Shorthand Notation for Simple Functions

Learn when and how to write functions in shorthand notation for concise and readable code.

WE'LL COVER THE FOLLOWING

Functions that simply return the result of a single expression can be abbreviated in Kotlin.

Refactoring a Function

Consider the following function:

```
fun isValidUsername(username: String): Boolean {
  if (username.length >= 3) {
    return true
  } else {
    return false
  }
}
```

How would you refactor it? Can you make it more concise and improve its readabilty?

You could improve this in several steps, e.g., by moving out the `return`

keyword and using `if` as an expression. This particular example is even

simpler and can be rewritten as:

```
fun isValidUsername(username: String): Boolean {
  return username.length >= 3
}
```

This is the simplest form this function could take in many other languages such as Java or C++. The function body of `isValidUsername` **consists only of a** `return` **keyword followed by an expression**.

But in Kotlin, you can go one step further.

Introducing Shorthand Functions

For *single-expression functions* like this, Kotlin allows a shorthand notation:

```
fun isValidUsername(username: String): Boolean = username.length >= 3
```

The function body with curly braces is now replaced by an assignment, the same way you would assign a variable. This makes the code more succinct and, once you're used to this syntax, it improves readability by removing boilerplate (curly braces and `return`).

Combining with Type Inference

At this point in the refactoring, you can go one step further and make use of Kotlin's type inference:

```
fun isValidUsername(username: String) = username.length >= 3
```

This only works with shorthand notation. The compiler now infers that the `isValidUsername` function returns a `Boolean` (the same way it can infer it for variables).

> This is not to say you should always remove the explicit return type. You should evaluate if it actually helps readability or not. In this simple example above, it's immediately obvious to developers that it returns a

`Boolean`, even from the function name.

In other cases, it may not be so clear. You should keep in mind that, without an explicit return type, you may accidentally change a function's return type by changing the function's body.

Quiz

Shorthand notation for functions

1 Which of the following functions could be *directly* written in shorthand notation (without further refactoring)? Why?

COMPLETED 0%

1 of 2

Exercises

Fibonacci

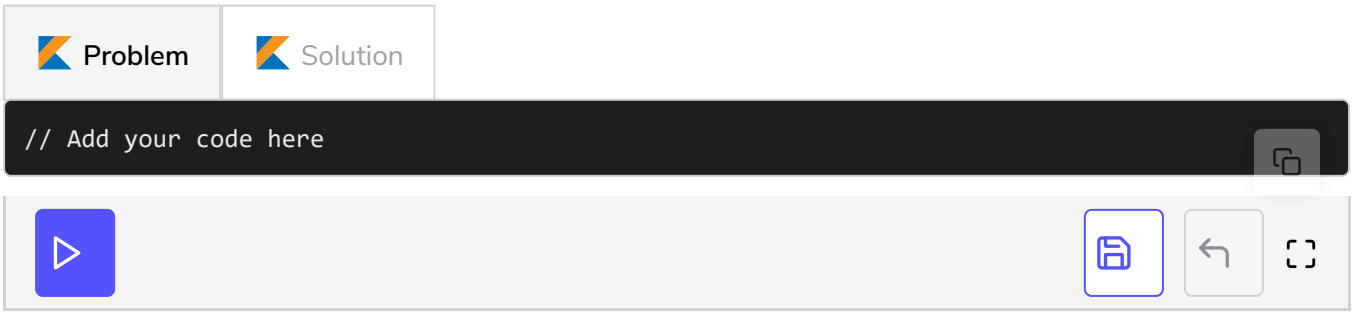
1. Implement a function that returns the *n*-th Fibonacci number.
2. In the next step, compress your function body into a single expression (including the `return` keyword).
3. Finally, transform it into shorthand notation.

4. Call your function with several inputs in a `main` function

(Of course you can do this all at once, but it's easier to go step by step).

GCD

Rewrite your GCD function so that you can fit the entire function into a single readable line of code using shorthand notation.



LCM

Rewrite your LCM function in the same way. You may still reuse your GCD function by pasting it here.

Summary

Kotlin allows a very concise function notation in some cases.

- The function body must contain only a `return` followed by a single expression.
- It can then be defined as `fun foo() = 42`.

    - The return type can be omitted due to type inference.

- The shorthand can improve or hamper readability, depending on the function.

In the following lesson, you'll set default values on function parameters to increase your functions' usability and flexibility.