Objective:

The goal of this programming exercise is to demonstrate your ability to design a solution to a problem and implement this solution in **Python** using software engineering best practices.

The specific task will be to create a "related courses" application which, given a course description will suggest similar courses offered at USC. To build this application, you will crawl course descriptions from the USC Schedule of Classes, use token-based sets, or word or document embeddings to create a high-dimensional representation of each course, then apply a similarity algorithm to quickly find related courses.

There are three different tracks to use these outputs: a backend track for storing and quickly querying results, a data science track to analyze the quality of the similarity algorithms, and a frontend track to visualize the outputs.

Steps:

1. Create a **Github** repo and **Python 3** environment for this project and start a requirements.txt file to capture the packages required to run your code.

2. **Scrape** the course number, course name, course description and prerequisites of each course offered by the Viterbi School of Engineering and the Keck School of Medicine on the USC Schedule of Courses (https://classes.usc.edu/term-20193/). Note that courses are offered by departments within these schools.
   Example Output:

```
1  {'id': 'CSCI-670',
    'name': 'CSCI 670: Advanced Analysis of Algorithms (4.0 units)',
    'url': 'https://classes.usc.edu/term-20193/course/csci-670/',
    'desc': None,
    'prereqs': [
      'https://classes.usc.edu/term-20193/course/csci-570/' ] }
```

```
2  {'id': 'CSCI-675',
    'name': 'CSCI 675: Convex and Combinatorial Optimization (4.0 units)',
    'url': 'https://classes.usc.edu/term-20193/course/csci-675/',
    'desc': 'Topics include: Convex sets and functions; convex optimization
   problems; geometric and Lagrangian duality; simplex algorithm; ellipsoid
   algorithm and its implications; matroid theory; submodular optimization.',
    'prereqs': [
      'https://classes.usc.edu/term-20193/course/csci-570/',
      'https://classes.usc.edu/term-20193/course/csci-670/' ]}
```

Suggested approach is to use the BeautifulSoup Python library.

(continued on next page)

3. **Pre-process** the course descriptions as necessary. Some ideas for preprocessing include lowercasing, removing very short descriptions, or stop word removal.

4. Separate your dataset into **training and test data**, ideally using cross-validation (https://scikit-learn.org/stable/modules/cross_validation.html) folds

5. **Build a system** for comparing course descriptions using different algorithms. You should implement a function where the **input is a string containing a phrase** and the **output is the ten most related courses**, **each with a score**. To compute the course similarity, you should implement several algorithms (best to implement one or two, complete step 6, then come back to implement more)
    a. One very simple approach would be to compute the Jaccard similarity of two course descriptions
    b. Another simple approach would be to compute the Levenshtein similarity of the course name (see https://pypi.org/project/python-Levenshtein/)
    c. A better approach would be to use the Gensim library (https://github.com/RaRe-Technologies/gensim) to create a vector embedding of words in course descriptions and compute the cosine similarity of the average vector for each course
    d. Gensim also supports creating document-level vectors for each course description (see this tutorial [https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/doc2vec-lee.ipynb] if you're really lost)
    e. Gensim embeddings can be enhanced using Google News vectors [https://code.google.com/archive/p/word2vec/] or pre-trained GloVe embeddings [https://radimrehurek.com/gensim/models/keyedvectors.html] since you are using very little training data.

(continued on next page)

6. **BE track:** Create a fast lookup system for course similarities.
    a. Build a database to store course information as well as course vectors
    b. Create a fast index using a system like annoy to support fast pairwise similarity queries across all courses
    c. Develop a documented API and accompanying web service to create a public resource for users to query for similar courses or provide an input set of textual keywords and find related courses

7. **DS Track:** Perform an experiment on your cross-validated test set for each of the 5 approaches to:
    a. Determine how often a prerequisite for a course appears in the top-3 most related courses
    b. Determine what percentage of the top-10 similar courses are from the same school (Viterbi or Keck)
    c. Build a system that finds the closest analogue class at UCLA for each USC course (from UCLA registrar https://www.registrar.ucla.edu/Academics/Course-Descriptions/Course-Details?SA=COM+SCI&funsel=3)

8. **FE Track:** Build a user interface that allows a user to navigate the course similarity network as a graph with nodes as courses and edges as similarities between these course. Allow the user to input keywords and re-color the nodes based on the relevancy of the course re-using your similarity function.

9. Create a **summary of your results** and provide high-level documentation of your code.