National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2024/2025

Tutorial 09
Advanced List Processing & Classes

Release date: 28th October 2024
Due: 3rd November 2024, 23:59

General Restrictions

- No importing packages unless explicitly allowed to do so.
- You are only allowed to use data structures taught so far.

Questions

1. A matrix can be represented in Python by a list of lists (nested lists). For example, `m = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]` represents the following 3×4 matrix:

$$\begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{vmatrix}$$

   (a) Implement a function `transpose` which takes in a matrix and transposes it. Basically, this converts an $m \times n$ matrix into an $n \times m$ matrix. The function should return a **new** matrix.

   (b) Now re-implement `transpose` such that it modifies and returns the **original** matrix instead.

   (c) Implement a function `row_sum` which takes in a matrix and returns a list, where the i-th element is the sum of the elements in the i-th row of the matrix. You can assume that the matrix will not be empty, and has exactly $m \times n$ elements, where m and n are positive integers.

```
>>> row_sum(m)
[10, 26, 42]
```

   (d) Implement a function `col_sum` which takes in a matrix and returns a list, where the i-th element is the sum of the elements in the i-th column of the matrix.

```
>>> col_sum(m)
[15, 18, 21, 24]
```

2. You are given a list of students in the following form (name, letter grades, score). For example,

```
students = [
    ('tiffany', 'A', 15),
    ('jane', 'B', 10),
    ('ben', 'C', 8),
    ('simon', 'A', 21),
    ('eugene', 'A', 21),
    ('john', 'A', 15),
    ('jimmy', 'F', 1),
    ('charles', 'C', 9),
    ('freddy', 'D', 4),
    ('dave', 'B', 12)]
```

The functions that you implement for this question should work with any arbitrary list of students and not just for this sample list.

(a) Implement a function `mode_score` which takes in a list of students and returns a list of the mode scores (scores that appear the most number of times). If there is only one score with the highest frequency, then this list would only have one element.

For example:

```
>>> mode_score(students)
[15, 21]
```

(b) Implement a function `top_k` which takes in a list of students and an integer k and returns a **new** list of k students with the highest scores in alphabetical order. If there are students in the range $(k+1, \cdots k+i)$ who have the same score as the kth student, include them in the list as well. **Do not modify the original list.**

For example:

```
>>> top_k(students, 5)
[('eugene', 'A', 21), ('simon', 'A', 21), ('john', 'A', 15),
('tiffany', 'A', 15), ('dave', 'B', 12)]

>>> top_k(students, 3)
[('eugene', 'A', 21), ('simon', 'A', 21), ('john', 'A', 15),
('tiffany', 'A', 15)]
```

3. Much like a real-life queue, the `Queue` data structure is one which stores a sequence and deals with the objects in the sequence by the principle of first-in first-out (FIFO). As object-oriented programming is the concept where you can define data types outside of the currently available ones (`int`, `str`, `tuple` etc.), now it's your turn to implement a `Queue` data structure!

   Implement the following methods:

   (a) `__init__`, which instantiates the object (in other words, create the thing).

   (b) `enqueue`, which takes in **one** object as input and adds it to the end of the queue.

   (c) `dequeue`, which removes the object at the front of the queue and returns said object. If the queue is empty, return `None`.

   (d) `peek`, which returns the object at the front of the queue without removing it. If the queue is empty, return `None`.