```
"""
Homework 03 -- Fraction Testing and Debugging

Implement a fraction calculator including operations:
1.  use python magic methods to implement fraction calculator;
2.  use unittest to test all methods under all potential cases;
3.  use® debugger to debugging code.

"""
import unittest

class Fraction:
    """
    Implement addition, subtraction, multiplication, division and
comparison of two fractions
    """
    def __init__(self, numerator, denominator):
        """
        set the numerator and denominator of a fraction
        raise a ValueError Exception when the denominator is zero
        """
        self.num = numerator
        self.den = denominator

        if self.den == 0:
            raise ZeroDivisionError("Error! The denominator of a fraction
cannot be zero!")

    def __add__(self, other):
        """
        return a fraction with the addition of self and other
        """
        newnum = self.num * other.den + other.num * self.den
        newden = self.den * other.den
        return Fraction(newnum, newden)
        # return Fraction(newnum, newden)

    def __sub__(self, other):
        """
        return a fraction with the subtraction of self and other
        """
        newnum = self.num * other.den - other.num * self.den
        newden = self.den * other.den
        return Fraction(newnum, newden)

    def __mul__(self, other):
        """
        return a fraction with the product of self and other
        """
        newnum = self.num * other.num
        newden = self.den * other.den
        return Fraction(newnum, newden)

    def __truediv__(self, other):
```

```
        """
        return a fraction representing the result that self divided by
other
        """
        newnum = self.num * other.den
        newden = self.den * other.num

        if newden == 0:
            raise ZeroDivisionError("Error! The denominator of a fraction
cannot be zero!")

        return Fraction(newnum, newden)

    def __eq__(self, other):
        """
        identify whether self is equal to other, return True/False
        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.
        """
        return self.num * other.den == self.den * other.num

    def __ne__(self, other):
        """
        identify whether self is not equal to other, return True/False
        """
        return self.num * other.den != self.den * other.num

    def __lt__(self, other):
        """
        identify whether self is less than other, return True/False
        """
        return self.num * other.den < self.den * other.num

    def __le__(self, other):
        """
        identify whether self is less than or equal to other, return
True/False
        """
        return self.num * other.den <= self.den * other.num

    def __gt__(self, other):
        """
        identify whether self is greater than other, return True/False
        """
        return self.num * other.den > self.den * other.num

    def __ge__(self, other):
        """
        identify whether self is greater than or equal to other, return
True/False
        """
        return self.num * other.den >= self.den * other.num
```

```
    def __str__(self):
        """
        return a string to display the Fraction
        """
        return "{}/{}".format(self.num , self.den)


def get_number(prompt):
    """
    read and return an integer from the user.
    cite: hw02-outline.py
    """
    while True:
        inp = input(prompt)
        try:
            return float(inp)
        except ValueError:
            print('Error:', inp, 'is not a number. Please try again...')


def get_fraction():
    while True:
        num = get_number("Enter the numerator:")
        den = get_number("Enter the denominator:")

        try:
            f = Fraction(num, den)
            return f
        except ZeroDivisionError as e:
            print(e)


def compute(f1, operator, f2):
    result = None

    if operator == '+':
        result = (f1 + f2)
    elif operator == '-':
        result = (f1 - f2)
    elif operator == '*':
        result = (f1 * f2)
    elif operator == '/':
        result = (f1 / f2)
    elif operator == '==':
        result = (f1 == f2)
    elif operator == '!=':
        result = (f1 != f2)
    elif operator == '<':
        result = (f1 < f2)
    elif operator == '<=':
        result = (f1 <= f2)
    elif operator == '>':
        result = (f1 > f2)
```

```
    elif operator == '>=':
        result = (f1 >= f2)
    else:
        print('Error! Invalid operation!')

    return result


def main():
    """
    Fraction Calculator
    """
    print('Welcome to the fraction calculator!')

    f1 = get_fraction()

    operator = input("\nOperation (+, -, *, /, ==, !=, <, <=, >, >=):")
    while True:
        if operator in ['+', '-', '*', '/', '==', '!=', '<', '<=', '>',
'>=']:
            break
        else:
            print("Error!", operator, "is an invalid operator! Please try
again...")
            operator = input("Operation (+, -, *, /, ==):")

    f2 = get_fraction()

    result = compute(f1, operator, f2)

    print("\nThe result is\n", f1, operator, f2, "=", result)


class FractionTest(unittest.TestCase):
    def test_init(self):
        """ verify fraction's numerator and denominator are setting
properly """
        f = Fraction(1, 2)
        self.assertTrue(f)
        with self.assertRaises(ZeroDivisionError):
            f2 = Fraction(1, 0)

    def test_str(self):
        """ verify that __str__ works properly """
        f = Fraction(1, 2)
        self.assertTrue(f)

    def test_add(self):
        """ verify that fraction addition works properly """
        f1 = Fraction(1, 2)
        f2 = Fraction(1.2, 2.5)
        f3 = Fraction(2, 3)
        self.assertTrue((f1 + f2) == Fraction(4.9, 5.0))
        self.assertTrue((f1 + f3) == Fraction(7, 6))
```

```
        self.assertTrue((f1 + f1 + f1) == Fraction(12, 8))

    def test_sub(self):
        """ verify that fraction subtraction works properly """
        f1 = Fraction(1, 2)
        f2 = Fraction(1, 4)
        f3 = Fraction(1, 8)
        self.assertTrue((f1 - f1) == Fraction(0, 4))
        self.assertTrue((f1 - f2) == Fraction(2, 8))
        self.assertTrue((f3 - f1) == Fraction(-6, 16))
        self.assertTrue((f1 - f2 - f3) == Fraction(8, 64))

    def test_mul(self):
        """ verify that fraction multiplication works properly """
        f1 = Fraction(1, 2)
        f2 = Fraction(1.2, 2.5)
        f3 = Fraction(0, 3)
        self.assertTrue((f1 * f2) == Fraction(1.2, 5.0))
        self.assertTrue((f1 * f3) == Fraction(0, 6))
        self.assertTrue((f1 * f1 * f1) == Fraction(1, 8))

    def test_truediv(self):
        """ verify that fraction division works properly """
        f1 = Fraction(1, 2)
        f2 = Fraction(1.2, 2.5)
        f3 = Fraction(0, 3)
        f4 = Fraction(2, 3)
        self.assertTrue((f1 / f2) == Fraction(2.5, 2.4))
        self.assertTrue((f1 / f4) == Fraction(3, 4))
        with self.assertRaises(ZeroDivisionError):
            f = f1 / f3

    def test_eq(self):
        """ verify that fraction equality works properly """
        f1 = Fraction(1, 2)
        f2 = Fraction(-1.2, -2.4)
        f3 = Fraction(6, 9)
        self.assertTrue(f1 == f1)
        self.assertTrue(f1 == f2)
        self.assertTrue(f2 == f2)
        self.assertFalse(f1 == f3)

    def test_ne(self):
        """ verify that fraction inequality works properly """
        f1 = Fraction(1, 2)
        f2 = Fraction(-1.2, -2.4)
        f3 = Fraction(6, 9)
        self.assertFalse(f1 != f1)
        self.assertFalse(f1 != f2)
        self.assertTrue(f2 != f3)
        self.assertTrue(f1 != f3)

    def test_lt(self):
```

```
        """ verify that fraction comparison -- less than works properly
"""
        f1 = Fraction(1, 2)
        f2 = Fraction(-1, 2)
        f3 = Fraction(2, 3)
        self.assertTrue(f2 < f1)
        self.assertFalse(f1 < f1)
        self.assertTrue(f1 < f3)
        self.assertFalse(f3 < f1)

    def test_le(self):
        """ verify that fraction comparison -- less than and equal to
works properly """
        f1 = Fraction(1, 2)
        f2 = Fraction(-1, 2)
        f3 = Fraction(2, 3)
        self.assertTrue(f2 <= f1)
        self.assertTrue(f1 <= f1)
        self.assertTrue(f1 <= f3)
        self.assertFalse(f3 <= f1)

    def test_gt(self):
        """ verify that fraction comparison -- greater than works properly
"""
        f1 = Fraction(1, 2)
        f2 = Fraction(-1, 2)
        f3 = Fraction(2, 3)
        self.assertFalse(f2 > f1)
        self.assertFalse(f1 > f1)
        self.assertFalse(f1 > f3)
        self.assertTrue(f3 > f1)

    def test_ge(self):
        """ verify that fraction comparison -- greater than and equal to
works properly """
        f1 = Fraction(1, 2)
        f2 = Fraction(-1, 2)
        f3 = Fraction(2, 3)
        self.assertFalse(f2 >= f1)
        self.assertTrue(f1 >= f1)
        self.assertFalse(f1 >= f3)
        self.assertTrue(f3 >= f1)

if __name__ == '__main__':
    unittest.main(exit=False, verbosity=2)
```