Sets

Understand the data structure of sets, how to use them, and how they differ from lists.

WE'LL COVER THE FOLLOWING

Sets are linear data structures very similar to lists. However, they do have a few notable differences.

What is a Set?

Just like a list, a set is a linear data structure that contains multiple elements. However, sets are different from lists because:

- Sets have no concept of ordering, meaning there is no such thing as “the third element” in a set.
- Sets cannot contain duplicates; you cannot have a set that, for instance, contains `42` twice.

Both these properties of sets have implications for programming with them that you should be aware of when using them.

Creating a Set

The way to create a set should look familiar to you:

```
val awards = setOf("World's Best Programmer 2019", "Best Programming Language 2020")
val members = mutableSetOf("Susan", "Jake", "Jenny")
```

The impact of using read-only vs mutable sets is the same as for lists in the previous lesson. Try using read-only sets over mutable sets.

> If you already know your way around data structures, you may want to use any of the following functions to create a particular type of set:
>
> - `hashSetOf(...)`
> - `linkedSetOf(...)`
> - `sortedSetOf(...)`

Accessing Elements in a Set

Because there's no concept of ordering in a set, you *cannot* use an indexed access operator:

```
val members = mutableSetOf("Susan", "Jake", "Jenny")
println(members[0])  // Causes compile-time error
```

When using sets, you normally search for a subset of elements by trait or perform operations on the entire set. Because these concepts require a few more prerequisites, we'll come back to working with sets later.

For now, you may use Kotlin's utility functions such as `mySet.first()` to get an element from a set to try it out.

> **Note:** As per the definition of sets, the "first" element may be any of the values you passed in. You have no guarantee it's the first one you passed in.

After learning about loops in the next section, you'll be able to iterate over any collection to access each element.

Adding and Removing Elements

Adding and removing elements is similar to how you do it with lists:

```
val members = mutableSetOf("Susan", "Jake", "Jenny")
members.add("Greg")
members.remove("Jake")
members.clear()
members.addAll(setOf("Adam", "Eve"))
```

Although you cannot *yet* use sets in many ways, they are fundamental for practical applications. Oftentimes, we're not interested in getting the *i*-th element of a collection. Instead, we want to implement more complex queries on the elements, such as:

- "search all users that have logged in within the past 24h,"
- "find books that have not been returned on time," or
- "what are the permissions required for this action?".

For all of these, sets can be preferable to lists because there's no need for ordering.

Quiz

Mutable and Readonly Sets

1 How are sets *different* from lists? You can select multiple answers.

COMPLETED 0%

1 of 2

Exercises

In the following code widget:

- Create a read-only set of your `favoriteActors`
- Create a mutable set of your `favoriteMovies`
- Try to add a duplicate into one of your sets and see what happens.

Problem Solution

```
// Add your code here
```

Summary

Similar to lists, sets are a linear data structure of elements.

- Sets cannot have duplicates, so adding an existing element has no effect
- Sets have no concept of ordering, so you can't access an element by index
- You can create sets using `setOf` and `mutableSetOf`

Next, you will learn how to use arrays in Kotlin and how they differ from lists and sets as a data structure.