Basic Functions

Learn how to modularize chunks of code into functions and how to call your functions in Kotlin. Understand the difference between function signatures and function declarations.

WE'LL COVER THE FOLLOWING

Functions separate useful chunks of code into a named entity you can reference in your code. Along with variables, they are the absolute fundamental language construct to avoid code duplication.

Function Signatures

A *function signature* defines a function's name, inputs, and outputs. In Kotlin, it looks like this:

```
fun fibonacci(index: Int): Long
```

This signature defines a function named `fibonacci` with one input parameter named `index` of type `Int` and a return value of type `Long`. In other words, you give the function an integer and get back a `Long`. This function

implements the well-known Fibonacci sequence.

Declaring a Function

In a *function declaration*, both a function signature and a *function body* are required:

```
fun fibonacci(index: Int): Long {
  return if (index < 2) {
    1
  } else {
    fibonacci(index-2) + fibonacci(index-1)  // Calls `fibonacci` recursively
  }
}
```

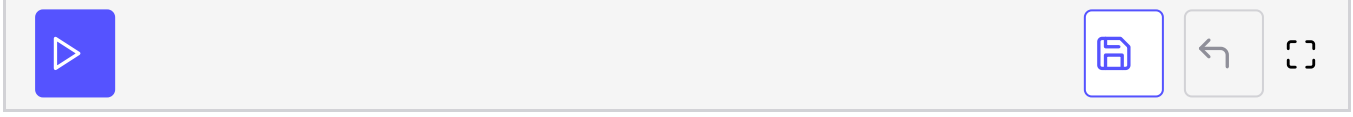
The function body is a code block (thus surrounded by curly braces) and defines what the function does. The `return` keyword defines the return value of the function call.

> **Note:** Notice how the `return` keyword is moved outside of the condition blocks. This is possible because the `if` block is an expression and therefore has a value.

Calling a Function

Once you have declared a function, you can call it by passing in a value for each required input parameter:

```
val output = fibonacci(6)
println(output)
```



This *function call* sets the `index` parameter to `6` and runs the function with that input. For the `output` variable, the Kotlin compiler infers the type `Long` based on the function's return type.

> ### Function Types

Functions have well-defined types in Kotlin. For instance, the `fibonacci` function has type `(Int) -> Long`, which means it accepts an `Int` as input and returns a `Long`. In other words, it's "a function from Int to Long". This concept becomes important once you get into *functional programming* in Kotlin.

Quiz

Working with Functions in Kotlin

1 Which of the following is a valid function signature in Kotlin?

COMPLETED 0%

1 of 3

Exercises

At this point, you've mastered enough concepts to solve basic algorithmic exercises. Such exercises are by far the most effective way to really grasp the language and use it confidently.

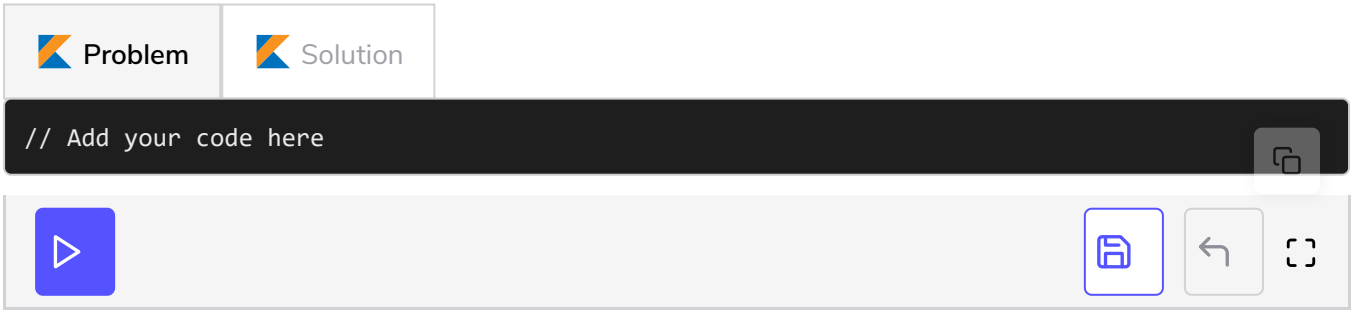
Note: This lesson contains several exercises, and these may take longer than the ones from previous lessons.

> This is because you are now at a point where you can solve actual computing problems. Please take your time to complete these exercises to build a strong foundation for the following lessons.

Multiplication Table

Implement a function that prints a multiplication table. As inputs, it accepts two ranges: one for the rows to use, and one for the columns.

Call your function to print a multiplication table with rows `1..5` and columns `1..8`.

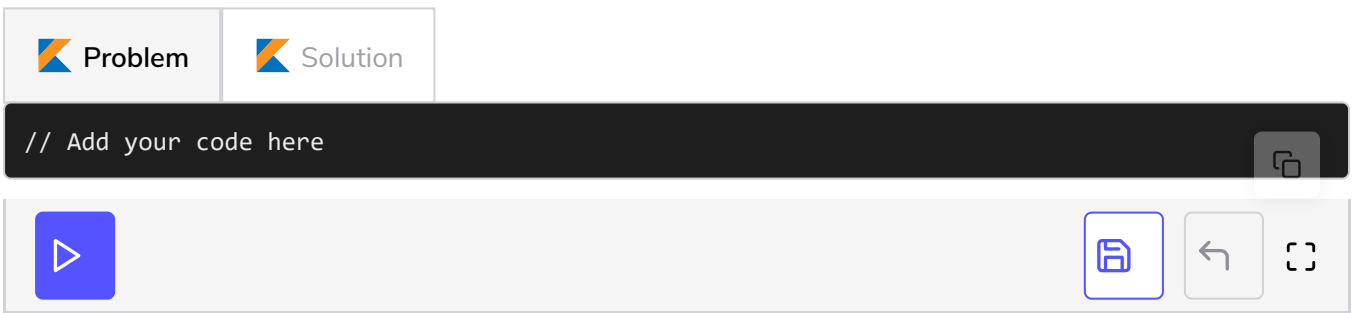


Hints:

- Use the type `IntRange` for the input parameters
- You can use the standard library function `System.out.format("%-8d", value)` to output a left-aligned value with a padding of 8 chars. This is necessary to print a proper table.

Greatest Common Divisor

Write a function that computes the Greatest Common Divisor (GCD) of two given integers. Call your function to find the GCD of (54, 24), (81, 153), and (137, 73).



Expected output:

- `gcd(54, 24) = 6`
- `gcd(81, 153) = 9`
- `gcd(137, 73) = 1`

Least Common Multiple

Write a function that computes the Least Common Multiple (LCM) of two given integers. Sanity-check your function by calling it with several example inputs. Try to vary the inputs and explore edge cases (e.g. cases where `lcm(a, b) == a`, `lcm(a, b) == b`, or `lcm(a, b) == a*b`).

Hint: There is a way to calculate the LCM based on the GCD so you can reuse your GCD function for this exercise.

Summary

Functions are possibly the most crucial concept to understand as they are one of the main building blocks in your code. Most day-to-day programming consists of adding, splitting up, removing, and otherwise working with functions.

- Kotlin uses the `fun` keyword to introduce functions.
- A function consists of function signature and function body.
  - The signature defines a function's name, parameters, and return type.
  - The function body defines what the function does.
- Functions in Kotlin have well-defined types that contain parameter types and return type.

In the next lesson, you will finally learn how to write your own `main()` function to implement an executable Kotlin file. So far, this part of the code has been hidden in the code listings to facilitate your learning.