CS109A Introduction to Data Science

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/
HTML(styles)
```

Out[1]:

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.

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 requests
import json
```

Help a professor parse their publications and extract information.

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/ (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 dot
s:\n Ab initio calculations"',
 '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 (https://www.dataquest.io/blog/web-scraping-tutorial-python/).
- More about the BibTex format (http://www.bibtex.org).

Answers

In [4]:
```
# import the necessary libraries
from bs4 import BeautifulSoup
import urllib.request
```

1.1 Write a function called `make_soup` ...

In [5]:
```
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, encoding="utf-8") as fd:
        data=fd.read()
        soup=BeautifulSoup(data,'html.parser')
    return(soup)
```

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

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" na
me="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 n
anoribbons"</a>
<br/>Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang
 Zhang, Mark Ming-Cheng Cheng,
<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 n
anoparticles"</a>
<br/>JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, Sheng Meng,
<i>PHYSICAL REVIEW B </i> <b>84</b>,  035325 (2011)
<br/>
</li>
</ol>

...
```

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

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

In [7]:
```
# clear output before making pdf
# your code here

def get_author(soup_node):
    author = soup_node.contents[4] if soup_node.contents[4] else 'NA'
    return(author.strip().strip('\n').strip(','))

def get_title(soup_node):
    title = soup_node.select_one('a').text if soup_node.select_one('a').text else
    return(title.strip('\n').strip('"'))

def get_url(soup_node):
    url = soup_node.select_one('a').get('href') if soup_node.select_one('a').get(
    return(url.strip())

def get_journal(soup_node):
    journal = soup_node.select_one('i').text if soup_node.select_one('i').text el
    return(journal.strip())

def get_volume(soup_node):
    volume = soup_node.select_one('b').text if soup_node.select_one('b') else 'NA
    return(volume.strip())
```

In [8]:
```
# your code here
def make_dict(soup):
    soup_nodes = soup.select('li')
    return [
    dict(author=get_author(soup_node),
        title=get_title(soup_node),
        URL=get_url(soup_node),
        journal=get_journal(soup_node),
        volume=get_volume(soup_node)
        )
        for soup_node in soup_nodes
    ]

soup_dict = make_dict(soup)
make_dict(soup)[:2]
```

Out[8]:
```
[{'author': 'Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiya
ng Zhang, Mark Ming-Cheng Cheng',
  'title': 'Approaching the intrinsic band gap in suspended high-mobility graph
ene nanoribbons',
  'URL': 'Papers/2011/PhysRevB_84_125411_2011.pdf',
  'journal': 'PHYSICAL REVIEW B',
  'volume': '84'},
 {'author': 'JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, Sheng Meng',
  'title': 'Effect of symmetry breaking on the optical absorption of semiconduc
tor nanoparticles',
  'URL': 'Papers/2011/PhysRevB_84_035325_2011.pdf',
  'journal': 'PHYSICAL REVIEW B',
  'volume': '84'}]
```

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 [9]:
```
# your code here
publist = ''
for s in soup_dict:
    publist += """
    @article{
    author="%s",
    title="%s",
    URL="%s",
    journal="%s",
    volume=%s
    }

    """ % (s['author'], s['title'], s['URL'], s['journal'], s['volume'],)
```

In [10]:
```
# your code here
with open('publist.bib', 'w', encoding="utf-8") as bibfile:
    bibfile.write(publist)
```

In [ ]:
```
# 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 Kiouss
is, Yiyang Zhang, Mark Ming-Cheng Cheng",
     title = "Approaching the intrinsic band gap in suspended high-mobili
ty graphene nanoribbons",
     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 meet
s 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 [12]:
```
# make sure you use head() when printing the dataframe
# your code here
df=pd.DataFrame.from_dict(soup_dict)
df.head()
```

Out[12]:

	URL	author	journal	title	volume
0	Papers/2011/PhysRevB_84_125411_2011.pdf	Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nic...	PHYSICAL REVIEW B	Approaching the intrinsic band gap in suspende...	84
1	Papers/2011/PhysRevB_84_035325_2011.pdf	JAdam Gali, Efthimios Kaxiras, Gergely T. Zima...	PHYSICAL REVIEW B	Effect of symmetry breaking on the optical abs...	84
2	Papers/2011/PhysRevB_83_054204_2011.pdf	Jan M. Knaup, Han Li, Joost J. Vlassak, and Ef...	PHYSICAL REVIEW B	Influence of CH2 content and network defects o...	83
3	Papers/2011/PhysRevB_83_045303_2011.pdf	Martin Heiss, Sonia Conesa-Boj, Jun Ren, Hsian...	PHYSICAL REVIEW B	Direct correlation of crystal structure and op...	83
4	Papers/2011/PhilTransRSocA_369_2354_2011.pdf	Simone Melchionna, Efthimios Kaxiras, Massimo ...	Phil. Trans. R. Soc. A	Endothelial shear stress from large-scale bloo...	369

In [13]:
```
# your code here
df.to_csv("publist.csv", index=False, header=True, sep='\t')
```

In [14]:
```
# your code here
f = pd.read_csv('publist.csv', sep = '\t')
f.head()
```

Out[14]:

	URL	author	journal	title	volume
0	Papers/2011/PhysRevB_84_125411_2011.pdf	Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nic...	PHYSICAL REVIEW B	Approaching the intrinsic band gap in suspende...	84.0
1	Papers/2011/PhysRevB_84_035325_2011.pdf	JAdam Gali, Efthimios Kaxiras, Gergely T. Zima...	PHYSICAL REVIEW B	Effect of symmetry breaking on the optical abs...	84.0
2	Papers/2011/PhysRevB_83_054204_2011.pdf	Jan M. Knaup, Han Li, Joost J. Vlassak, and Ef...	PHYSICAL REVIEW B	Influence of CH2 content and network defects o...	83.0
3	Papers/2011/PhysRevB_83_045303_2011.pdf	Martin Heiss, Sonia Conesa-Boj, Jun Ren, Hsian...	PHYSICAL REVIEW B	Direct correlation of crystal structure and op...	83.0
4	Papers/2011/PhilTransRSocA_369_2354_2011.pdf	Simone Melchionna, Efthimios Kaxiras, Massimo ...	Phil. Trans. R. Soc. A	Endothelial shear stress from large-scale bloo...	369.0

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

Overview

In this part, your goal is to extract information from IMDb's Top 100 Stars for 2017 (https://www.imdb.com/list/ls025814950/ (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/ (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 detai
ls
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.

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.

Resources

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

Answers

In [15]:
```
import requests
```

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

In [16]:
```
# your code here
url = 'https://www.imdb.com/list/ls025814950/'
my_page = requests.get(url)
#print(my_page.text[:200])
#print(my_page.content[:200])
#print(my_page.status_code)
```

Answer:

my_page.text executes the string commands such as '\n' and returns a cleaner verion of my_page.content. It is the content of the response in Unicode.

my_page.content gives the entire content on in the HTML file without executing any string commands. Itis the content of the response in bytes.

my_page.status_code is the response code after making the get call to indicate whether it was successful or not. In this case the output is 200, which is the HTTP status code for OK.

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

In [17]:
```
# your code here
star_soup=BeautifulSoup(my_page.text, '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 data:

    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 d
etails
    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 [19]:
```
soup_node = star_soup.select_one('.lister-item')
```

In [20]:
```
# your code here
def get_name(soup_node):
    name = soup_node.select_one('.lister-item-content').select_one('h3').select_o
    return(name.strip('\n'))

def get_gender(soup_node):
    gender = soup_node.select_one('.lister-item-content').select_one('p').text.st
    if gender[0:5] == 'Actor':
        gender=0
    elif gender[0:7] == 'Actress':
        gender=1
    else: 'NA'
    return(gender)

def get_url2(soup_node):
    h='https://www.imdb.com'
    url = soup_node.select_one('a').get('href') if soup_node.select_one('a').get(
    url2=h+url
    return(url2)

def get_page(soup_node):
    url2=get_url2(soup_node)
    star_page = requests.get(url2)
    star_page_soup = BeautifulSoup(star_page.text,'html.parser')
    return(star_page_soup)
```

In [21]:
```
def parse_stars(star_soup):
    soup_nodes = star_soup.select('.lister-item')
    return [
    dict(name=get_name(soup_node),
        gender=get_gender(soup_node),
        url=get_url2(soup_node),
        page=get_page(soup_node)
        )
        for soup_node in soup_nodes
    ]

starlist = parse_stars(star_soup)
```

This should give you 100

In [22]:
```
len(starlist)
```

Out[22]: 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[0]
```

Your output should look like this:

```
{'name': 'Gal Gadot',
 'gender': 1,
 'url': 'https://www.imdb.com/name/nm2933757?ref_=nmls_hd',
 'page':
 <!DOCTYPE html>

 <html xmlns:fb="http://www.facebook.com/2008/fbml" xmlns:og="http://ogp.
me/ns#">
 <head>
 <meta charset="utf-8"/>
 <meta content="IE=edge" http-equiv="X-UA-Compatible"/>
 <meta content="app-id=342792525, app-argument=imdb:///name/nm2933757?src
=mdot" name="apple-itunes-app"/>
 <script type="text/javascript">var IMDbTimer={starttime: new Date().getT
ime(),pt:'java'};</script>
 <script>
     if (typeof uet == 'function') {
       uet("bb", "LoadTitle", {wb: 1});
     }
 </script>
 <script>(function(t){ (t.events = t.events || {})["csm_head_pre_title"]
 = new Date().getTime(); })(IMDbTimer);</script>

...
```

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 data:

    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 [24]:
```
def get_yr_born(s):
    s=s['page']
    born_info = s.find('div',{'id':'name-born-info'})
    year_born = born_info.find_all('a')[1].text if born_info else 'NA'
    return(year_born)

def get_credits(s):
    credits = s['page'].find('div', class_ = 'head').text.strip('\n')
    end=credits.find('credits')
    start=credits.find('(')
    return(int(credits[start+1:end-1]))

def get_first_movie(s):
    movie_details = s['page'].find('div', class_ = 'filmo-category-section' )
    first_movie_detials = movie_details.findAll('div', class_ = 'filmo-row')[-1]
    first_movie = first_movie_detials.select_one('b').find('a').text
    return(first_movie)

def get_first_movie_year(s):
    movie_details = s['page'].find('div', class_ = 'filmo-category-section' )
    first_movie_detials = movie_details.findAll('div', class_ = 'filmo-row')[-1]
    span=first_movie_detials.find('span', class_ = 'year_column') if first_movie_
    year_first_movie = span.text.strip('\n')[1:5]
    return(year_first_movie)
```

In [25]:
```
def create_star_table(starlist: list) -> list:
    return[
        dict(
            star_name=s['name'].strip(),
            gender=s['gender'],
            year_born= get_yr_born(s),
            first_movie = get_first_movie(s),
            year_first_movie = get_first_movie_year(s),
            credits = get_credits(s)
            )
        for s in starlist
    ]
star_table = create_star_table(starlist)
```

In [ ]:
```
# check your code
# 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 [27]:
```
# your code here
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 [ ]:
```
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 [29]:
```
# your code here
with open("data/staff_starinfo.json", "r") as fd:
    star_table = json.load(fd)
```

In [30]:
```
fr=pd.DataFrame.from_dict(star_table)
fr.head()
```

Out[30]:

	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 [31]:
```
fr.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 [32]:
```
def clean(frame):
    frame.year_born=frame.year_born.astype(int)
    frame.credits=frame.credits.astype(int)
    frame.gender=frame.gender.astype('category')
    frame['year_first_movie']=frame.year_first_movie.str[:4]
    frame.year_first_movie=frame.year_first_movie.astype(int)
    frame['age_at_first_movie']=frame.year_first_movie-frame.year_born
    return(frame)

frame=clean(fr)
frame.head()
```

Out[32]:

	credits	first_movie	gender	name	year_born	year_first_movie	age_at_first_movie
0	25	Bubot	1	Gal Gadot	1985	2007	22
1	55	Tommaso	0	Tom Hardy	1977	2001	24
2	17	Doctors	1	Emilia Clarke	1986	2009	23
3	51	All My Children	1	Alexandra Daddario	1986	2002	16
4	30	Järngänget	0	Bill Skarsgård	1990	2000	10

In [33]:
```
# your code here
frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 7 columns):
credits               100 non-null int32
first_movie           100 non-null object
gender                100 non-null category
name                  100 non-null object
year_born             100 non-null int32
year_first_movie      100 non-null int32
age_at_first_movie    100 non-null int32
dtypes: category(1), int32(4), object(2)
memory usage: 3.4+ KB
```

Answer:

As we can see in dataframe 'fr', there are rows with a range of years for the first appearance (most likely a TV series). In such cases, we would like to keep the minimum of the range since that gives us when a particlar star began their career. Also, 'year_born' and 'year_first_movie' are strings, so they have to be converted to integer before calculating 'age_at_first_movie'. Also, 'credits' was converted to integer and 'gender' was converted to category.

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 [34]:
```
# your code here
performers_at_17 = len(frame[frame.age_at_first_movie==17])
print('%s performers made their first movie at 17' %performers_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 [35]:
```
# your code here

dfgb_age = frame.groupby('age_at_first_movie')
#child_actors = len(dfgb_age[dfgb_age.age_at_first_movie<12])
dfgb_age.count().head(12)
```

Out[35]:

age_at_first_movie	credits	first_movie	gender	name	year_born	year_first_movie
-1	1	1	1	1	1	1
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
5	1	1	1	1	1	1
6	1	1	1	1	1	1
7	1	1	1	1	1	1
8	3	3	3	3	3	3
9	3	3	3	3	3	3
10	5	5	5	5	5	5
11	2	2	2	2	2	2
12	1	1	1	1	1	1

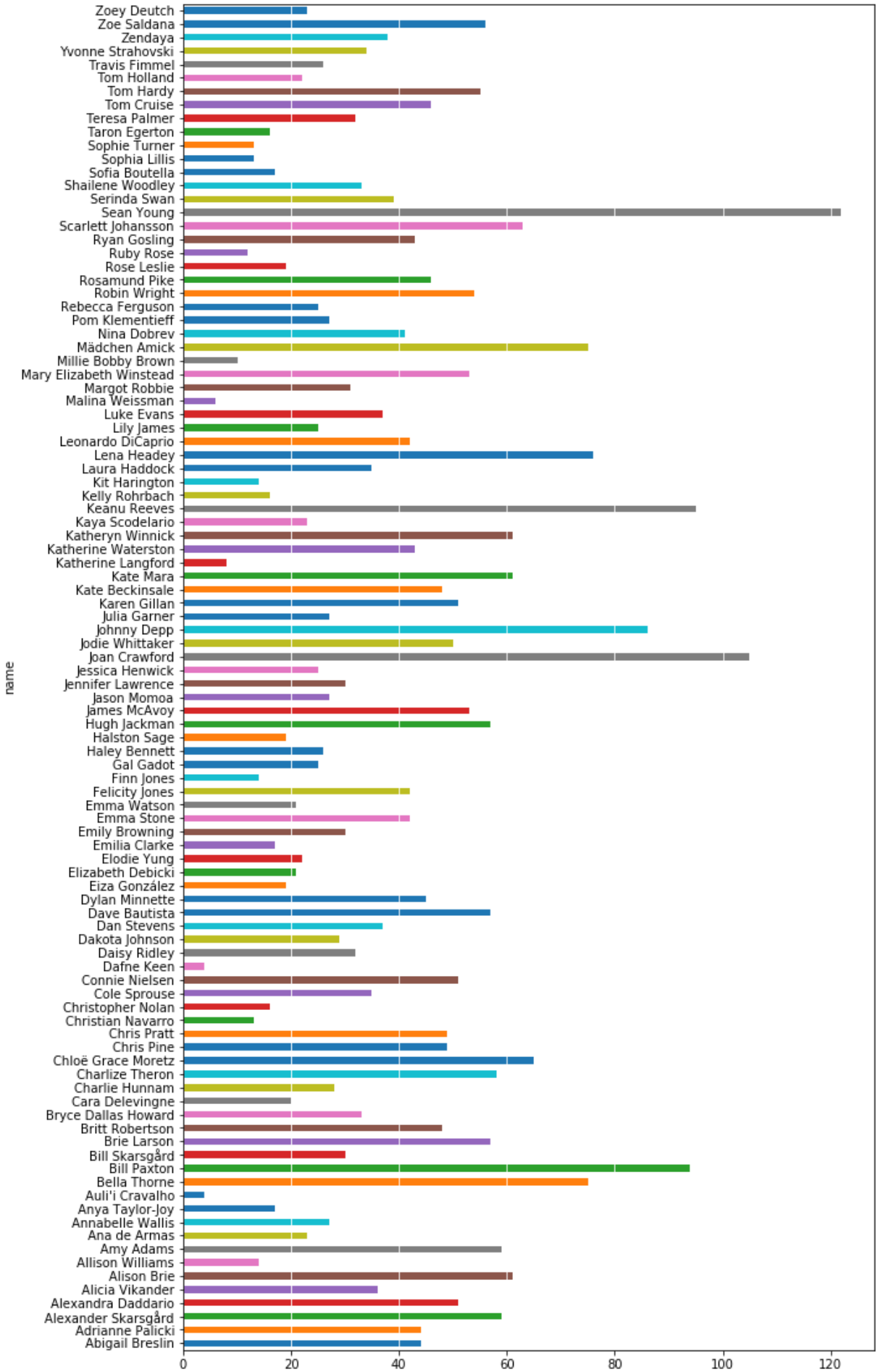
There is one actor with age -1, which is clearly wrong. So we need to exclude that from our calculations.

In [36]:
```
less_12 = frame[frame.age_at_first_movie<12]
child_actors = len(less_12[less_12.age_at_first_movie>=0])
print('%s performers started as child actors'%child_actors)
```

```
19 performers started as child actors
```

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

In [37]:
```
# your code here
plt_df = frame.groupby('name').credits.sum()
plt_df.plot(kind='barh', figsize=(10,20))
plt.grid(axis = 'x', color ='white', linestyle='-')
```


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

From the above bar chart, we see that Sean Young has the most number of credits (>120). So he is the most prolific performer by our definition. We can also verify that below.

In [38]:
```
# your code here
prolific_name = frame[frame['credits']==np.max(frame['credits'])]['name'].iloc[0]
prolific_credits = frame[frame['credits']==np.max(frame['credits'])]['credits'].i
print('The most prolific performer is %s with %s credits'%(prolific_name, prolifi
```

```
The most prolific performer is Sean Young with 122 credits
```

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.

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.

Resources

- **Regular expressions:** a) https://docs.python.org/3.3/library/re.html (https://docs.python.org/3.3/library/re.html), b) https://regexone.com (https://regexone.com), and c) https://docs.python.org/3/howto/regex.html (https://docs.python.org/3/howto/regex.html).
- **HTML:** if you are not familiar with HTML see https://www.w3schools.com/html/ (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 (https://www.w3schools.com/js/js_htmldom.asp).

Answers

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

In [39]:
```
# first import the necessary reg expr library
import re
```

In [40]:
```
# use this file provided
PUB_FILENAME = 'data/publist_super_clean.html'
```

In [41]:
```
# your code here

def get_pubs(filename):
    with open(filename, 'r') as myfile:
        prof_pubs = myfile.read()
    return prof_pubs
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)

```
<LI>
<A HREF="Papers/2011/PhysRevB_84_125411_2011.pdf" target="paper244">
&quot;Approaching the intrinsic band gap in suspended high-mobility graph
ene nanoribbons&quot;</A>
<BR>Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang
 Zhang, Mark Ming-Cheng Cheng,
<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">
&quot;Effect of symmetry breaking on the optical absorption of semiconduc
tor nanoparticles&quot;</A>
<BR>JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, Sheng Meng,
<I>PHYSICAL REVIEW B </I> <b>84</b>,  035325 (2011)
<BR>
</LI>
</OL>

<OL START=242>
<LI>
<A HREF="Papers/2011/PhysRevB_83_054204_2011.pdf" target="paper242">
&quot;Influence of CH2 content and network defects on the elastic propert
ies of organosilicate glasses&quot;</A>
<BR>Jan M. Knaup, Han Li, Joost J. Vlassak, and Efthimios Kaxiras,
<I>PHYSICAL REVIEW B </I> <b>83</b>,  054204 (2011)
<BR>
</LI>
</OL>
```

3.2 Calculate how many times the author ...

In [43]:
```
# your code here
pattern_auth = re.compile("C.M. Friend")
auth_ls = re.findall(pattern_auth, prof_pubs)
auth_ct = len(auth_ls)
print("The author named C.M. Friend appears %s times."%auth_ct)
```

```
The author named C.M. Friend appears 5 times.
```

3.3 Find all unique journals and copy ...

In [44]:
```
# your code here

def strip(p: list):
    prof_journal=[]
    for i in p:
        j=i.replace('NanoLett', 'Nano Lett')
        k=j.replace('Phys. Rev. B', 'PHYSICAL REVIEW B')
        l=k.replace('New J. Phys.', 'New Journal of Physics')
        prof_journal.append(l.strip())
    return(prof_journal)

pattern_I = re.compile("<I>(.*?)</I>", re.I)
prof_I = re.findall(pattern_I, prof_pubs)

prof_journal = strip(prof_I)

unique = np.unique(prof_journal)
journals = unique.tolist()
```

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

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 [46]:
```
# your code here
pattern_authors = re.compile("""<BR>(.+?)
<I>""", re.I)
pub_authors = re.findall(pattern_authors, prof_pubs)
len(pub_authors)
```

Out[46]: 45

In [47]:
```
# check your code: print the list of strings containing the author(s)' names
for item in pub_authors:
    print (item.strip())
```

```
Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang Zhang, Mar
k Ming-Cheng Cheng,
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 Rothenbe
rger,
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. Fr
iend,
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. Sodro
ski,
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. Ramanathan,
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 C.L. Feldman,
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. Kaxir
as, S. Succi, P.H. Stone and C.L. Feldman,
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 Zhan
g, Mark Ming-Cheng Cheng,
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,
```

In [ ]: