# Functional Programming

## Overview

In this lab you'll explore the wonderful world of functional programming. You'll define various lambdas, pass them in and out of functions, and explore various techniques.

## Roadmap

There are 4 exercises in this lab, the last of which is "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. Define and call a lambda
2. Define and use a type alias to specify a lambda signature
3. Define and use a generic lambda
4. (If Time Permits) Define and call a higher-order function

## Exercise 1: Define and call a lambda

Define a lambda expression that tests if an integer parameter is even. The lambda should take an Int parameter and returns true (if it's even) or false (if it's odd). Assign the lambda expression to a variable named isEven.

Call isEven from client code and pass an even number. Display the return value on the console, it should be true.

Call isEven from client code again, and this time pass an odd number. Display the return value on the console, it should be false.

## Exercise 2: Define and use a type alias to specify a lambda signature

Define a type alias named IntPredicate to represent the following function signature:

* Int parameter
* Boolean return type

Using this type alias, define a variable named is isNonNegative and assign a lambda expression that tests if a number is non-negative. Use the bare minimum syntax you can possibly get away with for the lambda…

Call isNonNegative from client code several times, passing a variety of parameters. Verify it works correctly in each case.

## Exercise 3: Define and use a generic lambda

Define a type alias named Predicate to represent the following generic function signature:

* A parameter of some generic type T
* Boolean return type

In your client code, declare a variable named anIntPredicate of type Predicate<Int>. This variable can point to any function that takes an Int, performs some test on that Int, and returns a Boolean result.

Now write some code to assign various different lambda expressions to the variable:

1. A lambda that tests if a number is non-negative
2. A lambda that tests if a number is a valid exam mark (i.e. 0 to 100)
3. A lambda that tests if a number is a multiple of 10

For each lambda, call it several times with different parameters. For example, call lambda a) 3 separate times with numbers such as 15, 0, 10. Verify the lambda returns the correct result in each case.

Now declare a variable named aStringPredicate of type Predicate<String>. This variable can point to any function that takes a String, performs some test on that String, and returns a Boolean result.

Now write some code to assign various different lambda expressions to the variable:

1. A lambda that tests if a string is all-lowercase
2. A lambda that tests if a string has no leading or trailing whitespace

For each lambda, call it several times with different parameters. For example, call lambda a) 3 separate times with strings such as "Matthew", "mark", "LUKE", "john".

## Exercise 4 (If Time Permits): Define and call a higher-order function

A higher-order function is a function that takes another function as a parameter, and/or returns a function as the result…

Define a higher-order function named not as follows:

* The not function takes a Predicate<Int> as a parameter. Let's call this parameter origTest.
* Implement the not function so that it returns another Predicate<Int> lambda, the logical negation of origTest. In other words, return a lambda that takes an Int parameter, passes it into origTest, and then negates the result.

In the client code:

* Declare a variable named isShortMonth, of type Predicate<Int>. Assign it a lambda that tests if the Int parameter represents a short month (i.e. 2, 4, 6, 9, or 11 representing February, April, June, September, or November).
* Declare a variable named isLongMonth, also of type Predicate<Int>. Assign it the result of calling not(isShortMonth), i.e. a lambda that negates the result of the isShortMonth lambda.
* Invoke the isLongMonth lambda several times with different parameters. It should return true for all months except 2, 4, 6, 9, and 11.