

Fraction/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.
```

Fraction/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 __eq__(self, other):
38:         return (self._num == other._num) and (self._den == other._den)
39:
40:     def __ne__(self, other):
41:         return not self == other
42:
43:     def __lt__(self, other):
44:         return (self._num * other._den) < (other._num * self._den)
45:
46:     def __gt__(self, other):
47:         return (self._num * other._den) > (other._num * self._den)
48:
49:     def __le__(self, other):
50:         return not self > other
51:
52:     def __ge__(self, other):
53:         return not self < other
54:
55:     def __neg__(self):
56:         return Fraction(-self._num, self._den)
57:
58:     def __add__(self, other):
59:         new_num = (self._num * other._den) + (other._num * self._den)
60:         new_den = self._den * other._den
61:         return Fraction(new_num, new_den)
62:
63:     def __sub__(self, other):
64:         new_num = (self._num * other._den) - (other._num * self._den)
65:         new_den = self._den * other._den

```

Fraction/fraction.py (Page 2 of 2)

```

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

```

Fraction/fractionclient.py (Page 1 of 1)

```

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:         if frac1 == frac2: # Same as frac1.__eq__(frac2)
32:             print('frac1 equals frac2')
33:         if frac1 != frac2: # Same as frac1.__ne__(frac2)
34:             print('frac1 does not equal frac2')
35:         if frac1 < frac2: # Same as frac1.__lt__(frac2)
36:             print('frac1 is less than frac2')
37:         if frac1 > frac2: # Same as frac1.__gt__(frac2)
38:             print('frac1 is greater than frac2')
39:         if frac1 <= frac2: # Same as frac1.__le__(frac2)
40:             print('frac1 is less than or equal to frac2')
41:         if frac1 >= frac2: # Same as frac1.__ge__(frac2)
42:             print('frac1 is greater than or equal to frac2')
43:
44:         frac3 = -frac1 # Same as frac1.__neg__()
45:         print('-frac1:', frac3)
46:
47:         frac3 = frac1 + frac2 # Same as frac1.__add__(frac2)
48:         print('frac1 + frac2:', frac3)
49:
50:         frac3 = frac1 - frac2 # Same as frac1.__sub__(frac2)
51:         print('frac1 - frac2:', frac3)
52:
53:         frac3 = frac1 * frac2 # Same as frac1.__mul__(frac2)
54:         print('frac1 * frac2:', frac3)
55:
56:         frac3 = frac1 / frac2 # Same as frac1.__truediv__(frac2)
57:         print('frac1 / frac2:', frac3)
58:
59:     except Exception as ex:
60:         print(str(ex), file=sys.stderr)
61:
62: #-----
63:
64: if __name__ == '__main__':
65:     main()

```

blank (Page 1 of 1)

```

1: This page is intentionally blank.

```

PennyTest/runserver.py (Page 1 of 1)

```
1: #!/usr/bin/env python
2:
3: #-----
4: # runserver.py
5: # Author: Bob Dondero
6: #-----
7:
8: import sys
9: import penny
10:
11: def main():
12:
13:     if len(sys.argv) != 2:
14:         print(f'usage: {sys.argv[0]} port', file=sys.stderr)
15:         sys.exit(1)
16:
17:     try:
18:         port = int(sys.argv[1])
19:     except Exception:
20:         print(f'{sys.argv[0]}: Port must be an integer.',
21:               file=sys.stderr)
22:         sys.exit(1)
23:
24:     try:
25:         penny.app.run(host='0.0.0.0', port=port, debug=True)
26:     except Exception as ex:
27:         print(f'{sys.argv[0]}: {ex}', file=sys.stderr)
28:         sys.exit(1)
29:
30: if __name__ == '__main__':
31:     main()
```

PennyTest/penny.sql (Page 1 of 1)

```
1: DROP TABLE IF EXISTS books;
2:
3: CREATE TABLE books (isbn TEXT PRIMARY KEY, author TEXT, title TEXT);
4:
5: INSERT INTO books (isbn, author, title)
6:     VALUES ('123', 'Kernighan', 'The Practice of Programming');
7: INSERT INTO books (isbn, author, title)
8:     VALUES ('234', 'Kernighan', 'The C Programming Language');
9: INSERT INTO books (isbn, author, title)
10:    VALUES ('345', 'Sedgewick', 'Algorithms in C');
```

PennyTest/database.py (Page 1 of 1)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # database.py
5: # Author: Bob Dondero
6: #-----
7:
8: import sqlite3
9: import contextlib
10:
11: #-----
12:
13: _DATABASE_URL = 'file:penny.sqlite'
14:
15: #-----
16:
17: def get_books(author):
18:     books = []
19:
20:     with contextlib.closing(
21:         sqlite3.connect(_DATABASE_URL + '?mode=ro',
22:             isolation_level=None, uri=True)) as connection:
23:
24:         with contextlib.closing(connection.cursor()) as cursor:
25:
26:             query_str = '''
27:                 SELECT isbn, author, title FROM books
28:                 WHERE author LIKE ?
29:             '''
30:
31:             cursor.execute(query_str, [author+'%'])
32:
33:             table = cursor.fetchall()
34:             for row in table:
35:                 book = {'isbn': row[0], 'author': row[1],
36:                     'title': row[2]}
37:                 books.append(book)
38:
39:     return books
40:

```

PennyTest/penny.py (Page 1 of 1)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # penny.py
5: # Author: Bob Dondero
6: #-----
7:
8: import json
9: import flask
10: import database
11:
12: #-----
13:
14: app = flask.Flask(__name__)
15:
16: #-----
17:
18: @app.route('/', methods=['GET'])
19: @app.route('/index', methods=['GET'])
20: def index():
21:
22:     return flask.send_file('index.html')
23:
24: #-----
25:
26: @app.route('/searchresults', methods=['GET'])
27: def search_results():
28:
29:     author = flask.request.args.get('author')
30:     if author is None:
31:         author = ''
32:     author = author.strip()
33:
34:     if author == '':
35:         books = []
36:     else:
37:         books = database.get_books(author) # Exception handling omitted
38:
39:     json_doc = json.dumps(books)
40:     response = flask.make_response(json_doc)
41:     response.headers['Content-Type'] = 'application/json'
42:     return response

```

PennyTest/index.html (Page 1 of 2)

```

1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Penny.com</title>
5:   </head>
6:
7:   <body>
8:
9:     <hr>
10:    Good <span id="ampmSpan"></span> and welcome to
11:    <strong>Penny.com</strong>
12:    <hr>
13:
14:    <h1>Author Search</h1>
15:    Please enter an author name:
16:    <input type="text" id="authorInput" autoFocus>
17:    <hr>
18:    <div id="resultsDiv"></div>
19:
20:    <hr>
21:    Date and time: <span id="datetimeSpan"></span><br>
22:    Created by <a href="https://www.cs.princeton.edu/~rdonero">
23:    Bob Donero</a>
24:    <hr>
25:
26:    <script src=
27:    "https://cdn.jsdelivr.net/npm/mustache@4.2.0/mustache.min.js">
28:    </script>
29:
30:    <!-- <script src="/static/mustache.min.js"></script> -->
31:
32:    <script>
33:
34:      'use strict';
35:
36:      function getAmPm() {
37:        let dateTime = new Date();
38:        let hours = dateTime.getHours();
39:        let amPm = (hours < 12) ? 'morning' : 'afternoon';
40:        let ampMspan = document.getElementById('ampmSpan');
41:        ampMspan.innerHTML = amPm;
42:      }
43:
44:      function getDateTime() {
45:        let dateTime = new Date();
46:        let datetimeSpan =
47:        document.getElementById('datetimeSpan');
48:        datetimeSpan.innerHTML = dateTime.toLocaleString();
49:      }
50:
51:      function convertToHtml(books) {
52:        let template = `
53:          {{#books}}
54:            {{isbn}}:
55:            <strong>{{author}}</strong>:
56:            {{title}}
57:            <br>
58:          {{/books}}
59:        `;
60:        let map = {books: books};
61:        let html = Mustache.render(template, map);
62:        return html;
63:      }
64:
65:      function handleResponse() {

```

PennyTest/index.html (Page 2 of 2)

```

66:        if (this.status !== 200) {
67:          alert('Error: Failed to fetch data from server');
68:          return;
69:        }
70:        let books = JSON.parse(this.response);
71:        let html = convertToHtml(books);
72:        let resultsDiv = document.getElementById('resultsDiv');
73:        resultsDiv.innerHTML = html;
74:      }
75:
76:      function handleError() {
77:        alert('Error: Failed to fetch data from server');
78:      }
79:
80:      let request = null;
81:
82:      function getResults() {
83:        let authorInput = document.getElementById('authorInput');
84:        let author = authorInput.value;
85:        let encodedAuthor = encodeURIComponent(author);
86:        let url = '/searchresults?author=' + encodedAuthor;
87:        if (request !== null)
88:          request.abort();
89:        request = new XMLHttpRequest();
90:        request.onload = handleResponse;
91:        request.onerror = handleError;
92:        request.open('GET', url);
93:        request.send();
94:      }
95:
96:      let timer = null;
97:
98:      function debouncedGetResults() {
99:        clearTimeout(timer);
100:        timer = window.setTimeout(getResults, 500);
101:      }
102:
103:      function setupHeader() {
104:        getAmPm();
105:        let apInterval = window.setInterval(getAmPm, 1000);
106:        window.addEventListener('beforeunload',
107:          function(e) {window.clearInterval(apInterval);})
108:      }
109:
110:      function setupFooter() {
111:        getDateTime();
112:        let dtInterval = window.setInterval(getDateTime, 1000);
113:        window.addEventListener('beforeunload',
114:          function(e) {window.clearInterval(dtInterval);})
115:      }
116:
117:      function setup() {
118:        setupHeader();
119:        setupFooter();
120:        let authorInput = document.getElementById('authorInput');
121:        authorInput.addEventListener('input', debouncedGetResults);
122:      }
123:
124:      document.addEventListener('DOMContentLoaded', setup);
125:
126:    </script>
127:  </body>
128: </html>

```

Fraction/testfraction.py (Page 1 of 2)

```

1: #-----
2: # testfraction.py
3: # Author: Bob Dondero
4: #-----
5:
6: import unittest
7: from fraction import Fraction
8:
9: class TestFraction(unittest.TestCase):
10:
11:     def test_str(self):
12:         self.assertEqual(str(Fraction(1, 2)), '1/2')
13:         self.assertEqual(str(Fraction(2, 1)), '2')
14:         self.assertEqual(str(Fraction(-2, 1)), '-2')
15:
16:     def test_constructor(self):
17:         self.assertEqual(str(Fraction(2)), '2')
18:         self.assertEqual(str(Fraction()), '0')
19:         self.assertEqual(str(Fraction(100, 200)), '1/2')
20:         self.assertEqual(str(Fraction(3684, 3084)), '307/257')
21:         self.assertEqual(str(Fraction(1, -2)), '-1/2')
22:         self.assertEqual(str(Fraction(-1, -2)), '1/2')
23:         self.assertEqual(str(Fraction(-2, 4)), '-1/2')
24:         self.assertEqual(str(Fraction(2, -4)), '-1/2')
25:         self.assertEqual(str(Fraction(-2, -4)), '1/2')
26:         self.assertEqual(str(Fraction(0, 2)), '0')
27:         with self.assertRaises(ZeroDivisionError):
28:             _ = Fraction(1, 0)
29:
30:     def test_eq(self):
31:         self.assertTrue(Fraction(1, 2) == Fraction(2, 4))
32:         self.assertFalse(Fraction(1, 2) == Fraction(2, 3))
33:
34:     def test_ne(self):
35:         self.assertTrue(Fraction(1, 2) != Fraction(2, 3))
36:         self.assertFalse(Fraction(1, 2) != Fraction(2, 4))
37:
38:     def test_lt(self):
39:         self.assertTrue(Fraction(1, 2) < Fraction(2, 3))
40:         self.assertFalse(Fraction(2, 3) < Fraction(1, 2))
41:         self.assertFalse(Fraction(1, 2) < Fraction(2, 4))
42:
43:     def test_gt(self):
44:         self.assertTrue(Fraction(2, 3) > Fraction(1, 2))
45:         self.assertFalse(Fraction(1, 2) > Fraction(2, 3))
46:         self.assertFalse(Fraction(1, 2) > Fraction(2, 4))
47:
48:     def test_le(self):
49:         self.assertTrue(Fraction(1, 2) <= Fraction(2, 3))
50:         self.assertFalse(Fraction(2, 3) <= Fraction(1, 2))
51:         self.assertTrue(Fraction(1, 2) <= Fraction(2, 4))
52:
53:     def test_ge(self):
54:         self.assertTrue(Fraction(2, 3) >= Fraction(1, 2))
55:         self.assertFalse(Fraction(1, 2) >= Fraction(2, 3))
56:         self.assertTrue(Fraction(1, 2) >= Fraction(2, 4))
57:
58:     def test_neg(self):
59:         self.assertEqual(-Fraction(1, 2), Fraction(-1, 2))
60:         self.assertEqual(-Fraction(-1, 2), Fraction(1, 2))
61:
62:     def test_add(self):
63:         self.assertEqual(Fraction(1, 2) + Fraction(2, 3),
64:             Fraction(7, 6))
65:         self.assertEqual(Fraction(-1, 2) + Fraction(2, 3),

```

Fraction/testfraction.py (Page 2 of 2)

```

66:         Fraction(1, 6))
67:         self.assertEqual(Fraction(1, 2) + Fraction(0, 1),
68:             Fraction(1, 2))
69:         self.assertEqual(Fraction(0, 1) + Fraction(2, 3),
70:             Fraction(2, 3))
71:
72:     def test_sub(self):
73:         self.assertEqual(Fraction(1, 2) - Fraction(2, 3),
74:             Fraction(-1, 6))
75:         self.assertEqual(Fraction(-1, 2) - Fraction(2, 3),
76:             Fraction(-7, 6))
77:         self.assertEqual(Fraction(1, 2) - Fraction(0, 1),
78:             Fraction(1, 2))
79:         self.assertEqual(Fraction(0, 1) - Fraction(2, 3),
80:             Fraction(-2, 3))
81:
82:     def test_mul(self):
83:         self.assertEqual(Fraction(1, 2) * Fraction(2, 3),
84:             Fraction(1, 3))
85:         self.assertEqual(Fraction(-1, 2) * Fraction(2, 3),
86:             Fraction(-1, 3))
87:         self.assertEqual(Fraction(1, 2) * Fraction(0, 1),
88:             Fraction(0, 1))
89:         self.assertEqual(Fraction(0, 1) * Fraction(2, 3),
90:             Fraction(0, 1))
91:
92:     def test_truediv(self):
93:         self.assertEqual(Fraction(1, 2) / Fraction(2, 3),
94:             Fraction(3, 4))
95:         self.assertEqual(Fraction(-1, 2) / Fraction(2, 3),
96:             Fraction(-3, 4))
97:         with self.assertRaises(ZeroDivisionError):
98:             _ = Fraction(1, 2) / Fraction(0, 1)
99:         self.assertEqual(Fraction(0, 1) / Fraction(2, 3),
100:             Fraction(0, 1))
101:
102: #-----
103:
104: if __name__ == '__main__':
105:     unittest.main(verbosity=2, module='testfraction')

```

PennyTest/testdatabase.py (Page 1 of 1)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # testdatabase.py
5: # Author: Bob Dondero
6: #-----
7:
8: import unittest
9: from database import get_books
10:
11: #-----
12:
13: class TestDatabase(unittest.TestCase):
14:
15:     def setUp(self):
16:         self._author = None
17:         self._expected = None
18:
19:     def tearDown(self):
20:         # Fetch from the DB a list of books for the given author.
21:         books = get_books(self._author)
22:
23:         # Compare that list of books with the expected list of books.
24:         self.assertEqual(len(books), len(self._expected))
25:         books.sort(key=lambda book: book['isbn'])
26:         for i in range(len(books)):
27:             self.assertEqual(books[i], self._expected[i])
28:
29:     def test_ker(self):
30:         self._author = 'ker'
31:         self._expected = [
32:             {'isbn': '123', 'author': 'Kernighan',
33:              'title': 'The Practice of Programming'},
34:             {'isbn': '234', 'author': 'Kernighan',
35:              'title': 'The C Programming Language'}
36:         ]
37:
38:     def test_sed(self):
39:         self._author = 'sed'
40:         self._expected = [
41:             {'isbn': '345', 'author': 'Sedgewick',
42:              'title': 'Algorithms in C'}
43:         ]
44:
45:     def test_abc(self):
46:         self._author = 'abc'
47:         self._expected = []
48:
49:     def test_ern(self):
50:         self._author = 'ern'
51:         self._expected = []
52:
53: if __name__ == '__main__':
54:     unittest.main(verbosity=2, module='testdatabase')

```

blank (Page 1 of 1)

```

1: This page is intentionally blank.

```

PennyTest/testpenny.py (Page 1 of 2)

```

1: #-----
2: # testpenny.py
3: # Author: Bob Dondero
4: #-----
5:
6: import os
7: import sys
8: import time
9: import unittest
10: import playwright.sync_api
11: import dotenv
12:
13: #-----
14:
15: dotenv.load_dotenv()
16: SERVER_URL = os.getenv('SERVER_URL', 'http://localhost:5000')
17: BROWSER = os.getenv('TEST_BROWSER', 'chrome')
18: DELAY = int(os.getenv('DELAY', '2'))
19:
20: #-----
21:
22: class PennyTestCase(unittest.TestCase):
23:
24:     @classmethod
25:     def setUpClass(cls):
26:         # Launch the browser.
27:         cls.pw = playwright.sync_api.sync_playwright().start()
28:         if BROWSER == 'chrome':
29:             cls.browser_process = cls.pw.chromium.launch()
30:         else:
31:             cls.browser_process = cls.pw.firefox.launch()
32:         sys.stdout.flush()
33:
34:     @classmethod
35:     def tearDownClass(cls):
36:         # Tell the browser to exit.
37:         cls.pw.stop()
38:
39:     def setUp(self):
40:         self._author = None
41:         self._expected = None
42:
43:         # Tell the browser to load a new page.
44:         self._page = self.browser_process.new_page()
45:
46:         # Tell the browser to load the app's index page.
47:         self._page.goto(SERVER_URL)
48:
49:     def tearDown(self):
50:         # Enter an author, thus triggering an AJAX call.
51:         author_input = self._page.locator('#authorInput')
52:         author_input.fill(self._author)
53:
54:         # Wait for the AJAX call to finish.
55:         time.sleep(DELAY)
56:
57:         # Fetch the contents of the resultsDiv element.
58:         results_div = self._page.locator('#resultsDiv')
59:         results_text = results_div.inner_text()
60:         books = results_text.split('\n')[:-1]
61:
62:         # Compare those contents with the expected contents.
63:         self.assertEqual(len(books), len(self._expected))
64:         books.sort()
65:         for i in range(len(books)):

```

PennyTest/testpenny.py (Page 2 of 2)

```

66:             self.assertEqual(books[i], self._expected[i])
67:
68:     def test_ker(self):
69:         self._author = 'ker'
70:         self._expected = [
71:             '123: Kernighan: The Practice of Programming',
72:             '234: Kernighan: The C Programming Language'
73:         ]
74:
75:     def test_sed(self):
76:         self._author = 'sed'
77:         self._expected = [
78:             '345: Sedgewick: Algorithms in C'
79:         ]
80:
81:     def test_abc(self):
82:         self._author = 'abc'
83:         self._expected = []
84:
85:     def test_spaces(self):
86:         self._author = ' '
87:         self._expected = []
88:
89:     def test_ker_spaces(self):
90:         self._author = ' kernigh '
91:         self._expected = [
92:             '123: Kernighan: The Practice of Programming',
93:             '234: Kernighan: The C Programming Language'
94:         ]
95:
96:     def test_ker_bad_spaces(self):
97:         self._author = ' ker nigh '
98:         self._expected = []
99:
100: #-----
101:
102: if __name__ == '__main__':
103:     unittest.main(verbosity=2, module='testpenny')

```