CSCI 1133
Exercise Set 12

You should complete as many of these problems as you can. Practice is *essential* to learning and reinforcing computational problem solving skills. We encourage you to do these without any outside assistance.

Create a directory named `exercise12` and save each of your problem solutions in this directory. You will need to save your source files in this directory and push them to your repository in order to get help/feedback from the TAs. This requires that you follow a rigid set of naming requirements. Name your individual Python source files using "`ex12`" followed by the letter of the exercise, e.g., "`ex12a.py`", "`ex12b.py`", etc.

Automatic testing of your solutions is performed using a GitHub agent, so it is necessary to name your source files precisely as shown and push them to your repository as you work.

These problems are provided for practice writing simple Java programs.

A. **Calculus**

In this exercise you need to write a program in Java that does the following.

- Prompt the user to enter three numbers, a, b, and c, representing the quadratic equation:
  $ax^2 + bx + c = 0$
- Display the formula based on the entered values.
- Display the derivative of the equation $ax^2 + bx + c$ as,
  $2ax + b$
- Display the anti-derivative of the equation $ax^2 + bx + c$ as,
  $a/3 * x^3 + b/2 * x^2 + cx$

Example:

```
Enter three numbers: 3 5 8
The formula you entered is 3x^2 + 5x + 8
The derivative of the equation is 6x + 5
The anti-derivative of the equation is x^3 + 2.5x^2 + 8x
```

B. **Gravity**

Construct a Java program that computes the position of a falling object in meters, given the time, initial position, initial velocity and acceleration. The formula to compute this position is,

$$x(t) = 0.5\ a * t^2 + v_{initial} * t + x_{initial}$$

where `t` is the time, `a` is the acceleration, $v_{initial}$ is the initial velocity, and $x_{initial}$ is the initial position. Assume the gravitational acceleration, `a`, is -9.81 m/s^2.

Example:

```
Enter the initial position (m): 1000
Enter the initial velocity (m/s): 0
Enter the time elapsed (s): 10
The final position of the object is 509.5 m
```

C. **Sieve of Eratosthenes**

Although exhaustive enumeration is a simple way to identify all the prime numbers in some range, it is very inefficient. A much more efficient method is ascribed to the ancient Greek scholar Eratosthenes of Cyrene. The so-called *Sieve of Eratosthenes* is a prime filter that works like this: Starting with an array of all the integers from 2 through *n*, you begin by "crossing off" every value that is a multiple of the first prime value (i.e., *p=2*) They're crossed off because all *multiples* of a prime number are not prime by definition.

$$2p=4, 3p=6, 4p=8, 5p=10, \ldots, x : x <= n$$

Repeat this process by setting the value of *p* to the *next* prime in the array, which happens to be the next element that hasn't already been "crossed off" (*p=3*) and proceed to cross off all the multiples of this prime:

$$2p=6, 3p=9, 4p=12, 5p=15, \ldots, x : x <= n$$

You continue until all the remaining elements > *p* are "crossed off". The primes will be those values remaining in the array that have not been eliminated.

Write a Java program that will input a positive integer $n > 1$ and implement *Eratosthenes' sieve* to output all of the prime numbers in the closed interval [2, *n*]. You should validate the input, but you do not need to provide a validation/correction loop (unless you want to).

Hint: Start with an `array` of the integers 2 through *n* and "cross-off" the non-primes by overwriting them with some value such as zero.