## Carleton University Department of Systems and Computer Engineering SYSC 2100 — Algorithms and Data Structures — Winter 2021

## Lab 2 - Implementing an ADT as a Python Class

## **Submitting Lab Work for Grading**

Remember, you don't have to finish the lab by the end of your lab period. For this lab, the deadline for submitting your solutions to cuLearn for grading is 11:55 pm (Ottawa time) two days after your scheduled lab. Solutions that are emailed to your instructor or a TA will not be graded, even if they are emailed before the deadline.

Please read *Important Considerations When Submitting Files to cuLearn*, on the last page of the course outline.

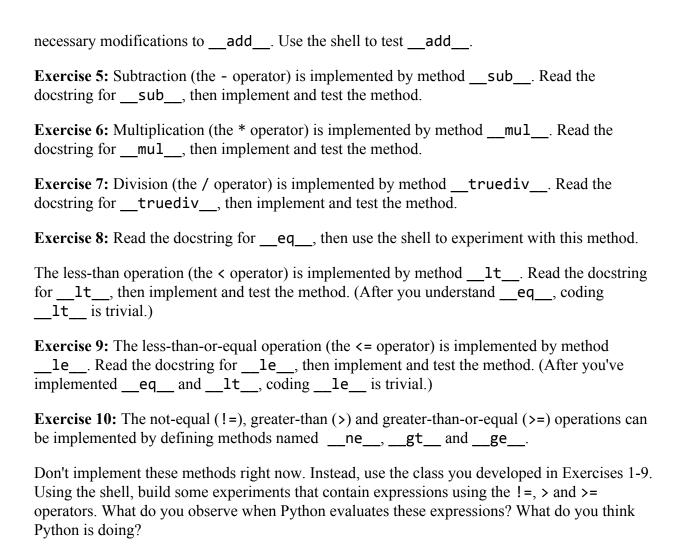
## **Getting Started**

Log on to cuLearn and download Fraction.py from the *Lab Materials* section of the main course page. The code in this file was adapted from the Fraction ADT presented in the course textbook, *Problem Solving with Algorithms and Data Structures using Python, Third Edition, Section 1.13.1*. The following changes were made to the code:

- Instance variables num and den have been renamed \_num and \_den, to denote that they are "private" attributes of Fraction objects.
- Type annotations were added to the method headers. Note: if a parameter is not annotated, we assume that its type is the enclosing class. For example, parameter self is not annotated, so we assume that its type is Fraction. Similarly, if a method's return type is not annotated, we assume that the method returns an instance of the class. For an example of this, see \_\_add\_\_.
- Docstrings were added to all the methods. Each doctring has a concise summary of what the method does and examples of tests that we can execute in the Python shell.
- "Stub" implementations have been provided for methods \_\_repr\_\_, numerator, denominator, \_\_le\_\_, \_\_lt\_\_, \_\_sub\_\_, \_\_mul\_\_ and \_\_truediv\_\_. If you call any of these methods on a Fraction object, it will throw a NotImplementedError exception.
- Method show has been removed (we won't need it).

Information about arithmetic operations on fractions can be found at the MathWorld website: <a href="https://mathworld.wolfram.com/Fraction.html">https://mathworld.wolfram.com/Fraction.html</a>.

Exercise 1: Open Fraction.py in Wing 101. Try this experiment:			
>>> f = Fraction(3, 4) >>> f			
Clearly, we need to implementrepr to provide the string representation of the object that will be displayed by the Python shell.			
Read the docstring forrepr Replace the raise statement with a correct implementation of the method. Use the shell to testrepr			
<b>Exercise 2:</b> Read the docstrings for accessor ("getter") methods numerator and denominator, then implement and test the methods.			
Exercise 3: Methodinit in the textbook's Fraction class has limitations:			
• A Fraction object's denominator must be positive. Code that creates negative fractions must ensure that parameter top is less than or equal to 0 and parameter bottom is greater than 0. This constraint will complicate the implementation of some of the arithmetic and comparison operations.			
• A <i>reduced fraction</i> is a fraction $a/b$ written in lowest terms, by dividing the numerator and denominator by their greatest common divisor. For example, $2/3$ is the reduced fraction of $8/12$ .			
For our purposes, we'll also include the following in our definition of reduced fractions:			
o if the numerator is equal to 0, the denominator is always 1;			
o if the numerator is not equal to 0, the denominator is always positive. This means that negative fractions always have a negative numerator and a positive denominator.			
The textbook's implementation ofinit doesn't produce reduced fractions. This means thatadd and the other methods that perform arithmetic operations must include code to reduce the fractions they produce. Code duplicated across multiple functions and methods is often an indication of poor programming style ("smelly code").			
Modifyinit so that newly created Fraction objects are reduced fractions. The docstring describes what this method should do after you've made the required changes. Note thatinit should throw a ValueException if the fraction's denominator is 0.			
Test your modified Fraction class thoroughly before you move on to the next exercise.			
<b>Exercise 4:</b> The textbook's implementation ofadd contains code to produce reduced fractions. It calls gcd, then cancels out the common terms in the new fraction's numerator and denominator. Now that you've modifiedinit,add no longer needs to do this. (When _add "calls" Fraction,init will put the new fraction in reduced form.) Make the			



*Instructions for submitting your lab work are on the next page.* 

Wrap Up

The submission deadlines for this lab are:

Lab Section	Lab Date/Time	Submission Deadline (Ottawa Time)
L5	Tuesday, 11:35 - 13:25	Thursday, Jan. 21, 23:55
L2	Thursday, 9:35 - 11:25	Saturday, Jan. 23, 23:55
L4	Thursday, 12:35 - 14:25	Saturday, Jan. 23, 23:55
L3	Friday, 9:35 - 11:25	Sunday, Jan. 24, 23:55
L1	Friday, 14:35 - 16:25	Sunday, Jan. 24, 23:55

To submit your lab work, go to the cuLearn page **for your lab section** (not the main course page). Submit Fraction.py. Ensure you submit the version of the file that contains your solutions, and not the unmodified file you downloaded from cuLearn! You are permitted to make changes to your solutions and resubmit the file as many times as you want, up to the deadline. Only the most recent submission is saved by cuLearn.

Last edited: Jan. 17, 2021