Extra Credit Project (Due-Last day of class)

A philological search system for Latin text

Task Overview

Procure foreign language information from the web and make it available for analysis.

Please read through the whole set of directions before starting. The project will require skills that you have acquired in the previous programming projects. This project should be done individually.

This extra credit project, is divided into two phases.

1. **Phase One**: Data Extraction + Database Population [**10**]
2. **Phase Two**: The rest of the project (including Phase One) [**10**]

Phase One is broken into two parts: **(a)** Data Extraction and **(b)** Database Population. This phase can be very time consuming. You should get started right away. We will check that you have completed all parts of this task, but you will not receive graded feedback on Phase One before you turn in Phase Two.

In Phase Two we add two more parts. Project 1 includes: (a) Data Extraction (b) Database Population (c) Translation Services (d) Search Results ~~and Visualization~~. You will be expected to submit the entire project in a .tar.gz file. This means that you will have a chance to edit, correct, improve or elaborate on the work you have done during Phase One.

For each phase, include a README discussion of how you developed the part and how we can rerun your code. Phase two may be an expanded version of Phase one README. Use the py.test framework to create simple test cases that will allow us to run and verify your code. That is, running `py.test –v test_yourcode.py` or `python3 setup.py test`, for example, will execute test cases for each phase.

Grading

Please read through the Grading Criteria document before beginning. Each day after the due date 10% will be deducted.

Part A: Data Extraction [5pts]

The CS 5970 class has received permission to procure data from the thelatinlibrary.com. This website contains a collection of Latin works from authors in what is called the classical literature. Each student will be responsible for a subset of the collection. Use the google sheet to claim your subset of texts from the collection https://docs.google.com/spreadsheets/d/1UUeCT6Y6oZAEDwD_d8dPZtybSDsBvQbJI8VCYlZPEOY/edit?usp=sharing. Wrire your name next to the collection that you would like to claim.

Each student should sign-up for at least **3** collections. Note that collection sizes vary. Also, no more than two students can choose the same file. Choose quickly! If you wait too long, your favorite collection may be taken.

In your README, you should address how you know that all files have been downloaded accurately. This should also be verified with a python test. Describe your process can be replicated. Again, you should utilize `py.test` to run the code and ensure all files are downloaded. Your code should NOT contain any absolute paths and your code should be runnable on the class gpel environment. If your code requires any external packages ensure they are described and added to the `requirements.txt` file as well as the `README`.

Part B: Database population [5pts]

After the code is extracted, add the extracted text in a structured way into a SQLite database. The database should have the following schema

```
Schema (title|book|language|author|
        dates|chapter|verse|passage|link)
```

Each row of the database describes the lineage and content of the work. Below we describe each attribute, using the Bible as a general example in the table below.

attribute	Example
title	The name of the collection, e.g. “The King James Bible”
book	the title name of a particular book in the collection, e.g. “Genesis”
language	this should be “Latin” in most cases. If you suspect this is not the case with a collection that you are working with, please let me know.
author	any author name associated with a book, e.g. “Moses”. Leave this as null if you are not able to discover it.
dates	the date or date range that the book was written. As with the author information, if this is not available leave it as null.
chapter	also called a section or paragraph, is the delimited portion of text in the book, e.g. ``Genesis 1"
verse	a numeric number, similar to a sentence id, that indexes the sentence within a chapter. The verse could also be a line number in poems. Keep in mind that many of of the collections did not originally contain punctuation or sentence boundaries, these are ones that are generally accepted.
passage	the actual text of the verse
link	the URL of thelatinlibrary.com source containing the passage.

Any of the fields could be null or missing, except for `title` or `link`. You may have to slightly alter the definition for chapter, verse or even dates. Add these assumptions into your README files. No primary key structure is necessary.

Later, in Phase Two, an FTS search system should be performed over the passage attribute. And we will include a translation API in the search.

Part C: Translation Services [5pts]

As English speakers, it is difficult to read Latin. Therefore, we will add a translation service to the requested search queries. The online translation service will read the keywords from the requested search and run them through the translator. Then, use your database to search for the translated terms and return the results like a typical search.

We suggest that you use the free mymemory.translated.net API. Be aware that you have a limit of 1000

searches per day! You are permitted to use another translation API, if you prefer.

Below is an example translation for the word death using translated.net.

```
$ wget -O - 'http://mymemory.translated.net/api/get?q=death&langpair=en|it' | python -r
{
    "matches": [
        {
            "create-date": "2013-08-11 15:03:21",
            "created-by": "Matecat",
            "id": "439969461",
            "last-update-date": "2013-08-11 15:03:21",
            "last-updated-by": "Matecat",
            "match": 1,
            "quality": "80",
            "reference": "",
            "segment": "death",
            "subject": "All",
            "tm_properties": null,
            "translation": "morte",
            "usage-count": 45
        },
        {
            "create-date": "2015-09-22 11:06:23",
            "created-by": "Matecat",
            "id": "475900872",
            "last-update-date": "2015-09-22 11:06:23",
            "last-updated-by": "Matecat",
            "match": 0.96,
            "quality": "74",
            "reference": "",
            "segment": "Death",
            "subject": "All",
            "tm_properties": "",
            "translation": "Morti",
            "usage-count": 1
        }
    ],
    "responderId": "235",
    "responseData": {
        "match": 1,
        "translatedText": "morte"
    },
    "responseDetails": "",
    "responseStatus": 200
}
```

Part D: Search Results and Visualization [5pts]

You will create a search interface that allows a user to explore the data set. You may either create a website or use the command prompt to facilitate interaction with users. The interface should support the following actions.

1. **Latin term search**. Given a Latin search term, use the FTS index to return the snippet and link to the original link where that term appears.

2. **English term search**. Given an English search, use the translation service function you created in Phase II Part A to translate the search term to Latin and search that term over the FTS system. Returning the translated word, snippets, and links to the original test location.

3. **Usage chart** (Extra Credit). Given a search term, create a bar chart for each document containing the term. A document can be a collection or a section, the latter if you only have a few collections assigned. Search terms should be either Latin or English. If English is selected, the translation service should be used. The y-axis is the number of times the term has appeared in a document (or chapter) and the x-axis should be sorted by the height.

Submission

The output of Part A & B can be combined into one README document, code package and inserted into one .tar.gz file. Also include the pre-populated database in the compressed file.

Package your code using the setup.py and directory structure discussed earlier in the course. You should additionally add tests to this project submission.

```
from setuptools import setup, find_packages

setup(
    name='extracredit',
    version='1.0',
    author='You Name',
    authour_email='your ou email',
    packages=find_packages(exclude=('tests', 'docs')),
    setup_requires=['pytest-runner'],
    tests_require=['pytest']
)
```

In the tests folder, add a set of files that test the different features of your code. Test do not have to be too creative but they should show that your code works as expected. There are several testing frameworks for

python, for this project use the py.test framework. For questions use the message board and see the pytest documentation for more examples (http://doc.pytest.org/en/latest/assert.html).

```
extracredit/
        extracredit/
            __init__.py
            extracredit.py
        tests/
            test_extraction.py
            test_population.py
            ...
        docs/
        README
        requirements.txt
        setup.cfg
        setup.py
```

Note, the `setup.cfg` file should have at least the following text inside:

```
[aliases]
test=pytest
```

Typing `python3 setup.py test` should execute your tests using the pytest-runner.

Upload a .tar.gz file containing the following items:

- README description
- Extraction code package
- Pre-populated database

Please do not include your env or venv folder contents. It can be easily recreated and only adds to the size of your compressed file. You code may be selected added to an open source project. If you have any objections to this please let the professor know.