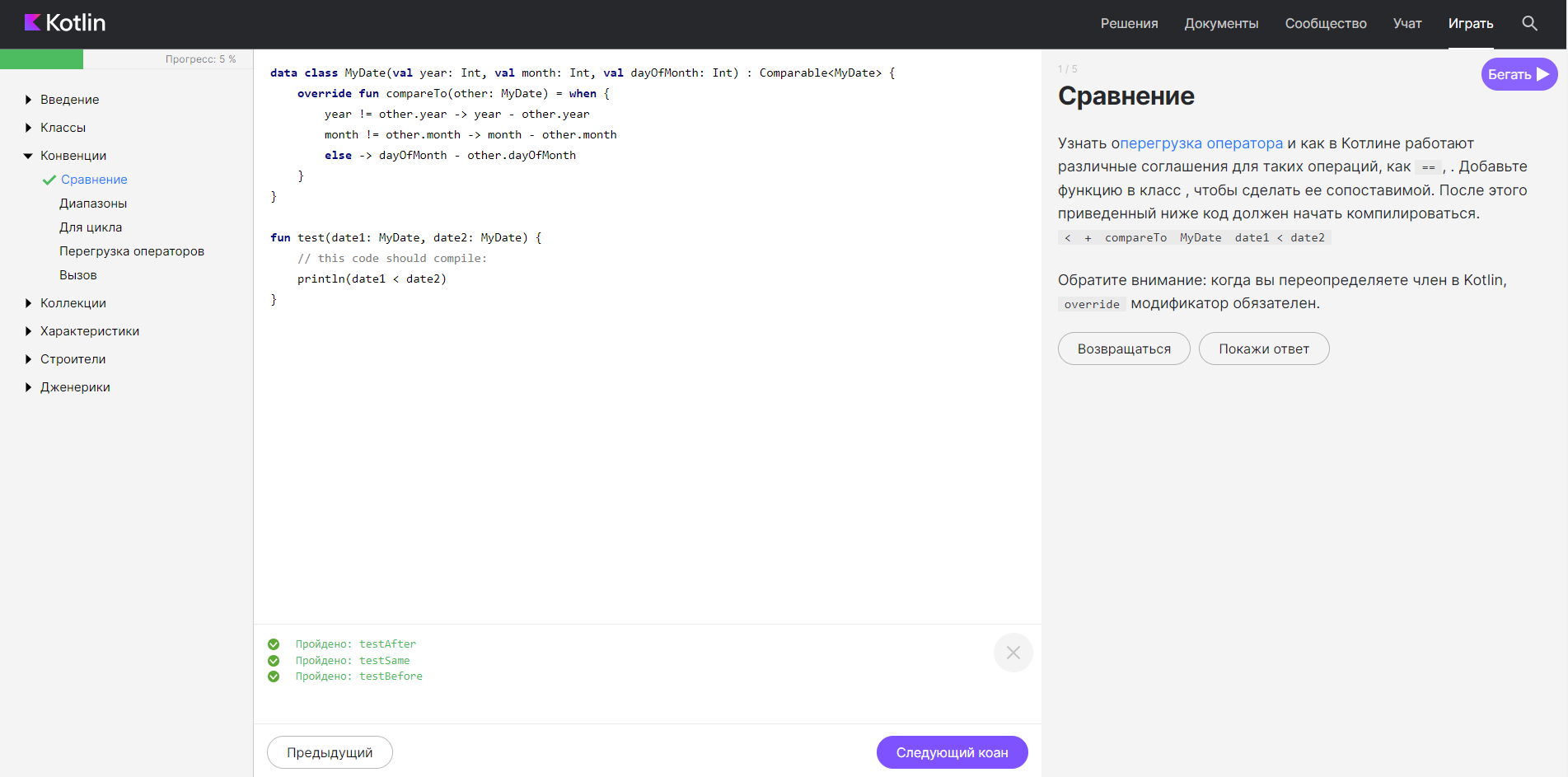
fun Int.r(): RationalNumber = RationalNumber(this, 1)

fun Pair<Int, Int>.r(): RationalNumber = RationalNumber(first, second)

data class RationalNumber(val numerator: Int, val denominator: Int)

**КОНВЕНЦИИ**

1. Задание «Сравнение»



data class MyDate(val year: Int, val month: Int, val dayOfMonth: Int) : Comparable<MyDate> {

override fun compareTo(other: MyDate) = when {

year != other.year -> year - other.year

month != other.month -> month - other.month

else -> dayOfMonth - other.dayOfMonth

}

}

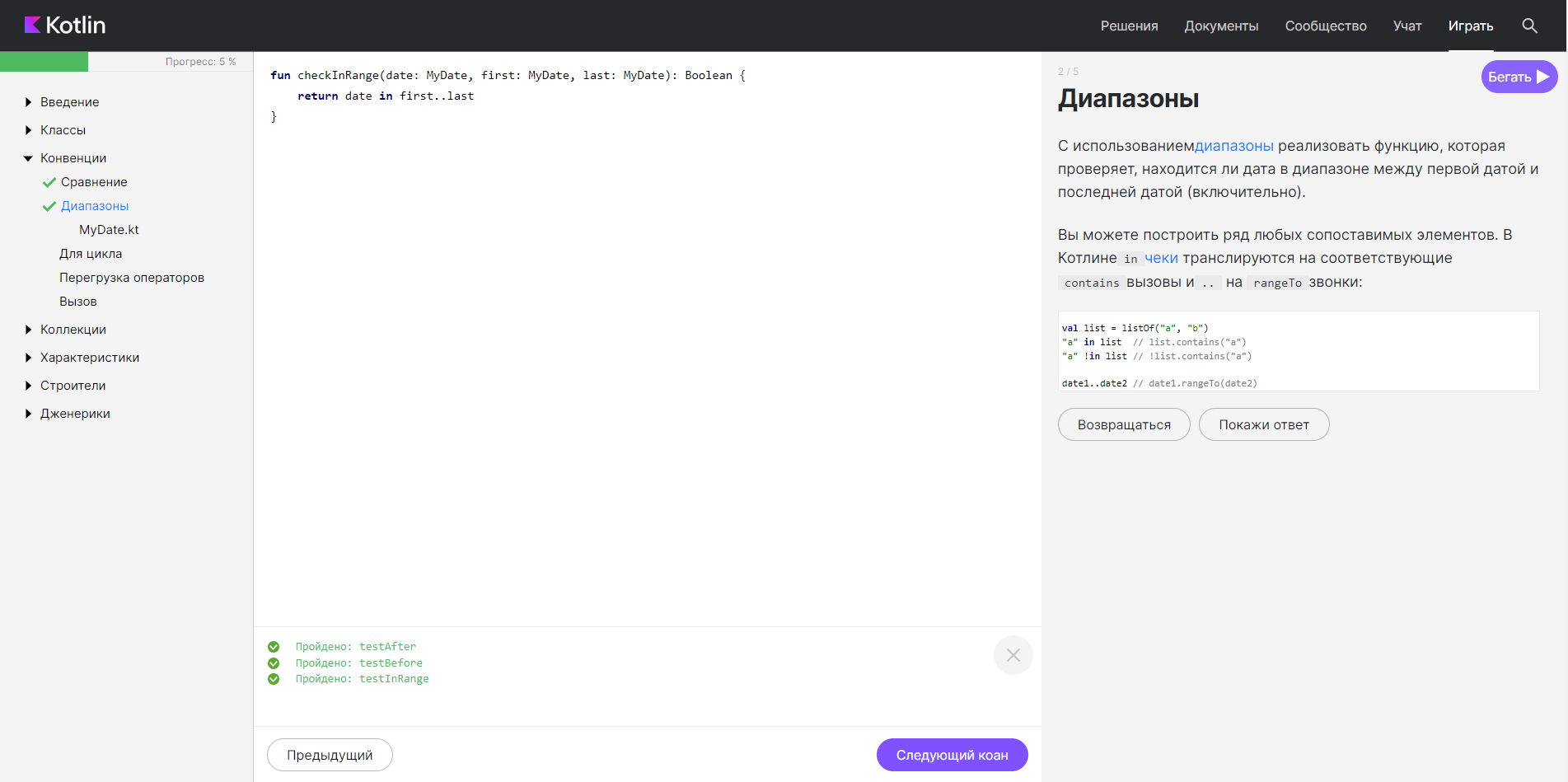
fun test(date1: MyDate, date2: MyDate) {

// this code should compile:

println(date1 < date2)

}

1. Задание «Диапазоны»

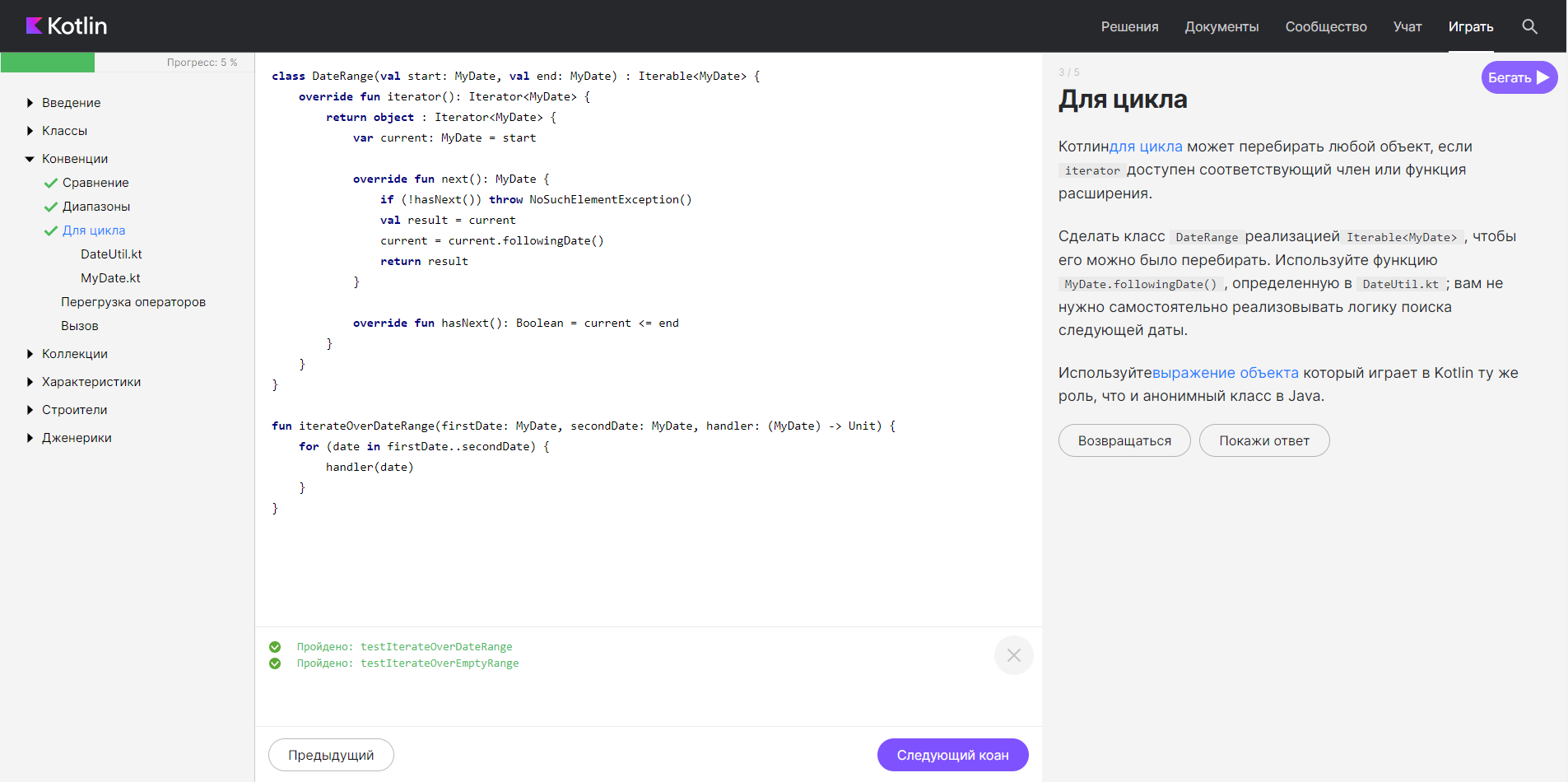


fun checkInRange(date: MyDate, first: MyDate, last: MyDate): Boolean {

return date in first..last

}

1. Задание «Цикл FOR»



class DateRange(val start: MyDate, val end: MyDate) : Iterable<MyDate> {

override fun iterator(): Iterator<MyDate> {

return object : Iterator<MyDate> {

var current: MyDate = start

override fun next(): MyDate {

if (!hasNext()) throw NoSuchElementException()

val result = current

current = current.followingDate()

return result

}

override fun hasNext(): Boolean = current <= end

}

}

}

fun iterateOverDateRange(firstDate: MyDate, secondDate: MyDate, handler: (MyDate) -> Unit) {

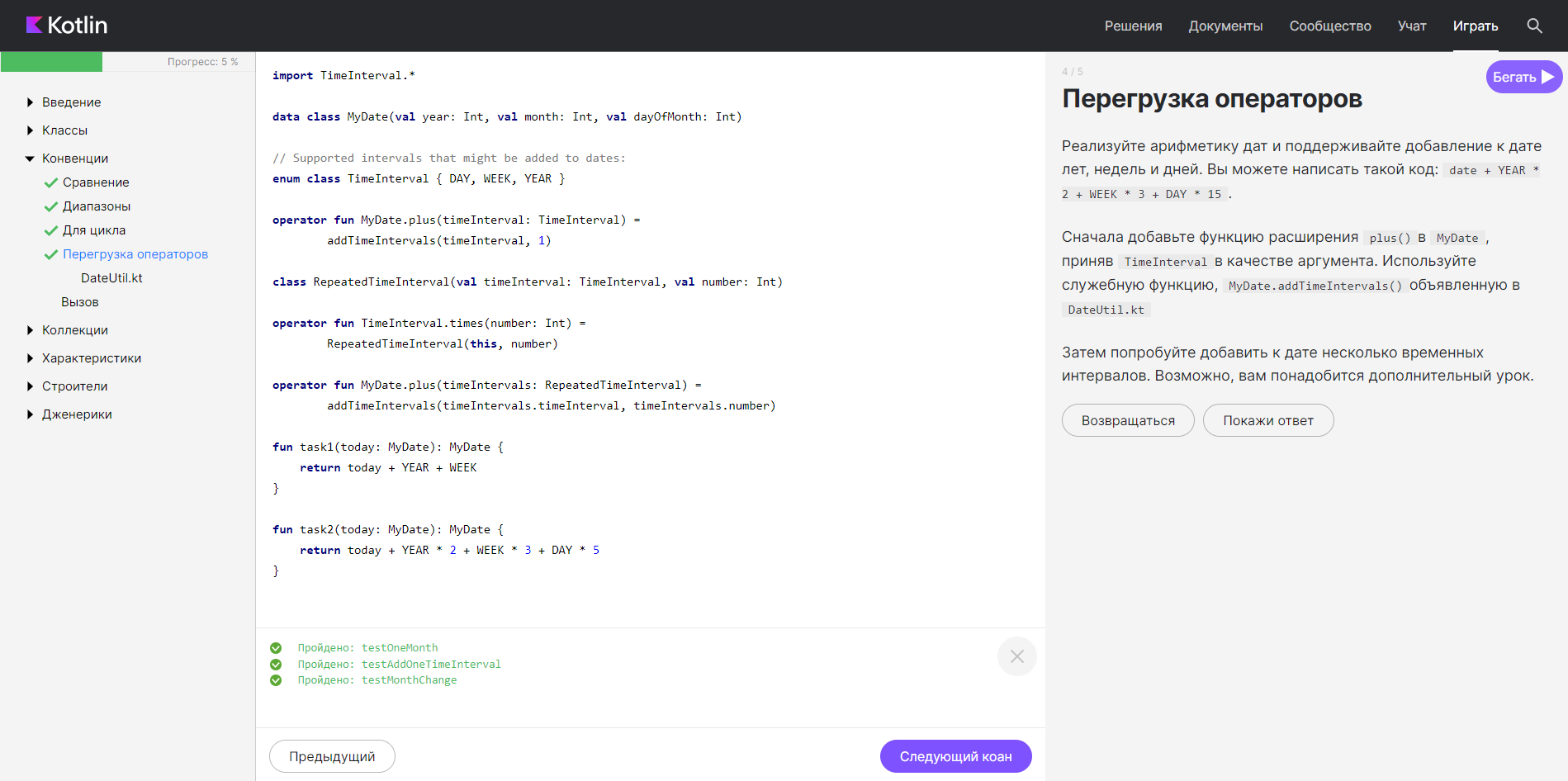
for (date in firstDate..secondDate) {

handler(date)

}

}

1. Задание «Перегрузка операторов»



import TimeInterval.\*

data class MyDate(val year: Int, val month: Int, val dayOfMonth: Int)

// Supported intervals that might be added to dates:

enum class TimeInterval { DAY, WEEK, YEAR }

operator fun MyDate.plus(timeInterval: TimeInterval) =

addTimeIntervals(timeInterval, 1)

class RepeatedTimeInterval(val timeInterval: TimeInterval, val number: Int)

operator fun TimeInterval.times(number: Int) =

RepeatedTimeInterval(this, number)

operator fun MyDate.plus(timeIntervals: RepeatedTimeInterval) =

addTimeIntervals(timeIntervals.timeInterval, timeIntervals.number)

fun task1(today: MyDate): MyDate {

return today + YEAR + WEEK

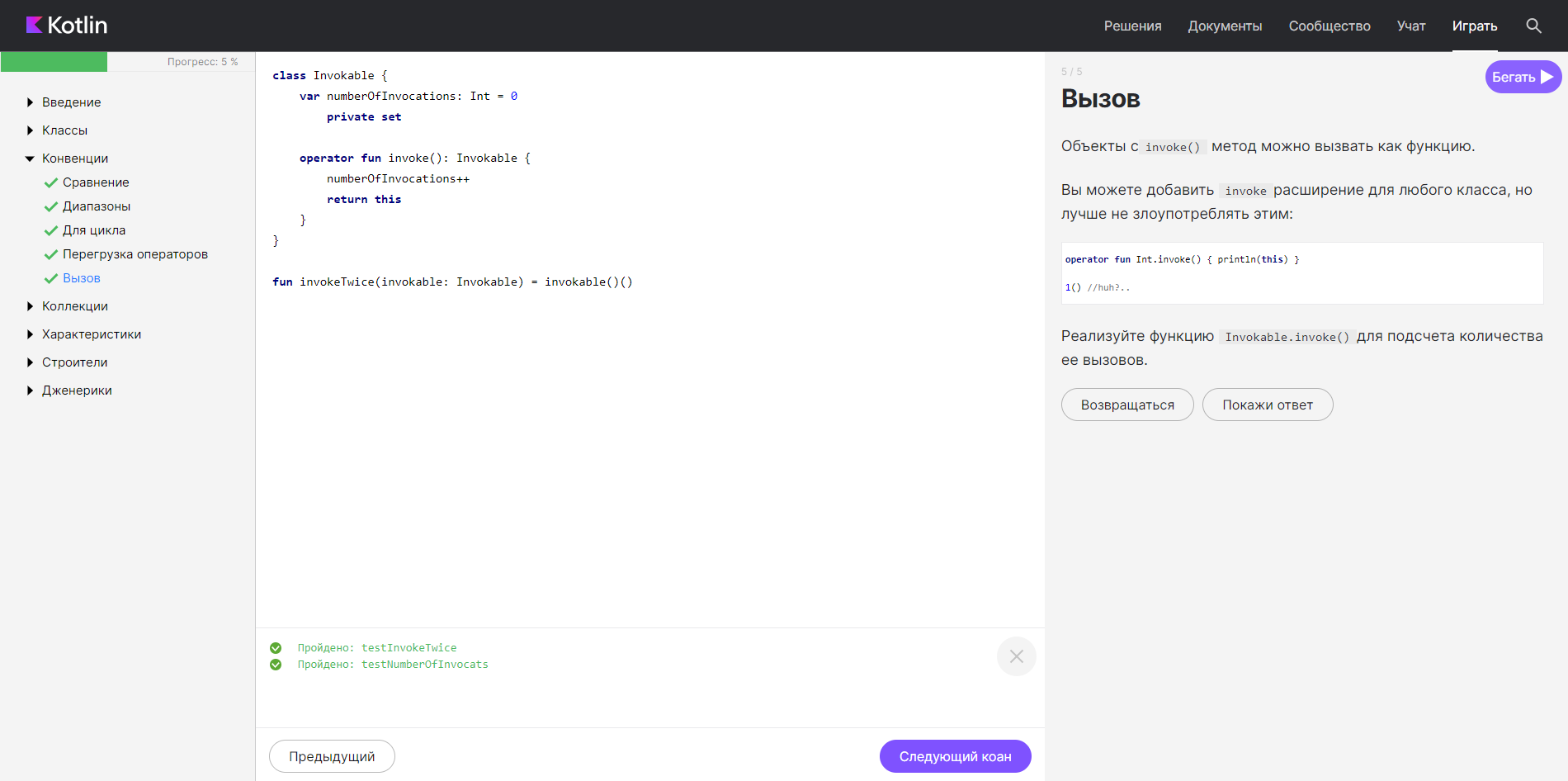
}

fun task2(today: MyDate): MyDate {

return today + YEAR \* 2 + WEEK \* 3 + DAY \* 5

}

1. Задание «Вызов»



class Invokable {

var numberOfInvocations: Int = 0

private set

operator fun invoke(): Invokable {

numberOfInvocations++

return this

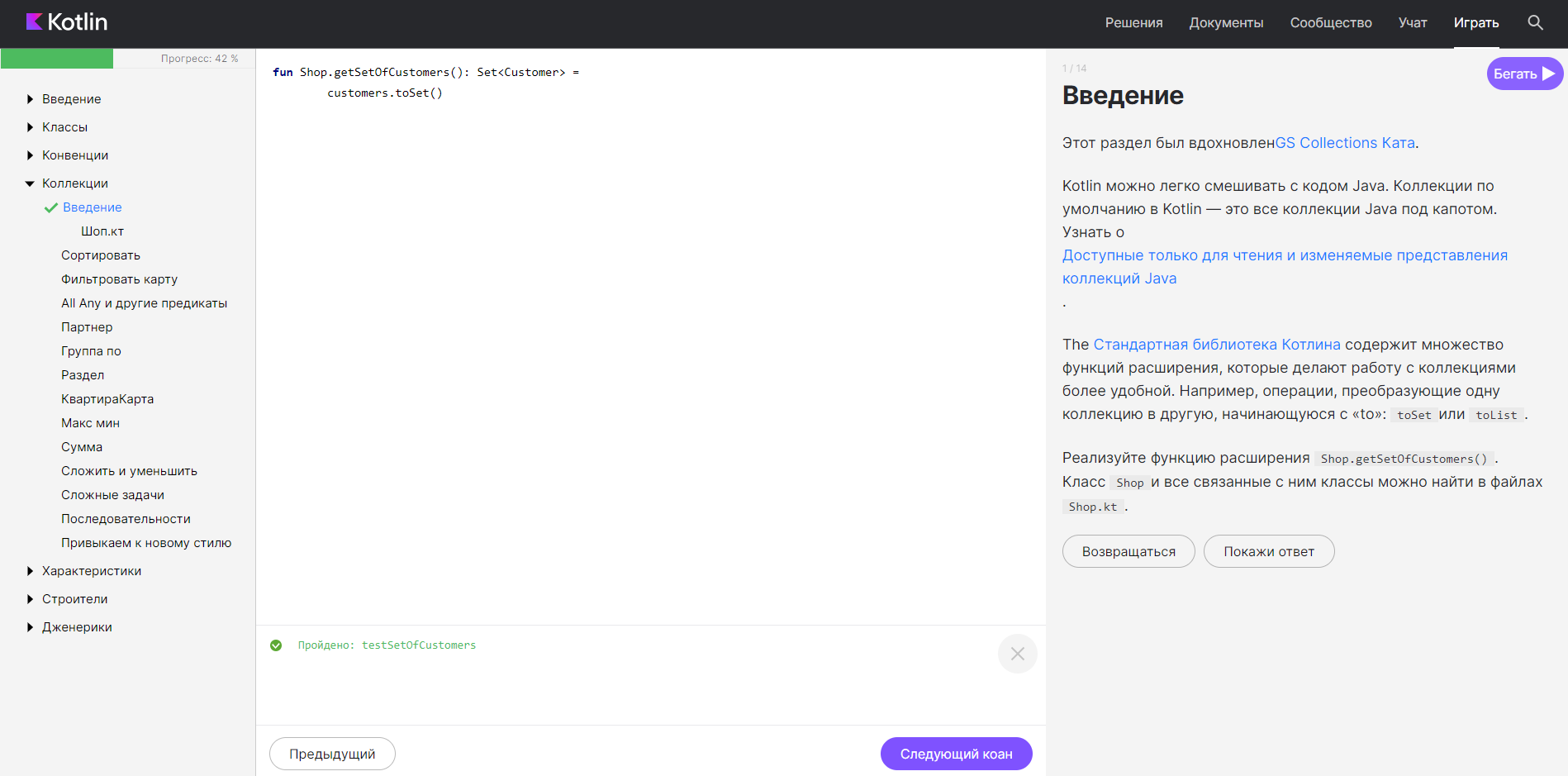
}

}

fun invokeTwice(invokable: Invokable) = invokable()()

**КОЛЛЕКЦИИ**

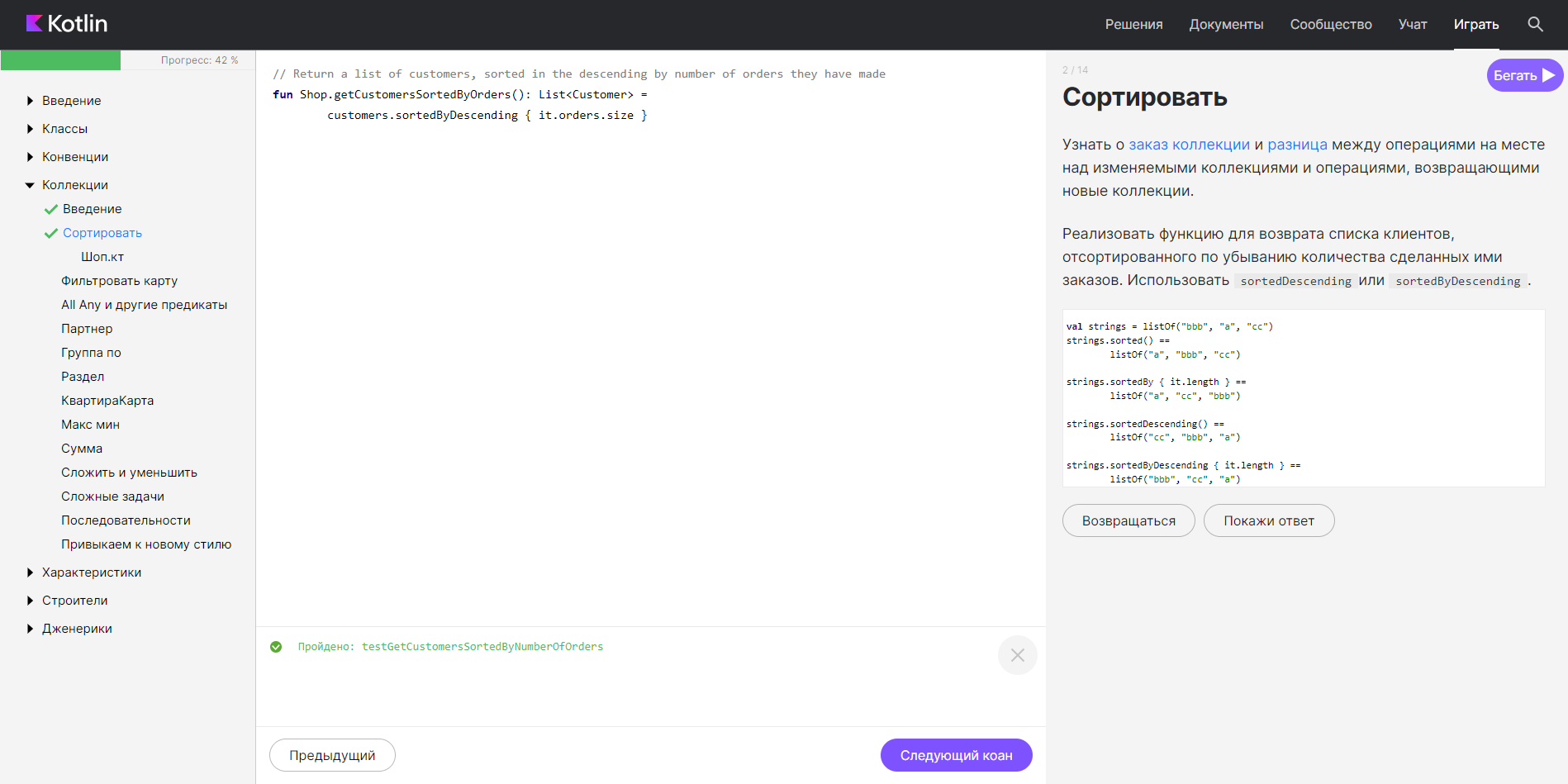
1. Задание «Введение»



fun Shop.getSetOfCustomers(): Set<Customer> =

customers.toSet()

1. Задание «Сортировать»

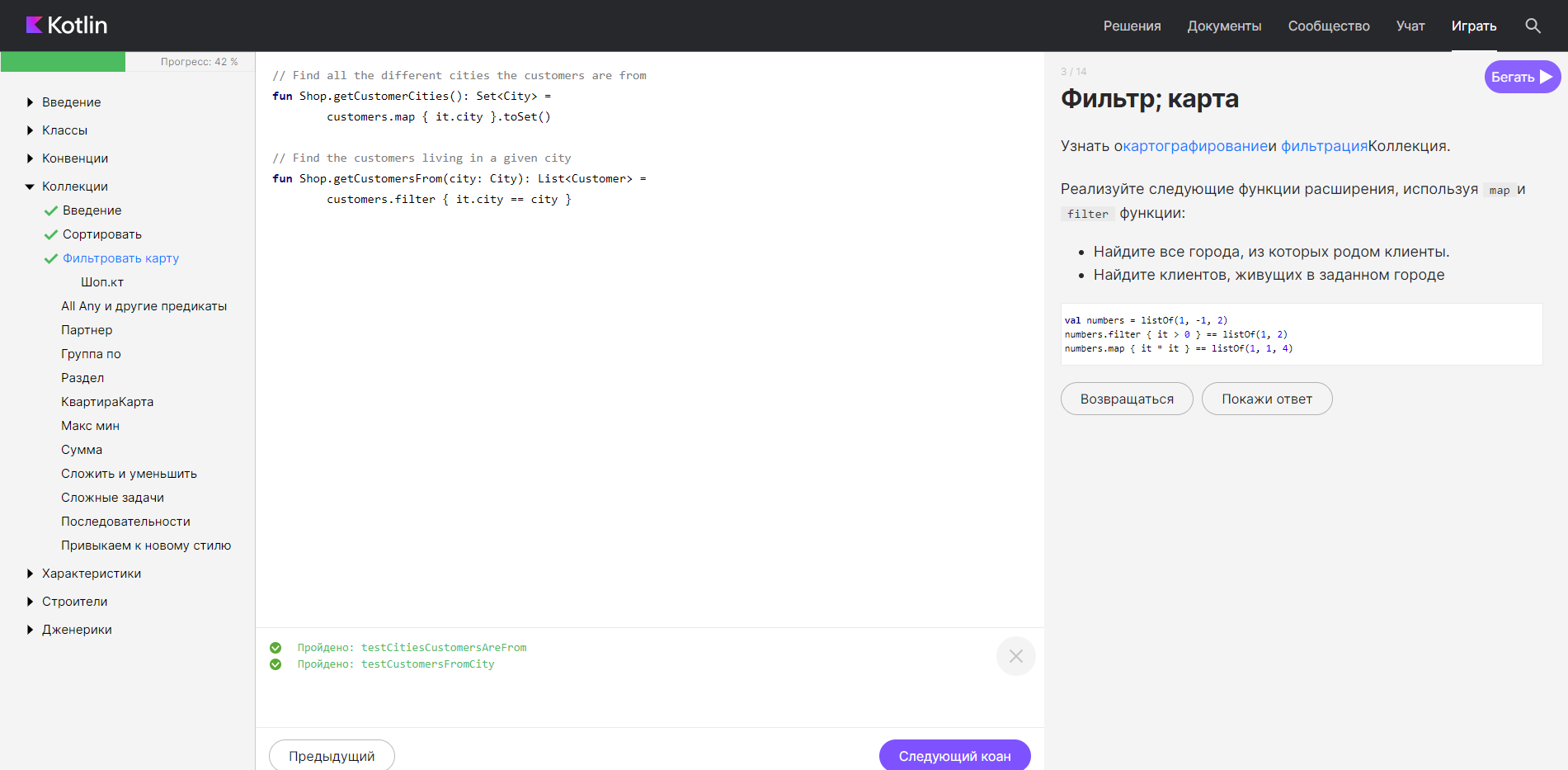


// Return a list of customers, sorted in the descending by number of orders they have made

fun Shop.getCustomersSortedByOrders(): List<Customer> =

customers.sortedByDescending { it.orders.size }

1. Задание «Фильтровать Map»



// Find all the different cities the customers are from

fun Shop.getCustomerCities(): Set<City> =

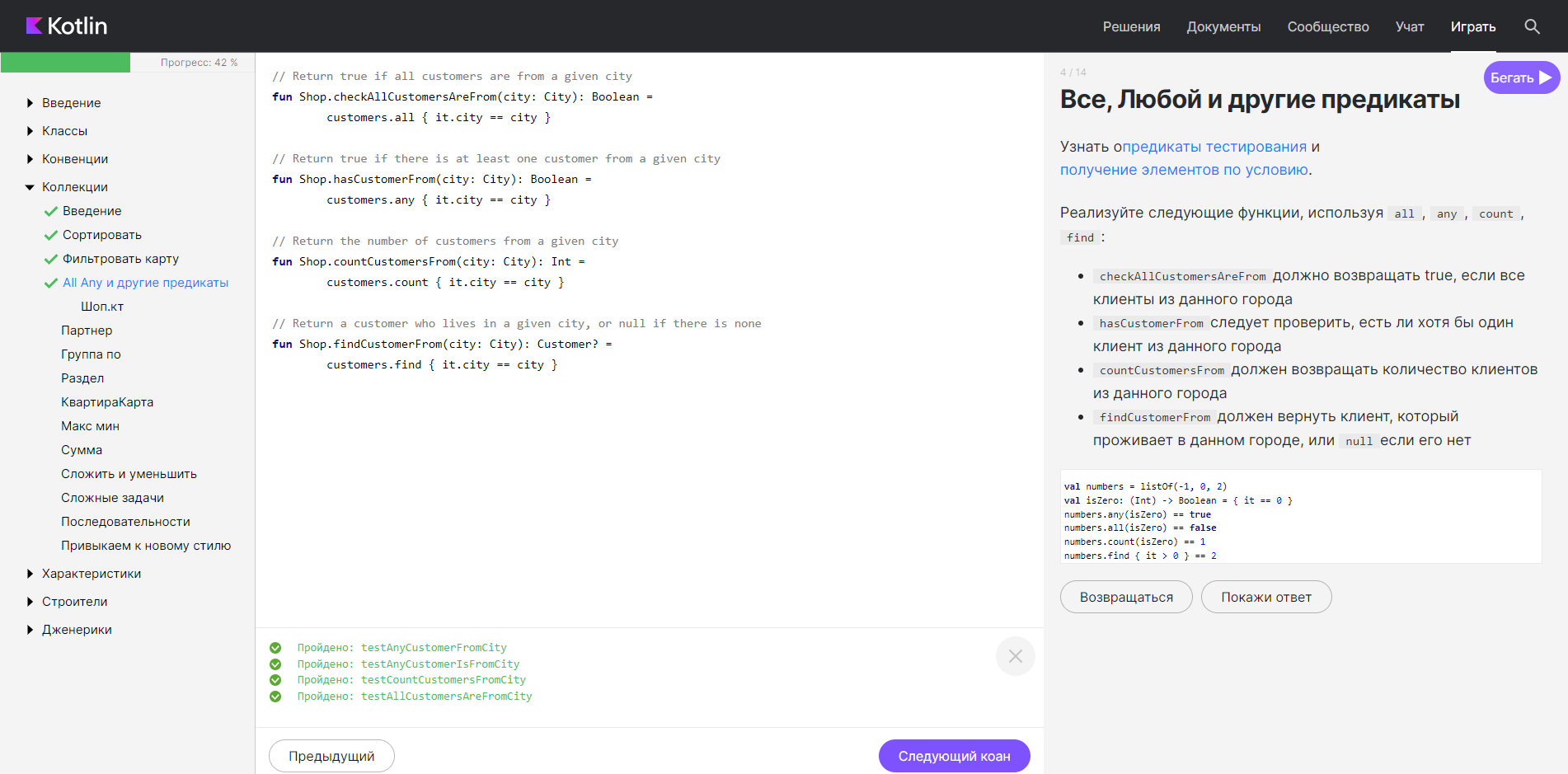
customers.map { it.city }.toSet()

// Find the customers living in a given city

fun Shop.getCustomersFrom(city: City): List<Customer> =

customers.filter { it.city == city }

1. Задание «All, Any и другие предикаты»



// Return true if all customers are from a given city

fun Shop.checkAllCustomersAreFrom(city: City): Boolean =

customers.all { it.city == city }

// Return true if there is at least one customer from a given city

fun Shop.hasCustomerFrom(city: City): Boolean =

customers.any { it.city == city }

// Return the number of customers from a given city

fun Shop.countCustomersFrom(city: City): Int =

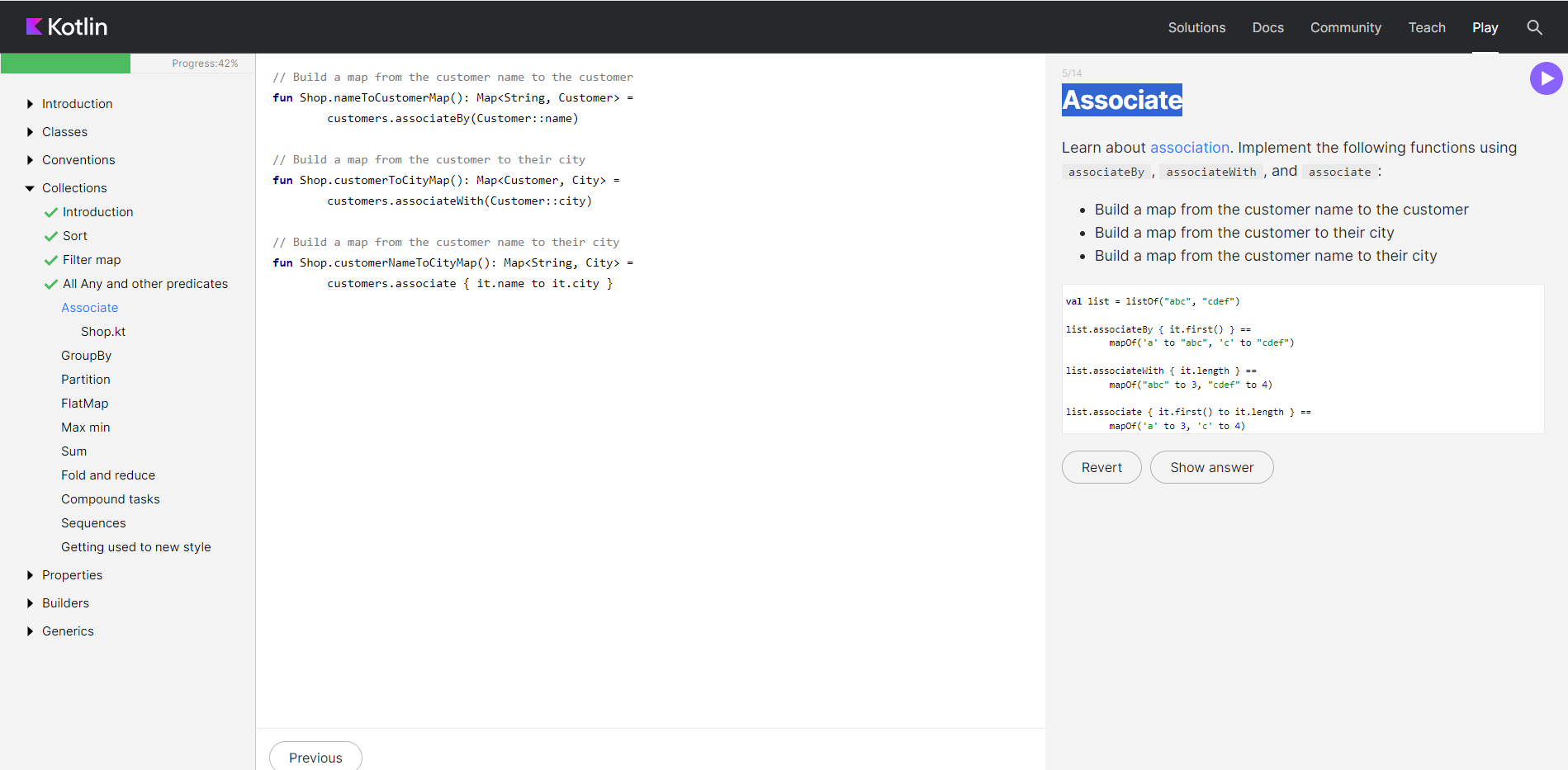
customers.count { it.city == city }

// Return a customer who lives in a given city, or null if there is none

fun Shop.findCustomerFrom(city: City): Customer? =

customers.find { it.city == city }

1. Задание «Associate»



// Build a map from the customer name to the customer

fun Shop.nameToCustomerMap(): Map<String, Customer> =

customers.associateBy(Customer::name)

// Build a map from the customer to their city

fun Shop.customerToCityMap(): Map<Customer, City> =

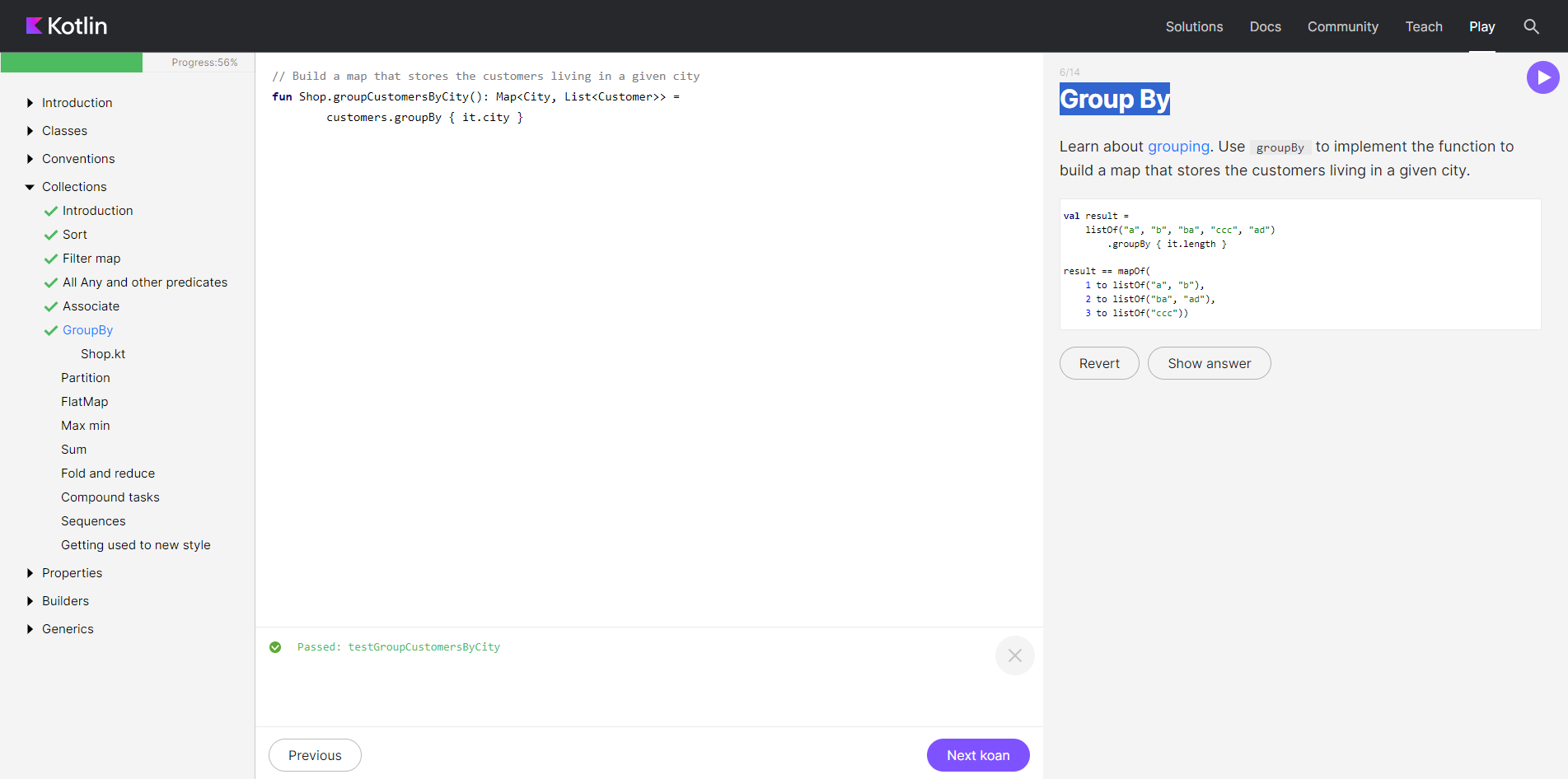
customers.associateWith(Customer::city)

// Build a map from the customer name to their city

fun Shop.customerNameToCityMap(): Map<String, City> =

customers.associate { it.name to it.city }

1. Задание «Group By»

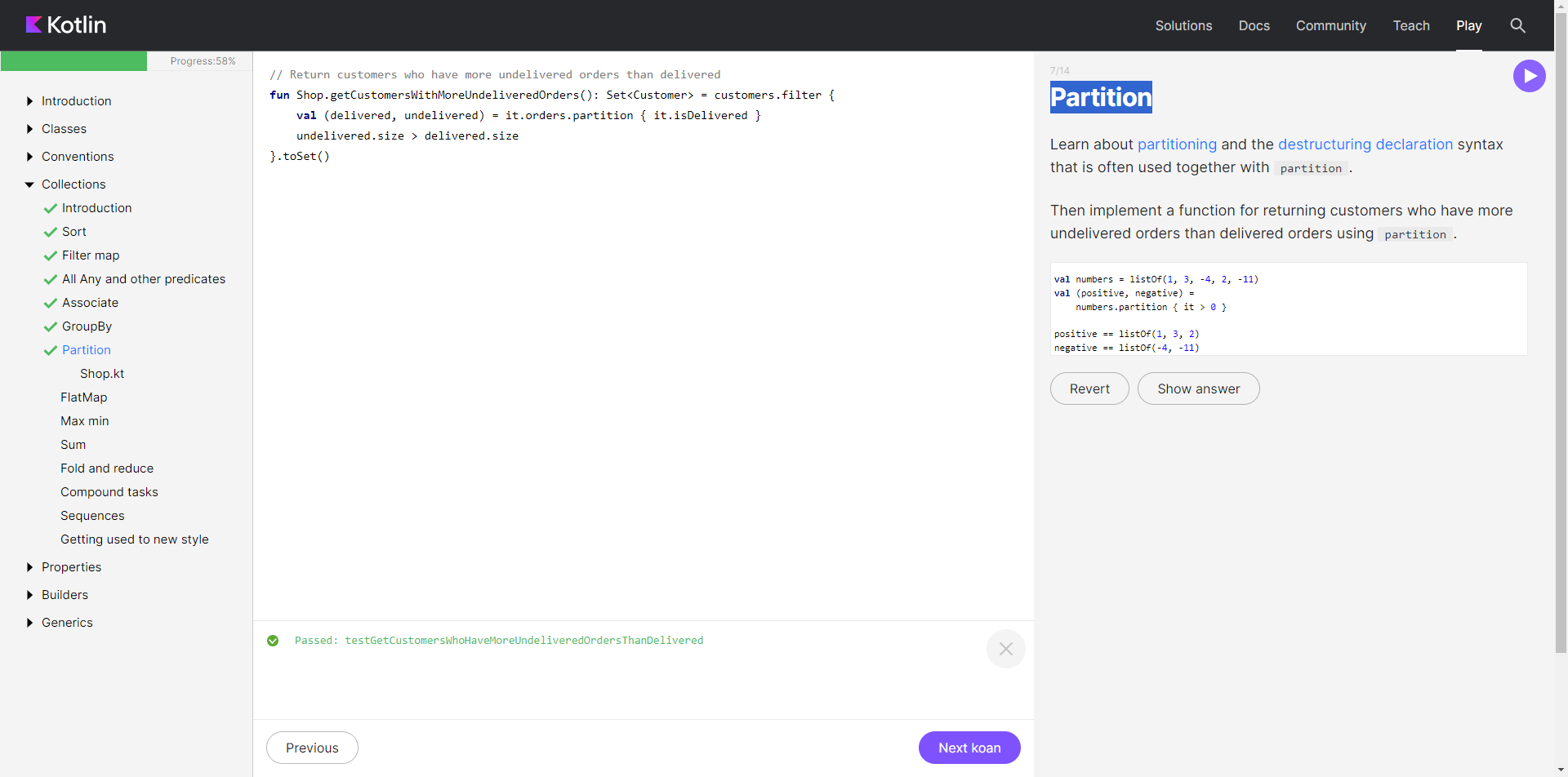


// Build a map that stores the customers living in a given city

fun Shop.groupCustomersByCity(): Map<City, List<Customer>> =

customers.groupBy { it.city }

1. Задание «Partition»



// Return customers who have more undelivered orders than delivered

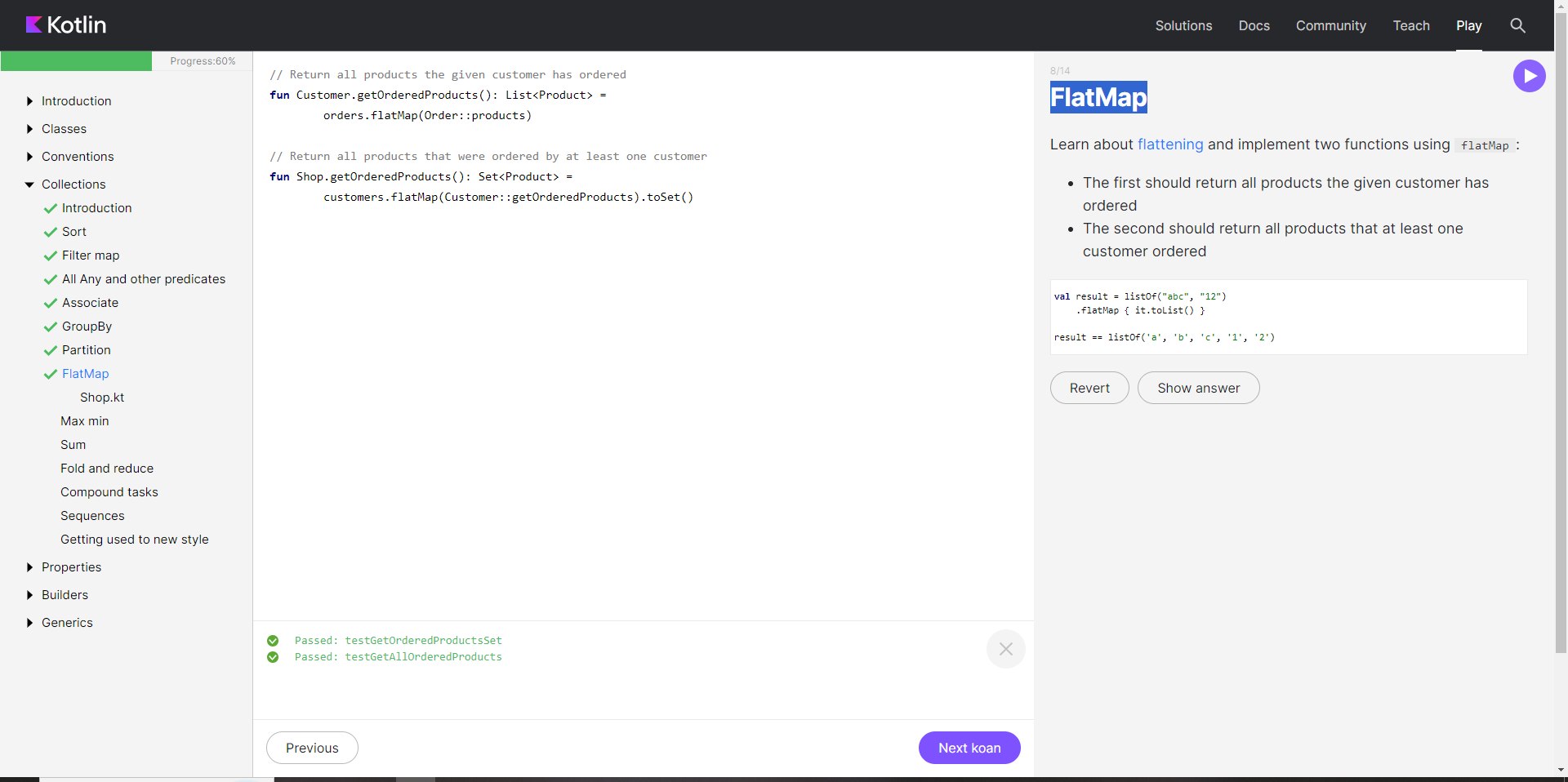
fun Shop.getCustomersWithMoreUndeliveredOrders(): Set<Customer> = customers.filter {

val (delivered, undelivered) = it.orders.partition { it.isDelivered }

undelivered.size > delivered.size

}.toSet()

1. Задание «FlatMap»



// Return all products the given customer has ordered

fun Customer.getOrderedProducts(): List<Product> =

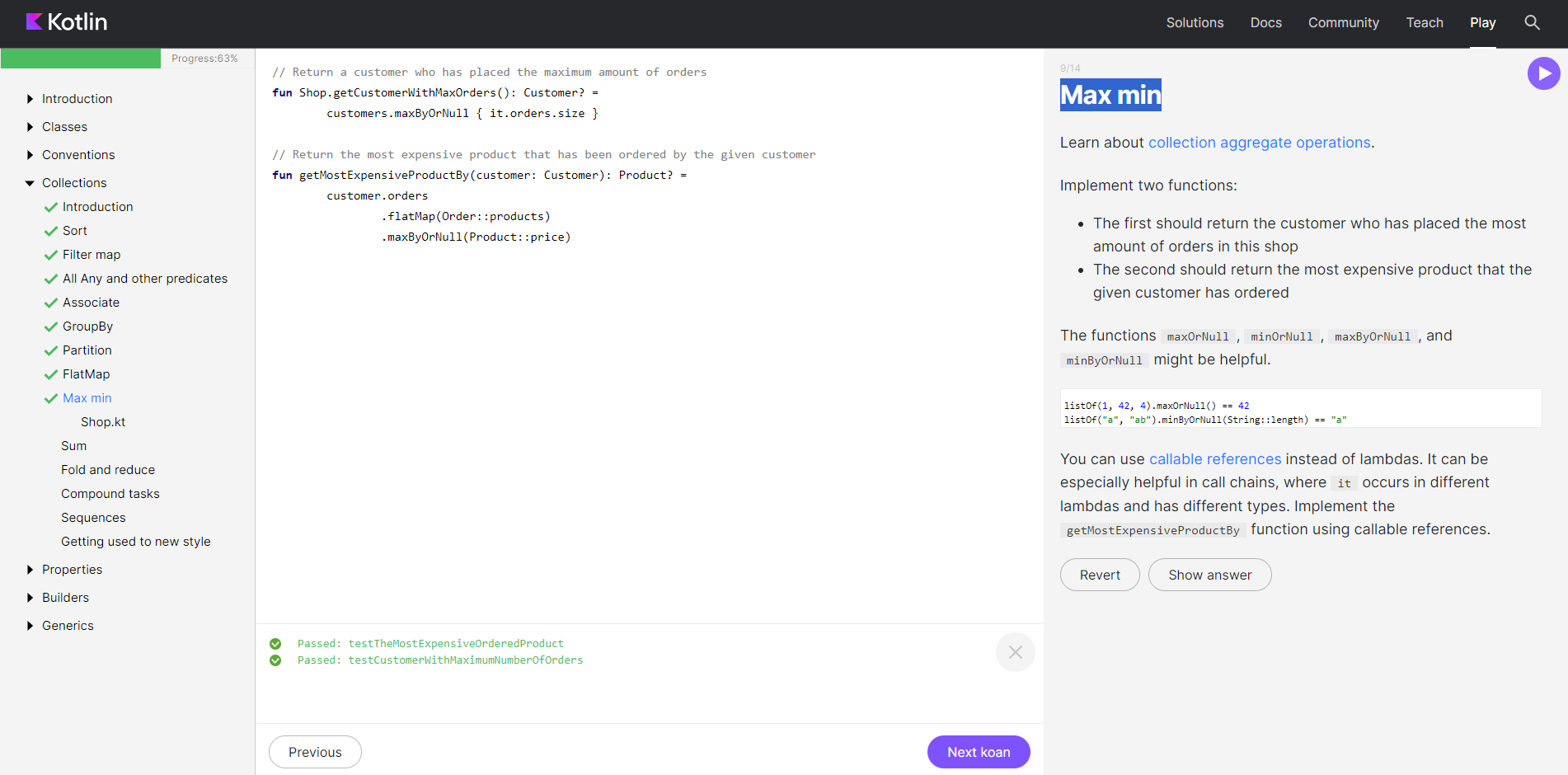
orders.flatMap(Order::products)

// Return all products that were ordered by at least one customer

fun Shop.getOrderedProducts(): Set<Product> =

customers.flatMap(Customer::getOrderedProducts).toSet()

1. Задание «Max min»



// Return a customer who has placed the maximum amount of orders

fun Shop.getCustomerWithMaxOrders(): Customer? =

customers.maxByOrNull { it.orders.size }

// Return the most expensive product that has been ordered by the given customer

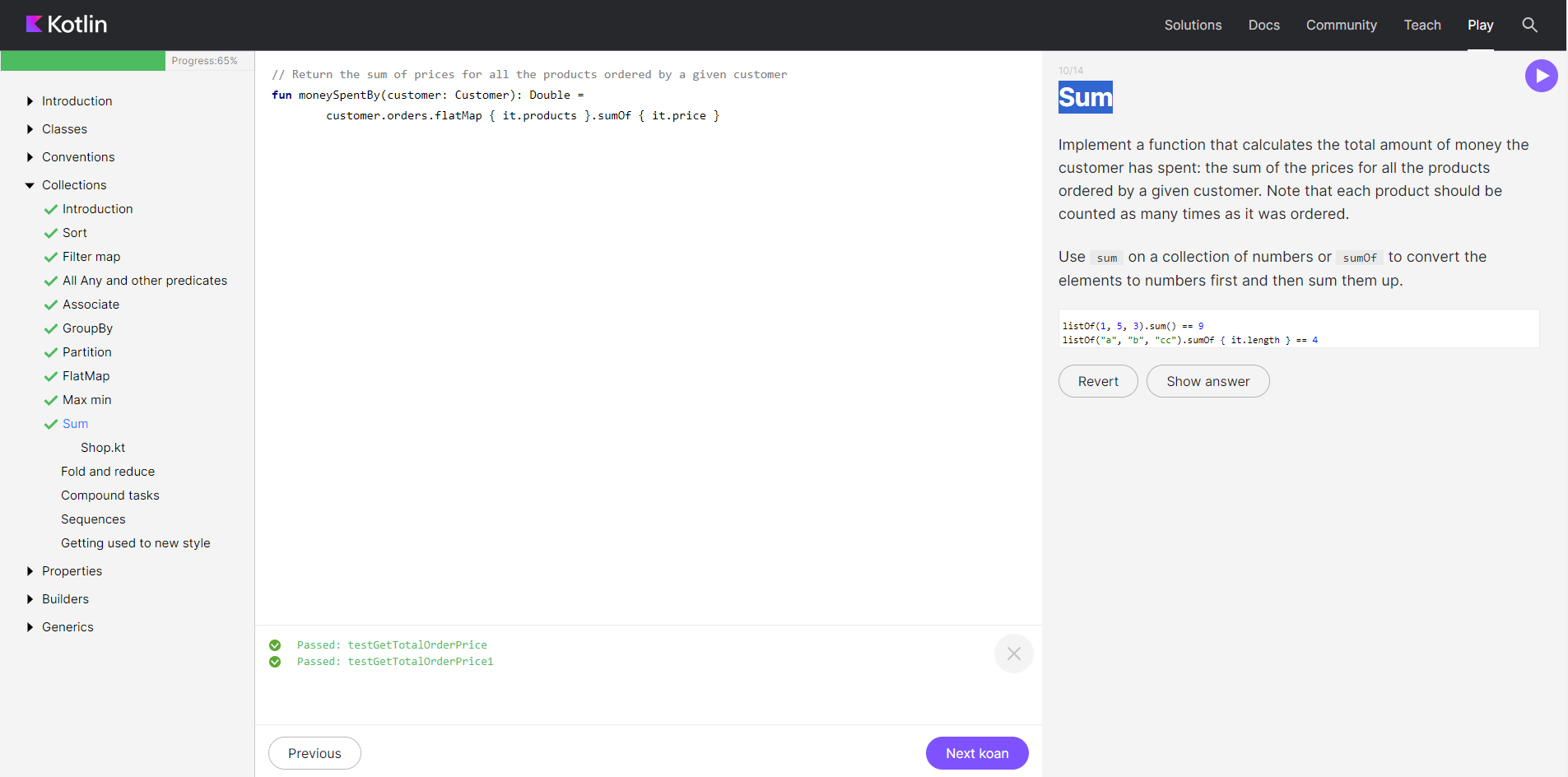
fun getMostExpensiveProductBy(customer: Customer): Product? =

customer.orders

.flatMap(Order::products)

.maxByOrNull(Product::price)

1. Задание «Sum»

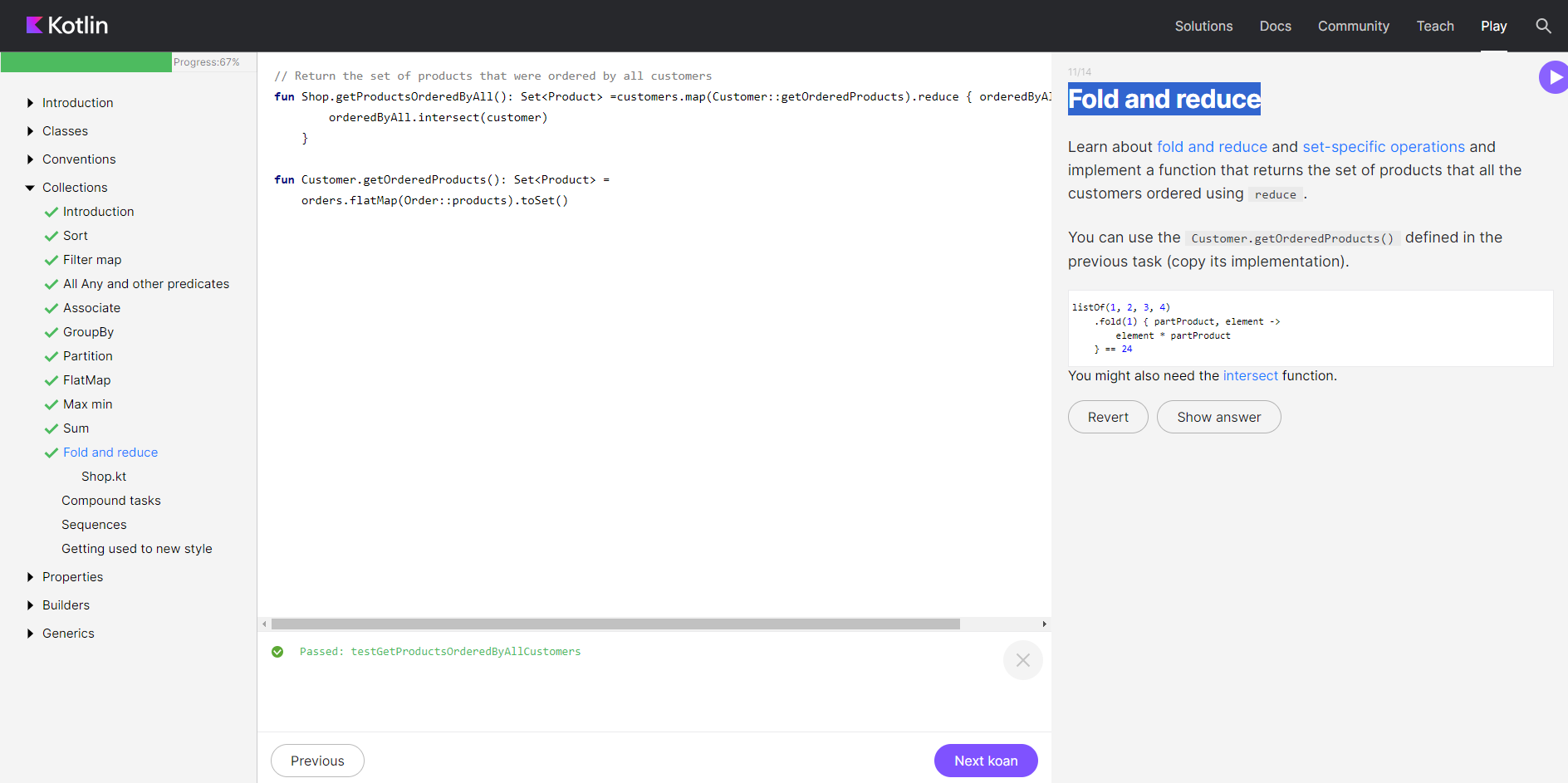


// Return the sum of prices for all the products ordered by a given customer

fun moneySpentBy(customer: Customer): Double =

customer.orders.flatMap { it.products }.sumOf { it.price }

1. Задание «Fold and reduce»



// Return the set of products that were ordered by all customers

fun Shop.getProductsOrderedByAll(): Set<Product> =customers.map(Customer::getOrderedProducts).reduce { orderedByAll, customer ->

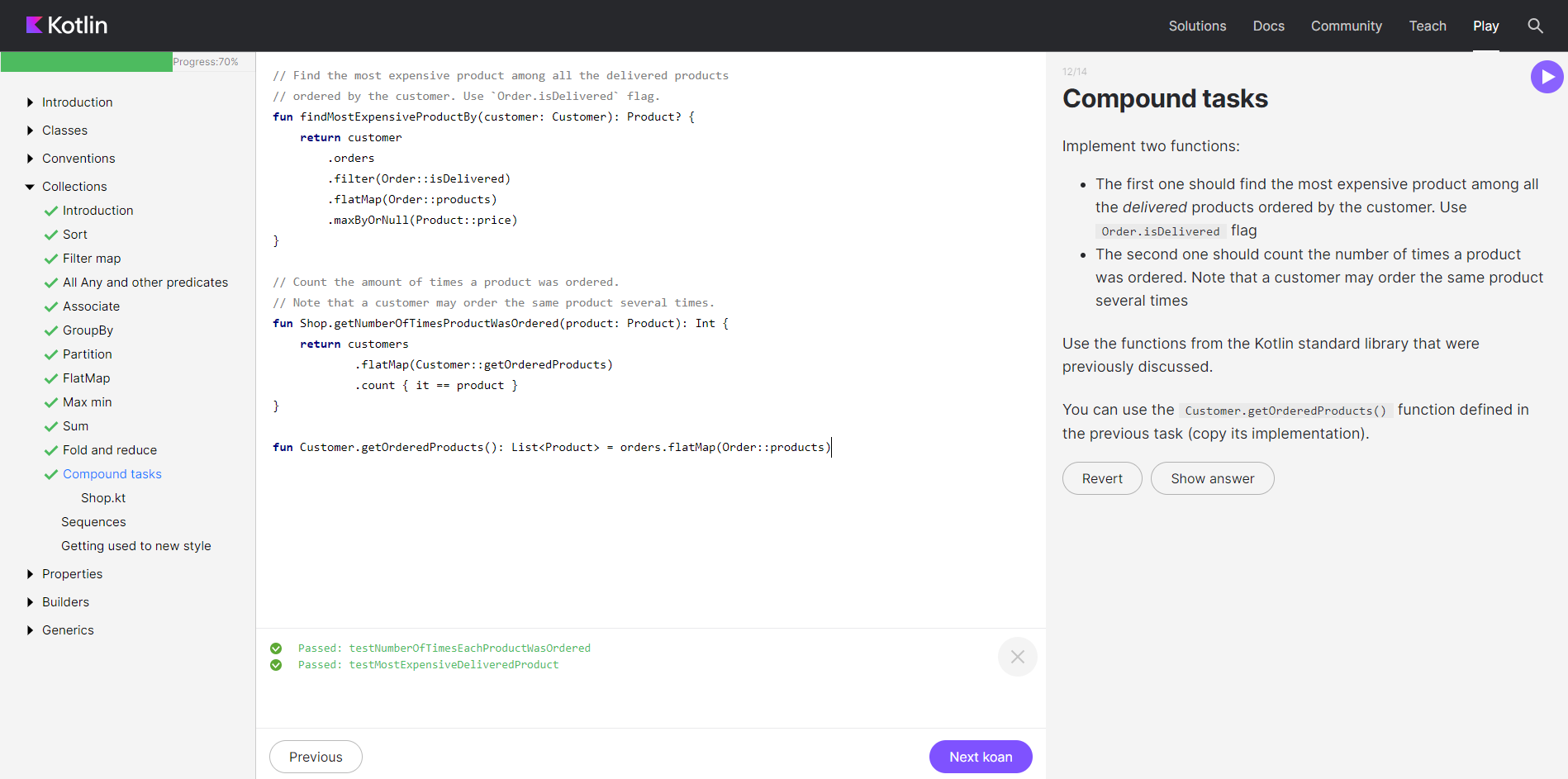
orderedByAll.intersect(customer)

}

fun Customer.getOrderedProducts(): Set<Product> =

orders.flatMap(Order::products).toSet()

1. Задание «Compound tasks»



// Find the most expensive product among all the delivered products

// ordered by the customer. Use `Order.isDelivered` flag.

fun findMostExpensiveProductBy(customer: Customer): Product? {

return customer

.orders

.filter(Order::isDelivered)

.flatMap(Order::products)

.maxByOrNull(Product::price)

}

// Count the amount of times a product was ordered.

// Note that a customer may order the same product several times.

fun Shop.getNumberOfTimesProductWasOrdered(product: Product): Int {

return customers

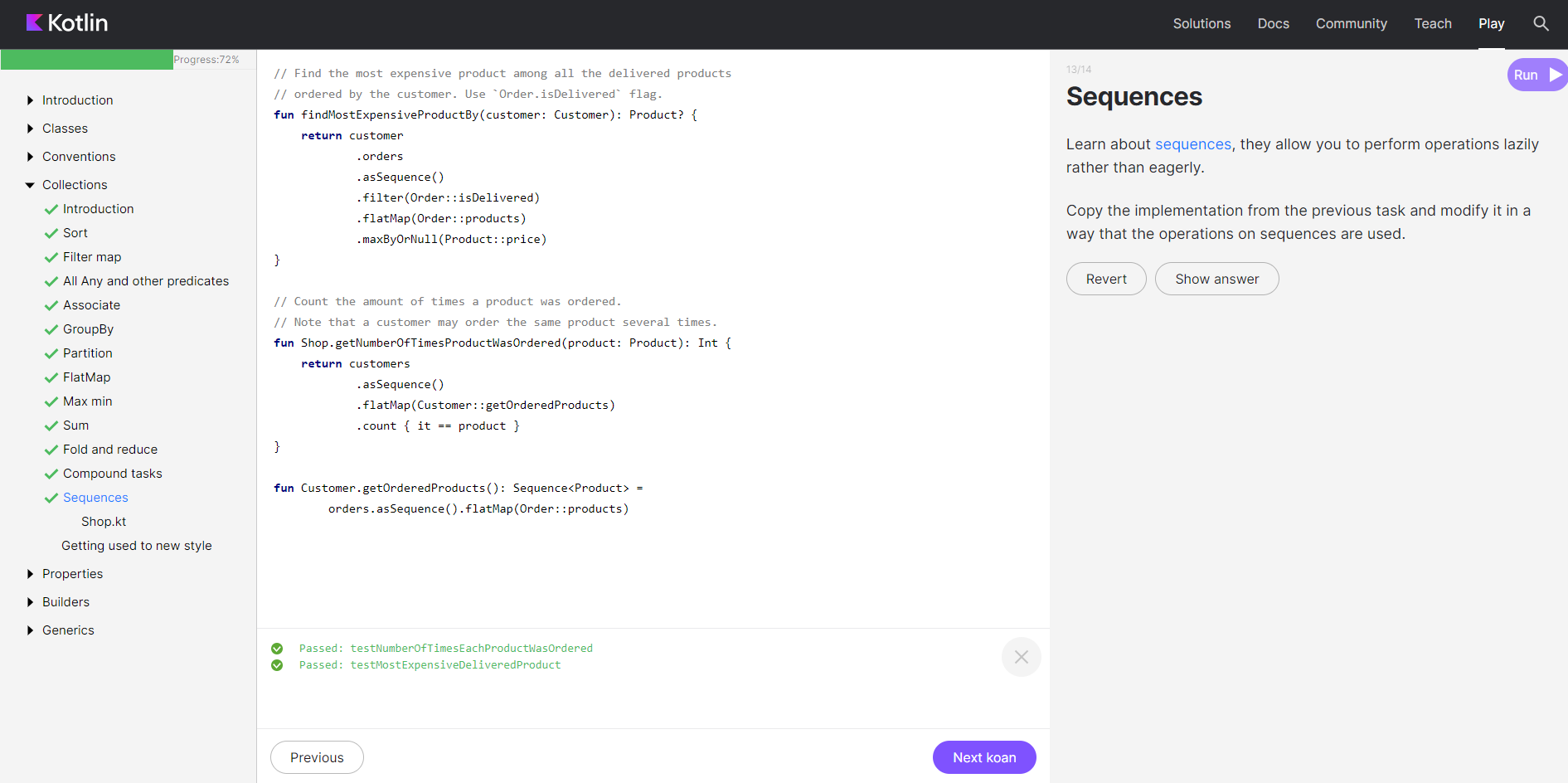
.flatMap(Customer::getOrderedProducts)

.count { it == product }

}

fun Customer.getOrderedProducts(): List<Product> = orders.flatMap(Order::products)

1. Задание «Sequences»



// Find the most expensive product among all the delivered products

// ordered by the customer. Use `Order.isDelivered` flag.

fun findMostExpensiveProductBy(customer: Customer): Product? {

return customer

.orders

.asSequence()

.filter(Order::isDelivered)

.flatMap(Order::products)

.maxByOrNull(Product::price)

}

// Count the amount of times a product was ordered.

// Note that a customer may order the same product several times.

fun Shop.getNumberOfTimesProductWasOrdered(product: Product): Int {

return customers

.asSequence()

.flatMap(Customer::getOrderedProducts)

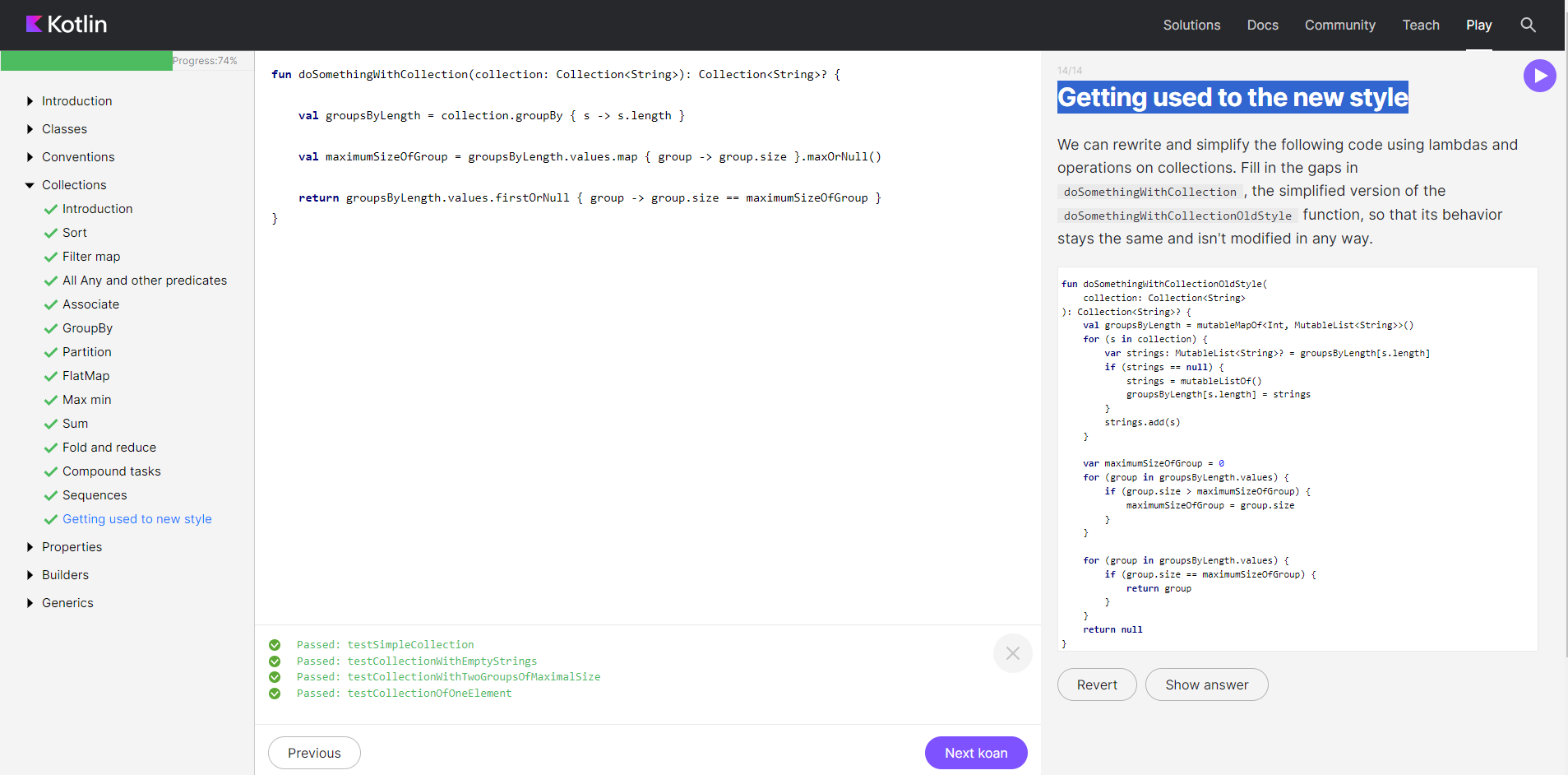
.count { it == product }

}

fun Customer.getOrderedProducts(): Sequence<Product> =

orders.asSequence().flatMap(Order::products)

1. Задание «Getting used to the new style»



fun doSomethingWithCollection(collection: Collection<String>): Collection<String>? {

val groupsByLength = collection.groupBy { s -> s.length }

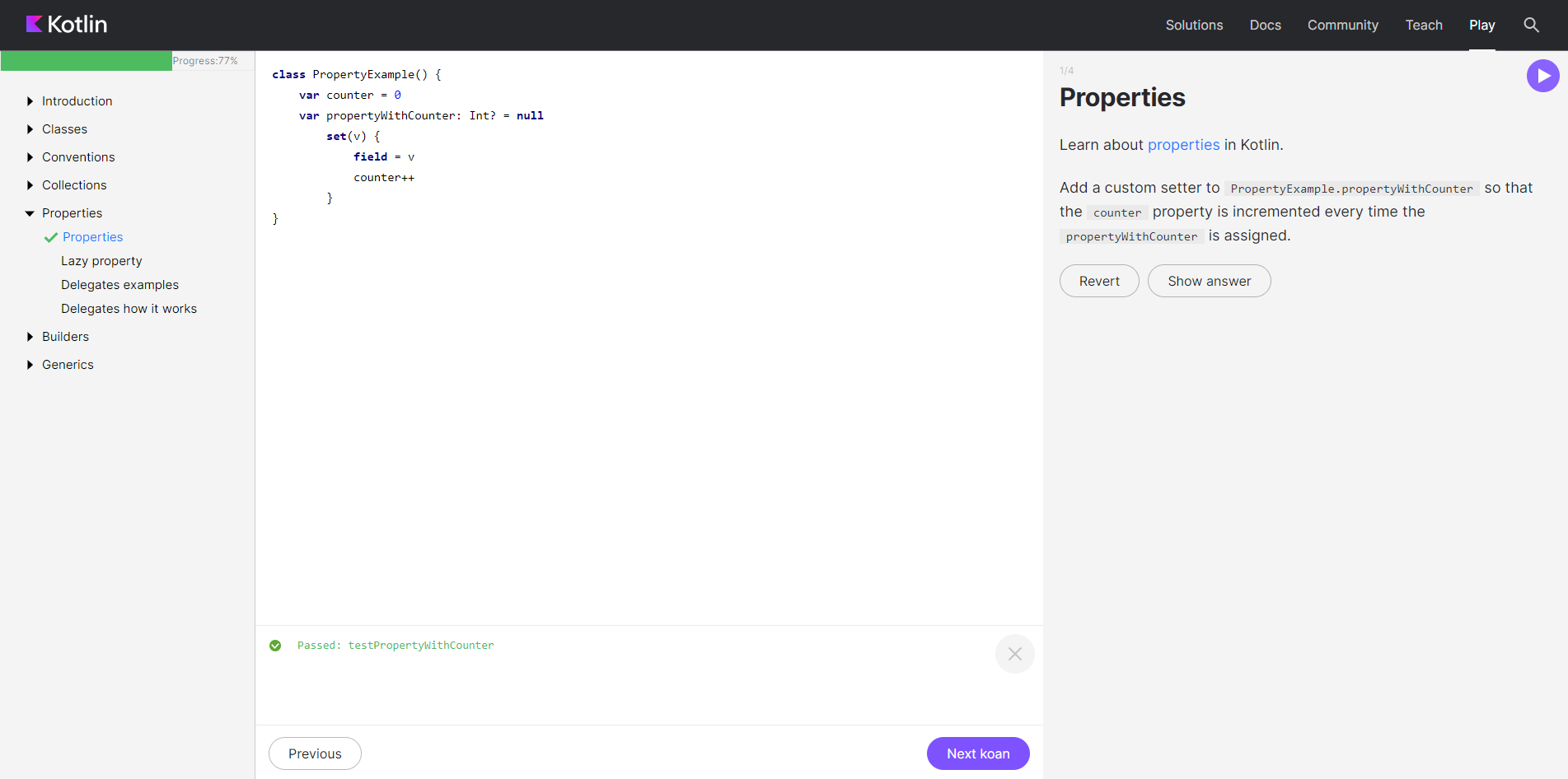
val maximumSizeOfGroup = groupsByLength.values.map { group -> group.size }.maxOrNull()

return groupsByLength.values.firstOrNull { group -> group.size == maximumSizeOfGroup }

}

**PROPERTIES**

1. Задание «Properties»



class PropertyExample() {

var counter = 0

var propertyWithCounter: Int? = null

set(v) {

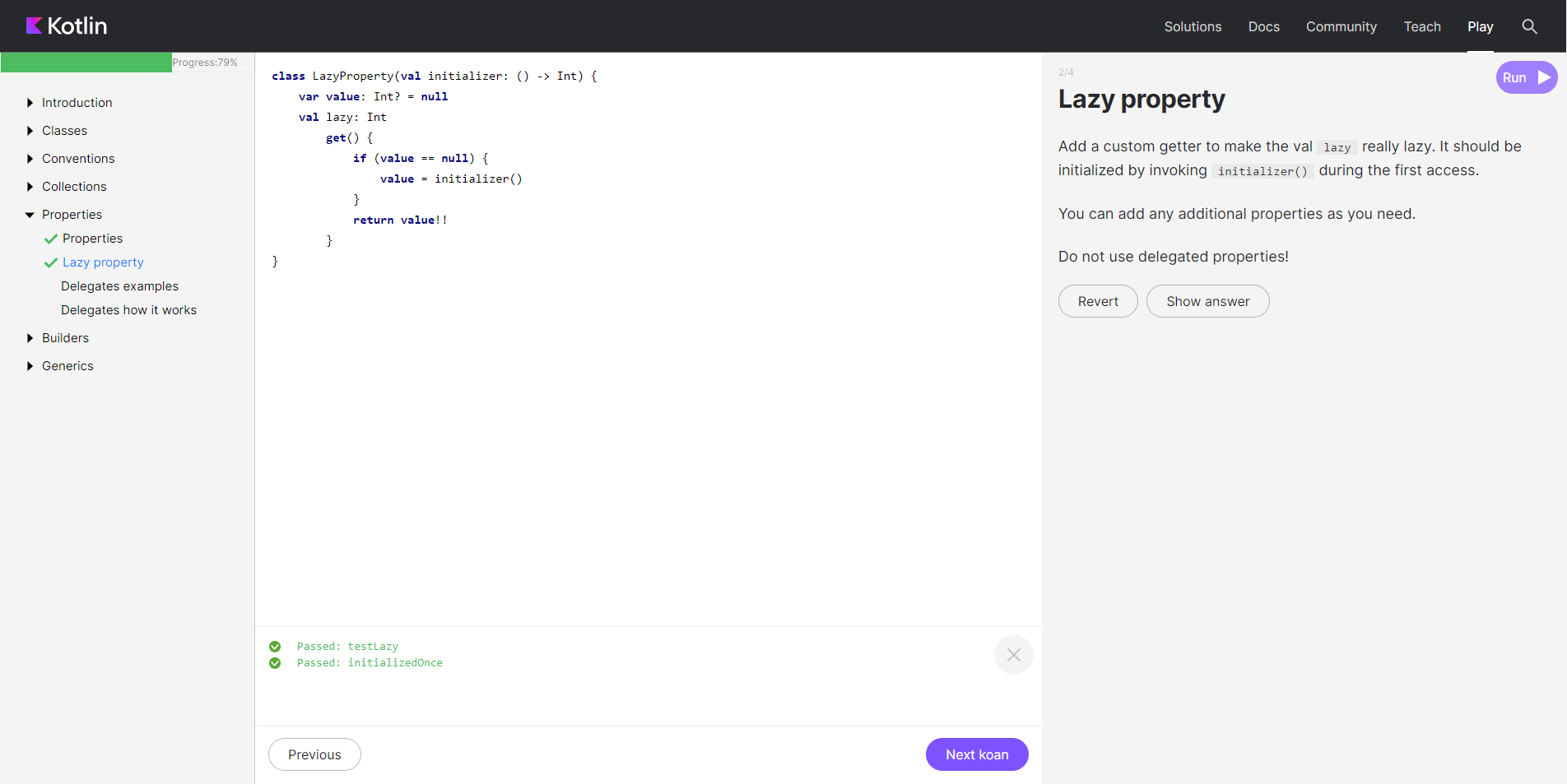
field = v

counter++

}

}

1. Задание «Lazy property»



class LazyProperty(val initializer: () -> Int) {

var value: Int? = null

val lazy: Int

get() {

if (value == null) {

value = initializer()

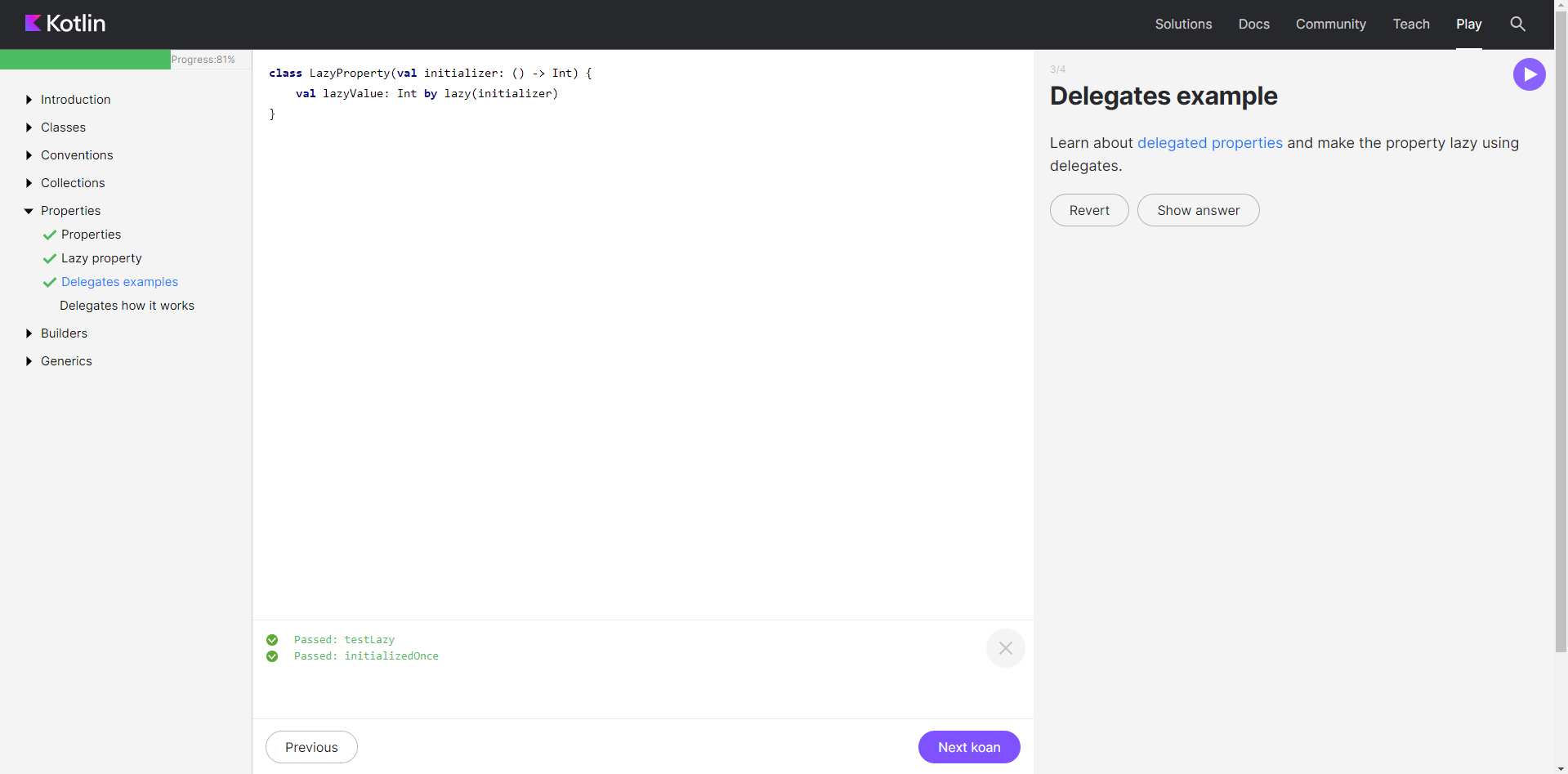
}

return value!!

}

}

1. Задание «Delegates example»

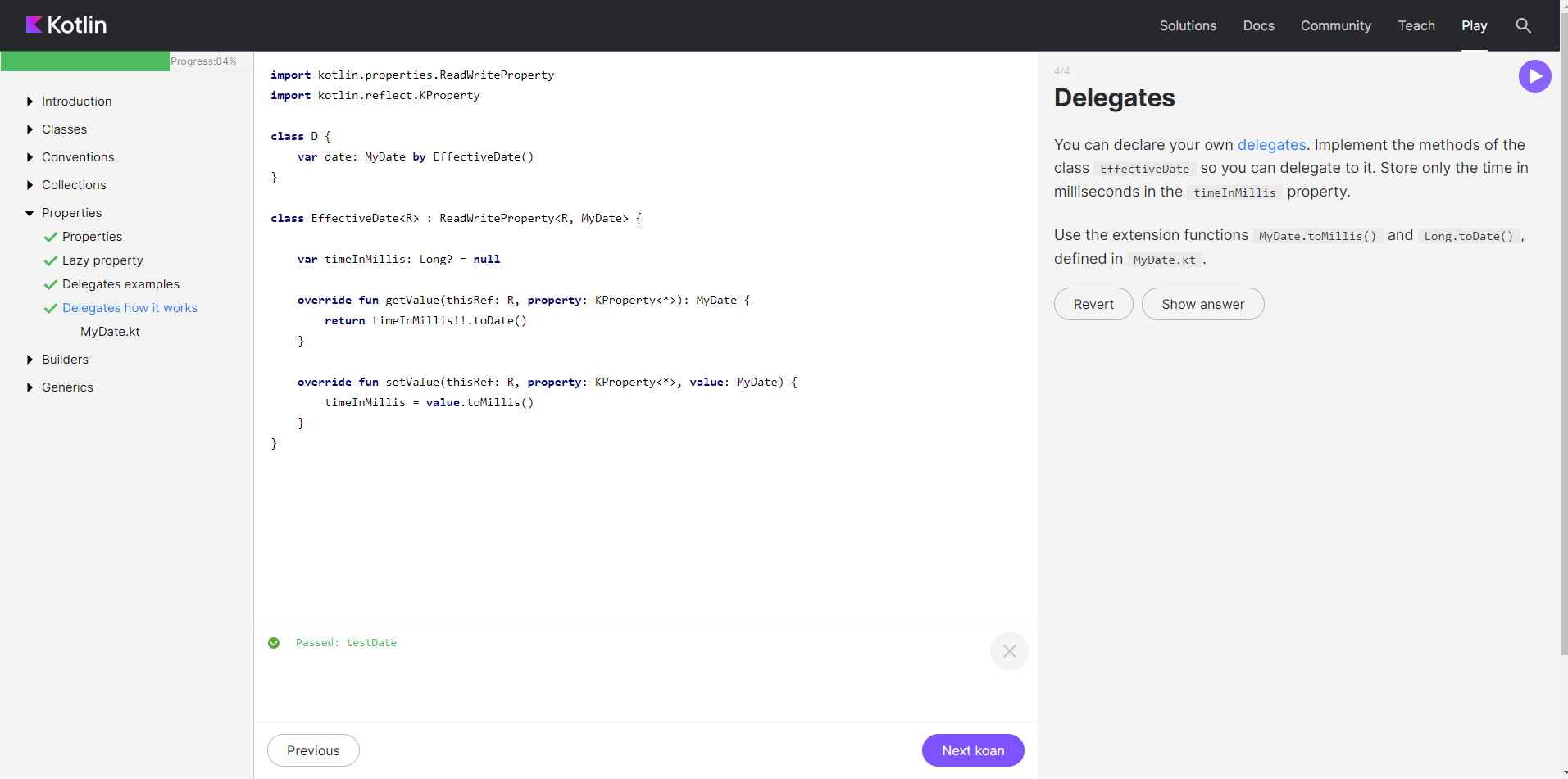


class LazyProperty(val initializer: () -> Int) {

val lazyValue: Int by lazy(initializer)

}

1. Задание «Delegates»



import kotlin.properties.ReadWriteProperty

import kotlin.reflect.KProperty

class D {

var date: MyDate by EffectiveDate()

}

class EffectiveDate<R> : ReadWriteProperty<R, MyDate> {

var timeInMillis: Long? = null

override fun getValue(thisRef: R, property: KProperty<\*>): MyDate {

return timeInMillis!!.toDate()

}

override fun setValue(thisRef: R, property: KProperty<\*>, value: MyDate) {

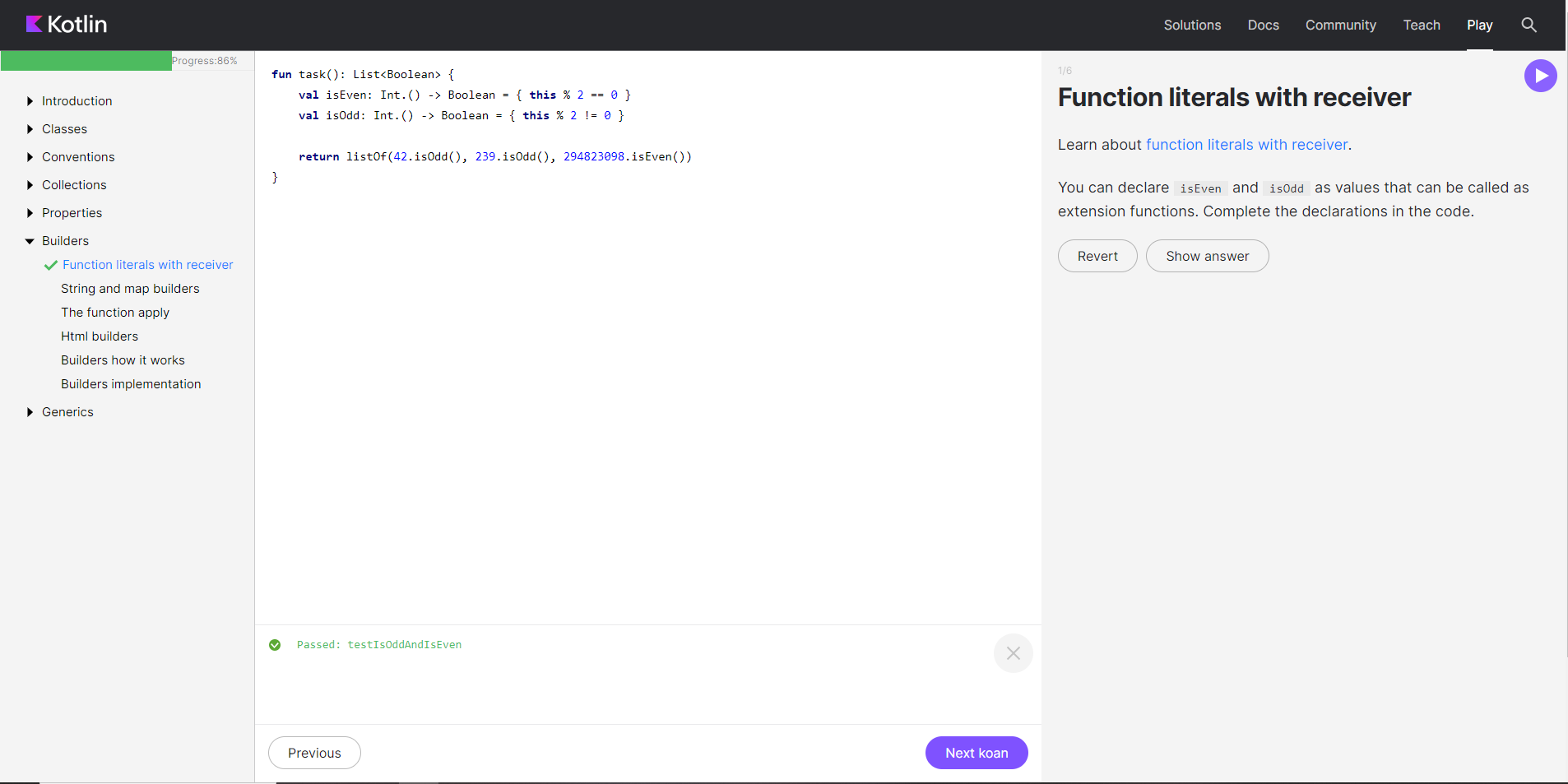
timeInMillis = value.toMillis()

}

}

**BUILDERS**

1. Задание «Function literals with receiver»



fun task(): List<Boolean> {

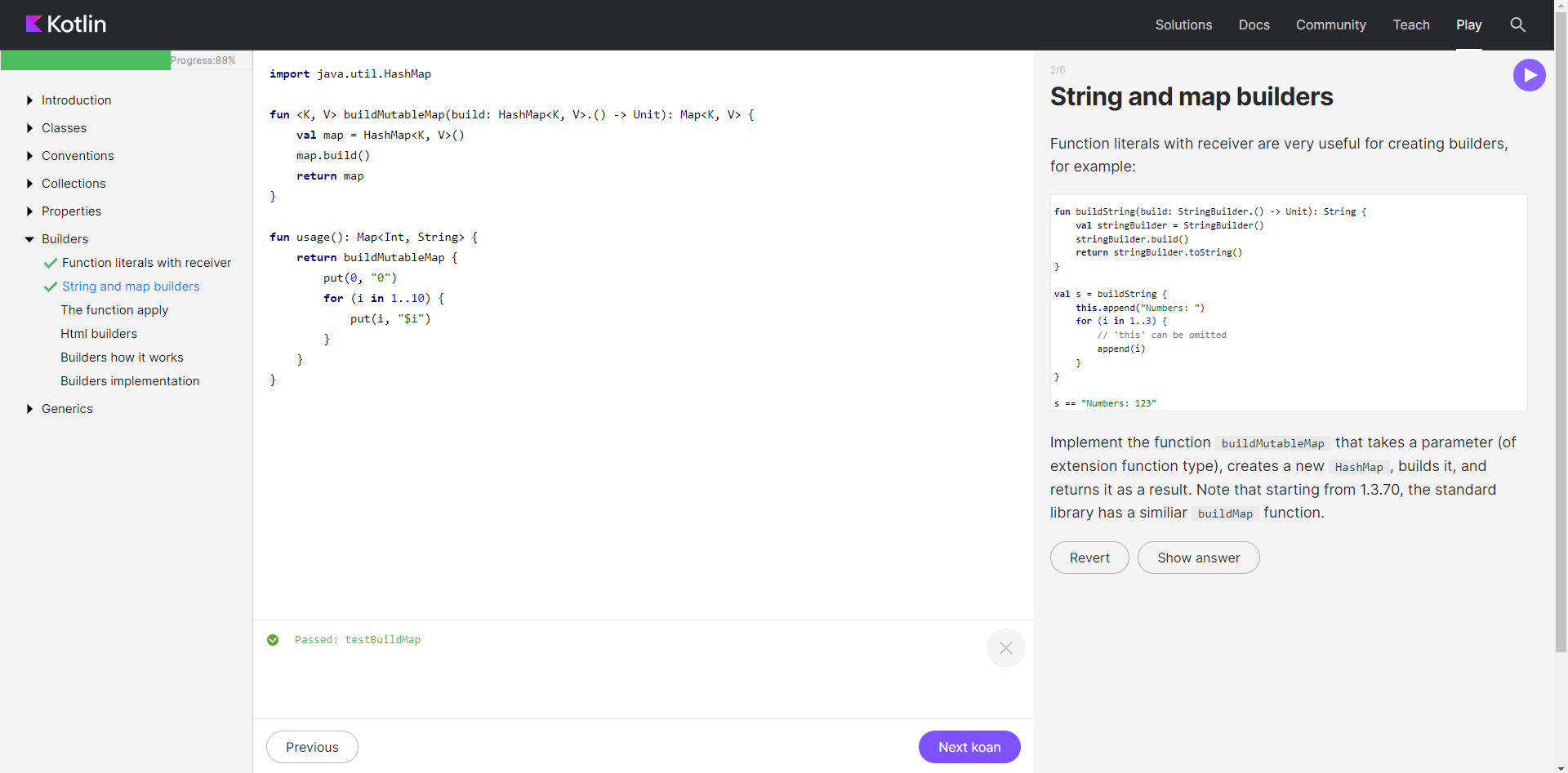
val isEven: Int.() -> Boolean = { this % 2 == 0 }

val isOdd: Int.() -> Boolean = { this % 2 != 0 }

return listOf(42.isOdd(), 239.isOdd(), 294823098.isEven())

}

1. Задание «String and map builders»



import java.util.HashMap

fun <K, V> buildMutableMap(build: HashMap<K, V>.() -> Unit): Map<K, V> {

val map = HashMap<K, V>()

map.build()

return map

}

fun usage(): Map<Int, String> {

return buildMutableMap {

put(0, "0")

for (i in 1..10) {

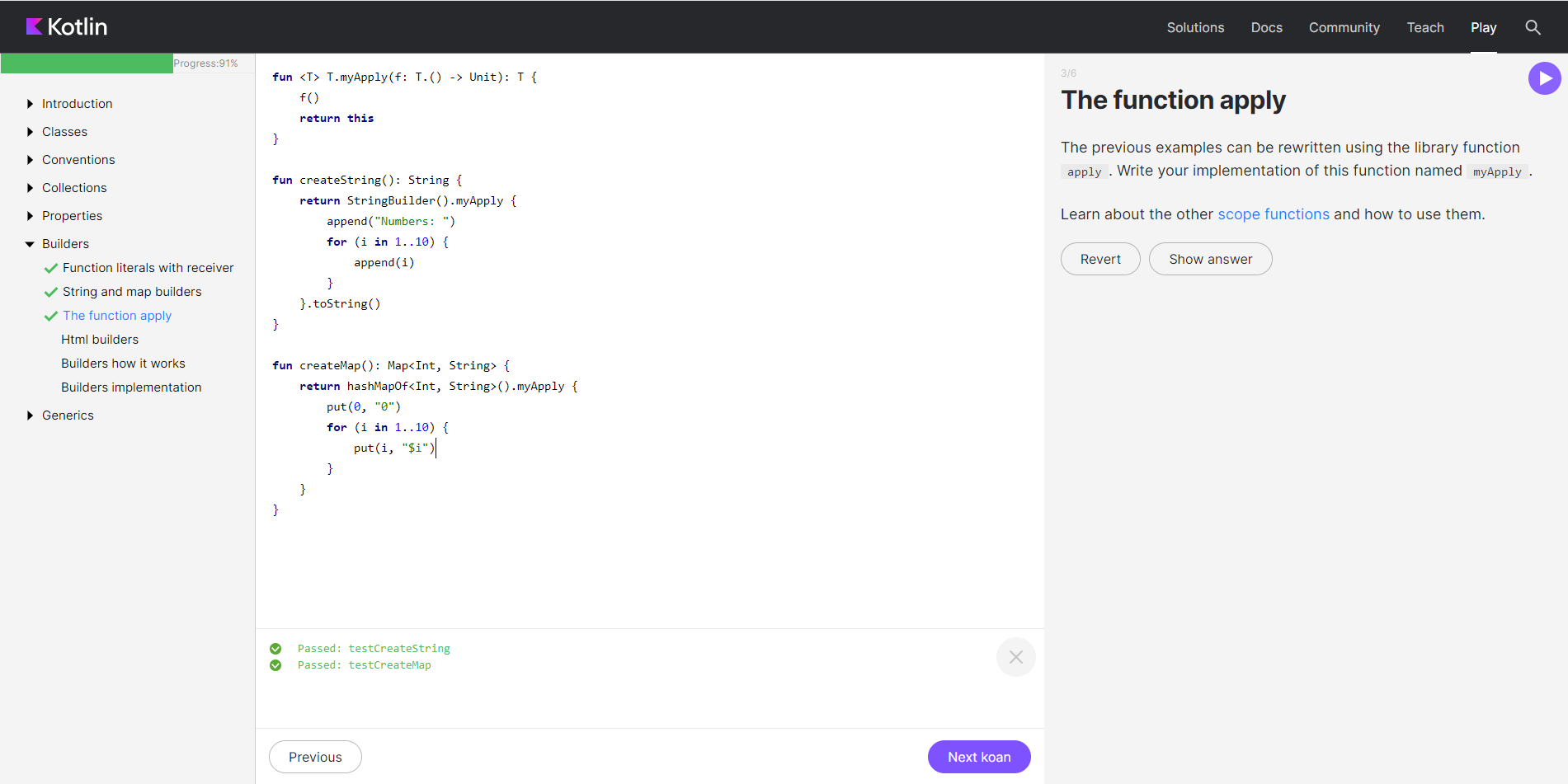
put(i, "$i")

}

}

}

1. Задание «The function apply»



fun <T> T.myApply(f: T.() -> Unit): T {

f()

return this

}

fun createString(): String {

return StringBuilder().myApply {

append("Numbers: ")

for (i in 1..10) {

append(i)

}

}.toString()

}

fun createMap(): Map<Int, String> {

return hashMapOf<Int, String>().myApply {

put(0, "0")

for (i in 1..10) {

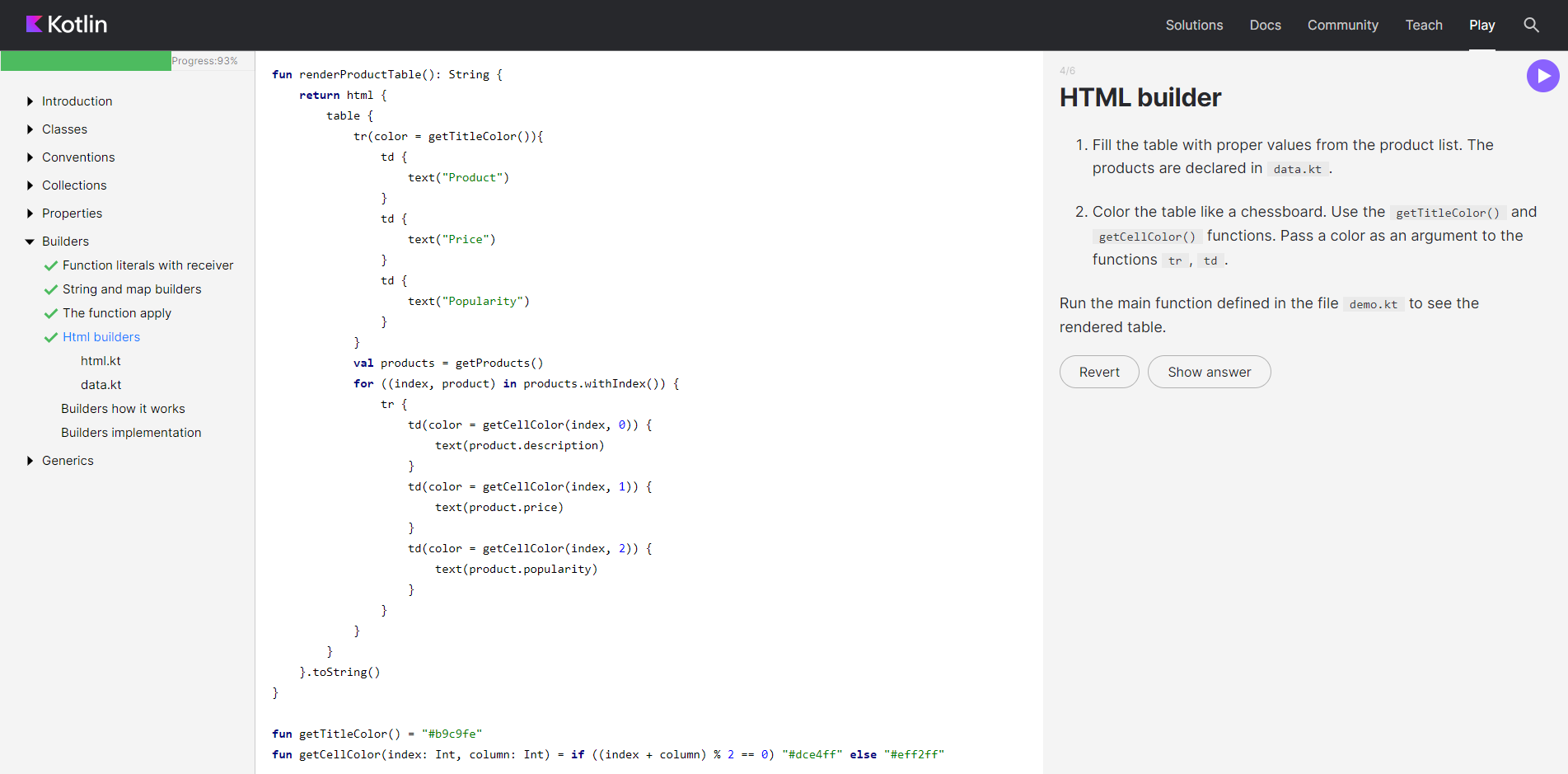
put(i, "$i")

}

}

}

1. Задание «HTML builder»



fun renderProductTable(): String {

return html {

table {

tr(color = getTitleColor()){

td {

text("Product")

}

td {

text("Price")

}

td {

text("Popularity")

}

}

val products = getProducts()

for ((index, product) in products.withIndex()) {

tr {

td(color = getCellColor(index, 0)) {

text(product.description)

}

td(color = getCellColor(index, 1)) {

text(product.price)

}

td(color = getCellColor(index, 2)) {

text(product.popularity)

}

}

}

}

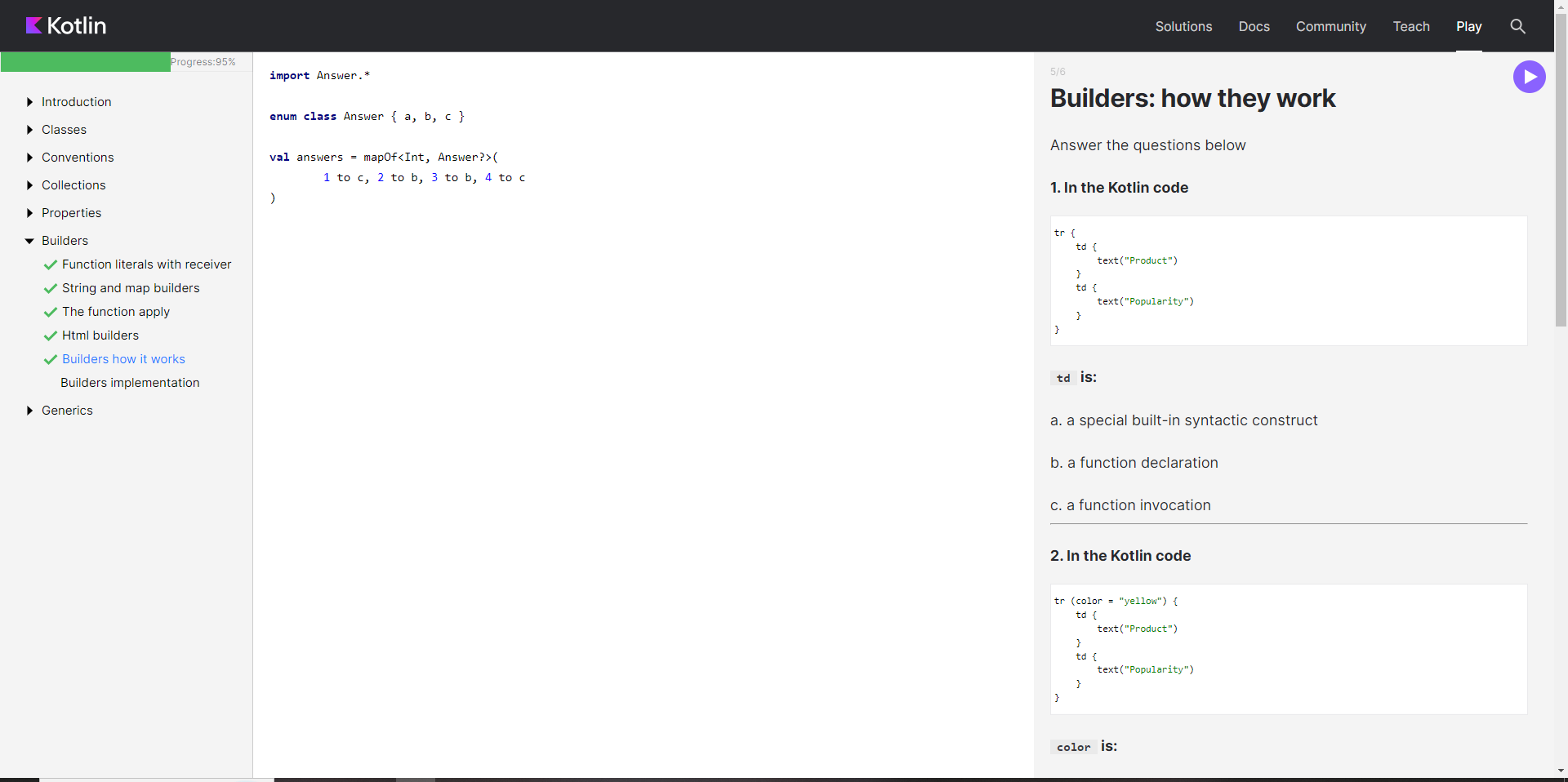
}.toString()

}

fun getTitleColor() = "#b9c9fe"

fun getCellColor(index: Int, column: Int) = if ((index + column) % 2 == 0) "#dce4ff" else "#eff2ff"

1. Задание «Builders: how they work»



import Answer.\*

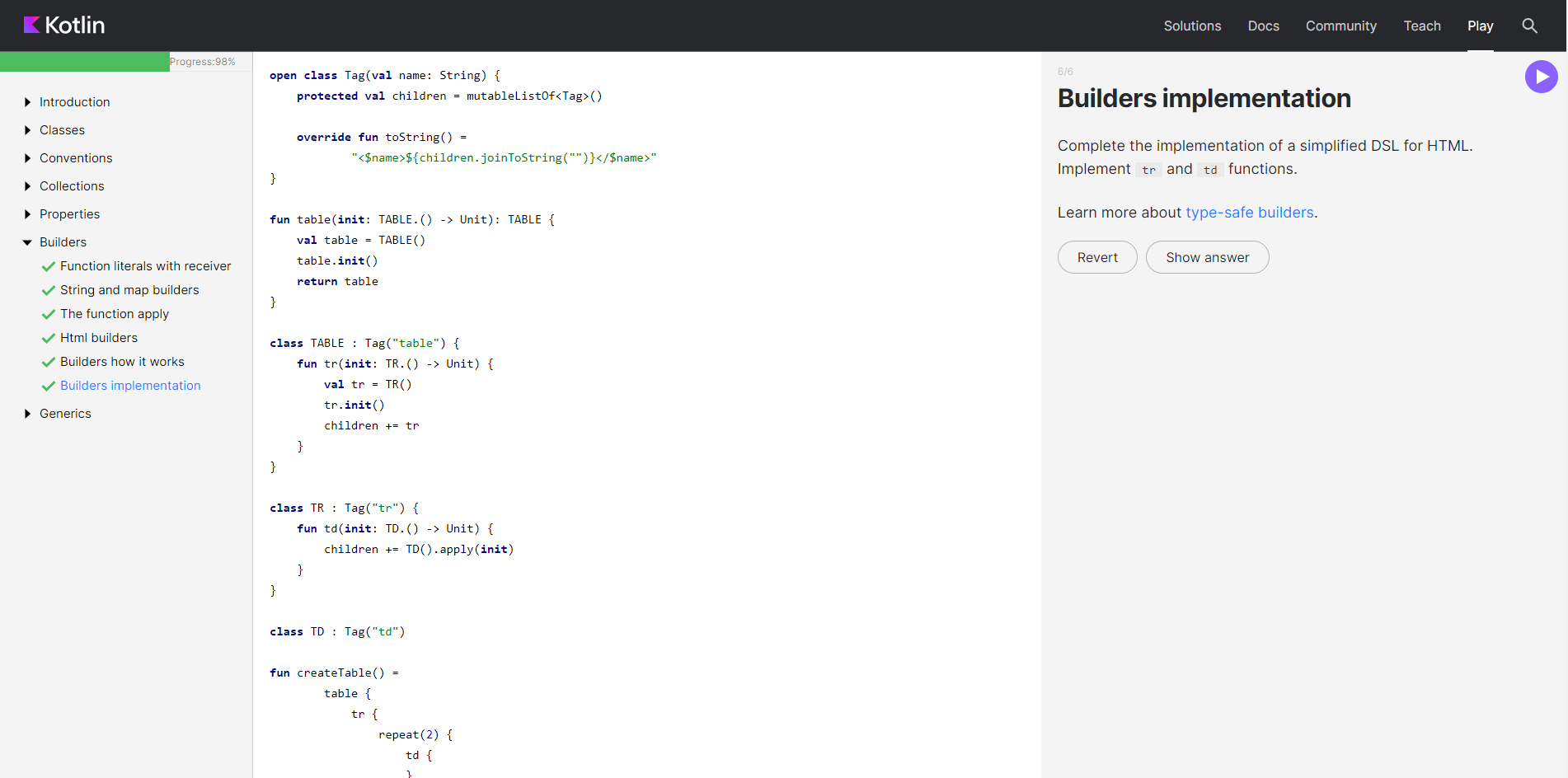
enum class Answer { a, b, c }

val answers = mapOf<Int, Answer?>(

1 to c, 2 to b, 3 to b, 4 to c

)

1. Задание «Builders implementation»



open class Tag(val name: String) {

protected val children = mutableListOf<Tag>()

override fun toString() =

"<$name>${children.joinToString("")}</$name>"

}

fun table(init: TABLE.() -> Unit): TABLE {

val table = TABLE()

table.init()

return table

}

class TABLE : Tag("table") {

fun tr(init: TR.() -> Unit) {

val tr = TR()

tr.init()

children += tr

}

}

class TR : Tag("tr") {

fun td(init: TD.() -> Unit) {

children += TD().apply(init)

}

}

class TD : Tag("td")

fun createTable() =

table {

tr {

repeat(2) {

td {

}

}

}

}

fun main() {

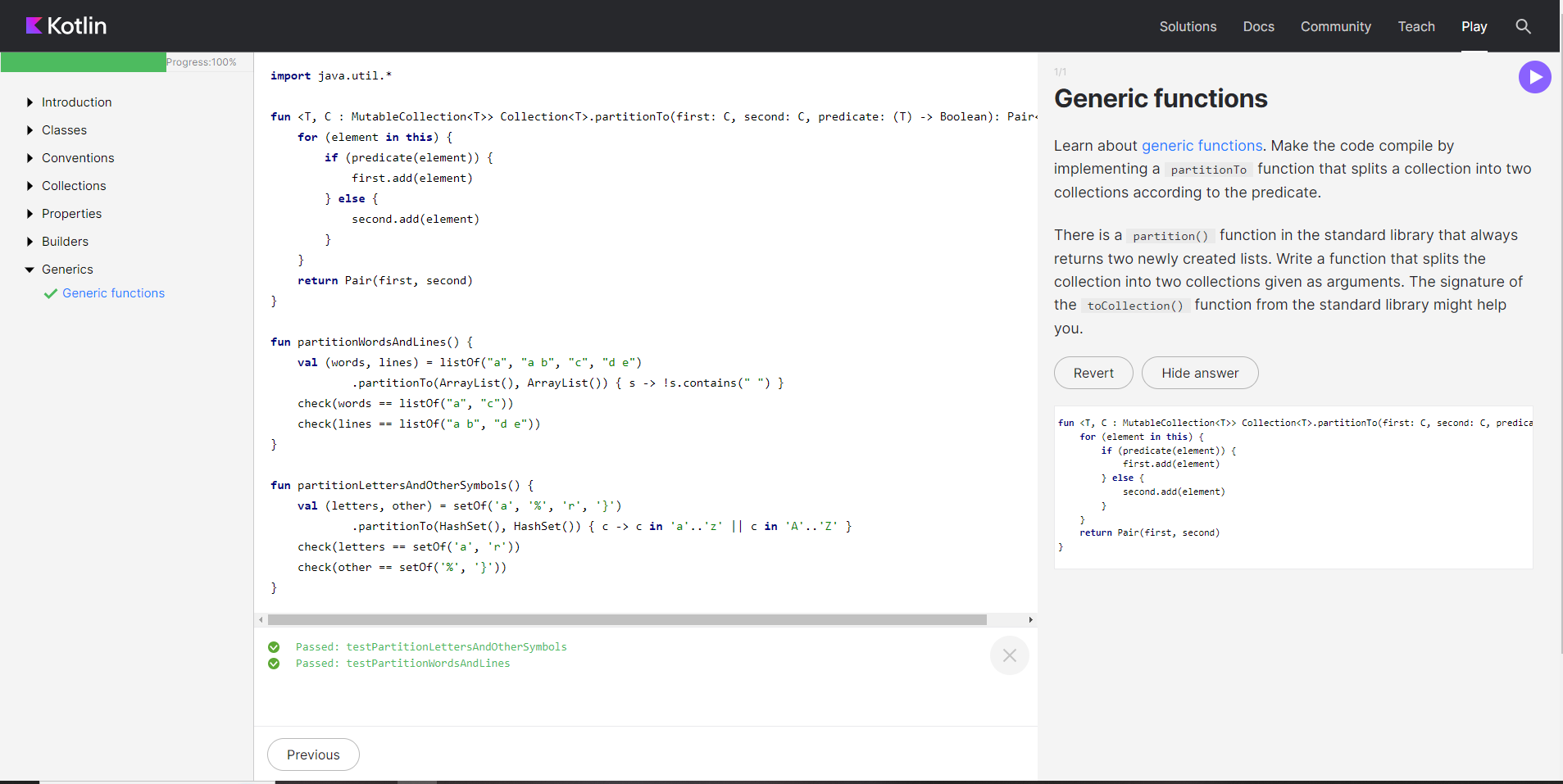
println(createTable())

//<table><tr><td></td><td></td></tr></table>

}

**GENERICS**

1. Задание «Generic functions»



import java.util.\*

fun <T, C : MutableCollection<T>> Collection<T>.partitionTo(first: C, second: C, predicate: (T) -> Boolean): Pair<C, C> {

for (element in this) {

if (predicate(element)) {

first.add(element)

} else {

second.add(element)

}

}

return Pair(first, second)

}

fun partitionWordsAndLines() {

val (words, lines) = listOf("a", "a b", "c", "d e")

.partitionTo(ArrayList(), ArrayList()) { s -> !s.contains(" ") }

check(words == listOf("a", "c"))

check(lines == listOf("a b", "d e"))

}

fun partitionLettersAndOtherSymbols() {

val (letters, other) = setOf('a', '%', 'r', '}')

.partitionTo(HashSet(), HashSet()) { c -> c in 'a'..'z' || c in 'A'..'Z' }

check(letters == setOf('a', 'r'))

check(other == setOf('%', '}'))

}