The Third Midterm Will Cover Content From:

Material covered in Lectures #1-16 (up to and including ‘Associative Collections')
Chapters 1-9 of Python for Everybody
Practice Problems PP1, PP2, and PP3, PP4, PP5, PP6, and PP7

Important topics to ensure you understand include: dictionaries, lists, string operations, functions, loops, complex branching and logical operations, as well as applying problem solving and computational thinking to these sorts of problems.

Any questions regarding the midterm content, structure, practice questions, etc., should be posted in the Midterm and Exam Discussion board on cuLearn or on the course Discord server.

For problems 3-5 of this practice midterm, you will also be given tester files that will evaluate your solution. If you run either of these files, they will load your solution and run several test cases by calling your function. They will then output the results of the tests. You should be comfortable using the files in this way as several test files may be provided in both the third midterm and the final exam. One tip: do not get caught up in only trying to increase the number of correct results from the test file and forget to work towards solving the problem in general.

Additionally, you will be provided reference PDFs for Python lists, strings, dictionaries, and the random module. These can be found in the resources directory within the practice midterm directory.

Problem 1

Write a function called **search_employees** that accepts two string arguments. The first string given to this function will be the name of a file that contains lines with employee information. The general format of each line in the file will be:

employee_number, employee_name, employee_pay

The only things you can assume about the structure of this file are that each employee's information is stored on a single line and each piece of information on that line is separated by a comma. The second string given to the function will be a search keyword. The function must return a list of employee names that match the search keyword. An employee name should match the search keyword if that keyword can be found anywhere in the employee name. The search should also be done in a case-insensitive manner.

Write a second function called analyze_pay that accepts a single string argument that will represent the same type of file as above. The function should return a list containing the average pay, largest pay, and smallest pay, in that order. You are given 5 employee_infoX.txt files. The correct analyze_pay outputs for each file are:

employee_info0.txt: [84361.66666666667, 135941.0, 41039.0]
employee_info1.txt: [84169.36363636363, 157986.0, 30757.0]
employee_info2.txt: [92658.71428571429, 146165.0, 31054.0]
employee_info3.txt: [92356.5, 151740.0, 31173.0]
employee_info4.txt: [91087.1, 158467.0, 31613.0]

Problem 2

Write a function called **removehtml** that takes a single string argument and returns that string with the HTML removed. For the purposes of this question, you can assume that anything found between < and > is HTML and must be removed (including the < and > symbols). Note: do not worry about any lack of spacing between words after the removal is completed.

Examples:
removehtml("<html><head><title>Page Title<title><head><html>") → Page Title
removehtml('<a href="p1">Link 1</a><a href="p2">Link 2</a>') → Link 1Link2
removehtml("<p>Click <a href="here.html">here</a></p>") → Click here

Problem 3

The Jaccard index is a measure of similarity between two sets, computed as the size of the intersection of the sets divided by the size of the union of the sets. In other words, the Jaccard index of two sets A and B can be calculated as:

$$\frac{\#\text{ unique elements that are present in both A and B}}{\#\text{ unique elements present in A or B}}$$

Write a function called **jaccard** that computes the Jaccard index for two argument lists. You can assume that the lists will contain only integer values. Note: don't forget that only unique elements should be counted in the above equation, any duplicate elements should be ignored. **You may not use list comprehensions, additional modules, or the built-in union/intersection functions**.

Examples:
A = [1, 2, 3, 4, 5]   B = [2, 4, 6, 8]
intersection(A,B) → [2, 4]
union(A,B) → [1, 2, 3, 4, 5, 6, 8]
jaccard(A,B) → 2 / 7 → 0.2857

A = [0, 6, 1, 7]   B = [1, 7, 6, 3]
intersection(A,B) → [6, 1, 7]
union(A,B) → [0, 6, 1, 7, 3]
jaccard(A,B) → 3 / 5 → 0.6

Problem 4

Write a function called **xmostfrequent** that takes one string argument (S) and one integer argument (X). The function must return a list of the X most frequent words in the string S. Additionally, the returned list must have the X most frequent words sorted from highest to lowest frequency. You may assume all words are separated by spaces and that there is no additional punctuation. You may also assume that X is less that the number of unique words in the string. **You may not use additional modules or use any of the built-in min/max/sort functions.**

Examples:
xmostfrequent("peach apple peach orange apple peach", 2) → ["peach", "apple"]
xmostfrequent("pea apple pea orange pea orange", 2) → ["pea", "orange"]
xmostfrequent("b a b a c c a c a d b c c", 3) → ["c", "a", "b"]

Problem 5

Write a function called **isvalidseries** that accepts a list of positive integers (L), an integer (X), and an integer (SUM) as arguments. A series of integers will be considered valid if there is no sequence of X consecutive numbers in the list that sum to larger than SUM. The isvalidseries function must return True if the given list is considered valid for the given X/SUM values, and False otherwise.

Examples (bold numbers show invalid consecutive numbers that exceed SUM):

isvalidseries([**8, 4, 8**, 3, 1, 2, 7, 9], 3, 19) → False
isvalidseries([2, 4, 8, 3, 1, 2, 7, 9], 3, 19) → True
isvalidseries([2, 4, 8, 3, 1, **2, 7, 9**], 3, 16) → False
isvalidseries([5,5,5,5,5,5,5,5], 3, 19) → True
isvalidseries([2,2,**5,5,5,5**,5,5], 4, 19) → False
isvalidseries([5,5,5,5,5,5,5,5], 4, 20) → True