cs109a_hw1_submit

September 19, 2018

1 CS109A Introduction to Data Science

1.1 Homework 1: Data Collection - Web Scraping - Data Parsing

Harvard University Fall 2018 Instructors: Pavlos Protopapas and Kevin Rader

```
In [30]: ## RUN THIS CELL TO GET THE RIGHT FORMATTING
         import requests
         from IPython.core.display import HTML
         styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-CS109A/mas
         HTML(styles)

Out[30]: <IPython.core.display.HTML object>
```

Instructions

- To submit your assignment follow the instructions given in Canvas.
- The deliverables in Canvas are:

  a) This python notebook with your code and answers, plus a pdf version of it (see Canvas for details),
  b) the bibtex file you created,
  c) The CSV file you created,
  d) The JSON file you created.

- Exercise **responsible scraping**. Web servers can become slow or unresponsive if they receive too many requests from the same source in a short amount of time. Use a delay of 10 seconds between requests in your code. This helps not to get blocked by the target website. Run the webpage fetching part of the homework only once and do not re-run after you have saved the results in the JSON file (details below).
- Web scraping requests can take several minutes. This is another reason why you should not wait until the last minute to do this homework.
- For this assignment, we will use Python 3.5 for grading.

2 Data Collection - Web Scraping - Data Parsing

In this homework, your goal is to learn how to acquire, parse, clean, and analyze data. Initially you will read the data from a file, and then later scrape them directly from a website. You will look

for specific pieces of information by parsing the data, clean the data to prepare them for analysis, and finally, answer some questions.

In doing so you will get more familiar with three of the common file formats for storing and transferring data, which are: - CSV, a text-based file format used for storing tabular data that are separated by some delimiter, usually comma or space. - HTML/XML, the stuff the web is made of. - JavaScript Object Notation (JSON), a text-based open standard designed for transmitting structured data over the web.

```
In [31]: # import the necessary libraries
         %matplotlib inline
         import numpy as np
         import scipy as sp
         import matplotlib as mpl
         import matplotlib.cm as cm
         import matplotlib.pyplot as plt
         import pandas as pd
         import time
         pd.set_option('display.width', 500)
         pd.set_option('display.max_columns', 100)
         pd.set_option('display.notebook_repr_html', True)
         import seaborn as sns
```

2.1 Help a professor parse their publications and extract information.

2.1.1 Overview

In this part your goal is to parse the HTML page of a professor containing some of his/her publications, and answer some questions. This page is provided to you in the file `data/publist_super_clean.html`. There are 45 publications in descending order from No. 244 to No. 200.

```
In [32]: # use this file provided
         PUB_FILENAME = 'data/publist_super_clean.html'
```

Question 1 [40 pts]: Parsing and Converting to bibTex and CSV using Beautiful Soup and python string manipulation

A lot of the bibliographic and publication information is displayed in various websites in a not-so-structured HTML files. Some publishers prefer to store and transmit this information in a .bibTex file which looks roughly like this (we've simplified a few things):

```
@article {
     author = "John Doyle"
     title = "Interaction between atoms"
     URL = "Papers/PhysRevB_81_085406_2010.pdf"
     journal = "Phys. Rev. B"
     volume = "81"
}
```

You will notice that this file format is a set of items, each of which is a set of key-value pairs. In the python world, you can think of this as a list of dictionaries. If you think about spreadsheets

(as represented by CSV files), they have the same structure. Each line is an item, and has multiple features, or keys, as represented by that line's value for the column corresponding to the key.

You are given an .html file containing a list of papers scraped from the author's website and you are to write the information into .bibTex and .CSV formats. A useful tool for parsing websites is BeautifulSoup (http://www.crummy.com/software/BeautifulSoup/) (BS). In this problem, will parse the file using BS, which makes parsing HTML a lot easier.

1.1 Write a function called `make_soup` that accepts a filename for an HTML file and returns a BS object.

1.2 Write a function that reads in the BS object, parses it, converts it into a list of dictionaries: one dictionary per paper. Each of these dictionaries should have the following format (with different values for each publication):

```
{'author': 'L.A. Agapito, N. Kioussis and E. Kaxiras',
 'title': '"Electric-field control of magnetism in graphene quantum dots:\n Ab initio calculati
 'URL': 'Papers/PhysRevB_82_201411_2010.pdf',
 'journal': 'Phys. Rev. B',
 'volume': '82'}
```

1.3 Convert the list of dictionaries into standard .bibTex format using python string manipulation, and write the results into a file called `publist.bib`.

1.4 Convert the list of dictionaries into standard tabular .csv format using pandas, and write the results into a file called `publist.csv`. The csv file should have a header and no integer index.

HINT

- Inspect the HTML code for tags that indicate information chunks such as `title` of the paper. The `find_all` method of BeautifulSoup might be useful.
- Question 1.2 is better handled if you break the code into functions, each performing a small task such as finding the author(s) for each paper.
- Question 1.3 is effectively tackled by first using python string formatting on a template string.
- Make sure you catch exceptions when needed.
- Make sure you check for **missing data** and handle these cases as you see fit.

Resources

- BeautifulSoup Tutorial.
- More about the BibTex format.

2.1.2 Answers

```
In [34]: # import the necessary libraries
         from bs4 import BeautifulSoup
```

1.1 Write a function called `make_soup` ...

```
In [35]: def make_soup(filename: str) -> BeautifulSoup:
             '''Open the file and convert into a BS object.
```

```
            Args:
                filename: A string name of the file.

            Returns:
                A BS object containing the HTML page ready to be parsed.
        '''
        # your code here
        with open(filename) as fdr:
            data = fdr.read()
        soup = BeautifulSoup(data, 'html.parser')

        return soup

    soup = make_soup(PUB_FILENAME)
```

1.2 Write a function that reads in the BS object, parses it, converts it into a list of dictionaries...

```
In [43]: # your code here
         def parse_journal(index, publication):
             if publication.find('i'):
                 journal = publication.i.text.lstrip().rstrip()
                 if journal == '':
                     print('Missing journal: ' + str(index))
                     print(publication)

                 return journal
             else:
                 print("Missing journal: " + str(index))
                 print(publication)
                 return ''

         def parse_volume(index, publication):
             if publication.find('b'):
                 volume = publication.b.text.lstrip().rstrip()
                 if volume == '':
                     print("Missing volume: " + str(index))
                     print(publication)
                 return volume
             else:
                 print("Missing volume: " + str(index))
                 print(publication)
                 return ''

         def parse_url(index, publication):
             if publication.find('a'):
                 url = publication.a['href'].lstrip().rstrip()
                 if url == '':
```

```
            print("Missing URL: " + str(index))
            print(publication)
        return url
    else:
        print("Missing URL: " + str(index))
        print(publication)
        return ''

def parse_title(index, publication):
    if publication.find('a'):
        title = publication.a.text.lstrip().rstrip()
        if title == '':
            print("Missing Title: " + str(index))
            print(publication)
        return title
    else:
        print("Missing Title: " + str(index))
        print(publication)
        return ''

def parse_author(index, publication):
    if publication.find('br'):
        author = publication.br.next_sibling.lstrip().rstrip()
        if author != '':
            if author[-1] == ',':
                author = author[:-1]
        else:
            print("Missing author: " + str(index))
            print(publication)
        return author
    else:
        print("Missing author: " + str(index))
        print(publication)
        return ''

def parse_soup(soup):
    results = []
    for index, publication in enumerate(soup.find_all('ol')):
        info = {}
        info['journal'] = parse_journal(index, publication)
        info['volume'] = parse_volume(index, publication)
        info['URL'] = parse_url(index, publication)
        info['title'] = parse_title(index, publication)
        info['author'] = parse_author(index, publication)

        results.append(info)

    return results
```

```
        articles = parse_soup(soup)
        #print(articles)
```

```
Missing volume: 18
<ol start="226">
<li>
<a href="Papers/IEEE-SC10_2010.pdf" target="paper226">
"Multiscale simulation of cardiovascular flows on the IBM Bluegene/P:
full heart-circulation system at near red-blood cell resolution"</a>
<br/> A. Peters, S. Melchionna, E. Kaxiras, J. Latt, J. Sircar, S. Succi,
<i>2010 ACM/IEEE International Conference for High Performance </i>,
 doi: 10.1109/SC.2010.33 (2010).
<br/>
</li>
</ol>
```

1.3 Convert the list of dictionaries into the .bibTex format using python string manipulation (python string formatting on a template string is particularly useful)..

```
In [92]: # your code here
         bib = ''
         for article in articles:
             artical_str = '@article{\n\tauthor = "%s",\n\ttitle = %s,\n\tURL = "%s"\
                           ,\n\tjournal = "%s",\n\tvolume = %s\n}'\
                           % (article['author'], article['title'], \
                           article['URL'], article['journal'], article['volume'])
             bib = bib + artical_str + '\n'

         with open('publist.bib', 'w') as f:
             f.write(bib)
```

```
In [94]: # check your answer - print the bibTex file
         # clear/remove output before making pdf
         f = open('publist.bib','r')
         #print(f.read())
```

Your output should look like this

```
@article{
     author = "Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang Zhang, Mar
     title = "Approaching the intrinsic band gap in suspended high-mobility graphene nanoribbo
     URL = "Papers/2011/PhysRevB_84_125411_2011.pdf",
     journal = "PHYSICAL REVIEW B",
     volume = 84
}

...
```

```
@article{
     author = "E. Kaxiras and S. Succi",
     title = "Multiscale simulations of complex systems: computation meets reality",
     URL = "Papers/SciModSim_15_59_2008.pdf",
     journal = "Sci. Model. Simul.",
     volume = 15
}
```

** 1.4 Convert the list of dictionaries into the .csv format using pandas, and write the data into `publist.csv`. The csv file should have a header and no integer index...**

```
In [74]: # make sure you use head() when printing the dataframe
         # your code here
         df = pd.DataFrame(articles)
         df.to_csv('publist.csv', index=False)
```

2.2 Follow the stars in IMDb's list of "The Top 100 Stars for 2017"

2.2.1 Overview

In this part, your goal is to extract information from IMDb's Top 100 Stars for 2017 (https://www.imdb.com/list/ls025814950/) and perform some analysis on each star in the list. In particular we are interested to know: a) how many performers made their first movie at 17? b) how many performers started as child actors? c) who is the most proliferate actress or actor in IMDb's list of the Top 100 Stars for 2017? . These questions are addressed in more details in the Questions below.

When data is not given to us in a file, we need to fetch them using one of the following ways: - download a file from a source URL - query a database - query a web API - scrape data from the web page

Question 2 [52 pts]: Web Scraping using Beautiful Soup and exploring using Pandas

2.1 Download the webpage of the "Top 100 Stars for 2017" (https://www.imdb.com/list/ls025814950/) into a `requests` object and name it `my_page`. Explain what the following attributes are:

- `my_page.text`,
- `my_page.status_code`,
- `my_page.content`.

2.2 Create a Beautiful Soup object named `star_soup` using `my_page` as input.

2.3 Write a function called `parse_stars` that accepts `star_soup` as its input and generates a list of dictionaries named `starlist` (see definition below; order of dictionaries does not matter). One of the fields of this dictionary is the `url` of each star's individual page, which you need to scrape and save the contents in the `page` field. Note that there is a ton of information about each star on these webpages.

```
name: the name of the actor/actress as it appears at the top
gender: 0 or 1: translate the word 'actress' into 1 and 'actor' into '0'
url: the url of the link under their name that leads to a page with details
page: BS object with html text acquired by scraping the above 'url' page'
```

2.4 Write a function called `create_star_table` which takes `starlist` as an input and extracts information about each star (see function definition for the exact information to be extracted and the exact output definition). Only extract information from the first box on each star's page. If the first box is acting, consider only acting credits and the star's acting debut, if the first box is Directing, consider only directing credits and directorial debut.

2.5 Now that you have scraped all the info you need, it's good practice to save the last data structure you created to disk. Save the data structure to a JSON file named `starinfo.json` and submit this JSON file in Canvas. If you do this, if you have to restart, you won't need to redo all the requests and parsings from before.

2.6 We provide a JSON file called `data/staff_starinfo.json` created by CS109 teaching staff for consistency, which you should use for the rest of the homework. Import the contents of this JSON file into a pandas dataframe called `frame`. Check the types of variables in each column and clean these variables if needed. Add a new column to your dataframe with the age of each actor when they made their first appearance, movie or TV, (name this column `age_at_first_movie`). Check some of the values of this new column. Do you find any problems? You don't need to fix them.

2.7 You are now ready to answer the following intriguing questions: - **2.7.1** How many performers made their first appearance (movie or TV) when he/she was 17 years old?

- **2.7.2** How many performers started as child actors? Define child actor as a person younger than 12 years old.

2.8 Make a plot of the number of credits against the name of actor/actress. Who is the most prolific actress or actor in IMDb's list of the Top 100 Stars for 2017? Define **most prolific** as the performer with the most credits.

2.2.2 Hints

- Create a variable that groups actors/actresses by the age of their first movie. Use pandas' `.groupby` to divide the dataframe into groups of performers that for example started performing as children (age < 12). The grouped variable is a `GroupBy` pandas object and this object has all of the information needed to then apply operations to each of the groups.
- When cleaning the data make sure the variables with which you are performing calculations are in numerical format.
- The column with the year has some values that are double, e.g. **'2000-2001'** and the column with age has some empty cells. You need to deal with these in a reasonable fashion before performing calculations on the data.
- You should include both movies and TV shows.

2.2.3 Resources

- The `requests` library makes working with HTTP requests powerful and easy. For more on the `requests` library see http://docs.python-requests.org/

2.2.4 Answers

```
In [99]: import requests
```

2.1 Download the webpage of the "Top 100 Stars for 2017 ...

```
In [100]: # your code here
          my_page=requests.get("https://www.imdb.com/list/ls025814950/")
          print(my_page.status_code)
```

```
200
```

your answer here

status_code shows if the request is successfully returned. Since it returns 200, it means getting the page back successfully

my_page.text returns the request of response in unicode

my_page.content returns the request of response in byte

2.2 Create a Beautiful Soup object named star_soup giving my_page as input.

```
In [101]: # your code here
          star_soup = BeautifulSoup(my_page.text, 'html.parser')
```

```
In [102]: # check your code - you should see a familiar HTML page
          # clear/remove output before making pdf
          #print (star_soup.prettify()[:])
```

2.3 Write a function called `parse_stars` that accepts `star_soup` as its input ...

```
Function
--------
parse_stars

Input
------
star_soup: the soup object with the scraped page

Returns
-------
a list of dictionaries; each dictionary corresponds to a star profile and has the following da

    name: the name of the actor/actress as it appears at the top
    gender: 0 or 1: translate the word 'actress' into 1 and 'actor' into '0'
    url: the url of the link under their name that leads to a page with details
    page: BS object with 'html text acquired by scraping the above 'url' page'

Example:
--------
{'name': Tom Hardy,
  'gender': 0,
  'url': https://www.imdb.com/name/nm0362766/?ref_=nmls_hd,
  'page': BS object with 'html text acquired by scraping the 'url' page'
}
```

```
In [112]: # your code here

          def parse_stars(star_soup):

              starlist = []

              for e in star_soup.select('.lister-item-content'):
                  star ={}
                  info = (list(e.children)[1].find('a'))

                  star['name'] = list(info.children)[0].rstrip().lstrip()
                  gender_info = list(e.children)[3]
                  gender = (list(gender_info.children)[0]).rstrip().lstrip()
                  star['gender'] = 0 if gender == "Actor"  else 1
                  star['url'] = "https://www.imdb.com" + info['href']
                  star_page = requests.get(star['url'])
                  star['page'] = BeautifulSoup(star_page.text, 'html.parser')

                  starlist.append(star)
                  time.sleep(10)

              return starlist

          starlist = parse_stars(star_soup=star_soup)
```

This should give you 100

```
In [113]: len(starlist)

Out[113]: 100

In [116]: # check your code
          starlist = parse_stars(star_soup=star_soup)
          # this list is large because of the html code into the `page` field
          # to get a better picture, print only the first element
          # clear/remove output before making pdf
          #starlist[0]
```

Your output should look like this: "' {'name': 'Gal Gadot', 'gender': 1, 'url': 'https://www.imdb.com/name/nm2933757?ref_=nmls_hd', 'page':

...

"'

2.4 Write a function called `create_star_table` to extract information about each star ...

```
Function
--------
create_star_table

Input
```

```
------
the starlist

Returns
-------

a list of dictionaries; each dictionary corresponds to a star profile and has the following da

    star_name: the name of the actor/actress as it appears at the top
    gender: 0 or 1 (1 for 'actress' and 0 for 'actor')
    year_born : year they were born
    first_movie: title of their first movie or TV show
    year_first_movie: the year they made their first movie or TV show
    credits: number of movies or TV shows they have made in their career.

--------
Example:

{'star_name': Tom Hardy,
  'gender': 0,
  'year_born': 1997,
  'first_movie' : 'Batman',
  'year_first_movie' : 2017,
  'credits' : 24}
```

```
In [107]: def create_star_table(starlist: list) -> list:
          # your code here
              star_table = []
              for star in starlist:
                  dic = {}
                  dic['name'] = star['name']
                  dic['gender'] = star['gender']

                  born = star['page'].find(id="name-born-info")
                  if born is not None:
                      time = list(born.children)[3]['datetime']
                      dic['year_born'] = time.split('-')[0]
                  else:
                      dic['year_born'] = None


                  films = star['page'].find(id="filmography")
                  summary = list(films.children)[1]
                  dic['credits'] = list(summary.children)[-1]\
                                      .lstrip().split(' ')[0].split('(')[1]

                  films = list(films.children)[3]
```

```
                fir_film = (list(films.children)[-2])
                dic['year_first_movie'] = fir_film.select(".year_column")[0]\
                                            .get_text().lstrip().rstrip()

                dic['first_movie'] = fir_film.find('a').get_text()

                star_table.append(dic)
            return star_table
```

```
In [110]: # check your code
          star_table = create_star_table(starlist=starlist)
          # clear/remove output before making the pdf file
          #star_table
```

Your output should look like this (the order of elements is not important):

```
[{'name': 'Gal Gadot',
  'gender': 1,
  'year_born': '1985',
  'first_movie': 'Bubot',
  'year_first_movie': '2007',
  'credits': '25'},
 {'name': 'Tom Hardy',
  'gender': 0,
  'year_born': '1977',
  'first_movie': 'Tommaso',
  'year_first_movie': '2001',
  'credits': '55'},

...
```

2.5 Now that you have scraped all the info you need, it's a good practice to save the last data structure you ...

```
In [21]: # your code here

         import json
         with open('starinfo.json', 'w') as outfile:
             json.dump(star_table, outfile)
```

To check your JSON saving, re-open the JSON file and reload the code

```
In [23]: with open("starinfo.json", "r") as fd:
             star_table = json.load(fd)

         # output should be the same
         # clear/remove output before making the pdf file
         #star_table
```

2.6 Import the contents of the staff's JSON file (`data/staff_starinfo.json`) into a pandas dataframe. ...

```
In [7]: # your code here
        frame = pd.read_json('data/staff_starinfo.json')
```

```
In [8]: # your code here
        print(frame.dtypes)
```

```
credits              int64
first_movie         object
gender               int64
name                object
year_born            int64
year_first_movie    object
dtype: object
```

```
In [85]: # your code here
         # fix the year_first_movie column and make the column to a integer type
         frame['year_first_movie'] = frame.year_first_movie.astype(str)\
                                     .str.split('-').str[0].str.split('/').str[0].astype(int)
```

```
In [84]: # your code here
         frame['age_at_first_movie']= (frame['year_first_movie'] - frame['year_born'])

         print(frame.age_at_first_movie.describe())

         print(frame.loc[frame['age_at_first_movie'].idxmin()])
```

```
count    100.000000
mean      17.570000
std        7.064286
min       -1.000000
25%       14.750000
50%       19.000000
75%       21.000000
max       48.000000
Name: age_at_first_movie, dtype: float64
credits                          32
first_movie            Only Yesterday
gender                              1
name                    Daisy Ridley
year_born                       1992
year_first_movie                1991
age_at_first_movie                -1
Name: 63, dtype: object
```

your answer here Yes. The value of 'age_at_first_movie' column should be non-negative. However, while getting the summary of 'age_at_first_movie', the minimum value is -1 which does not make sense. While taking a further look at the dataset, Daisy Ridley was born in 1992 but his first movie is in 1991.

2.7 You are now ready to answer the following intriguing questions:

2.7.1 How many performers made their first movie at 17?

```
In [86]: # your code here
         print(frame[frame['age_at_first_movie']==17].shape[0],\
               "performers made their first movie at 17")
```

```
8 performers made their first movie at 17
```

Your output should look like this: 8 performers made their first movie at 17

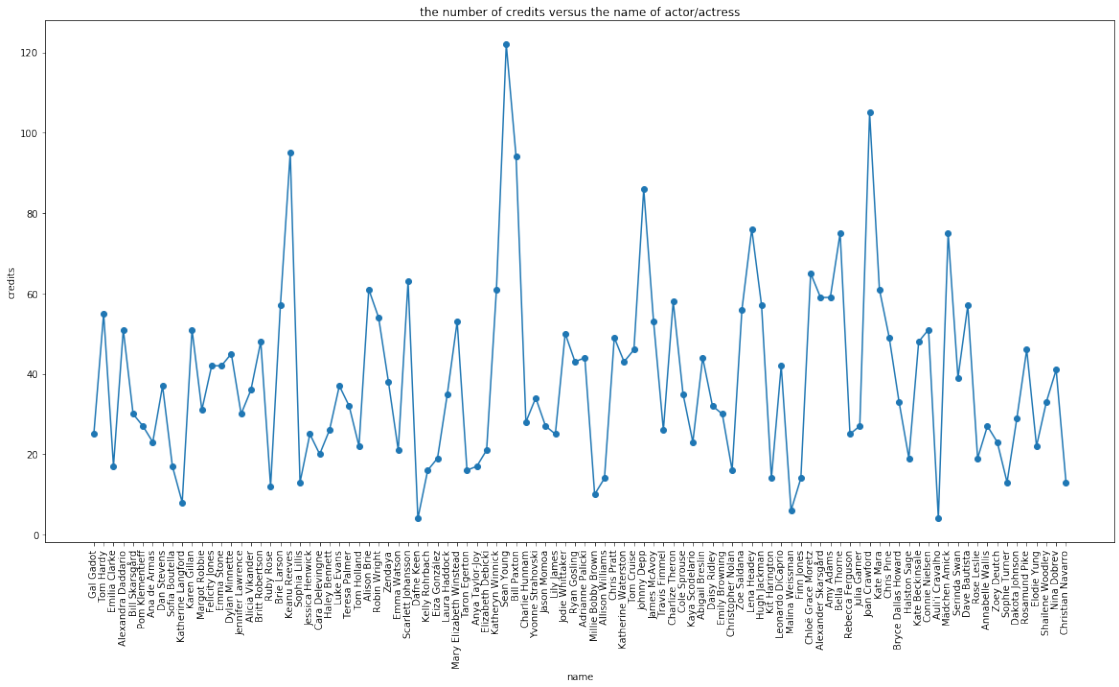
2.7.2 How many performers started as child actors? Define child actor as a person less than 12 years old.

```
In [87]: # your code here
         print(frame[frame['age_at_first_movie']<12].shape[0],\
               "performers started as child actors")
```

```
20 performers started as child actors
```

2.8 Make a plot of the number of credits versus the name of actor/actress.

```
In [32]: # your code here
         plt.figure(figsize=(20,10))
         plt.plot(frame.name,frame.credits, '-o')
         plt.xticks(rotation=90)
         plt.ylabel('credits')
         plt.xlabel('name')
         plt.title('the number of credits versus the name of actor/actress')
         plt.show()
```




```
In [88]: # your code here

         print(frame.loc[frame['credits'].idxmax()]['name']\
               ,"is the most prolific actress or actor in IMDb's list of the Top 100 Stars for
```

```
Sean Young is the most prolific actress or actor in IMDb's list of the Top 100 Stars for 2017
```

2.3 Going the Extra Mile

Be sure to complete problems 1 and 2 before tackling this problem...it is worth only 8 points.

Question 3 [8 pts]: Parsing using Regular Expressions (regex)

Even though scraping HTML with regex is sometimes considered bad practice, you are to use python's **regular expressions** to answer this problem. Regular expressions are useful to parse strings, text, tweets, etc. in general (for example, you may encounter a non-standard format for dates at some point). Do not use BeautifulSoup to answer this problem.

3.1 Write a function called `get_pubs` that takes an .html filename as an input and returns a string containing the HTML page in this file (see definition below). Call this function using `data/publist_super_clean.html` as input and name the returned string `prof_pubs`.

3.2 Calculate how many times the author named '`C.M. Friend`' appears in the list of publications.

3.3 Find all unique journals and copy them in a variable named `journals`.

3.4 Create a list named `pub_authors` whose elements are strings containing the authors' names for each paper.

2.3.1 Hints

- Look for patterns in the HTML tags that reveal where each piece of information such as the title of the paper, the names of the authors, the journal name, is stored. For example, you might notice that the journal name(s) is contained between the <I> HTML tag.
- Learning about your domain is always a good idea: you want to check the names to make sure that they belong to actual journals. Thus, while journal name(s) is contained between the <I> HTML tag, please note that all strings found between <I> tags may not be journal names.
- Each publication has multiple authors.
- `C.M. Friend` also shows up as `Cynthia M. Friend` in the file. Count just `C. M. Friend`.
- There is a comma at the end of the string of authors. You can choose to keep it in the string or remove it and put it back when you write the string as a BibTex entry.
- You want to remove duplicates from the list of journals. Duplicates may also occur due to misspellings or spaces, such as: `Nano Lett.`, and `NanoLett.` You can assume that any journals with the same initials (e.g., `NL` for `NanoLett.`) are the same journal.

2.3.2 Resources

- **Regular expressions:** a) https://docs.python.org/3.3/library/re.html, b) https://regexone.com, and c) https://docs.python.org/3/howto/regex.html.
- ** HTML:** if you are not familiar with HTML see https://www.w3schools.com/html/ or one of the many tutorials on the internet.
- ** Document Object Model (DOM):** for more on this programming interface for HTML and XML documents see https://www.w3schools.com/js/js_htmldom.asp.

```
In [55]: # first import the necessary reg expr library
         import re

In [56]: # use this file provided
         PUB_FILENAME = 'data/publist_super_clean.html'

In [57]: # your code here
         def get_pubs(filehtml):
             with open(filehtml) as fdr:
                 data = fdr.read()
             return data

In [58]: # your code here
         prof_pubs = get_pubs(PUB_FILENAME)

In [60]: # checking your code
         # clear/remove output before creating the pdf file
         #print(prof_pubs)
```

You should see an HTML page that looks like this (colors are not important) "'html
"Approaching the intrinsic band gap in suspended high-mobility graphene nanoribbons" Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang Zhang, Mark Ming-Cheng Cheng, PHYSICAL REVIEW B 84, 125411 (2011)

"Effect of symmetry breaking on the optical absorption of semiconductor nanoparticles" JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, Sheng Meng, PHYSICAL REVIEW B 84, 035325 (2011)

"Influence of CH2 content and network defects on the elastic properties of organosilicate glasses" Jan M. Knaup, Han Li, Joost J. Vlassak, and Efthimios Kaxiras, PHYSICAL REVIEW B 83, 054204 (2011)

```

### 3.2 Calculate how many times the author ...

```
In [61]: # your code here
         CMF_num = len(re.findall(r"C.M. Friend", prof_pubs))
         print("C.M. Friend appears %d times." % CMF_num)
```

```
C.M. Friend appears 5 times.
```

3.3 Find all unique journals and copy ...

```
In [67]: # your code here
         journals = re.findall(r"(?<=<I>).*(?=</I>)", prof_pubs)
         journals = set(journals)

In [68]: # According to domain knowledge, 'Ab initio' is not a journal, so drop it.
         journals.remove('Ab initio')

In [69]: # Drop duplicates
         df = pd.DataFrame({'raw': list(journals)})

In [70]: df.sort_values(by='raw', inplace=True)
         df['init'] = df['raw'].apply(lambda x: ''.join([c for c in x if c.isupper()]))
         df['dup'] = df.duplicated(subset=['init'])
         df = df[df['dup']==False]

         df['split'] = df['raw'].apply(lambda x: ''.join([word[0] for word in x.split()]))
         df['dup2'] = df.duplicated(subset=['split'])
         df = df[df['dup2']==False]

In [71]: # check your code
         journals = set(df['raw'])
         journals

Out[71]: {'2010 ACM/IEEE International Conference for High Performance ',
          'ACSNano. ',
          'Acta Mater. ',
          'Catal. Sci. Technol. ',
          'Chem. Eur. J. ',
          'Comp. Phys. Comm. ',
          'Concurrency Computat.: Pract. Exper. ',
```
```

```
 'Energy & Environmental Sci. ',
 'Int. J. Cardiovasc. Imaging ',
 'J. Chem. Phys. ',
 'J. Chem. Theory Comput. ',
 'J. Phys. Chem. B ',
 'J. Phys. Chem. C ',
 'J. Phys. Chem. Lett. ',
 'J. Stat. Mech: Th. and Exper. ',
 'Langmuir ',
 'Molec. Phys. ',
 'Nano Lett. ',
 'New J. Phys. ',
 'PHYSICAL REVIEW B ',
 'Phil. Trans. R. Soc. A ',
 'Phys. Rev. E - Rap. Comm. ',
 'Phys. Rev. Lett. ',
 'Sci. Model. Simul. ',
 'Sol. St. Comm. ',
 'Top. Catal. '}
```

Your output should look like this (no duplicates): `{'2010 ACM/IEEE International Conference for High Performance', 'ACSNano.', 'Acta Mater.', 'Catal. Sci. Technol.', 'Chem. Eur. J.', 'Comp. Phys. Comm.', 'Concurrency Computat.: Pract. Exper.', 'Energy & Environmental Sci.', 'Int. J. Cardiovasc. Imaging', 'J. Chem. Phys.', 'J. Chem. Theory Comput.', 'J. Phys. Chem. B', 'J. Phys. Chem. C', 'J. Phys. Chem. Lett.', 'J. Stat. Mech: Th. and Exper.', 'Langmuir', 'Molec. Phys.', 'Nano Lett.', 'New Journal of Physics', 'PHYSICAL REVIEW B', 'Phil. Trans. R. Soc. A', 'Phys. Rev. E - Rap. Comm.', 'Phys. Rev. Lett.', 'Sci. Model. Simul.', 'Sol. St. Comm.', 'Top. Catal.'}`

3.4 Create a list named `pub_authors`...

```
In [72]: # your code here
         raw_authors = re.findall(r"(?<=<BR>).{5,}(?=\n)", prof_pubs)
         pub_authors = [x.lstrip().rstrip() for x in raw_authors]
```

```
In [73]: # check your code: print the list of strings containing the author(s)' names
         for item in pub_authors:
             print (item)
```

```
Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang Zhang, Mark Ming-Cheng Che
JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, Sheng Meng,
Jan M. Knaup, Han Li, Joost J. Vlassak, and Efthimios Kaxiras,
Martin Heiss, Sonia Conesa-Boj, Jun Ren, Hsiang-Han Tseng, Adam Gali,
Simone Melchionna, Efthimios Kaxiras, Massimo Bernaschi and Sauro Succi,
J R Maze, A Gali, E Togan, Y Chu, A Trifonov,
Kejie Zhao, Wei L. Wang, John Gregoire, Matt Pharr, Zhigang Suo,
Masataka Katono, Takeru Bessho, Sheng Meng, Robin Humphry-Baker, Guido Rothenberger,
Thomas D. Kuhne, Tod A. Pascal, Efthimios Kaxiras, and Yousung Jung,
Sheng Meng, Efthimios Kaxiras, Md. K. Nazeeruddin, and Michael Gratzel,
```

```
Bingjun Xu, Jan Haubrich, Thomas A. Baker, Efthimios Kaxiras, and Cynthia M. Friend,
Jun Ren, Sheng Meng, Yi-Lin Wang, Xu-Cun Ma, Qi-Kun Xue, Efthimios Kaxiras,
Jan Haubrich, Efthimios Kaxiras, and Cynthia M. Friend,
Thomas A. Baker, Bingjun Xu, Stephen C. Jensen, Cynthia M. Friend and Efthimios Kaxiras,
Youdong Mao, Wei L. Wang, Dongguang Wei, Efthimios Kaxiras, and Joseph G. Sodroski,
H. Li, J.M. Knaup, E. Kaxiras and J.J. Vlassak,
W.L. Wang and E. Kaxiras,
L.A. Agapito, N. Kioussis and E. Kaxiras,
A. Peters, S. Melchionna, E. Kaxiras, J. Latt, J. Sircar, S. Succi,
J. Ren, E. Kaxiras and S. Meng,
T.A. Baker, E. Kaxiras and C.M. Friend,
H.P. Chen, R.K. Kalia, E. Kaxiras, G. Lu, A. Nakano, K. Nomura,
S. Meng and E. Kaxiras,
C.L. Chang, S.K.R.S. Sankaranarayanan, D. Ruzmetov, M.H. Engelhard, E. Kaxiras and S. Ramanatha
T.A. Baker, C.M. Friend and E. Kaxiras,
S. Melchionna, M. Bernaschi, S. Succi, E. Kaxiras, F.J. Rybicki, D. Mitsouras, A.U. Coskun and
M. Bernaschi, M. Fatica, S. Melchionna, S. Succi and E. Kaxiras,
E. Manousakis, J. Ren, S. Meng and E. Kaxiras,
A. Gali, E. Janzen, P. Deak, G. Kresse and E. Kaxiras,
S.K.R.S. Sankaranarayanan, E. Kaxiras and S. Ramanathan,
M. Bernaschi, S. Melchionna, S. Succi, M. Fyta, E. Kaxiras
T.A. Baker, B.J. Xu, X.Y. Liu, E. Kaxiras and C.M. Friend,
F.J. Rybicki, S. Melchionna, D. Mitsouras, A.U. Coskun, A.G. Whitmore, E. Kaxiras, S. Succi, P
H. Chen, W.G. Zhu, E. Kaxiras, and Z.Y. Zhang,
M. Fyta, S. Melchionna, M. Bernaschi, E. Kaxiras and S. Succi,
E.M. Kotsalis, J.H. Walther, E. Kaxiras and P. Koumoutsakos,
C.E. Lekka, J. Ren, S. Meng and E. Kaxiras,
W.L. Wang, O.V. Yazyev, S. Meng and E. Kaxiras,
A. Gali and E. Kaxiras,
S. Melchionna, M. Bernaschi, M. Fyta, E. Kaxiras and S. Succi,
S.K.R.S. Sankaranarayanan, E. Kaxiras, S. Ramanathan,
T.A. Baker, C.M. Friend and E. Kaxiras,
T.A. Baker, C.M. Friend and E. Kaxiras,
E. Kaxiras and S. Succi,
E. Manousakis, J. Ren, S. Meng and E. Kaxiras,
```

Your output should look like this (a line for each paper's authors string of names)

```
Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang Zhang, Mark Ming-Cheng Che
JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, Sheng Meng,
Jan M. Knaup, Han Li, Joost J. Vlassak, and Efthimios Kaxiras,
Martin Heiss, Sonia Conesa-Boj, Jun Ren, Hsiang-Han Tseng, Adam Gali,

...

T.A. Baker, C.M. Friend and E. Kaxiras,
T.A. Baker, C.M. Friend and E. Kaxiras,
```

E. Kaxiras and S. Succi,
E. Manousakis, J. Ren, S. Meng and E. Kaxiras,