

---

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) (<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 notebook
    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]:

```
def answer_one():  
  
    # Your Code Here  
    # f = open('Employee_Movie_Choices.txt', 'r')  
    # file = f.read()  
    # f.close()  
    # edges = [tuple(x.split('\t')) for x in file.splitlines()][1:]  
    # G = nx.Graph()  
    # G.add_edges_from(edges)  
  
    G = nx.read_adjlist('Employee_Movie_Choices.txt', delimiter = '\t')  
  
    return G # Your Answer Here
```

## 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 [3]:

```
def answer_two():  
  
    # Your Code Here  
    G = answer_one()  
  
    for node in G.nodes():  
        G.add_node(node, type = 'movie' if node in movies else 'employee')  
  
    return G # Your Answer Here
```

## 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 [4]:

```
def answer_three():  
  
    # Your Code Here  
    G = answer_two()  
  
    G3 = bipartite.weighted_projected_graph(G, employees)  
  
    return G3 # Your Answer Here
```

## 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 [5]:

```
G3 = answer_three()
def weight(x,y):
    try:
        return G3[x][y]['weight']
    except:
        return 0
```

In [6]:

```
def answer_four():

    # Your Code Here
    G4 = nx.read_edgelist('Employee_Relationships.txt', delimiter = '\t', data = [('relatio
    for edge in G4.edges():
        G4.add_edge(edge[0], edge[1], weight = weight(edge[0], edge[1]))
    rel = [x[2]['relation'] for x in G4.edges(data = True)]
    wgh = [x[2]['weight'] for x in G4.edges(data = True)]
    df4 = pd.DataFrame(np.column_stack([rel, wgh]), columns=['rel', 'wgh'])
    ans = df4['rel'].corr(df4['wgh'])

    return ans # Your Answer Here
```