Classes Practice – Rational Numbers

A *rational number* is a number that can be expressed as the quotient of two integers, where the denominator is not zero. For instance, 1/2, 3/4, 40/17, and 5/1 are all rational numbers. Python does not have a rational data type, and therefore stores all rational numbers as floats or doubles. This can cause problems because some rational numbers, such as 1/3, cannot be represented exactly in decimal notation (at least not with a finite number of digits). In this lab, you will create a simple `Rational` class to better represent (positive) rational numbers.

1. Create a class called `rational`. Think about how a rational number could be represented. This information will become your `Rational` class's *implementation*. Add fields (variables) to your class that will represent the rational number. Should these be public or private?

2. Add two constructors to your class. One should be a "default" constructor (no arguments) that constructs the rational number 1/1. The other constructor should let the user create any rational number they want. This constructor will take two arguments: a numerator and a denominator. You should never let the user create a rational number with a denominator of zero. For now, you can just assume that the user won't do this.

   Example of usage:
   ```
   one = Rational()            # construct the rational number 1/1
   onehalf = Rational(1, 2)    # construct a rational number representing 1/2
   ```

3. Overload the __str__ method so that `print(onehalf)` will output `1/2` to the screen (assuming you have the `onehalf` declaration line from question 2).

4. Rational numbers are usually always given in lowest terms (where the numerator and denominator have no factors in common other than 1). Add a method called `reduce()` that reduces a rational number to lowest terms.

   Hint: there are lots of ways to write this method. A simple way is to find the greatest common divisor (gcd) of the numerator and denominator (the largest integer that is a factor of both). As long as the gcd is greater than 1, you know the rational is not in lowest terms and you can divide both the numerator and denominator by this factor. (You could write a gcd function outside your class definition, and use it in your class.)

5. Add a method to your class that lets you multiply two rational numbers together.

   Your method should take one rational number argument and ***return*** the product of the current rational number and the argument:

   ```
   a = Rational (1, 2)
   b = Rational (3, 4)
   c = a.multiply(b)       # a and b are unchanged, c is 3/8
   ```

6. Add more capabilities to your class; e.g., addition, taking the inverse of a rational, allowing for negative rational numbers, subtraction, division, preventing division by zero, testing if one rational is less than another.