Stage 1: Data Plumbing

A lot of data science work often involves *plumbing*, the process of getting messy data into a more useful format. Data plumbing is the focus of stage 1. We'll develop and test three functions that will be helpful in stage 2:

1. `get_mapping(path)`: this loads a file that can be used to lookup names from IDs
2. `get_raw_movies(path)`: this loads movie data with info represented using IDs
3. `get_movies(movies_path, mapping_path)`: this uses the other two functions to load movie data, then replace IDs with names

---

Start by writing a function that starts like this:

```
def get_mapping(path):
```

When called, the `path` should refer to one of the mapping files (e.g., "small_mapping.csv"). The function should return a dictionary that maps IDs (as keys) to names (as values), based on the file referenced by `path`. For example, this code:

```
mapping = get_mapping("small_mapping.csv")
print(mapping)
```

Should print this:

```
{
  "nm0000131": "John Cusack",
  "nm0000154": "Mel Gibson",
  "nm0000163": "Dustin Hoffman",
  "nm0000418": "Danny Glover",
  "nm0000432": "Gene Hackman",
  "nm0000997": "Gary Busey",
  "nm0001149": "Richard Donner",
  "nm0001219": "Gary Fleder",
  "nm0752751": "Mitchell Ryan",
  "tt0093409": "Lethal Weapon",
  "tt0313542": "Runaway Jury"
}
```

Note that the mapping files DO NOT have a CSV header.

The following questions pertain to `small_mapping.csv` unless otherwise specified.

Question 1: what is returned by your `get_mapping("small_mapping.csv")` function?

In addition to displaying the result in the `Out [N]` area, keep the result in a variable for use in subsequent questions.

Question 2: what is the value associated with the key "tt0313542"?

Use the dictionary returned earlier. Do not call `get_mapping` a second time (that would be inneficient).

Question 3: what are the values in the mapping associated with keys beginning with "tt"?

Answer with a Python list.

Question 4: which keys in the mapping map to people with a first name of "Gary"?

Answer with a Python list. To get full points, you should write code that will count somebody named "gary" but will not count somebody named "Garyyy".

Build a function named `get_raw_movies` that takes a path to a movies CSV (e.g., "small_movies.csv" or "movies.csv") as a parameter and returns a list of dictionaries where each dictionary represents a movie as follows:

```
{
    "title": "movie-id",
    "year": <the year as an integer>,
    "rating": <the rating as a float>,
    "directors": ["director-id1", "director-id2", ...],
    "actors": ["actor-id1", "actor-id2", ....],
    "genres": ["genre1", "genre2", ...]
}
```

Note that the movie files DO have a CSV header.

Also note that the values for `directors`, `actors`, and `genres` always contain lists, even if those lists contain a single value.

Question 5: what does `get_raw_movies("small_movies.csv")` return?

The result should be this:

```
[{'title': 'tt0313542',
  'year': 2003,
  'rating': 7.1,
  'directors': ['nm0001219'],
  'actors': ['nm0000131', 'nm0000432', 'nm0000163'],
  'genres': ['Crime', 'Drama', 'Thriller']},
 {'title': 'tt0093409',
  'year': 1987,
  'rating': 7.6,
  'directors': ['nm0001149'],
  'actors': ['nm0000154', 'nm0000418', 'nm0000997', 'nm0752751'],
  'genres': ['Action', 'Crime', 'Thriller']}]
```

Also, keep the value returned by `get_raw_movies` in a variable. You should not call the function more often than necessary.

Question 6: how many genres did the movie at index 0 have?

Use the data from Q5.

Question 7: what is the ID of the last actor listed for the move at index 1?

Use the data from Q5.

Note that for the keys `actors`, `directors`, and `title`, we currently only use IDs instead of actual names. Write a function named `get_movies(movies_path, mapping_path)` that loads data from the `movies_path` file using `get_movies_raw` and converts the IDs to names using a mapping based on the `mapping_path` file, which you should load using your `get_mapping` function.

Each dictionary in the list should look something like this:

```
{
    "title": "the movie name",
    "year": <the year as an integer>,
    "rating": <the rating as a float>,
    "directors": ["director-name1", "director-name2", ...],
    "actors": ["actor-name1", "actor-name2", ....],
    "genres": ["genre1", "genre2", ...]
}
```

Notice the difference between the previous one and this (names instead of IDs). This list of dictionaries is vital for almost all of the following questions.

We recommend you get the translation from ID to name working for title before you start trying to translate actors and directors.

After you implement your function (or implement enough of it to answer some of the below questions), call it and store the result in `movies` as follows:

```
small = get_movies("small_movies.csv", "small_mapping.csv")
```

Question 8: what is `small[0]["title"]`?

Just paste `small[0]["title"]` into a cell and run it. We're doing this to check that the structures in `small` (as returned by `get_movies` above) contain the correct data.

Question 9: what is `small[1]["directors"]`?

Question 10: what is `small[-1]["actors"]`?

Question 11: what is `small`?

The result should look like this:

```
[{'title': 'Runaway Jury',
  'year': 2003,
  'rating': 7.1,
  'directors': ['Gary Fleder'],
  'actors': ['John Cusack', 'Gene Hackman', 'Dustin Hoffman'],
  'genres': ['Crime', 'Drama', 'Thriller']},
 {'title': 'Lethal Weapon',
  'year': 1987,
  'rating': 7.6,
  'directors': ['Richard Donner'],
  'actors': ['Mel Gibson', 'Danny Glover', 'Gary Busey', 'Mitchell Ryan'],
  'genres': ['Action', 'Crime', 'Thriller']}]
```

If you've gotten this far, your functions must be working pretty well with the small data. So let's try the full dataset!

```
movies = get_movies("movies.csv", "mapping.csv")
```

You are not allowed to call `get_movies` more than once for the "movies.csv" file in your notebook. Reuse the `movies` variable instead (that's more efficient).

Question 12: what are the first 3 rows in movies?

The result should look like this:

```
[{'title': 'The Big Wedding',
  'year': 2013,
  'rating': 5.6,
  'directors': ['Justin Zackham'],
  'actors': ['Robert De Niro'],
  'genres': ['Comedy', 'Drama', 'Romance']},
 {'title': 'The Affair of the Necklace',
  'year': 2001,
  'rating': 6.1,
```

```
  'directors': ['Charles Shyer'],
  'actors': ['Simon Baker', 'Jonathan Pryce', 'Adrien Brody'],
  'genres': ['Drama', 'History', 'Romance']},
 {'title': 'The Barefoot Executive',
  'year': 1971,
  'rating': 6.0,
  'directors': ['Robert Butler'],
  'actors': ['Kurt Russell', 'Joe Flynn', 'Harry Morgan', 'Wally Cox'],
  'genres': ['Comedy', 'Family']}]
```

Question 13: what are the last 3 rows in movies?

The result should look like this:

```
[{'title': 'Fortitude and Glory: Angelo Dundee and His Fighters',
  'year': 2012,
  'rating': 7.2,
  'directors': ['Chris Tasara'],
  'actors': ['Angelo Dundee', 'George Foreman', 'Freddie Roach'],
  'genres': ['Sport']},
 {'title': 'Ivanhoe',
  'year': 1952,
  'rating': 6.8,
  'directors': ['Richard Thorpe'],
  'actors': ['Robert Taylor', 'George Sanders'],
  'genres': ['Adventure', 'Drama', 'History']},
 {'title': 'The Great Gatsby',
  'year': 1949,
  'rating': 6.6,
  'directors': ['Elliott Nugent'],
  'actors': ['Alan Ladd', 'Macdonald Carey'],
  'genres': ['Drama']}]
```

Copy the following function to your notebook, but don't change it in any way.

```python
# you are not allowed to change this function
def filter_movies_by_year(movies, year):
    i = 0
    while i < len(movies):
        if movies[i]["year"] != year:
            movies.pop(i)
        else:
            i += 1
    return movies
```

The `movies` parameter is for a list of movie dictionaries (similar to what is retured by `get_movies`) and `year` is a year to filter on. The function returns the movies in `movies` that were in the given year.

Question 14: what are the movies from 1930?

Requirements:

1. answer using `filter_movies_by_year`
2. do NOT call `get_movies` on "movies.csv" more than once in your notebook

The answer should look like this:

```
[{'title': 'Hook Line and Sinker',
  'year': 1930,
  'rating': 6.4,
  'directors': ['Edward F. Cline'],
  'actors': ['Bert Wheeler', 'Robert Woolsey', 'Ralf Harolde'],
  'genres': ['Comedy', 'Romance']},
 {'title': 'The Big Trail',
  'year': 1930,
  'rating': 7.2,
  'directors': ['Raoul Walsh', 'Louis R. Loeffler'],
  'actors': ['John Wayne', 'El Brendel', 'Tully Marshall'],
  'genres': ['Adventure', 'Romance', 'Western']}]
```

Question 15: what are the movies from 1931?

Hint: we've set you up a bit to encounter a bug. Review the copy functions in the `copy` module and see if you can use one of them to overcome the shortcomings of the `filter_movies_by_year` function we're forcing you to use. You might need to go back and tweak your q14 answer and potentially do a "Restart & Run All" on your notebook after you've fixed the bug.

If you get it right, you'll get this output:

```
[{'title': 'Arizona',
  'year': 1931,
  'rating': 6.0,
  'directors': ['George B. Seitz'],
  'actors': ['John Wayne', 'Forrest Stanley'],
  'genres': ['Drama', 'Romance']},
 {'title': 'City Lights',
  'year': 1931,
  'rating': 8.5,
  'directors': ['Charles Chaplin'],
  'actors': ['Charles Chaplin', 'Harry Myers'],
  'genres': ['Comedy', 'Drama', 'Romance']},
 {'title': 'The Range Feud',
  'year': 1931,
  'rating': 5.8,
  'directors': ['D. Ross Lederman'],
  'actors': ['Buck Jones', 'John Wayne', 'Edward LeSaint'],
  'genres': ['Mystery', 'Western']}]
```

Question 16: how many unique genres are there in the dataset?

Think about whether you can write a function that helps you with Q17 and Q18 at the same time.

Question 17: how many unique actor names are there in the dataset?

Question 18: how many unique director names are there in the dataset?

Question 19: what is the average movie rating?

Question 20: what is the longest movie title in the dataset (in terms of most characters)?