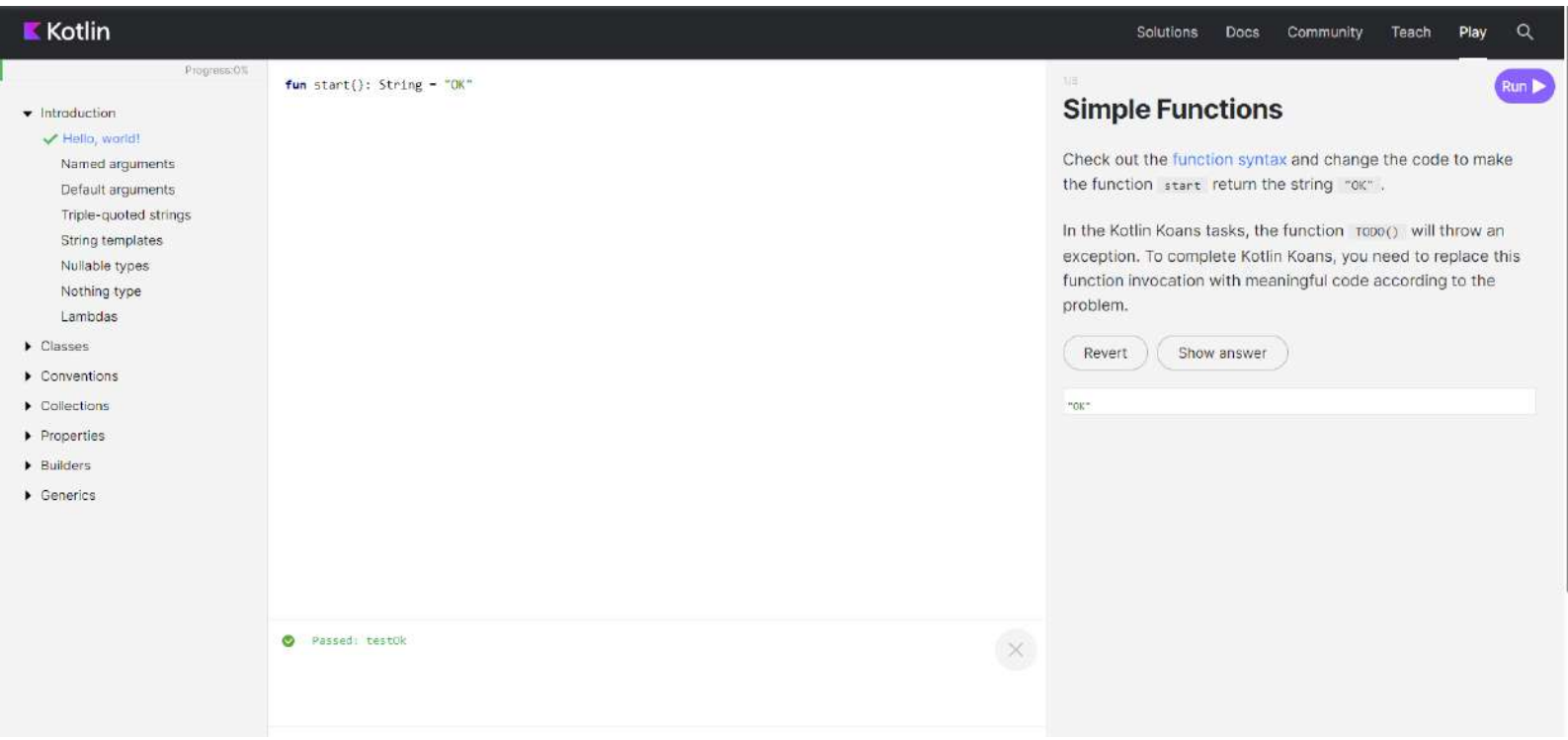
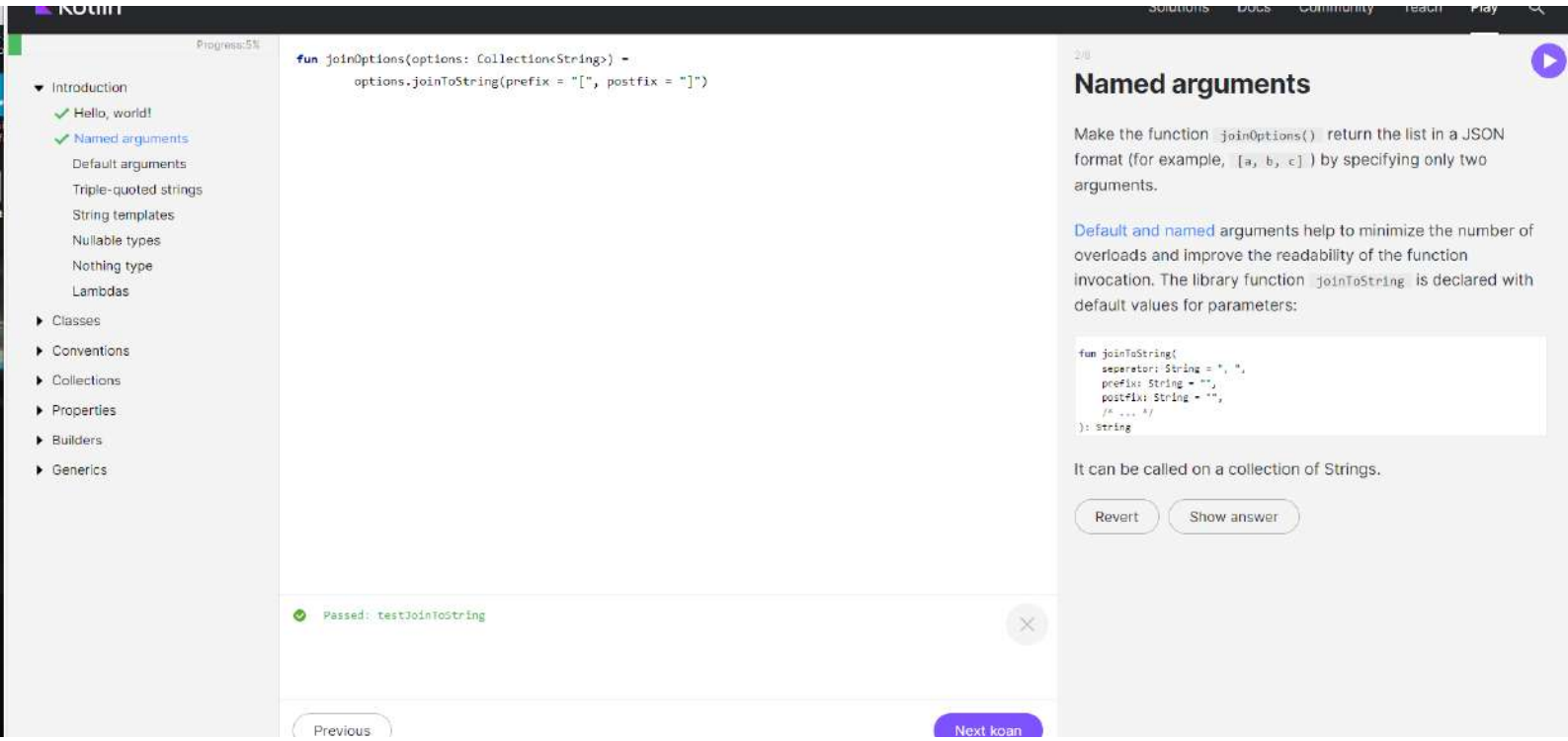
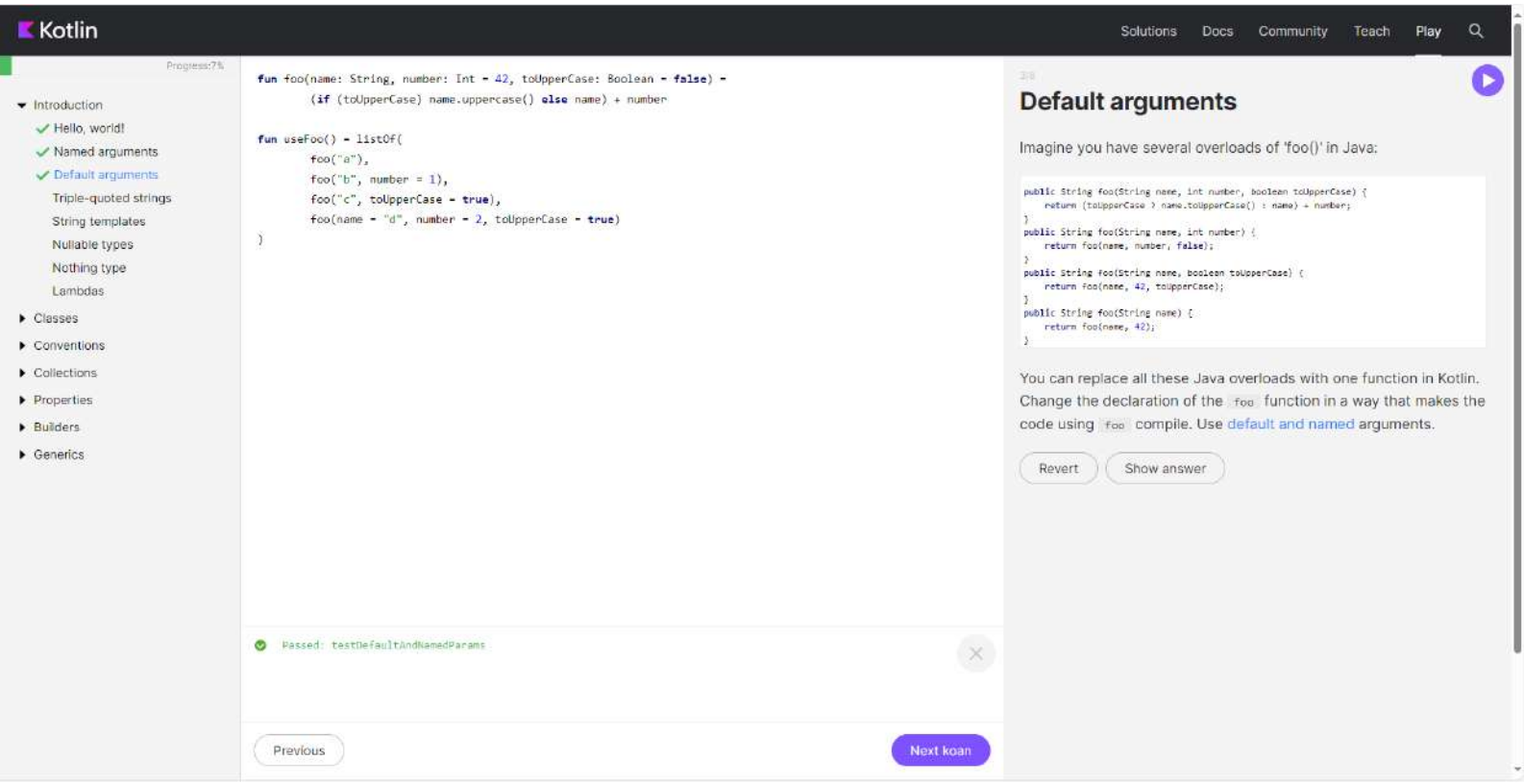
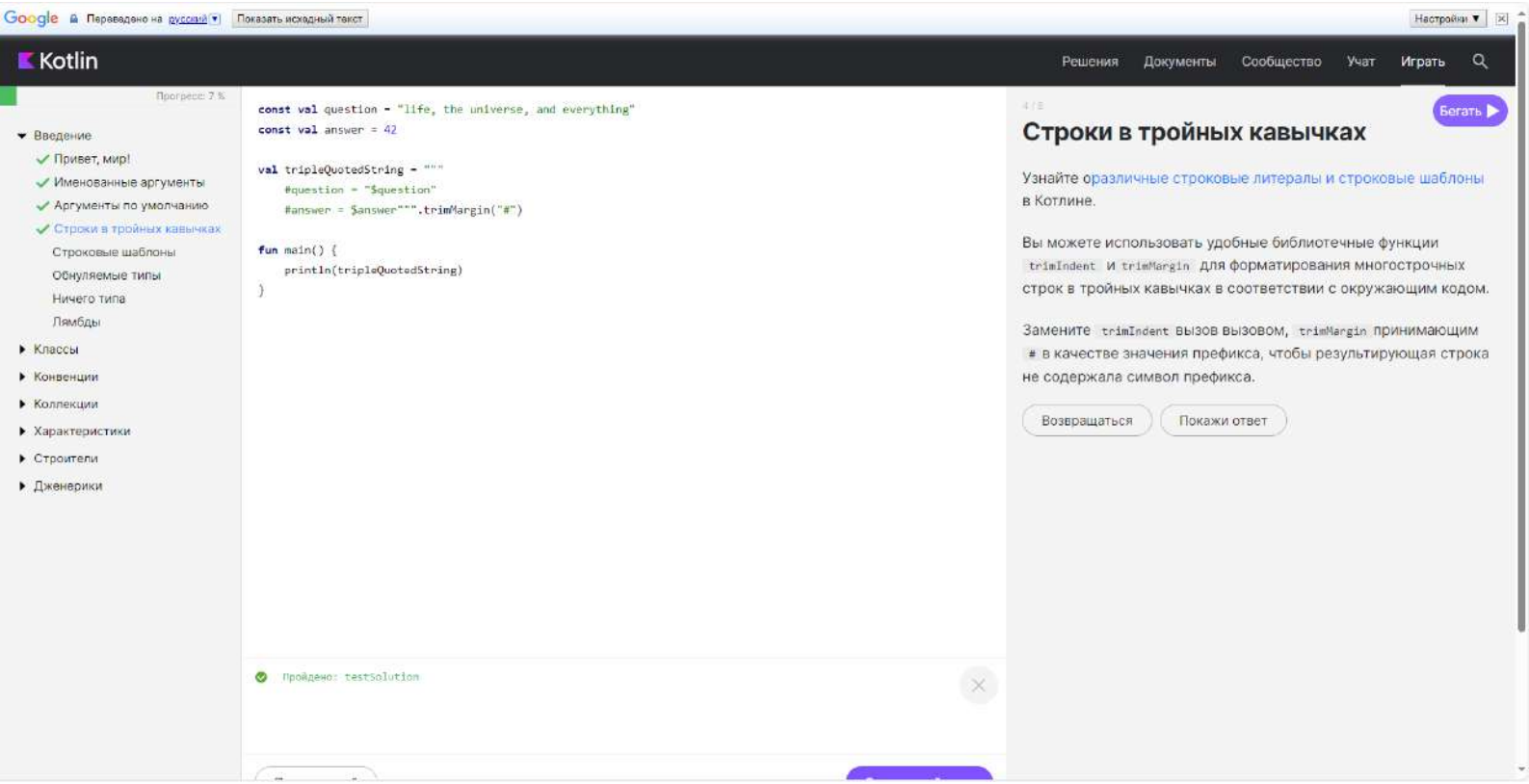
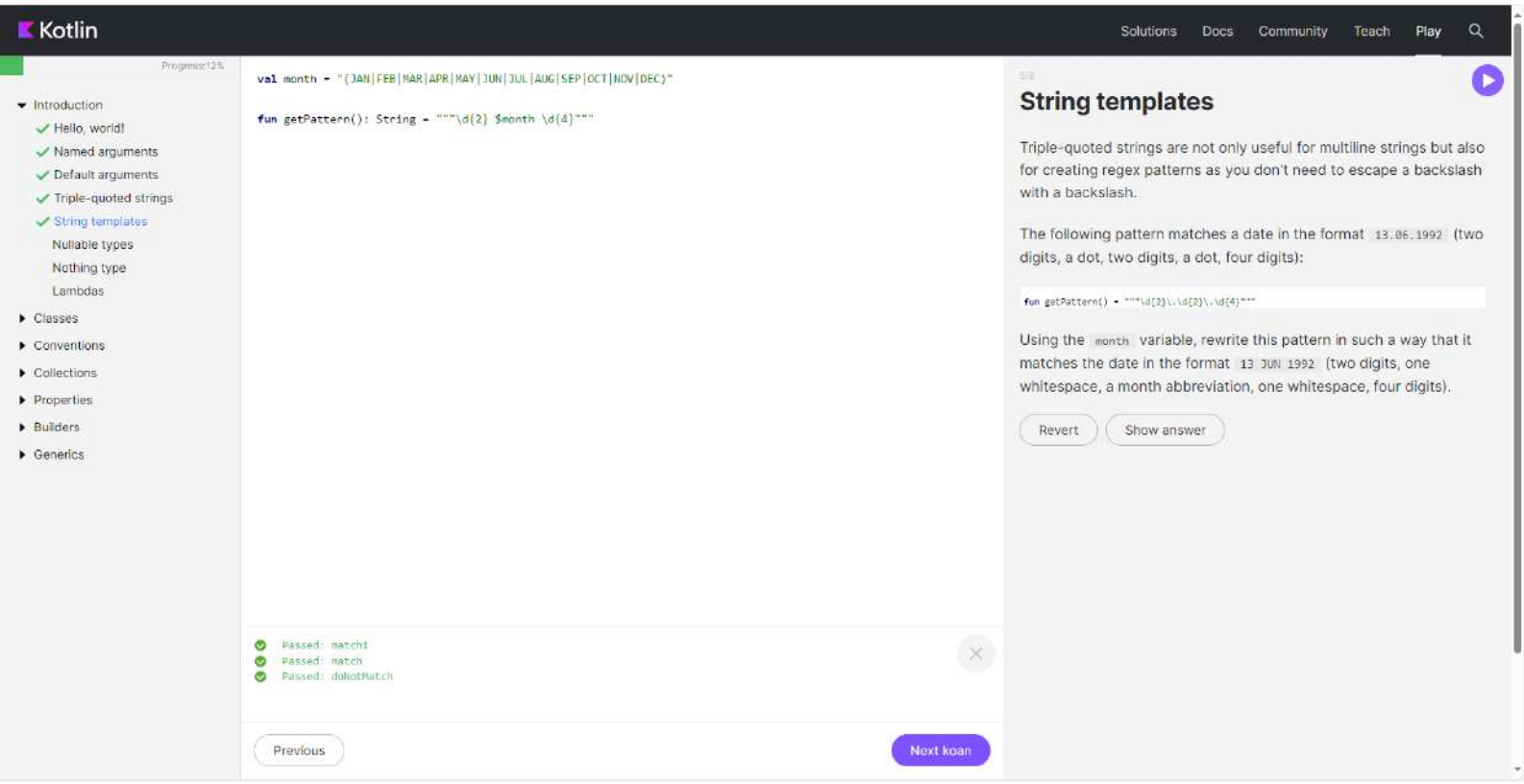
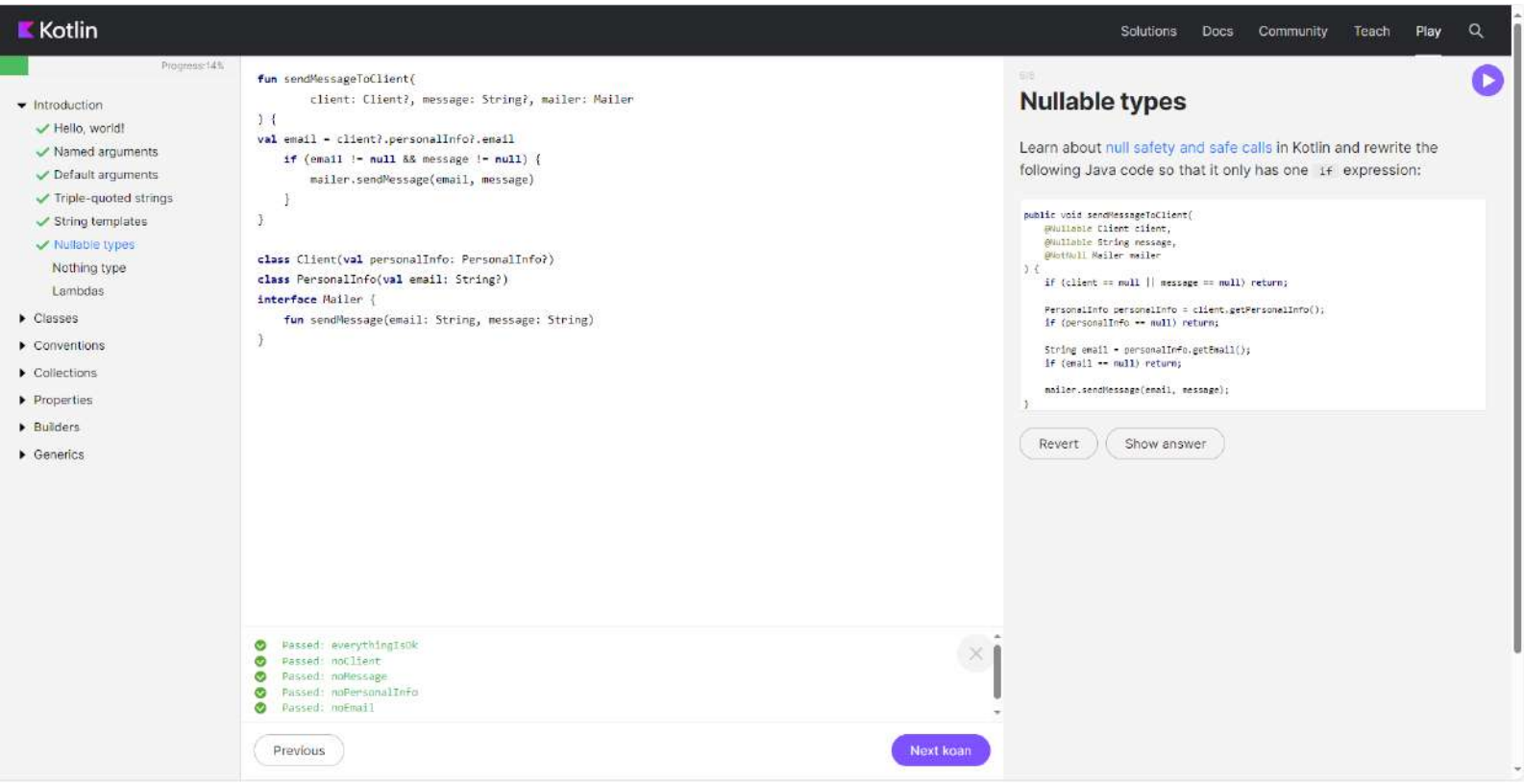

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:0%
Introduction
Hello, world!
Named arguments
Default arguments
Triple-quoted strings
String templates
Nullable types
Nothing type
Lambdas
Classes
Conventions
Collections
Properties
Builders
Generics
fun start(): String = "OK"
Passed: testOk
Run
Simple Functions
Check out the function syntax and change the code to make the function start return the string "OK".
In the Kotlin Koans tasks, the function TODO() will throw an exception. To complete Kotlin Koans, you need to replace this function invocation with meaningful code according to the problem.
Revert
Show answer
"OK"

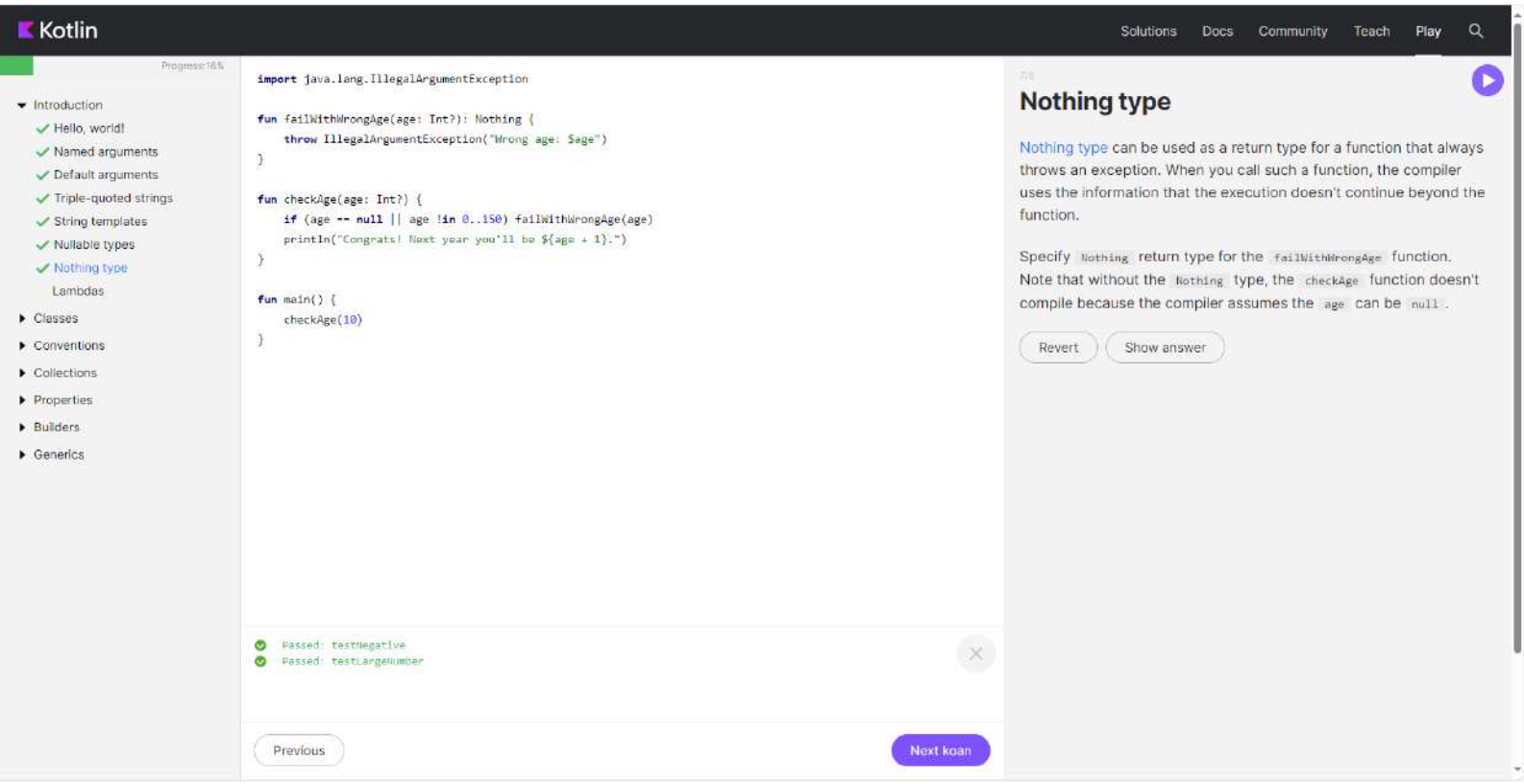

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:5%
Introduction
Hello, world!
Named arguments
Default arguments
Triple-quoted strings
String templates
Nullable types
Nothing type
Lambdas
Classes
Conventions
Collections
Properties
Builders
Generics
fun joinOptions(options: Collection<String>) =
options.joinToString(prefix = "[", postfix = "]")
Passed: testJoinToString
Previous
Next koan
2/8
Named arguments
Make the function joinOptions() return the list in a JSON format (for example, [a, b, c]) by specifying only two arguments.
Default and named arguments help to minimize the number of overloads and improve the readability of the function invocation. The library function joinToString is declared with default values for parameters:
fun joinToString(
seperator: String = ", ",
prefix: String = "",
postfix: String = "",
/* ... */
): String
It can be called on a collection of Strings.
Revert
Show answer

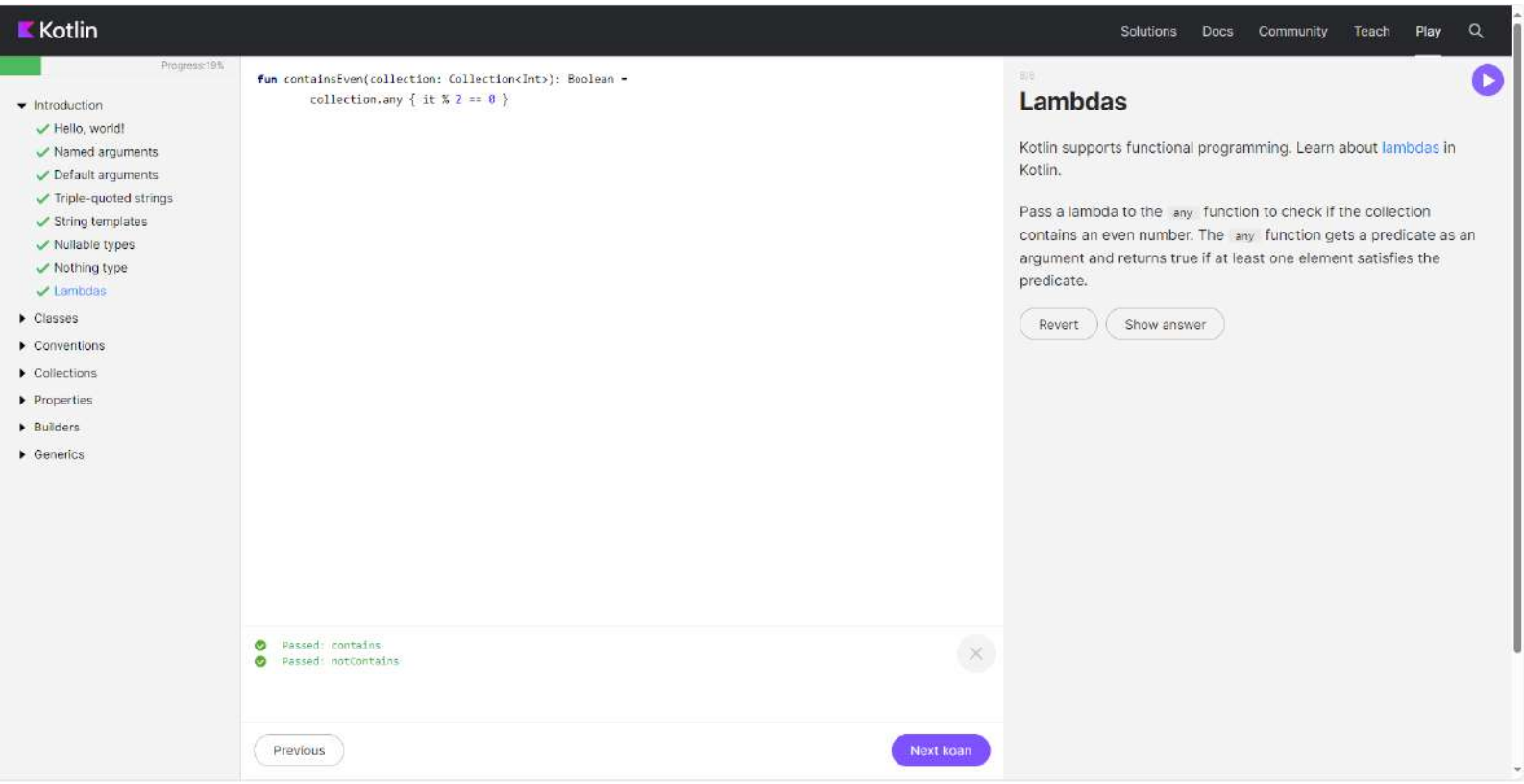

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:7%
Introduction
Hello, world!
Named arguments
Default arguments
Triple-quoted strings
String templates
Nullable types
Nothing type
Lambdas
Classes
Conventions
Collections
Properties
Builders
Generics
fun foo(name: String, number: Int = 42, toUpperCase: 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)
)
Passed: testDefaultAndNamedParams
Previous
Next koan
Default arguments
Imagine you have several overloads of 'foo()' in Java:
public String foo(String name, int number, boolean toUpperCase) {
return (toUpperCase ? name.toUpperCase() : name) + number;
}
public String foo(String name, int number) {
return foo(name, number, false);
}
public String foo(String name, boolean toUpperCase) {
return foo(name, 42, toUpperCase);
}
public String foo(String name) {
return foo(name, 42);
}
You can replace all these Java overloads with one function in Kotlin. Change the declaration of the foo function in a way that makes the code using foo compile. Use default and named arguments.
Revert
Show answer

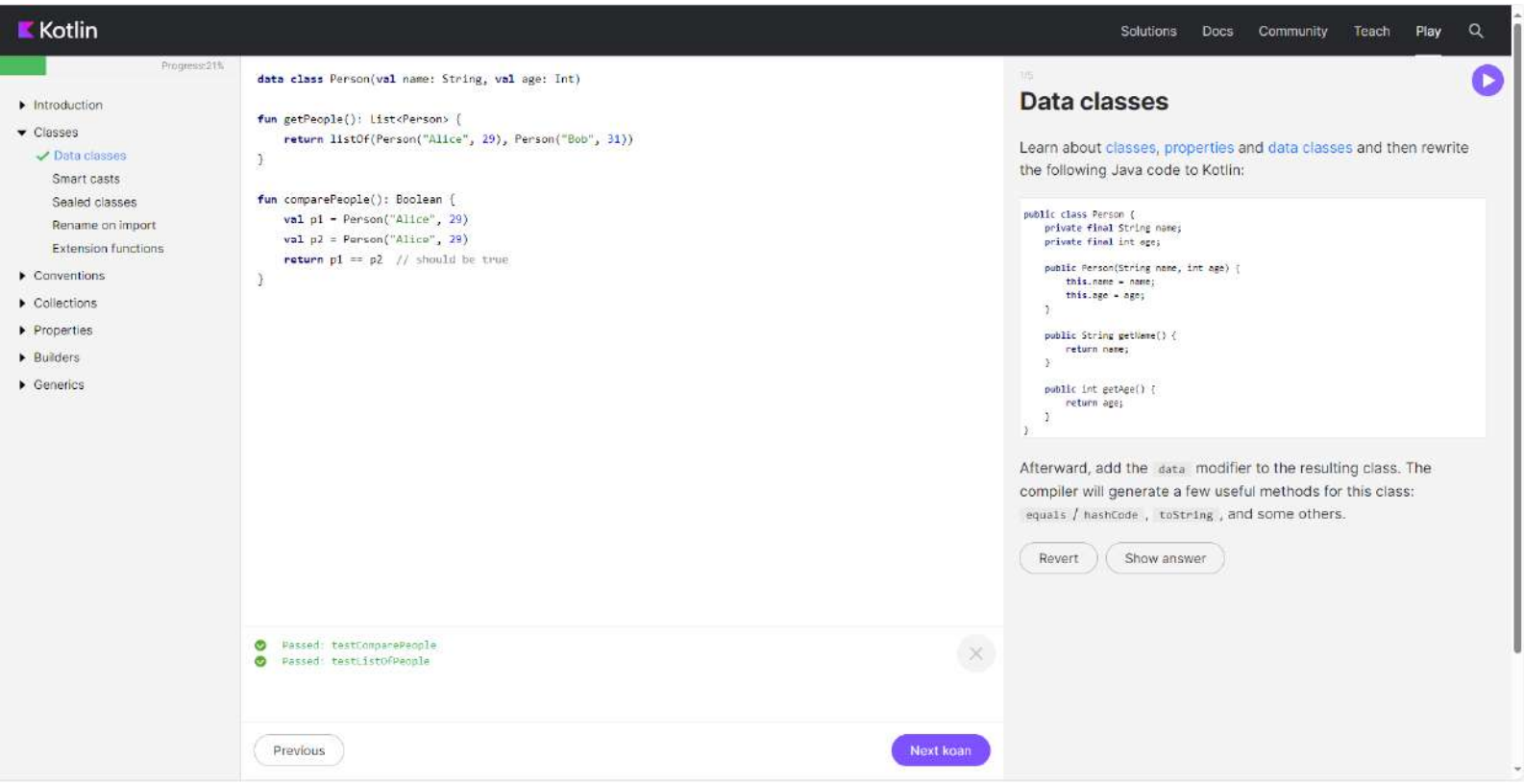

Google Переведено на русский Показать исходный текст Настройки
Kotlin
Решения Документы Сообщество Учат Играть
Прогресс: 7 %
Введение
Привет, мир!
Именованные аргументы
Аргументы по умолчанию
Строки в тройных кавычках
Строковые шаблоны
Обнуляемые типы
Ничего типа
Лямбды
Классы
Конвенции
Коллекции
Характеристики
Строители
Дженерики
const val question = "life, the universe, and everything"
const val answer = 42
val tripleQuotedString = """
#question = "$question"
#answer = $answer""".trimMargin("#")
fun main() {
println(tripleQuotedString)
}
Пройдено: testSolution
4 / 8
Бегать
Строки в тройных кавычках
Узнайте о различные строковые литералы и строковые шаблоны в Котлине.
Вы можете использовать удобные библиотечные функции trimIndent и trimMargin для форматирования многострочных строк в тройных кавычках в соответствии с окружающим кодом.
Замените trimIndent вызов вызовом, trimMargin принимающим # в качестве значения префикса, чтобы результирующая строка не содержала символ префикса.
Возвращаться
Покажи ответ

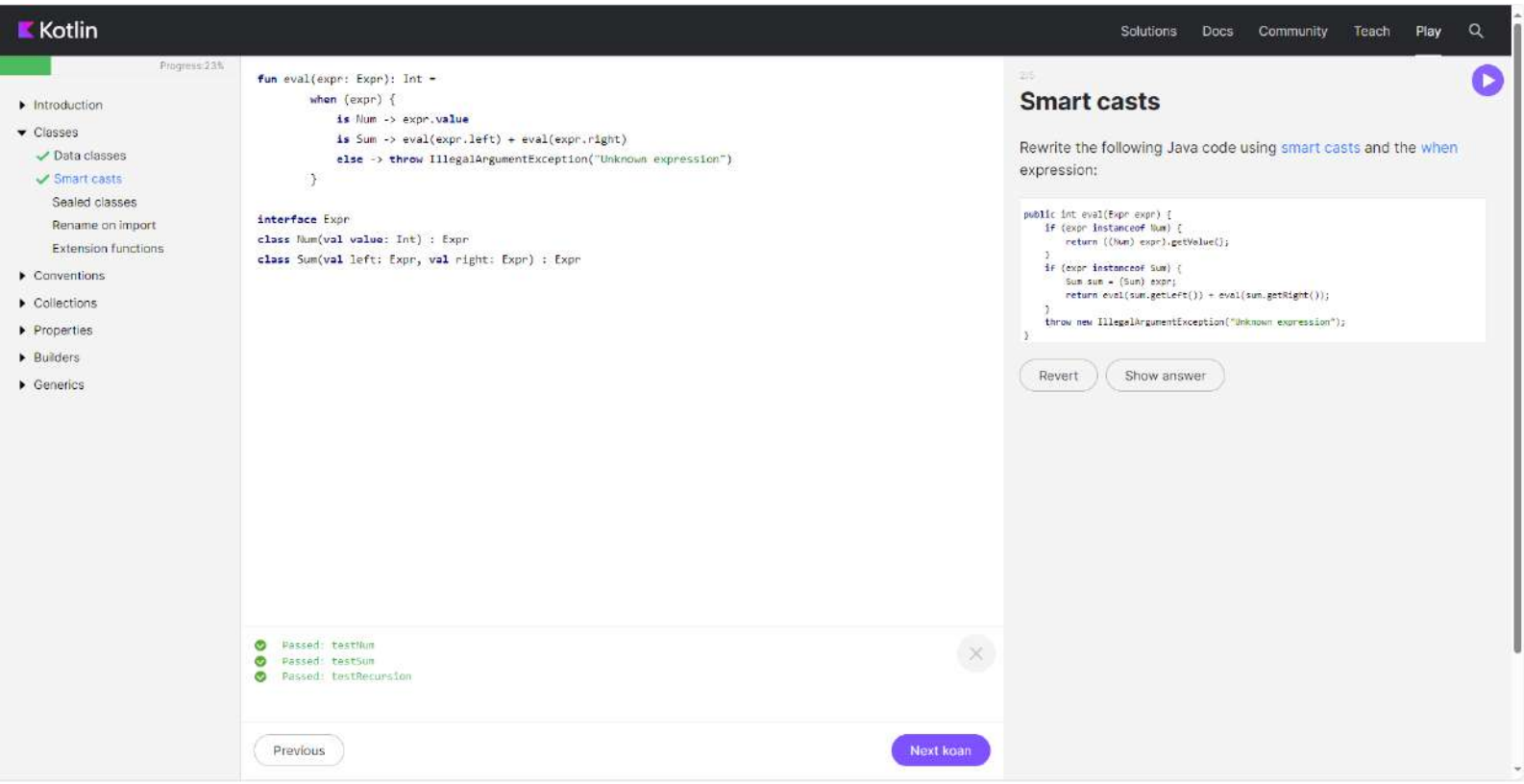

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:12%
Introduction
Hello, world!
Named arguments
Default arguments
Triple-quoted strings
String templates
Nullable types
Nothing type
Lambdas
Classes
Conventions
Collections
Properties
Builders
Generics
val month = "(JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC)"
fun getPattern(): String = """\d{2} $month \d{4}"""
Passed: match1
Passed: match
Passed: doNotMatch
Previous
Next koan
String templates
Triple-quoted strings are not only useful for multiline strings but also for creating regex patterns as you don't need to escape a backslash with a backslash.
The following pattern matches a date in the format 13.06.1992 (two digits, a dot, two digits, a dot, four digits):
fun getPattern() = """\d{2}\.\d{2}\.\d{4}"""
Using the month variable, rewrite this pattern in such a way that it matches the date in the format 13 JUN 1992 (two digits, one whitespace, a month abbreviation, one whitespace, four digits).
Revert
Show answer

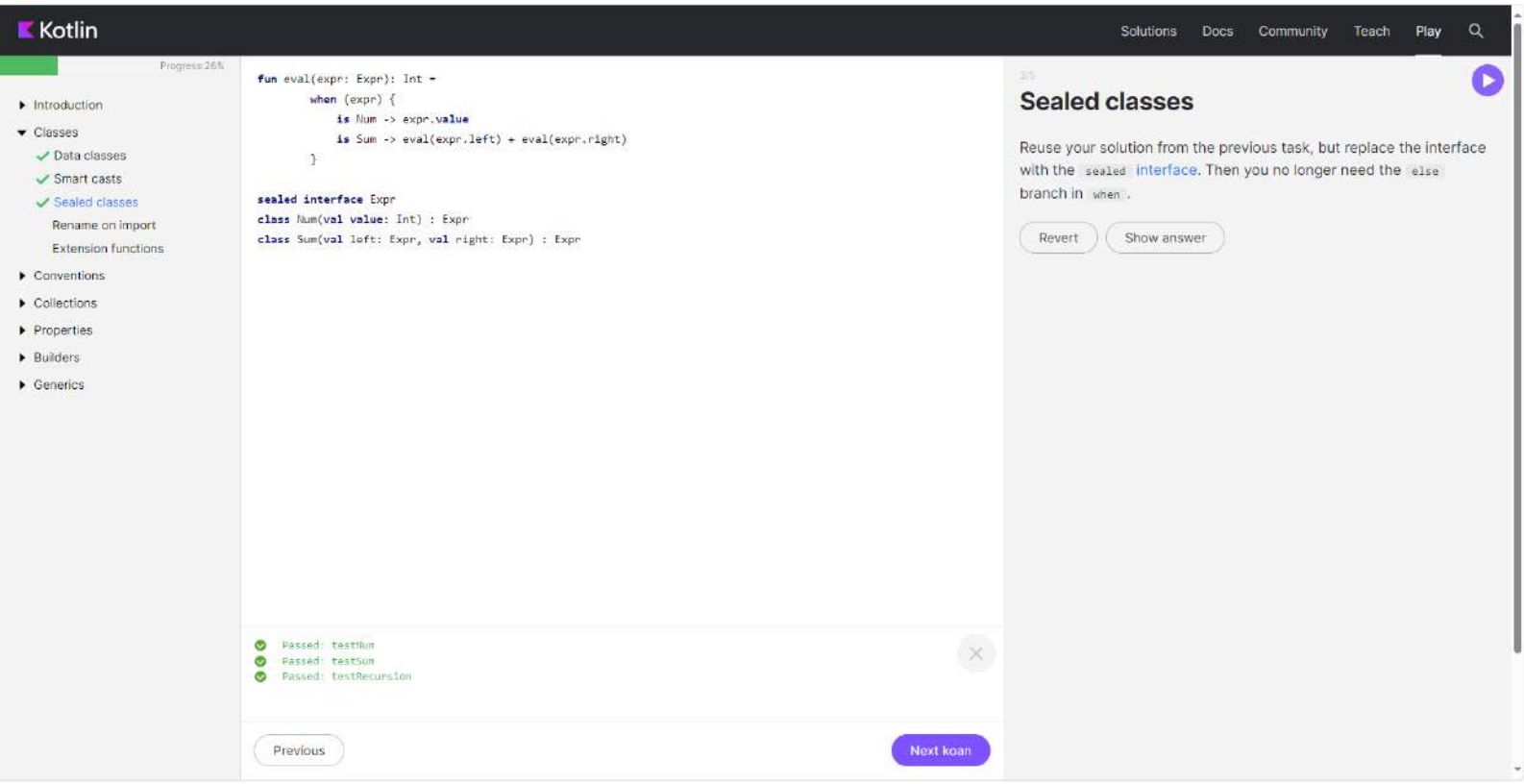

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:14%
Introduction
Hello, world!
Named arguments
Default arguments
Triple-quoted strings
String templates
Nullable types
Nothing type
Lambdas
Classes
Conventions
Collections
Properties
Builders
Generics
fun sendMessageToClient(
client: Client?, message: 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)
}
Passed: everythingIsOk
Passed: noClient
Passed: noMessage
Passed: noPersonalInfo
Passed: noEmail
Previous
Next koan
Nullable types
Learn about null safety and safe calls in Kotlin and rewrite the following Java code so that it only has one if expression:
public void sendMessageToClient(
@Nullable Client client,
@Nullable String message,
@NotNull Mailer mailer
) {
if (client == null || message == null) return;
PersonalInfo personalInfo = client.getPersonalInfo();
if (personalInfo == null) return;
String email = personalInfo.getEmail();
if (email == null) return;
mailer.sendMessage(email, message);
}
Revert
Show answer

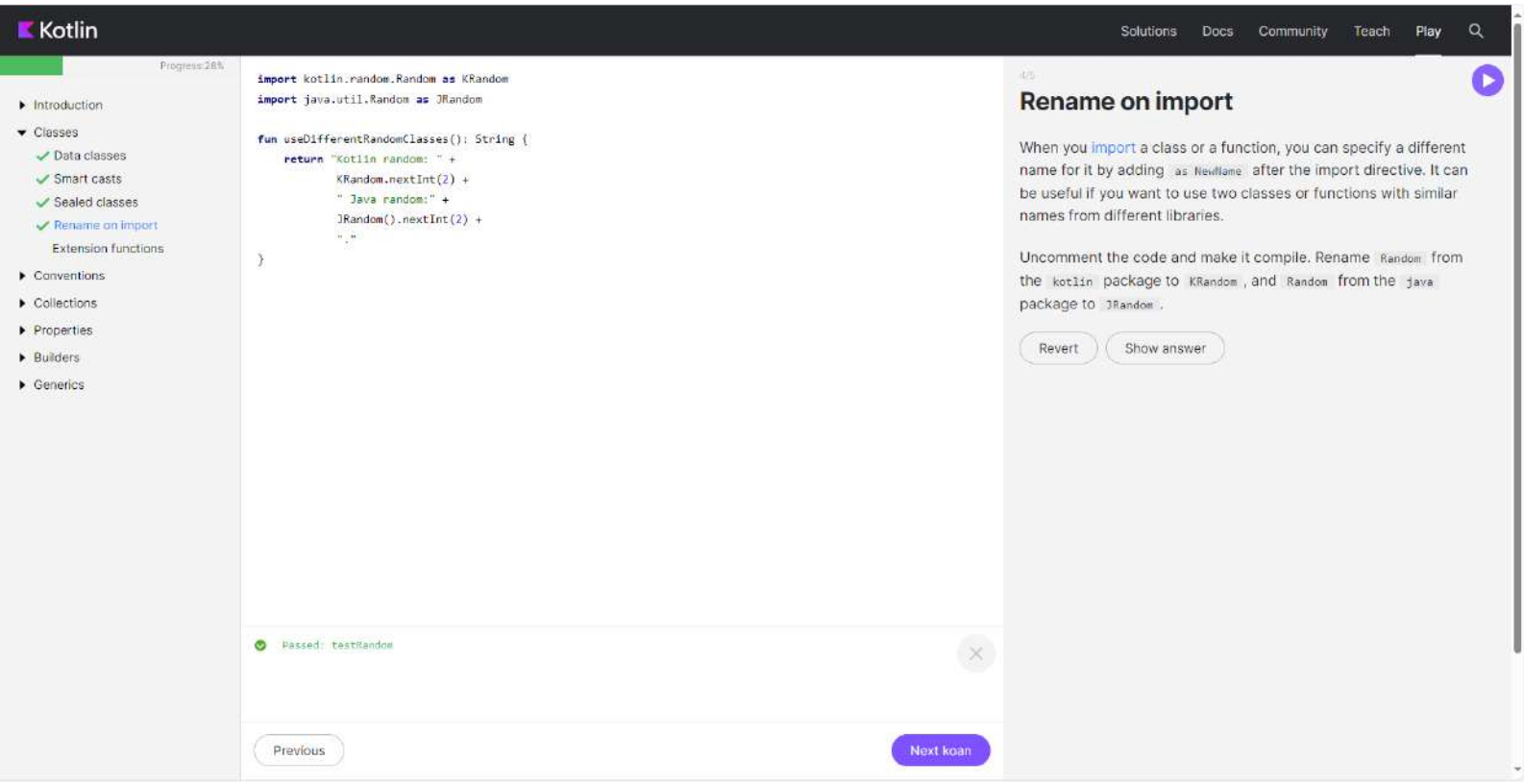

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:16%
Introduction
Hello, world!
Named arguments
Default arguments
Triple-quoted strings
String templates
Nullable types
Nothing type
Lambdas
Classes
Conventions
Collections
Properties
Builders
Generics
import java.lang.IllegalArgumentException
fun failWithWrongAge(age: 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)
}
Passed: testNegative
Passed: testLargeNumber
Previous
Next koan
Nothing type
Nothing type can be used as a return type for a function that always throws an exception. When you call such a function, the compiler uses the information that the execution doesn't continue beyond the function.
Specify Nothing return type for the failWithWrongAge function. Note that without the Nothing type, the checkAge function doesn't compile because the compiler assumes the age can be null.
Revert
Show answer

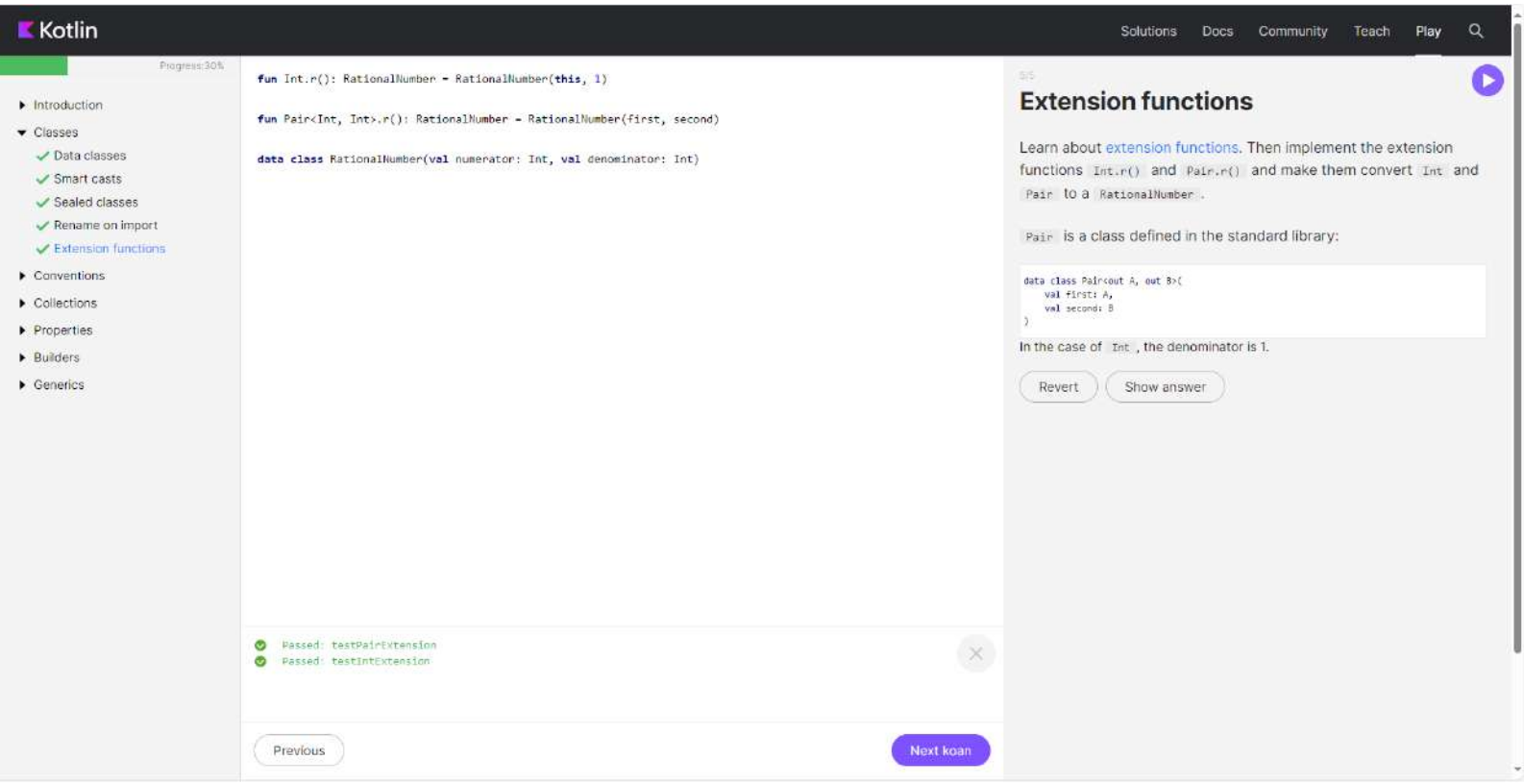

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:19%
Introduction
Hello, world!
Named arguments
Default arguments
Triple-quoted strings
String templates
Nullable types
Nothing type
Lambdas
Classes
Conventions
Collections
Properties
Builders
Generics
fun containsEven(collection: Collection<Int>): Boolean =
collection.any { it % 2 == 0 }
Passed: contains
Passed: notContains
Previous
Next koan
8/8
Lambdas
Kotlin supports functional programming. Learn about lambdas in Kotlin.
Pass a lambda to the any function to check if the collection contains an even number. The any function gets a predicate as an argument and returns true if at least one element satisfies the predicate.
Revert
Show answer

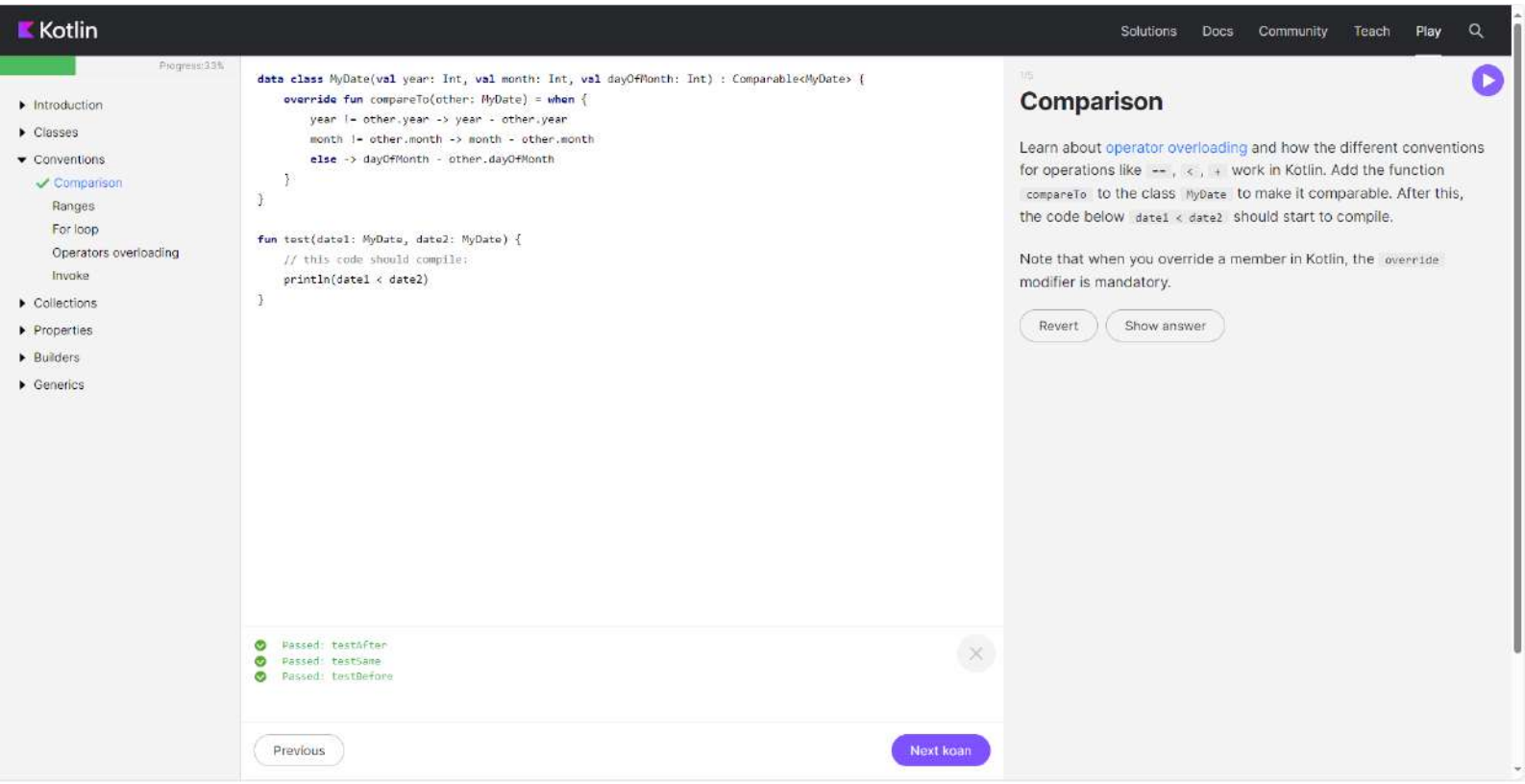

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:21%
Introduction
Classes
Data classes
Smart casts
Sealed classes
Rename on import
Extension functions
Conventions
Collections
Properties
Builders
Generics
data class Person(val name: String, val age: Int)
fun getPeople(): List<Person> {
return listOf(Person("Alice", 29), Person("Bob", 31))
}
fun comparePeople(): Boolean {
val p1 = Person("Alice", 29)
val p2 = Person("Alice", 29)
return p1 == p2 // should be true
}
Passed: testComparePeople
Passed: testListOfPeople
Previous
Next koan
1/5
Data classes
Learn about classes, properties and data classes and then rewrite the following Java code to Kotlin:
public class Person {
private final String name;
private final int age;
public Person(String name, int age) {
this.name = name;
this.age = age;
}
public String getName() {
return name;
}
public int getAge() {
return age;
}
}
Afterward, add the data modifier to the resulting class. The compiler will generate a few useful methods for this class: equals / hashCode , toString , and some others.
Revert
Show answer

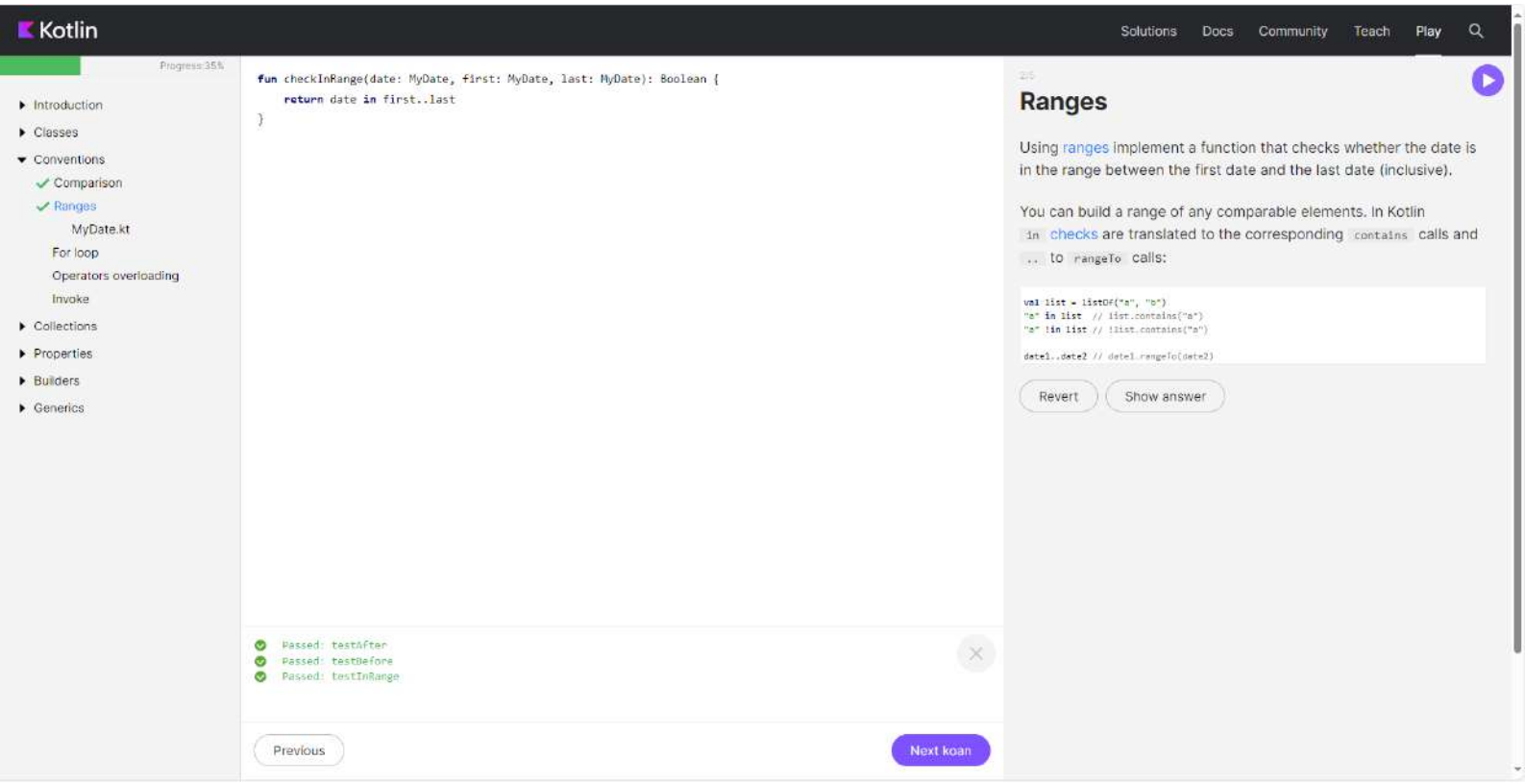

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:23%
Introduction
Classes
Data classes
Smart casts
Sealed classes
Rename on import
Extension functions
Conventions
Collections
Properties
Builders
Generics
fun eval(expr: Expr): Int =
when (expr) {
is Num -> expr.value
is Sum -> eval(expr.left) + eval(expr.right)
else -> throw IllegalArgumentException("Unknown expression")
}
interface Expr
class Num(val value: Int) : Expr
class Sum(val left: Expr, val right: Expr) : Expr
Passed: testNum
Passed: testSum
Passed: testRecursion
Previous
Next koan
2/5
Smart casts
Rewrite the following Java code using smart casts and the when expression:
public int eval(Expr expr) {
if (expr instanceof Num) {
return ((Num) expr).getValue();
}
if (expr instanceof Sum) {
Sum sum = (Sum) expr;
return eval(sum.getLeft()) + eval(sum.getRight());
}
throw new IllegalArgumentException("Unknown expression");
}
Revert
Show answer

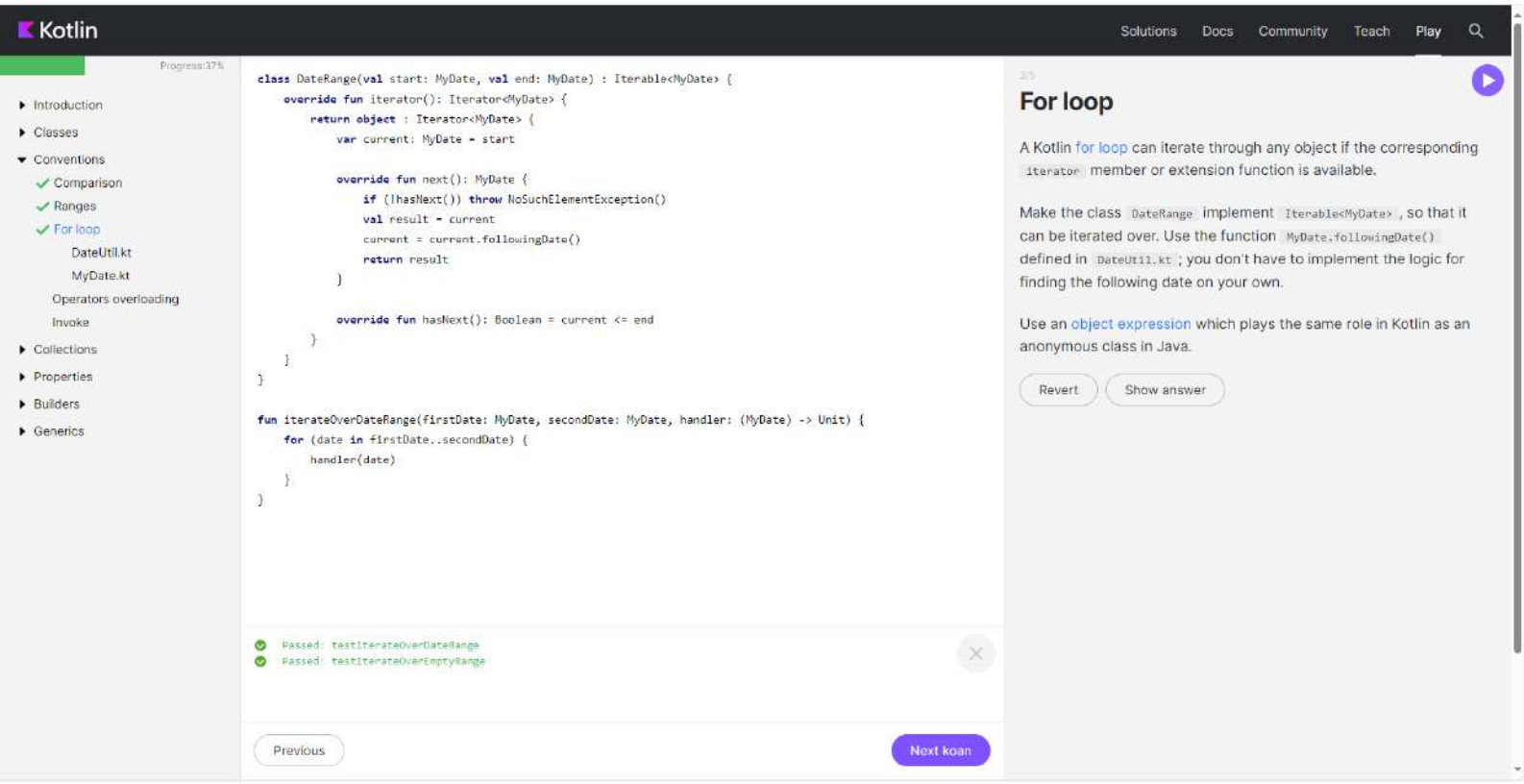

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:26%
Introduction
Classes
Data classes
Smart casts
Sealed classes
Rename on import
Extension functions
Conventions
Collections
Properties
Builders
Generics
fun eval(expr: Expr): Int =
when (expr) {
is Num -> expr.value
is Sum -> eval(expr.left) + eval(expr.right)
}
sealed interface Expr
class Num(val value: Int) : Expr
class Sum(val left: Expr, val right: Expr) : Expr
Passed: testNum
Passed: testSum
Passed: testRecursion
Previous
Next koan
Sealed classes
Reuse your solution from the previous task, but replace the interface with the sealed interface. Then you no longer need the else branch in when.
Revert
Show answer

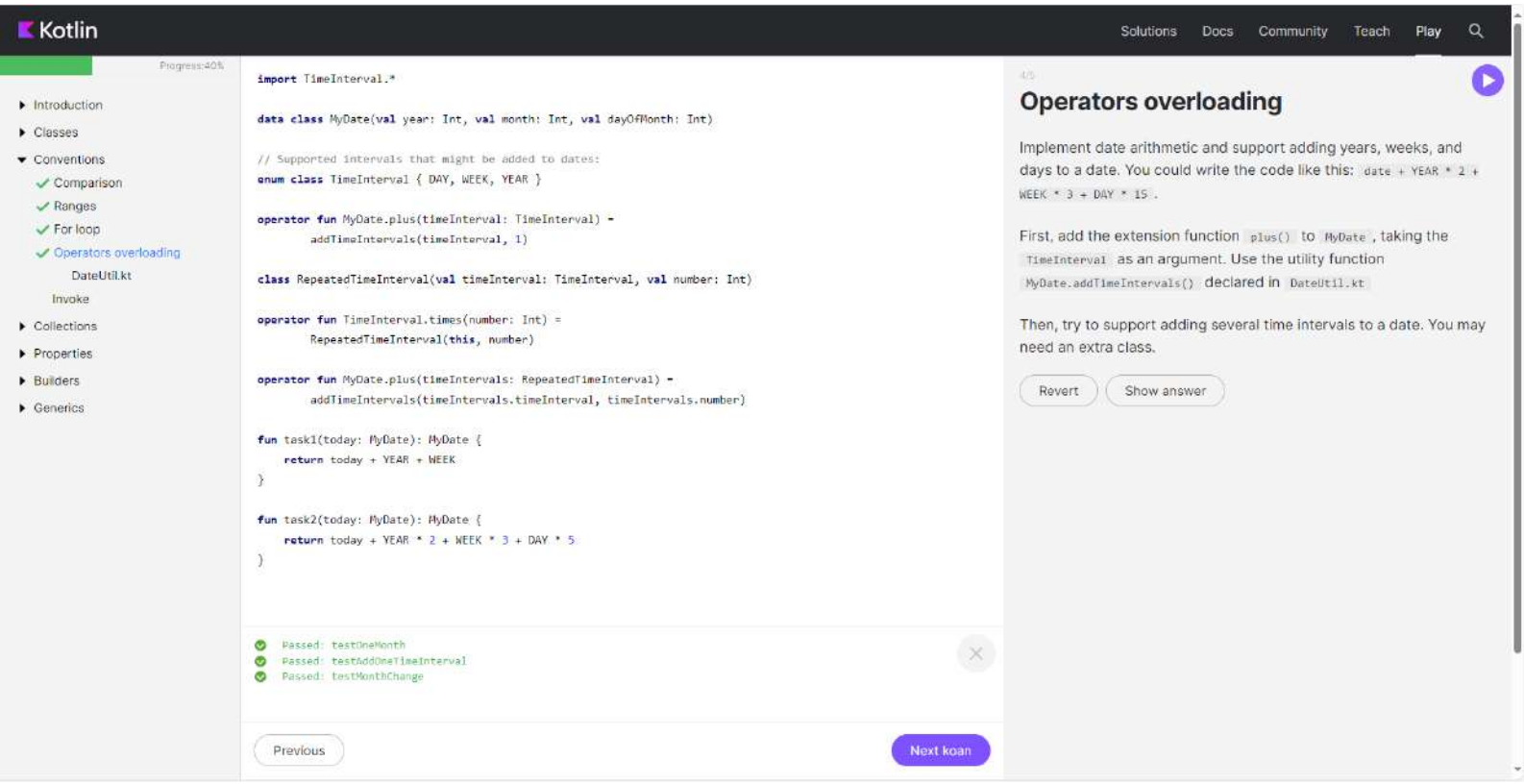

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:26%
Introduction
Classes
Data classes
Smart casts
Sealed classes
Rename on import
Extension functions
Conventions
Collections
Properties
Builders
Generics
import kotlin.random.Random as KRandom
import java.util.Random as JRandom
fun useDifferentRandomClasses(): String {
return "Kotlin random: " +
KRandom.nextInt(2) +
" Java random:" +
JRandom().nextInt(2) +
"."
}
Passed: testRandom
Previous
Next koan
4/5
Rename on import
When you import a class or a function, you can specify a different name for it by adding as NewName after the import directive. It can be useful if you want to use two classes or functions with similar names from different libraries.
Uncomment the code and make it compile. Rename Random from the kotlin package to KRandom, and Random from the java package to JRandom.
Revert
Show answer

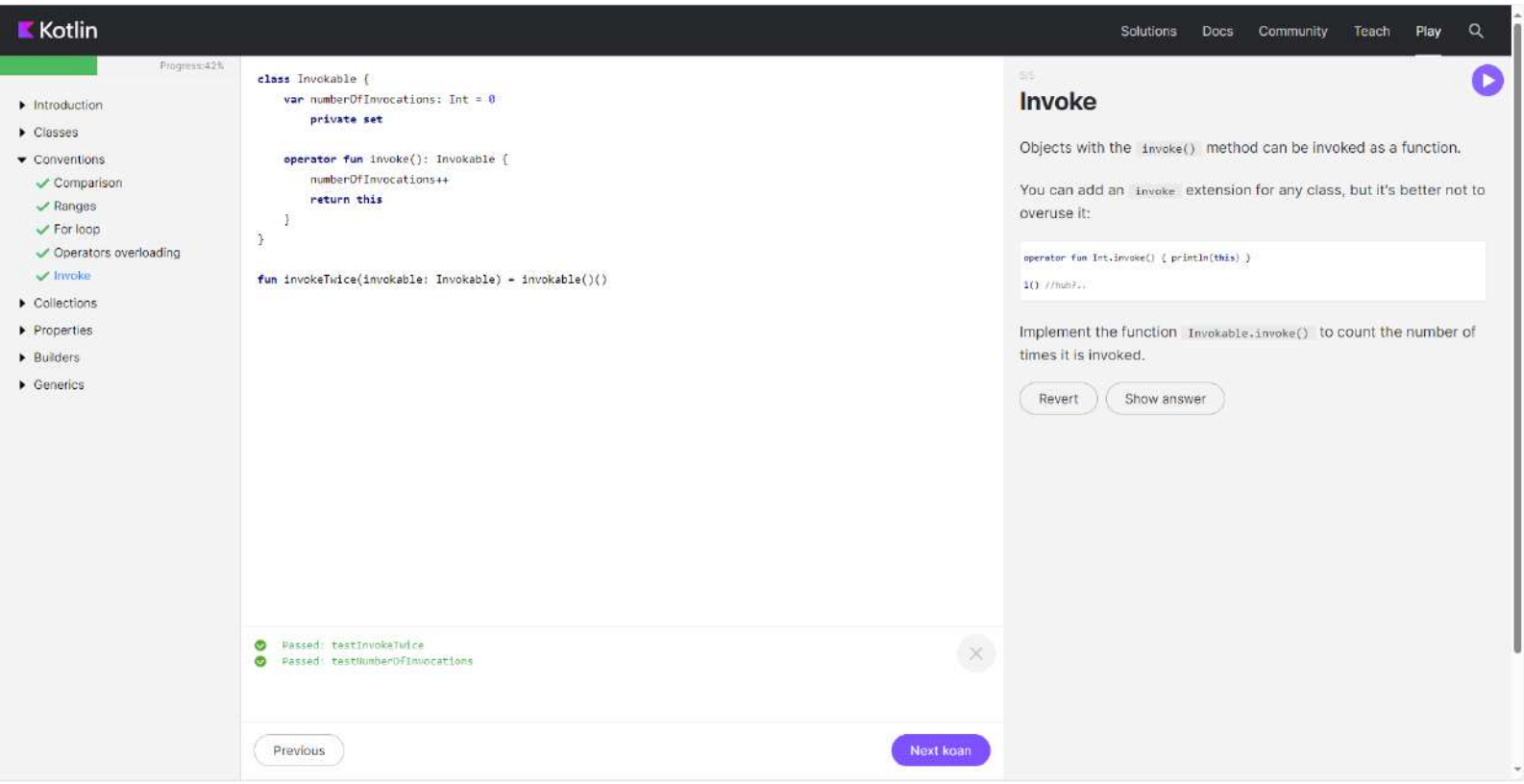

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:30%
Introduction
Classes
Data classes
Smart casts
Sealed classes
Rename on import
Extension functions
Conventions
Collections
Properties
Builders
Generics
fun Int.r(): RationalNumber = RationalNumber(this, 1)
fun Pair<Int, Int>.r(): RationalNumber = RationalNumber(first, second)
data class RationalNumber(val numerator: Int, val denominator: Int)
Passed: testPairExtension
Passed: testIntExtension
Previous
Next koan
Extension functions
Learn about extension functions. Then implement the extension functions Int.r() and Pair.r() and make them convert Int and Pair to a RationalNumber.
Pair is a class defined in the standard library:
data class Pair<out A, out B>(
val first: A,
val second: B
)
In the case of Int, the denominator is 1.
Revert
Show answer

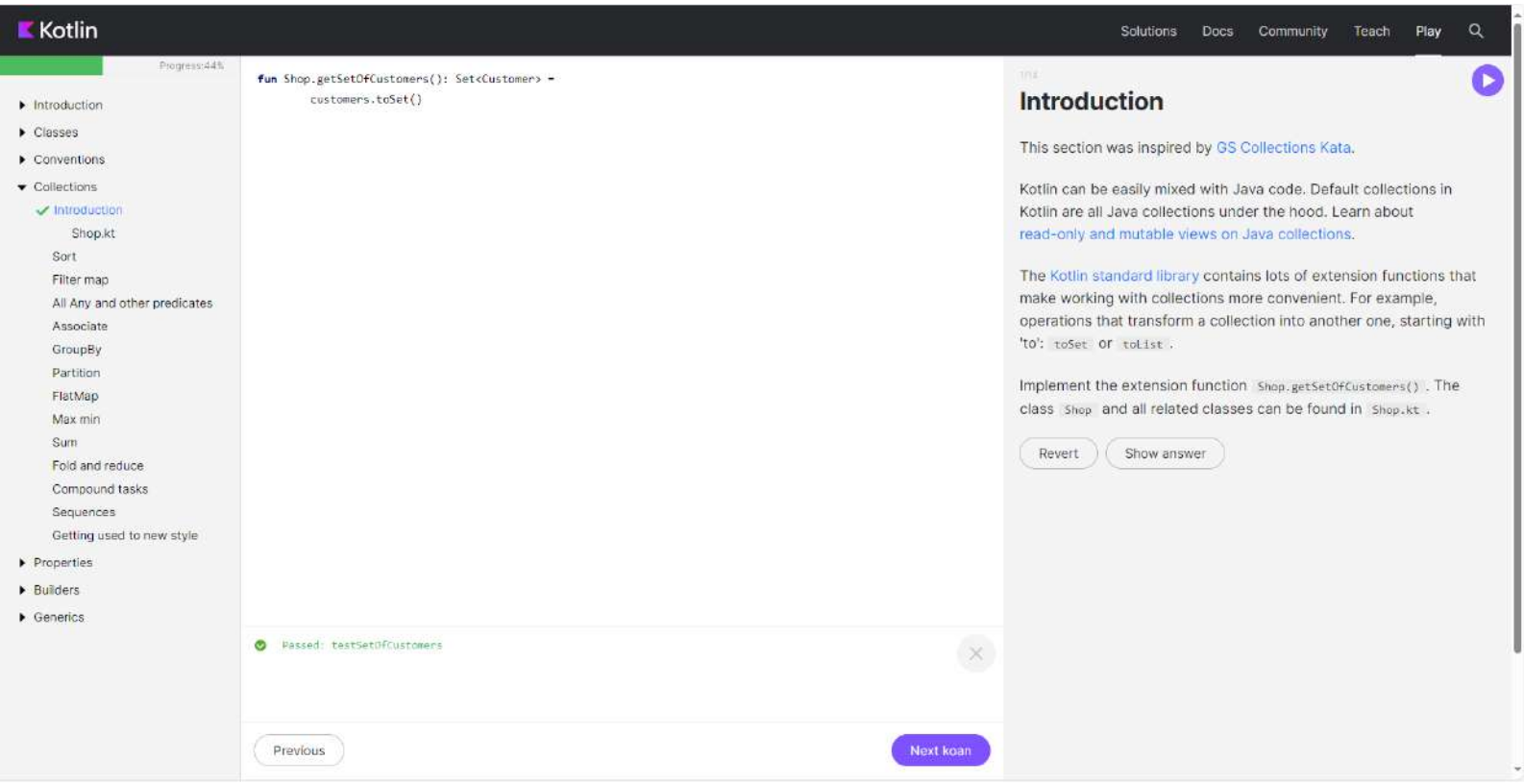

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:33%
Introduction
Classes
Conventions
Comparison
Ranges
For loop
Operators overloading
Invoke
Collections
Properties
Builders
Generics
data class MyDate(val year: Int, val month: Int, val dayOfMonth: Int) : Comparable<MyDate> {
override fun compareTo(other: 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)
}
Passed: testAfter
Passed: testSame
Passed: testBefore
Previous
Next koan
Comparison
Learn about operator overloading and how the different conventions for operations like == , < , + work in Kotlin. Add the function compareTo to the class MyDate to make it comparable. After this, the code below date1 < date2 should start to compile.
Note that when you override a member in Kotlin, the override modifier is mandatory.
Revert
Show answer

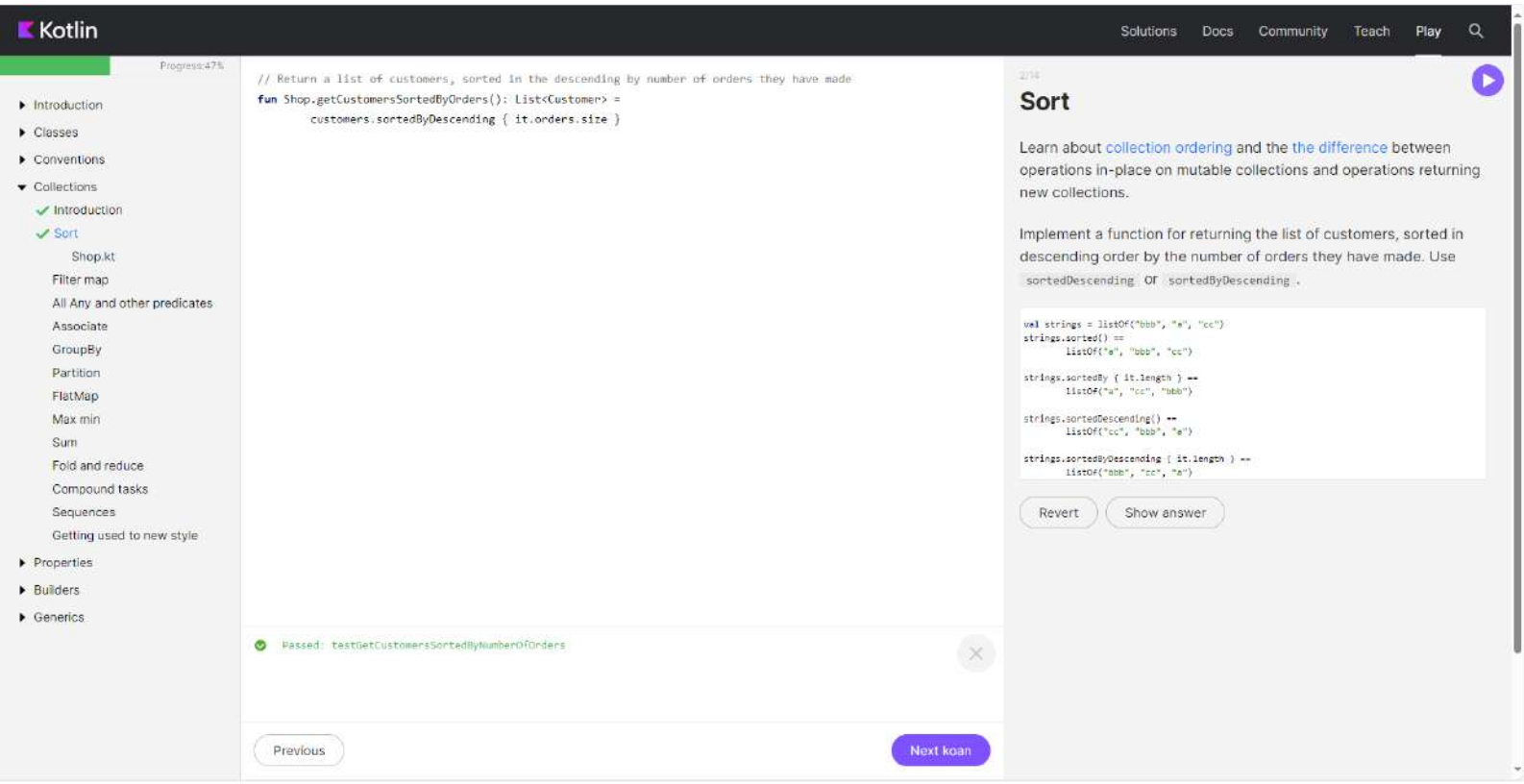

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:35%
Introduction
Classes
Conventions
Comparison
Ranges
MyDate.kt
For loop
Operators overloading
Invoke
Collections
Properties
Builders
Generics
fun checkInRange(date: MyDate, first: MyDate, last: MyDate): Boolean {
return date in first..last
}
Passed: testAfter
Passed: testBefore
Passed: testInRange
Previous
Next koan
2/5
Ranges
Using ranges implement a function that checks whether the date is in the range between the first date and the last date (inclusive).
You can build a range of any comparable elements. In Kotlin in checks are translated to the corresponding contains calls and .. to rangeTo calls:
val list = listOf("a", "b")
"a" in list // list.contains("a")
"a" !in list // !list.contains("a")
date1..date2 // date1.rangeTo(date2)
Revert
Show answer

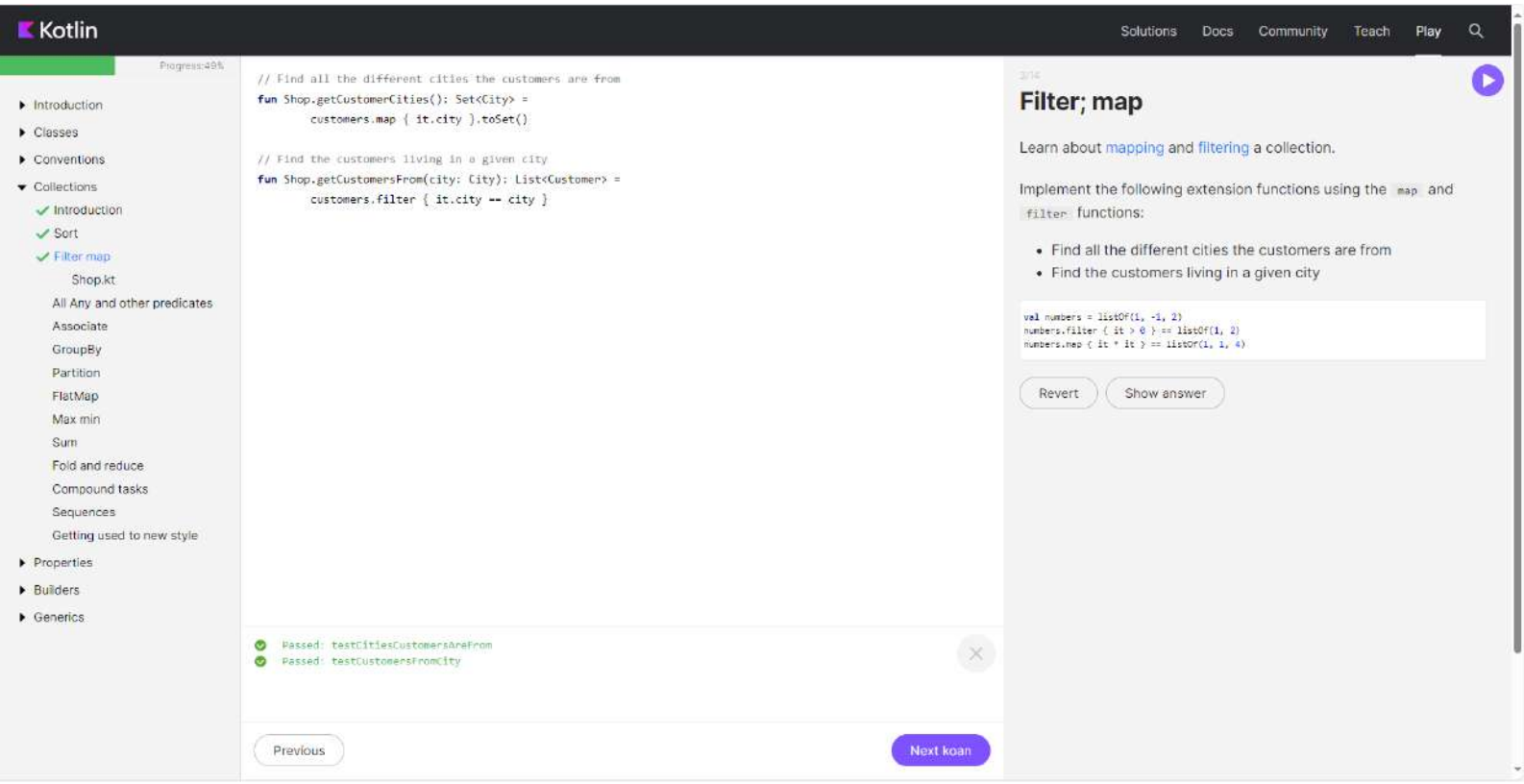

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:37%
Introduction
Classes
Conventions
Comparison
Ranges
For loop
DateUtil.kt
MyDate.kt
Operators overloading
Invoke
Collections
Properties
Builders
Generics
class DateRange(val start: MyDate, val end: MyDate) : Iterable<MyDate> {
override fun iterator(): 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)
}
}
Passed: testIterateOverDateRange
Passed: testIterateOverEmptyRange
Previous
Next koan
2/5
For loop
A Kotlin for loop can iterate through any object if the corresponding iterator member or extension function is available.
Make the class DateRange implement Iterable<MyDate>, so that it can be iterated over. Use the function MyDate.followingDate() defined in DateUtil.kt; you don't have to implement the logic for finding the following date on your own.
Use an object expression which plays the same role in Kotlin as an anonymous class in Java.
Revert
Show answer

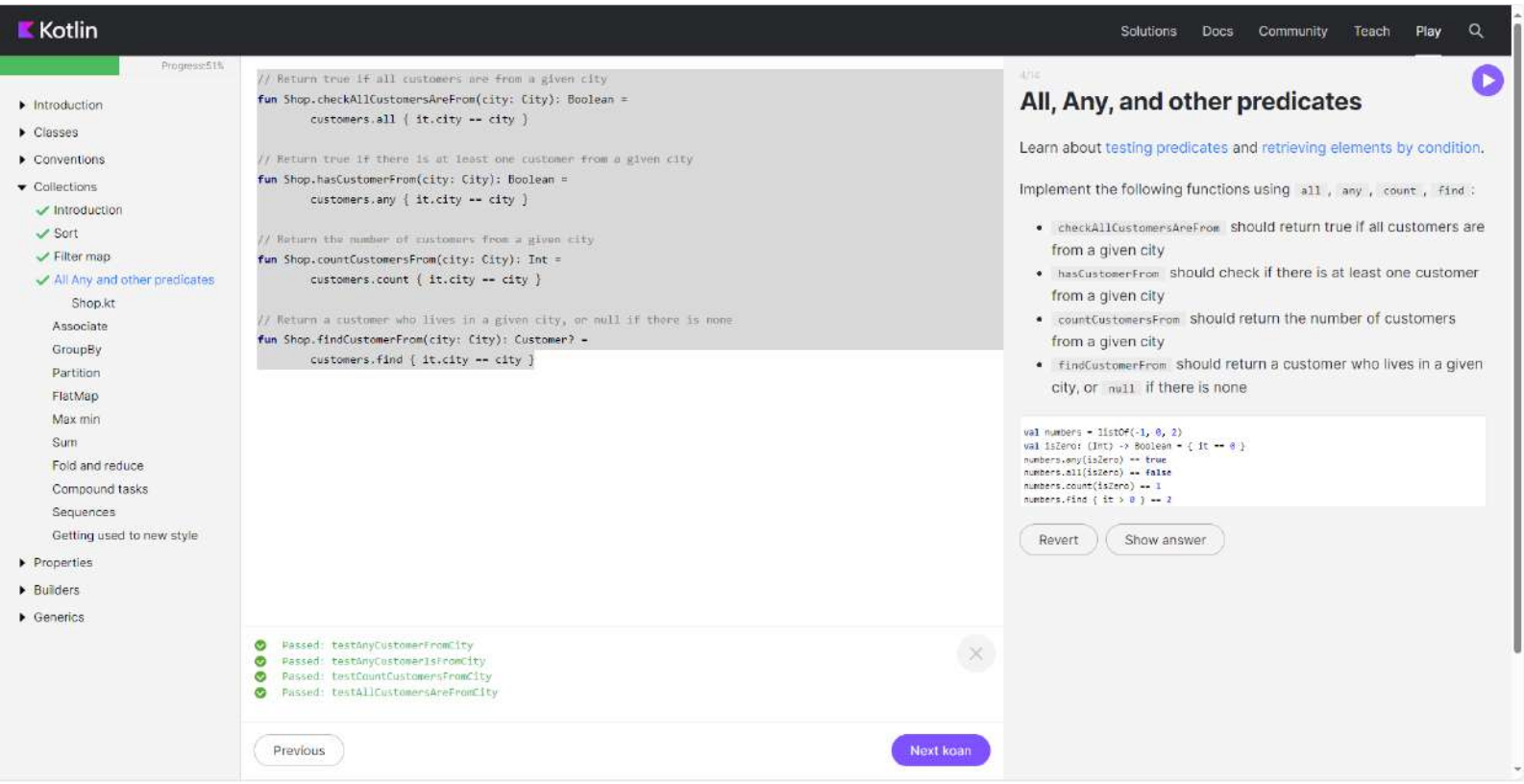

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:40%
Introduction
Classes
Conventions
Comparison
Ranges
For loop
Operators overloading
DateUtil.kt
Invoke
Collections
Properties
Builders
Generics
import TimeInterval.*
data class MyDate(val year: Int, val month: Int, val dayOfMonth: Int)
// Supported 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
}
Passed: testOneMonth
Passed: testAddOneTimeInterval
Passed: testMonthChange
Previous
Next koan
4/5
Operators overloading
Implement date arithmetic and support adding years, weeks, and days to a date. You could write the code like this: date + YEAR * 2 + WEEK * 3 + DAY * 15 .
First, add the extension function plus() to MyDate , taking the TimeInterval as an argument. Use the utility function MyDate.addTimeIntervals() declared in DateUtil.kt
Then, try to support adding several time intervals to a date. You may need an extra class.
Revert
Show answer

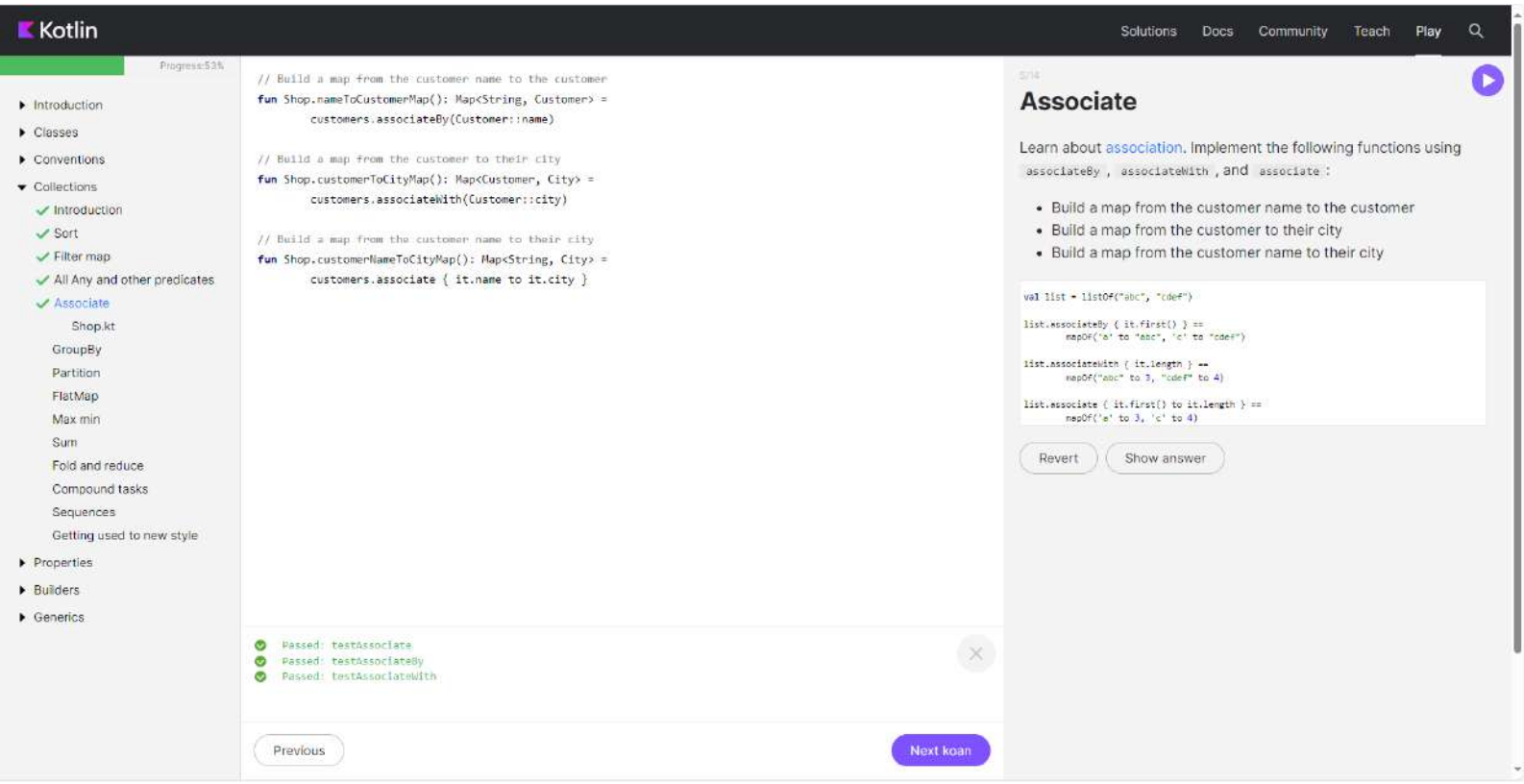

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:42%
Introduction
Classes
Conventions
Comparison
Ranges
For loop
Operators overloading
Invoke
Collections
Properties
Builders
Generics
class Invokable {
var numberOfInvocations: Int = 0
private set
operator fun invoke(): Invokable {
numberOfInvocations++
return this
}
}
fun invokeTwice(invokable: Invokable) = invokable()()
Passed: testInvokeTwice
Passed: testNumberOfInvocations
Previous
Next koan
Invoke
Objects with the invoke() method can be invoked as a function.
You can add an invoke extension for any class, but it's better not to overuse it:
operator fun Int.invoke() { println(this) }
1() //huh?..
Implement the function Invokable.invoke() to count the number of times it is invoked.
Revert
Show answer

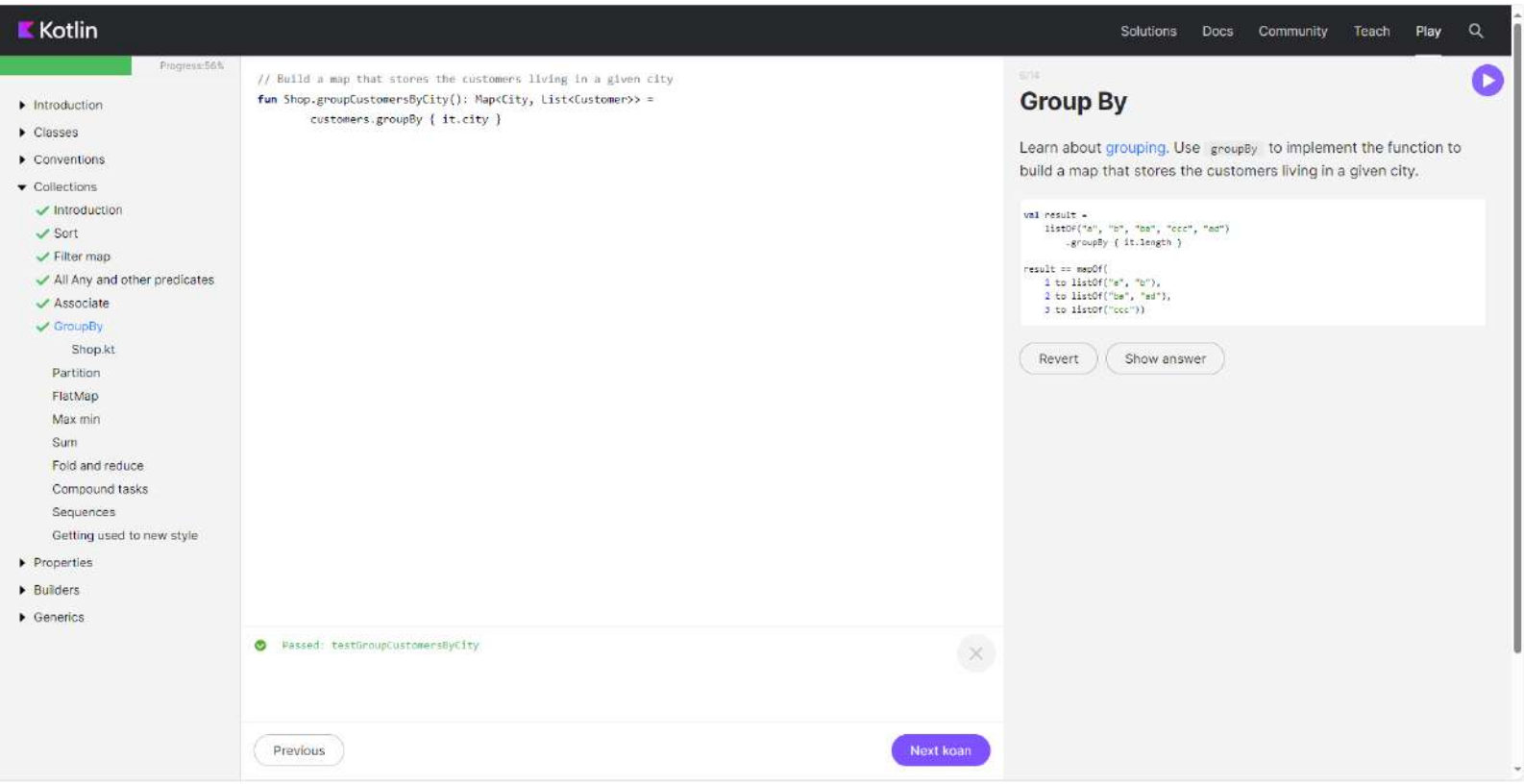

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:44%
Introduction
Classes
Conventions
Collections
Introduction
Shop.kt
Sort
Filter map
All Any and other predicates
Associate
GroupBy
Partition
FlatMap
Max min
Sum
Fold and reduce
Compound tasks
Sequences
Getting used to new style
Properties
Builders
Generics
fun Shop.getSetOfCustomers(): Set<Customer> =
customers.toSet()
Passed: testSetOfCustomers
Previous
Next koan
Introduction
This section was inspired by GS Collections Kata.
Kotlin can be easily mixed with Java code. Default collections in Kotlin are all Java collections under the hood. Learn about read-only and mutable views on Java collections.
The Kotlin standard library contains lots of extension functions that make working with collections more convenient. For example, operations that transform a collection into another one, starting with 'to': toSet or toList .
Implement the extension function Shop.getSetOfCustomers() . The class Shop and all related classes can be found in Shop.kt .
Revert
Show answer

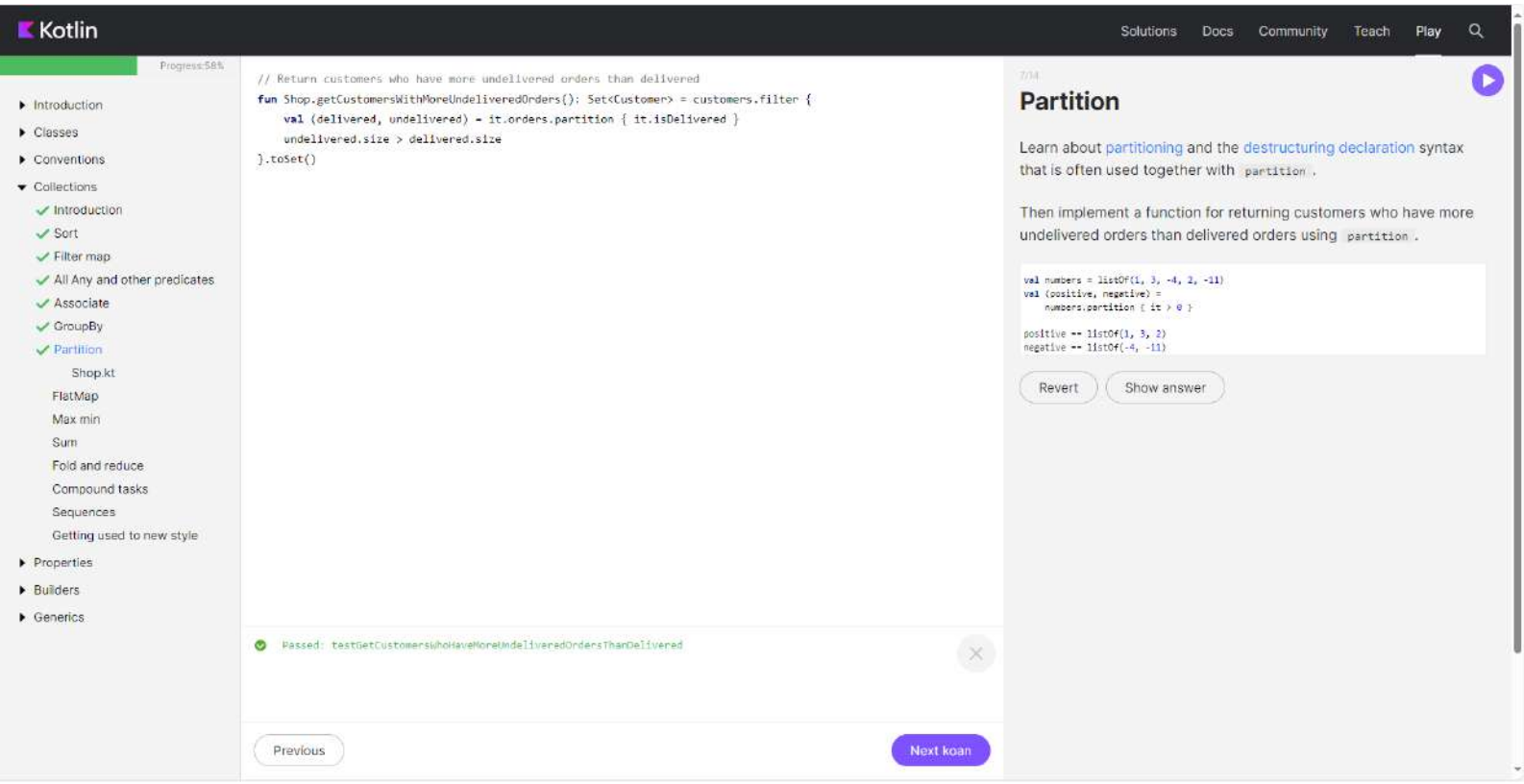

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:47%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Shop.kt
Filter map
All Any and other predicates
Associate
GroupBy
Partition
FlatMap
Max min
Sum
Fold and reduce
Compound tasks
Sequences
Getting used to new style
Properties
Builders
Generics
// 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 }
Passed: testGetCustomersSortedByNumberOfOrders
Previous
Next koan
2/14
Sort
Learn about collection ordering and the the difference between operations in-place on mutable collections and operations returning new collections.
Implement a function for returning the list of customers, sorted in descending order by the number of orders they have made. Use sortedDescending or sortedByDescending.
val strings = listOf("bbb", "a", "cc")
strings.sorted() ==
listOf("a", "bbb", "cc")
strings.sortedBy { it.length } ==
listOf("a", "cc", "bbb")
strings.sortedDescending() ==
listOf("cc", "bbb", "a")
strings.sortedByDescending { it.length } ==
listOf("bbb", "cc", "a")
Revert
Show answer

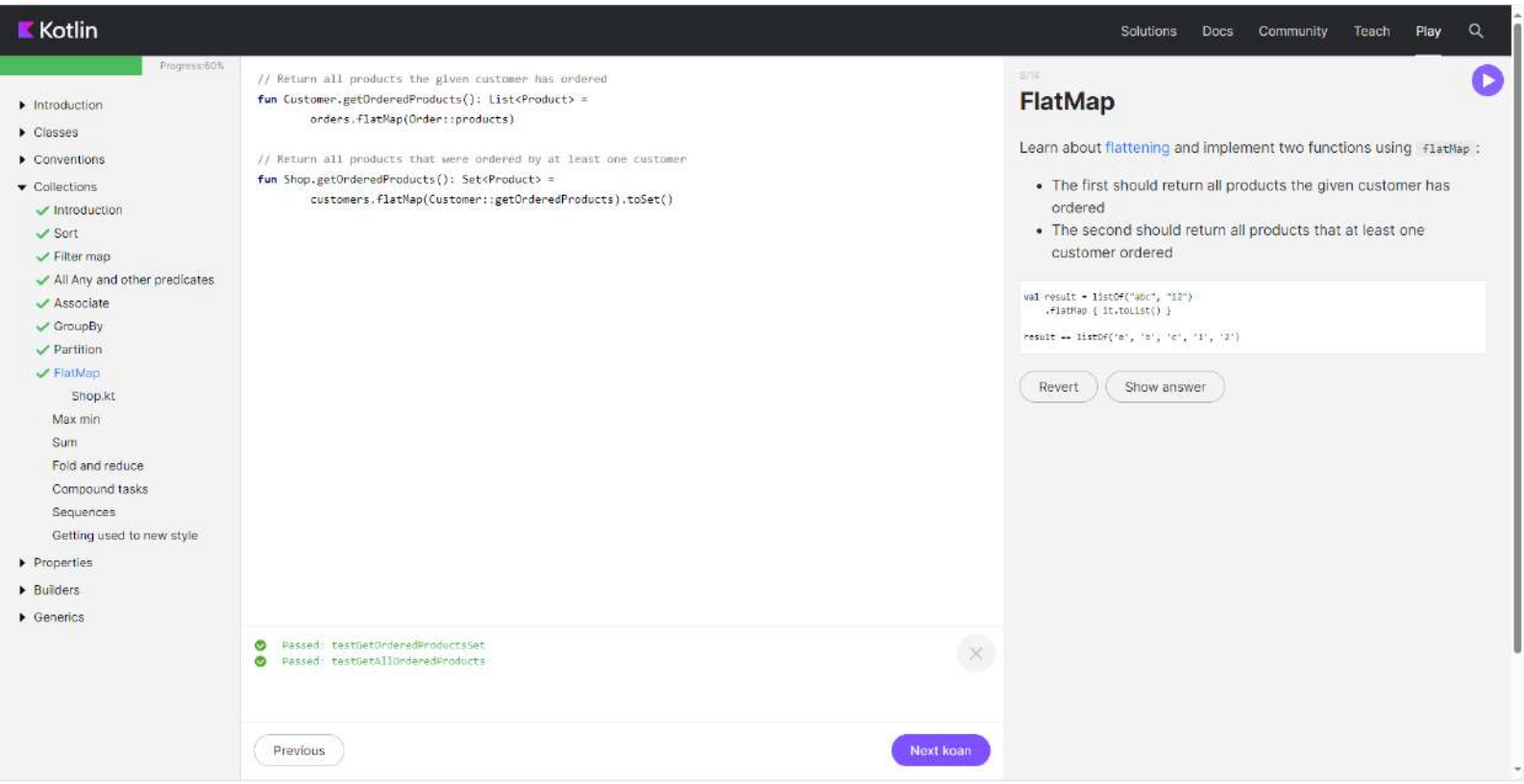

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:48%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Filter map
Shop.kt
All Any and other predicates
Associate
GroupBy
Partition
FlatMap
Max min
Sum
Fold and reduce
Compound tasks
Sequences
Getting used to new style
Properties
Builders
Generics
// 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 }
Passed: testCitiesCustomersAreFrom
Passed: testCustomersFromCity
Previous
Next koan
2/14
Filter; map
Learn about mapping and filtering a collection.
Implement the following extension functions using the map and filter functions:
Find all the different cities the customers are from
Find the customers living in a given city
val numbers = listOf(1, -1, 2)
numbers.filter { it > 0 } == listOf(1, 2)
numbers.map { it * it } == listOf(1, 1, 4)
Revert
Show answer

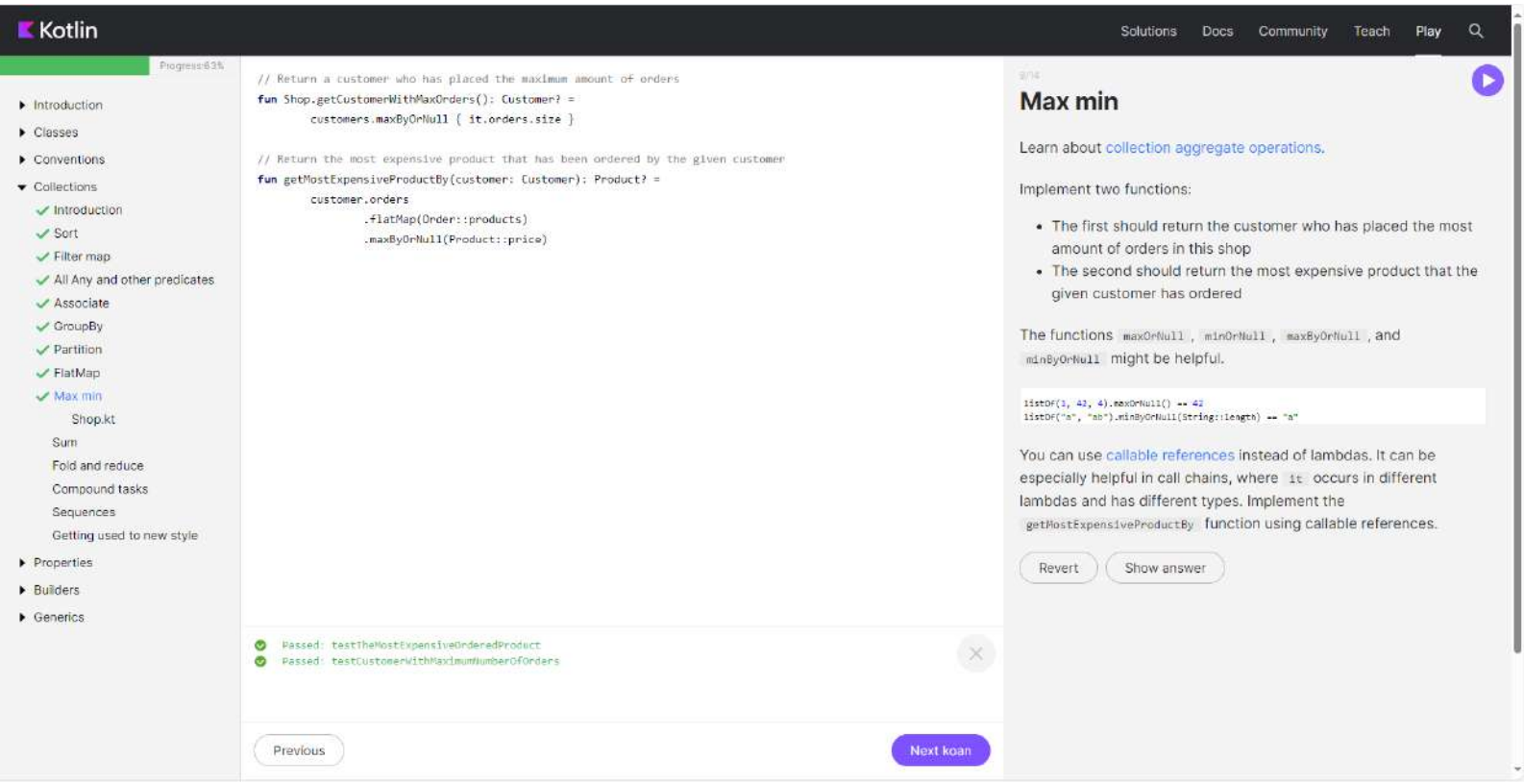

Kotlin
Solutions
Docs
Community
Teach
Play
Progress: 51%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Filter map
All Any and other predicates
Shop.kt
Associate
GroupBy
Partition
FlatMap
Max min
Sum
Fold and reduce
Compound tasks
Sequences
Getting used to new style
Properties
Builders
Generics
// 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 }
Passed: testAnyCustomerFromCity
Passed: testAnyCustomerIsFromCity
Passed: testCountCustomersFromCity
Passed: testAllCustomersAreFromCity
Previous
Next koan
4/14
All, Any, and other predicates
Learn about testing predicates and retrieving elements by condition.
Implement the following functions using all , any , count , find :
checkAllCustomersAreFrom should return true if all customers are from a given city
hasCustomerFrom should check if there is at least one customer from a given city
countCustomersFrom should return the number of customers from a given city
findCustomerFrom should return a customer who lives in a given city, or null if there is none
val numbers = listOf(-1, 0, 2)
val isZero: (Int) -> Boolean = { it == 0 }
numbers.any(isZero) == true
numbers.all(isZero) == false
numbers.count(isZero) == 1
numbers.find { it > 0 } == 2
Revert
Show answer

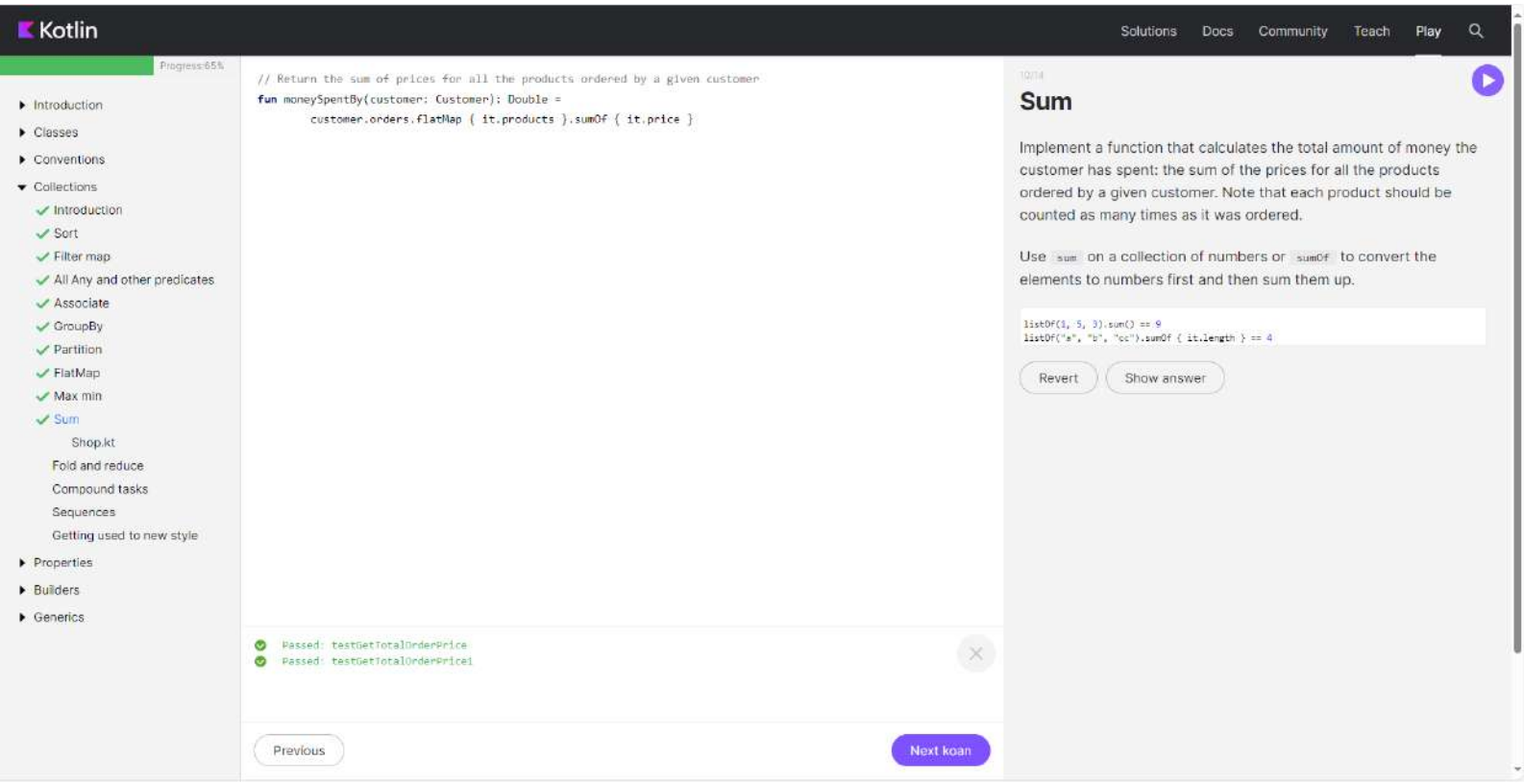

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:53%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Filter map
All Any and other predicates
Associate
Shop.kt
GroupBy
Partition
FlatMap
Max min
Sum
Fold and reduce
Compound tasks
Sequences
Getting used to new style
Properties
Builders
Generics
// 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 }
Passed: testAssociate
Passed: testAssociateBy
Passed: testAssociateWith
Previous
Next koan
5/14
Associate
Learn about association. Implement the following functions using associateBy , associateWith , and associate :
Build a map from the customer name to the customer
Build a map from the customer to their city
Build a map from the customer name to their city
val list = listOf("abc", "cdef")
list.associateBy { it.first() } ==
mapOf('a' to "abc", 'c' to "cdef")
list.associateWith { it.length } ==
mapOf("abc" to 3, "cdef" to 4)
list.associate { it.first() to it.length } ==
mapOf('a' to 3, 'c' to 4)
Revert
Show answer

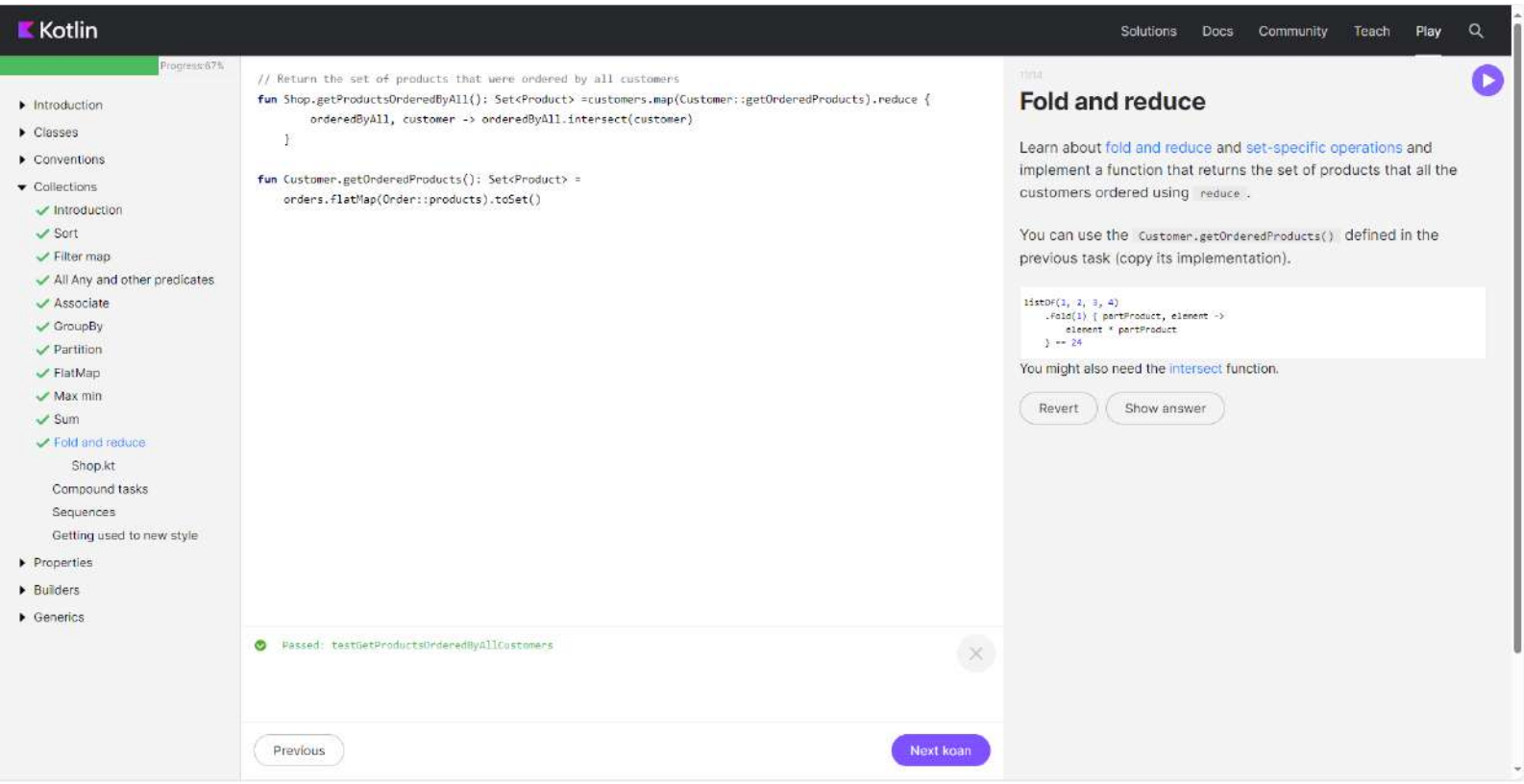

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:56%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Filter map
All Any and other predicates
Associate
GroupBy
Shop.kt
Partition
FlatMap
Max min
Sum
Fold and reduce
Compound tasks
Sequences
Getting used to new style
Properties
Builders
Generics
// Build a map that stores the customers living in a given city
fun Shop.groupCustomersByCity(): Map<City, List<Customer>> =
customers.groupBy { it.city }
Passed: testGroupCustomersByCity
Previous
Next koan
6/14
Group By
Learn about grouping. Use groupBy to implement the function to build a map that stores the customers living in a given city.
val result =
listOf("a", "b", "ba", "ccc", "ad")
.groupBy { it.length }
result == mapOf(
1 to listOf("a", "b"),
2 to listOf("ba", "ad"),
3 to listOf("ccc"))
Revert
Show answer

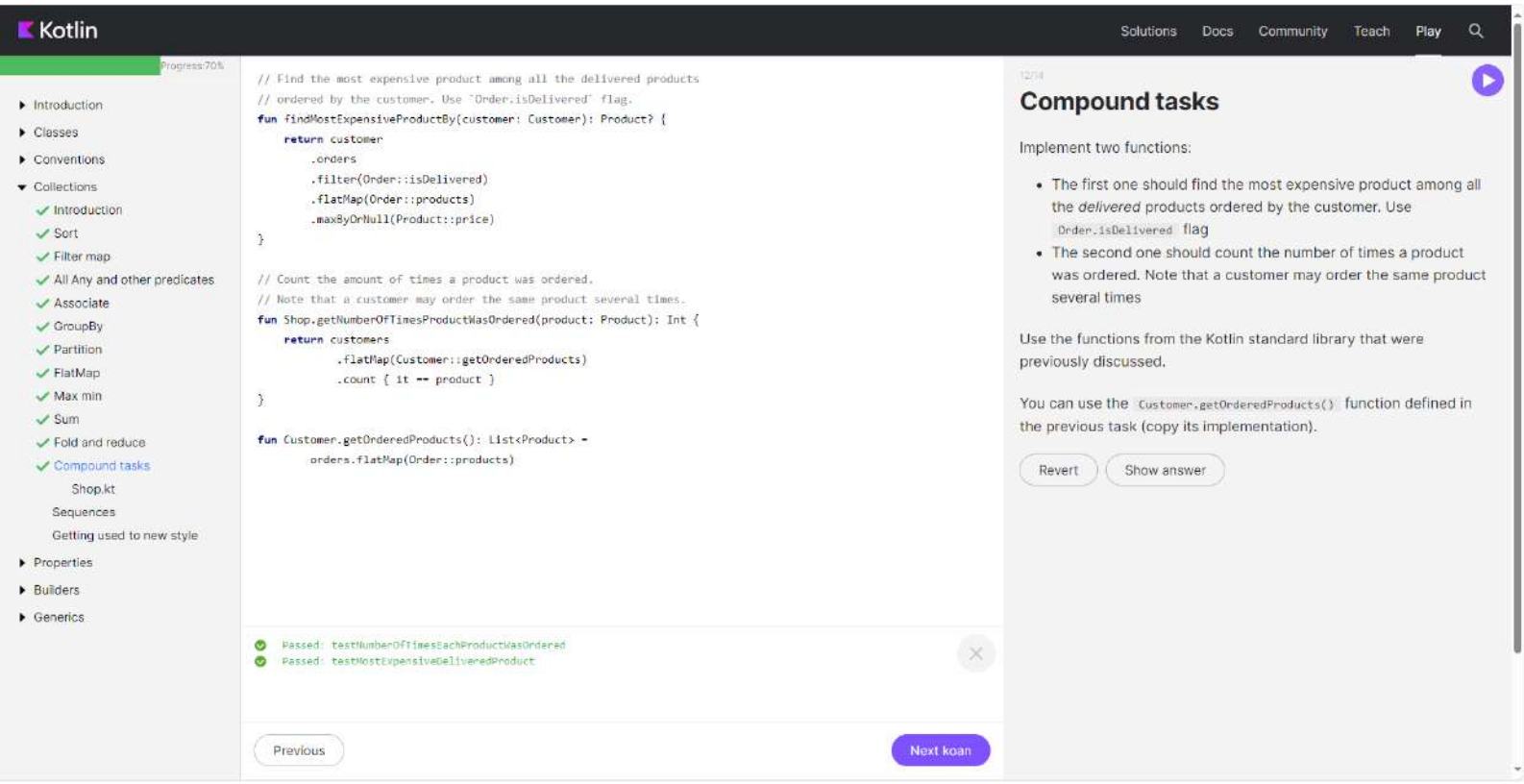

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:58%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Filter map
All Any and other predicates
Associate
GroupBy
Partition
Shop.kt
FlatMap
Max min
Sum
Fold and reduce
Compound tasks
Sequences
Getting used to new style
Properties
Builders
Generics
// 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()
Passed: testGetCustomersWhoHaveMoreUndeliveredOrdersThanDelivered
Previous
Next koan
7/14
Partition
Learn about partitioning and the destructuring declaration syntax that is often used together with partition.
Then implement a function for returning customers who have more undelivered orders than delivered orders using partition.
val numbers = listOf(1, 3, -4, 2, -11)
val (positive, negative) =
numbers.partition { it > 0 }
positive == listOf(1, 3, 2)
negative == listOf(-4, -11)
Revert
Show answer

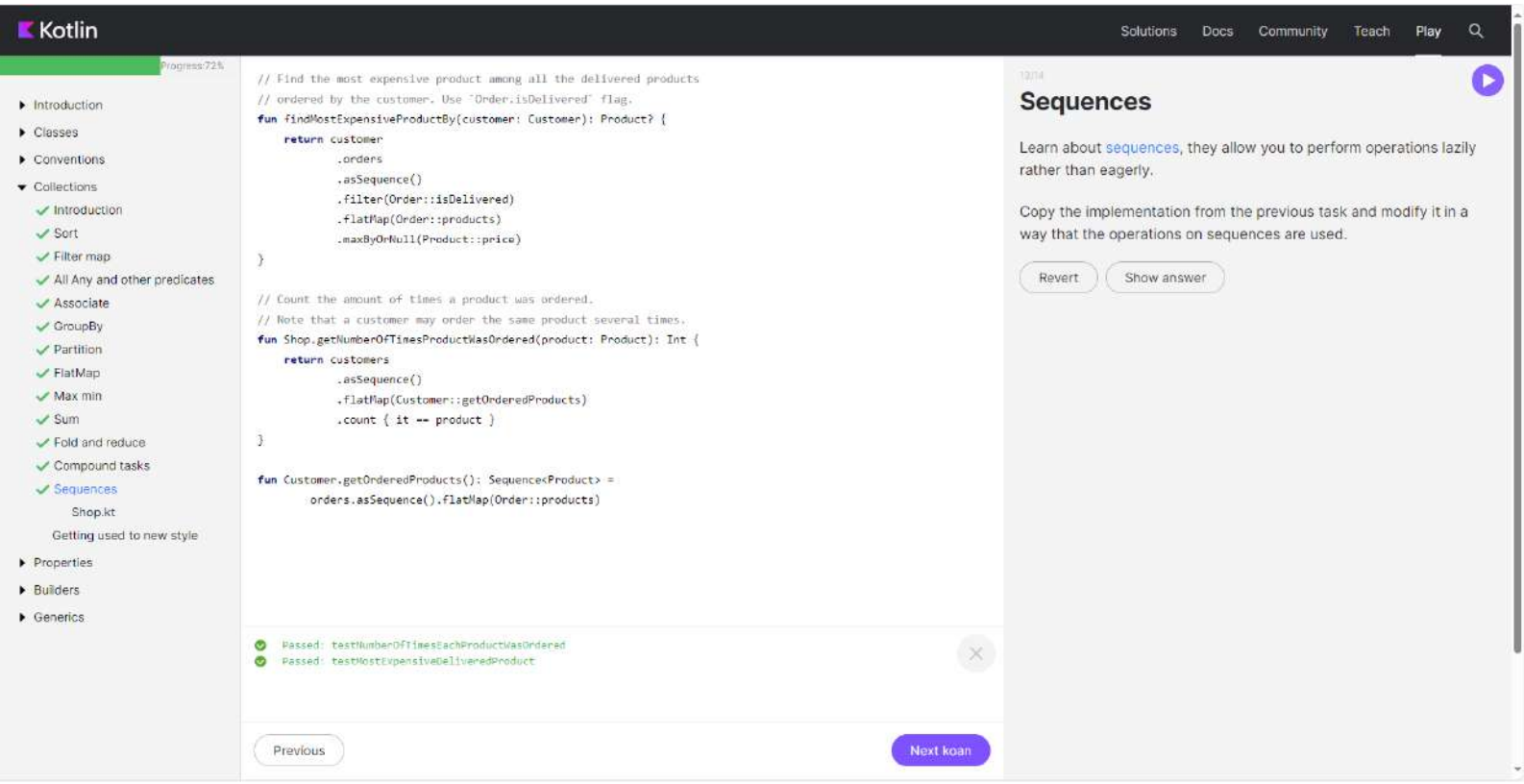

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:60%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Filter map
All Any and other predicates
Associate
GroupBy
Partition
FlatMap
Shop.kt
Max min
Sum
Fold and reduce
Compound tasks
Sequences
Getting used to new style
Properties
Builders
Generics
// 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()
Passed: testGetOrderedProductsSet
Passed: testGetAllOrderedProducts
Previous
Next koan
8/14
FlatMap
Learn about flattening and implement two functions using flatMap :
The first should return all products the given customer has ordered
The second should return all products that at least one customer ordered
val result = listOf("abc", "12")
.flatMap { it.toList() }
result == listOf('a', 'b', 'c', '1', '2')
Revert
Show answer

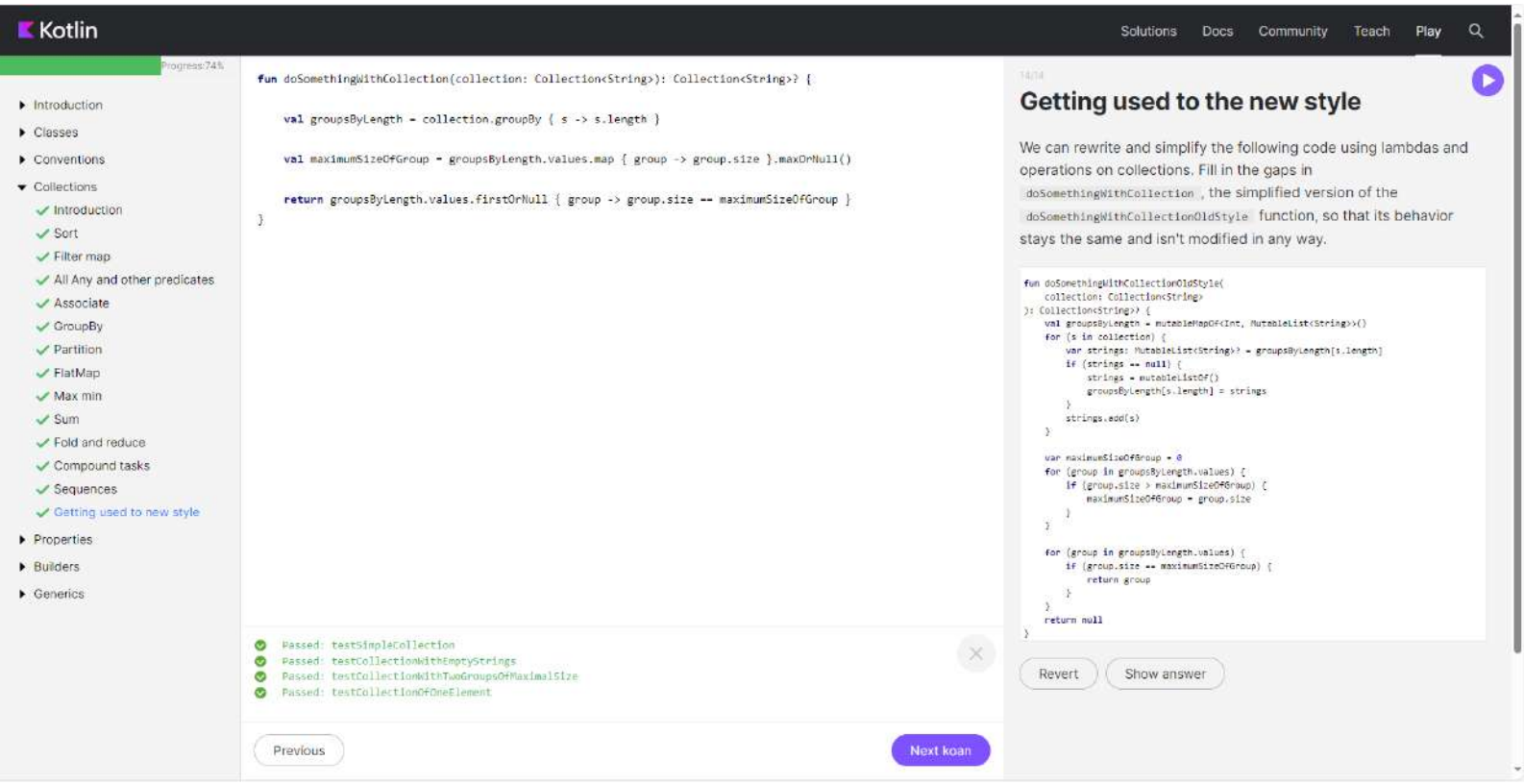

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:63%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Filter map
All Any and other predicates
Associate
GroupBy
Partition
FlatMap
Max min
Shop.kt
Sum
Fold and reduce
Compound tasks
Sequences
Getting used to new style
Properties
Builders
Generics
// 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)
Passed: testTheMostExpensiveOrderedProduct
Passed: testCustomerWithMaximumNumberOfOrders
Previous
Next koan
9/14
Max min
Learn about collection aggregate operations.
Implement two functions:
The first should return the customer who has placed the most amount of orders in this shop
The second should return the most expensive product that the given customer has ordered
The functions maxOrNull , minOrNull , maxByOrNull , and minByOrNull might be helpful.
listOf(1, 42, 4).maxOrNull() == 42
listOf("a", "ab").minByOrNull(String::length) == "a"
You can use callable references instead of lambdas. It can be especially helpful in call chains, where it occurs in different lambdas and has different types. Implement the getMostExpensiveProductBy function using callable references.
Revert
Show answer

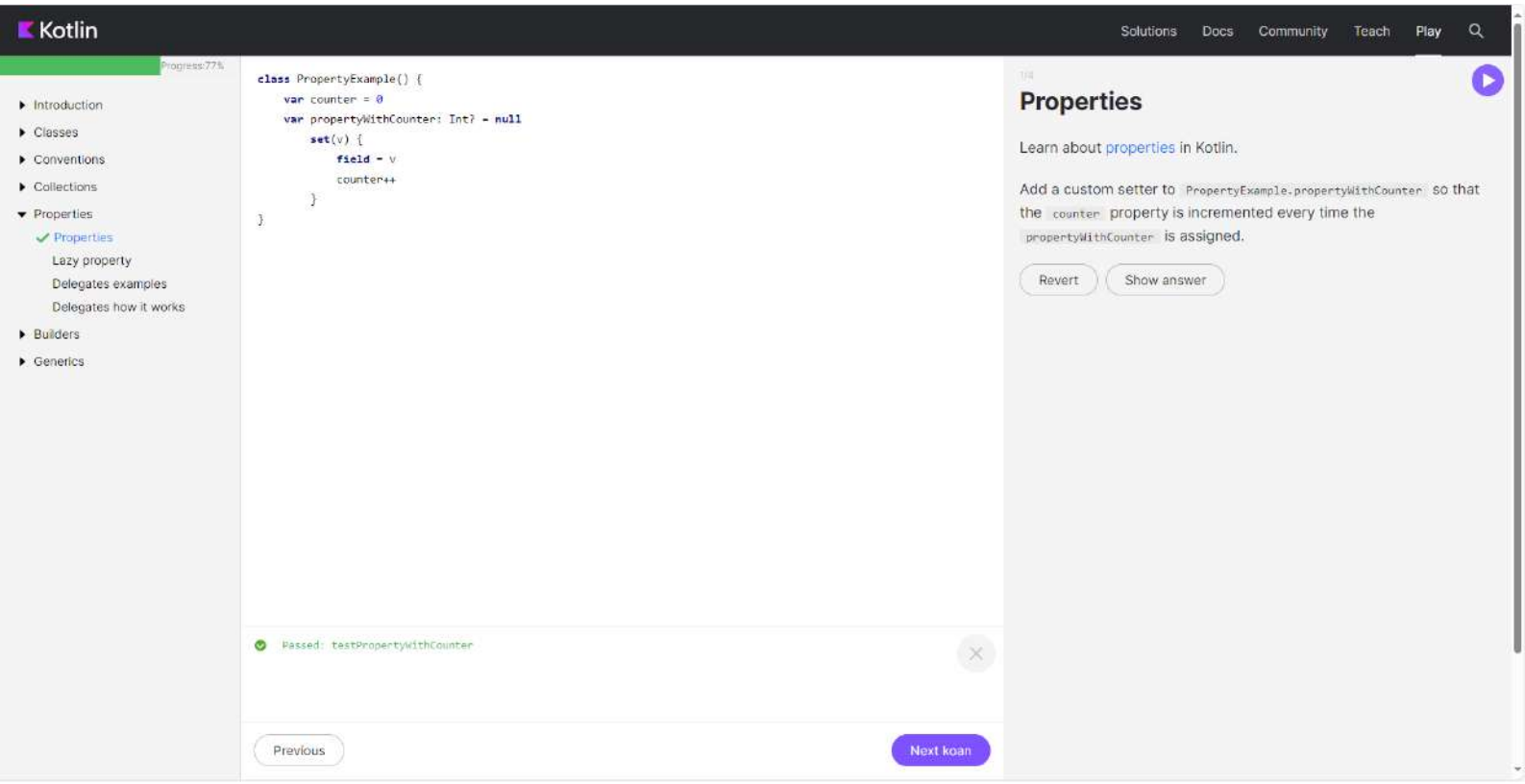

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:65%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Filter map
All Any and other predicates
Associate
GroupBy
Partition
FlatMap
Max min
Sum
Shop.kt
Fold and reduce
Compound tasks
Sequences
Getting used to new style
Properties
Builders
Generics
// 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 }
Passed: testGetTotalOrderPrice
Passed: testGetTotalOrderPrice1
Previous
Next koan
10/14
Sum
Implement a function that calculates the total amount of money the customer has spent: the sum of the prices for all the products ordered by a given customer. Note that each product should be counted as many times as it was ordered.
Use sum on a collection of numbers or sumOf to convert the elements to numbers first and then sum them up.
listOf(1, 5, 3).sum() == 9
listOf("a", "b", "cc").sumOf { it.length } == 4
Revert
Show answer

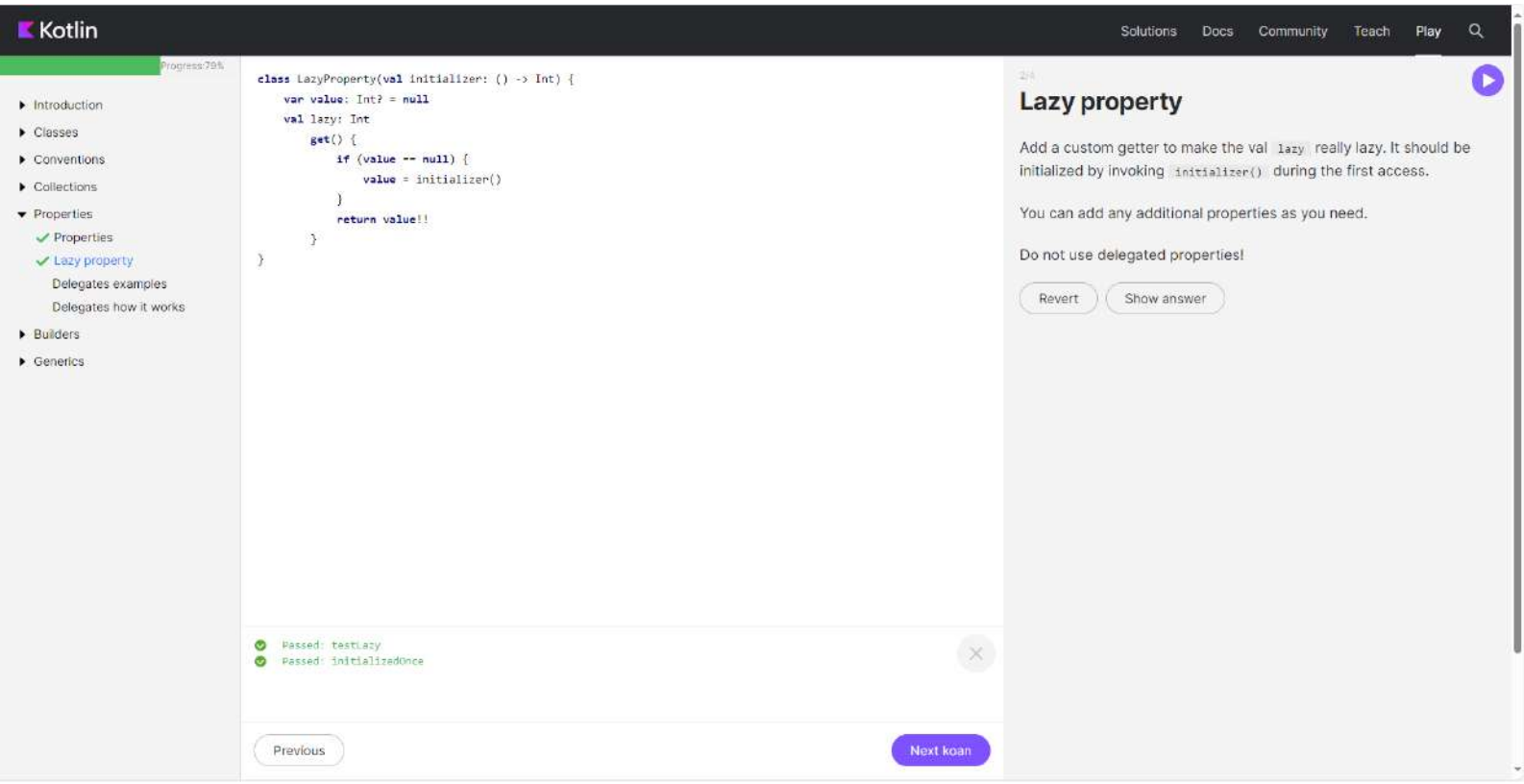

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:67%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Filter map
All Any and other predicates
Associate
GroupBy
Partition
FlatMap
Max min
Sum
Fold and reduce
Shop.kt
Compound tasks
Sequences
Getting used to new style
Properties
Builders
Generics
// 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()
Passed: testGetProductsOrderedByAllCustomers
Previous
Next koan
Fold and reduce
Learn about fold and reduce and set-specific operations and implement a function that returns the set of products that all the customers ordered using reduce .
You can use the Customer.getOrderedProducts() defined in the previous task (copy its implementation).
listOf(1, 2, 3, 4)
.fold(1) { partProduct, element ->
element * partProduct
} == 24
You might also need the intersect function.
Revert
Show answer

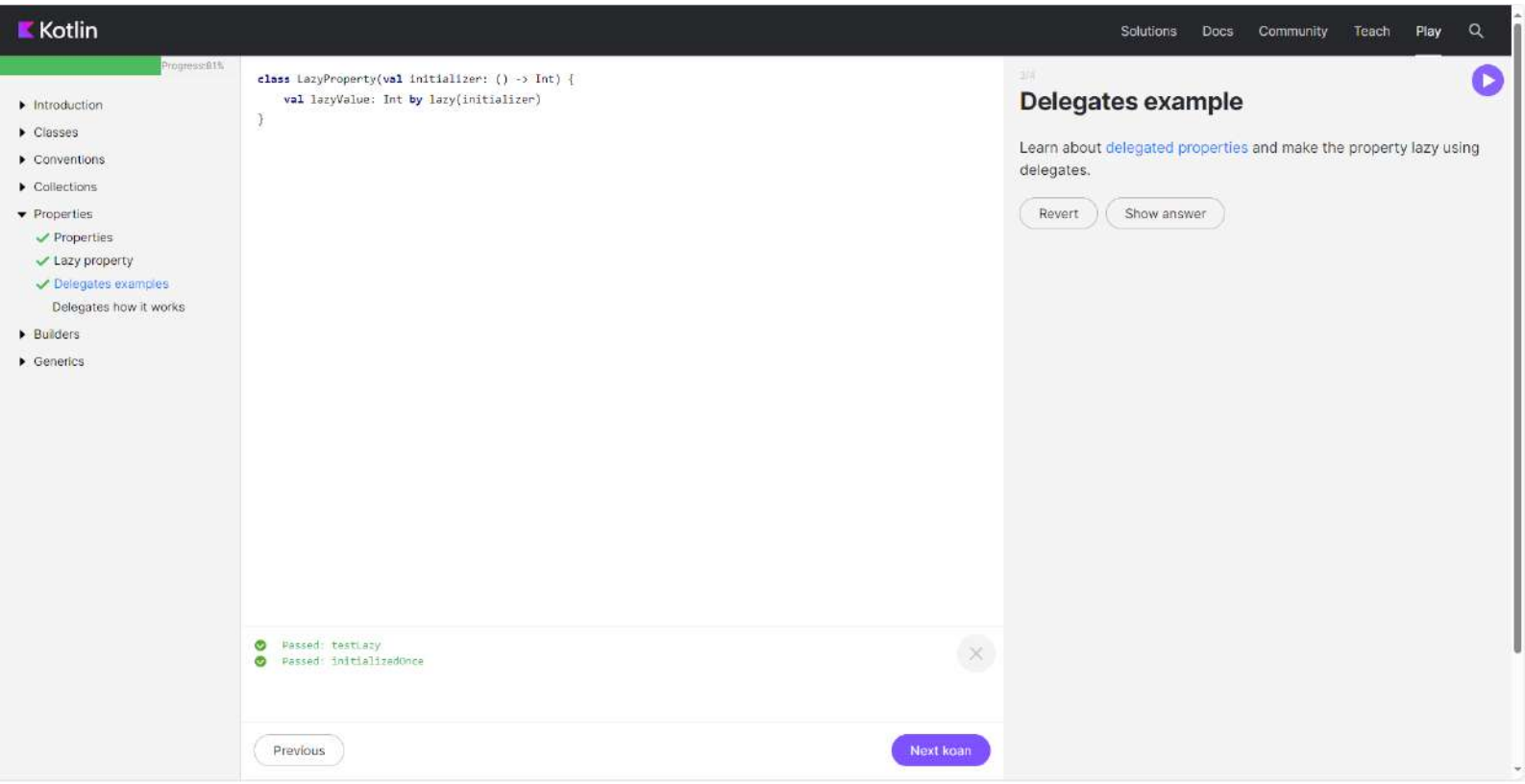

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:70%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Filter map
All Any and other predicates
Associate
GroupBy
Partition
FlatMap
Max min
Sum
Fold and reduce
Compound tasks
Shop.kt
Sequences
Getting used to new style
Properties
Builders
Generics
// 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)
Passed: testNumberOfTimesEachProductWasOrdered
Passed: testMostExpensiveDeliveredProduct
Previous
Next koan
12/14
Compound tasks
Implement two functions:
The first one should find the most expensive product among all the delivered products ordered by the customer. Use Order.isDelivered flag
The second one should count the number of times a product was ordered. Note that a customer may order the same product several times
Use the functions from the Kotlin standard library that were previously discussed.
You can use the Customer.getOrderedProducts() function defined in the previous task (copy its implementation).
Revert
Show answer

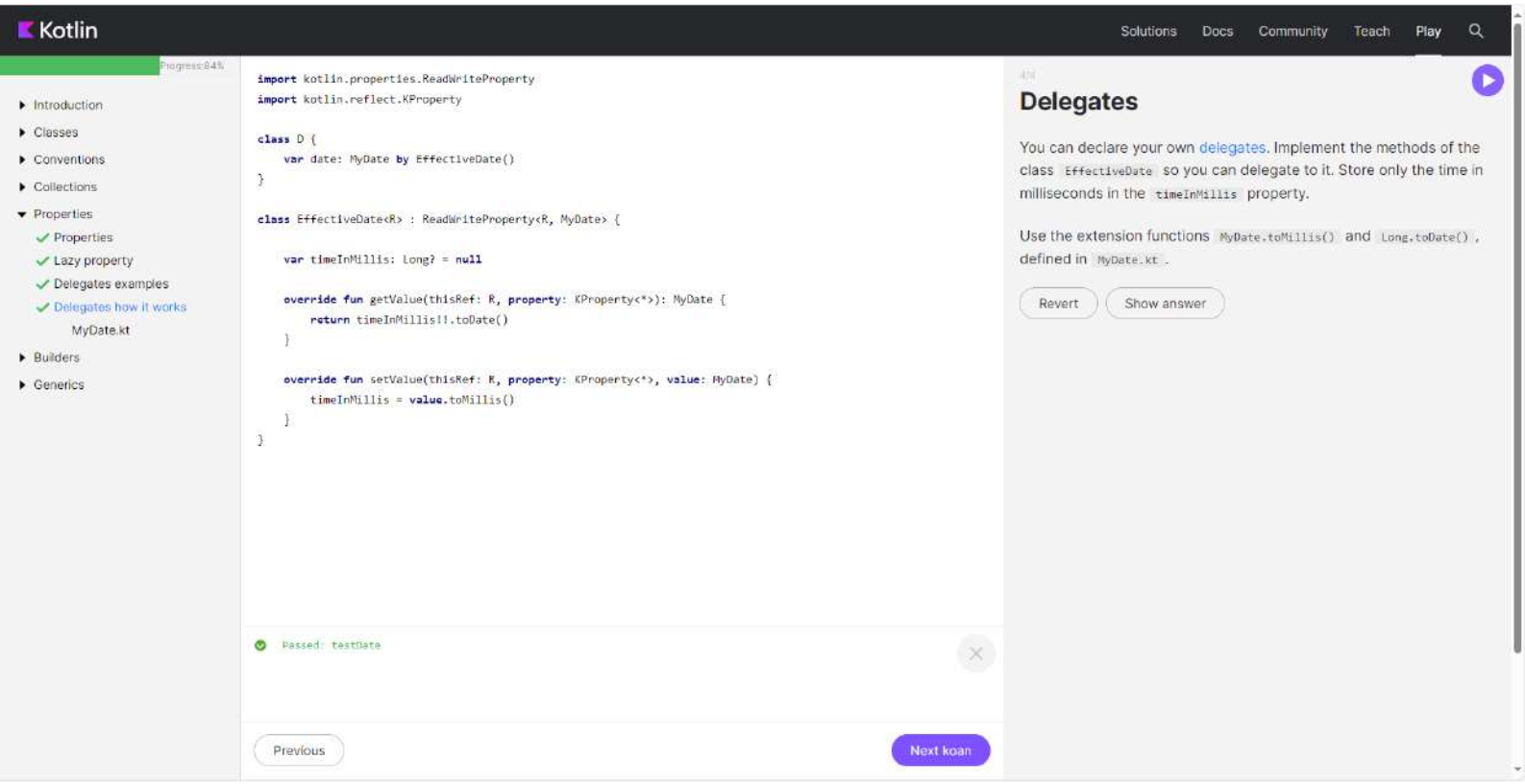

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:72%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Filter map
All Any and other predicates
Associate
GroupBy
Partition
FlatMap
Max min
Sum
Fold and reduce
Compound tasks
Sequences
Shop.kt
Getting used to new style
Properties
Builders
Generics
// 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)
Passed: testNumberOfTimesEachProductWasOrdered
Passed: testMostExpensiveDeliveredProduct
Previous
Next koan
12/14
Sequences
Learn about sequences, they allow you to perform operations lazily rather than eagerly.
Copy the implementation from the previous task and modify it in a way that the operations on sequences are used.
Revert
Show answer

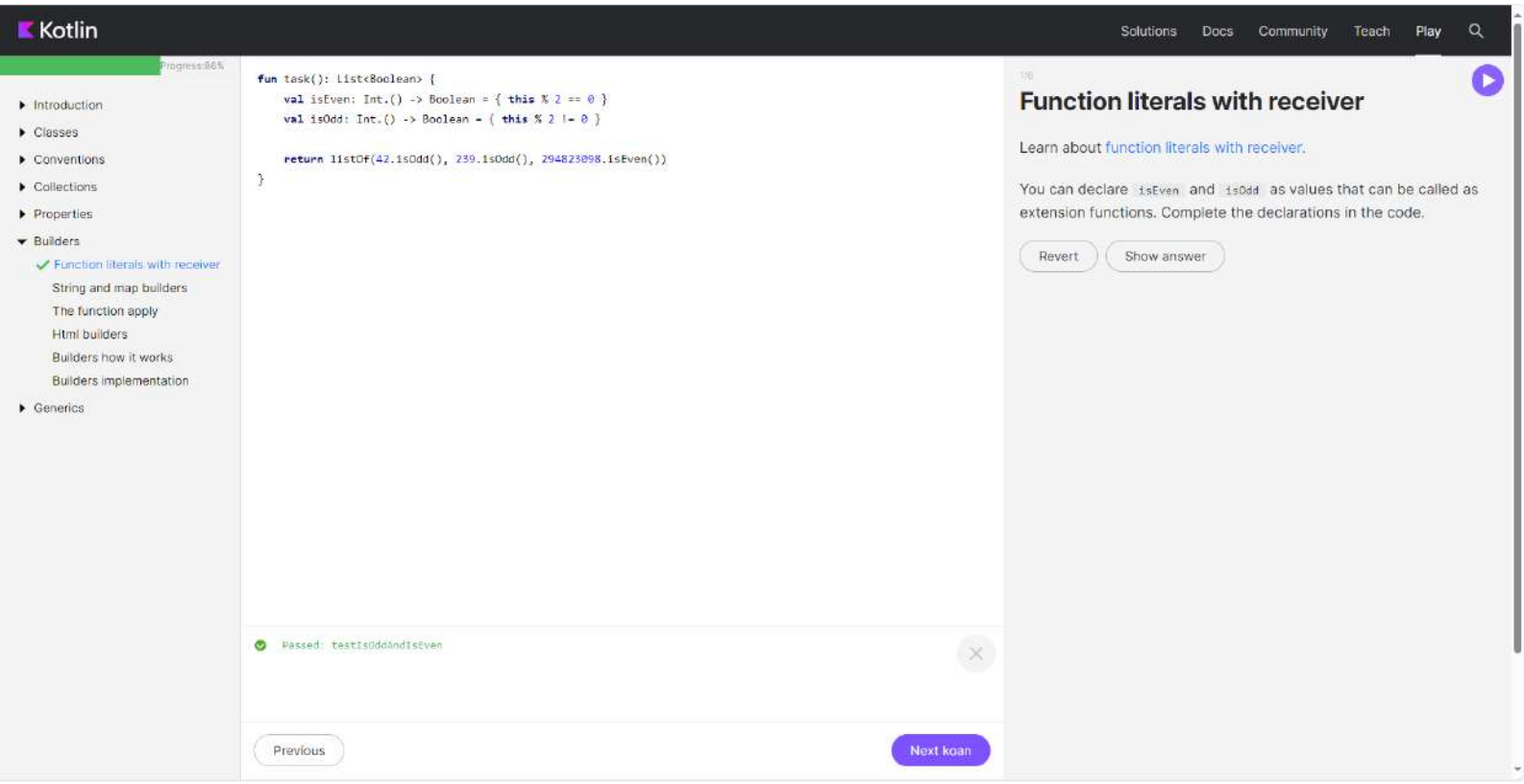

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:74%
Introduction
Classes
Conventions
Collections
Introduction
Sort
Filter map
All Any and other predicates
Associate
GroupBy
Partition
FlatMap
Max min
Sum
Fold and reduce
Compound tasks
Sequences
Getting used to new style
Properties
Builders
Generics
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 }
}
Passed: testSimpleCollection
Passed: testCollectionWithEmptyStrings
Passed: testCollectionWithTwoGroupsOfMaximalSize
Passed: testCollectionOfOneElement
Previous
Next koan
14/14
Getting used to the new style
We can rewrite and simplify the following code using lambdas and operations on collections. Fill in the gaps in doSomethingWithCollection, the simplified version of the doSomethingWithCollectionOldStyle function, so that its behavior stays the same and isn't modified in any way.
fun doSomethingWithCollectionOldStyle(
collection: Collection<String>
): Collection<String>? {
val groupsByLength = mutableMapOf<Int, MutableList<String>>()
for (s in collection) {
var strings: MutableList<String>? = groupsByLength[s.length]
if (strings == null) {
strings = mutableListOf()
groupsByLength[s.length] = strings
}
strings.add(s)
}
var maximumSizeOfGroup = 0
for (group in groupsByLength.values) {
if (group.size > maximumSizeOfGroup) {
maximumSizeOfGroup = group.size
}
}
for (group in groupsByLength.values) {
if (group.size == maximumSizeOfGroup) {
return group
}
}
return null
}
Revert
Show answer

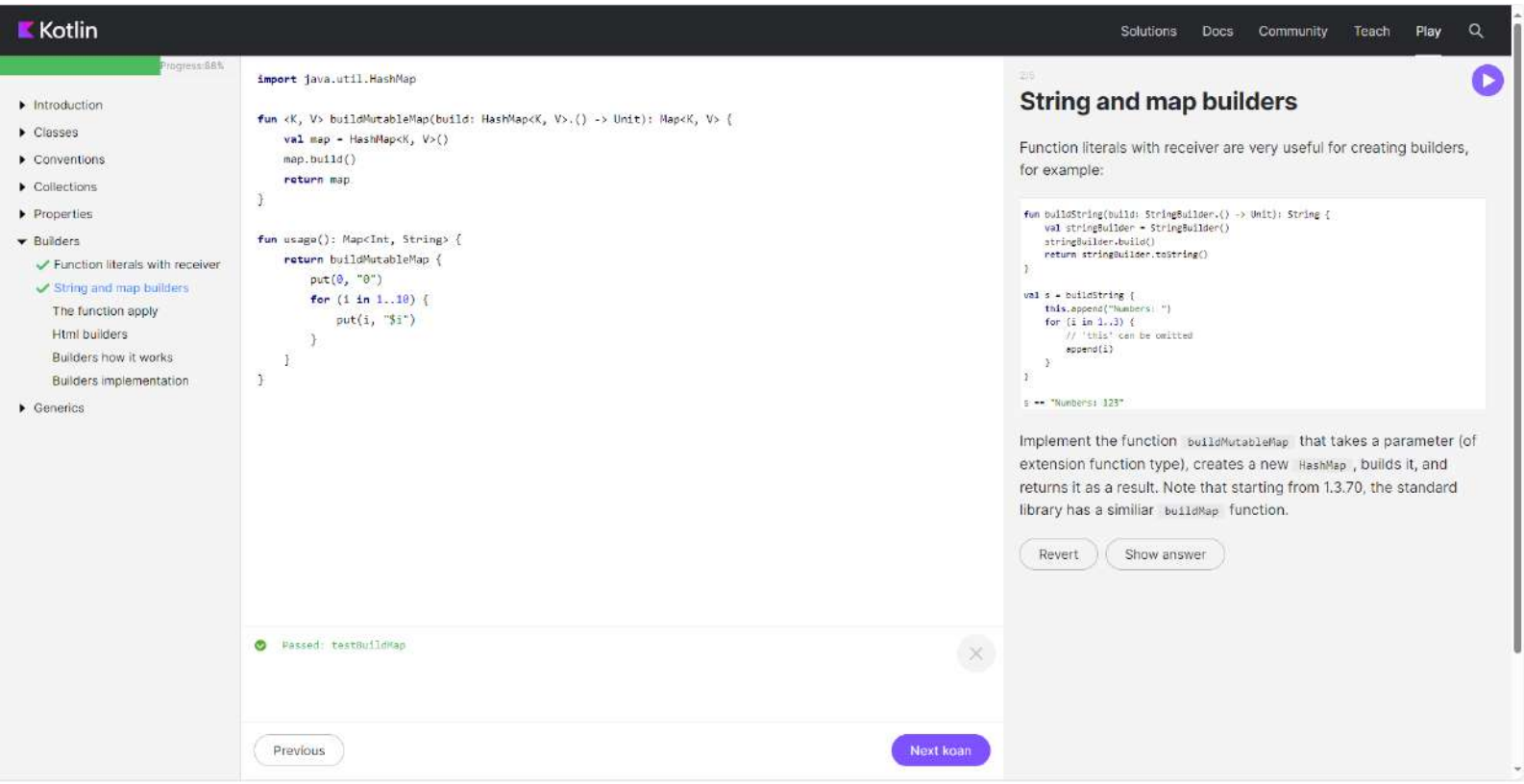

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:77%
Introduction
Classes
Conventions
Collections
Properties
Properties
Lazy property
Delegates examples
Delegates how it works
Builders
Generics
class PropertyExample() {
var counter = 0
var propertyWithCounter: Int? = null
set(v) {
field = v
counter++
}
}
Passed: testPropertyWithCounter
Previous
Next koan
Properties
Learn about properties in Kotlin.
Add a custom setter to PropertyExample.propertyWithCounter so that the counter property is incremented every time the propertyWithCounter is assigned.
Revert
Show answer

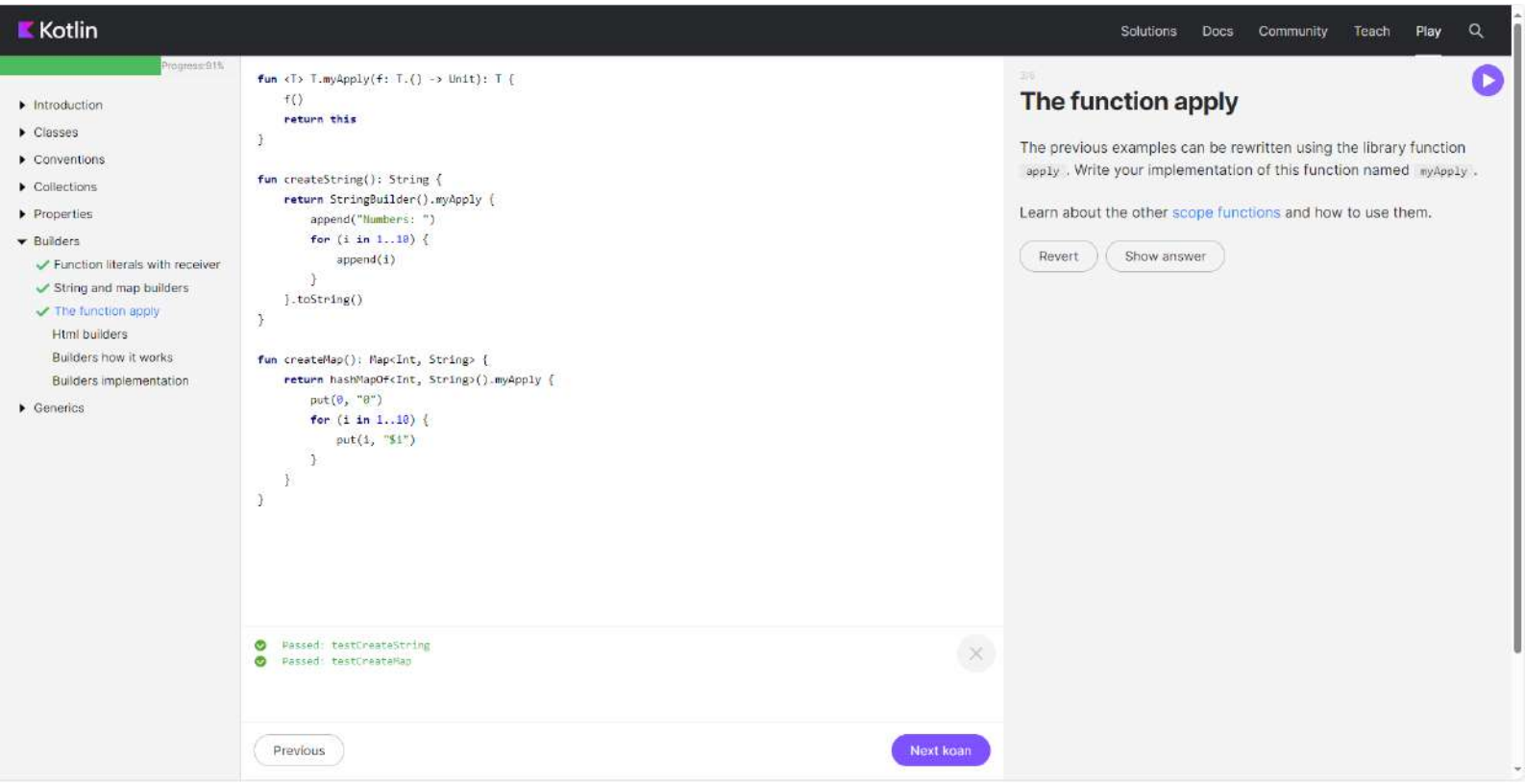

Kotlin
Solutions
Docs
Community
Teach
Play
Progress:79%
Introduction
Classes
Conventions
Collections
Properties
Properties
Lazy property
Delegates examples
Delegates how it works
Builders
Generics
class LazyProperty(val initializer: () -> Int) {
var value: Int? = null
val lazy: Int
get() {
if (value == null) {
value = initializer()
}
return value!!
}
}
Passed: testLazy
Passed: initializedOnce
Previous
Next koan
2/4
Lazy property
Add a custom getter to make the val lazy really lazy. It should be initialized by invoking initializer() during the first access.
You can add any additional properties as you need.
Do not use delegated properties!
Revert
Show answer


Kotlin
Solutions
Docs
Community
Teach
Play
Progress: 61%
Introduction
Classes
Conventions
Collections
Properties
Properties
Lazy property
Delegates examples
Delegates how it works
Builders
Generics
class LazyProperty(val initializer: () -> Int) {
val lazyValue: Int by lazy(initializer)
}
Passed: testLazy
Passed: initializedOnce
Previous
Next koan
2/4
Delegates example
Learn about delegated properties and make the property lazy using delegates.
Revert
Show answer


Kotlin
Solutions
Docs
Community
Teach
Play
Progress:84%
Introduction
Classes
Conventions
Collections
Properties
Properties
Lazy property
Delegates examples
Delegates how it works
MyDate.kt
Builders
Generics
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()
}
}
Passed: testDate
Previous
Next koan
Delegates
You can declare your own delegates. Implement the methods of the class EffectiveDate so you can delegate to it. Store only the time in milliseconds in the timeInMillis property.
Use the extension functions MyDate.toMillis() and Long.toDate(), defined in MyDate.kt.
Revert
Show answer


Kotlin
Solutions
Docs
Community
Teach
Play
Progress:86%
Introduction
Classes
Conventions
Collections
Properties
Builders
Function literals with receiver
String and map builders
The function apply
Html builders
Builders how it works
Builders implementation
Generics
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())
}
Passed: testIsOddAndIsEven
Previous
Next koan
Function literals with receiver
Learn about function literals with receiver.
You can declare isEven and isOdd as values that can be called as extension functions. Complete the declarations in the code.
Revert
Show answer


Kotlin
Solutions
Docs
Community
Teach
Play
Progress:88%
Introduction
Classes
Conventions
Collections
Properties
Builders
Function literals with receiver
String and map builders
The function apply
Html builders
Builders how it works
Builders implementation
Generics
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")
}
}
}
Passed: testBuildMap
Previous
Next koan
2/5
String and map builders
Function literals with receiver are very useful for creating builders, for example:
fun buildString(build: StringBuilder.() -> Unit): String {
val stringBuilder = StringBuilder()
stringBuilder.build()
return stringBuilder.toString()
}
val s = buildString {
this.append("Numbers: ")
for (i in 1..3) {
// 'this' can be omitted
append(i)
}
}
s == "Numbers: 123"
Implement the function buildMutableMap that takes a parameter (of extension function type), creates a new HashMap, builds it, and returns it as a result. Note that starting from 1.3.70, the standard library has a similiar buildMap function.
Revert
Show answer


Kotlin
Solutions
Docs
Community
Teach
Play
Progress:61%
Introduction
Classes
Conventions
Collections
Properties
Builders
Function literals with receiver
String and map builders
The function apply
Html builders
Builders how it works
Builders implementation
Generics
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")
}
}
}
Passed: testCreateString
Passed: testCreateMap
Previous
Next koan
The function apply
The previous examples can be rewritten using the library function apply. Write your implementation of this function named myApply.
Learn about the other scope functions and how to use them.
Revert
Show answer

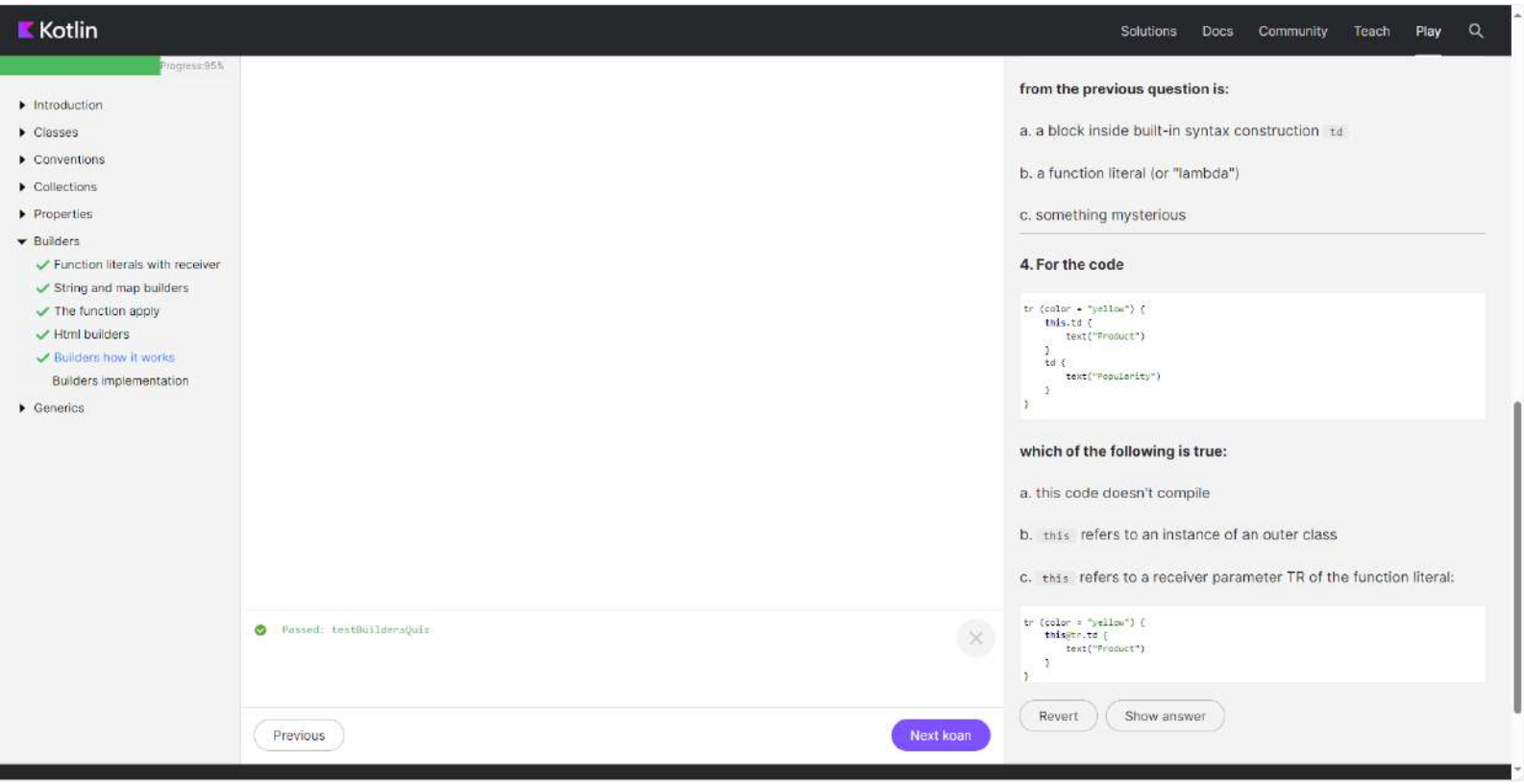
```
                text("Product")
            }
            td {
                text("Price")
            }
            td {
                text("Popularity")
            }
        }
        val products = getProducts()
        for ((index, product) in products.withIndex()) {
            tr {
                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"
```

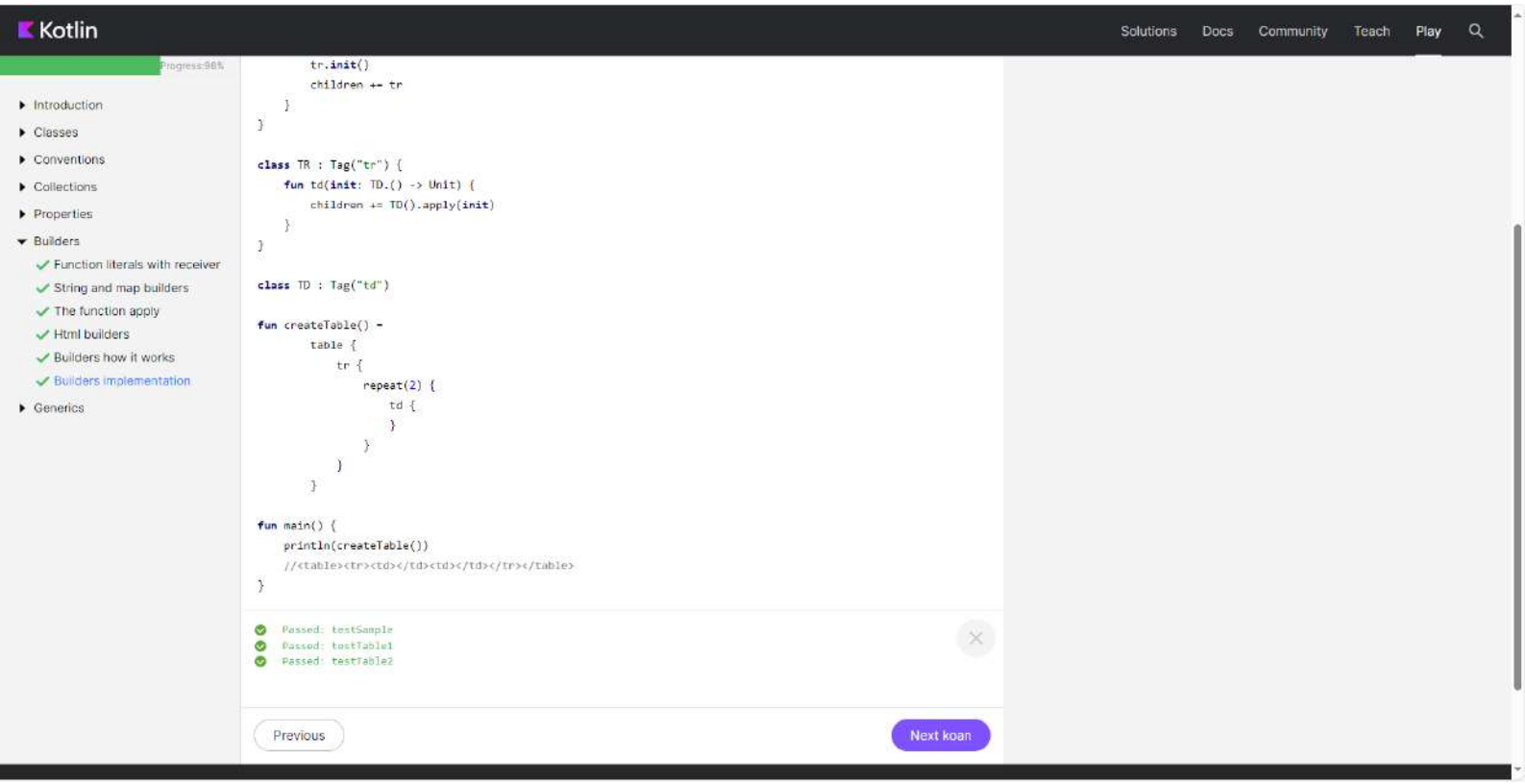
Passed: productTableIsColored
Passed: productTableIsFilled

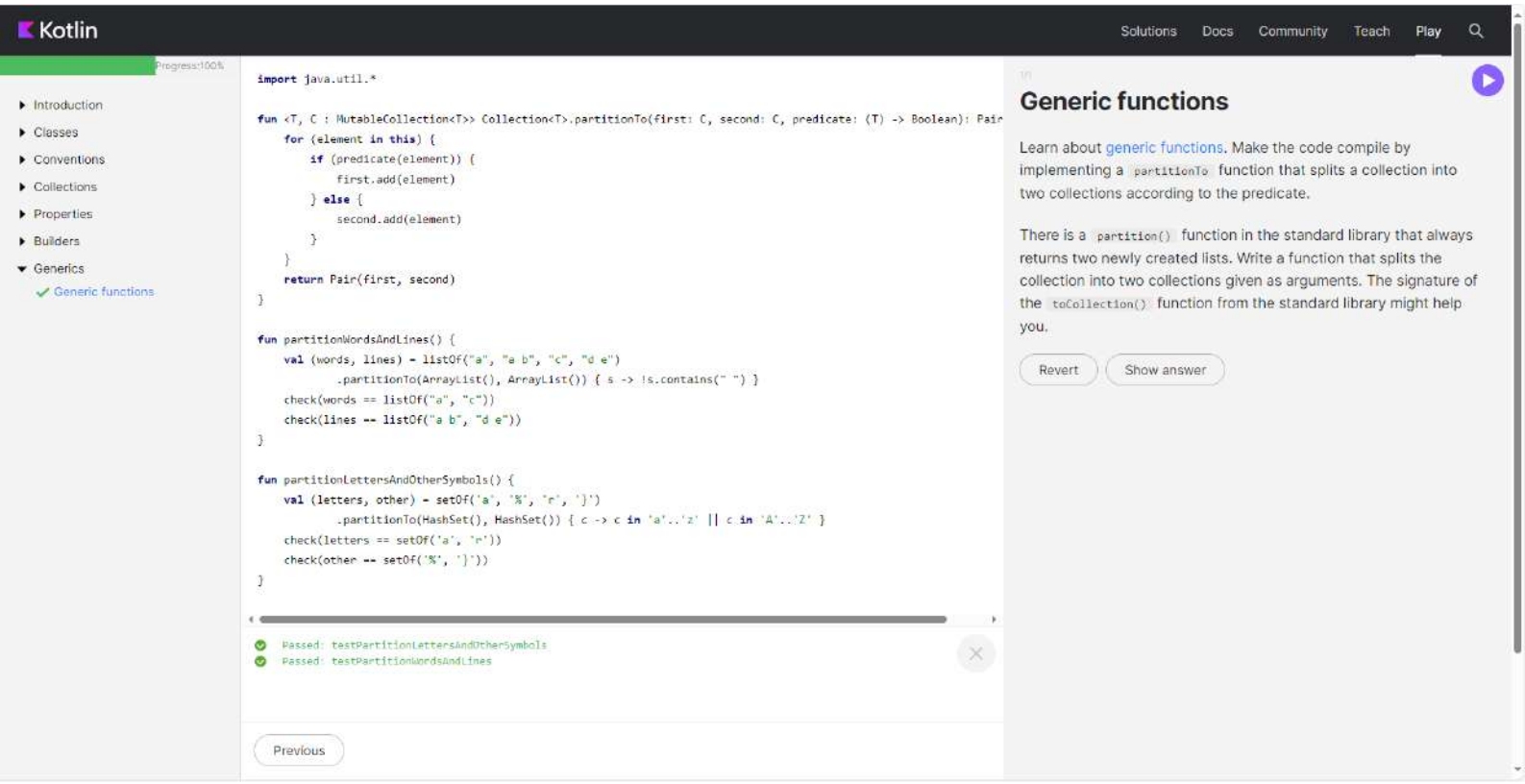
2. Color the table like a chessboard. Use the `getTitleColor()` and `getCellColor()` functions. Pass a color as an argument to the functions `tr`, `td`.

Run the main function defined in the file `demo.kt` to see the rendered table.

Revert Show answer


Kotlin
Solutions
Docs
Community
Teach
Play
Progress:95%
Introduction
Classes
Conventions
Collections
Properties
Builders
Function literals with receiver
String and map builders
The function apply
Html builders
Builders how it works
Builders implementation
Generics
from the previous question is:
a. a block inside built-in syntax construction td
b. a function literal (or "lambda")
c. something mysterious
4. For the code
tr {color = "yellow") {
this.td {
text("Product")
}
td {
text("Popularity")
}
}
which of the following is true:
a. this code doesn't compile
b. this refers to an instance of an outer class
c. this refers to a receiver parameter TR of the function literal:
tr {color = "yellow") {
this@tr.td {
text("Product")
}
}
Passed: testBuildersQuiz
Revert
Show answer
Previous
Next koan


Kotlin
Solutions
Docs
Community
Teach
Play
Progress:96%
Introduction
Classes
Conventions
Collections
Properties
Builders
Function literals with receiver
String and map builders
The function apply
Html builders
Builders how it works
Builders implementation
Generics
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>
}
Passed: testSample
Passed: testTable1
Passed: testTable2
Previous
Next koan


Kotlin
Solutions
Docs
Community
Teach
Play
Progress:100%
Introduction
Classes
Conventions
Collections
Properties
Builders
Generics
Generic functions
import java.util.*
fun <T, C : MutableCollection<T>> Collection<T>.partitionTo(first: C, second: C, predicate: (T) -> Boolean): Pair
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('%', ']'))
}
Passed: testPartitionLettersAndOtherSymbols
Passed: testPartitionWordsAndLines
Previous
Generic functions
Learn about generic functions. Make the code compile by implementing a partitionTo function that splits a collection into two collections according to the predicate.
There is a partition() function in the standard library that always returns two newly created lists. Write a function that splits the collection into two collections given as arguments. The signature of the toCollection() function from the standard library might help you.
Revert
Show answer