Exercise 6 - Functional Programming

Objective

The objective behind this exercise is to look at functional programming. While some of these exercises can be done without functions (by using loops or mutable state) try and finish the exercises using only the functional paradigm. We will look at functional constructs such as lambda and higher order functions.

References:

Chapter 6 of the slides: 117 - 141

Overview

The term predicate is used a lot in functional programming. This refers to a pure function, one without side effects. Anonymous or lambda functions are those which don’t have the full overhead of writing the full function declaration. They can be passed in scala as objects into other functions. We can give a lambda function a reference which can be used to identify it, but a traditional anonymous function would not have this.

An anonymous function takes the form

def addOne = (x: Int) => x + 1

1. Write an anonymous function which takes in an Int (x) and returns (x\*2)
2. Call your function with a few values to see if it is working correctly. By giving the anonymous function an identifier (the def value) we can now pass it into other functions and methods as if it was an object.
3. Write a function that takes in two Ints and returns their sum
4. Write a function which takes in an Int and a String and returns a Boolean based on whether the length of the String matches the Int value
5. Write an ‘isEven’ function which takes in an Int and returns a Boolean based on whether the int was even or not.
6. Define a negate function which will return the opposite Boolean value of whatever function it was given.
   1. Negate takes in a function of the type (Int => Boolean)
   2. If the function returns true, negate will return false. If the function returns false, negate will return true.
7. Use the isEven and negate function, create an isOdd function.
8. Write a function which takes in a minimum numbers, a maximum number and a function which applies the function to every number in the range min <= x <= max and outputs it as a collection. (Hint: look at the for/yield construct)

For example the function call would be: func(2,4, (x: Int) => x + 2)

The output would be [4,5,6]

1. Write a function which takes in three Int values and adds them all together
   1. Now curry that function (see slide 130) to turn it from a single three int function to three individual functions all wrapped inside one another
   2. The curried version of the function returns a function in the first two steps. Call your function and explore how it works. You can set the value for a specific step, give it an identifier and then reuse this. (see slide 131)
2. Compose your timesTwo and the addOne function (defined as the example at the start of this exercise). Try it with both the compose and the andThen methods
   1. Which version do you prefer?