CS2020
Introduction to Programming
Practice Exercise Set 5

List Processing—Part 2

This exercise set is a continuation of Exercise Set 3, basic list processing. In Set 3, you worked with one-level lists that have no nested structure. In this set, you will deal with nested lists of varying complexities.

Do not use map/filter operations or list comprehensions in this set. There is another exercise set for you to practice these techniques.

You may use list methods such as find, index, count, and others, unless indicated otherwise. You may also use any built-in functions such as max, min, sum, and others, unless indicated otherwise.

Practice Topic

- Processing nested lists without using list methods
- Processing nested lists using list methods
- Processing nested lists with functions

Prerequisite

You need to possess knowledge on

- Basic operations such as assignments and expressions
- Input and output (print)
- Data types int, float, and str.
- Simple selection control using if-else
- Basic looping (Practice Exercise Set 1)
- Defining and using functions (Practice Exercise Set 2)
- Basic list processing (Practice Exercise Set 3)

Example 1

M is a n-by-m matrix, a list of of lists of numbers. Here's an example

```
[[10, 4, 20, 40],
 [ 4, 3, 23, 13],
 [30, 9, 10, 33]]
```

Find the sum of the numbers in M without using any list methods or the built-in function sum.

a. Direct Access Loop

```
total = 0
for row in M:
   for num in row:
      total += num
```

b. Indexing Access with for

```
total = 0
for i in range(len(M)):
   for j in range(len(M[i]):
         total += M[i][j]
```

c. Indexing Access with while

```
total, i = 0, 0
while i < len(M):
   j = 0
   while j < len(M[i]):
      total += M[i][j]
      j += 1
   i += 1
```

Example 2

Write a function get_total that accepts one parameter M—an n-by-m matrix. Return the sum of the elements in M without using any list methods or the built-in function sum.

a. Direct Access Loop

```
def get_total(M):
   total = 0
   for row in M:
      for num in row:
         total += num
   return total
```

b. Indexing Access with for

```
def get_total(M):
   total = 0
   for i in range(len(M)):
      for j in range(len(M[i]):
            total += M[i][j]
        return total
```

c. Indexing Access with while

```
    def get_total(M):
        total, i = 0, 0
        while i < len(M):
            j = 0
            while j < len(M[i]):
                total += M[i][j]
                j += 1
            i += 1
        return total
```

For each question, provide a solution using direct access, indexing with the for loop, and indexing with the while loop. Unless explicitly requested, you are not required to define a function.

1. Given an n-by-m matrix M, find the maximum number. Do not use the built-in function max.
2. Given an n-by-m matrix M, find the average. Do not use the built-int function sum.
3. Write a function extract_negatives that accepts an n-by-m matrix M and returns a list of negative numbers found in M.
4. Write a function flatten that accepts an n-by-m matrix M and returns a one-dimensional list. In other words, you convert a list of lists of numbers into a single-level list of numbers.
5. Given LM, a list of n-by-m matrices, find the maximum. Do not use the built-in function max. Note here that LM is a list in which each member is an n-by-m matrix.
6. Write a function find_max_matrix that accepts LM, a list of n-by-m matrices, and return the matrix that includes the maximum among values in all matrices.
7. Write a function find_averages that accepts LM, a list of n-by-m matrices, and returns the average of each matrix as a single list.
8. Given a roster list, find the names of the youngest students. There could be more than one student. Store their names in a list. The roster is organized as a list of student records. A student record is a list of his/her id (string), name (string), class (int), age (int) and a list of test scores (float). The class is a single digit number to designate the student's class level: 0 - Freshman, 1 - Sophomore, 2 - Junior, and 3 - Senior. The id value for a student is unique; there will be no two students with the same id value.
9. Write a function find_average_scores that accepts a roster list (Exercise 8) and returns the average test scores of each student. Return the result as a list of pairs (tuples) containing the student name and his/her average.
10. Write a function find_student that accepts a roster list (Exercise 8) and a student id and returns the student record if found and None otherwise.
11. Write a function group that accepts a roster list (Exercise 8) and the class id and returns a list of students belonging to the class. You return a complete record for the students.