# Task 3: Report

## 1. In Task 1, What did you set things in settings.py to achieve the politeness of your crawler?

Defining my user agent including me app name and my email contact, telling the data provider what is the purpose of my crawl and who am I, how to reach me.

In [ ]:
```
USER_AGENT = 'MovieCrawler (ruijiera@usc.edu)'
```

Normally, this need to be turned to true to obey the rules set by the site. However, in this hw it is necessary to turn off.

In [ ]:
```
ROBOTSTXT_OBEY = False
```

Configuring the number of concurrent requests limits the number of requests my crawler can make at any second. This forbids the crawler to give too much pressure to the site every second.

In [ ]:
```
CONCURRENT_REQUESTS = 5
```

Configuring the download delay the the same website slows down the fetch, but significantly lowers the pressure given by my crawler to the site.

In [ ]:
```
DOWNLOAD_DELAY = 3
```

Autothrottle is a pluggin by scrapy that controls the download delay automatically according to server response latency.

In [ ]:
```
AUTOTHROTTLE_ENABLED = True
```

Http Cache is another function in scrapy that temporarily stores the fetched website. In this way, the crawler wont need to access the same page again in a period of time, releasing pressure from the site.

In [ ]:
```
HTTPCACHE_ENABLED = True
```

## 2. What lexical rules did you use to extract information in Task 2? Why?

### Lexical

**Education**: I set 2 lexical patterns, matching the phrase of "was educated at {institute}" and "attend {institute}". To match with insitution names, I used a combination of IS_TITLE and "of" with OP tags. This is because inistitutions names are all titles except for "of" in between, and this lexical pattern can avoid matches to be made half way.

In [ ]:
```
patterns = [
    [{'LOWER': 'was'}, {'LOWER':{'REGEX': '^educat\w*$'}}, {'LOWER': 'at'}, {'IS_TITLE
    [{'LOWER':{'REGEX': '^attend\w*$'}}, {'IS_TITLE': True, 'OP': "+"}, {'LOWER': 'of
]
```

**Birthplace**: I set 1 lexcal pattern on the phrase "born in {location}". Location names like towns and cities are often represented in Title texts, therefore using IS_TITLE.

In [ ]:
```
patterns = [
    [{'LOWER':'born'}, {'LOWER': 'in'}, {'IS_TITLE': True, 'OP': "+"}],
]
```

**Movies**: I set 1 lexical pattern because movie names are very formatted in biographies. I observed that there is always a year in brackets following the movie names, where the movie names are often in title texts. Therefore, I use IS_TITLE to match for the movie name, regex [0-9]{4} to match for the year following the name. Some movie names are in '', so this is added too. OP * means that there can be 0 to any number of instances.

In [ ]:
```
patterns = [
    [{'TEXT': '"', 'OP': '*'}, {'TEXT': "'", 'OP': '*'}, {'IS_TITLE': True, 'OP': "+"
]
```

**Awards**: I set 2 lexical patterns but they are the same. I observes that award names sometimes follows "the", so i add an option "then" at the beginning. Then, there is always some title text followed by the word "Award", so I make a OP:+ on title texts before "award". I also considered instances when "Something and Something Award" and "Something's Award" appears.

In [ ]:
```
patterns = [
    [{'LOWER': 'the', 'OP': '*'}, {'IS_TITLE': True, 'OP': "*"}, {'LOWER': 'and', 'OP
    [{'LOWER': 'the', 'OP': '*'}, {'IS_TITLE': True, 'OP': "*"}, {'LOWER': 'and', 'OP
]
```

**Actors**: Actor names are always in title texts. To specify the person is an actor, the biography always needs to mention the word "actor" somewhere around the person's name. So I made the word actor in front of the person's name.

In [ ]:
```
patterns = [
    [{'LOWER': 'actor'}, {'IS_TITLE': True, 'OP': "+"}]
]
```

Syntactic

**Education**: I set 3 syntactic patterns. Instead of matching with IS_TITLE, I choose to check the phrase's entity type and set it to organization.

In [ ]:
```
patterns = [
    [{'LEMMA': 'educate'}, {'LOWER': 'at'}, {'ENT_TYPE': 'ORG', 'OP': '+'}],
    [{'LEMMA': 'attend'}, {'ENT_TYPE': 'ORG', 'OP': '+'}],
    [{'LEMMA': 'study'}, {'LOWER': 'at'}, {'ENT_TYPE': 'ORG', 'OP': '+'}]
]
```

**Birthplace**: I set 1 syntactic pattern on the phrase "born in {location}". Using Lemma on born matches with any tense of the verb, POS:ADP matches any proposition follows born, and the location is matched using entity type of location.

In [ ]:
```
patterns = [
    [{'LEMMA': 'born'}, {'POS': 'ADP'}, {'ENT_TYPE': 'LOC', 'OP': '+'}],
]
```

**Movies**: The pattern is pretty clear just like in the lexical, instead i changed the regex to entity type of date, which should work just the same.

In [ ]:

```
patterns = [
    [{'TEXT': '"', 'OP': '*'}, {'TEXT': "'", 'OP': '*'}, {'IS_TITLE': True, 'OP': "+"
]
```

**Awards**: for this I used Dependency Matcher. I set the anchor to the verb "win", and set the subject dependent on win as a entity type of organization.

In [ ]:

```
patterns = [
    #[{'LEMMA': 'win'}, {'ENT_TYPE': 'ORG', 'OP': '*'}, {'LEMMA': 'Award'}, {'ENT_TYPE
    [
        {
            "RIGHT_ID": "win",
            "RIGHT_ATTRS": {"LEMMA": {"IN": ["win", "won"]}}
        },
        {
            "LEFT_ID": "win",
            "REL_OP": ">",
            "RIGHT_ID": "subject",
            "RIGHT_ATTRS": {'ENT_TYPE': 'ORG'}
        }
    ]
]
```

**Actors**: Similar as the lexical, but changed the title text to entity type of person with multiple words.

In [ ]:

```
patterns = [
    [{'LOWER': 'actor'}, {'ENT_TYPE': "PERSON", 'OP': "+"}]
]
```