

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.

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/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: PORT = 5000
12:
13: def main():
14:
15:     if len(sys.argv) != 1:
16:         print('Usage: ' + sys.argv[0], file=sys.stderr)
17:         sys.exit(1)
18:
19:     try:
20:         penny.app.run(host='0.0.0.0', port=PORT, debug=True)
21:     except Exception as ex:
22:         print(ex, file=sys.stderr)
23:         sys.exit(1)
24:
25: if __name__ == '__main__':
26:     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?mode=ro'
14:
15: #-----
16:
17: def get_books(author):
18:
19:     books = []
20:
21:     with sqlite3.connect(_DATABASE_URL, isolation_level=None,
22:         uri=True) as connection:
23:
24:         with contextlib.closing(connection.cursor()) as cursor:
25:
26:             query_str = "SELECT isbn, author, title FROM books "
27:             query_str += "WHERE author LIKE ?"
28:             cursor.execute(query_str, [author+'%'])
29:
30:             table = cursor.fetchall()
31:             for row in table:
32:                 book = {'isbn': row[0], 'author': row[1],
33:                     'title': row[2]}
34:                 books.append(book)
35:
36:     return books

```

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:   <body>
7:
8:
9:     <hr>
10:    Hello and welcome to <strong>Penny.com</strong>
11:    <hr>
12:
13:    <h1>Author Search</h1>
14:    Please enter an author name:
15:    <input type="text" id="authorInput" autoFocus>
16:    <hr>
17:    <div id="resultsDiv"></div>
18:
19:    <hr>
20:    Created by <a href="https://www.cs.princeton.edu/~rdondero">
21:    Bob Dondero</a>
22:    <hr>
23:
24:    <script>
25:      'use strict';
26:
27:
28:      function escape(s) {
29:        s = s.replace('&', '&amp;');
30:        s = s.replace('<', '&lt;');
31:        s = s.replace('>', '&gt;');
32:        s = s.replace('"', '&quot;');
33:        s = s.replace("'", '&apos;');
34:        return s;
35:      }
36:
37:      function convertToHtml(books) {
38:        let html = '';
39:        for (let book of books) {
40:          html += escape(book['isbn']) + ': ';
41:          html += '<strong>'
42:          html += escape(book['author']);
43:          html += '</strong>: ';
44:          html += escape(book['title']);
45:          html += '<br>';
46:        }
47:        return html;
48:      }
49:
50:      function handleResponse() {
51:        if (this.status !== 200) {
52:          alert('Error: Failed to fetch data from server');
53:          return;
54:        }
55:        let books = JSON.parse(this.response);
56:        let html = convertToHtml(books);
57:        let resultsDiv = document.getElementById('resultsDiv');
58:        resultsDiv.innerHTML = html;
59:      }
60:
61:      function handleError() {
62:        alert('Error: Failed to fetch data from server');
63:      }
64:
65:      let request = null;

```

PennyTest/index.html (Page 2 of 2)

```

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

```

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')

```

PennyTestCas/index.html (Page 1 of 2)

```

1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Penny.com</title>
5:   </head>
6:   <body>
7:     <hr>
8:     Hello {{username}} and welcome to
9:     <strong>Penny.com</strong>
10:    <hr>
11:    <h1>Author Search</h1>
12:    Please enter an author name:
13:    <input type="text" id="authorInput" autoFocus>
14:    <hr>
15:    <div id="resultsDiv"></div>
16:    <hr>
17:    <a href="/logoutapp">Log out of the app</a><br>
18:    <a href="/logoutcas">Log out of the app and CAS</a><br>
19:    Created by <a href="https://www.cs.princeton.edu/~rdondero">
20:    Bob Dondero</a>
21:    <hr>
22:    <script>
23:      'use strict';
24:
25:      function escape(s) {
26:        s = s.replace('&', '&amp;');
27:        s = s.replace('<', '&lt;');
28:        s = s.replace('>', '&gt;');
29:        s = s.replace('"', '&quot;');
30:        s = s.replace("'", '&apos;');
31:        return s;
32:      }
33:
34:      function convertToHtml(books) {
35:        let html = '';
36:        for (let book of books) {
37:          html += escape(book['isbn']) + ': ';
38:          html += '<strong>'
39:          html += escape(book['author']);
40:          html += '</strong>: ';
41:          html += escape(book['title']);
42:          html += '<br>';
43:        }
44:        return html;
45:      }
46:
47:      function handleResponse() {
48:        if (this.status !== 200) {
49:          alert('Error: Failed to fetch data from server');
50:          return;
51:        }
52:        let books = JSON.parse(this.response);
53:        let html = convertToHtml(books);
54:        let resultsDiv = document.getElementById('resultsDiv');
55:        resultsDiv.innerHTML = html;
56:      }
57:
58:      function handleError() {
59:        alert('Error: Failed to fetch data from server');

```

PennyTestCas/index.html (Page 2 of 2)

```

60:    }
61:
62:    let request = null;
63:
64:    function getResults() {
65:      let authorInput = document.getElementById('authorInput');
66:      let author = authorInput.value;
67:      let encodedAuthor = encodeURIComponent(author);
68:      let url = '/searchresults?author=' + encodedAuthor;
69:      if (request !== null)
70:        request.abort();
71:      request = new XMLHttpRequest();
72:      request.onload = handleResponse;
73:      request.onerror = handleError;
74:      request.open('GET', url);
75:      request.send();
76:    }
77:
78:    let timer = null;
79:
80:    function debouncedGetResults() {
81:      clearTimeout(timer);
82:      timer = window.setTimeout(getResults, 500);
83:    }
84:
85:    function setup() {
86:      let authorInput = document.getElementById('authorInput');
87:      authorInput.addEventListener('input', debouncedGetResults);
88:    }
89:
90:    document.addEventListener('DOMContentLoaded', setup);
91:
92:    </script>
93:  </body>
94: </html>

```

PennyTestCas/loggedout.html (Page 1 of 1)

```
1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Penny.com</title>
5:   </head>
6:   <body>
7:     <h2>You are logged out of Penny.com</h2>
8:     <a href="/index">Revisit the application</a>
9:   </body>
10: </html>
```

PennyTestCas/top.py (Page 1 of 1)

```
1: #!/usr/bin/env python
2:
3: #-----
4: # top.py
5: # Author: Bob Dondero
6: #-----
7:
8: import flask
9:
10: app = flask.Flask(__name__, template_folder='.')
```

PennyTestCas/penny.py (Page 1 of 2)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # penny.py
5: # Author: Bob Dondero
6: #-----
7:
8: import os
9: import json
10: import flask
11: import flask_session
12: import flask_sqlalchemy
13: import dotenv
14: import database
15:
16: from top import app
17:
18: dotenv.load_dotenv()
19:
20: TESTING = os.getenv('TESTING', 'no')
21: if TESTING == 'yes':
22:     import authstub as auth
23: else:
24:     import auth
25:
26: #-----
27:
28: # Session configuration to store session data in a database:
29:
30: session_database_url = os.getenv('SESSION_DATABASE_URL',
31:     'sqlite:///pennysessions.sqlite')
32: session_database_url = session_database_url.replace(
33:     'postgres://', 'postgresql://')
34: app.config['SESSION_PERMANENT'] = False
35: app.config['SESSION_TYPE'] = 'sqlalchemy'
36: app.config['SQLALCHEMY_DATABASE_URI'] = session_database_url
37: app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
38: app.config['SESSION_SQLALCHEMY'] = flask_sqlalchemy.SQLAlchemy(app)
39: flask_session.Session(app)
40:
41: #-----
42:
43: @app.route('/', methods=['GET'])
44: @app.route('/index', methods=['GET'])
45: def index():
46:
47:     auth.authenticate()
48:     username = auth.get_username()
49:
50:     html = flask.render_template('index.html', username=username)
51:     response = flask.make_response(html)
52:     return response
53:
54: #-----
55:
56: @app.route('/searchresults', methods=['GET'])
57: def search_results():
58:
59:     if not auth.is_authenticated():
60:         flask.abort(403)
61:
62:     author = flask.request.args.get('author')
63:     if author is None:
64:         author = ''
65:     author = author.strip()

```

PennyTestCas/penny.py (Page 2 of 2)

```

66:
67:     if author == '':
68:         books = []
69:     else:
70:         books = database.get_books(author) # Exception handling omitted
71:
72:     json_doc = json.dumps(books)
73:     response = flask.make_response(json_doc)
74:     response.headers['Content-Type'] = 'application/json'
75:     return response

```

PennyTestCas/auth.py (Page 1 of 3)

```

1:#!/usr/bin/env python
2:
3:#-----
4:# auth.py
5:# Authors: Alex Halderman, Scott Karlin, Brian Kernighan, Bob Dondero,
6:#           and Joshua Lau '26
7:#-----
8:
9:import urllib.request
10:import urllib.parse
11:import re
12:import json
13:import flask
14:
15:from top import app
16:
17:#-----
18:
19:_CAS_URL = 'https://fed.princeton.edu/cas/'
20:
21:#-----
22:
23:@app.route('/logoutapp', methods=['GET'])
24:def logoutapp():
25:
26:    # Log out of the application.
27:    flask.session.clear()
28:    return flask.send_file('loggedout.html')
29:
30:#-----
31:
32:@app.route('/logoutcas', methods=['GET'])
33:def logoutcas():
34:
35:    # Log out of the CAS session, and then the application.
36:    logout_url = (_CAS_URL + 'logout?service='
37:                  + urllib.parse.quote(
38:                      re.sub('logoutcas', 'logoutapp', flask.request.url)))
39:    flask.abort(flask.redirect(logout_url))
40:
41:#-----
42:
43:# Return url after stripping out the "ticket" parameter that was
44:# added by the CAS server.
45:
46:def strip_ticket(url):
47:    if url is None:
48:        return "something is badly wrong"
49:    url = re.sub(r'ticket=[^&]*&', '', url)
50:    url = re.sub(r'?&?&|&$', '', url)
51:    return url
52:
53:#-----
54:
55:# Validate a login ticket by contacting the CAS server. If
56:# valid, return the user's user_info; otherwise, return None.
57:
58:def validate(ticket):
59:    val_url = (_CAS_URL + "validate"
60:              + '?service='
61:              + urllib.parse.quote(strip_ticket(flask.request.url))
62:              + '&ticket='
63:              + urllib.parse.quote(ticket)
64:              + '&format=json')
65:    with urllib.request.urlopen(val_url) as flo:

```

PennyTestCas/auth.py (Page 2 of 3)

```

66:        result = json.loads(flo.read()).decode('utf-8'))
67:
68:    if (not result) or ('serviceResponse' not in result):
69:        return None
70:
71:    service_response = result['serviceResponse']
72:
73:    if 'authenticationSuccess' in service_response:
74:        user_info = service_response['authenticationSuccess']
75:        return user_info
76:
77:    if 'authenticationFailure' in service_response:
78:        print('CAS authentication failure:', service_response)
79:        return None
80:
81:    print('Unexpected CAS response:', service_response)
82:    return None
83:
84:#-----
85:
86:def is_authenticated():
87:
88:    return 'user_info' in flask.session
89:
90:#-----
91:
92:def get_user_info():
93:
94:    return flask.session.get('user_info')
95:
96:#-----
97:
98:def get_username():
99:
100:    user_info = flask.session.get('user_info')
101:    if user_info is None:
102:        return ''
103:    return user_info.get('user', '')
104:
105:#-----
106:
107:# Authenticate the user. Do not return unless the user is
108:# successfully authenticated.
109:
110:def authenticate():
111:
112:    # If the user_info is in the session, then the user was
113:    # authenticated previously. So return.
114:    if flask.session.get('user_info') is not None:
115:        return
116:
117:    # If the request does not contain a login ticket, then redirect
118:    # the browser to the login page to get one.
119:    ticket = flask.request.args.get('ticket')
120:    if ticket is None:
121:        login_url = (_CAS_URL + 'login?service=' +
122:                    urllib.parse.quote(flask.request.url))
123:        flask.abort(flask.redirect(login_url))
124:
125:    # If the login ticket is invalid, then redirect the browser
126:    # to the login page to get a new one.
127:    user_info = validate(ticket)
128:    if user_info is None:
129:        login_url = (_CAS_URL + 'login?service='
130:                    + 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)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # authstub.py
5: # Author: Bob Dondero
6: #-----
7:
8: import flask
9:
10: from top import app
11:
12: #-----
13:
14: @app.route('/logoutapp', methods=['GET'])
15: def logoutapp():
16:
17:     # Log out of the application.
18:     flask.session.clear()
19:     return flask.send_file('loggedout.html')
20:
21: #-----
22:
23: @app.route('/logoutcas', methods=['GET'])
24: def logoutcas():
25:
26:     flask.abort(flask.redirect('/logoutapp'))
27:
28: #-----
29:
30: def is_authenticated():
31:
32:     return 'username' in flask.session
33:
34: #-----
35:
36: def get_user_info():
37:
38:     return None
39:
40: #-----
41:
42: def get_username():
43:
44:     return flask.session.get('username', '')
45:
46: #-----
47:
48: @app.route('/handlelogin', methods=['GET'])
49: def handlelogin():
50:     username = flask.request.args.get('username')
51:     flask.session['username'] = username
52:     flask.abort(flask.redirect('/index'))
53:
54: #-----
55:
56: @app.route('/login', methods=['GET'])
57: def login():
58:     html_code = '''
59:     <form action="handlelogin" method="get">
60:         Username:
61:         <input type="text" name="username" id="username">
62:         <input type="submit" id="submit">'''
63:     response = flask.make_response(html_code)
64:     return response
65:

```

PennyTestCas/authstub.py (Page 2 of 2)

```

66: #-----
67:
68: # Authenticate the user. Do not return unless the user is
69: # successfully authenticated.
70:
71: def authenticate():
72:
73:     # If the user_info is in the session, then the user was
74:     # authenticated previously. So return.
75:     if flask.session.get('username') is not None:
76:         return
77:
78:     flask.abort(flask.redirect('/login'))

```

PennyTestCas/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:         # Move past the phony login page.
50:         username_input = self._page.locator('#username')
51:         username_input.fill('rdondero')
52:         submit_button = self._page.locator('#submit')
53:         submit_button.click()
54:
55:     def tearDown(self):
56:         # Enter an author, thus triggering an AJAX call.
57:         author_input = self._page.locator('#authorInput')
58:         author_input.fill(self._author)
59:
60:         # Wait for the AJAX call to finish.
61:         time.sleep(DELAY)
62:
63:         # Fetch the contents of the resultsDiv element.
64:         results_div = self._page.locator('#resultsDiv')
65:         results_text = results_div.inner_text()

```

PennyTestCas/testpenny.py (Page 2 of 2)

```

66:         books = results_text.split('\n')[:-1]
67:
68:         # Compare those contents with the expected contents.
69:         self.assertEqual(len(books), len(self._expected))
70:         books.sort()
71:         for i in range(len(books)):
72:             self.assertEqual(books[i], self._expected[i])
73:
74:     def test_ker(self):
75:         self._author = 'ker'
76:         self._expected = [
77:             '123: Kernighan: The Practice of Programming',
78:             '234: Kernighan: The C Programming Language'
79:         ]
80:
81:     def test_sed(self):
82:         self._author = 'sed'
83:         self._expected = [
84:             '345: Sedgewick: Algorithms in C'
85:         ]
86:
87:     def test_abc(self):
88:         self._author = 'abc'
89:         self._expected = []
90:
91:     def test_spaces(self):
92:         self._author = ' '
93:         self._expected = []
94:
95:     def test_ker_spaces(self):
96:         self._author = ' kernigh '
97:         self._expected = [
98:             '123: Kernighan: The Practice of Programming',
99:             '234: Kernighan: The C Programming Language'
100:         ]
101:
102:     def test_ker_bad_spaces(self):
103:         self._author = ' ker nigh '
104:         self._expected = []
105:
106: #-----
107:
108: if __name__ == '__main__':
109:     unittest.main(verbosity=2, module='testpenny')

```


PennyTestGoogle/index.html (Page 1 of 2)

```

1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Penny.com</title>
5:   </head>
6:   <body>
7:     <hr>
8:     Hello {{username}} and welcome to
9:     <strong>Penny.com</strong>
10:    <hr>
11:    <h1>Author Search</h1>
12:    Please enter an author name:
13:    <input type="text" id="authorInput" autoFocus>
14:    <hr>
15:    <div id="resultsDiv"></div>
16:
17:    <hr>
18:    <a href="/logoutapp">Log out of the app</a><br>
19:    <a href="/logoutgoogle">Log out of the app and Google</a><br>
20:    Created by <a href="https://www.cs.princeton.edu/~rdondero">
21:    Bob Dondero</a>
22:    <hr>
23:
24:    <script>
25:      'use strict';
26:
27:      function escape(s) {
28:        s = s.replace('&', '&amp;');
29:        s = s.replace('<', '&lt;');
30:        s = s.replace('>', '&gt;');
31:        s = s.replace('"', '&quot;');
32:        s = s.replace("'", '&apos;');
33:        return s;
34:      }
35:
36:      function convertToHtml(books) {
37:        let html = '';
38:        for (let book of books) {
39:          html += escape(book['isbn']) + ': ';
40:          html += '<strong>'
41:          html += escape(book['author']);
42:          html += '</strong>: ';
43:          html += escape(book['title']);
44:          html += '<br>';
45:        }
46:        return html;
47:      }
48:
49:      function handleResponse() {
50:        if (this.status !== 200) {
51:          alert('Error: Failed to fetch data from server');
52:          return;
53:        }
54:        let books = JSON.parse(this.response);
55:        let html = convertToHtml(books);
56:        let resultsDiv = document.getElementById('resultsDiv');
57:        resultsDiv.innerHTML = html;
58:      }
59:
60:      function handleError() {
61:        alert('Error: Failed to fetch data from server');

```

PennyTestGoogle/index.html (Page 2 of 2)

```

62:    }
63:
64:    let request = null;
65:
66:    function getResults() {
67:      let authorInput = document.getElementById('authorInput');
68:      let author = authorInput.value;
69:      let encodedAuthor = encodeURIComponent(author);
70:      let url = '/searchresults?author=' + encodedAuthor;
71:      if (request !== null)
72:        request.abort();
73:      request = new XMLHttpRequest();
74:      request.onload = handleResponse;
75:      request.onerror = handleError;
76:      request.open('GET', url);
77:      request.send();
78:    }
79:
80:    let timer = null;
81:
82:    function debouncedGetResults() {
83:      clearTimeout(timer);
84:      timer = window.setTimeout(getResults, 500);
85:    }
86:
87:    function setup() {
88:      let authorInput = document.getElementById('authorInput');
89:      authorInput.addEventListener('input', debouncedGetResults);
90:    }
91:
92:    document.addEventListener('DOMContentLoaded', setup);
93:
94:    </script>
95:  </body>
96: </html>

```

PennyTestGoogle/loggedout.html (Page 1 of 1)

```
1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Penny.com</title>
5:   </head>
6:   <body>
7:     <h2>You are logged out of Penny.com</h2>
8:     <a href="/index">Revisit the application</a>
9:   </body>
10: </html>
```

PennyTestGoogle/top.py (Page 1 of 1)

```
1: #!/usr/bin/env python
2:
3: #-----
4: # top.py
5: # Author: Bob Dondero
6: #-----
7:
8: import flask
9:
10: app = flask.Flask(__name__, template_folder='.')
```

PennyTestGoogle/penny.py (Page 1 of 2)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # penny.py
5: # Author: Bob Dondero
6: #-----
7:
8: import os
9: import json
10: import flask
11: import flask_session
12: import flask_sqlalchemy
13: import dotenv
14: import database
15:
16: from top import app
17:
18: dotenv.load_dotenv()
19:
20: TESTING = os.getenv('TESTING', 'no')
21: if TESTING == 'yes':
22:     import authstub as auth
23: else:
24:     import auth
25:
26: #-----
27:
28: # Session configuration to store session data in a database:
29:
30: session_database_url = os.getenv('SESSION_DATABASE_URL',
31:     'sqlite:///pennysessions.sqlite')
32: session_database_url = session_database_url.replace(
33:     'postgres://', 'postgresql://')
34: app.config['SESSION_PERMANENT'] = False
35: app.config['SESSION_TYPE'] = 'sqlalchemy'
36: app.config['SQLALCHEMY_DATABASE_URI'] = session_database_url
37: app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
38: app.config['SESSION_SQLALCHEMY'] = flask_sqlalchemy.SQLAlchemy(app)
39: flask_session.Session(app)
40:
41: #-----
42:
43: @app.route('/', methods=['GET'])
44: @app.route('/index', methods=['GET'])
45: def index():
46:
47:     auth.authenticate()
48:     username = auth.get_username()
49:
50:     html = flask.render_template('index.html', username=username)
51:     response = flask.make_response(html)
52:     return response
53:
54: #-----
55:
56: @app.route('/searchresults', methods=['GET'])
57: def search_results():
58:
59:     if not auth.is_authenticated():
60:         flask.abort(403)
61:
62:     author = flask.request.args.get('author')
63:     if author is None:
64:         author = ''
65:     author = author.strip()

```

PennyTestGoogle/penny.py (Page 2 of 2)

```

66:
67:     if author == '':
68:         books = []
69:     else:
70:         books = database.get_books(author) # Exception handling omitted
71:
72:     json_doc = json.dumps(books)
73:     response = flask.make_response(json_doc)
74:     response.headers['Content-Type'] = 'application/json'
75:     return response

```

PennyTestGoogle/auth.py (Page 1 of 4)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # auth.py
5: # Author: Bob Dondero
6: # With lots of help from https://realpython.com/flask-google-login/
7: #-----
8:
9: import os
10: import json
11: import requests
12: import flask
13: import oauthlib.oauth2
14: import dotenv
15:
16: from top import app
17:
18: #-----
19:
20: GOOGLE_DISCOVERY_URL = (
21:     'https://accounts.google.com/.well-known/openid-configuration')
22:
23: dotenv.load_dotenv()
24: GOOGLE_CLIENT_ID = os.environ['GOOGLE_CLIENT_ID']
25: GOOGLE_CLIENT_SECRET = os.environ['GOOGLE_CLIENT_SECRET']
26:
27: client = oauthlib.oauth2.WebApplicationClient(GOOGLE_CLIENT_ID)
28:
29: #-----
30:
31: @app.route('/login', methods=['GET'])
32: def login():
33:
34:     original_url = flask.request.args.get('originalurl')
35:
36:     # Determine the URL for Google login.
37:     google_provider_cfg = requests.get(
38:         GOOGLE_DISCOVERY_URL, timeout=2).json()
39:     authorization_endpoint = (
40:         google_provider_cfg['authorization_endpoint'])
41:
42:     # Construct the request URL for Google login, providing scopes
43:     # to fetch the user's profile data.
44:     request_uri = client.prepare_request_uri(
45:         authorization_endpoint,
46:         redirect_uri = flask.request.base_url + '/callback',
47:         scope=['openid', 'email', 'profile'],
48:         state=original_url
49:     )
50:
51:     #-----
52:     # For learning:
53:     # print('request_uri:', request_uri, file=sys.stderr)
54:     #-----
55:
56:     # Redirect to the request URL.
57:     return flask.redirect(request_uri)
58:
59: #-----
60:
61: @app.route('/login/callback', methods=['GET'])
62: def callback():
63:
64:     original_url = flask.request.args.get('state')
65:

```

PennyTestGoogle/auth.py (Page 2 of 4)

```

66:     # Get the authorization code that Google sent.
67:     code = flask.request.args.get('code')
68:
69:     #-----
70:     # For learning:
71:     # print('code:', code, file=sys.stderr)
72:     #-----
73:
74:     # Determine the URL to fetch tokens that allow the application to
75:     # ask for the user's profile data.
76:     google_provider_cfg = requests.get(
77:         GOOGLE_DISCOVERY_URL, timeout=2).json()
78:     token_endpoint = google_provider_cfg['token_endpoint']
79:
80:     # Construct a request to fetch the tokens.
81:     token_url, headers, body = client.prepare_token_request(
82:         token_endpoint,
83:         authorization_response=flask.request.url,
84:         redirect_url=flask.request.base_url,
85:         code=code
86:     )
87:
88:     #-----
89:     # For learning:
90:     # print('token_url:', token_url, file=sys.stderr)
91:     # print('headers:', headers, file=sys.stderr)
92:     # print('body:', body, file=sys.stderr)
93:     #-----
94:
95:     # Fetch the tokens.
96:     token_response = requests.post(
97:         token_url,
98:         headers=headers,
99:         data=body,
100:         auth=(GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET),
101:         timeout=2
102:     )
103:
104:     #-----
105:     # For learning:
106:     # print('token_response.json():', token_response.json(),
107:     #       file=sys.stderr)
108:     #-----
109:
110:     # Parse the tokens.
111:     client.parse_request_body_response(
112:         json.dumps(token_response.json()))
113:
114:     # Using the tokens, fetch the user's profile data,
115:     # including the user's Google profile image and email address.
116:     userinfo_endpoint = google_provider_cfg['userinfo_endpoint']
117:     uri, headers, body = client.add_token(userinfo_endpoint)
118:
119:     #-----
120:     # For learning:
121:     # print('uri:', uri, file=sys.stderr)
122:     # print('headers:', headers, file=sys.stderr)
123:     # print('body:', body, file=sys.stderr)
124:     #-----
125:
126:     userinfo_response = requests.get(uri, headers=headers, data=body,
127:                                     timeout=2)
128:
129:     #-----
130:     # 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)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # authstub.py
5: # Author: Bob Dondero
6: #-----
7:
8: import flask
9:
10: from top import app
11:
12: #-----
13:
14: @app.route('/logoutapp', methods=['GET'])
15: def logoutapp():
16:
17:     # Log out of the application.
18:     flask.session.clear()
19:     return flask.send_file('loggedout.html')
20:
21: #-----
22:
23: @app.route('/logoutgoogle', methods=['GET'])
24: def logoutcas():
25:
26:     flask.abort(flask.redirect('/logoutapp'))
27:
28: #-----
29:
30: def is_authenticated():
31:
32:     return 'username' in flask.session
33:
34: #-----
35:
36: def get_user_info():
37:
38:     return None
39:
40: #-----
41:
42: def get_username():
43:
44:     return flask.session.get('username', '')
45:
46: #-----
47:
48: @app.route('/handlelogin', methods=['GET'])
49: def handlelogin():
50:     username = flask.request.args.get('username')
51:     flask.session['username'] = username
52:     flask.abort(flask.redirect('/index'))
53:
54: #-----
55:
56: @app.route('/login', methods=['GET'])
57: def login():
58:     html_code = '''
59:         <form action="handlelogin" method="get">
60:             Username:
61:             <input type="text" name="username" id="username">
62:             <input type="submit" id="submit">'''
63:     response = flask.make_response(html_code)
64:     return response
65:

```

PennyTestGoogle/authstub.py (Page 2 of 2)

```

66: #-----
67:
68: # Authenticate the user. Do not return unless the user is
69: # successfully authenticated.
70:
71: def authenticate():
72:
73:     # If the user_info is in the session, then the user was
74:     # authenticated previously. So return.
75:     if flask.session.get('username') is not None:
76:         return
77:
78:     flask.abort(flask.redirect('/login'))

```

PennyTestGoogle/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:         # Move past the phony login page.
50:         username_input = self._page.locator('#username')
51:         username_input.fill('rdondero')
52:         submit_button = self._page.locator('#submit')
53:         submit_button.click()
54:
55:     def tearDown(self):
56:         # Enter an author, thus triggering an AJAX call.
57:         author_input = self._page.locator('#authorInput')
58:         author_input.fill(self._author)
59:
60:         # Wait for the AJAX call to finish.
61:         time.sleep(DELAY)
62:
63:         # Fetch the contents of the resultsDiv element.
64:         results_div = self._page.locator('#resultsDiv')
65:         results_text = results_div.inner_text()

```

PennyTestGoogle/testpenny.py (Page 2 of 2)

```

66:         books = results_text.split('\n')[:-1]
67:
68:         # Compare those contents with the expected contents.
69:         self.assertEqual(len(books), len(self._expected))
70:         books.sort()
71:         for i in range(len(books)):
72:             self.assertEqual(books[i], self._expected[i])
73:
74:     def test_ker(self):
75:         self._author = 'ker'
76:         self._expected = [
77:             '123: Kernighan: The Practice of Programming',
78:             '234: Kernighan: The C Programming Language'
79:         ]
80:
81:     def test_sed(self):
82:         self._author = 'sed'
83:         self._expected = [
84:             '345: Sedgewick: Algorithms in C'
85:         ]
86:
87:     def test_abc(self):
88:         self._author = 'abc'
89:         self._expected = []
90:
91:     def test_spaces(self):
92:         self._author = ' '
93:         self._expected = []
94:
95:     def test_ker_spaces(self):
96:         self._author = ' kernigh '
97:         self._expected = [
98:             '123: Kernighan: The Practice of Programming',
99:             '234: Kernighan: The C Programming Language'
100:         ]
101:
102:     def test_ker_bad_spaces(self):
103:         self._author = ' ker nigh '
104:         self._expected = []
105:
106: #-----
107:
108: if __name__ == '__main__':
109:     unittest.main(verbosity=2, module='testpenny')

```

PennyTestEntra/index.html (Page 1 of 2)

```

1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Penny.com</title>
5:   </head>
6:   <body>
7:
8:     <hr>
9:     Hello {{username}} and welcome to
10:    <strong>Penny.com</strong>
11:    <hr>
12:
13:    <h1>Author Search</h1>
14:    Please enter an author name:
15:    <input type="text" id="authorInput" autoFocus>
16:    <hr>
17:    <div id="resultsDiv"></div>
18:
19:    <hr>
20:    <a href="/logoutapp">Log out of the app</a><br>
21:    <a href="/logoutentra">Log out of the app and Entra ID</a><br>
22:    Created by <a href="https://www.cs.princeton.edu/~rdondero">
23:    Bob Dondero</a>
24:    <hr>
25:
26:    <script>
27:      'use strict';
28:
29:      function escape(s) {
30:        s = s.replace('&', '&amp;');
31:        s = s.replace('<', '&lt;');
32:        s = s.replace('>', '&gt;');
33:        s = s.replace('"', '&quot;');
34:        s = s.replace("'", '&apos;');
35:        return s;
36:      }
37:
38:      function convertToHtml(books) {
39:        let html = '';
40:        for (let book of books) {
41:          html += escape(book['isbn']) + ': ';
42:          html += '<strong>';
43:          html += escape(book['author']);
44:          html += '</strong>: ';
45:          html += escape(book['title']);
46:          html += '<br>';
47:        }
48:        return html;
49:      }
50:
51:      function handleResponse() {
52:        if (this.status !== 200) {
53:          alert('Error: Failed to fetch data from server');
54:          return;
55:        }
56:        let books = JSON.parse(this.response);
57:        let html = convertToHtml(books);
58:        let resultsDiv = document.getElementById('resultsDiv');
59:        resultsDiv.innerHTML = html;
60:      }
61:
62:      function handleError() {
63:        alert('Error: Failed to fetch data from server');
64:      }
65:

```

PennyTestEntra/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>

```


PennyTestEntra/loggedout.html (Page 1 of 1)

```
1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Penny.com</title>
5:   </head>
6:   <body>
7:     <h2>You are logged out of Penny.com</h2>
8:     <a href="/index">Revisit the application</a>
9:   </body>
10: </html>
```

PennyTestEntra/top.py (Page 1 of 1)

```
1: #!/usr/bin/env python
2:
3: #-----
4: # top.py
5: # Author: Bob Dondero
6: #-----
7:
8: import flask
9:
10: app = flask.Flask(__name__, template_folder='.')
```

PennyTestEntra/penny.py (Page 1 of 2)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # penny.py
5: # Author: Bob Dondero
6: #-----
7:
8: import os
9: import json
10: import flask
11: import flask_sqlalchemy
12: import dotenv
13: import database
14: from top import app
15:
16: dotenv.load_dotenv()
17:
18: TESTING = os.getenv('TESTING', 'no')
19: if TESTING == 'yes':
20:     import authstub as identity_flask
21: else:
22:     import identity.flask as identity_flask
23:
24: #-----
25:
26: # Session configuration to store session data in a database:
27:
28: session_database_url = os.getenv('SESSION_DATABASE_URL',
29:     'sqlite:///pennysessions.sqlite')
30: session_database_url = session_database_url.replace(
31:     'postgres://', 'postgresql://')
32: app.config['SESSION_PERMANENT'] = False
33: app.config['SESSION_TYPE'] = 'sqlalchemy'
34: app.config['SQLALCHEMY_DATABASE_URI'] = session_database_url
35: app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
36: app.config['SESSION_SQLALCHEMY'] = flask_sqlalchemy.SQLAlchemy(app)
37:
38: #-----
39:
40: # Microsoft Entra ID configuration
41:
42: @app.route('/logoutapp', methods=['GET'])
43: def logoutapp():
44:
45:     flask.session.clear()
46:     return flask.send_file('loggedout.html')
47:
48: @app.route('/logoutentra', methods=['GET'])
49: def logoutentra():
50:
51:     response = flask.redirect(flask.url_for('identity.logout'))
52:     return response
53:
54: app.config['SCOPE'] = os.environ['SCOPE']
55: app.config['ENDPOINT'] = os.environ['ENDPOINT']
56: auth = identity_flask.Auth(
57:     app,
58:     authority=os.environ['AUTHORITY'],
59:     client_id=os.environ['CLIENT_ID'],
60:     client_credential=os.environ['CLIENT_SECRET'],
61:     redirect_uri=os.environ['REDIRECT_URI'],
62:     post_logout_view=logoutapp
63: )
64:
65: #-----

```

PennyTestEntra/penny.py (Page 2 of 2)

```

66:
67: @app.route('/', methods=['GET'])
68: @app.route('/index', methods=['GET'])
69: @auth.login_required
70: def index(*, context):
71:
72:     username = context['user'].get('preferred_username', '')
73:
74:     html = flask.render_template('index.html', username=username)
75:     response = flask.make_response(html)
76:     return response
77:
78: #-----
79:
80: @app.route('/searchresults', methods=['GET'])
81: @auth.login_required
82: def search_results(*, context):
83:
84:     author = flask.request.args.get('author')
85:     if author is None:
86:         author = ''
87:     author = author.strip()
88:
89:     if author == '':
90:         books = []
91:     else:
92:         books = database.get_books(author) # Exception handling omitted
93:
94:     json_doc = json.dumps(books)
95:     response = flask.make_response(json_doc)
96:     response.headers['Content-Type'] = 'application/json'
97:     return response

```

PennyTestEntra/authstub.py (Page 1 of 1)

```

1: #!/usr/bin/env python
2:
3: #-----
4: # authstub.py
5: # Author: Bob Dondero
6: #-----
7:
8: from functools import wraps
9: import flask
10:
11: from top import app
12:
13: # For testing only, so OK to define in source code:
14: app.secret_key = '12345'
15:
16: #-----
17:
18: @app.route('/login', methods=['GET'])
19: def login():
20:     html_code = '''
21:         <form action="handlelogin" method="get">
22:             Username:
23:             <input type="text" name="username" id="username">
24:             <input type="submit" id="submit">'''
25:     response = flask.make_response(html_code)
26:     return response
27:
28: #-----
29:
30: @app.route('/handlelogin', methods=['GET'])
31: def handle_login():
32:     username = flask.request.args.get('username')
33:     flask.session['username'] = username
34:     flask.abort(flask.redirect('/index'))
35:
36: #-----
37:
38: class Auth:
39:     def __init__(self, application, *args, **kwargs):
40:         pass
41:
42:     def login_required(self, function=None):
43:         @wraps(function)
44:         def wrapper(*args, **kwargs):
45:             username = flask.session.get('username', None)
46:             if username is None:
47:                 flask.abort(flask.redirect('/login'))
48:             context = {"user": {"preferred_username": username}}
49:             return function(*args, context=context, **kwargs)
50:         return wrapper

```

blank (Page 1 of 1)

1: This page is intentionally blank.

PennyTestEntra/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:         # Move past the phony login page.
50:         username_input = self._page.locator('#username')
51:         username_input.fill('rdondero')
52:         submit_button = self._page.locator('#submit')
53:         submit_button.click()
54:
55:     def tearDown(self):
56:         # Enter an author, thus triggering an AJAX call.
57:         author_input = self._page.locator('#authorInput')
58:         author_input.fill(self._author)
59:
60:         # Wait for the AJAX call to finish.
61:         time.sleep(DELAY)
62:
63:         # Fetch the contents of the resultsDiv element.
64:         results_div = self._page.locator('#resultsDiv')
65:         results_text = results_div.inner_text()

```

PennyTestEntra/testpenny.py (Page 2 of 2)

```

66:         books = results_text.split('\n')[:-1]
67:
68:         # Compare those contents with the expected contents.
69:         self.assertEqual(len(books), len(self._expected))
70:         books.sort()
71:         for i in range(len(books)):
72:             self.assertEqual(books[i], self._expected[i])
73:
74:     def test_ker(self):
75:         self._author = 'ker'
76:         self._expected = [
77:             '123: Kernighan: The Practice of Programming',
78:             '234: Kernighan: The C Programming Language'
79:         ]
80:
81:     def test_sed(self):
82:         self._author = 'sed'
83:         self._expected = [
84:             '345: Sedgewick: Algorithms in C'
85:         ]
86:
87:     def test_abc(self):
88:         self._author = 'abc'
89:         self._expected = []
90:
91:     def test_spaces(self):
92:         self._author = ' '
93:         self._expected = []
94:
95:     def test_ker_spaces(self):
96:         self._author = ' kernigh '
97:         self._expected = [
98:             '123: Kernighan: The Practice of Programming',
99:             '234: Kernighan: The C Programming Language'
100:         ]
101:
102:     def test_ker_bad_spaces(self):
103:         self._author = ' ker nigh '
104:         self._expected = []
105:
106: #-----
107:
108: if __name__ == '__main__':
109:     unittest.main(verbosity=2, module='testpenny')

```