Lab Exercise 2

Learning Outcomes Covered

- Review: user input, format output, conditional statements, looping statements
- Functions
- Manipulations on Python lists

Instructions

- Unless specified otherwise,
  - you are allowed to use anything in the Python Standard Library;
  - you are NOT allowed to use any third-party library in your code;
  - you can assume all user inputs are always valid and your program for each question need not include code to validate them.
- The underlined blue texts in the Sample Runs section of some questions refer to the user inputs. It does NOT mean that your program needs to underline them.
- **Please name each of your Python script files exactly as the name in parenthesis as shown in each question heading. Marks will be deducted if violating this instruction.**

Part I: In-Class Exercise (Deadline: Jan 27, 2:15pm)

Question 1 (q1.py) (20%)

Taxes are often categorized as *flat* and *progressive*. A *flat tax* is a tax with a single constant rate on the taxable amount. A progressive tax is one in which the rate increases as the taxable amount increases. In Hong Kong, individuals are taxed based on their net chargeable income at progressive rates as follows:

- 2% on the first $40,000
- 7% on the following $ 40,000
- 12% on the succeeding $ 40,000
- 17% on the amounts thereafter

Write a Python program, which takes several peoples' net chargeable incomes one by one and output their taxes payable. The number of people is entered by the user via console input. The program must define a function `get_tax_payable()` that takes an argument representing a person's net chargeable income and returns the tax to be paid.

Sample Output:

```
Enter the number of people: 4
#1:
Enter the net chargeable income: 30000↵
The tax payable is: 600.0
#2:
Enter the net chargeable income: 60000↵
The tax payable is: 2200.0
#3:
Enter the net chargeable income: 100000↵
The tax payable is: 6000.0
```

```
#4:
Enter the net chargeable income: 150000↵
The tax payable is: 13500.0
```

Part II : Take-home Exercise (Deadline: 11:59pm 31 Jan)

Question 2 (q2.py) (20%)

Define a function called `print_second_largest()` that takes a list of numbers as its only argument and prints a sentence telling the second largest number and its index in the list. If the number of elements in the list is smaller than 2, it should print an error message. You may assume no duplicate numbers in the list. Note that no sorting is allowed in this question.

For sake of separation of concerns, the function defines an inner function `get_second_largest()` to carry out the searching process. It takes the list of numbers and returns the second largest number and its index in the list.

Your Python script should be modified from the following one by inserting appropriate code for the function bodies. You may also better remove the words "TODO: " from the given comments.

```
def print_second_largest(nums):

    def get_second_largest(nums):
        #TODO: Return the second largest element and its index

    #TODO: Call the inner function and handle the output message

print_second_largest([4, 8, 3])
print_second_largest([1])
print_second_largest([3, 4, 5, 6, 7])
```

Sample Output:

```
The second largest number is 4, at index 0.
Oops, too few numbers in the list!
The second largest number is 6, at index 3.
```

Question 3 (q3.py) (20%)

Write a function which takes two matrices (implemented using two nested lists) and an arithmetic operator (denoted by a string) as its arguments, and returns the computation result. The function supports only two operations, which are implemented in two separate inner functions:

- **Matrix addition**: this operation is done by adding the corresponding entries in the two matrices together. You may assume that the two matrices are always of the same dimensions.

- **Matrix multiplication**: recall the definition of this operation as follows:

If $\mathbf{A}$ is an $m \times n$ matrix and $\mathbf{B}$ is an $n \times p$ matrix,

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

the *matrix product* $\mathbf{C} = \mathbf{AB}$ (denoted without multiplication signs or dots) is defined to be the $m \times p$ matrix

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

such that

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj},$$

for $i = 1, ..., m$ and $j = 1, ..., p$.

You may assume that the two matrices are always of the right dimensions so that they can multiply.

Your Python script should be modified from the following one by inserting appropriate code for the function bodies. You may also better remove the words "TODO: " from the given comments.

```
def do_matrix_operation(A, opt, B):
    def add(A, B):
        #TODO: return a matrix resulted from the addition of A and B

    def multiply(A, B):
        #TODO: return a matrix resulted from the multiplication of A and B

    #TODO: check opt and call the corresponding operator function

# 3x3 matrix
X = [[12, 7, 3],
     [ 4, 5, 6],
     [ 7, 8, 9]]
# 3x3 matrix
Y = [[5, 8, 1],
     [6, 7, 3],
     [4, 5, 9]]

print(do_matrix_opeartion(X, '+', Y)) # addition
print(do_matrix_opeartion(X, '.', Y)) # multiplication
```

Note: You are required to use some looping statements to attempt this question instead of hard coding the output. You are NOT allowed to use any existing function imported from any modules or 3rd party packages like NumPy to carry out the matrix addition and multiplication.

Sample Output:

```
[[17, 15, 4], [10, 12, 9], [11, 13, 18]]
[[114, 160, 60], [74, 97, 73], [119, 157, 112]]
```

Question 4 (q4.py) (20%)

You are studying how to make the most profit in stock market. Suppose that we are talking about only one lot of shares of a particular stock, the profit is defined as the price you sell minus the price you bought it. Given a list of prices of the stock in time order, write a function called `max_profit()` to find the maximum profit you can make. Suppose that you do not owe any shares of this stock beforehand, and there is only one buy action and one sell action possible. Remember that you cannot sell a stock before buying it!

Your Python script should be modified from the following one by inserting appropriate code for the function body. You may also better remove the word "TODO: " from the given comment.

```
def max_profit(prices):
    #TODO: return the maximum profit

AAPL = [12.5, 20.3, 23.2, 1.5, 2.6, 5.3, 7.5, 10.9]
GOOG = [1.8, 3.2, 8.0, 7.5, 1.0, 5.3, 12.8, 0.9]

print(max_profit(AAPL))
print(max_profit(GOOG))
```

Sample Output:

```
10.7
11.8
```

The output was so because for AAPL, the maximum profit is made by buying at $12.5 and selling at 23.2. Note that you cannot buy at $1.5 and sell at $23.2 which was a previous price already. Similarly, for GOOG, the best buy price is $1.0 and the best sell price is $12.8, giving the biggest profit $11.8. You can't buy at $0.9 but sell at $12.8 for the same reason.

Question 5 (q5.py) (20%)

A prime number (or a prime) is a whole number greater than 1 that is divisible by 1 or itself only. Write a function that returns a list of all prime numbers smaller than the input number *n*.

For an integer $n > 1$, the most straightforward way to test primality is to check whether *n* is divisible by any integer between 2 and $n - 1$. However, this is inefficient for checking a big *n*. To improve the efficiency, your function should only test whether *n* is divisible by any prime number smaller than it.

Your Python script should be modified from the following one by inserting appropriate code. You may also better remove the words "TODO: " from the given comments.

```
def get_prime_numbers(n):
    primes = []
    for num in range(2, n):
        #TODO: Check if num is prime by looping over the primes list

    return primes

#TODO: Handle console input and output
```

Sample Runs:

```
Enter a number: 10↵
[2, 3, 5, 7] are prime numbers smaller than 10.
```

```
Enter a number: 31↵
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29] are prime numbers smaller than 31.
```