Fraction/euclid.py (Page 1 of 1)

```
#!/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.
```

Fraction/fraction.py (Page 1 of 2)

```
#!/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 __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)

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

Fraction/fraction.py (Page 2 of 2)

```
        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)
```

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)

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

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

import sys
import penny

def main():

    if len(sys.argv) != 2:
        print(f'usage: {sys.argv[0]} port', file=sys.stderr)
        sys.exit(1)

    try:
        port = int(sys.argv[1])
    except Exception:
        print(f'{sys.argv[0]}: Port must be an integer.',
              file=sys.stderr)
        sys.exit(1)

    try:
        penny.app.run(host='0.0.0.0', port=port, debug=True)
    except Exception as ex:
        print(f'{sys.argv[0]}: {ex}', file=sys.stderr)
        sys.exit(1)

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

PennyTest/penny.sql (Page 1 of 1)

```
DROP TABLE IF EXISTS books;

CREATE TABLE books (isbn TEXT PRIMARY KEY, author TEXT, title TEXT);

INSERT INTO books (isbn, author, title)
   VALUES ('123', 'Kernighan','The Practice of Programming');
INSERT INTO books (isbn, author, title)
   VALUES ('234', 'Kernighan','The C Programming Language');
INSERT INTO books (isbn, author, title)
   VALUES ('345', 'Sedgewick','Algorithms in C');
```

PennyTest/database.py (Page 1 of 1)

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

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

import sqlite3
import contextlib

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

_DATABASE_URL = 'file:penny.sqlite'

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

def get_books(author):

    books = []

    with contextlib.closing(
        sqlite3.connect(_DATABASE_URL + '?mode=ro',
        isolation_level=None, uri=True)) as connection:

        with contextlib.closing(connection.cursor()) as cursor:

            query_str = '''
                SELECT isbn, author, title FROM books
                WHERE author LIKE ?
                '''
            cursor.execute(query_str, [author+'%'])

            table = cursor.fetchall()
            for row in table:
                book = {'isbn': row[0], 'author': row[1],
                    'title': row[2]}
                books.append(book)

    return books

```


PennyTest/penny.py (Page 1 of 1)

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

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

import json
import flask
import database

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

app = flask.Flask(__name__)

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

@app.route('/', methods=['GET'])
@app.route('/index', methods=['GET'])
def index():

    return flask.send_file('index.html')

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

@app.route('/searchresults', methods=['GET'])
def search_results():

    author = flask.request.args.get('author')
    if author is None:
        author = ''
    author = author.strip()

    if author == '':
        books = []
    else:
        books = database.get_books(author) # Exception handling omitted

    json_doc = json.dumps(books)
    response = flask.make_response(json_doc)
    response.headers['Content-Type'] = 'application/json'
    return response
```

PennyTest/index.html (Page 1 of 2)

```
<!DOCTYPE html>
<html>
   <head>
      <title>Penny.com</title>
   </head>

   <body>

      <hr>
      Good <span id="ampmSpan"></span> and welcome to
      <strong>Penny.com</strong>
      <hr>

      <h1>Author Search</h1>
      Please enter an author name:
      <input type="text" id="authorInput" autoFocus>
      <hr>
      <div id="resultsDiv"></div>

      <hr>
      Date and time: <span id="datetimeSpan"></span><br>
      Created by <a href="https://www.cs.princeton.edu/~rdondero">
      Bob Dondero</a>
      <hr>

      <script src=
         "https://cdn.jsdelivr.net/npm/mustache@4.2.0/mustache.min.js">
      </script>

      <!-- <script src="/static/mustache.min.js"></script> -->

      <script>

         'use strict';

         function getAmPm() {
            let dateTime = new Date();
            let hours = dateTime.getHours();
            let amPm = (hours < 12) ? 'morning' : 'afternoon';
            let ampmSpan = document.getElementById('ampmSpan');
            ampmSpan.innerHTML = amPm;
         }

         function getDateTime() {
            let dateTime = new Date();
            let datetimeSpan =
               document.getElementById('datetimeSpan');
            datetimeSpan.innerHTML = dateTime.toLocaleString();
         }

         function convertToHtml(books) {
            let template = `
               {{#books}}
                  {{isbn}}:
                  <strong>{{author}}</strong>:
                  {{title}}
                  <br>
               {{/books}}
            `;
            let map = {books: books};
            let html = Mustache.render(template, map);
            return html;
         }

         function handleResponse() {
```

PennyTest/index.html (Page 2 of 2)

```
            if (this.status !== 200) {
               alert('Error: Failed to fetch data from server');
               return;
            }
            let books = JSON.parse(this.response);
            let html = convertToHtml(books);
            let resultsDiv = document.getElementById('resultsDiv');
            resultsDiv.innerHTML = html;
         }

         function handleError() {
            alert('Error: Failed to fetch data from server');
         }

         let request = null;

         function getResults() {
            let authorInput = document.getElementById('authorInput');
            let author = authorInput.value;
            let encodedAuthor = encodeURIComponent(author);
            let url = '/searchresults?author=' + encodedAuthor;
            if (request !== null)
               request.abort();
            request = new XMLHttpRequest();
            request.onload = handleResponse;
            request.onerror = handleError;
            request.open('GET', url);
            request.send();
         }

         let timer = null;

         function debouncedGetResults() {
            clearTimeout(timer);
            timer = window.setTimeout(getResults, 500);
         }

         function setupHeader() {
            getAmPm();
            let apInterval = window.setInterval(getAmPm, 1000);
            window.addEventListener('beforeunload',
               function(e) {window.clearInterval(apInterval);})
         }

         function setupFooter() {
            getDateTime();
            let dtInterval = window.setInterval(getDateTime, 1000);
            window.addEventListener('beforeunload',
               function(e) {window.clearInterval(dtInterval);})
         }

         function setup() {
            setupHeader();
            setupFooter();
            let authorInput = document.getElementById('authorInput');
            authorInput.addEventListener('input', debouncedGetResults);
         }

         document.addEventListener('DOMContentLoaded', setup);

      </script>
   </body>
</html>
```

Fraction/testfraction.py (Page 1 of 2)

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

import unittest
from fraction import Fraction

class TestFraction(unittest.TestCase):

    def test_str(self):
        self.assertEqual(str(Fraction(1, 2)), '1/2')
        self.assertEqual(str(Fraction(2, 1)), '2')
        self.assertEqual(str(Fraction(-2, 1)), '-2')

    def test_constructor(self):
        self.assertEqual(str(Fraction(2)), '2')
        self.assertEqual(str(Fraction()), '0')
        self.assertEqual(str(Fraction(100, 200)), '1/2')
        self.assertEqual(str(Fraction(3684, 3084)), '307/257')
        self.assertEqual(str(Fraction(1, -2)), '-1/2')
        self.assertEqual(str(Fraction(-1, -2)), '1/2')
        self.assertEqual(str(Fraction(-2, 4)), '-1/2')
        self.assertEqual(str(Fraction(2, -4)), '-1/2')
        self.assertEqual(str(Fraction(-2, -4)), '1/2')
        self.assertEqual(str(Fraction(0, 2)), '0')
        with self.assertRaises(ZeroDivisionError):
            _ = Fraction(1, 0)

    def test_eq(self):
        self.assertTrue(Fraction(1, 2) == Fraction(2, 4))
        self.assertFalse(Fraction(1, 2) == Fraction(2, 3))

    def test_ne(self):
        self.assertTrue(Fraction(1, 2) != Fraction(2, 3))
        self.assertFalse(Fraction(1, 2) != Fraction(2, 4))

    def test_lt(self):
        self.assertTrue(Fraction(1, 2) < Fraction(2, 3))
        self.assertFalse(Fraction(2, 3) < Fraction(1, 2))
        self.assertFalse(Fraction(1, 2) < Fraction(2, 4))

    def test_gt(self):
        self.assertTrue(Fraction(2, 3) > Fraction(1, 2))
        self.assertFalse(Fraction(1, 2) > Fraction(2, 3))
        self.assertFalse(Fraction(1, 2) > Fraction(2, 4))

    def test_le(self):
        self.assertTrue(Fraction(1, 2) <= Fraction(2, 3))
        self.assertFalse(Fraction(2, 3) <= Fraction(1, 2))
        self.assertTrue(Fraction(1, 2) <= Fraction(2, 4))

    def test_ge(self):
        self.assertTrue(Fraction(2, 3) >= Fraction(1, 2))
        self.assertFalse(Fraction(1, 2) >= Fraction(2, 3))
        self.assertTrue(Fraction(1, 2) >= Fraction(2, 4))

    def test_neg(self):
        self.assertEqual(-Fraction(1, 2), Fraction(-1, 2))
        self.assertEqual(-Fraction(-1, 2), Fraction(1, 2))

    def test_add(self):
        self.assertEqual(Fraction(1, 2) + Fraction(2, 3),
            Fraction(7, 6))
        self.assertEqual(Fraction(-1, 2) + Fraction(2, 3),
            Fraction(1, 6))
        self.assertEqual(Fraction(1, 2) + Fraction(0, 1),
            Fraction(1, 2))
        self.assertEqual(Fraction(0, 1) + Fraction(2, 3),
            Fraction(2, 3))

    def test_sub(self):
        self.assertEqual(Fraction(1, 2) - Fraction(2, 3),
            Fraction(-1, 6))
        self.assertEqual(Fraction(-1, 2) - Fraction(2, 3),
            Fraction(-7, 6))
        self.assertEqual(Fraction(1, 2) - Fraction(0, 1),
            Fraction(1, 2))
        self.assertEqual(Fraction(0, 1) - Fraction(2, 3),
            Fraction(-2, 3))

    def test_mul(self):
        self.assertEqual(Fraction(1, 2) * Fraction(2, 3),
            Fraction(1, 3))
        self.assertEqual(Fraction(-1, 2) * Fraction(2, 3),
            Fraction(-1, 3))
        self.assertEqual(Fraction(1, 2) * Fraction(0, 1),
            Fraction(0, 1))
        self.assertEqual(Fraction(0, 1) * Fraction(2, 3),
            Fraction(0, 1))

    def test_truediv(self):
        self.assertEqual(Fraction(1, 2) / Fraction(2, 3),
            Fraction(3, 4))
        self.assertEqual(Fraction(-1, 2) / Fraction(2, 3),
            Fraction(-3, 4))
        with self.assertRaises(ZeroDivisionError):
            _ = Fraction(1, 2) / Fraction(0, 1)
        self.assertEqual(Fraction(0, 1) / Fraction(2, 3),
            Fraction(0, 1))

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

if __name__ == '__main__':
    unittest.main(verbosity=2, module='testfraction')
```

PennyTest/testdatabase.py (Page 1 of 1)

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

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

import unittest
from database import get_books

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

class TestDatabase(unittest.TestCase):

    def setUp(self):
        self._author = None
        self._expected = None

    def tearDown(self):
        # Fetch from the DB a list of books for the given author.
        books = get_books(self._author)

        # Compare that list of books with the expected list of books.
        self.assertEqual(len(books), len(self._expected))
        books.sort(key=lambda book: book['isbn'])
        for i in range(len(books)):
            self.assertEqual(books[i], self._expected[i])

    def test_ker(self):
        self._author = 'ker'
        self._expected = [
            {'isbn': '123', 'author': 'Kernighan',
                'title': 'The Practice of Programming'},
            {'isbn': '234', 'author': 'Kernighan',
                'title': 'The C Programming Language'}
        ]

    def test_sed(self):
        self._author = 'sed'
        self._expected = [
            {'isbn': '345', 'author': 'Sedgewick',
                'title': 'Algorithms in C'}
        ]

    def test_abc(self):
        self._author = 'abc'
        self._expected = []

    def test_ern(self):
        self._author = 'ern'
        self._expected = []

if __name__ == '__main__':
    unittest.main(verbosity=2, module='testdatabase')
```

blank (Page 1 of 1)

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

PennyTest/testpenny.py (Page 1 of 2)

```
#-----------------------------------------------------------------------
# testpenny.py
# Author: Bob Dondero
#-----------------------------------------------------------------------

import os
import sys
import time
import unittest
import playwright.sync_api
import dotenv

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

dotenv.load_dotenv()
SERVER_URL = os.getenv('SERVER_URL', 'http://localhost:5000')
BROWSER = os.getenv('TEST_BROWSER', 'chrome')
DELAY = int(os.getenv('DELAY', '2'))

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

class PennyTestCase(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        # Launch the browser.
        cls.pw = playwright.sync_api.sync_playwright().start()
        if BROWSER == 'chrome':
            cls.browser_process = cls.pw.chromium.launch()
        else:
            cls.browser_process = cls.pw.firefox.launch()
        sys.stdout.flush()

    @classmethod
    def tearDownClass(cls):
        # Tell the browser to exit.
        cls.pw.stop()

    def setUp(self):
        self._author = None
        self._expected = None

        # Tell the browser to load a new page.
        self._page = self.browser_process.new_page()

        # Tell the browser to load the app's index page.
        self._page.goto(SERVER_URL)

    def tearDown(self):
        # Enter an author, thus triggering an AJAX call.
        author_input = self._page.locator('#authorInput')
        author_input.fill(self._author)

        # Wait for the AJAX call to finish.
        time.sleep(DELAY)

        # Fetch the contents of the resultsDiv element.
        results_div = self._page.locator('#resultsDiv')
        results_text = results_div.inner_text()
        books = results_text.split('\n')[:-1]

        # Compare those contents with the expected contents.
        self.assertEqual(len(books), len(self._expected))
        books.sort()
        for i in range(len(books)):
```

PennyTest/testpenny.py (Page 2 of 2)

```
            self.assertEqual(books[i], self._expected[i])

    def test_ker(self):
        self._author = 'ker'
        self._expected = [
            '123: Kernighan: The Practice of Programming',
            '234: Kernighan: The C Programming Language'
        ]

    def test_sed(self):
        self._author = 'sed'
        self._expected = [
            '345: Sedgewick: Algorithms in C'
        ]

    def test_abc(self):
        self._author = 'abc'
        self._expected = []

    def test_spaces(self):
        self._author = '   '
        self._expected = []

    def test_ker_spaces(self):
        self._author = ' kernigh '
        self._expected = [
            '123: Kernighan: The Practice of Programming',
            '234: Kernighan: The C Programming Language'
        ]

    def test_ker_bad_spaces(self):
        self._author = ' ker nigh '
        self._expected = []

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

if __name__ == '__main__':
    unittest.main(verbosity=2, module='testpenny')
```