Homework 02 - Fractions

Re-submit Assignment

Due Feb 5 by 10pm **Points** 100 **Submitting** a file upload

The goal of this assignment is for you to get experience with Python classes and exceptions. You'll need to remember your elementary school math class on fractions because you're going to implement a fraction calculator that asks the user for two fractions and an operator and then prints the result. To make it easy, don't worry about reducing fractions to the least common denominator, just multiply the two denominators to find the greatest common denominator, then adjust the numerators appropriately.

Deliverables:

1. Use CRC Cards to identify the classes, responsibilities and collaborators in your design for the Fraction calculator.

   You may use any physical manifestation of CRC cards that you like, e.g. 3x5 index cards, scraps of paper, spreadsheets, or text documents.

   To help you to get started, think about what's involved? Start with a calculator, identify the responsibilities and collaborators, and then work your way through the solution until you feel you have a complete solution. My solution has only a Fraction class but your solution may be different. Be sure to identify the class, the responsibilities of that class, and the collaborators.

   If you choose to use 3x5 index cards (or just scraps of paper), take photos of your cards and submit those to Canvas. If you use an electronic solution, then upload the document with the classes, responsibilities and collaborators. I prefer index cards because they are easy to move around, stack, and rip up when a better idea comes along.

2. Write a Python program that includes:

- class Fraction with the following methods:
  - __init__(self, numerator, denominator)
    - e.g. f12: Fraction = Fraction(1, 2) # f12 represents the fraction 1/2
    - Raise a ValueError exception if the denominator == 0
  - plus(self, other: "Fraction") # **return a new instance of class Fraction** with the sum of self and other where other is another Fraction
  - minus(self, other: "Fraction") # **return a new instance of class Fraction** with the difference of self and other where other is another Fraction

    - times(self, other: "Fraction") # **return a new instance of class Fraction** with the product of self and other where other is another Fraction
    - divide(self, other: "Fraction") # **return a new instance of class Fraction** with the quotient of self and other where other is another Fraction
    - equal(self, other: "Fraction") # return True/False if the two fractions are equal - Hint: compare the product of the numerator of self and the denominator of other to the product of the numerator of other and the denominator of self. E.g. given 2/3 and 4/6, the two fractions equal if (2 * 6) == (3 * 4). This will be very helpful for your test cases.
    - __str__(self) # return a string to represent the Fraction
- a test_suite() function that demonstrates that your Fraction class works properly. For now, just use **print** or **assert** statements to demonstrate that your code works properly. Next week we'll explore some better techniques for automating test cases.
    - Each test case should include both the computed result and a string with the expected result so the user can easily determine if the calculated value is correct by comparing the computed and expected value. (See below for an example.
    - Your test suite must include at least one example of adding three fractions together in a single operation, e.g. 1/2 + 3/4 + 4/4 == 72/32
    - Your test suite should include examples of all of the operations.
- main() function that asks the user for the first numerator and denominator, the operator, and the second numerator and denominator, then prints the result of applying the operator to the two fractions.
- Be sure that your program handles error cases where the user enters invalid input for the fractions or operator.

You must include type hints for every function return type, parameter, and local variable.

Be sure to include a docstring at the top of your file to explain what the program does and a docstring at the top of every class/function/method to explain what the function/method/class does.

Here's the sample output from my solution:

```
Welcome to the fraction calculator!
Fraction 1 numerator: 1 <-- user entered 1
Fraction 1 denominator: 2 <-- user entered 1
Operation (+, -, *, /, ==): + <-- user entered +
Fraction 2 numerator: 3 <-- user entered 1
Fraction 1 denominator: 4 <-- user entered 1
1/2 + 3/4 = 10/8 <-- final result - The code converted 1/2 to 4/8 and 3/4 to 6/8 and returns 10/8
```

Here's an example of the code for one of my test cases:

```
f12: Fraction = Fraction(1, 2)
f44: Fraction = Fraction(4, 4)
```

```
print(f"{f12} + {f12} = {f12.plus(f12)} [4/4]")
print(f"{f44} - {f12} = {f44.minus(f12)} [4/8]")
```

```
Here's the output from the test cases.

1/2 + 1/2 = 4/4 [4/4]
4/4 - 1/2 = 4/8 [4/8]
```

The print statement in each test case includes the expected result so the reader can easily determine if the code is working properly by comparing the computed result against the expected result.

Here's a few test cases you should include in your test suite:

```
f12: Fraction = Fraction(1, 2)
f44: Fraction = Fraction(4, 4)
f128: Fraction = Fraction(12, 8)
f32: Fraction = Fraction(3, 2)
```

```
print(f"{f12} + {f12} = {f12.plus(f12)} [4/4]")
print(f"{f44} - {f12} = {f44.minus(f12)} [4/8]")
print(f"{f12} + {f44} = {f12.plus(f44)} [12/8]")
print(f"{f128} == {f32} is {f128.equal(f32)} [True]")
```

Note that these test cases require the user to do lots of work and are not automated or reliable. We'll address that issue next week.

Please attempt the solution on your own. If you get stuck, I've included a **framework of a solution** to help you to get started.