Purpose

Often times in academic and corporate settings, engineers and professionals seek to derive meaningful information from data to meet a given objective. Before one can begin cleaning or analyzing data, they first have to collect it. In some cases, data collection is as simple as downloading a CSV file, but this is infeasible for many applications. For example, an application that displays weather data should collect live data through an API instead of downloading a CSV file. Data exchange allows data to be shared between different computer programs. This assignment will give you experience in how data is exchanged by having you input, manipulate, and export data using different data exchange formats.

Skills

This assignment will give you practice with comma separated value (csv) files, Javascript Object Notation (json) files, scraping data from a website using the bs4 module, and accessing data stored in API using the requests module, all while transforming the raw data into the desired format. You will learn how to locate and make meaning from the information in these files and also create new files stored in each of these formats.

Knowledge

By working extensively with these types of data exchange formatted files, you will begin to see how data can be stored in many ways. In the future you will be better able to understand the differences between data exchange types and have a better understanding of how these data formats are selected to fit a particular purpose.

Important

1. Due Date: **03/13/2022 at 11:59 pm**
2. This homework is graded out of **100** points.
3. This is an individual assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. You should not exchange code or write code for others. For individual assignments, each student must turn in a unique program. Your submission must not be substantially similar to another student's submission. Collaboration at a reasonable level will not result in substantially similar code.
4. For Help:
    - TA Helpdesk / Ed Discussion Board
    - Data Exchange Formats Handout
    - Requests, RegEx, and BeautifulSoup Handout
5. Comment out or delete all your function calls. When your code is run, all it should do is run without any errors.
6. Do not wait until the last minute to do this assignment in case you run into problems.
7. **Read the entire specifications document before starting this assignment.**
8. **IF YOUR CODE CANNOT RUN BECAUSE OF AN ERROR, IT IS A 0%**

Introduction

Please read through the entire document before starting this homework. The goal of this homework is to demonstrate your understanding of data exchange formats (CSV and JSON), APIs, and HTML. You will be defining a few functions that deal with file input and output and data manipulation. You should test them out first on your own computer, then when you have one or more of them working, upload the entire file to GradeScope to see how well your code performs on the test cases we have created. The score shown on Gradescope will not reflect your final grade on this assignment, as we reserve the right to run different test cases after the deadline. You can submit the homework files as many times as you'd like before the deadline.

Allowed imports: `csv, json, requests, bs4, pprint, re`

pprint Module

When using Python to manipulate data, programmers will want to be able to see what is stored in their outputs and variables. While the traditional print statement will most likely suffice, there will be times where you need to visualize more complex data structures in a way that the traditional print statement fails to do. One way around this is to utilize pprint. Short for Pretty Printer, pprint is a native Python library which allows you to customize the formatting of your outputs. This is especially useful when handling large data, as it will format your output so it is more readable. Since you will work with large quantities of data for this assignment, we highly encourage you to import and utilize this library, as test cases will be showcased using this format.

```
>>> from pprint import pprint

>>> ta_list = [{'Erin': ['kooky', 'courageous', 'caring'], 'major': 'ISYE'},
{'Braden': ['selfless', 'dependable', 'leader'], 'major': 'ISYE'}, {'Emily':
['silly', 'goofy', 'considerate'], 'major': 'CS'}]

>>> print(ta_list) # prints without formatting
[{'Erin': ['kooky', 'courageous', 'caring'], 'major': 'ISYE'}, {'Braden':
['selfless', 'dependable', 'leader'], 'major': 'ISYE'}, {'Emily': ['silly',
'goofy', 'considerate'], 'major': 'CS'}]

>>> pprint(ta_list) # prints with pretty formatting
[
 {'Erin': ['kooky', 'courageous', 'caring'],
  'major': 'ISYE'},
 {'Braden': ['selfless', 'dependable', 'leader'],
  'major': 'ISYE'},
 {'Emily': ['silly', 'goofy', 'considerate'],
  'major': 'CS'}
]
```

Introduction to the API

For this assignment, you will be utilizing the Disney API. This API allows users to access information about all Disney characters. This information includes the films and tv shows a character has been in, their allies, enemies, and more. To read more about how to use this api, see: API Documentation

Since APIs often return very long and complicated dictionaries, it is advisable to use Python's built–in module called `pprint` and to download and use the JSONView Google Chrome Extension. These tools allow users to better visualize the keys and values of the returned dictionaries.

Google Chrome Inspect Tool

When dealing with web scraping, a handy tool to utilize is Google Chrome's Inspect Tool. This allows you to better visualize and read the source code for each HTML page.

There are 3 ways to access the Inspect Tool:

1. Click on the three vertical dots in the upper right hand corner → More tools → Developer Tools
2. Right–click on the page and choose "Inspect"
3. Ctrl + Shift + C (**Windows**) or Cmd + Opt + C (**Mac**)

You should now be able to see the page's source code include other various details. You can select the square with the cursor at the top left corner of the menu to bring up "Inspect Element" mode, which will highlight the specific HTML line when hovering over your page.

Data Sources

For this assignment, you will utilize multiple different data files to complete your functions. The shrek.txt data is the Shrek script where each line in the file is a line from the script. The `netflix.csv` and `netflix.json` data were extracted and modified from this **Netflix** dataset on Kaggle, and `glee.html` was extracted from Glee Wikipedia. **The sources here are provided as additional info. Do not use the data from their original sources, only use the data files provided to you on Canvas alongside this prompt.**

t

Functions

A grading rubric including point distribution is provided at the end of this document.

Function name: `shrek_script`
Parameter(s): `file_name` (`str`), `output_file` (`str`)
Return Type: list
Description: Create a function that takes in `file_name` (a string representing the Shrek script) and writes to a new CSV file that is named `output_file`. The `file_name` and `output_file` parameters will **NOT** include file extensions.
The CSV file should contain the following information on each line from `file_name`:

Column	Value	Expected Data Type
`line_num`	Current line number of the script. (Ex: First line has `line_num` = 1, second line has `line_num` = 2, etc)	int
`num_words`	The number of words the current line contains	int
`total_words`	The number of words the script contains up until (and including) the current line	int
`shrek_present`	True if `'Shrek'` is present in the line. False otherwise.	bool
`line_text`	The current line **WITHOUT** trailing newline characters.	str

Python Test Case:

Return a list of lists containing the information you wrote into the CSV file. This is what your returned list should look like:

```
>>> shrek_script('shrek', 'shrek_clean')
[[1, 9, 9, False, 'Once upon a time there was a lovely princess.'],
[2, 20, 29, False, 'But she had an enchantment upon her of a
fearful sort which could only be broken by love's first kiss.'],
[3, 13, 42, True, 'She was locked away in a castle guarded by a
terrible fire-breathing dragon.'], [4, 15, 57, False, 'Many brave
knights had attempted to free her from this dreadful prison, but
none prevailed.']
...
```

```
[1592, 3, 8000, False, 'I can't breathe.'], [1593, 3, 8003, False,
'I can't breathe.']]
>>> len(shrek_script('shrek'))
1593
```

f'{output_file}.csv' looks like:

```
1,9,9,False,Once upon a time there was a lovely princess.
2,20,29,False,But she had an enchantment upon her of a fearful sort
which could only be broken by love's first kiss.
3,13,42,False,She was locked away in a castle guarded by a terrible
fire-breathing dragon.
4,15,57,False,"Many brave knights had attempted to free her from
this dreadful prison, but none prevailed."
5,23,80,False,She waited in the dragon's keep in the highest room
of the tallest tower for her true love and true love's first kiss.
...
1591,1,7997,False,Oh.
1592,3,8000,False,I can't breathe.
1593,3,8003,False,I can't breathe.
```

Function name: `csv_parser`

Parameter(s): `filename` (`str`)
Return Type: list
Description: Write a function that reads in and cleans the data from the `netflix.csv` file, whose name is passed into the function as `filename`, and returns a list of lists containing the data. Each sub–list is a list representing the information of each row (data for a single movie or TV show). The first sub-list, however, is an exception because it will hold the column headers. You should format each sub–list in the following order:

Column	Value	Expected Data Type	Additional Info
"ID"	Given id for the show	*str*	show_id is always given
"IsMovie"	True if it is a Movie; False otherwise	*bool*	type is always given
"IsShow"	True if it is a TV Show; False otherwise	*bool*	type is always given
"Title"	Title for the show	*str*	title is always given
"Director/s"	Name(s) of the director(s)	*str*	If blank set to "NIA" (No Info. Available)
"CastList"	List of Actors*	*list of str*	Format explained below. If blank set to []
"CastSize"	Number of Actors	*int*	Set to 0 if no actors are given
"Rating"	PG Rating	*str*	rating is always given
"DateAdded"	Date the show was added to the platform	*str*	Convert to MM/DD/YYYY format Set to "NIA" if missing
"IsHorror"	"Horror" if it is a horror movie/show; else "Not Horror".	*str*	Check the listed_in column
"Synopsis"	The first 10 words of the description followed by "..."	*str*	description is always given

Clarifications:

1. The list of actors should contain strings. Each string should be the initial of the actors's first name followed by a period, a space, and the last name. We will be ignore middle names and we will assume that the actors' last names are always the last word in their full name. If an actor only has a first name, we will write the first name stripped of all whitespace. See the example below:

If the cast is "Kamal Hassan, J.K Simmons, Meena, Yu Xia, Antony del Rio, Zendaya ", then we should have the following CastList:
["K. Hassan", "J. Simmons", "Meena", "Y. Xia", "A. Rio", "Zendaya"]

2. Leave no space between the last word in the synopsis and the ellipsis ("...")

3. The provided date is always given as "Month Day, Year". When you translate to the "MM/DD/YYYY" format make sure that one digit numbers are translated correctly. July 4, 1776 would be translated as 07/04/1776, NOT as 7/4/1776.

   This mapping might be helpful:

```
mapping ={"January":"01", "February":"02", "March":"03",
          "April":"04", "May":"05", "June":"06",
          "July":"07", "August":"08", "September":"09",
          "October":"10", "November":"11","December":"12"}
```

4. Note the first list within your returned value should be a list of the specific header names. You should preserve the order of rows as found in the csv file.

5. Also, keep in mind that blank entries in a csv are usually empty strings ("").

If you are experiencing difficulty in reading the csv file and receive an encoding error, try specifying the `utf8` encoding:

```
open("example.csv", encoding = "utf8")
```

Below is a snippet of what the returned value should look like:
Python Test Case:

```
>>> csv_parser('netflix.csv')
[['ID',
  'IsMovie',
  'IsShow',
  'Title',
  'Director/s',
  'CastList',
  'CastSize',
  'Rating',
  'DateAdded',
  'IsHorror',
  'Synopsis'],
 ['s1',
  True,
  False,
  'Dick Johnson Is Dead',
```

```
  'Kirsten Johnson',
  [],
  0,
  'PG-13',
  '09/25/2021',
  'Not Horror',
  'As her father nears the end of his life, filmmaker...'],
 ['s3',
  False,
  True,
  'Ganglands',
  'Julien Leclercq',
  ['S. Bouajila',
   'T. Gotoas',
   'S. Jouy',
   'N. Akkari',
   'S. Lesaffre',
   'S. Kechiouche',
   'N. Farihi',
   'G. Rampelberg',
   'B. Diombera'],
  9,
  'TV-MA',
  '09/24/2021',
  'Not Horror',
  'To protect his family from a powerful drug lord, skilled...'],
  ...
  ]
>>> len(csv_parser('netflix.csv'))
4405
```

CS2316 Spring 2021 HW 02: Data Exchange Formats and Manipulation

Function name: `json_parser`
Parameter(s): `filename` (str)
Return Type: list
Description:
Write a function that reads in and cleans the data from the JSON file `netflix.json`, whose name is passed in to the function as `filename`. You should return a new list of dictionaries, where each dictionary represents information for a single movie or TV show in the provided dataset.

You should format each dictionary as listed (Keys are case-sensitive):

Key	Value	Expected Data Type	Additional Info
"ID"	Given id for the show	*str*	show_id is always given
"IsMovie"	True if it is a Movie; False otherwise	*bool*	type is always given
"IsShow"	True if it is a TV Show; False otherwise	*bool*	type is always given
"Title"	Title for the show	*str*	title is always given
"Director/s"	Name(s) of the director(s)	*str*	If blank set to "NIA" (No Info. Available)
"CastList"	List of Actors	*list of str*	Format explained below. If blank set to []
"CastSize"	Number of Actors	*int*	Set to 0 if no actors are given
"Rating"	PG Rating	*str*	rating is always given
"DateAdded"	Date the show was added to the platform	*str*	Convert to MM/DD/YYYY format. If blank, set to "NIA"
"IsHorror"	True if it is a "Horror" movie/show; else "Not Horror".	*str*	Check the listed_in column
"Synopsis"	The first 10 words of the description followed by "..."	*str*	description is always given

Clarifications :
Everything from the previous question still applies.
The **headers should be the keys** of your dictionaries.

In contrast with csv files, blank entries in json files are typically null, which is converted to None when the file is converted to a python dict.

Python Test Case:

```
>>> json_parser('netflix.json')
[{'CastList': ['A. Qamata',
               'K. Ngema',
               'G. Mabalane',
               'T. Molaba',
               'D. Windvogel',
               'N. Thahane',
               'A. Greeff',
               'X. Tshabalala',
               'G. Sithole',
               'C. Mahlangu',
               'R. Morny',
               'G. Fincham',
               'S. Ka-Ncube',
               'O. Gwanya',
               'M. Mathys',
               'S. Schultz',
               'D. Williams',
               'S. Miller',
               'P. Mofokeng'],
  'CastSize': 19,
  'DateAdded': '09/24/2021',
  'Director/s': 'NIA',
  'ID': 's2',
  'IsHorror': 'Not Horror',
  'IsMovie': False,
  'IsShow': True,
  'Rating': 'TV-MA',
  'Synopsis': 'After crossing paths at a party, a Cape Town teen sets...',
  'Title': 'Blood & Water'},
 {'CastList': [],
  'CastSize': 0,
  'DateAdded': '09/24/2021',
  'Director/s': 'NIA',
  'ID': 's4',
  'IsHorror': 'Not Horror',
  'IsMovie': False,
  'IsShow': True,
  'Rating': 'TV-MA',
  'Synopsis': 'Feuds, flirtations and toilet talk go down among the '
              'incarcerated women...',
  'Title': 'Jailbirds New Orleans'},
...
]
>>> len(json_parser('netflix.json'))
4403
```

Function name: `horror`
Parameter(s): `csv_data` (list), `json_data` (list), `filename`(str)
Return Type: dict
Description: Write a function that passes in the `csv_data` from the `csv_parser()` function, the `json_data` from the `json_parser()` function, and the `filename` of the JSON file that the data will be written to.
Your manager thinks the number of new horror movies is highest in October, but the number of new horror shows is not. To begin testing their claim, your first task is to create a dictionary with the following structure and **write it as JSON** to `filename`.

<table>
<tr><th>First Layer Key</th><th colspan="3">First Layer Value</th></tr>
<tr><td rowspan="3">Type of media based on 'IsMovie' boolean value, e.g. 'Movie' or 'Show</td><td rowspan="2">Second Layer Key</td><td colspan="2">Second Layer Value</td></tr>
<tr><td>Third Layer Key</td><td>Third Layer Value</td></tr>
<tr><td>Release month, e.g. '09'</td><td>'num_horror'</td><td>Number of movies (shows) in the horror genre</td></tr>
<tr><td></td><td>'total'</td><td>Total number of movies (shows)</td></tr>
<tr><td></td><td>'overall'</td><td colspan="2">Same as above, but for the entire media type</td></tr>
</table>

Note: There are 10 shows for which a release date is not specified. These should be handled by an "NIA" key in the second layer as shown in the output below.

Python Test Case:

```
>>> c_data = csv_parser('netflix.csv')
>>> j_data = json_parser('netflix.json')
>>> output = horror(c_data, j_data, 'double_analysis.json')
>>> pprint(output)
{'Movie': {'01': {'num_horror': 48, 'total': 546},
           '02': {'num_horror': 20, 'total': 382},
           '03': {'num_horror': 21, 'total': 529},
           '04': {'num_horror': 33, 'total': 550},
           '05': {'num_horror': 19, 'total': 439},
           '06': {'num_horror': 26, 'total': 492},
           '07': {'num_horror': 34, 'total': 565},
           '08': {'num_horror': 36, 'total': 519},
           '09': {'num_horror': 28, 'total': 519},
           '10': {'num_horror': 42, 'total': 545},
           '11': {'num_horror': 24, 'total': 498},
           '12': {'num_horror': 26, 'total': 547},
           'overall': {'num_horror': 357, 'total': 6131},
 'Show': {'01': {'num_horror': 4, 'total': 192},
          '02': {'num_horror': 6, 'total': 181},
```

```
        '03': {'num_horror': 6, 'total': 213},
        '04': {'num_horror': 6, 'total': 214},
        '05': {'num_horror': 9, 'total': 193},
        '06': {'num_horror': 9, 'total': 236},
        '07': {'num_horror': 6, 'total': 262},
        '08': {'num_horror': 5, 'total': 236},
        '09': {'num_horror': 9, 'total': 251},
        '10': {'num_horror': 5, 'total': 215},
        '11': {'num_horror': 3, 'total': 207},
        '12': {'num_horror': 7, 'total': 266},
        'NIA': {'num_horror': 0, 'total': 10},
        'overall': {'num_horror': 75, 'total': 2676}}}
```

Function name: `character_info`
Parameter(s): `page` (int)
Return Type: list of tuples
Description: Write a function that makes a request to the Disney API and cleans the retrieved data. Below are some examples of Disney API requests:

Url	Output
https://api.disneyapi.dev/characters?page=**1**	Information about the first 50 Disney characters when sorted alphabetically, containing characters like Achilles, Agatha, and Addison.
https://api.disneyapi.dev/characters?page=**2**	Information about the next 50 Disney characters when sorted alphabetically, containing characters like Air Bud, Alice, and Alley Cats.
https://api.disneyapi.dev/characters?page=**149**	Information about the final 38 Disney characters when sorted alphabetically, containing characters like Zeus, Zuni, and Zuzu.

Write a function that takes in a page parameter and will make a Disney API request for the characters of that specific page. Get the `name`, `id`, `films`, and `tvShows` for each of the 50 characters on that page. If a character is not in any tvShows, replace the tvShows list with the string 'None'.

Only include characters that have been in at least 1 movie. Sort the characters in descending order based on the number of movies they have been in.

APIs can be updated frequently, so your returned values may differ from the below test case.

Python Test Cases:

```
>>> character_info(49)
[('Claude Frollo', 2498, ['The Hunchback of Notre Dame', "Mickey's House of
Villains", 'Once Upon a Halloween'], ['House of Mouse']), ('Galactic Armada',
2536, ['Lilo & Stitch', 'Stitch! The Movie', 'Leroy & Stitch'], ['Lilo &
Stitch: The Series']), ("Frollo's soldiers", 2497, ['The Hunchback of Notre
Dame', 'The Hunchback of Notre Dame II'], 'None'), ('Julie Gaffney', 2534,
['D2: The Mighty Ducks', 'D3: The Mighty Ducks'], 'None'), ('Fritz Stahlbaum',
2493, ['The Nutcracker and the Four Realms'], 'None'), ('Frog Servants', 2496,
['Alice in Wonderland (2010 film)'], 'None'), ('Frou-Frou', 2502, ['The
Aristocats'], 'None'), ('Fru Fru', 2503, ['Zootopia'], ['Zootopia+']),
('Fudgy', 2506, ['Leroy & Stitch'], ['Lilo & Stitch: The Series', 'Stitch!']),
('Fuli', 2508, ['The Lion Guard: Return of the Roar'], ['The Lion Guard']),
('Fun Bun and Puddles', 2510, ['Ralph Breaks the Internet'], 'None'), ('Doug
Funnie', 2512, ["Doug's 1st Movie"], ["Disney's Doug"]), ('Judy Funnie', 2513,
["Doug's 1st Movie"], ["Disney's Doug"]), ('Phil Funnie', 2514, ["Doug's 1st
Movie"], ["Disney's Doug"]), ('Furies', 2515, ['Hercules (film)'], ['Hercules
(TV series)', 'Once Upon a Time']), ('Nancy Furman', 2516, ['Pollyanna'],
'None'), ('Fury', 2517, ['Tinker Bell and the Legend of the NeverBeast'],
'None'), ('Dr. Fusion', 2519, ['Teen Beach Movie'], 'None'), ('G-Force', 2523,
['G-Force'], 'None'), ('Judge G. Whelan', 2524, ['The Santa Clause'], 'None'),
('G2', 2525, ['Inspector Gadget 2'], 'None'), ('Abigail and Amelia Gabble',
2526, ['The Aristocats'], ['House of Mouse'])]
>>> len(character_info(49))
22

>>> character_info(72)
[('Shere Khan', 3650, ['The Jungle Book', 'The Jungle Book 2', "Rudyard
Kipling's The Jungle Book", "The Jungle Book: Mowgli's Story", 'Once Upon a
Halloween', 'The Jungle Book (2016 film)'], ['TaleSpin', 'Bonkers', 'Jungle
Cubs', 'House of Mouse', 'Walt Disney anthology series']), ('The King', 3673,
['Cinderella (1950 film)', 'Cinderella II: Dreams Come True', 'Cinderella III:
A Twist in Time', 'The Little Mermaid', 'Cinderella (2015 film)'], ['House of
Mouse', 'Once Upon a Time']), ('Kiara', 3654, ['The Lion King', "The Lion King
II: Simba's Pride", 'The Lion Guard: Return of the Roar', 'The Lion King (2019
film)'], ['The Lion Guard']), ('Khan', 3648, ['Mulan', 'Mulan II', 'Mulan
(2020 film)'], 'None'), ('The King of Hearts', 3686, ['Alice in Wonderland
(1951 film)', "Mickey's Magical Christmas: Snowed in at the House of Mouse",
"Mickey's House of Villains"], ['House of Mouse', 'Once Upon a Time']),
('Kerchak', 3638, ['Tarzan', 'Tarzan II'], ['The Legend of Tarzan']),
('Kessie', 3642, ['Winnie the Pooh: Seasons of Giving', 'The Book of Pooh:
Stories from the Heart'], ['The New Adventures of Winnie the Pooh', 'The Book
of Pooh']), ("Kerchak and Kala's baby", 3639, ['Tarzan (film)'], 'None'),
('Kernel', 3640, ['Leroy & Stitch'], ['Lilo & Stitch: The Series']), ('Prince
Khalid', 3647, ['Sofia the First: Once Upon a Princess'], ['Sofia the First',
```

```
'Fancy Nancy']), ('Ki', 3653, ['Mars Needs Moms'], 'None'), ('Kiche', 3656,
['White Fang'], 'None'), ('Killjoy Margret', 3668, ['Fantasia 2000'], 'None'),
('The King', 3672, ['Beauty and the Beast (2017 film)'], 'None'), ('The King
and the Duke', 3676, ['The Adventures of Huck Finn'], 'None'), ("King Henry's
Army", 3679, ['Maleficent (film)'], 'None'), ('The King of Costa Luna', 3684,
['Princess Protection Program'], 'None')]
>>> len(character_info(72))
17
```

Function name: `glee_dict`
Parameters(s): `filename` (`str`)
Return type: dict
Description: As a Glee fanatic, you're curious about how popular each season was. You decide that the popularity of a season can be determined by viewership retention between the season premiere and finale. Write a function that reads data from `glee.html` and returns a dictionary that maps each season to the following information: the total viewers for the premier, the total viewers for the finale, and the percent change in viewers (calculate using finale/premiere, round to 2 decimal places), and whether the season was successful. For your strings, make sure to format the number of viewers as '{num} million' and the percent change as '{percent}% increase' or '{percent}% decrease'.

Although you must scrape data from the HTML file provided, you may view the live representative website, where the HTML page source was extracted: Glee Wikipedia. We **highly recommend** you utilize Google Chrome's Inspect Tool to inspect the page source so you can identify the parts of the source code efficiently.

Python Test Case:

```
>>> seasons = glee_dict('glee.html')
>>> pprint(seasons)
{'Season 1': {'Finale Views': '10.92 million',
              'Premiere Views': '9.62 million',
              'Retention': '13.51% increase',
              'Success': True},
 'Season 2': {'Finale Views': '11.80 million',
              'Premiere Views': '12.45 million',
              'Retention': '5.22% decrease',
              'Success': False},
  ...,
 'Season 6': {'Finale Views': '2.54 million',
              'Premiere Views': '2.34 million',
              'Retention': '8.55% increase',
              'Success': True}}
```

Function name: `colorful_film`
Parameter(s): `None`
Return Type: list
Description: You are interested in academy award-winning films and would like to see which films won the *Academy Award for Best Picture* among all films. Write a function that reads the data from Academy Award Wikipedia, and returns a list of lists that each contain the movie name, year, awards, and nominations. Format year, awards, and nominations to int, and you can ignore numbers after special characters. (Example: *2020/21 → 2020, 0(1) → 0*)

Sort the output list in ascending order of year released.

Hint: Splitting a string on non-integer may be useful

Python Test Cases:

```
>>> films = colorful_film()
>>> pprint(films)
[['Wings', 1927, 2, 2],
 ['The Broadway Melody', 1928, 1, 3],
 ['All Quiet on the Western Front', 1929, 2, 4],
 ['Cimarron', 1930, 3, 7],
 ['Grand Hotel', 1931, 1, 1],
 ['Cavalcade', 1932, 3, 4],
 ['It Happened One Night', 1934, 5, 5],
 ['Mutiny on the Bounty', 1935, 1, 8],
 ...,
 ['The Shape of Water', 2017, 4, 13],
 ['Green Book', 2018, 3, 5],
 ['Parasite', 2019, 4, 6],
 ['Nomadland', 2020, 3, 6]]
```

Gradescope Requirements

- **NO PRINT STATEMENTS.** this will break the autograder
- **NO FUNCTION CALLS** outside of function definitions. This will also break the autograder
- Make sure the file submitted is titled **HW02.py**
- **Only import csv, json, requests, bs4, re, and pprint modules**

- **All points of this grade will be lost** if it is shown that you have collaborated in an unauthorized manner on this assignment or otherwise violated the Georgia Tech honor code.

Grading Rubric

shrek_script	10 pts
csv_parser	15 pts
json_parser	15 pts
horror	20 pts
character_info	10 pts
glee_dict	15 pts
colorful_film	15 pts
Total Possible	**100**/100 pts