

statementtesting/euclid.py (Page 1 of 1)

```
1: #!/usr/bin/env python
2:
3: #-----
4: # euclid.py
5: # Author: Bob Dondero
6: #-----
7:
8: def gcd(i, j):
9:
10:     if (i == 0) and (j == 0):
11:         raise ZeroDivisionError(
12:             'gcd(i,j) is undefined if i and j are 0')
13:     i = abs(i)
14:     j = abs(j)
15:     while j != 0: # Euclid's algorithm
16:         i, j = j, i%j
17:     return i
18:
19: #-----
20:
21: def lcm(i, j):
22:
23:     if (i == 0) or (j == 0):
24:         raise ZeroDivisionError(
25:             'lcm(i,j) is undefined if i or j is 0')
26:     i = abs(i)
27:     j = abs(j)
28:     return (i // gcd(i, j)) * j
```

blank (Page 1 of 1)

```
1: This page is intentionally blank.
```

statementtesting/fraction.py (Page 1 of 2)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # fraction.py
5: # Author: Bob Dondero
6: #-----
7:
8: import euclid
9:
10: #-----
11:
12: class Fraction:
13:
14:     def __init__(self, num=0, den=1):
15:         if den == 0:
16:             raise ZeroDivisionError('Denominator cannot be 0')
17:         self._num = num
18:         self._den = den
19:         self._normalize()
20:
21:     def _normalize(self):
22:         if self._den < 0:
23:             self._num *= -1
24:             self._den *= -1
25:         if self._num == 0:
26:             self._den = 1
27:         else:
28:             gcden = euclid.gcd(self._num, self._den)
29:             self._num //= gcden
30:             self._den //= gcden
31:
32:     def __str__(self):
33:         if self._den == 1:
34:             return str(self._num)
35:         return '%d/%d' % (self._num, self._den)
36:
37:     def __hash__(self):
38:         return hash((self._num, self._den)) # Use tuple's __hash__().
39:
40:     def __eq__(self, other):
41:         return (self._num == other._num) and (self._den == other._den)
42:
43:     def __ne__(self, other):
44:         return not self == other
45:
46:     def __lt__(self, other):
47:         return (self._num * other._den) < (other._num * self._den)
48:
49:     def __gt__(self, other):
50:         return (self._num * other._den) > (other._num * self._den)
51:
52:     def __le__(self, other):
53:         return not self > other
54:
55:     def __ge__(self, other):
56:         return not self < other
57:
58:     def __neg__(self):
59:         return Fraction(-self._num, self._den)
60:
61:     def __add__(self, other):
62:         new_num = (self._num * other._den) + (other._num * self._den)
63:         new_den = self._den * other._den
64:         return Fraction(new_num, new_den)
65:

```

statementtesting/fraction.py (Page 2 of 2)

```

66:     def __sub__(self, other):
67:         new_num = (self._num * other._den) - (other._num * self._den)
68:         new_den = self._den * other._den
69:         return Fraction(new_num, new_den)
70:
71:     def __mul__(self, other):
72:         new_num = self._num * other._num
73:         new_den = self._den * other._den
74:         return Fraction(new_num, new_den)
75:
76:     def __truediv__(self, other):
77:         new_num = self._num * other._den
78:         new_den = self._den * other._num
79:         return Fraction(new_num, new_den)

```

statementtesting/fractionclient.py (Page 1 of 2)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # fractionclient.py
5: # Author: Bob Dondero
6: #-----
7:
8: import sys
9: import fraction
10:
11: def main():
12:
13:     try:
14:
15:         line = input('Numerator 1: ')
16:         num1 = int(line)
17:         line = input('Denominator 1: ')
18:         den1 = int(line)
19:         line = input('Numerator 2: ')
20:         num2 = int(line)
21:         line = input('Denominator 2: ')
22:         den2 = int(line)
23:
24:         frac1 = fraction.Fraction(num1, den1)
25:         print('frac1:', str(frac1)) # Same as frac1.__str__()
26:
27:         frac2 = fraction.Fraction(num2, den2)
28:         print('frac2:', frac2) # print() calls str(frac2)
29:                               # Same as frac2.__str__()
30:
31:         print('frac1 hashcode:', hash(frac1)) # Same as frac1.__hash__()
32:
33:         if frac1 == frac2: # Same as frac1.__eq__(frac2)
34:             print('frac1 equals frac2')
35:         if frac1 != frac2: # Same as frac1.__ne__(frac2)
36:             print('frac1 does not equal frac2')
37:         if frac1 < frac2: # Same as frac1.__lt__(frac2)
38:             print('frac1 is less than frac2')
39:         if frac1 > frac2: # Same as frac1.__gt__(frac2)
40:             print('frac1 is greater than frac2')
41:         if frac1 <= frac2: # Same as frac1.__le__(frac2)
42:             print('frac1 is less than or equal to frac2')
43:         if frac1 >= frac2: # Same as frac1.__ge__(frac2)
44:             print('frac1 is greater than or equal to frac2')
45:
46:         frac3 = -frac1 # Same as frac1.__neg__()
47:         print('-frac1:', frac3)
48:
49:         frac3 = frac1 + frac2 # Same as frac1.__add__(frac2)
50:         print('frac1 + frac2:', frac3)
51:
52:         frac3 = frac1 - frac2 # Same as frac1.__sub__(frac2)
53:         print('frac1 - frac2:', frac3)
54:
55:         frac3 = frac1 * frac2 # Same as frac1.__mul__(frac2)
56:         print('frac1 * frac2:', frac3)
57:
58:         frac3 = frac1 / frac2 # Same as frac1.__truediv__(frac2)
59:         print('frac1 / frac2:', frac3)
60:
61:     except Exception as ex:
62:         print(ex, file=sys.stderr)
63:
64: #-----
65:

```

statementtesting/fractionclient.py (Page 2 of 2)

```

66: if __name__ == '__main__':
67:     main()

```

statementtesting/buildandrun (Page 1 of 1)

```
1: #!/usr/bin/env bash
2:
3: #-----
4: # buildandrun
5: # Author: Bob Dondero
6: #-----
7:
8: set -o verbose
9:
10: # Create file .coverage
11: python -m coverage run fractionclient.py
12:
13: # Create directory htmlcov
14: python -m coverage html
15:
16: # View the results, htmlcov/index.html, using a browser
```

statementtesting/buildandrun.bat (Page 1 of 1)

```
1: @ECHO OFF
2: REM -----
3: REM buildandrun.bat
4: REM Author: Bob Dondero
5: REM -----
6:
7: REM Create file .coverage
8: python -m coverage run fractionclient.py
9:
10: REM Create directory htmlcov
11: python -m coverage html
12:
13: REM View the results, htmlcov\index.html, using a browser
```

testautomationgood/testfraction.py (Page 1 of 1)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # testfraction.py
5: # Author: Bob Dondero
6: #-----
7:
8: import unittest
9: import fraction
10:
11: class FractionTestCase(unittest.TestCase):
12:     def setUp(self):
13:         self._f1 = fraction.Fraction(1, 2)
14:         self._f2 = fraction.Fraction(3, 4)
15:     def tearDown(self):
16:         self._f1 = None
17:         self._f2 = None
18:
19: class AddTestCase(FractionTestCase):
20:     def runTest(self):
21:         sum = self._f1 + self._f2
22:         expected = fraction.Fraction(5, 4)
23:         self.assertEqual(sum, expected, 'Incorrect sum')
24:
25: class SubTestCase(FractionTestCase):
26:     def runTest(self):
27:         diff = self._f1 - self._f2
28:         expected = fraction.Fraction(-1, 4)
29:         self.assertEqual(diff, expected, 'Incorrect difference')
30:
31: class MulTestCase(FractionTestCase):
32:     def runTest(self):
33:         prod = self._f1 * self._f2
34:         expected = fraction.Fraction(3, 8)
35:         self.assertEqual(prod, expected, 'Incorrect product')
36:
37: class DivTestCase(FractionTestCase):
38:     def runTest(self):
39:         quo = self._f1 / self._f2
40:         expected = fraction.Fraction(2, 3)
41:         self.assertEqual(quo, expected, 'Incorrect quotient')
42:
43: class ZeroDenCase(FractionTestCase):
44:     def runTest(self):
45:         try:
46:             fraction.Fraction(1, 0)
47:             self.fail('Incorrect handling of zero denom')
48:         except ZeroDivisionError:
49:             pass
50:
51: # And many more...
52:
53: #-----
54:
55: if __name__ == "__main__":
56:     unittest.main()

```

blank (Page 1 of 1)

```

1: This page is intentionally blank.

```

testautomationgood/buildandrun (Page 1 of 1)

```
1: #!/usr/bin/env bash
2:
3: #-----
4: # buildandrun
5: # Author: Bob Dondero
6: #-----
7:
8: set -o verbose
9:
10: # Run unit tests
11: python testfraction.py
```

testautomationgood/buildandrun.bat (Page 1 of 1)

```
1: @ECHO OFF
2: REM -----
3: REM buildandrun.bat
4: REM Author: Bob Dondero
5: REM -----
6:
7: REM Run unit tests
8: python testfraction.py
```

testautomationbad/fraction.py (Page 1 of 2)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # fraction.py
5: # Author: Bob Dondero
6: #-----
7:
8: import euclid
9:
10: #-----
11:
12: class Fraction:
13:
14:     def __init__(self, num=0, den=1):
15:         if den == 0:
16:             raise ZeroDivisionError('Denominator cannot be 0')
17:         self._num = num
18:         self._den = den
19:         self._normalize()
20:
21:     def _normalize(self):
22:         if self._den < 0:
23:             self._num *= -1
24:             self._den *= -1
25:         if self._num == 0:
26:             self._den = 1
27:         else:
28:             gcden = euclid.gcd(self._num, self._den)
29:             self._num //= gcden
30:             self._den //= gcden
31:
32:     def __str__(self):
33:         if self._den == 1:
34:             return str(self._num)
35:         return '%d/%d' % (self._num, self._den)
36:
37:     def __hash__(self):
38:         return hash((self._num, self._den)) # Use tuple's __hash__().
39:
40:     def __eq__(self, other):
41:         return (self._num == other._num) and (self._den == other._den)
42:
43:     def __ne__(self, other):
44:         return not self == other
45:
46:     def __lt__(self, other):
47:         return (self._num * other._den) < (other._num * self._den)
48:
49:     def __gt__(self, other):
50:         return (self._num * other._den) > (other._num * self._den)
51:
52:     def __le__(self, other):
53:         return not self > other
54:
55:     def __ge__(self, other):
56:         return not self < other
57:
58:     def __neg__(self):
59:         return Fraction(-self._num, self._den)
60:
61:     def __add__(self, other):
62:         new_num = (self._num * other._den) + (other._num * self._den)
63:         new_den = self._den * other._den
64:         return Fraction(new_num, new_den)
65:

```

testautomationbad/fraction.py (Page 2 of 2)

```

66:     def __sub__(self, other):
67:         new_num = (self._num * other._den) - (other._num * self._den)
68:         new_den = self._den * other._den
69:         return Fraction(new_num, new_den)
70:
71:     def __mul__(self, other):
72:         new_num = self._num * other._num
73:         new_den = self._den * other._num # error here
74:         return Fraction(new_num, new_den)
75:
76:     def __truediv__(self, other):
77:         new_num = self._num * other._den
78:         new_den = self._den * other._num
79:         return Fraction(new_num, new_den)

```