cs109a_hw1

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 [1]: ## 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/maste
        HTML(styles)
```

```
Out[1]: <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 [2]: # 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
        import re as re
```

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 [3]: # 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 [4]: # import the necessary libraries
        from bs4 import BeautifulSoup
```

1.1 Write a function called `make_soup` ...

```
In [20]: def make_soup(PUB_FILENAME):
             data = open(PUB_FILENAME).read()
             soup = BeautifulSoup(data, 'html.parser')
             return soup
         #data = open(PUB_FILENAME).read()
         #soup = BeautifulSoup(data, 'html.parser')

In [ ]: # check your code - print the BS object, you should get a familiar HTML page as text
        # clear/remove output before making pdf
        print(soup.prettify())
```

Your output should look **like** this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<title>Kaxiras E journal publications</title>
<head>
<meta content="text/html;charset=utf-8" http-equiv="Content-Type"/>
<link href="../styles/style_pubs.css" rel="stylesheet" type="text/css"/>
<meta content="" name="description"/>
<meta content="Kaxiras E, Multiscale Methods, Computational Materials" name="keywords"/>
</head>
<body>
<ol start="244">
<li>
<a href="Papers/2011/PhysRevB_84_125411_2011.pdf" target="paper244">
"Approaching the intrinsic band gap in suspended high-mobility graphene nanoribbons"</a>
<br/>Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang Zhang, Mark Ming-Che
<i>PHYSICAL REVIEW B </i> <b>84</b>,  125411 (2011)
<br/>
</li>
</ol>
<ol start="243">
<li>
<a href="Papers/2011/PhysRevB_84_035325_2011.pdf" target="paper243">
"Effect of symmetry breaking on the optical absorption of semiconductor nanoparticles"</a>
<br/>JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, Sheng Meng,
<i>PHYSICAL REVIEW B </i> <b>84</b>,  035325 (2011)
<br/>
</li>
</ol>

...
```

```
In [13]: publication = soup.find('body').get_text()
```

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

```
In [14]: content = soup.find('body').text
         fetched=[]
         for i in range(1,90,2):
             body_tag=soup.body
             body_tag
             body_tag.contents
             pub_tag=body_tag.contents[i]
```

First lets try to find some article links

```
In [ ]: #clear output before making pdf
        #your code here
        listofurls=[]
        for e in soup.select('a'):
            listofurls.append(e['href'])
        print(listofurls)
```

OK, now let's try to find the titles...

```
In [16]: all_pubs = soup.select('li')
         each_pub = all_pubs[1]
         HTML(each_pub.prettify())
         each_pub.select_one('a').text[2:][:-1]

Out[16]: 'Effect of symmetry breaking on the optical absorption of semiconductor nanoparticles

In [17]: len(all_pubs)

Out[17]: 45

In [18]: def get_pub_title(each_pub):
             return each_pub.select_one('a').text.replace('"', '').replace('\n','')
```

Make a list of all the publication titles

```
In [22]: listoftitles = [get_pub_title(each_pub) for each_pub in all_pubs]
```

Next let's try to get the volume number

```
In [23]: def get_pub_vol(each_pub):
             vol_node = each_pub.select_one('b')
             if not vol_node:
                 return None
             return each_pub.select_one('b').text

In [24]: listofvolumes = [get_pub_vol(each_pub) for each_pub in all_pubs]

In [25]: len(listofvolumes)

Out[25]: 45
```

Next let's try to get the journal names

```
In [26]: listofjournals=[]
         for e in soup.select('li'):
             listofjournals.append(e('i')[0].text[:-1])

In [27]: len(listofjournals)

Out[27]: 45
```

Finally let's find the author names

```
In [28]: listofauth=[]
         for i in range(1,91,2):
             authors = soup.contents[6].contents[i].contents[1].contents[4]
             listofauth.append(authors.strip().rstrip(','))

In [29]: len(listofauth)

Out[29]: 45

In [ ]: # ok so now we have a bunch of lists, but we need to compile them into a list of dicti
        # So, we have to loop so that we call on the "n_th" element and compile it into the di
        listofdicts=[]
        for i in range (0,45):
            td={}
            td['author']=listofauth[i]
            td['title']= listoftitles[i]
            td['URL']=listofurls[i]
            td['journal']=listofjournals[i]
            td['volume']= listofvolumes[i]
            listofdicts.append(td)
        print(listofdicts)
```

Let's just pull one out and make sure it looks good

```
In [31]: listofdicts[44]

Out[31]: {'URL': 'Papers/PhysRevB_78_205112_2008.pdf',
          'author': 'E. Manousakis, J. Ren, S. Meng and E. Kaxiras',
          'journal': 'Phys. Rev. B',
          'title': 'Effective Hamiltonian for FeAs-based superconductors',
          'volume': '78'}
```

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 [33]: # your code here
         bibtex_article_template = "@article{{author = \"{author}\"), \n title = \"{title}\"),'
         publists_bibtex=[]
```

```
        for i in range (0,45):
            td={}
            td['@article']= listofdicts[i]
            bibtex_article_template.format(**{'author': 'E. Manousakis, J. Ren, S. Meng and E
         'title': 'Effective Hamiltonian for FeAs-based superconductors',
         'URL': 'Papers/PhysRevB_78_205112_2008.pdf',
         'journal': 'Phys. Rev. B',
         'volume': '78'})
            publists_bibtex.append(td)
```

```
In [34]: with open('data/publist.bib', 'w') as f:
             for item in publists_bibtex:
                 f.write("%s\n" % item)
```

```
In [ ]: # check your answer - print the bibTex file
        # clear/remove output before making pdf
        f = open('data/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 [38]: # make sure you use head() when printing the dataframe
         # your code here
         df_pubs_raw = pd.DataFrame(listofdicts)
         df_pubs_raw.head()
```

```
Out[38]:                                                 URL
         0       Papers/2011/PhysRevB_84_125411_2011.pdf  Ming-Wei Lin, Cheng Ling, Luis A. Aga
```

```
1       Papers/2011/PhysRevB_84_035325_2011.pdf  JAdam Gali, Efthimios Kaxiras, Gergel
2       Papers/2011/PhysRevB_83_054204_2011.pdf  Jan M. Knaup, Han Li, Joost J. Vlassa
3       Papers/2011/PhysRevB_83_045303_2011.pdf  Martin Heiss, Sonia Conesa-Boj, Jun L
4  Papers/2011/PhilTransRSocA_369_2354_2011.pdf  Simone Melchionna, Efthimios Kaxiras
```

```
In [39]: # your code here
         df_pubs_raw.to_csv("data/publist.csv", index=False, header=True)
```

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 [40]: import requests
```

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

```
In [41]: # your code here
         my_page_url = 'https://www.imdb.com/list/ls025814950/'
```

```
In [46]: my_page = requests.get(my_page_url)
         my_page.status_code
```

```
Out[46]: 200
```

- `my_page.text` provides all the raw text on that page.
- `my_page.status_code` provides the server status of that url.
- `my_page.content` provides all the html content of that url.

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

```
In [47]: # your code here
         raw_html = my_page.text
         print(raw_html[:500])
```

```
<!DOCTYPE html>
<html
    xmlns:og="http://ogp.me/ns#"
    xmlns:fb="http://www.facebook.com/2008/fbml">
    <head>

        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="apple-itunes-app" content="app-id=342792525, app-argument=imdb:///list/ls02581

        <script type="text/javascript">var IMDbTimer={starttime: new Date().getTime(),pt:'java

<script>
    if (typeof uet == 'function') {
      uet(
```

```
In [48]: star_soup = BeautifulSoup(raw_html, 'html.parser')
```

```
In [ ]: # 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 dat

    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 [50]: # parse each actor node
         actor_nodes = star_soup.select('.lister-item-content')
         len(actor_nodes)
```

```
Out[50]: 100
```

```
In [51]: actor_node = actor_nodes[3]
```

```
In [52]: HTML(actor_node.prettify())
```

```
Out[52]: <IPython.core.display.HTML object>
```

This looks good. Let's see if we can just get her name.

```
In [53]: actor_node.select_one('a').text[:-1][1:]
```

```
Out[53]: 'Alexandra Daddario'
```

```
In [54]: def get_actor_name(actor_node):
             return actor_node.select_one('a').text[:-1][1:]
```

Let's get the actor's gender

```
In [55]: actor_node.select_one('p').text
```

```
Out[55]: '\n                        Actress |\n Baywatch\n '
```

```
In [56]: def get_actor_gender(actor_node):
             act_gender = actor_node.select_one('p').text
             if "Actress" not in act_gender:
                 return int(0)
             else:
                 return int(1)
```

Awesome, let's get the URL

```
In [57]: def get_actor_url(actor_node):
             return "https://www.imdb.com" + actor_node.select_one('a')['href']
         print("https://www.imdb.com" + actor_node.select_one('a')['href'])
```

```
https://www.imdb.com/name/nm1275259?ref_=nmls_hd
```

```
In [59]: actor_page_url = [get_actor_url(actor_node) for actor_node in actor_nodes]
```

OK let's get the HTML from those individual actor pages

```
In [60]: actor_html = requests.get(actor_page_url[3])
```

```
In [62]: raw_actor_html = actor_html.text
```

Cool, let's turn it into a beautiful soup object

```
In [63]: actor_soup = BeautifulSoup(raw_actor_html, 'html.parser')
```

```
In [64]: def get_actor_soup(actor_node):
             actor_page_url = [get_actor_url(actor_node) for actor_node in actor_nodes]
             actor_html = requests.get(actor_page_url)
             raw_actor_html = actor_html.text
             actor_soup = BeautifulSoup(raw_actor_html, 'html.parser')
             time.sleep(1)
```

OK, let's put it into a for loop and index each page so we don't overwhelm the server

```
In [ ]: # let's get an html file for each actor
        for i in range(0,100):
            actor_page_stuff = actor_page_url[i]
            stuff=requests.get(actor_page_stuff)
            filetowrite="data/actor"+ '%02d' % i + ".html"
            print("FTW", filetowrite)
            fd=open(filetowrite,"w")
            fd.write(stuff.text)
            fd.close()
            time.sleep(2)
```

Ok let's make a list of all the pages

```
In [ ]: pagelist = []
        for i in range(0,100):
            actor_file = "data/page"+ '%02d' % i + ".html"
            actor_data = open(actor_file).read()
            actor_soup[i] = BeautifulSoup(actor_data, 'html.parser')
            pagelist.append(actor_soup[i])
            time.sleep(2)
```

Let's make a list of actor names

```
In [66]: nameslist = [get_actor_name(actor_node) for actor_node in actor_nodes]
```

Now a list of actor genders

```
In [67]: genderlist = [get_actor_gender(actor_node) for actor_node in actor_nodes]
```

Now a list of URLs

```
In [68]: urllist = [get_actor_url(actor_node) for actor_node in actor_nodes]
```

Ok let's put this all into a list of dictionaries

```
In [69]: # your code here
         starlist=[]
         for i in range (0,100):
             ad={}
             ad['name']= nameslist[i]
             ad['gender']= genderlist[i]
             ad['URL']= urllist[i]
             ad['page']= pagelist[i]
             starlist.append(ad)
```

This should give you 100

```
In [70]: len(starlist)

Out[70]: 100
```

```
In [ ]: # check your code
        # 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[99]
```

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 dat

    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}
```

Let's start with getting year_born

```
In [72]: def get_date(star):
             birthdate = star.time.attrs['datetime'][0:4]
             return birthdate

         star = starlist[0]['page']
         get_date(star)

Out[72]: '1985'
```

ok now let's find the credits

```
In [73]: def get_credits(star):
             credit_raw = star.find("div", id="filmography").text
             credit = re.findall('\((.*?)\)', credit_raw)[0]
             return credit
         star = starlist[3]['page']
         get_credits(star)

Out[73]: '55 credits'
```

Great, how do we find the first movie and year?

```
In [74]: def get_first_movie(star):
             movies_raw = star.select('.filmo-category-section')[0]
```

```
            first_movie = movies_raw.select('b')[-1].text
            return first_movie
        star = starlist[0]['page']
        get_first_movie(star)
```

```
Out[74]: 'Bubot'
```

```
In [75]: def get_first_year(star):
            movies_raw = star.select('.filmo-category-section')[0]
            first_year = movies_raw.select('.year_column')[-1].text[2:].rstrip("\n")
            return first_year
        star = starlist[55]['page']
        get_first_year(star)
```

```
Out[75]: '1981'
```

```
In [76]: for star in starlist:
            try:
                star_table = {'star_name': star["name"], 'gender': star["gender"], 'year_born
            except AttributeError:
                star_table = {'star_name': star["name"], 'gender': star["gender"], 'year_born
                continue
        print(star_table)
```

```
{'star_name': 'Christian Navarro', 'gender': 0, 'year_born': None, 'credits': None, 'first_movi
```

```
In [77]: def create_star_table(starlist):
            star_table_list=[]
            for star in starlist:
                try:
                    star_table = {'star_name': star["name"], 'gender': star["gender"], 'year_
                except AttributeError:
                    star_table = {'star_name': star["name"], 'gender': star["gender"], 'year_
                    continue
                star_table_list.append(star_table)
            return star_table_list
        star_table_list = create_star_table(starlist)
```

```
In [ ]: # check your code
        # clear/remove output before making the pdf file
        star_table_list
```

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 [80]: # your code here
         import json
         with open('data/star_table_list.json', 'w') as fp:
             json.dump(star_table_list, fp)
```

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

```
In [ ]: with open('data/star_table_list.json', 'r') as json_data:
            df = json.load(json_data)

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

2.6 We provide a JSON file called data/staff_starinfo.json created by CS109...

```
In [82]: # your code here
         with open('data/staff_starinfo.json', 'r') as f:
             data = json.load(f)
         df = pd.DataFrame(data)

In [83]: # your code here
         df.head()
```

Out[83]:

	credits	first_movie	gender	name	year_born	year_first_movie
0	25	Bubot	1	Gal Gadot	1985	2007
1	55	Tommaso	0	Tom Hardy	1977	2001
2	17	Doctors	1	Emilia Clarke	1986	2009
3	51	All My Children	1	Alexandra Daddario	1986	2002-2003
4	30	Järngänget	0	Bill Skarsgård	1990	2000

```
In [84]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
```

```
Data columns (total 6 columns):
credits             100 non-null object
first_movie         100 non-null object
gender              100 non-null int64
name                100 non-null object
year_born           100 non-null object
year_first_movie    100 non-null object
dtypes: int64(1), object(5)
memory usage: 4.8+ KB
```

```
In [86]: df.year_first_movie.value_counts().head()

Out[86]: 1999    6
         2005    6
         2004    6
         2006    5
         2011    5
         Name: year_first_movie, dtype: int64
```

We need to change the type of these variables and clean up the year_first_movie variable

```
In [92]: df['year_initial'] = df['year_first_movie'].str[:4]
         df.dtypes
         df['year_initial'] = df['year_initial'].astype('int')
         df.head()

Out[92]:    credits      first_movie gender               name  year_born year_first_movie  ye
         0       25            Bubot      1          Gal Gadot       1985              2007
         1       55          Tommaso      0          Tom Hardy       1977              2001
         2       17          Doctors      1      Emilia Clarke       1986              2009
         3       51  All My Children      1  Alexandra Daddario       1986         2002-2003
         4       30       Järngänget      0     Bill Skarsgård       1990              2000

In [93]: df.dtypes
         df['credits'] = df['credits'].astype('int')

In [94]: df.dtypes
         df['year_born'] = df['year_born'].astype('int')

In [95]: df.dtypes
         df['gender'] = df['gender'].astype('category')

In [96]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 7 columns):
credits             100 non-null int64
```

```
first_movie         100 non-null object
gender              100 non-null category
name                100 non-null object
year_born           100 non-null int64
year_first_movie    100 non-null object
year_initial        100 non-null int64
dtypes: category(1), int64(3), object(3)
memory usage: 5.0+ KB
```

```
In [ ]: df['age_at_first_movie'] = df['year_initial'] - df['year_born']
        df = df.drop(['year_first_movie'], axis=1)
        df.rename(index=int, columns={"year_initial": "year_first_movie"})
```

```
In [105]: df.head()
```

```
Out[105]:    credits      first_movie gender               name  year_born  year_initial  age
          0       25            Bubot      1          Gal Gadot       1985          2007
          1       55          Tommaso      0          Tom Hardy       1977          2001
          2       17          Doctors      1      Emilia Clarke       1986          2009
          3       51  All My Children      1  Alexandra Daddario       1986          2002
          4       30       Järngänget      0     Bill Skarsgård       1990          2000
```

There is a person whose first movie was when they were -1 years old. This is obviously wrong. There is also a person whose first movie was when they were 1 years old. This may be true, but it seems unlikely.

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 [107]: df.age_at_first_movie.value_counts()
```

```
Out[107]: 21    12
          19    11
          17     8
          16     8
          22     8
          Name: age_at_first_movie, dtype: int64
```

```
In [108]: # your code here
          movie_at_17 = (df.age_at_first_movie == 17).value_counts()
          n_at_17 = movie_at_17[1]
          print(n_at_17,"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.

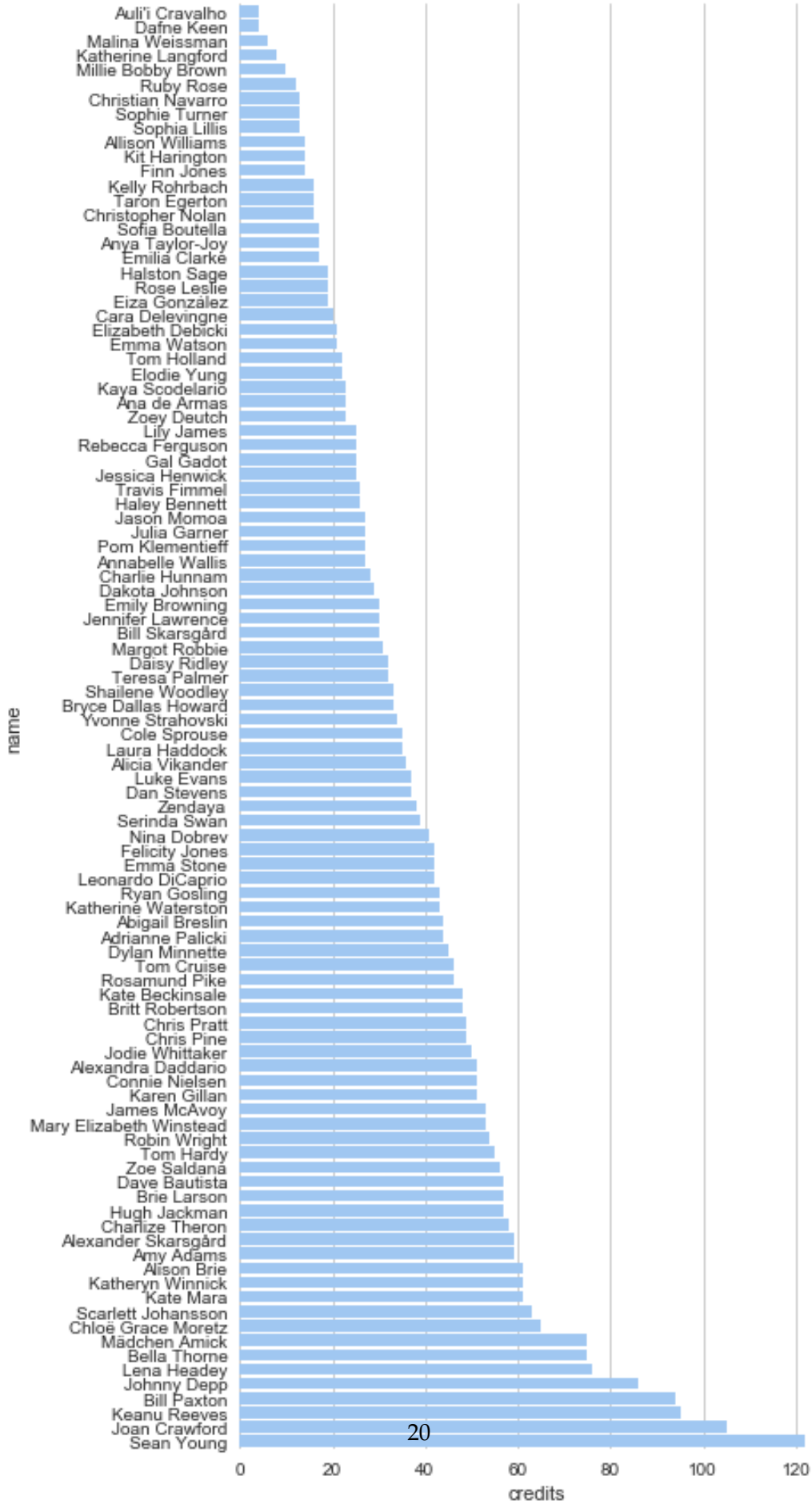
In [109]:
```
# your code here
under_12 = (df.age_at_first_movie > 4.8).value_counts()
n_under_12 = under_12[1]
print(n_under_12,"performers started as child actors.")
```

```
96 performers started as child actors.
```

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

In [110]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_context("notebook")
sns.set(style="whitegrid")
```

In [112]:
```
# your code here
f, ax = plt.subplots(figsize=(6, 15))
credits_df = df.sort_values(['credits']).reset_index(drop=True)
ax = sns.barplot(x="credits", y="name", data=credits_df)
sns.set_color_codes("pastel")
sns.barplot(x="credits", y="name", data=credits_df,
            label="Total", color="b")
sns.despine(left=True, bottom=True)
```


Auli'i Cravalho
Dafne Keen
Malina Weissman
Katherine Langford
Millie Bobby Brown
Ruby Rose
Christian Navarro
Sophie Turner
Sophia Lillis
Allison Williams
Kit Harington
Finn Jones
Kelly Rohrbach
Taron Egerton
Christopher Nolan
Sofia Boutella
Anya Taylor-Joy
Emilia Clarke
Halston Sage
Rose Leslie
Eiza González
Cara Delevingne
Elizabeth Debicki
Emma Watson
Tom Holland
Elodie Yung
Kaya Scodelario
Ana de Armas
Zoey Deutch
Lily James
Rebecca Ferguson
Gal Gadot
Jessica Henwick
Travis Fimmel
Haley Bennett
Jason Momoa
Julia Garner
Pom Klementieff
Annabelle Wallis
Charlie Hunnam
Dakota Johnson
Emily Browning
Jennifer Lawrence
Bill Skarsgård
Margot Robbie
Daisy Ridley
Teresa Palmer
Shailene Woodley
Bryce Dallas Howard
Yvonne Strahovski
Cole Sprouse
Laura Haddock
Alicia Vikander
Luke Evans
Dan Stevens
Zendaya
Serinda Swan
Nina Dobrev
Felicity Jones
Emma Stone
Leonardo DiCaprio
Ryan Gosling
Katherine Waterston
Abigail Breslin
Adrianne Palicki
Dylan Minnette
Tom Cruise
Rosamund Pike
Kate Beckinsale
Britt Robertson
Chris Pratt
Chris Pine
Jodie Whittaker
Alexandra Daddario
Connie Nielsen
Karen Gillan
James McAvoy
Mary Elizabeth Winstead
Robin Wright
Tom Hardy
Zoe Saldana
Dave Bautista
Brie Larson
Hugh Jackman
Charlize Theron
Alexander Skarsgård
Amy Adams
Alison Brie
Katheryn Winnick
Kate Mara
Scarlett Johansson
Chloë Grace Moretz
Mädchen Amick
Bella Thorne
Lena Headey
Johnny Depp
Bill Paxton
Keanu Reeves
Joan Crawford
Sean Young
name
0
20
40
60
80
100
120
credits

```
In [113]: # your code here
          max_credits = df.credits.max()
          most_prolific = df.loc[df['credits'] == 122, 'name'].iloc[0]
          print("The actor with the most credits is "+ most_prolific)
```

```
The actor with the most credits is Sean Young
```

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.

2.3.3 Answers

** 3.1 Write a function called `get_pubs` that takes an .html filename as an input and returns a string ... **

```
In [114]: # first import the necessary reg expr library
          import re

In [115]: # use this file provided
          PUB_FILENAME = 'data/publist_super_clean.html'

In [121]: # your code here
          def get_pubs(PUB_FILENAME):
              file = open(PUB_FILENAME).read()
              return(file)

In [122]: # your code here
          prof_pubs = get_pubs(PUB_FILENAME)

In [ ]: # 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 [124]: # your code here
          count_author = re.findall(r'\bC.\s*M.\s*Friend\b', prof_pubs)
          total_author = 0
          for i in count_author:
              total_author += 1
          print(total_author)
```
```

5

3.3 Find all unique journals and copy ...

In [125]:
```
# your code here
journals_list = re.findall(r'<I>(.+?)</I>', prof_pubs)
print(journals_list)
len(journals_list)
```

```
['PHYSICAL REVIEW B ', 'PHYSICAL REVIEW B ', 'PHYSICAL REVIEW B ', 'PHYSICAL REVIEW B ', 'Phil
```

Out[125]: 46

In [126]:
```
# check your code
journals=[]
for x in listofjournals:
    if x not in journals:
        journals.append(x)
    journals = sorted(journals)
```

In [127]:
```
# check your code
journals
```

Out[127]:
```
['2010 ACM/IEEE International Conference for High Performance',
 'ACSNano.',
 'Ab initi',
 '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.',
 'NanoLett.',
 'New J. Phys.',
 'New Journal of Physics',
 'PHYSICAL REVIEW B',
 'Phil. Trans. R. Soc. A',
```

```
'Phys. Rev. B',
'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.', 'Ab initio', '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 [129]: # your code here
          authors_list = re.findall(('<BR>(.*) '), prof_pubs)
          pub_authors = [re.sub('^ ',' ',a) for a in authors_list]
```

```
In [132]: # 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
JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, Sheng
Jan M. Knaup, Han Li, Joost J. Vlassak, and Efthimios
Martin Heiss, Sonia Conesa-Boj, Jun Ren, Hsiang-Han Tseng, Adam
Simone Melchionna, Efthimios Kaxiras, Massimo Bernaschi and Sauro Succi,
J R Maze, A Gali, E Togan, Y Chu, A
Kejie Zhao, Wei L. Wang, John Gregoire, Matt Pharr, Zhigang
Masataka Katono, Takeru Bessho, Sheng Meng, Robin Humphry-Baker, Guido
Thomas D. Kuhne, Tod A. Pascal, Efthimios Kaxiras, and Yousung
Sheng Meng, Efthimios Kaxiras, Md. K. Nazeeruddin, and Michael
Bingjun Xu, Jan Haubrich, Thomas A. Baker, Efthimios Kaxiras, and Cynthia M.
Jun Ren, Sheng Meng, Yi-Lin Wang, Xu-Cun Ma, Qi-Kun Xue, Efthimios
Jan Haubrich, Efthimios Kaxiras, and Cynthia M.
Thomas A. Baker, Bingjun Xu, Stephen C. Jensen, Cynthia M. Friend and Efthimios
Youdong Mao, Wei L. Wang, Dongguang Wei, Efthimios Kaxiras, and Joseph G.
H. Li, J.M. Knaup, E. Kaxiras and J.J.
W.L. Wang and E.
L.A. Agapito, N. Kioussis and E.
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.
S. Meng and E.
```

```
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.
M. Fyta, S. Melchionna, M. Bernaschi, E. Kaxiras and S.
E.M. Kotsalis, J.H. Walther, E. Kaxiras and P. Koumoutsakos,
C.E. Lekka, J. Ren, S. Meng and E.
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.
T.A. Baker, C.M. Friend and E.
T.A. Baker, C.M. Friend and E.
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,
```