Read-Only vs Mutable Variables

Learn the fundamental principle of read-only and mutable variables, plus how and when to use them in Kotlin.

WE'LL COVER THE FOLLOWING

Let’s start with the very basics: declaring variables. Kotlin fundamentally differentiates between read-only and mutable data.

Mutable Variables

To declare a mutable variable, you use the `var` keyword:

```
var number = 17
println("number = $number")

number = 42  // var can be reassigned
println("number = $number")
```

Mutable means that the variable can be reassigned to a different value after initial assignment.

Read-Only Variables

In contrast, a read-only variable can be declared using `val` (instead of `var`):

```
val number = 17
println("number = $number")


number = 42  // Not allowed, throws an exception
```

Read-only means that the variable cannot be reassigned once initialized.

You should prefer read-only variables to mutable ones whenever possible, i.e., whenever you don’t have to change the value after initialization.

> **Tip**: Prefer `val` to `var` to simplify data flow and facilitate reasoning about your code.

Quiz

Readonly and Mutable Variables

1 Which of the following code snippets correctly initialize readonly variables? You can select multiple answers.

COMPLETED 0%

1 of 3

Exercise

Fill in the gaps in this code to create a mutable variable `age` and a read-only variable `name`:

Problem Solution

```
// Create your variables
```

Summary

Kotlin differentiates between read-only and mutable variables at declaration time, forcing you to think about the mutability of state from the start.

- `val` lets you declare read-only variables.
- `var` lets you declare mutable variables.
- Good practice is to prefer `val` over `var` whenever possible to reduce mutability and therefore facilitate understanding of the program's state and data flow.

In the next lesson, you will learn about Kotlin's fundamental data types and how to use them.