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 [2]: ## 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[2]: <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 [3]: # 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 [4]: # 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 calculat
 '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 [5]: # import the necessary libraries
        from bs4 import BeautifulSoup
```

1.1 Write a function called `make_soup` ...

```
In [6]: 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, 'r') as f:
            html_text = f.read()
        return BeautifulSoup(html_text, 'html.parser')
```

```
In [13]: # check your code - print the BS object, you should get a familiar HTML page as text
         # 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" 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-Chen
<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>
```

...

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

```
In [8]: # clear output before making pdf
        # your code here
        def list_of_html_citations(soup):
            body = soup.find('body')
            return body.find_all('ol')

        def clean_text(text):
            text_to_strip = ['\n',' ',',']
            for t in text_to_strip:
                text = text.strip(t)
            text = text.replace("\n"," ")
            return text

        def get_title(citation):
            text = citation.find('a').get_text()
            return clean_text(text)

        def get_author(citation):
            text = citation.find('li').contents[4]
            return clean_text(text)

        def get_url(citation):
            text = citation.find('a').get('href')
            return clean_text(text)

        def get_journal(citation):
            text = citation.find('i').get_text()
            return clean_text(text)

        def get_volume(citation):
            tag = citation.find('b')
            if tag is None:
                return None
            else:
                text = tag.get_text()
                return clean_text(text)

        def create_citation_dict(citation):
            return {
                'author': get_author(citation),
                'title': get_title(citation),
                'URL': get_url(citation),
                'journal': get_journal(citation),
                'volume': get_volume(citation)
            }

        def parse_for_citations(soup):
            return [create_citation_dict(c) for c in list_of_html_citations(soup)]
```

```
In [9]: # your code here
        parse_for_citations(soup)
```

```
Out[9]: [{'author': 'Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang Zhan
          'title': '"Approaching the intrinsic band gap in suspended high-mobility graphene na
          '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 semiconductor na
          'URL': 'Papers/2011/PhysRevB_84_035325_2011.pdf',
          'journal': 'PHYSICAL REVIEW B',
          'volume': '84'},
         {'author': 'Jan M. Knaup, Han Li, Joost J. Vlassak, and Efthimios Kaxiras',
          'title': '"Influence of CH2 content and network defects on the elastic properties of
          'URL': 'Papers/2011/PhysRevB_83_054204_2011.pdf',
          'journal': 'PHYSICAL REVIEW B',
          'volume': '83'},
         {'author': 'Martin Heiss, Sonia Conesa-Boj, Jun Ren, Hsiang-Han Tseng, Adam Gali',
          'title': '"Direct correlation of crystal structure and optical properties in wurtzit
          'URL': 'Papers/2011/PhysRevB_83_045303_2011.pdf',
          'journal': 'PHYSICAL REVIEW B',
          'volume': '83'},
         {'author': 'Simone Melchionna, Efthimios Kaxiras, Massimo Bernaschi and Sauro Succi',
          'title': '"Endothelial shear stress from large-scale blood flow simulations"',
          'URL': 'Papers/2011/PhilTransRSocA_369_2354_2011.pdf',
          'journal': 'Phil. Trans. R. Soc. A',
          'volume': '369'},
         {'author': 'J R Maze, A Gali, E Togan, Y Chu, A Trifonov',
          'title': '"Properties of nitrogen-vacancy centers in diamond: the group theoretic ap
          'URL': 'Papers/2011/NewJPhys_13_025025_2011.pdf',
          'journal': 'New Journal of Physics',
          'volume': '13'},
         {'author': 'Kejie Zhao, Wei L. Wang, John Gregoire, Matt Pharr, Zhigang Suo',
          'title': '"Lithium-Assisted Plastic Deformation of Silicon Electrodes in Lithium-Ion
          'URL': 'Papers/2011/NanoLett_11_2962-2967_2011.pdf',
          'journal': 'Nano Lett.',
          'volume': '11'},
         {'author': 'Masataka Katono, Takeru Bessho, Sheng Meng, Robin Humphry-Baker, Guido Ro
          'title': '"D--A Dye System Containing Cyano-Benzoic Acid as Anchoring Group for Dye-
```

```
 'URL': 'Papers/2011/Langmuir_27_14248_2011.pdf',
 'journal': 'Langmuir',
 'volume': '27'},
{'author': 'Thomas D. Kuhne, Tod A. Pascal, Efthimios Kaxiras, and Yousung Jung',
 'title': '"New Insights into the Structure of the Vapor/Water Interface from Large-Sc
 'URL': 'Papers/2011/JPhysChemLett_2_105-113_2011.pdf',
 'journal': 'J. Phys. Chem. Lett.',
 'volume': '2'},
{'author': 'Sheng Meng, Efthimios Kaxiras, Md. K. Nazeeruddin, and Michael Gratzel',
 'title': '"Design of Dye Acceptors for Photovoltaics from First-Principles Calculatio
 'URL': 'Papers/2011/JPhysChemC_115_9276-9282_2011.pdf',
 'journal': 'J. Phys. Chem. C',
 'volume': '115'},
{'author': 'Bingjun Xu, Jan Haubrich, Thomas A. Baker, Efthimios Kaxiras, and Cynthia
 'title': '"Theoretical Study of O-Assisted Selective Coupling of Methanol on Au(111)"
 'URL': 'Papers/2011/JPhysChemC_115_3703-3708_2011.pdf',
 'journal': 'J. Phys. Chem. C',
 'volume': '115'},
{'author': 'Jun Ren, Sheng Meng, Yi-Lin Wang, Xu-Cun Ma, Qi-Kun Xue, Efthimios Kaxiras
 'title': '"Properties of copper (fluoro-)phthalocyanine layers deposited on epitaxial
 'URL': 'Papers/2011/JChemPhys_134_194706_2011.pdf',
 'journal': 'J. Chem. Phys.',
 'volume': '134'},
{'author': 'Jan Haubrich, Efthimios Kaxiras, and Cynthia M. Friend',
 'title': '"The Role of Surface and Subsurface Point Defects for Chemical Model Studie
 'URL': 'Papers/2011/Chemistry_17_4496-4506_2011.pdf',
 'journal': 'Chem. Eur. J.',
 'volume': '17'},
{'author': 'Thomas A. Baker, Bingjun Xu, Stephen C. Jensen, Cynthia M. Friend and Eft
 'title': '"Role of defects in propene adsorption and reaction on a partially O-cover
 'URL': 'Papers/2011/CatalSciTechnol_1_1166_2011.pdf',
 'journal': 'Catal. Sci. Technol.',
 'volume': '1'},
{'author': 'Youdong Mao, Wei L. Wang, Dongguang Wei, Efthimios Kaxiras, and Joseph G.
 'title': '"Graphene Structures at an Extreme Degree of Buckling"',
 'URL': 'Papers/2011/ACSNano_5_1395-1400_2011.pdf',
 'journal': 'ACSNano.',
 'volume': '5'},
{'author': 'H. Li, J.M. Knaup, E. Kaxiras and J.J. Vlassak',
 'title': '"Stiffening of organosilicate glasses by organic cross-linking"',
 'URL': 'Papers/ActaMater_59_44-52_2011.pdf',
 'journal': 'Acta Mater.',
 'volume': '59'},
{'author': 'W.L. Wang and E. Kaxiras',
 'title': '"Graphene hydrate: theoretical prediction of a new insulating  form of grap
 'URL': 'Papers/NewJPhys_12_125012_2010.pdf',
 'journal': 'New J. Phys.',
 'volume': '12'},
{'author': 'L.A. Agapito, N. Kioussis and E. Kaxiras',
 'title': '"Electric-field control of magnetism in graphene quantum dots:  Ab initio c
 'URL': 'Papers/PhysRevB_82_201411_2010.pdf',
 'journal': 'Phys. Rev. B',
 'volume': '82'},
{'author': 'A. Peters, S. Melchionna, E. Kaxiras, J. Latt, J. Sircar, S. Succi',
 'title': '"Multiscale simulation of cardiovascular flows on the IBM Bluegene/P:  full
 'URL': 'Papers/IEEE-SC10_2010.pdf',
 'journal': '2010 ACM/IEEE International Conference for High Performance',
 'volume': None},
{'author': 'J. Ren, E. Kaxiras and S. Meng',
 'title': '"Optical properties of clusters and molecules from real-time time-dependent
 'URL': 'Papers/MolPhys_108_1829-1844_2010.pdf',
 'journal': 'Molec. Phys.',
 'volume': '108'},
{'author': 'T.A. Baker, E. Kaxiras and C.M. Friend',
 'title': '"Insights from Theory on the Relationship Between Surface Reactivity and Go
 'URL': 'Papers/TopicsCatal_53_365-377_2010.pdf',
 'journal': 'Top. Catal.',
 'volume': '53'},
{'author': 'H.P. Chen, R.K. Kalia, E. Kaxiras, G. Lu, A. Nakano, K. Nomura',
 'title': '"Embrittlement of Metal by Solute Segregation-Induced Amorphization"',
 'URL': 'Papers/PhysRevLett_104_155502_2010.pdf',
 'journal': 'Phys. Rev. Lett.',
 'volume': '104'},
{'author': 'S. Meng and E. Kaxiras',
 'title': '"Electron and Hole Dynamics in Dye-Sensitized Solar Cells: Influencing Fact
 'URL': 'Papers/NanoLett_10_1238-1247_2010.pdf',
 'journal': 'NanoLett.',
 'volume': '10'},
{'author': 'C.L. Chang, S.K.R.S. Sankaranarayanan, D. Ruzmetov, M.H. Engelhard, E. Kax
 'title': '"Compositional tuning of ultrathin surface oxides on metal and alloy substr
 'URL': 'Papers/PhysRevB_81_085406_2010.pdf',
 'journal': 'Phys. Rev. B',
 'volume': '81'},
{'author': 'T.A. Baker, C.M. Friend and E. Kaxiras',
 'title': '"Local Bonding Effects in the Oxidation of CO on Oxygen-Covered Au(111) fro
 'URL': 'Papers/JChemTheComp_6_279-287_2010.pdf',
 'journal': 'J. Chem. Theory Comput.',
 'volume': '6'},
{'author': 'S. Melchionna, M. Bernaschi, S. Succi, E. Kaxiras, F.J. Rybicki, D. Mitsou
 'title': '"Hydrokinetic approach to large-scale cardiovascular blood flow"',
 'URL': 'Papers/CompPhysComm_181_462-472_2010.pdf',
 'journal': 'Comp. Phys. Comm.',
 'volume': '181'},
{'author': 'M. Bernaschi, M. Fatica, S. Melchionna, S. Succi and E. Kaxiras',
 'title': '"A flexible high-performance Lattice Boltzmann GPU code for the simulations
 'URL': 'Papers/ConcComp_22_1-14_2010.pdf',
 'journal': 'Concurrency Computat.: Pract. Exper.',
 'volume': '22'},
{'author': 'E. Manousakis, J. Ren, S. Meng and E. Kaxiras',
 'title': '"Is the nature of magnetic order in copper-oxides and iron-pnictides  diffe
 'URL': 'Papers/SolStComm_150_62-65_2010.pdf',
 'journal': 'Sol. St. Comm.',
 'volume': '150'},
{'author': 'A. Gali, E. Janzen, P. Deak, G. Kresse and E. Kaxiras',
 'title': '"Theory of Spin-Conserving Excitation of the N-V Center in Diamond"',
 'URL': 'Papers/PhysRevLett_103_186404_2009.pdf',
 'journal': 'Phys. Rev. Lett.',
 'volume': '103'},
{'author': 'S.K.R.S. Sankaranarayanan, E. Kaxiras and S. Ramanathan',
 'title': '"Electric field tuning of oxygen stoichiometry at oxide surfaces: molecula
 'URL': 'Papers/EnEnviSci_2_1196-1204_2009.pdf',
 'journal': 'Energy & Environmental Sci.',
 'volume': '2'},
{'author': 'M. Bernaschi, S. Melchionna, S. Succi, M. Fyta, E. Kaxiras',
 'title': '"MUPHY: A parallel MUlti PHYsics/scale code for high performance  bio-flui
 'URL': 'Papers/CompPhysComm_180_1495-1502_2009.pdf',
 'journal': 'Comp. Phys. Comm.',
 'volume': '180'},
{'author': 'T.A. Baker, B.J. Xu, X.Y. Liu, E. Kaxiras and C.M. Friend',
 'title': '"Nature of Oxidation of the Au(111) Surface: Experiment and Theoretical In
 'URL': 'Papers/JPhysChemC_113_16561-16564_2009.pdf',
 'journal': 'J. Phys. Chem. C',
 'volume': '113'},
{'author': 'F.J. Rybicki, S. Melchionna, D. Mitsouras, A.U. Coskun, A.G. Whitmore, E.
 'title': '"Prediction of coronary artery plaque progression and potential rupture fr
 'URL': 'Papers/IntJCardImag_25_289-299_2009.pdf',
 'journal': 'Int. J. Cardiovasc. Imaging',
 'volume': '25'},
{'author': 'H. Chen, W.G. Zhu, E. Kaxiras, and Z.Y. Zhang',
 'title': '"Optimization of Mn doping in group-IV-based dilute magnetic semiconductor
 'URL': 'Papers/PhysRevB_79_235202_2009.pdf',
 'journal': 'Phys. Rev. B',
 'volume': '79'},
{'author': 'M. Fyta, S. Melchionna, M. Bernaschi, E. Kaxiras and S. Succi',
 'title': '"Numerical simulation of conformational variability in biopolymer transloc
 'URL': 'Papers/JStatMech_2009.pdf',
 'journal': 'J. Stat. Mech: Th. and Exper.',
 'volume': '06'},
{'author': 'E.M. Kotsalis, J.H. Walther, E. Kaxiras and P. Koumoutsakos',
 'title': '"Control algorithm for multiscale flow simulations of water"',
 'URL': 'Papers/PhysRevE_79_045701RC_2009.pdf',
 'journal': 'Phys. Rev. E - Rap. Comm.',
 'volume': '79'},
{'author': 'C.E. Lekka, J. Ren, S. Meng and E. Kaxiras',
'title': '"Structural, Electronic, and Optical Properties of Representative Cu-Flavor
 'URL': 'Papers/JPhysChemB_113_6478_2009.pdf',
 'journal': 'J. Phys. Chem. B',
 'volume': '113'},
{'author': 'W.L. Wang, O.V. Yazyev, S. Meng and E. Kaxiras',
 'title': '"Topological Frustration in Graphene Nanoflakes: Magnetic Order and Spin Lo
 'URL': 'Papers/PhysRevLett_102_157201_2009.pdf',
 'journal': 'Phys. Rev. Lett.',
 'volume': '102'},
{'author': 'A. Gali and E. Kaxiras',
 'title': '"Comment on \'Ab initio Electronic and Optical Properties of the N-V-Center
 'URL': 'Papers/PhysRevLett_102_149703_2009.pdf',
 'journal': 'Ab initio',
 'volume': '102'},
{'author': 'S. Melchionna, M. Bernaschi, M. Fyta, E. Kaxiras and S. Succi',
 'title': '"Quantized biopolymer translocation through nanopores: Departure from simpl
 'URL': 'Papers/PhysRevE_79_030901RC_2009.pdf',
 'journal': 'Phys. Rev. E - Rap. Comm.',
 'volume': '79'},
{'author': 'S.K.R.S. Sankaranarayanan, E. Kaxiras, S. Ramanathan',
 'title': '"Atomistic Simulation of Field Enhanced Oxidation of Al(1000) Beyond the Mo
 'URL': 'Papers/PhysRevLett_102_095504_2009.pdf',
 'journal': 'Phys. Rev. Lett.',
 'volume': '102'},
{'author': 'T.A. Baker, C.M. Friend and E. Kaxiras',
 'title': '"Effects of chlorine and oxygen coverage on the structure of the Au(111) su
 'URL': 'Papers/JChemPhys_130_084701_2009.pdf',
 'journal': 'J. Chem. Phys.',
 'volume': '130'},
{'author': 'T.A. Baker, C.M. Friend and E. Kaxiras',
 'title': '"Atomic Oxygen Adsorption on Au(111) Surfaces with Defects"',
 'URL': 'Papers/JPhysChemC_113_3232_2009.pdf',
 'journal': 'J. Phys. Chem. C',
 'volume': '113'},
{'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'},
{'author': 'E. Manousakis, J. Ren, S. Meng and E. Kaxiras',
 'title': '"Effective Hamiltonian for FeAs-based superconductors"',
 'URL': 'Papers/PhysRevB_78_205112_2008.pdf',
 'journal': 'Phys. Rev. B',
 '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 [10]: # your code here
```

```
        def create_bibtex(citation_list):
            with open("publist.bib", "w") as f:
                for c in citation_list:
                    if c['volume'] is None:
                        f.write('@article{\n\tauthor = "%s"\n\ttitle = %s\n\tURL = "%s"\n\tjou
                            % (c['author'],c['title'],c['URL'],c['journal']))
                    else:
                        f.write('@article{\n\tauthor = "%s"\n\ttitle = %s\n\tURL = "%s"\n\tjou
                            % (c['author'],c['title'],c['URL'],c['journal'],c['volume']))
```

```
In [11]: # your code here
         citation_list = parse_for_citations(soup)
         create_bibtex(citation_list)
```

```
In [14]: # 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, Mark
     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 [457]: # make sure you use head() when printing the dataframe
          # your code here
          def convert_to_pandas(citation_list):
              return pd.DataFrame(citation_list)
          citation_list = parse_for_citations(soup)
          citations_pd = convert_to_pandas(citation_list)
          citations_pd.head()
```

```
Out[457]:                                          URL
          0       Papers/2011/PhysRevB_84_125411_2011.pdf  Ming-Wei Lin, Cheng Ling, Luis A. Ag
          1       Papers/2011/PhysRevB_84_035325_2011.pdf  JAdam Gali, Efthimios Kaxiras, Gerge
          2       Papers/2011/PhysRevB_83_054204_2011.pdf  Jan M. Knaup, Han Li, Joost J. Vlass
          3       Papers/2011/PhysRevB_83_045303_2011.pdf  Martin Heiss, Sonia Conesa-Boj, Jun
          4  Papers/2011/PhilTransRSocA_369_2354_2011.pdf  Simone Melchionna, Efthimios Kaxiras
```

```
In [458]: # your code here
          citations_pd.to_csv("publist.csv", index = False)
```

```
In [459]: !head -3 publist.csv
```

```
URL,author,journal,title,volume
Papers/2011/PhysRevB_84_125411_2011.pdf,"Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Ki
Papers/2011/PhysRevB_84_035325_2011.pdf,"JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, She
```

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 [15]: import requests
```

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

```
In [201]: # your code here
          my_page = requests.get("https://www.imdb.com/list/ls025814950/")
```

your answer here * my_page.text is the decoded html content of the website. Requests automatically decodes the server's content. * my_page.status_code is the status of the HTTP request (i.e. 202 = accepted, 400 = bad request, etc.). More information available here: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes * my_page.content is the html content of the website provided by the server

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

```
In [202]: # your code here
          star_soup = BeautifulSoup(my_page.text, 'html.parser')
```

```
In [15]: # 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 [252]: # your code here
          def list_of_html_profiles(soup):
              return soup.find_all('div', class_ = "lister-item-content")

          def clean_text(text):
              text_to_strip = ['\n',' ',',']
              for t in text_to_strip:
                  text = text.strip(t)
              text = text.replace("\n"," ")
              return text

          def get_name(profile):
              text = profile.find('a').get_text()
              return clean_text(text)

          def get_gender(profile):
              occupation = profile.find('p', class_ = "text-muted text-small").contents[0].str
              if occupation.lower() == 'actor':
                  return 0
              elif occupation.lower() == 'actress':
                  return 1
              else:
                  return 2

          def get_url(profile):
              text = "https://www.imdb.com" + profile.find('a')['href']
              return text

          def get_page(profile):
              url = get_url(profile)
              my_page = requests.get(url)
              time.sleep(10)
              return BeautifulSoup(my_page.text, 'html.parser')

          def create_profile_dict(profile):
              profile_dict =  {
                  'name': get_name(profile),
                  'gender': get_gender(profile),
                  'url': get_url(profile),
                  'page': get_page(profile)
              }
              print("scraped %s" % profile_dict['name'])
              return profile_dict


          def parse_soup(soup):
              return [create_profile_dict(p) for p in list_of_html_profiles(soup)]
          starlist = parse_soup(star_soup)
```

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

This should give you 100

```
In [253]: len(starlist)
```

```
Out[253]: 100
```

```
In [16]: # 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

         # print(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 [286]: starlist_copy = starlist.copy()
```

```
In [371]: def get_year(star_page):
              try:
                  return star_page.find('div', {'id': 'name-born-info'}).find_all('a')[1].text
              except AttributeError:
                  print('missing value')
                  return None

          def get_first_movie(star_page):
              tags = star_page.find('div', class_='filmo-category-section').find_all('a')
              movies = [tag.text for tag in tags if 'pisode' not in tag.text]
              return movies[-1]

          def get_year_first_movie(star_page):
              text = star_page.find('div', class_='filmo-category-section').find_all('span', cl
              return re.sub('[^0-9-]', '', text)

          def get_credits(star_page):
              try:
                  text = star_page.find('div', id = re.compile('filmo-head-act')).text
                  return re.sub('[^0-9]', '', text)
              except AttributeError:
                  print('missing value')
                  return None

          def create_star_dict(old_star_dict):
              star_dict =  {
                  'star_name': old_star_dict['name'],
                  'gender': old_star_dict['gender'],
                  'year_born': get_year(old_star_dict['page']),
                  'first_movie': get_first_movie(old_star_dict['page']),
                  'year_first_movie': get_year_first_movie(old_star_dict['page']),
                  'credits': get_credits(old_star_dict['page'])
              }
              return star_dict

          def create_star_table(starlist: list) -> list:
              return [create_star_dict(old_star_dict) for old_star_dict in starlist]
```

```
In [372]: star_table = create_star_table(starlist)

missing value
missing value
missing value
```

```
In [17]: # 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 [374]: # your code here
          import json
          with open('starinfo.json', 'w') as file:
              json.dump(star_table, file)
```

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

```
In [19]: #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. ...

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.

```
In [392]: # your code here
          frame = pd.read_json('./data/staff_starinfo.json')
          frame.head()
```

```
Out[392]:    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 [393]: # your code here

          #check types
          print("types before cleaning:")
          print(frame.dtypes)

          #take only first year when year_first_movie is of format yyyy-yyyy
          frame.year_first_movie = frame.year_first_movie.apply(lambda x: x[:4])

          #cast year_first_movie to int
          frame.year_first_movie = frame.year_first_movie.astype(int)

          #check types
          print("types after cleaning:")
          print(frame.dtypes)
```

```
types before cleaning:
credits              int64
first_movie         object
gender               int64
name                object
year_born            int64
year_first_movie    object
dtype: object
types after cleaning:
credits              int64
first_movie         object
gender               int64
name                object
year_born            int64
```

```
year_first_movie     int64
dtype: object
```

```
In [397]: # your code here
          #check for missing values
          print('number of NA values: %i' %frame.isna().sum().sum())

number of NA values: 0
```

```
In [403]: # your code here
          frame['age_at_first_movie'] = frame['year_first_movie'] - frame['year_born']
          frame[frame['age_at_first_movie'] < 0]

Out[403]:     credits     first_movie  gender          name  year_born  year_first_movie  age_a
          63       32  Only Yesterday       1  Daisy Ridley       1992              1991
```

your answer here
Daisy Ridley's first movie was apparently released before she was born.
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 [409]: # your code here
          print("%i performers made their first movie at 17" % frame.groupby('age_at_first_movi

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.

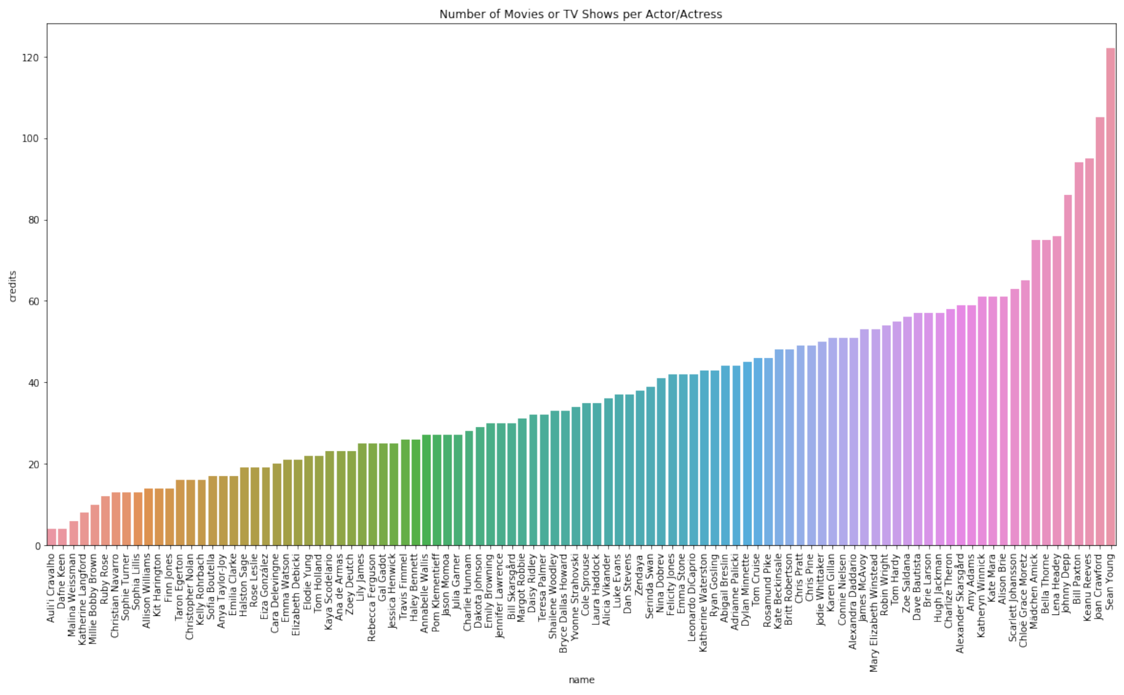
```
In [414]: # your code here
          print("%i performers started as child actors" % frame[frame.age_at_first_movie < 12]

20 performers started as child actors
```

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

```
In [437]: # your code here
          frame = frame.sort_values('credits')
          plt.subplots(figsize=(20,10))
          ax = sns.barplot(x = frame.name, y = frame.credits)
          ax.set_title('Number of Movies or TV Shows per Actor/Actress')
          ax.set_xticklabels(frame.name, rotation=90)
          plt.figure(figsize=(40,10))

Out[437]: <Figure size 2880x720 with 0 Axes>
```



```
<Figure size 2880x720 with 0 Axes>
```

```
In [449]: # your code here
          print("The most prolific actor is %s" % list(frame[frame.credits == max(frame.credit
```

```
The most prolific actor 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 [36]: # first import the necessary reg expr library
         import re
```

```
In [37]: # use this file provided
         PUB_FILENAME = 'data/publist_super_clean.html'
```

```
In [460]: # your code here
          def get_pubs(filename):
              with open(filename, "r") as f:
                  return f.read()
```

```
In [462]: # your code here
          prof_pubs = get_pubs(PUB_FILENAME)
```

```
In [20]: # 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 [524]: # your code here
          print("C.M. Friend appears %i times"%prof_pubs.count("C.M. Friend"))
```

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

3.3 Find all unique journals and copy ...

```
In [525]: # your code here
          journals_with_tags = set(re.findall("<I>.*</I>",prof_pubs))
          journals = [name[3:-5] for name in journals_with_tags]
          journals.sort()
          journals.remove('NanoLett.')
          journals.remove('New J. Phys.')
```

```
In [526]: journals
```

```
Out[526]: ['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.',
           '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 [527]: # your code here
          pub_authors_with_tags = re.findall("<BR>.*\n<I>",prof_pubs)
          pub_authors = [authors[4:-5].strip(" ,") for authors in pub_authors_with_tags]
```

```
In [528]: # 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,
```