Documentation

Installation

```
pip install requirements.txt
```

Files

`app.py` : executes scraper.
`document_scraper.py` : scraper class.
`utilities.py` : functions that are used in scraper class.

Execution

```
./app.py ticker
```

- String: `ticker` : company ticker to scrape.

Result

`{ticker}_recent_13F.tsv` : tsv file containing information from the ticker's most recent 13F xml info table.

Design Decisions

Dealing with Different Info Table Formats

By looking at the html of each info table, I noticed that they all shared the same field names. However, whether or not the corresponding xml element was in the the xml element tree depended on if the data existed in that cell. Therefore, by searching for each possible element name and saving its value if it exists or an empty string if it doesn't, the info table scrape method can remain DRY and used for all info tables.

Technologies - BeautifulSoup vs. Scrapy vs. Selenium.

The goal of this crawler is given one ticker, save the information from its most recent 13F report. Looking at the flow of the website, getting to the required information is as follows:

go to tickers' document list page ➜ find most recent 13F document link ➜ go to document page ➜ find information table xml link ➜ go to xml link ➜ scrape information.

Each request would have to finish completely before scraping can be done to get to the

next step. Because this must be done in serial, there is no need to leverage any sort of asynchronousity, which Scrapy's event driven networking provides. Furthermore, there was no dynamic content on any of the pages visited which means there is no need to use Selenium, which can extract information from dynamically generated data.

Implementing either Scrapy or Selenium would be overkill for this use case, and to keep it simple stupid, BeautifulSoup would be the simplest and quickest approach.

Limitations

Dealing with Different Info Table Formats

Because I hardcoded the xml element names to search for inside of info tables, the scraper could be less modular.

Serial Implementation

The current web scraper accepts one ticker at a time. If this use case was to change to accept a list of tickers, performance would not be optimal because the program would wait until each request finishes. There would be room to make this more efficient by implementing Scrapy to traverse the flow of each ticker asynchronously.
