# Problem 1: Movie Night!¶

*Version 1.7*
This problem will test your mastery of basic Python data structures. It consists of five (5) exercises, numbered 0 through 4, worth a total of ten (10) points.

For this problem, you will be dealing with Rotten Tomatoes' list of Top 40 All Time Sci-Fi/Fantasy Movies. These rankings are based on critic reviews, and the list is sorted by "Adjusted Rank" - a (secret) weighted formula that RT uses to account for variation in number of reviews. To test our Python skills in this problem, we will explore and manipulate the data, and ultimately create our very own ranking system.

## Loading the data

First, run the following code cell to set up the problem. The provided output shows the data you will be working with. The data is a read in from the csv as a list of dictionaries, one for each row, with key-value pairs representing the attributes: Rating, Reviews, Title, and Year.

In [1]:

```
# Some modules you'll need in this part
from pprint import pprint
import random
from collections import defaultdict
from problem_utils import load_movies_data

movies = load_movies_data('movies.csv')

print("First five movies in this dataset:")
pprint(movies[:5])
```

```
First five movies in this dataset:
[{'Rating': 96, 'Reviews': 483, 'Title': 'Black Panther', 'Year': 2018},
 {'Rating': 98, 'Reviews': 114, 'Title': 'The Wizard of Oz', 'Year': 1939},
 {'Rating': 97, 'Reviews': 408, 'Title': 'Mad Max: Fury Road', 'Year': 2015},
 {'Rating': 93, 'Reviews': 434, 'Title': 'Wonder Woman', 'Year': 2017},
 {'Rating': 98,
  'Reviews': 128,
  'Title': 'E.T. The Extra-Terrestrial',
  'Year': 1982}]
```

**Exercise 0** (2 points). Define a function **`find_movies(t_rating, t_reviews)`** that, given a rating threshold and # of reviews threshold, returns a list of movie names that meet or exceed both thresholds. It should use the movies dataset stored in the `movies` variable, defined previously.
*Note*: The test cell refers to hidden tests, but in fact, the test is not hidden per se. Instead, we are hashing the strings returned by your solution to be able to check your answer without revealing it to you directly.

In [2]:

```
def find_movies(t_rating, t_reviews):
    ### BEGIN SOLUTION
```

```
    res = []
    for item in movies:
        if item["Rating"] >= t_rating and item["Reviews"] >= t_reviews:
            res.append(item['Title'])
    return res
    ### END SOLUTION
```

In [3]:

```
# Test cell: `check_find_movies`

### BEGIN HIDDEN TESTS
# Reference solution:
def fm(t_rating, t_reviews):
    res = []
    for item in movies:
        if item["Rating"] >= t_rating and item["Reviews"] >= t_reviews:
            res.append(item['Title'])
    return res

# Generate and store some reference solutions
def gen_find_movies_solns(filename='find_movies_solns.csv'):
    from os.path import isfile
    from problem_utils import make_hash
    if not isfile(filename):
        all_ratings = [i["Rating"] for i in movies]
        min_rating, max_rating = min(all_ratings), max(all_ratings)
        all_reviews = [i["Reviews"] for i in movies]
        min_reviews, max_reviews = min(all_reviews), max(all_reviews)
        with open(filename, 'wt') as fp_soln:
            for k in range(1000):
                tra = random.randrange(min_rating, max_rating)
                tre = random.randrange(min_reviews, max_reviews)
                soln = fm(tra, tre)
                hashed_soln = make_hash('|'.join(sorted(soln)))
                fp_soln.write(f'{tra},{tre},{hashed_soln}\n')

gen_find_movies_solns()
### END HIDDEN TESTS

def check_find_movies_solns(filename='find_movies_solns.csv'):
    from problem_utils import check_hash, open_file
    with open_file(filename, 'rt') as fp_soln:
        for line in fp_soln.readlines():
            tra, tre, ref_soln_hashed = line.strip().split(',')
            your_soln = find_movies(int(tra), int(tre))
```

```
            your_soln_str = '|'.join(sorted(your_soln))

            assert type(your_soln) is list, 'The result should be a list.'
            assert check_hash(your_soln_str, ref_soln_hashed), \
                   f'Your solution for t_rating={tra} and t_reviews={tre} does
not match our reference solution.'

check_find_movies_solns()
print("\n(Passed!)")
```

```
(Passed!)
```

**Exercise 1** (2 points). Define a function **`create_mv_dict(rating_threshold, review_threshold)`** that re-structures the original data, taking into account two thresholds (`rating_threshold` and `review_threshold`). In particular, it should return a new nested data structure defined as follows.

The outer data structure should be a dictionary with three keys: `"years"`, `"ratings"`, and `"reviews"`. Suppose we call this dictionary, `result`. Its values should be as follows.

(1) **`result["years"]`** should be a **dictionary**. Each key is a year and each value is a **list** of the movie titles appearing in that year. For instance, if you inspect the `movies` dataset, you'll see there were four movies in 2018. Therefore, `result["years"][2018] == ['Black Panther', 'Ant-Man and the Wasp', 'Sorry to Bother You', 'Avengers: Infinity War']` (or whatever the four movies were). The order in which the movies appears does not matter.
(2) **`result["ratings"]`** should also be a **dictionary**. It should have two keys: `"high_rating"` and `"low_rating"`. For the first, the value of `result["ratings"]["high_rating"]` should be a list of all movies whose rating is at least (≥≥) the ratings threshold, `rating_threshold`. Similarly, `result["ratings"]["low_rating"]` should hold the list of movies strictly less than (<<) the threshold.
(3) **`result["reviews"]`** should also be a **dictionary**, with two keys: `"high_review"` and `"low_review"`. For the first, the value of `result["reviews"]["high_review"]` should be a **list** of all movies whose number of reviews is at least (≥≥) the reviews threshold, `review_threshold`. Similarly, `result["reviews"]["low_review"]` should be a list of all movies whose number of reviews is strictly less than (<<) the threshold.

In [4]:

```
# Exercise 1 Solution

def create_mv_dict(rating_threshold, review_threshold):
    ### BEGIN SOLUTION
    new_dict = {"years": defaultdict(list), "reviews": defaultdict(list),
"ratings":defaultdict(list)}
    for mv in movies:
        new_dict["years"][mv["Year"]].append(mv["Title"])
        if mv["Reviews"] >= review_threshold:
            new_dict["reviews"]["high_review"].append(mv["Title"])
        else:
```

```
            new_dict["reviews"]["low_review"].append(mv["Title"])
        if mv["Rating"]>=rating_threshold:
            new_dict["ratings"]["high_rating"].append(mv["Title"])
        else:
            new_dict["ratings"]["low_rating"].append(mv["Title"])
    return new_dict
    ### END SOLUTION
```

In [5]:

```
# Test cell: `check_create_mv_dict`

testd = create_mv_dict(97,50)

assert (type(testd) is dict or type(testd) is defaultdict), 'You did not
create a dictionary.'
assert set(('years','ratings','reviews')) <= set((list(testd.keys()))), 'Your
dictionary does not have the correct keys.'
assert len(list(testd['years'].keys())) == 25, 'Your years dictionary does not
contain all of the years in the data set.'
assert len(list(testd['ratings'].keys())) == 2, 'You did not create store the
ratings data correctly.'


t_rtgs = [80,85,90]
t_rvw = [50,100,150]
dl = []
for rt,rv in list(zip(t_rtgs,t_rvw)):
    dl.append(create_mv_dict(rt,rv))



assert len(dl[0]['years'][1964]) == 1, 'Number of movies in 1964 is
incorrect.'
assert len(dl[0]['years'][2015]) == 3, 'Number of movies in 2015 is
incorrect.'
assert len(dl[0]['years'][2017]) == 8, 'Number of movies in 2017 is
incorrect.'
assert dl[0]['years'][1940] == ['Pinocchio'], 'Movie list for 1940 is
incorrect.'
assert dl[0]['years'][2016] == ['Arrival','Captain America: Civil War','Kubo
and the Two Strings'], 'Movie list for 2016 is incorrect.'
assert len(dl[1]['ratings']['high_rating']) == 40, 'Number of movies in "high-
rating" is incorrect.'
assert len(dl[2]['reviews']['low_review']) == 16, 'Number of movies in "low-
review" is incorrect.'
```

```
print("\n(Passed!)")
```

```
(Passed!)
```

**Exercise 2** (2 points). Define a function **`avg_score_query(start_year, stop_year)`** to calculate the average rating and number of reviews between (inclusively) a time period. The two parameters/inputs are `start_year` and `stop_year`. The final result will be a text string to tell user the time period, the list of movies within that time period, and its average rating and average number of reviews.

**For example:**

`avg_score_query(1977, 2013)` will return this string:

"From 1977 to 2013, these movies (Alien, Aliens, E.T. The Extra-Terrestrial, Gravity, Harry Potter and the Deathly Hallows - Part 2, How to Train Your Dragon, Marvel's The Avengers, Star Trek, Star Wars: Episode IV - A New Hope, The Dark Knight, The Terminator) have a rating of 96.2 and 218 reviews on average."

**In order to pass the test cell, please**

1. Remember to round the rating calculation to the neareast tenth and store it as float number before printing, and the number of review calculation to the nearest integer;
2. Pay close attention to the format of the result above, look out for the punctuation, spaces;
3. Sort the movie titles in ascending order;
4. And use `statistics.mean`, which is imported for you, because other mean functions (e.g., `numpy.mean`) might give you a different result.

To help you debug, we've provided an extra cell and a demo of calling a function we've provided, called `where_strings_differ()`, which you can use to compare two strings and detect the first place where a difference exists.
*Note*: The test cell refers to hidden tests, but in fact, the test is not hidden per se. Instead, we are hashing the strings returned by your solution to be able to check your answer without revealing it to you directly.

In [6]:

```
from statistics import mean

def avg_score_query(start_year=1977, stop_year=2013):
    ### BEGIN SOLUTION
    mv_names = []
    mv_ratings = []
    mv_reviews = []
    for mv in movies:
        if mv["Year"] >= start_year and mv["Year"] <= stop_year:
            mv_names.append(mv["Title"])
            mv_ratings.append(mv["Rating"])
            mv_reviews.append(mv["Reviews"])
```

```
    return "From " + str(start_year) + " to " +str(stop_year) + ", these
movies ("+ ", ".join(sorted(mv_names)) + ") have a rating of " +
str(float(round(mean(mv_ratings),1))) + " and " + str(round(mean(mv_reviews)))
+" reviews on average."
    ### END SOLUTION
```

In [7]:

```
# Use this cell for any intermediate debugging
from problem_utils import where_strings_differ

your_solution = avg_score_query()
true_solution = r"From 1977 to 2013, these movies (Alien, Aliens, E.T. The
Extra-Terrestrial, Gravity, Harry Potter and the Deathly Hallows - Part 2, How
to Train Your Dragon, Marvel's The Avengers, Star Trek, Star Wars: Episode IV
- A New Hope, The Dark Knight, The Terminator) have a rating of 96.2 and 218
reviews on average."

where_strings_differ(your_solution, true_solution)
```

```
==> Strings appear to be identical.
```

In [8]:

```
# Test cell: `check_avg_score`

### BEGIN HIDDEN TESTS
# Reference solution:
def asq(start_year=1977, stop_year=2013):
    from statistics import mean
    mv_names = []
    mv_ratings = []
    mv_reviews = []
    for mv in movies:
        if mv["Year"] >= start_year and mv["Year"] <= stop_year:
            mv_names.append(mv["Title"])
            mv_ratings.append(mv["Rating"])
            mv_reviews.append(mv["Reviews"])
    return "From " + str(start_year) + " to " +str(stop_year) + ", these
movies ("+ ", ".join(sorted(mv_names)) + ") have a rating of " +
str(float(round(mean(mv_ratings),1))) + " and " + str(round(mean(mv_reviews)))
+" reviews on average."

# Generate and store some reference solutions
def gen_avg_score_query_solns(filename='avg_score_query_solns.csv'):
    from os.path import isfile
    from problem_utils import make_hash
    year = sorted([i["Year"] for i in movies])
```

```
    if not isfile(filename):
        with open(filename, 'wt') as fp_soln:
            for k in range(1, 1000):
                sy = random.randint(0, len(year)-1)
                stop = random.randint(sy+1, len(year))
                sy, stop = year[sy], year[stop-1]
                ref_soln = asq(sy, stop)
                ref_soln_hashed = make_hash(ref_soln)
                fp_soln.write(f"{sy},{stop},{ref_soln_hashed}\n")

gen_avg_score_query_solns()
### END HIDDEN TESTS

def check_avg_score_query_solns(filename='avg_score_query_solns.csv'):
    from problem_utils import check_hash, open_file
    with open_file(filename, 'rt') as fp_soln:
        for line in fp_soln.readlines():
            sy_raw, stop_raw, ref_soln_hashed = line.strip().split(',')
            your_soln = avg_score_query(int(sy_raw), int(stop_raw))
            assert type(your_soln) is str, 'the result should be a string of
text'
            assert check_hash(your_soln, ref_soln_hashed), f"Your solution for
`avg_score_query({sy_raw}, {stop_raw})` appears to be incorrect."

check_avg_score_query_solns()
print("\n(Passed!)")
```

```
(Passed!)
```

**Exercise 3** (1 point). Define a function **`movie_details()`** that returns a list of movie dictionaries, just like the original input with all of its original attributes, **plus two new attributes**. These new attributes will be derived from the data: `Tomato_Rank` and `Age`. Here is a precise description; see also the example below.

- `"Tomato_Rank"`: Assume that the original `movies` list is already in order by Tomato rank. That is, assume that the first movie has a rank of 1. Store this rank as the value for a `"Tomato_Rank"` key.
- `"Age"`: Store the current approximate age in years, as an integer value associated with the key, `"Age"`. Since this problem is part of the year 2019, a movie with the year of 2017 would have an age of 2019-2017=2 years.

**Example.** The third movie, `movies[2]`, has the following structure:

```
    {'Rating': 97, 'Title': 'Mad Max: Fury Road', 'Year': 2015,
    'Reviews': 408}
```

Therefore, in your returned output, this movie will look like

```
    {'Rating': 97, 'Title': 'Mad Max: Fury Road', 'Year': 2015,
    'Reviews': 408, 'Tomato_Rank': 3, 'Age': 4}
```

In [9]:

```
def movie_details():
    ### BEGIN SOLUTION
    mv_details = []
    for rank, mv in enumerate(movies):
        mv_details.append(mv)
        mv_details[-1]["Tomato_Rank"] = rank+1
        mv_details[-1]["Age"] = 2019-mv_details[-1]["Year"]
    return mv_details
    ### END SOLUTION

print(movie_details()[2]) # Should match the example
```

```
{'Rating': 97, 'Title': 'Mad Max: Fury Road', 'Year': 2015, 'Reviews': 408,
'Tomato_Rank': 3, 'Age': 4}
```

In [10]:

```
# Test cell: `check_movie_details`

md = movie_details()
assert type(md) is list , 'Your function does not create a list'
assert len(md) == 40 , 'Your list does not contain all of the movies'
assert [len(m.items())==6 for m in md] , 'Your dictionaries do not contain all
of the necessary movie attributes'

assert list(filter(lambda m: m['Title'] == 'E.T. The Extra-Terrestrial', md))
[0]['Tomato_Rank'] == 5 , 'Your Tomato_Rank is incorrect for E.T. The Extra-
Terrestrial'
assert list(filter(lambda m: m['Title'] == 'Nosferatu, a Symphony of Horror',
md))[0]['Tomato_Rank'] == 16 , 'Your Tomato_Rank is incorrect for Nosferatu, a
Symphony of Horror'
assert list(filter(lambda m: m['Title'] == 'Blade Runner 2049', md))[0]
['Tomato_Rank'] == 31 , 'Your Tomato_Rank is incorrect for Blade Runner 2049'
assert list(filter(lambda m: m['Title'] == 'Dr. Strangelove', md))[0]['Age']
== 55 , 'Your Tomato_Rank is incorrect for Dr. Strangelove'
assert list(filter(lambda m: m['Title'] == 'Star Wars: Episode VII - The Force
Awakens', md))[0]['Age'] == 4 , 'Your Tomato_Rank is incorrect for Star Wars:
Episode VII - The Force Awakens'
assert list(filter(lambda m: m['Title'] == 'How to Train Your Dragon', md))[0]
['Age'] == 9 , 'Your Tomato_Rank is incorrect for How to Train Your Dragon'

print("\n(Passed!)")
```

```
(Passed!)
```

**Exercise 4** (3 points). For this final exercise, let's create a custom movie ranking system and compare it to the Rotten Tomatoes rank.

First, we need a formula to score each movie in the dataset. Suppose we believe that the `Rating`, `Reviews`, and `Age` should be factors. For a given movie with attributes `rating`, `reviews`, and `age`, let's assume a linear score of the form,

```
custom_score = weights[0]*rating + weights[1]*reviews +
weights[2]*age
```

where the `weights` are provided by another end-user analyst.

**Your task**: Define a function **`custom_rankings(weights)`** that, given a set of weights as input, returns a dictionary of the top 5 movies and this custom score. More specifically, it should return a dictionary where each key is a movie name and the corresponding value is a pair (2-tuple), `(custom_score, abs(custom_rank - tomato_rank))`. Here, `custom_score` is computed from the formula above, and `custom_rank` is the movie's rank according to the custom scores.
**Note 0.** The index of the input weights should correspond to the ordering of metrics provided above, i.e, `weights[0]` corresponds to `Rating`, `weights[1]` to `Reviews`, and `weights[2]` to `Age`.

**Note 1.** The ranking convention you use for the custom ranks should match the ranking system in Exercise 3 (1 = best).

Here is a sample output:

```
custom_rankings([0.65,0.1,0.25]) ->

          {

          'Black Panther': (110.95000000000002, 0),

          'Mad Max: Fury Road': (104.85000000000001, 1),

          'Star Wars: The Last Jedi': (104.75, 4),

          'Wonder Woman': (104.35000000000001, 0),

          'Star Wars: Episode VII - The Force Awakens': (103.25,
10)

          }
```

In [11]:

```
def custom_rankings(weights):
    assert type(weights) is list
    assert len(weights) == 3
    assert weights[0]+weights[1]+weights[2] == 1

    ### BEGIN SOLUTION
    movies = movie_details()
```

```
    rankings = {mv['Title']: ((weights[0]*mv['Rating']) +
(weights[1]*mv['Reviews']) + (weights[2]*mv['Age'])
                              ,
                             (mv['Tomato_Rank']))
                for mv in movies}
    return {mov: (rankings[mov][0],abs((c_rnk+1)-rankings[mov][1]))
            for c_rnk,mov in enumerate(sorted(rankings, key=lambda i:
rankings[i][0],reverse=True)[:5])}
    ### END SOLUTION
```

In [12]:

```
# Test cell: `check_custom_rankings`

t_rnk = custom_rankings([.1,.1,.8])

assert type(t_rnk) is dict, 'Your function does not return a dictionary'
assert type(t_rnk[list(t_rnk.keys())[0]]) is tuple, 'Your dictionary does not
contain tuples as values'
assert len(t_rnk) == 5, 'Your dictionary does not contain the correct number
of movies'

wl = [
    [0.3,0.3,0.4],
    [0.6,0.05,0.35],
    [0.5,0.1,0.4],
    [0.7,0.01,0.29],
    [0.4,0.3,0.3]
]


my_rnks = [custom_rankings(w) for w in wl]


assert set(('Black Panther','Star Wars: The Last Jedi','Avengers: Infinity
War')) <= set(my_rnks[0]), 'Your top 5 movies are not correct.'
assert set(('Metropolis','Frankenstein','The Wizard of Oz')) <=
set(my_rnks[1]), 'Your top 5 movies are not correct.'
assert set(('Nosferatu, a Symphony of Horror','Star Wars: The Last Jedi','The
Wizard of Oz')) <= set(my_rnks[2]), 'Your top 5 movies are not correct.'
assert set(('Frankenstein','The Bride of Frankenstein','Pinocchio')) <=
set(my_rnks[3]), 'Your top 5 movies are not correct.'
assert set(('Black Panther','Wonder Woman','Avengers: Infinity War')) <=
set(my_rnks[4]), 'Your top 5 movies are not correct.'
```

```
assert my_rnks[0]['Avengers: Infinity War'][1] == 34, 'Your rank differences
are not correct.'
assert my_rnks[1]['Frankenstein'][1] == 19, 'Your rank differences are not
correct.'
assert my_rnks[2]['Star Wars: The Last Jedi'][1] == 2, 'Your rank differences
are not correct.'
assert my_rnks[3]['Pinocchio'][1] == 19, 'Your rank differences are not
correct.'
assert my_rnks[4]['Black Panther'][1] == 0, 'Your rank differences are not
correct.'
```

```
assert abs(my_rnks[4]['Avengers: Infinity War'][0]-167.5) < 1e-16 , 'Your
custom scores are not correct.'
assert abs(my_rnks[3]['The Bride of Frankenstein'][0]-94.8) < 1e-16, 'Your
custom scores are not correct.'
assert abs(my_rnks[2]['Metropolis'][0]-98.5) < 1e-16, 'Your custom scores are
not correct.'
assert abs(my_rnks[1]['The Wizard of Oz'][0]-92.5) < 1e-16, 'Your custom
scores are not correct.'
assert abs(my_rnks[0]['Wonder Woman'][0]-158.9) < 1e-16, 'Your custom scores
are not correct.'
```

```
print("\n(Passed!)")
```

```
(Passed!)
```

**Fin!** Remember to test your solutions by running them as the autograder will: restart the kernel and run all cells from "top-to-bottom." Also remember to submit to the autograder; otherwise, you will not get credit for your hard work!