Per-lecture Temporary Student ID - Sprint 1

Equivalence classes

- Acceptable input = list of non-null ASCII strings with maximum length of twenty characters
- Unacceptable input = non-ASCII input, null input, input over twenty in length
- Acceptable output is a dictionary of capitalized five-character alphanumeric strings or appropriate error message.

Acceptable Input Test (First test):

1. Randomly generated ten sets of 25,000 non-collision checked random acceptable strings.
2. For each string in each set, generate a temporary ID and pair them.
3. Return the generated dictionary.
4. Expected output:
    a. A dictionary with input elements as keys and collisionless temporary IDs as values

Unacceptable Input Test (Second test):

1. Randomly generate a set of 25,000 inputs, including both acceptable input and unacceptable input.
2. For each string in each set, check if the input is valid. If the input is invalid, match the input with an error message: “Input is invalid type,” if there is content, or “Input is null” if the input is null.
3. Return the generated dictionary.
4. Expected output:
    a. A dictionary with input elements as keys and appropriate error messages or temporary IDs as values.

Foundation Instructor Course Page - Sprint 1

Equivalence classes:

- Acceptable input: SQLite database with migrations for course models
- Acceptable output: An index page containing all Course instances within the database.
- Unacceptable output: Any unhandled error codes or unexpected redirects

Preliminary Test:

1. Populate an empty SQLite database with 500 generated (but valid) course instances number 0-499.
2. Run the localserver
3. Access <localhost:8000/course/> from a browser.
4. Expected output:
    a. 500 course titles listed vertically in ascending order, with 0 being the first element and 499 being the last element.

Integration Test:

1. Use existing course instances in the database.
2. Run the localserver.
3. Access <localhost:8000/course/> from a browser.
4. Expected output:
    a. All course titles listed in order of creation.

Attendance List Compiler - Sprint 1

Equivalence classes:

- Acceptable input: A null-or-otherwise-arbitrarily-sized set of attendant models with connection field containing any integer.
- Unacceptable input: Incorrect model field type, non-integer connection field data, null connection field data

Full Test:

1. Populate an empty SQLite database with 50,000 randomly named attendants with randomly generated integer connection counts.
2. Randomly swap a random number of attendants with invalid model types, non-integer connection field data, or null connection data.
3. Arbitrarily pick a positive “connection” quota.
4. Call the attendance list compiler function.
5. Print the resulting lists.
6. Expected output:
    a. Present list: Consists of all attendants with connection data greater than the connection quota.
    b. Partial list: Consists of all attendants with non-negative connection data less than the connection quota.
    c. Absent list: Consists of all attendants with negative connection data.
    d. Error list: Consists of all items that did not match valid input.

Course List Page Items Link to Relevant Course Detail Page - Sprint 2

There were no automated tests for this function. Instead, 500 courses were generated. Each course was then populated with 50 lectures and 50 random students.

1. Load the course list page at <localhost:8000/course/>
2. Clicking arbitrary items in the course list redirect to a page containing the lectures for that specific course.

Lecture List Page Allows Lecture Creation - **Sprint 2**

There were no automated tests for this function. Instead, a Course was created in the database. From the creation page, a variety of lecture title inputs were used to test boundary cases and equivalence classes. Invalid tested input included null entry, massive strings, and unicode strings.

Lecture List Page Items Link to Relevant Lecture Info Page - Sprint 2

There were no automated tests for this function. Instead, the database was populated with valid lectures. When the link to the lecture was clicked, the appropriate lecture pages were loaded.

Attendant Network Graph Generation - Sprint 3

Equivalence classes:

- Acceptable input: Any valid graph data, including null and massive graphs.
- Unacceptable input: Any non-graph input

The following code was executed from shell to populate the database and generate a Network. Throughout this process, a variety of visualizations were tested. Unfortunately, NetworkX has limited visualization capabilities, so a simple shell render was agreed upon for Release.

```
def db_fill_for_unit_test(given_course_title):

    course = Course.objects.get(course_title = given_course_title)

    for student in range(40):
         new_id = ''.join(random.choice(string.ascii_uppercase) for x in range(5))
         course.student_set.create(student_id = new_id)

    student_ids = course.student_set.values_list('student_id', flat = True)
    new_lecture = create_lecture(student_ids, "auto-gen lecture title")
    course.lecture_set.add(new_lecture)

    for drain_attendant in new_lecture.attendant_set.all():
         for x in range (random.randint(0, 3)):
                source_attendant = random.choice(new_lecture.attendant_set.all())
                if(source_attendant != drain_attendant):
                       add_edge(new_lecture,
                       drain_attendant.student_id,
                       source_attendant.temp_id)
    return course
```

Acceptable output was an accurate Graph displayed to the Lecture info page. Unacceptable output included:

- URL pattern reverse match errors
- No displayed render
- Incorrect render

Attendance Graph Customization - Sprint 3

Using similar automatic population as above, the various color mappings were tested.
Unacceptable output: Illegible graphs, cluttered graphs for lectures with under 150 students
Acceptable output: Appealing and legible graphs, with acceptable decreases to readability as lecture sizes grew.

Miscellaneous testing

A significant portion of the testing carried out involved automatically populating the database, test stubs, and test spies.

Many code blocks used for testing were run from shell, and most went unsaved as the project progressed.

We spent a lot of time on the inspection phase - far too much. We didn't have a robust testing practice ironed out until the end of Sprint 3, and by then, most of the testing was becoming manual.

That said, we squashed collectively a massive amount of bugs as we learned Django. Some bugs and unsatisfactory items remain, and they are outlined in the respective document.