# Defining and Calling Functions

## Overview

In this lab, you'll take an existing Kotlin application where all the code is written in one big chunk of code, and refactor it so that it makes proper use of functions.

## Roadmap

There are 4 exercises in this lab, of which the last two exercises are "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Refactoring the application to use functions
2. Getting user input
3. (If Time Permits) Validating user input
4. (If Time Permits) Using function overloading

## Familiarization

In the *student* module, take a look at the code in DateProcessing.kt. The code is quite lengthy, and performs the following tasks:

* Declares local variables to represent a day, month, and year, and initializes them with hard-coded values.
* Determines if the year is a leap year.
* Determines the number of days in the month.
* Determines the name of the month (e.g. month 1 is “January”)
* Displays all dates in a month in verbose format, such as 1st December 2023.

Make sure you’re happy with the code, then run it and verify it works as expected.

## Exercise 1: Refactoring the application to use functions

Refactor the code so that some of the functionality is placed into dedicated functions instead. We suggest the following functions:

* fun calcIsLeapYear(year: Int) : Boolean  
  Takes a year number (e.g. 2023) and returns true if a leap year or false otherwise.
* fun calcDaysInMonth(month: Int, isLeapYear: Boolean) : Int   
  Takes a month number (1…12) and a Boolean indicating whether it’s a leap year, and returns the number of days in that month. For example, if the month is 2 and it’s a leap year, then there are 29 days in that month.
* fun calcMonthName(month: Int) : String  
  Takes a month number (1…12) and returns the name of the month. For example, if the month is 2, the month name is “February”.
* fun calcSuffix(day: Int) : String  
  Takes a day number (1…31) and returns the suffix for that day number. For example, if the day number is 1, the suffix is “st”; if the day number is 2, the suffix is “nd”, etc.

Modify the main code so that it calls these functions. You should find the "client code" is a lot easier to read after you’ve made these changes. Don’t forget to run the application to make sure it still works!

## Exercise 2: Getting user input

Enhance the application so that it asks the user to enter values for the day, month, and year (rather than simply hard-coding these values in the client code).

## Exercise 3 (If time permits): Validating user input

Make the application more bullet-proof by validating the user’s input as follows:

* The year must be between 0 and 2099 inclusive.
* The month must be between 1 and 12 inclusive.
* The day must be between 1 and an appropriate upper limit inclusive. The upper limit depends on the month (i.e. 28, 29, 30, or 31).

If the user enters an invalid number, prompt the user to re-enter a value (use a do-while loop).

*Note:* There’s a certain amount of similarity in the code for validating the year, month, and day. In each case, you have to display a prompt message (e.g. “Please enter a year”), get an integer from the keyboard, and then wrap it all in a do-while loop to keep prompting the user if their input is invalid. To avoid code duplication, we suggest you write a single function as follows:

fun askUserForNumber(prompt: String, min: Int, max: Int) : Int

The function takes 3 parameters:

* prompt allows the client code to pass in an appropriate prompt message on each call (e.g. “Please enter a year”).
* min is the minimum value (inclusive) for validation purposes.
* max is the maximum value (inclusive) for validation purposes.

Refactor the client code to call this function as appropriate to get a valid day, month, and year. Then run the application and verify it traps all kinds of invalid user input.

## Exercise 4 (If time permits): Using function overloading

Implement an overloaded version of the askUserForNumber() function. This is similar to function you wrote in the previous exercise, except that it doesn’t do validation:

* fun askUserForNumber(prompt: String) : Int

Consider how you can invoke these functions from the client code, to test your new overloaded versions of the askUserForNumber() function.