statementtesting/euclid.py (Page 1 of 1)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# euclid.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

def gcd(i, j):

    if (i == 0) and (j == 0):
        raise ZeroDivisionError(
            'gcd(i,j) is undefined if i and j are 0')
    i = abs(i)
    j = abs(j)
    while j != 0:  # Euclid's algorithm
        i, j = j, i%j
    return i

#-----------------------------------------------------------------------

def lcm(i, j):

    if (i == 0) or (j == 0):
        raise ZeroDivisionError(
            'lcm(i,j) is undefined if i or j is 0')
    i = abs(i)
    j = abs(j)
    return (i // gcd(i, j)) * j
```

blank (Page 1 of 1)

```
This page is intentionally blank.
```

statementtesting/fraction.py (Page 1 of 2)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# fraction.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import euclid

#-----------------------------------------------------------------------

class Fraction:

    def __init__(self, num=0, den=1):
        if den == 0:
            raise ZeroDivisionError('Denominator cannot be 0')
        self._num = num
        self._den = den
        self._normalize()

    def _normalize(self):
        if self._den < 0:
            self._num *= -1
            self._den *= -1
        if self._num == 0:
            self._den = 1
        else:
            gcden = euclid.gcd(self._num, self._den)
            self._num //= gcden
            self._den //= gcden

    def __str__(self):
        if self._den == 1:
            return str(self._num)
        return '%d/%d' % (self._num, self._den)

    def __hash__(self):
        return hash((self._num, self._den))   # Use tuple's __hash__().

    def __eq__(self, other):
        return (self._num == other._num) and (self._den == other._den)

    def __ne__(self, other):
        return not self == other

    def __lt__(self, other):
        return (self._num * other._den) < (other._num * self._den)

    def __gt__(self, other):
        return (self._num * other._den) > (other._num * self._den)

    def __le__(self, other):
        return not self > other

    def __ge__(self, other):
        return not self < other

    def __neg__(self):
        return Fraction(-self._num, self._den)

    def __add__(self, other):
        new_num = (self._num * other._den) + (other._num * self._den)
        new_den = self._den * other._den
        return Fraction(new_num, new_den)

```

statementtesting/fraction.py (Page 2 of 2)

```python
    def __sub__(self, other):
        new_num = (self._num * other._den) - (other._num * self._den)
        new_den = self._den * other._den
        return Fraction(new_num, new_den)

    def __mul__(self, other):
        new_num = self._num * other._num
        new_den = self._den * other._den
        return Fraction(new_num, new_den)

    def __truediv__(self, other):
        new_num = self._num * other._den
        new_den = self._den * other._num
        return Fraction(new_num, new_den)
```

statementtesting/fractionclient.py (Page 1 of 2)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# fractionclient.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import sys
import fraction

def main():

    try:

        line = input('Numerator 1: ')
        num1 = int(line)
        line = input('Denominator 1: ')
        den1 = int(line)
        line = input('Numerator 2: ')
        num2 = int(line)
        line = input('Denominator 2: ')
        den2 = int(line)

        frac1 = fraction.Fraction(num1, den1)
        print('frac1:', str(frac1))  # Same as frac1.__str__()

        frac2 = fraction.Fraction(num2, den2)
        print('frac2:', frac2)  # print() calls str(frac2)
                                # Same as frac2.__str__()

        print('frac1 hashcode:', hash(frac1)) # Same as frac1.__hash__()

        if frac1 == frac2:  # Same as frac1.__eq__(frac2)
            print('frac1 equals frac2')
        if frac1 != frac2:  # Same as frac1.__ne__(frac2)
            print('frac1 does not equal frac2')
        if frac1 < frac2:  # Same as frac1.__lt__(frac2)
            print('frac1 is less than frac2')
        if frac1 > frac2:  # Same as frac1.__gt__(frac2)
            print('frac1 is greater than frac2')
        if frac1 <= frac2:  # Same as frac1.__le__(frac2)
            print('frac1 is less than or equal to frac2')
        if frac1 >= frac2:  # Same as frac1.__ge__(frac2)
            print('frac1 is greater than or equal to frac2')

        frac3 = -frac1  # Same as frac1.__neg__()
        print('-frac1:', frac3)

        frac3 = frac1 + frac2  # Same as frac1.__add__(frac2)
        print('frac1 + frac2:', frac3)

        frac3 = frac1 - frac2  # Same as frac1.__sub__(frac2)
        print('frac1 - frac2:', frac3)

        frac3 = frac1 * frac2  # Same as frac1.__mul__(frac2)
        print('frac1 * frac2:', frac3)

        frac3 = frac1 / frac2  # Same as frac1.__truediv__(frac2)
        print('frac1 / frac2:', frac3)

    except Exception as ex:
        print(ex, file=sys.stderr)

#-----------------------------------------------------------------------

```

statementtesting/fractionclient.py (Page 2 of 2)

```python
if __name__ == '__main__':
    main()
```

statementtesting/buildandrun (Page 1 of 1)

```
#!/usr/bin/env bash

#-----------------------------------------------------------------------
# buildandrun
# Author: Bob Dondero
#-----------------------------------------------------------------------

set -o verbose

# Create file .coverage
python -m coverage run fractionclient.py

# Create directory htmlcov
python -m coverage html

# View the results, htmlcov/index.html, using a browser
```

statementtesting/buildandrun.bat (Page 1 of 1)

```
@ECHO OFF
REM -------------------------------------------------------------------
REM buildandrun.bat
REM Author: Bob Dondero
REM -------------------------------------------------------------------

REM Create file .coverage
python -m coverage run fractionclient.py

REM Create directory htmlcov
python -m coverage html

REM View the results, htmlcov\index.html, using a browser
```

testautomationgood/testfraction.py (Page 1 of 1)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# testfraction.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import unittest
import fraction

class FractionTestCase(unittest.TestCase):
    def setUp(self):
        self._f1 = fraction.Fraction(1, 2)
        self._f2 = fraction.Fraction(3, 4)
    def tearDown(self):
        self._f1 = None
        self._f2 = None

class AddTestCase(FractionTestCase):
    def runTest(self):
        sum = self._f1 + self._f2
        expected = fraction.Fraction(5, 4)
        self.assertEqual(sum, expected, 'Incorrect sum')

class SubTestCase(FractionTestCase):
    def runTest(self):
        diff = self._f1 - self._f2
        expected = fraction.Fraction(-1, 4)
        self.assertEqual(diff, expected, 'Incorrect difference')

class MulTestCase(FractionTestCase):
    def runTest(self):
        prod = self._f1 * self._f2
        expected = fraction.Fraction(3, 8)
        self.assertEqual(prod, expected, 'Incorrect product')

class DivTestCase(FractionTestCase):
    def runTest(self):
        quo = self._f1 / self._f2
        expected = fraction.Fraction(2, 3)
        self.assertEqual(quo, expected, 'Incorrect quotient')

class ZeroDenCase(FractionTestCase):
    def runTest(self):
        try:
            fraction.Fraction(1, 0)
            self.fail('Incorrect handling of zero denom')
        except ZeroDivisionError:
            pass

# And many more...

#-----------------------------------------------------------------------

if __name__ == "__main__":
    unittest.main()
```

blank (Page 1 of 1)

This page is intentionally blank.

testautomationgood/buildandrun (Page 1 of 1)

```
#!/usr/bin/env bash

#-----------------------------------------------------------------------
# buildandrun
# Author: Bob Dondero
#-----------------------------------------------------------------------

set -o verbose

# Run unit tests
python testfraction.py
```

testautomationgood/buildandrun.bat (Page 1 of 1)

```
@ECHO OFF
REM -------------------------------------------------------------------
REM buildandrun.bat
REM Author: Bob Dondero
REM -------------------------------------------------------------------

REM Run unit tests
python testfraction.py
```

testautomationbad/fraction.py (Page 1 of 2)

```python
#!/usr/bin/env python

#-----------------------------------------------------------------------
# fraction.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import euclid

#-----------------------------------------------------------------------

class Fraction:

    def __init__(self, num=0, den=1):
        if den == 0:
            raise ZeroDivisionError('Denominator cannot be 0')
        self._num = num
        self._den = den
        self._normalize()

    def _normalize(self):
        if self._den < 0:
            self._num *= -1
            self._den *= -1
        if self._num == 0:
            self._den = 1
        else:
            gcden = euclid.gcd(self._num, self._den)
            self._num //= gcden
            self._den //= gcden

    def __str__(self):
        if self._den == 1:
            return str(self._num)
        return '%d/%d' % (self._num, self._den)

    def __hash__(self):
        return hash((self._num, self._den))   # Use tuple's __hash__().

    def __eq__(self, other):
        return (self._num == other._num) and (self._den == other._den)

    def __ne__(self, other):
        return not self == other

    def __lt__(self, other):
        return (self._num * other._den) < (other._num * self._den)

    def __gt__(self, other):
        return (self._num * other._den) > (other._num * self._den)

    def __le__(self, other):
        return not self > other

    def __ge__(self, other):
        return not self < other

    def __neg__(self):
        return Fraction(-self._num, self._den)

    def __add__(self, other):
        new_num = (self._num * other._den) + (other._num * self._den)
        new_den = self._den * other._den
        return Fraction(new_num, new_den)

```

testautomationbad/fraction.py (Page 2 of 2)

```python
    def __sub__(self, other):
        new_num = (self._num * other._den) - (other._num * self._den)
        new_den = self._den * other._den
        return Fraction(new_num, new_den)

    def __mul__(self, other):
        new_num = self._num * other._num
        new_den = self._den * other._num  # error here
        return Fraction(new_num, new_den)

    def __truediv__(self, other):
        new_num = self._num * other._den
        new_den = self._den * other._num
        return Fraction(new_num, new_den)
```