*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ (https://www.coursera.org/learn/python-social-network-analysis/resources/yPcBs) course resource.*

Assignment 1 - Creating and Manipulating Graphs

Eight employees at a small company were asked to choose 3 movies that they would most enjoy watching for the upcoming company movie night. These choices are stored in the file `Employee_Movie_Choices.txt`.

A second file, `Employee_Relationships.txt`, has data on the relationships between different coworkers.

The relationship score has value of `-100` (Enemies) to `+100` (Best Friends). A value of zero means the two employees haven't interacted or are indifferent.

Both files are tab delimited.

In [1]:
```
import networkx as nx
import pandas as pd
import numpy as np
from networkx.algorithms import bipartite

# This is the set of employees
employees = set(['Pablo',
                 'Lee',
                 'Georgia',
                 'Vincent',
                 'Andy',
                 'Frida',
                 'Joan',
                 'Claude'])

# This is the set of movies
movies = set(['The Shawshank Redemption',
              'Forrest Gump',
              'The Matrix',
              'Anaconda',
              'The Social Network',
              'The Godfather',
              'Monty Python and the Holy Grail',
              'Snakes on a Plane',
              'Kung Fu Panda',
              'The Dark Knight',
              'Mean Girls'])


# you can use the following function to plot graphs
# make sure to comment it out before submitting to the autograder

# def plot_graph(G, weight_name=None):
#     '''
#     G: a networkx G
#     weight_name: name of the attribute for plotting edge weights (if G is weighted)
#     '''
#     %matplotlib inline
#     import matplotlib.pyplot as plt

#     plt.figure()
#     pos = nx.spring_layout(G)
#     edges = G.edges()
#     weights = None

#     if weight_name:
#         weights = [int(G[u][v][weight_name]) for u,v in edges]
#         labels = nx.get_edge_attributes(G,weight_name)
#         nx.draw_networkx_edge_labels(G,pos,edge_labels=labels)
#         nx.draw_networkx(G, pos, edges=edges, width=weights);
#     else:
#         nx.draw_networkx(G, pos, edges=edges);
```

Question 1

Using NetworkX, load in the bipartite graph from `Employee_Movie_Choices.txt` and return that graph.

This function should return a networkx graph with 19 nodes and 24 edges

In [2]:
```
df = pd.read_csv('Employee_Movie_Choices.txt', sep='\t')
df.tail()
```

Out[2]:

	#Employee	Movie
19	Pablo	The Matrix
20	Pablo	The Shawshank Redemption
21	Vincent	The Godfather
22	Vincent	The Shawshank Redemption
23	Vincent	The Social Network

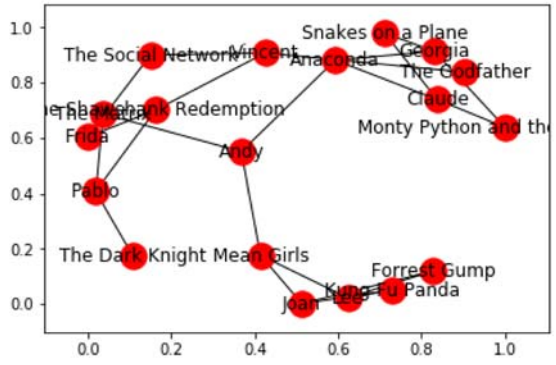
In [3]:
```
def answer_one():

    # Your Code Here

    G = nx.from_pandas_dataframe(df, '#Employee', 'Movie')

    return G
```

In [4]:
```
# plot_graph(answer_one(), weight_name=None)
```



Question 2

Using the graph from the previous question, add nodes attributes named `'type'` where movies have the value `'movie'` and employees have the value `'employee'` and return that graph.

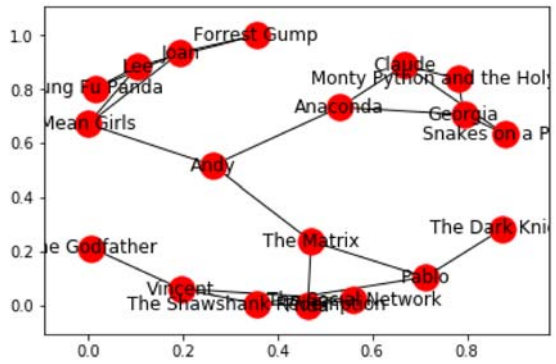
This function should return a networkx graph with node attributes `{'type': 'movie'}` or `{'type': 'employee'}`

In [5]:
```
def answer_two():

    # Your Code Here

    G = answer_one()
    for node in G.nodes():
        if node in employees:
            G.add_node(node, type='employee')
        elif node in movies:
            G.add_node(node, type='movie')
    return G
```

In [6]:
```
# plot_graph(answer_two(), weight_name=None)
```



Question 3

Find a weighted projection of the graph from `answer_two` which tells us how many movies different pairs of employees have in common.

This function should return a weighted projected graph.

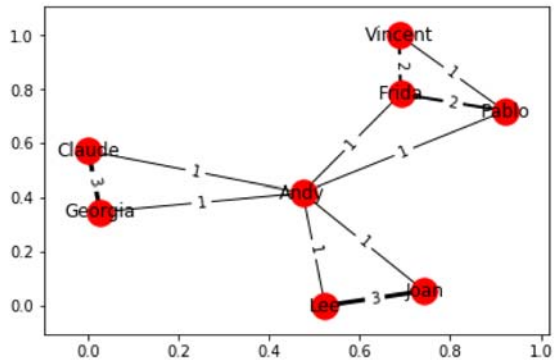
In [7]:
```
def answer_three():

    # Your Code Here

    G = answer_two()
    W = bipartite.weighted_projected_graph(G, employees)

    return W
```

In [8]:
```
# plot_graph(answer_three(), weight_name='weight')
```



Question 4

Suppose you'd like to find out if people that have a high relationship score also like the same types of movies.

Find the Pearson correlation (using `DataFrame.corr()`) between employee relationship scores and the number of movies they have in common. If two employees have no movies in common it should be treated as a 0, not a missing value, and should be included in the correlation calculation.

This function should return a float.

In [9]:
```
def answer_four():

    # Your Code Here

    R = nx.read_edgelist('Employee_Relationships.txt', data=[('relationship score', int)])
    R_df = pd.DataFrame(R.edges(data=True), columns=['From', 'To', 'relationship score'])

    G = answer_three()
    G_df = pd.DataFrame(G.edges(data=True), columns=['From', 'To', 'movies score'])
    G_copy_df = G_df.copy()

    G_copy_df.rename(columns={"From":"From_", "To":"From"}, inplace=True)
    G_copy_df.rename(columns={"From_":"To"}, inplace=True)
    G_final_df = pd.concat([G_df, G_copy_df])

    df = pd.merge(G_final_df, R_df, on = ['From', 'To'], how='right')
    df['movies score'] = df['movies score'].map(lambda x: x['weight'] if type(x)==dict else None)
    df['relationship score'] = df['relationship score'].map(lambda x: x['relationship score'])
    df['movies score'].fillna(value=0, inplace=True)

    Pearson_correlation = df['movies score'].corr(df['relationship score'])

    return Pearson_correlation

answer_four()
```

Out[9]:
```
0.78839622217334748
```