# Functional Programming

## Overview

This lab gives you an opportunity to write lambda expressions, define higher-order functions, and implement recursive operations.

## Source folders

Student folder : C:\PythonDev\Student\08-FunctionalProgramming

Solution folder: C:\PythonDev\Solutions\08-FunctionalProgramming

## Roadmap

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

1. Writing a lambda expression
2. Defining a higher-order function
3. Implementing recursive operations
4. Additional suggestions (if time permits)

## Exercise 1: Writing a lambda expression

Write a lambda expression that acts like an "is even" predicate, i.e. it takes a numeric parameter and returns a boolean indicating if the number is even.

Call the lambda expression with an even number, and verify the lambda returns true. Then call the lambda expression with an odd number, and verify the lambda returns false.

## Exercise 2: Defining a higher-order function

Define a higher-order function named negate that reverses the result returned by a predicate. Here are some hints:

* The negate function should take a single parameter, e.g. named f, representing a predicate function.
* The negate function should return a function that returns the inverse of f. To achieve this effect, call f, then use the not operator to invert it.

Define a variable named isOdd in terms of isEven and negate. Verify that isOdd returns true for odd numbers and false for even numbers.

## Exercise 3: Implementing recursive operations

Write a function named nth, which returns the nth item in a list. For example, imagine you define a list named mylist and set it to [100,101,102,103,104,105,106], then calling nth(4, mylist) will return 104.

Don't use a loop, or any mutable state – use recursion instead. Also, make use of the subscript features of the list class:

* someList[0] - Returns the first item in the list
* someList[1:] - Returns the rest of the list

## Exercise 4 (If time permits): Additional suggestions

Implement a binary tree. A binary tree is a recursive data structure:

* A binary tree is a node, with a binary tree on the left and a binary tree on the right.

Hints:

* Define a class called Node, which represents one node in a tree. A node has a piece of data, and a pointer to a left node, and a pointer to a right node.
* Define a class called BinaryTree, which represents the tree itself. The BinaryTree class only needs one property – a pointer to the head node. The BinaryTree class needs methods such as add() and display(). If you are feeling brave you can also implement remove(), good luck with that.