CSC/CPE 101: Fundamental of Computer Science I

Spring 2018 (LAB-5)

Due: 5/11/18

This lab provides exercises on lists.

Download lab5.zip, from PolyLearn (place it in your CPE101 directory, and unzip the file)

unzip lab5.zip

List Indexing

This part of the lab introduces indexing lists. You should **not use any loops** in your functions. Though they would be useful to generalize your code (which we will do in a later lab), the goal of this lab exercise is to understand the mechanics of using lists so you should focus only on that.

In the `poly` directory create a file named `poly.py`. Place your test cases in `poly_tests.py` (this file is not provided; you should be comfortable writing this by now).

You must provide at least two test cases for each of these functions.

This part will be executed with: **python poly_tests.py**

Polynomial Arithmetic

For this part of the lab you will develop two functions that perform basic arithmetic on polynomials. A polynomial will be represented as a list. The values in the list will represent the coefficients of the terms whereas the indices will represent the exponents for the terms.

This means that the polynomial **$2.7x^2 + 3.1x + 2$** will be represented by the following list. Notice that the term with exponent 0 is first in the list while the term with exponent 2 is last (i.e., the terms in the list are in reverse order of how they are typically written in mathematics; this is done so that an element's index represents that term's exponent).

```
poly = []
poly.append(2)
poly.append(3.1)
poly.append(2.7)
```

This list can also be created directly as follows. This is convenient when, for instance, writing test cases.

```
poly = [2, 3.1, 2.7]
```

You may think this mapping of a polynomial to a list is a bit odd. In fact, attributing meaning to indices of a list (and not just the values within the list) is a pretty important skill that allows a list to be used as more than just a substitution for a bunch of variables.

poly_add2

In `poly.py`, develop the `poly_add2` function. This function takes two polynomials of degree two (lists of length three) as arguments. This function must return a new list (i.e., do not modify the contents of the input lists) representing the sum of the input polynomials.

Though the testing framework does work with lists, it does not support an "almost" equal check on the contents of a list. In the provided testing file you will find `assertListAlmostEqual`. It can be used, in a testing function, as follows.

```
def test_poly(self):
    poly1 = [2.3, 4.7, 1.0]
    poly2 = [1.2, 2.1, -3.2]

    poly3 = poly.poly_add2(poly1, poly2)
    self.assertListAlmostEqual(poly3, [3.5, 6.8, -2.2])
```

poly_mult2

Develop the function `poly_mult2`. This function will take two polynomials of degree two and compute the product of the two polynomials. Polynomial multiplication is not a simple multiplication of values at the same index; instead, think of the distributive law (of which the FOIL method is a special, simple case).

Note carefully: The polynomial resulting from a multiplication will, in general, be of degree greater than the argument polynomials. In this case, the result can be of at most degree four, so your result list (checked against in your test cases) may be larger than initially expected.

Again, though the use of loops would allow one to generalize this function, for this lab you cannot use any loops. Think carefully about how to compute the product of polynomials and how that relates to the representation of polynomials in this lab.

Iteration Patterns

This part of the lab requires solving a number of relatively simple problems using specific iteration patterns. You will likely notice that the code in the functions for each pattern is very, very (extremely!) similar. That's because you will be using common patterns. Patterns are good; they allow programmers to think "I need to do something like that, but with different values or with different operations" and then do it by changing the parts that are actually different.

You must provide at least two test cases for each of the following functions.

Map Pattern

Develop these functions in the `map` directory in files `map.py` and `map_tests.py`.

Each of the functions for this part of the lab is to be implemented using the map pattern. In this pattern, each value in the result list is determined by a computation on the value in the corresponding position (i.e., at the same index) in the input list.

Note: You must implement one of these functions using a **list comprehension**, one using a **for** loop, and one using a **while** loop.

square_all

The `square_all` function computes and returns a list of the square of each value in the input list.

add_n_all

The `add_n_all` function (taking two parameters) computes and returns a list with each element set to the sum of the parameter `n` and the corresponding value in the input list.

even_or_odd_all

The `even_or_odd_all` function takes as input a list of integers and computes and returns a list containing True/False representing whether each corresponding number in the input list is even.

Filter Pattern

Develop these functions in the `filter` directory in files `filter.py` and `filter_tests.py`.

Each of the functions for this part of the lab is to be implemented using the filter pattern. In this pattern, the values in the result list are determined by a conditional test on each value in the input list. Only those values that satisfy the condition will be copied to the result list. It is

conventional for the values in the result list to be in the same relative order as in the input list.

Note: You must implement one of these functions using a **list comprehension**, one using a **for** loop, and one using a **while** loop.

are_positive

The `are_positive` function returns a list of all positive values in the input list.

are_greater_than_n

The `are_greater_than_n` function (taking two parameters) returns a list of all values in the input list that are greater than the `n` parameter.

are_divisible_by_n

The `are_dividable_by_n` function (taking two parameters) returns all integers in the input list that are divisible by the `n` parameter.

Demonstration

Demonstrate the test cases from each part of the lab to your instructor to have this lab recorded as completed. In addition, be prepared to show the source code to your instructor.

Submission

Demo the Lab5 and submit all .py files in the PolyLearn.