Lab 1: Python Review

GitHub: abhijit-baruah

Fraction Class

The textbook provides this minimally function `Fraction` class. You will complete several exercises to improve on the design of this custom data type.

The function `gcd` , defined below, is necessary for `Fraction` to work.

In [1]:

```
def gcd(m, n):
    """Greatest Common Divisor
    M&R listing 1.6: Greatest Common Divisor Function
    """
    #if m modulo(n) is zero then, n is the gcd and we return n anyway
    #In all other cases where: n=mx for some x, m!=n here is what follows

    while m % n != 0:
        oldm = m
        oldn = n

        m = oldn # the switch between m and n is made here for the second run under the whi
        n = oldm % oldn
    return n
```

In [2]:

```
gcd(-3, 6) #Checking if gcd functions works as expected
```

Out[2]:

```
3
```

Your Exercises

1. Implement these simple 'getter' methods for class `Fraction` :
   - `get_num` to return the numerator
   - `get_den` to return the denominator.
2. In many ways it would be better if all fractions were maintained in lowest terms right from the start. Modify the initializer for the `Fraction` class so that the GCD alogorithm is used to reduce fractions immediately. Notice that this means the `__add__` method no longer needs to reduce. Make the necessary modifications.
3. Implement the remaining relational operators to allow you to compare one `Fraction` object, with another.
   - `__gt__`
   - `__ge__`
   - `__lt__`
   - `__le__`

- `__ne__`

4. In the definition of fractions we assumed that negative fractions have a negative numerator and a positive denominator. Using a negative denominator would cause some of the relational operators to give incorrect results. In general, this is an unnecessary constraint. Modify the constructor to allow the user to pass a negative denominator so that all of the operators continue to work properly.

Your Solution

Implement your solution to the exercises by modifying the `Fraction` class, below, and add your code to it. To make it clear, please use docstrings and comments where appropriate to state which parts of the `Fraction` class are being modified, and for *which* exercise.

In [3]:

```
class Fraction:
    """A class to represent fractions

    This code needs to be improved according to the exercises!
    """
    def __init__(self, top, bottom):
        self.num = top
        self.den = bottom
        # Exercise 2: modification of initializer to reduce fractions
        self._newnum = self.num / gcd(top,bottom)
        self._newden = self.den / gcd(top,bottom)

        ## Part of Exercise 3 and 4: Modifying constructor to account for negative denomina

        # we construct the identifiers for the numerators and denominators of the functions
        # if at all there is a negative denominator then both numerator and denominator is
        # This should automatically take care of the issues that a negative denominator pos
        # in the fraction class.
        # At the same time, in accordance with the definition of fractions, any negative fr
        # a negative numerator and a positive denominator. This also conforms to our code.
        if self.den < 0:
            a=-self.num
            self.num=a
            b=-self.den
            self.den=b


    ## Exercise 1 ##

    # Getter methods to return numerator and denominator
    def get_num(self):
        return(self.num) #returns numerator (positive for positive fraction and negative ot
    def get_den(self):
        return(self.den) #returns denominator (always positive)

    def __str__(self):
        return str(self.num) + "/" + str(self.den)

    def show(self):
        """Display the fraction"""
        print(self.num, "/", self.den)

    ## Exercise 2: modification of add function after modifying initializers to reduce frac

    # since our fractions are already reduced to lowest, we ignored the reduction code in t
    def __add__(self, otherfraction):
        new_num = int(self._newnum*otherfraction._newden + self._newden*otherfraction._newn
        new_den = int(self._newden * otherfraction._newden)
        return Fraction(new_num, new_den)


    ## Exercise 3: Relational operators and consideration of negative fractions. ##

    # Although equality check is not affected by this, we still make the modifications to m
    # Since the modifications to our constructor takes care of negative fractions accordin
    # the incorrectness in relational operators due to an user input of a negative denomina
    # It remains to check for any two fractions a/b and x/y the relation between ay and bx.

    def __eq__(self, other):
```

```
        first_num = self.num * other.den
        second_num = other.num * self.den
        return first_num == second_num

    def __gt__(self, other):
        first_num = self.num * other.den
        second_num = other.num * self.den
        return first_num > second_num

    def __ge__(self, other):
        first_num = self.num * other.den
        second_num = other.num * self.den
        return first_num >= second_num

    def __lt__(self, other):
        first_num = self.num * other.den
        second_num = other.num * self.den
        return first_num < second_num

    def __le__(self, other):
        first_num = self.num * other.den
        second_num = other.num * self.den
        return first_num <= second_num

    def __ne__(self, other):
        first_num = self.num * other.den
        second_num = other.num * self.den
        return first_num != second_num
```

Testing

For full credit, you must also test your solution so that you can prove to the grade your solution works.

In [4]:

```
# Testing with positive fractions
x = Fraction(1, 2)
y = Fraction(2, 3)
```

In [5]:

```
# Testing with mixed fractions where user inputs a negative denominator
u = Fraction(3, -6)
v = Fraction(2, 5)
```

In [6]:

```
# Testing with both negtive fractions
p = Fraction(2,-5)
q = Fraction(-1, 3)
```

Test of addition

In [7]:

```
print(f'{x} + {y} =',x+y)
```

```
1/2 + 2/3 = 7/6
```

In [8]:

```
print(f'{u} + {v} =',u+v)
```

```
-3/6 + 2/5 = -1/10
```

In [9]:

```
print(f'{p} + {q} =',p+q)
```

```
-2/5 + -1/3 = -11/15
```

Test of Getter functions

In [10]:

```
print(p.get_num(),' ', p.get_den()) # For fraction p = -2/5
```

```
-2   5
```

In [11]:

```
print(x.get_num(),' ', y.get_den()) # For fraction x = 1/2 and y = 2/3
```

```
1   3
```

Test of Relational Operators

In [12]:

```
print(f'Is {x} = {y}?',x == y)
print(f'Is {x} > {y}?', x>y)
print(f'Is {x} >= {y}?', x>=y)
print(f'Is {x} < {y}?', x<y)
print(f'Is {x} <= {y}?', x<=y)
print(f'Is {x} != {y}?', x!=y)
```

```
Is 1/2 = 2/3? False
Is 1/2 > 2/3? False
Is 1/2 >= 2/3? False
Is 1/2 < 2/3? True
Is 1/2 <= 2/3? True
Is 1/2 != 2/3? True
```

In [13]:

```
print(f'Is {u} = {v}?',u == v)
print(f'Is {u} > {v}?', u>v)
print(f'Is {u} >= {v}?', u>=v)
print(f'Is {u} < {v}?', u<v)
print(f'Is {u} <= {v}?', u<=v)
print(f'Is {u} != {v}?', u!=v)
```

```
Is -3/6 = 2/5? False
Is -3/6 > 2/5? False
Is -3/6 >= 2/5? False
Is -3/6 < 2/5? True
Is -3/6 <= 2/5? True
Is -3/6 != 2/5? True
```

In [14]:

```
print(f'Is {p} = {q}?', p == q)
print(f'Is {p} > {q}?', p>q)
print(f'Is {p} >= {q}?', p>=q)
print(f'Is {p} < {q}?', p<q)
print(f'Is {p} <= {q}?', p<=q)
print(f'Is {p} != {q}?', p!=q)
```

```
Is -2/5 = -1/3? False
Is -2/5 > -1/3? False
Is -2/5 >= -1/3? False
Is -2/5 < -1/3? True
Is -2/5 <= -1/3? True
Is -2/5 != -1/3? True
```