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)

```
#!/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__()

        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(str(ex), file=sys.stderr)

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

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

blank (Page 1 of 1)

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

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/testfraction.py (Page 2 of 2)

```
            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/runserver.py (Page 1 of 1)

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

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

import sys
import penny

PORT = 5000

def main():

    if len(sys.argv) != 1:
        print('Usage: ' + sys.argv[0], file=sys.stderr)
        sys.exit(1)

    try:
        penny.app.run(host='0.0.0.0', port=PORT, debug=True)
    except Exception as ex:
        print(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?mode=ro'

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

def get_books(author):

    books = []

    with sqlite3.connect(_DATABASE_URL, isolation_level=None,
        uri=True) as connection:

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

            query_str = "SELECT isbn, author, title FROM books "
            query_str += "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>
      Hello 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>
      Created by <a href="https://www.cs.princeton.edu/~rdondero">
      Bob Dondero</a>
      <hr>

      <script>

         'use strict';

         function escape(s) {
            s = s.replace('&', '&amp;');
            s = s.replace('<', '&lt;');
            s = s.replace('>', '&gt;');
            s = s.replace('"', '&quot;');
            s = s.replace("'", '&apos;');
            return s;
         }

         function convertToHtml(books) {
            let html = '';
            for (let book of books) {
               html += escape(book['isbn']) + ': ';
               html += '<strong>'
               html += escape(book['author']);
               html += '</strong>: ';
               html += escape(book['title']);
               html += '<br>';
            }
            return html;
         }

         function handleResponse() {
            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;
```

PennyTest/index.html (Page 2 of 2)

```

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

         document.addEventListener('DOMContentLoaded', setup);

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

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')
```

PennyTestCas/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:       Hello {{username}} 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:       <a href="/logoutapp">Log out of the app</a><br>
22:       <a href="/logoutcas">Log out of the app and CAS</a><br>
23:       Created by <a href="https://www.cs.princeton.edu/~rdondero">
24:       Bob Dondero</a>
25:       <hr>
26:
27:       <script>
28:
29:          'use strict';
30:
31:          function escape(s) {
32:             s = s.replace('&', '&amp;');
33:             s = s.replace('<', '&lt;');
34:             s = s.replace('>', '&gt;');
35:             s = s.replace('"', '&quot;');
36:             s = s.replace("'", '&apos;');
37:             return s;
38:          }
39:
40:          function convertToHtml(books) {
41:             let html = '';
42:             for (let book of books) {
43:                html += escape(book['isbn']) + ': ';
44:                html += '<strong>'
45:                html += escape(book['author']);
46:                html += '</strong>: ';
47:                html += escape(book['title']);
48:                html += '<br>';
49:             }
50:             return html;
51:          }
52:
53:          function handleResponse() {
54:             if (this.status !== 200) {
55:                alert('Error: Failed to fetch data from server');
56:                return;
57:             }
58:             let books = JSON.parse(this.response);
59:             let html = convertToHtml(books);
60:             let resultsDiv = document.getElementById('resultsDiv');
61:             resultsDiv.innerHTML = html;
62:          }
63:
64:          function handleError() {
65:             alert('Error: Failed to fetch data from server');
```

PennyTestCas/index.html (Page 2 of 2)

```
66:          }
67:
68:          let request = null;
69:
70:          function getResults() {
71:             let authorInput = document.getElementById('authorInput');
72:             let author = authorInput.value;
73:             let encodedAuthor = encodeURIComponent(author);
74:             let url = '/searchresults?author=' + encodedAuthor;
75:             if (request !== null)
76:                request.abort();
77:             request = new XMLHttpRequest();
78:             request.onload = handleResponse;
79:             request.onerror = handleError;
80:             request.open('GET', url);
81:             request.send();
82:          }
83:
84:          let timer = null;
85:
86:          function debouncedGetResults() {
87:             clearTimeout(timer);
88:             timer = window.setTimeout(getResults, 500);
89:          }
90:
91:          function setup() {
92:             let authorInput = document.getElementById('authorInput');
93:             authorInput.addEventListener('input', debouncedGetResults);
94:          }
95:
96:          document.addEventListener('DOMContentLoaded', setup);
97:
98:       </script>
99:    </body>
100: </html>
```

PennyTestCas/loggedout.html (Page 1 of 1)

```
<!DOCTYPE html>
<html>
    <head>
        <title>Penny.com</title>
    </head>
    <body>
        <h2>You are logged out of Penny.com</h2>
        <a href="/index">Revisit the application</a>
    </body>
</html>
```


PennyTestCas/top.py (Page 1 of 1)

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

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

import flask

app = flask.Flask(__name__, template_folder='.')
```

PennyTestCas/penny.py (Page 1 of 2)

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

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

import os
import json
import flask
import flask_session
import flask_sqlalchemy
import dotenv
import database

from top import app

dotenv.load_dotenv()

TESTING = os.getenv('TESTING', 'no')
if TESTING == 'yes':
    import authstub as auth
else:
    import auth

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

# Session configuration to store session data in a database:

session_database_url = os.getenv('SESSION_DATABASE_URL',
    'sqlite:///pennysessions.sqlite')
session_database_url = session_database_url.replace(
    'postgres://', 'postgresql://')
app.config['SESSION_PERMANENT'] = False
app.config['SESSION_TYPE'] = 'sqlalchemy'
app.config['SQLALCHEMY_DATABASE_URI'] = session_database_url
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SESSION_SQLALCHEMY'] = flask_sqlalchemy.SQLAlchemy(app)
flask_session.Session(app)

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

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

    auth.authenticate()
    username = auth.get_username()

    html = flask.render_template('index.html', username=username)
    response = flask.make_response(html)
    return response

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

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

    if not auth.is_authenticated():
        flask.abort(403)

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

PennyTestCas/penny.py (Page 2 of 2)

```

    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
```

PennyTestCas/auth.py (Page 1 of 3)

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

#-----------------------------------------------------------------------
# auth.py
# Authors: Alex Halderman, Scott Karlin, Brian Kernighan, Bob Dondero,
#          and Joshua Lau '26
#-----------------------------------------------------------------------

import urllib.request
import urllib.parse
import re
import json
import flask

from top import app

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

_CAS_URL = 'https://fed.princeton.edu/cas/'

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

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

    # Log out of the application.
    flask.session.clear()
    return flask.send_file('loggedout.html')

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

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

    # Log out of the CAS session, and then the application.
    logout_url = (_CAS_URL + 'logout?service='
        + urllib.parse.quote(
            re.sub('logoutcas', 'logoutapp', flask.request.url)))
    flask.abort(flask.redirect(logout_url))

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

# Return url after stripping out the "ticket" parameter that was
# added by the CAS server.

def strip_ticket(url):
    if url is None:
        return "something is badly wrong"
    url = re.sub(r'ticket=[^&]*&?', '', url)
    url = re.sub(r'\?&?$|&$', '', url)
    return url

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

# Validate a login ticket by contacting the CAS server. If
# valid, return the user's user_info; otherwise, return None.

def validate(ticket):
    val_url = (_CAS_URL + "validate"
        + '?service='
        + urllib.parse.quote(strip_ticket(flask.request.url))
        + '&ticket='
        + urllib.parse.quote(ticket)
        + '&format=json')
    with urllib.request.urlopen(val_url) as flo:
```

PennyTestCas/auth.py (Page 2 of 3)

```
        result = json.loads(flo.read().decode('utf-8'))

    if (not result) or ('serviceResponse' not in result):
        return None

    service_response = result['serviceResponse']

    if 'authenticationSuccess' in service_response:
        user_info = service_response['authenticationSuccess']
        return user_info

    if 'authenticationFailure' in service_response:
        print('CAS authentication failure:', service_response)
        return None

    print('Unexpected CAS response:', service_response)
    return None

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

def is_authenticated():

    return 'user_info' in flask.session

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

def get_user_info():

    return flask.session.get('user_info')

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

def get_username():

    user_info = flask.session.get('user_info')
    if user_info is None:
        return ''
    return user_info.get('user', '')

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

# Authenticate the user. Do not return unless the user is
# successfully authenticated.

def authenticate():

    # If the user_info is in the session, then the user was
    # authenticated previously.  So return.
    if flask.session.get('user_info') is not None:
        return

    # If the request does not contain a login ticket, then redirect
    # the browser to the login page to get one.
    ticket = flask.request.args.get('ticket')
    if ticket is None:
        login_url = (_CAS_URL + 'login?service=' +
            urllib.parse.quote(flask.request.url))
        flask.abort(flask.redirect(login_url))

    # If the login ticket is invalid, then redirect the browser
    # to the login page to get a new one.
    user_info = validate(ticket)
    if user_info is None:
        login_url = (_CAS_URL + 'login?service='
            + urllib.parse.quote(strip_ticket(flask.request.url)))
```

PennyTestCas/auth.py (Page 3 of 3)

```
131:         flask.abort(flask.redirect(login_url))
132:
133:     # The user is authenticated, so store the user_info in
134:     # the session and return.
135:     flask.session['user_info'] = user_info
```

blank (Page 1 of 1)

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

PennyTestCas/authstub.py (Page 1 of 2)

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

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

import flask

from top import app

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

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

    # Log out of the application.
    flask.session.clear()
    return flask.send_file('loggedout.html')

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

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

    flask.abort(flask.redirect('/logoutapp'))

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

def is_authenticated():

    return 'username' in flask.session

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

def get_user_info():

    return None

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

def get_username():

    return flask.session.get('username', '')

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

@app.route('/handlelogin', methods=['GET'])
def handlelogin():
    username = flask.request.args.get('username')
    flask.session['username'] = username
    flask.abort(flask.redirect('/index'))

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

@app.route('/login', methods=['GET'])
def login():
    html_code = '''
        <form action="handlelogin" method="get">
        Username:
        <input type="text" name="username" id="username">
        <input type="submit" id="submit">'''
    response = flask.make_response(html_code)
    return response

```

PennyTestCas/authstub.py (Page 2 of 2)

```
#-----------------------------------------------------------------------

# Authenticate the user. Do not return unless the user is
# successfully authenticated.

def authenticate():

    # If the user_info is in the session, then the user was
    # authenticated previously.  So return.
    if flask.session.get('username') is not None:
        return

    flask.abort(flask.redirect('/login'))
```

PennyTestCas/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)

        # Move past the phony login page.
        username_input = self._page.locator('#username')
        username_input.fill('rdondero')
        submit_button = self._page.locator('#submit')
        submit_button.click()

    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()
```

PennyTestCas/testpenny.py (Page 2 of 2)

```
        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)):
            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')
```

PennyTestGoogle/index.html (Page 1 of 2)

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

   <body>

      <hr>
      Hello {{username}} 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>
      <a href="/logoutapp">Log out of the app</a><br>
      <a href="/logoutgoogle">Log out of the app and Google</a><br>
      Created by <a href="https://www.cs.princeton.edu/~rdondero">
      Bob Dondero</a>
      <hr>

      <script>

         'use strict';

         function escape(s) {
            s = s.replace('&', '&amp;');
            s = s.replace('<', '&lt;');
            s = s.replace('>', '&gt;');
            s = s.replace('"', '&quot;');
            s = s.replace("'", '&apos;');
            return s;
         }

         function convertToHtml(books) {
            let html = '';
            for (let book of books) {
               html += escape(book['isbn']) + ': ';
               html += '<strong>'
               html += escape(book['author']);
               html += '</strong>: ';
               html += escape(book['title']);
               html += '<br>';
            }
            return html;
         }

         function handleResponse() {
            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');
         }
```

PennyTestGoogle/index.html (Page 2 of 2)

```
         }

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

         document.addEventListener('DOMContentLoaded', setup);

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

PennyTestGoogle/loggedout.html (Page 1 of 1)

```
<!DOCTYPE html>
<html>
    <head>
        <title>Penny.com</title>
    </head>
    <body>
        <h2>You are logged out of Penny.com</h2>
        <a href="/index">Revisit the application</a>
    </body>
</html>
```


PennyTestGoogle/top.py (Page 1 of 1)

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

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

import flask

app = flask.Flask(__name__, template_folder='.')
```

PennyTestGoogle/penny.py (Page 1 of 2)

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

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

import os
import json
import flask
import flask_session
import flask_sqlalchemy
import dotenv
import database

from top import app

dotenv.load_dotenv()

TESTING = os.getenv('TESTING', 'no')
if TESTING == 'yes':
    import authstub as auth
else:
    import auth

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

# Session configuration to store session data in a database:

session_database_url = os.getenv('SESSION_DATABASE_URL',
    'sqlite:///pennysessions.sqlite')
session_database_url = session_database_url.replace(
    'postgres://', 'postgresql://')
app.config['SESSION_PERMANENT'] = False
app.config['SESSION_TYPE'] = 'sqlalchemy'
app.config['SQLALCHEMY_DATABASE_URI'] = session_database_url
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SESSION_SQLALCHEMY'] = flask_sqlalchemy.SQLAlchemy(app)
flask_session.Session(app)

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

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

    auth.authenticate()
    username = auth.get_username()

    html = flask.render_template('index.html', username=username)
    response = flask.make_response(html)
    return response

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

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

    if not auth.is_authenticated():
        flask.abort(403)

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

PennyTestGoogle/penny.py (Page 2 of 2)

```

    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
```

PennyTestGoogle/auth.py (Page 1 of 4)

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

#-----------------------------------------------------------------------
# auth.py
# Author: Bob Dondero
#   With lots of help from https://realpython.com/flask-google-login/
#-----------------------------------------------------------------------

import os
import json
import requests
import flask
import oauthlib.oauth2
import dotenv

from top import app

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

GOOGLE_DISCOVERY_URL = (
    'https://accounts.google.com/.well-known/openid-configuration')

dotenv.load_dotenv()
GOOGLE_CLIENT_ID = os.environ['GOOGLE_CLIENT_ID']
GOOGLE_CLIENT_SECRET = os.environ['GOOGLE_CLIENT_SECRET']

client = oauthlib.oauth2.WebApplicationClient(GOOGLE_CLIENT_ID)

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

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

    original_url = flask.request.args.get('originalurl')

    # Determine the URL for Google login.
    google_provider_cfg = requests.get(
        GOOGLE_DISCOVERY_URL, timeout=2).json()
    authorization_endpoint = (
        google_provider_cfg['authorization_endpoint'])

    # Construct the request URL for Google login, providing scopes
    # to fetch the user's profile data.
    request_uri = client.prepare_request_uri(
        authorization_endpoint,
        redirect_uri = flask.request.base_url + '/callback',
        scope=['openid', 'email', 'profile'],
        state=original_url
    )

    #-------------------------------------------------------------------
    # For learning:
    # print('request_uri:', request_uri, file=sys.stderr)
    #-------------------------------------------------------------------

    # Redirect to the request URL.
    return flask.redirect(request_uri)

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

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

    original_url = flask.request.args.get('state')

```

PennyTestGoogle/auth.py (Page 2 of 4)

```
    # Get the authorization code that Google sent.
    code = flask.request.args.get('code')

    #-------------------------------------------------------------------
    # For learning:
    # print('code:', code, file=sys.stderr)
    #-------------------------------------------------------------------

    # Determine the URL to fetch tokens that allow the application to
    # ask for the user's profile data.
    google_provider_cfg = requests.get(
        GOOGLE_DISCOVERY_URL, timeout=2).json()
    token_endpoint = google_provider_cfg['token_endpoint']

    # Construct a request to fetch the tokens.
    token_url, headers, body = client.prepare_token_request(
        token_endpoint,
        authorization_response=flask.request.url,
        redirect_url=flask.request.base_url,
        code=code
    )

    #-------------------------------------------------------------------
    # For learning:
    # print('token_url:', token_url, file=sys.stderr)
    # print('headers:', headers, file=sys.stderr)
    # print('body:', body, file=sys.stderr)
    #-------------------------------------------------------------------

    # Fetch the tokens.
    token_response = requests.post(
        token_url,
        headers=headers,
        data=body,
        auth=(GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET),
        timeout=2
    )

    #-------------------------------------------------------------------
    # For learning:
    # print('token_response.json():', token_response.json(),
    #     file=sys.stderr)
    #-------------------------------------------------------------------

    # Parse the tokens.
    client.parse_request_body_response(
        json.dumps(token_response.json()))

    # Using the tokens, fetch the user's profile data,
    # including the user's Google profile image and email address.
    userinfo_endpoint = google_provider_cfg['userinfo_endpoint']
    uri, headers, body = client.add_token(userinfo_endpoint)

    #-------------------------------------------------------------------
    # For learning:
    # print('uri:', uri, file=sys.stderr)
    # print('headers:', headers, file=sys.stderr)
    # print('body:', body, file=sys.stderr)
    #-------------------------------------------------------------------

    userinfo_response = requests.get(uri, headers=headers, data=body,
        timeout=2)

    #-------------------------------------------------------------------
    # For learning:
```

PennyTestGoogle/auth.py (Page 3 of 4)

```
131:     # print('userinfo_response.json():', userinfo_response.json(),
132:     #     file=sys.stderr)
133:     #-------------------------------------------------------------------
134:
135:     # Optional: Make sure the user's email address is verified.
136:     if not userinfo_response.json().get('email_verified'):
137:         message = 'User email not available or not verified by Google.'
138:         return message, 400
139:
140:     userinfo = userinfo_response.json()
141:     # Save the user profile data in the session.
142:     flask.session['userinfo'] = userinfo
143:
144:     return flask.redirect(original_url)
145:
146: #-------------------------------------------------------------------
147:
148: @app.route('/logoutapp', methods=['GET'])
149: def logoutapp():
150:
151:     # Log out of the application.
152:     flask.session.clear()
153:
154:     return flask.send_file('loggedout.html')
155:
156: #-------------------------------------------------------------------
157:
158: @app.route('/logoutgoogle', methods=['GET'])
159: def logoutgoogle():
160:
161:     # Log out of the application.
162:     flask.session.clear()
163:
164:     # Log out of Google.
165:     google_logout_page = 'https://www.google.com/accounts/Logout'
166:     #google_logout_page='https://mail.google.com/mail/u/0/?logout&hl=en'
167:     return flask.redirect(google_logout_page)
168:
169:     # How to redirect to /logoutapp?
170:
171: #-------------------------------------------------------------------
172:
173: def is_authenticated():
174:
175:     return 'userinfo' in flask.session
176:
177: #-------------------------------------------------------------------
178:
179: def get_userinfo():
180:
181:     return flask.session.get('userinfo')
182:
183: #-------------------------------------------------------------------
184:
185: def get_username():
186:
187:     userinfo = flask.session.get('userinfo')
188:     if userinfo is None:
189:         return ''
190:     return userinfo.get('email', '')
191:
192: #-------------------------------------------------------------------
193:
194: # Authenticate the user. Do not return unless the user is
195: # successfully authenticated.
```

PennyTestGoogle/auth.py (Page 4 of 4)

```
196:
197: def authenticate():
198:
199:     if 'userinfo' not in flask.session:
200:         flask.abort(flask.redirect(
201:             '/login?originalurl=' + flask.request.url))
```

PennyTestGoogle/authstub.py (Page 1 of 2)

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

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

import flask

from top import app

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

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

    # Log out of the application.
    flask.session.clear()
    return flask.send_file('loggedout.html')

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

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

    flask.abort(flask.redirect('/logoutapp'))

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

def is_authenticated():

    return 'username' in flask.session

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

def get_user_info():

    return None

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

def get_username():

    return flask.session.get('username', '')

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

@app.route('/handlelogin', methods=['GET'])
def handlelogin():
    username = flask.request.args.get('username')
    flask.session['username'] = username
    flask.abort(flask.redirect('/index'))

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

@app.route('/login', methods=['GET'])
def login():
    html_code = '''
        <form action="handlelogin" method="get">
        Username:
        <input type="text" name="username" id="username">
        <input type="submit" id="submit">'''
    response = flask.make_response(html_code)
    return response

```

PennyTestGoogle/authstub.py (Page 2 of 2)

```
#-----------------------------------------------------------------------

# Authenticate the user. Do not return unless the user is
# successfully authenticated.

def authenticate():

    # If the user_info is in the session, then the user was
    # authenticated previously.  So return.
    if flask.session.get('username') is not None:
        return

    flask.abort(flask.redirect('/login'))
```

PennyTestGoogle/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)

        # Move past the phony login page.
        username_input = self._page.locator('#username')
        username_input.fill('rdondero')
        submit_button = self._page.locator('#submit')
        submit_button.click()

    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()
```

PennyTestGoogle/testpenny.py (Page 2 of 2)

```
        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)):
            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')
```

PennyTestEntra/index.html (Page 1 of 2)

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

   <body>

      <hr>
      Hello {{username}} 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>
      <a href="/logoutapp">Log out of the app</a><br>
      <a href="/logoutentra">Log out of the app and Entra ID</a><br>
      Created by <a href="https://www.cs.princeton.edu/~rdondero">
      Bob Dondero</a>
      <hr>

      <script>

         'use strict';

         function escape(s) {
            s = s.replace('&', '&amp;');
            s = s.replace('<', '&lt;');
            s = s.replace('>', '&gt;');
            s = s.replace('"', '&quot;');
            s = s.replace("'", '&apos;');
            return s;
         }

         function convertToHtml(books) {
            let html = '';
            for (let book of books) {
               html += escape(book['isbn']) + ': ';
               html += '<strong>'
               html += escape(book['author']);
               html += '</strong>: ';
               html += escape(book['title']);
               html += '<br>';
            }
            return html;
         }

         function handleResponse() {
            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');
         }
```

PennyTestEntra/index.html (Page 2 of 2)

```
         }

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

         document.addEventListener('DOMContentLoaded', setup);

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

PennyTestEntra/loggedout.html (Page 1 of 1)

```
<!DOCTYPE html>
<html>
   <head>
      <title>Penny.com</title>
   </head>
   <body>
      <h2>You are logged out of Penny.com</h2>
      <a href="/index">Revisit the application</a>
   </body>
</html>
```

PennyTestEntra/top.py (Page 1 of 1)

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

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

import flask

app = flask.Flask(__name__, template_folder='.')
```

PennyTestEntra/penny.py (Page 1 of 2)

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

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

import os
import json
import flask
import flask_sqlalchemy
import dotenv
import database
from top import app

dotenv.load_dotenv()

TESTING = os.getenv('TESTING', 'no')
if TESTING == 'yes':
    import authstub as identity_flask
else:
    import identity.flask as identity_flask

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

# Session configuration to store session data in a database:

session_database_url = os.getenv('SESSION_DATABASE_URL',
    'sqlite:///pennysessions.sqlite')
session_database_url = session_database_url.replace(
    'postgres://', 'postgresql://')
app.config['SESSION_PERMANENT'] = False
app.config['SESSION_TYPE'] = 'sqlalchemy'
app.config['SQLALCHEMY_DATABASE_URI'] = session_database_url
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SESSION_SQLALCHEMY'] = flask_sqlalchemy.SQLAlchemy(app)

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

# Microsoft Entra ID configuration

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

    flask.session.clear()
    return flask.send_file('loggedout.html')

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

    response = flask.redirect(flask.url_for('identity.logout'))
    return response

app.config['SCOPE'] = os.environ['SCOPE']
app.config['ENDPOINT'] = os.environ['ENDPOINT']
auth = identity_flask.Auth(
    app,
    authority=os.environ['AUTHORITY'],
    client_id=os.environ['CLIENT_ID'],
    client_credential=os.environ['CLIENT_SECRET'],
    redirect_uri=os.environ['REDIRECT_URI'],
    post_logout_view=logoutapp
)

#-----------------------------------------------------------------------
```

PennyTestEntra/penny.py (Page 2 of 2)

```

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

    username = context['user'].get('preferred_username', '')

    html = flask.render_template('index.html', username=username)
    response = flask.make_response(html)
    return response

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

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

    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
```

PennyTestEntra/authstub.py (Page 1 of 1)

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

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

from functools import wraps
import flask

from top import app

# For testing only, so OK to define in source code:
app.secret_key = '12345'

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

@app.route('/login', methods=['GET'])
def login():
    html_code = '''
        <form action="handlelogin" method="get">
        Username:
        <input type="text" name="username" id="username">
        <input type="submit" id="submit">'''
    response = flask.make_response(html_code)
    return response

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

@app.route('/handlelogin', methods=['GET'])
def handle_login():
    username = flask.request.args.get('username')
    flask.session['username'] = username
    flask.abort(flask.redirect('/index'))

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

class Auth:
    def __init__(self, application, *args, **kwargs):
        pass

    def login_required(self, function=None):
        @wraps(function)
        def wrapper(*args, **kwargs):
            username = flask.session.get('username', None)
            if username is None:
                flask.abort(flask.redirect('/login'))
            context = {"user": {"preferred_username": username}}
            return function(*args, context=context, **kwargs)
        return wrapper
```

blank (Page 1 of 1)

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

PennyTestEntra/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)

        # Move past the phony login page.
        username_input = self._page.locator('#username')
        username_input.fill('rdondero')
        submit_button = self._page.locator('#submit')
        submit_button.click()

    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()
```

PennyTestEntra/testpenny.py (Page 2 of 2)

```
        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)):
            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')
```