

Design Document (IMDB Search)

By Kartik Chhapia

Coding Language

The code is implemented in the Python 3 programming language and Request and BeautifulSoup were the 2 libraries used in developing this code

How to Run

```
python3 search.py "search_term"
```

Example: python3 search.py "steven spielberg"

Will search for movies in which steven spielberg appears

Scraping IMDB

Using the requests library in Python, I have scraped the imdb top 1000 page

(https://www.imdb.com/search/title?groups=top_1000&view=simple&sort=user_rating.desc).

Using the BeautifulSoup Python library, I have extracted the url of the individual 1000 movies. Iteratively, using Requests and BeautifulSoup, I visited each of the 1000 movies and pulled information about various aspects of the movies. All the data was stored in two different python dictionaries. The keys of the first dictionary were the unique integer or the "rank" of the movie which ranged from 0 to 999. The value of the dictionary were the name of the movie stored as a string. Basically in this dictionary we are mapping every movie to a unique integer. The keys of the second dictionary were the unique integers that referred to a movie and the value of this dictionary were a dictionary of various aspects of the movie such as "actors", "directors", "writer", "genre".

The dictionaries are pickled and stored on the disk. This is possible given the size of our problem (1000 movies)

All the above steps were done with the script "*scrape_imdb.py*"

Indexing

The dictionaries created above contain all the information about the movie. But search is inefficient. This is because, if we want to search for "spielberg", we have to iteratively traverse the entire dictionary and search for "spieberg" in all aspects of the movies. This naively has a time complexity of $O(n)$ where n is the number of movies. Instead if we create a new dictionary that has indexed all the various aspects of the movies, then we can improve our search time. For instance, if we make "spielberg" as one of the keys in the new indexed dictionary and the value of this key will be a list of associated movies then we can improve the search time to $O(1)$. Thus, we build on this idea to index the original dictionary that contains the information about all the movies. Before we make an entity or an aspect of a

movie as a key in the indexed dictionary, we lowercase that entity. The indexed dictionary is indexed or created in the following way:

1. Every actor that appears in the original unindexed dictionary becomes a key in the new indexed dictionary. If the name of the actor contains a space, we split the actors' name and store the individual names as keys in the dictionary as well. Thus "steven spielberg", "steven" and "spielberg" become keys in the dictionary. This facilitates fast lookups in case the user searches by the first or last name of the actor.
2. Similarly to how we index actors, we also index directors, creators, writers and genres. (We do not split genre at a whitespace since it is a single word)
3. We also store the movie names as keys in the indexed dictionary and if the movie contains multiple words, we store each of those individual words as keys as well. For example "lord of the rings", "lord", "of", "the", "rings" will all be stored as keys in the indexed dictionary

Note: All aspects of the movie (names of actors, writers, directors, genres) are changed to lowercase before making them the keys of the indexed dictionary.

The indexed dictionary is pickled and stored on the disk. This is possible given the size of our problem (1000 movies)

Once the indexed dictionary is created, we can query it to perform fast lookups. The script `"create_indexes.py"` creates the indexes

Querying

A user enters a query and expects a list of movies associated with the query. We first convert the query to lowercase. Subsequently we follow a 2 step approach:

We lookup the query in our indexed dictionary.

1. If the query exists as one of the keys in our dictionary, we returned the list of movies that are associated with the query. This is basically the value of that query key in the indexed dictionary.
2. If the query does not exist in the dictionary, we break up the query into individual words. We search for all the individual words in the indexed dictionary. When we perform this search we follow a 2 step approach. If the word exists as an index in the dictionary we return a list of movies associated with that word. If the word does not exist in the dictionary, then we iteratively traverse the entire dictionary and see if the word exists as a substring to any of the indexes and return a list of the movies associated with that index. We do that for all the search terms and take an intersection of the movies.

For example: if the search term is "Lord Ring" we first convert the query to lowercase. The query becomes "lord ring". Now since "lord ring" does not exist as a key in our indexed dictionary (does not exactly match any key in the dictionary), we search for the word "lord" and search for the word "ring" and find the list of movies that are common to both the words "lord" and "ring". In this case "lord" and "ring" were present as keys in the indexed dictionary and hence search was fast.

However, if we searched for "lord Rin" then the following scenario would occur. Since "lord rin" does not exist as a key in our indexed dictionary, we would start searching for the individual words. We would first lookup movies for the word "lord". Since "lord" exists as a key in our indexed dictionary, our search would be fast and we would get a list of movies associated with the word "lord". Next we would search for the word "rin" and since "rin" does not exist as a key in our dictionary, we would not get any movie associated with it. Now we would iterate over all the keys in our indexed dictionary and check if "rin" is a substring of any of those keys. Since "rin" is a substring of "ring", we would get a list of all movies that are associated with the word "ring". Taking an intersection of the 2 lists, we would get all the 3 Lord of the Rings movies!

The script "search.py" creates the indexes

Everytime the user performs a query, the indexed dictionary which is stored in the pickle format, is read from the disk and is then used to answer the query.

How to run : `python3 search.py "search_term"` will query the indexed dictionary and return a list of movies associated with "search_term"

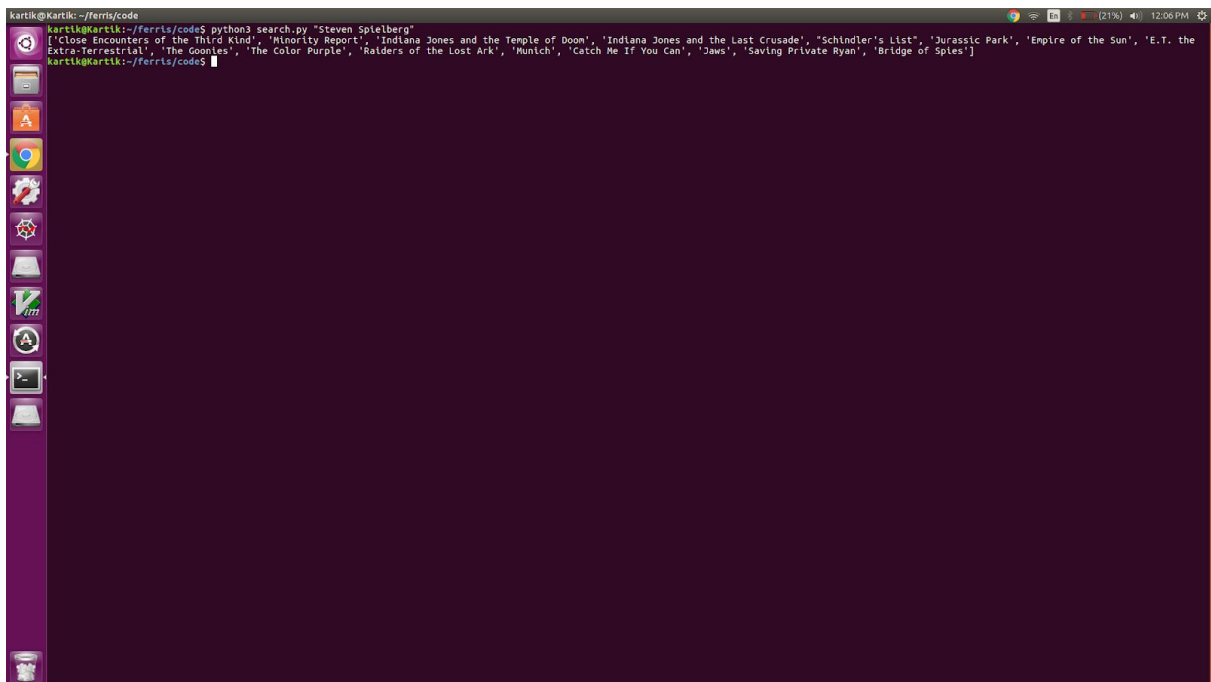
Look below for some example outputs

Some example outputs

1. `python3 search.py "Steven Spielberg"` returns the following

This query returns all the movies in which Steven Spielberg appears as the director or writer.

['Close Encounters of the Third Kind', 'Minority Report', 'Indiana Jones and the Temple of Doom', 'Indiana Jones and the Last Crusade', 'Schindler's List', 'Jurassic Park', 'Empire of the Sun', 'E.T. the Extra-Terrestrial', 'The Goonies', 'The Color Purple', 'Raiders of the Lost Ark', 'Munich', 'Catch Me If You Can', 'Jaws', 'Saving Private Ryan', 'Bridge of Spies']



```
kartik@Kartik: ~/Ferris/code
kartik@Kartik:~/Ferris/code$ python3 search.py "Steven Spielberg"
['Close Encounters of the Third Kind', 'Minority Report', 'Indiana Jones and the Temple of Doom', 'Indiana Jones and the Last Crusade', 'Schindler's List', 'Jurassic Park', 'Empire of the Sun', 'E.T. the Extra-Terrestrial', 'The Goonies', 'The Color Purple', 'Raiders of the Lost Ark', 'Munich', 'Catch Me If You Can', 'Jaws', 'Saving Private Ryan', 'Bridge of Spies']
kartik@Kartik:~/Ferris/code$
```

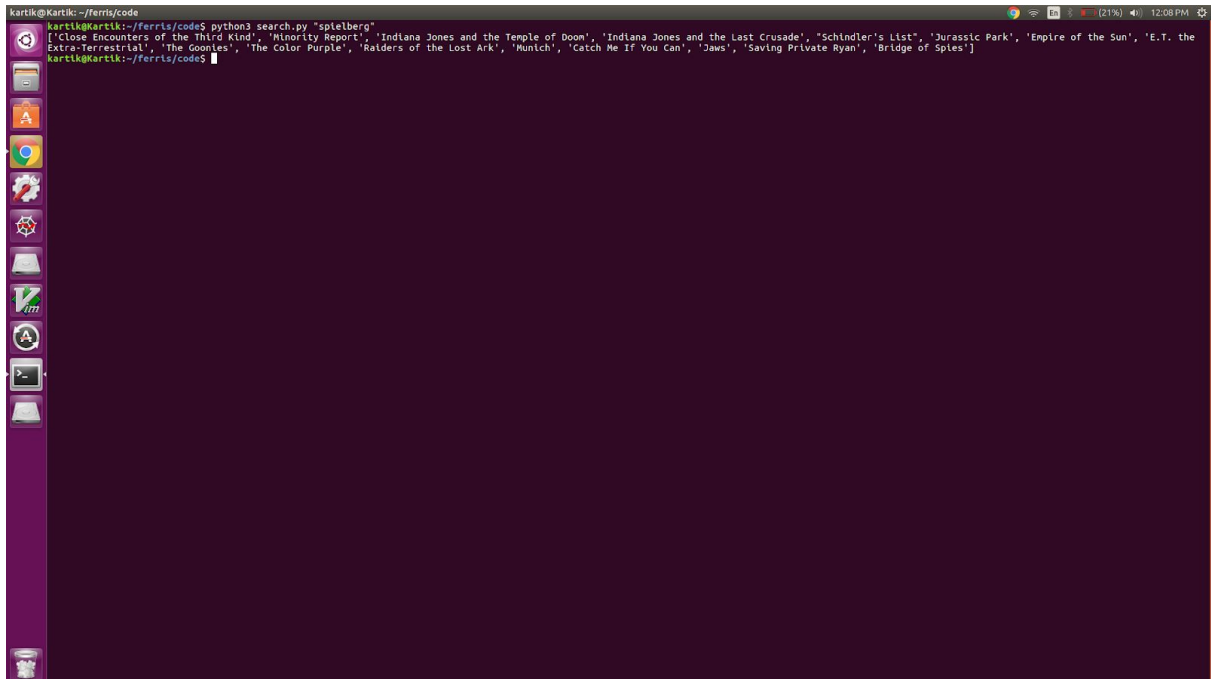
The image shows a terminal window on a Linux desktop. The terminal title is 'kartik@Kartik: ~/Ferris/code'. The command 'python3 search.py "Steven Spielberg"' has been executed, and the output is a list of 15 movie titles enclosed in single quotes and brackets. The desktop background is dark purple, and a vertical dock with various application icons is visible on the left side of the terminal window.

2. Searching for “spielberg”

`python3 search.py “spielberg”`

This returns all the movies in which spielberg appears as a director, writer, creator or actor.

`['Close Encounters of the Third Kind', 'Minority Report', 'Indiana Jones and the Temple of Doom', 'Indiana Jones and the Last Crusade', 'Schindler's List', 'Jurassic Park', 'Empire of the Sun', 'E.T. the Extra-Terrestrial', 'The Goonies', 'The Color Purple', 'Raiders of the Lost Ark', 'Munich', 'Catch Me If You Can', 'Jaws', 'Saving Private Ryan', 'Bridge of Spies']`

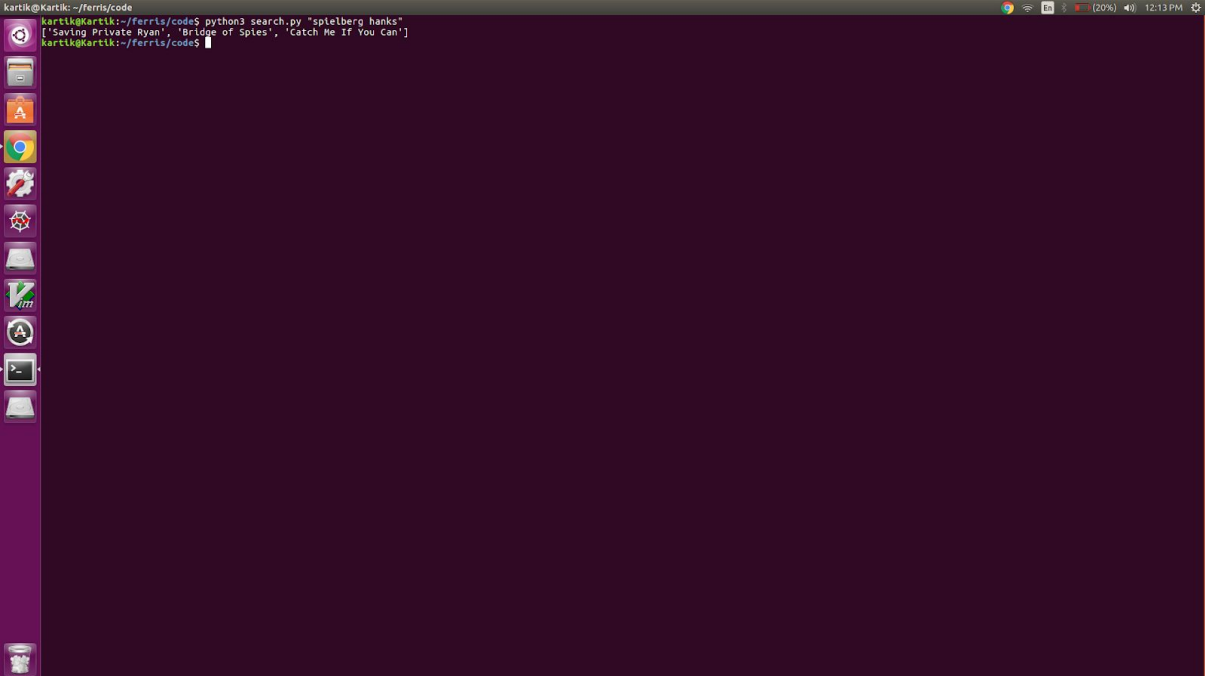


```
kartik@Kartik: ~/ferris/code
kartik@Kartik:~/ferris/code$ python3 search.py "spielberg"
['Close Encounters of the Third Kind', 'Minority Report', 'Indiana Jones and the Temple of Doom', 'Indiana Jones and the Last Crusade', 'Schindler's List', 'Jurassic Park', 'Empire of the Sun', 'E.T. the Extra-Terrestrial', 'The Goonies', 'The Color Purple', 'Raiders of the Lost Ark', 'Munich', 'Catch Me If You Can', 'Jaws', 'Saving Private Ryan', 'Bridge of Spies']
kartik@Kartik:~/ferris/code$
```

3. Searching for “spielberg hanks”
python3 search.py “spielberg hanks”

This returns all movies in which spielberg and hanks appear as a director

Output : ['Bridge of Spies', 'Catch Me If You Can', 'Saving Private Ryan']



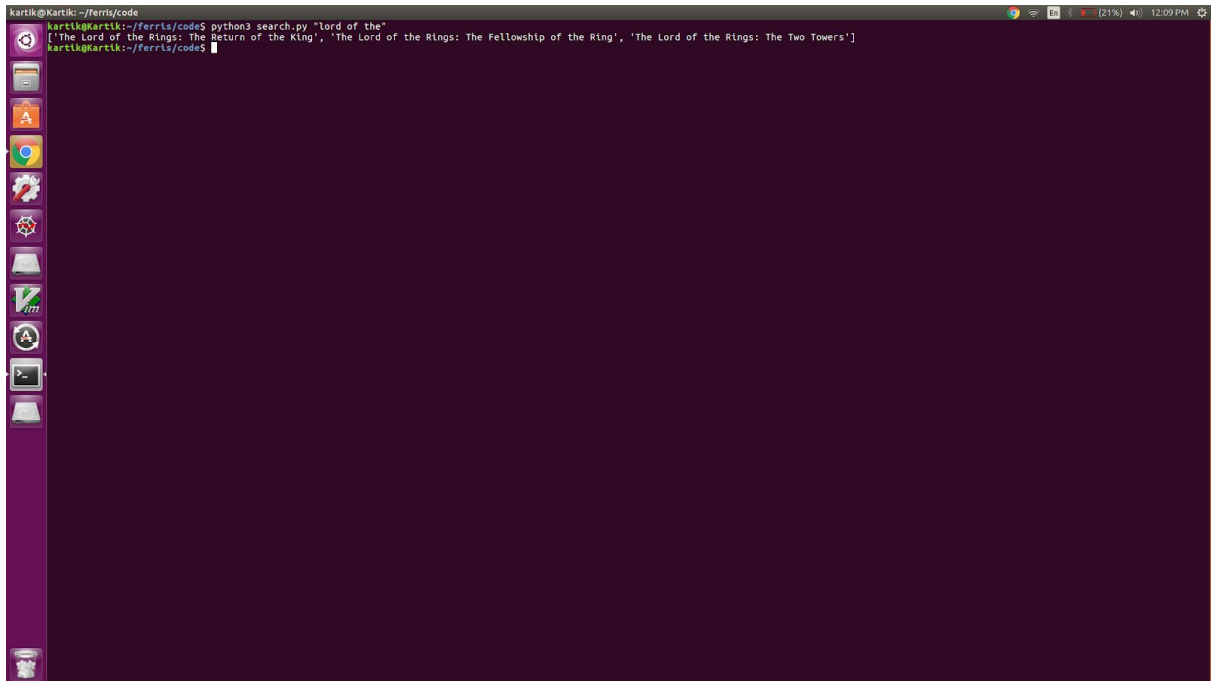
```
kartik@Kartik:~/ferris/code$ python3 search.py "spielberg hanks"
['Saving Private Ryan', 'Bridge of Spies', 'Catch Me If You Can']
kartik@Kartik:~/ferris/code$
```

The image shows a terminal window on a Linux desktop. The terminal title bar reads 'kartik@Kartik:~/ferris/code'. The prompt is 'kartik@Kartik:~/ferris/code\$'. The command entered is 'python3 search.py "spielberg hanks"'. The output is '["Saving Private Ryan", "Bridge of Spies", "Catch Me If You Can"]'. The prompt then changes to 'kartik@Kartik:~/ferris/code\$'. The desktop background is dark purple, and the left sidebar contains various application icons.

4. Searching for “lord of the”
python3 search.py “lord of the”

This query returns all movies in which the words “lord”, “of” and “the” appear

Output : ['The Lord of the Rings: The Return of the King', 'The Lord of the Rings: The Fellowship of the Ring', 'The Lord of the Rings: The Two Towers']

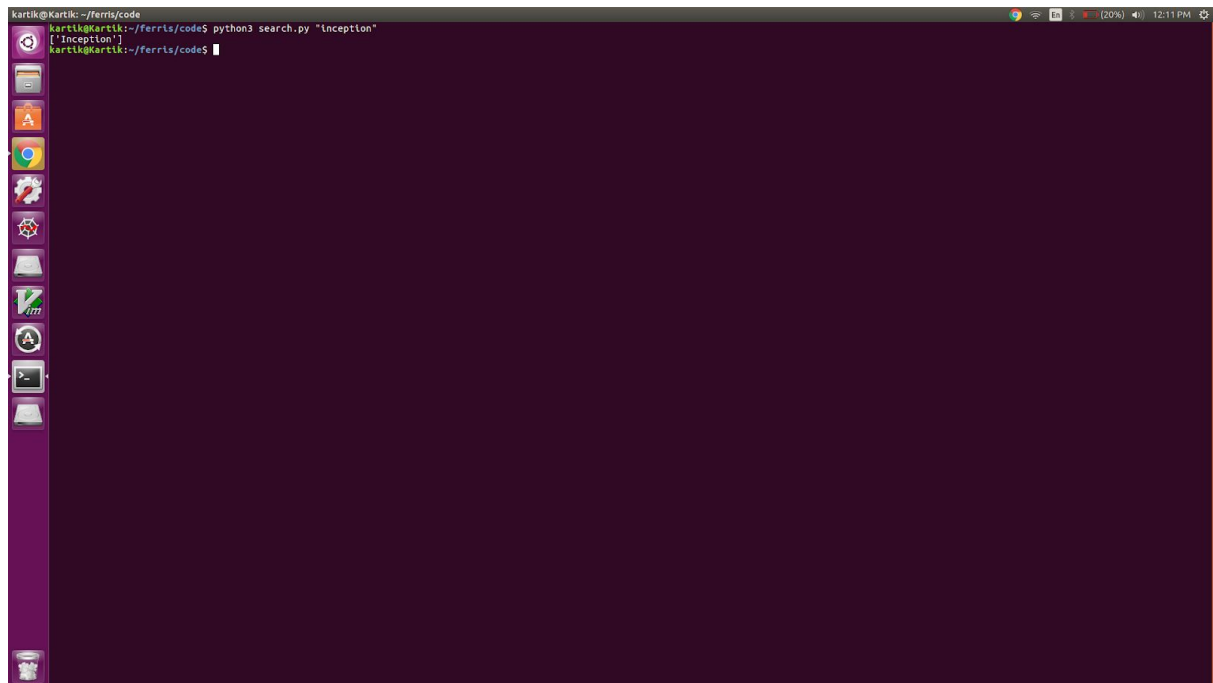
A screenshot of a Linux terminal window. The terminal has a dark purple background and a light purple sidebar on the left containing various application icons. The terminal text shows the user 'kartik' at host 'Kartik' in the directory '~/ferris/code'. They have executed the command 'python3 search.py "lord of the"'. The output of the script is displayed on the next line: ['The Lord of the Rings: The Return of the King', 'The Lord of the Rings: The Fellowship of the Ring', 'The Lord of the Rings: The Two Towers']. The prompt character is a vertical bar '|'. The top of the terminal window shows system status icons including network, battery (21%), and time (12:09 PM).

```
kartik@Kartik: ~/ferris/code
kartik@Kartik:~/ferris/code$ python3 search.py "lord of the"
['The Lord of the Rings: The Return of the King', 'The Lord of the Rings: The Fellowship of the Ring', 'The Lord of the Rings: The Two Towers']
kartik@Kartik:~/ferris/code$
```

5. Searching for “Inception”
python3 search.py “Inception”

This query returns the movie Inception

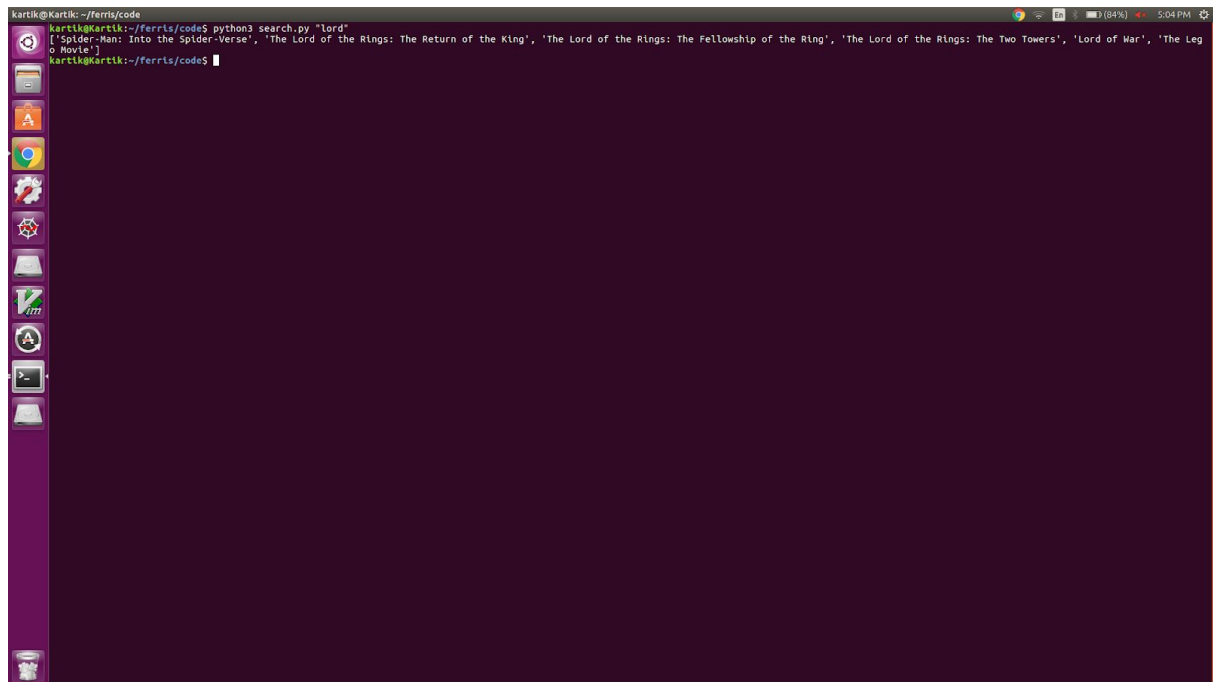
['Inception']

A screenshot of a Linux terminal window with a dark purple background. The window title is 'kartik@Kartik:~/ferris/code'. The terminal shows the command 'python3 search.py "Inception"' being executed, followed by the output '["Inception"]'. The prompt 'kartik@Kartik:~/ferris/code\$' is visible on the next line. On the left side of the terminal, there is a vertical dock containing several application icons: a gear, a folder, a terminal, a web browser, a file manager, a music player, a video player, a game, and a system monitor. The top of the terminal window shows system status icons including network, battery (20%), and time (12:11 PM).

6. Searching for lord
python3 search.py "lord"

This query searches for movies in which the word lord appears in the title and also searches for movies in which Phil Lord appears as the director.

['Spider-Man: Into the Spider-Verse', 'The Lord of the Rings: The Return of the King', 'The Lord of the Rings: The Fellowship of the Ring', 'The Lord of the Rings: The Two Towers', 'Lord of War', 'The Lego Movie']

A screenshot of a Linux terminal window. The window title is 'kartik@Kartik: ~/feris/code'. The prompt is 'kartik@Kartik:~/feris/code\$'. The command entered is 'python3 search.py "lord"'. The output is a list of movie titles: ['Spider-Man: Into the Spider-Verse', 'The Lord of the Rings: The Return of the King', 'The Lord of the Rings: The Fellowship of the Ring', 'The Lord of the Rings: The Two Towers', 'Lord of War', 'The Lego Movie']. The prompt is now 'kartik@Kartik:~/feris/code\$'. The terminal has a dark purple background and a vertical sidebar on the left with various application icons.

Assumptions

I have made the following simplifying assumptions. I have assumed that the user will search for movies only by a few aspects such as the name of the movie title, names of actors, directors, writers and creators or the user would search for the movie by its genre. In practice there are various other aspects about a movie that a user could potentially search by. Another assumption that I have made is that the user will always enter the correct spelling while searching.

Improvements

1. *Misspelt words*: Currently, if the user enters a misspelt word (for example dord instead of lord) we will not be returning any movies currently because such an index does not exist in the dictionary. However by using different algorithms, we can find a list of closest search terms to the misspelt query and ask the user if he or she wanted

to search for any of these terms instead. Thus we can add support for search terms that are misspelt.

2. *Search for different aspects* : Currently, the only aspects a user can search for about a movie are names of actors, directors, writers and creators. A user can also search for a movie by its genre or title name. However, there is no way a user can search for a list of movies by star ratings, or best rated movies in a particular year or movies that are similar to a movie. Given more time these are the features that could be added.
3. Currently, if we search for the word lord we get a list of movies in which the word lord appears in the title and we also get a list of movies in which Philip Lord was the writer. We could improve our performance by passing in an additional argument that could tell the program whether we would like to search for movies in which lord appears in the title or lord appears as an actor
4. *Indexing*: Instead of creating our own indexing mechanism, a better idea could be to use Elasticsearch to do the indexing. Elasticsearch is distributed, schema free and is extremely fast. Elasticsearch can also handle misspellings in the user search query