CSE/ISE 337: Scripting Languages

Stony Brook University

Programming Assignment #4

Fall 2020

Assignment Due: Friday, Dec 4th, 2020, by 11:59 PM (EST)

Learning Outcomes

After completion of this programming project, you should be able:

- Design and implement web scripts using a major web framework
- Extract test cases from specifications
- Develop unit tests in a scripting language

Instructions

- Take time to read the problem specification carefully.
- Zip all files and submit the zipped distribution (**lastname_firstname.zip**). The zip file should contain:
    - A directory **Library** for the Rails application
    - A file **test_impl.py** for the unit tests

Problem 1 (50 points)

Imagine that you have to develop a medium-sized library management system that will be hosted on a web server and can be accessed by library administrators through a browser. You will use Rails to develop such an application. The application will have the following features:

- An *administrator* should be able to add a new book via the application. Hence, the application should have a form with the following fields:
    - Title: string
    - Author: string
    - ISBN: 10-digit unique string
    - Copies: integer; indicates the no. of copies that can be checked out.

  Clicking on the form's save button should add a new book with the form's fields to the database. The form details should not be saved and an error message should be reported for each of the following:
    - Title, author, isbn, or copies are empty
    - ISBN is not a 10-digit number
    - ISBN is not unique
    - Copies in not an integer
- An *administrator* should be able to search a book by
    - Title
    - Author
    - ISBN

  The books found from the search should be displayed in a search results page as a table. Each row in the table should have the book title, author, and ISBN along with 2 links -- to edit the book entry and checkout the book entry. The checkout link should exist only for books with more than 0 copies.
- An *administrator* should be able to edit specific book entries. Hence, the application should provide a page to display the title, author, ISBN, and no. of copies of a book. An administrator should be able to change the fields and click a button to save the changes. If the changes are saved successfully, the administrator should be taken back to the search page; otherwise an error message should be displayed on the same edit page and the administrator should be asked to try again. The error message can be anything.
- An *administrator* can checkout a book by clicking on the checkout link beside the book entry. Clicking the checkout entry will reduce the no. of copies for the book by 1 and will take the administrator back to the search page. If the checkout fails, then the administrator should stay on the search results page while displaying an error message. The error message can be anything.
- A *patron* should be able to search a book by

    - Title
    - Author
    - ISBN

  The books found from the search should be displayed in a search results page as a table. Each row in the table should have the book title, author, and ISBN along with 2 links to add reviews and view reviews.
- Upon clicking the link to add reviews, a *patron* will be able to see a review page. The review page will have a form with two text fields – for name and review and a button to save the entered information.
- Upon clicking the link to view reviews, a patron will be taken to a page where the book title, author, isbn, and the reviews will be displayed.

The application should have a home page with 2 links – one for administrators and another for patrons. The links should be named *Administrator* and *Patron*. When a patron clicks on *Patron*, she is taken to the search book for patrons. When an administrator clicks on *Administrator*, she will be taken to a page with two links – *Add book* and *Search book*.

Name the Rails application **Library**.

Additional Resources
https://api.rubyonrails.org/v5.1.7/classes/ActionView/Helpers.html
https://guides.rubyonrails.org/active_record_querying.html
https://guides.rubyonrails.org/v3.1/routing.html

Problem 2 (50 points)

Upon Joe's request, Jane has implemented a *PhysicalInfo* class (in Python) to store the following information from physical examination of patients.

- Date of examination
- Patient's Name
- Patient's Gender
- Patient's Height
- Patient's Temperature

This class provides the following setter methods.

- *set_date(string)*
- *set_name(string)*
- *set_gender(string)*
- *set_height(int)*
- *set_temperature(float)*

If the values are invalid, then these setter methods will raise a *ValueError* exception. If not, then they will merely return. The specification for valid values are as follows:

- Name should
  - be a string
  - can only contain alphanumeric, ' ', or '-' character
  - contain at least 2 alphanumeric characters
  - contain at least one character from English alphabet
- Gender should
  - be a string
  - be either 'M' or 'F'
- Height should
  - be an integer
  - be a value between 17 and 84 (both inclusive)
  - measured in inches
- Temperature should
  - be a float
  - be a value between 95 and 104 (both inclusive)
  - measured in farenheit
- Date should be valid
  - be a year between 1900 and 2100 (both inclusive)

Your task is to help Joe by writing unit tests to test Jane's implementation.

Create your unit tests in a single file named **test_impl.py**.

You should assume the implementation being tested (Unit Under Test) is available in *impl.py* file. So, in your test file, you should use appropriate import statements (e.g., *import impl*) and name qualifications (e.g., *impl.PhysicalInfo*) to access the implementation. The composability of your tests with our implementation is worth 1 point.

To evaluate your tests, we will execute *nosetests* in a folder with your *test_impl.py* file and various of our *impl.py* files, i.e., one execution of nosetests for each variant of our implementation.

You can use either assert statements or *assertXXX* functions from nose library or unittest module in Python standard library.

Additional Resources
https://nose.readthedocs.io/en/latest/testing_tools.html
https://docs.python.org/3/library/unittest.html#unittest.TestCase