1) An overview of the function of the code (i.e., what it does and what it can be used for).

First, our implementation has a general crawler that can automatically scrape lecture slides in different courses from the UIUC CS courses platform (https://courses.grainger.illinois.edu). We were scraping all links from course websites and filtering them by finding potential links that could lead us to a lecture slide. Then, we split the slides crawled from those websites and split them into 1 page pdfs stored in folders corresponding to course names. After that, we implement a Jaccard similarity function to select related slides of each page, record the result in a csv file. Then based on the relationships of the slides, we render the final result on the website.

2) Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.

In crawling/scraper.py & crawling/utils.py:
For the crawler, we choose the https://courses.grainger.illinois.edu as our start page to scrape our courses' url that are listed on this website. For this final project, we are only scraping courses that have 'CS' as a prefix. After we have all the course urls, we will iterate all of them and step into every course website to find where could the lecture slides/notes be located using keywords(['slides', 'slide', 'lecture','lectures','note','notes','resources','resource']). Then we start to scrape all the tags <a href=true and determine whether the links are a potential lecture slide by simply checking whether the url ends with .pdf and .pptx (since we have already filtered those links with the keyword, this constraint could very likely be the lecture slides that we want to scrape). And we parse those links to a formal format and download them into the corresponding folder. All the tasks are automated and we can expect to see a system with a large scale of lecture slides using this crawler.

In pdf.js/static/getRelatedFiles.py: We first extract text in pdf pages, tokenize the content and select key words after eliminating punctuations, stopwords. Then compare Jaccard similarities of the keywords from one pdf with all other pdfs, when the similarity is greater than 0.3, we mark these two pdfs as related. Furthermore, we limit the size of the related slides as 12, so that when rendering them on the website, the list won't be too long. Then, we record them in ranking.csv after changing the format according to the original format in the platform EducationalWeb; and record all slide names in slide_names.txt.

In pdf.js/parsePDF.py: We put the results of the scraped folders under raw_slides. We filtered out those corrupted slides.Then we first followed the naming convention to change the course and slides name, then split each of the slides each long pdf into a folder containing single slides for rendering.

In model.py: We modified the model.py file so that other than cs410, other courses can also be rendered in the webpage. Moreover, we changed the path `related_slides_path`, `slides_path`,so that the related files are derived from our own algorithm and include all slides from different courses.

3) Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run a software, whichever is applicable.

The following instructions have been tested with Python2.7 on Linux and MacOS

1. You should have ElasticSearch installed and running -- https://www.elastic.co/guide/en/elasticsearch/reference/current/targz.html
2. Create the index in ElasticSearch by running python create_es_index.py from EducationalWeb/
3. Download tfidf_outputs.zip from here -- https://drive.google.com/file/d/19ia7CqaHnW3KKxASbnfs2clqRIgdTFiw/view?usp=sharing
   Unzip the file and place the folder under EducationalWeb/static

   =========== NEWLY ADDED FEATURES =========
4. Run python scraper.py from CourseProject/crawling/ to scrape lecture slides from the website
5. Then run python parsePDF under EducationalWeb/pdf.js to normalize the slides name and save one PDF into a folder with single slides.
6. Run python getRelatedFiles.py in EducationalWeb/pdf.js/static to get every single slide's related slides with ranking scores
7. From EducationalWeb/pdf.js/build/generic/web, run the following command: gulp server
8. In another terminal window, run python app.py from EducationalWeb/
9. The site should be available at http://localhost:8096/

4) Brief description of contribution of each team member in case of a multi-person team.

We reviewed the original code on the EducationalWeb, analyzed the structures and functions in their repository, then ran code based on the instructions together. Then we had meetings discussing the structures and functions that we aim to implement. Yusen is mainly responsible for implementing the crawling part, Tianli normalizes the slides name and saves one PDF into a folder with single slides, Yimeng takes charge of implementing the 'related slides' algorithm. After one finished his/her part, other team members did code review and added comments for improvements. We wrote documentations and recorded demo videos together.