# Getting Started with Kotlin

## Overview

In this lab you'll get a chance to get some experience using the Kotlin REPL interpreter shell. You'll also get an opportunity to work with Kotlin in IntelliJ.

## Code location

During the course you'll create various projects. We suggest you put the projects underneath the KotlinDev folder, e.g., C:\KotlinDev\student01-getting-started.

We provide full solutions for all the labs, in the appropriate project underneath KotlinDev, e.g., C:\KotlinDev\solution01-getting-started.

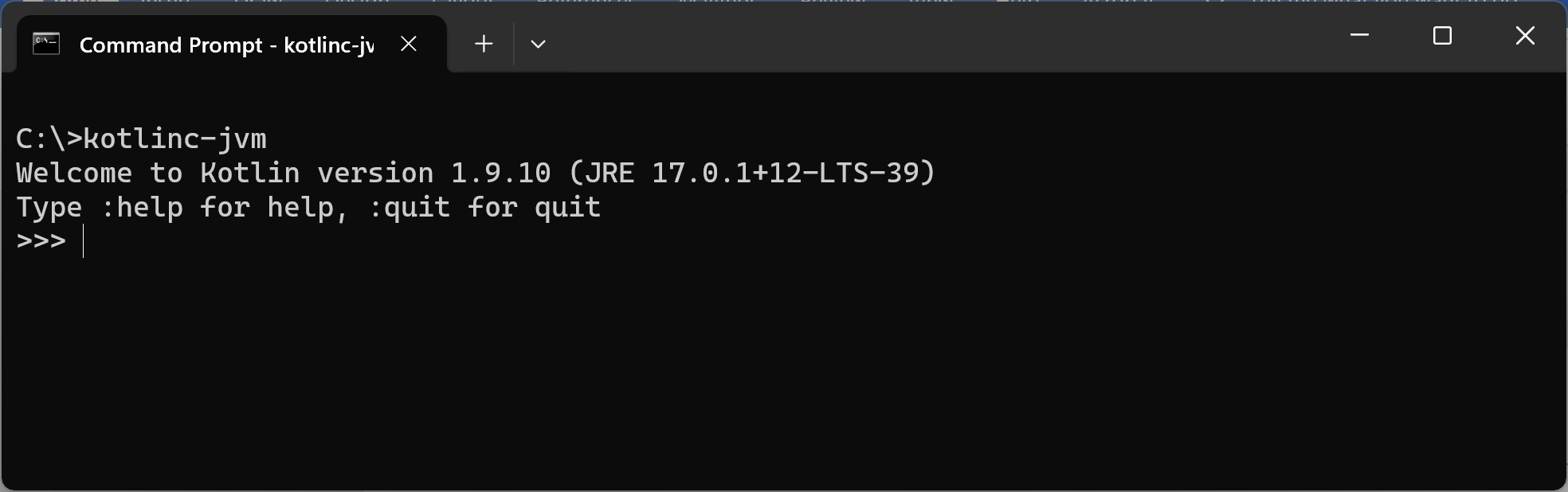
## Roadmap

There are 7 exercises in this lab, of which the last two exercises are "if time permits". Here's a brief summary of the tasks you'll perform in each exercise; more detailed instructions follow later in this lab doc:

1. Executing statements in the Kotlin command-line interpreter
2. A first look at declaring variables
3. A first look at strings
4. Loading scripts in the Kotlin interpreter
5. Creating a scratch file in IntelliJ
6. (If Time Permits) Creating a Kotlin application in IntelliJ
7. (If Time Permits) Additional suggestions

## Exercise 1: Executing statements in the Kotlin command-line interpreter

Kotlin has a command-line interpreter tool that enables you to run Kotlin expression on the fly. Open a command window and type in kotlinc-jvm as follows:



At the Kotlin prompt, enter a simple Kotlin expression. For example:

3 \* 4

The Kotlin interpreter interprets and runs the command. It displays the following result:

res0: kotlin.Int = 12

This indicates the Kotlin interpreter has evaluated the expression and the answer is 12. The interpreter stores this result in a variable named res0, which is of type kotlin.Int. In Kotlin, kotlin.Int is a standard class representing a 32-bit integer.

You can access the res0 variable in subsequent statements. For example, type the following:

res0 \* 10

The Kotlin interpreter interprets and runs the command. It displays the following result:

res1: kotlin.Int = 120

As you can see, now we have another variable named res1 that stores the result 120. In this way, you can build up statements based on previous results.

## Exercise 2: A first look at declaring variables

In the previous question, you evaluated expressions and the Kotlin interpreter automatically created variables named res0, res1 etc. to hold the results.

It's also possible for you to declare variables yourself. We'll discuss variables in more detail in the next chapter, but we'll take a sneak preview now. Type the following commands:

var a = 3

var b = 4

var p = a \* b

var q = a / b

println(p)

println(q)

When you run these statements, note that p is 12 and q is 0. The reason q is 0 is because of integer arithmetic; a and b are integers, and when you divide integers it rounds the answer downwards. Hence a/b is 0, not 0.75.

If you want to perform "floating-point" arithmetic, i.e., with fractions, then try the following:

var a = 3.0

var b = 4.0

var p = a \* b

var q = a / b

println(p)

println(q)

When you run these statements, note that the Kotlin REPL interpreter treats a and b as type kotlin.Double. In Kotlin, kotlin.Double is a class that represents a floating-point (fractional) value. When you perform arithmetic using kotlin.Double values, it retains fractions. Thus quotient is 0.75.

## Exercise 3: A first look at strings

Kotlin has a kotlin.String class, which represents textual content. You can get full documentation for the class at <https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-string/>.

Type the following into the REPL window:

var firstname = "Mickey"

var lastname = "Mouse"

var fullname = lastname.uppercase() + ", " + firstname

println(fullname)

Note the following points:

* The first two statements declare kotlin.String variables. Note that you use double-quotes to represent string literals in Kotlin.
* The last statement concatenates the strings together, using the + operator. The last name is converted to uppercase via the uppercase() method – the kotlin.String class has lots of methods like this, to manipulate textual content.

## Exercise 4: Loading scripts in the Kotlin interpreter

The Kotlin REPL interpreter lets you load and run prewritten Kotlin scripts via the :load command. This is easier than typing each command separately at the REPL prompt.

To try it out, follow these steps:

* Open a text editor, type in some Kotlin code, and save the file with a .kt file extension.
* In a Command Prompt window, run the Kotlin interpreter.
* Load your Kotlin file, via the :load command. The Kotlin REPL interpreter interprets and runs your code. If you run :load again, it will re-interpret and re-run your code again.

## Exercise 5: Creating a scratch file in IntelliJ

The Kotlin plugin for IntelliJ IDEA supports *scratches*. Scratches let you create code drafts in the same IDE window with your project and run them on the fly. Scratches are not tied to projects; you can access and run all your scratches from any IntelliJ IDEA window

To create a Kotlin scratch, click **File | New | Scratch File**, and select the **Kotlin** type. In your scratch, you can write any valid Kotlin code, including new functions and classes. Syntax highlighting, auto-completion, and other code editing features of IntelliJ IDEA are supported in scratches. Type some code into the scratch, and then click *Run*. The execution results will appear opposite the lines of your code.

## Exercise 6 (If time permits): Creating a Kotlin application in IntelliJ

In this exercise you'll see how to create a Kotlin application in IntelliJ. Follow these steps:

* Click **File | New | Project**
* In the New Project dialog:
  + In the left-hand-side panel, click **Kotlin Multiplatform**
  + Enter a suitable name and location
  + For the project template, select **Console Application**
  + For the build system, select **Gradle Kotlin**
  + For the JDK, select **17**
  + Click **Next**
* In the next screen:
  + For the target JVM version, select **17**
  + For the test framework, select **JUnit 5**
  + Click **Finish**
* Run the generated application as follows:
  + Expand the project, then expand **src**, **main**, **kotlin**
  + Right-click the .kt file and click the **Run** option
  + It should build ok and display *Hello, World* in the console.

## Exercise 7 (If time permits): Additional suggestions

* Experiment with some mathematical operations – adding, subtracting, multiplying values, etc. See what's available in the language.
* Explore the String class – see what methods are available and how they work.