```
#####################################################################
punctuation_chars = ["'", '"', ",", ".", "!", ":", ";", '#', '@']
def strip_punctuation(astrg):
    for char in astrg:
        if char in punctuation_chars:
            astrg=astrg.replace(char,'')
    return astrg
# lists of words to use
positive_words = []
with open("positive_words.txt") as pos_f:
    for lin in pos_f:
        if lin[0] != ';' and lin[0] != '\n':
            positive_words.append(lin.strip())
negative_words = []
with open("negative_words.txt") as pos_f:
    for lin in pos_f:
        if lin[0] != ';' and lin[0] != '\n':
            negative_words.append(lin.strip())
def get_pos(insentenses):
    n_pos=0
    insentenses=strip_punctuation(insentenses)
    sentenses=insentenses.lower()
    words=sentenses.split()
    for wrd in words:
        if wrd in positive_words:
            n_pos+=1
    return n_pos
def get_neg(insentenses):
    n_neg=0
    insentenses=strip_punctuation(insentenses)
    sentenses=insentenses.lower()
```

```
    words=sentenses.split()
    for wrd in words:
        if wrd in negative_words:
            n_neg+=1
    return n_neg
f_in=open("project_twitter_data.csv","r")
f_out=open("resulting_data.csv","w")
header="Number of Retweets, Number of Replies, Positive Score, Negative Score, Net Score"
f_out.write(header)
f_out.write('\n')
glue=','
lines_in=f_in.readlines()
for line in lines_in[1:]:
    lin=line.strip()
    lin_in=lin.split(',')
    pos_score=get_pos(lin_in[0])
    neg_score=get_neg(lin_in[0])
    net_score=pos_score-neg_score
    n_ret=lin_in[1]
    n_rep=lin_in[2]
    #f_out.write('\n')
    wrds=[str(n_ret),str(n_rep),str(pos_score),str(neg_score),str(net_score)]
    lin_out=glue.join(wrds)
    f_out.write(lin_out)
    f_out.write('\n')
f_in.close()
f_out.close()
#####################################################################
# get_sorted_recommendations(["Bridesmaids", "Sherlock Holmes"])
import requests_with_caching
import json
```

```
def get_movies_from_tastedive(astg):
    kval_pairs={'q':astg,'type':'movies','limit':5}
    page=requests_with_caching.get("https://tastedive.com/api/similar",params=kval_pairs)
    #print(page.url)
    x=page.json()
    #print(json.dumps(x,indent=2))
    return x
adict=get_movies_from_tastedive("Black Panther")
def extract_movie_titles(mydict):
    d1=mydict["Similar"]
    lst=d1["Results"]
    lst_names= [dic["Name"] for dic in lst]
    return lst_names
lst_movie_titles=extract_movie_titles(adict)
def get_related_titles(in_lst):
    out_lst=[]
    for title in in_lst:
        adict=get_movies_from_tastedive(title)
        lst_movie_titles=extract_movie_titles(adict)
        for item in list(lst_movie_titles):
            out_lst.append(item)
    return out_lst
def get_movie_data(astg):
    kval_pairs={'t':astg,'r':'json'}
    page=requests_with_caching.get('http://www.omdbapi.com/',params=kval_pairs)
    print(page.url)
    #print (page.text[:20])
    x=page.json()
    #print(json.dumps(x))
    return x
def get_movie_rating(a_dict):
```

```
    a_lst=a_dict["Ratings"]
    rating_num=0
    for item in a_lst:
        if item["Source"]=="Rotten Tomatoes":
            rating_v=item["Value"]
            rating_num=int(rating_v[0:2])
    return rating_num
def get_sorted_recommendations(a_lst):
    new_lst=get_related_titles(a_lst)
    new_dict={}
    for item in new_lst:
        new_dict[item]=get_movie_rating(get_movie_data(item))
    return sorted(new_dict, key=lambda k:new_dict[k],reverse=True)
#######################################################################
class WOFPlayer():
    def __init__(self,name):
        self.name=name
        self.prizeMoney=0
        self.prizes=[]
    def addMoney(self,amt):
        self.prizeMoney=self.prizeMoney+amt
    def goBankrupt(self):
        self.prizeMoney=0
    def addPrize(self,prize):
        self.prizes.append(prize)
    def __str__(self):
        return '{} (${})'.format(self.name,self.prizeMoney)
class WOFHumanPlayer(WOFPlayer):
    def getMove(self, category, obscuredPhrase, guessed):
        print('{} has ${}\n'.format(self.name,self.prizeMoney))
        print('Category: {}'.format(category))
```

```
        print('Phrase: {}'.format(obscuredPhrase))
        print('Guessed: {}\n'.format(guessed))
        ipt=input("Guess a letter, phrase, or type 'exit' or 'pass':")
        return ipt
class WOFComputerPlayer(WOFPlayer):
    def __init__(self,name,difficulty):
        self.name=name
        self.difficulty=difficulty
        self.prizeMoney=0
        self.prizes=[]
    #SORTED_FREQUENCIES='ZQXJKVBPYGFWMUCLDRHSNIOATE'
    def smartCoinFlip(self):
        rand_num=random.randint(1,10)
        if rand_num>self.difficulty:
            return False
        else:
            return True
    def getPossibleLetters(self,guessed):
        possib_Letters=[]
        for l in LETTERS:
            if l not in guessed:
                if (l not in VOWELS) or (l in VOWELS and self.prizeMoney>=VOWEL_COST):
                    possib_Letters.append(l)
        return possib_Letters
    def getMove(self, category, obscuredPhrase, guessed):
        SORTED_FREQUENCIES='ZQXJKVBPYGFWMUCLDRHSNIOATE'
        p_letters=self.getPossibleLetters(guessed)
        if p_letters==[]:
            return 'pass'
        cointFlip=self.smartCoinFlip()
        if cointFlip==True:
```

```
        for l in SORTED_FREQUENCIES[-1:]:
            if l in p_letters:
                good_move=l
                return good_move
                break
        #return good_move
    elif cointFlip==False:
        bad_move = random.choice(p_letters)
        return bad_move
import sys
sys.setExecutionLimit(600000) # let this take up to 10 minutes
import json
import random
import time
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
VOWELS  = 'AEIOU'
VOWEL_COST  = 250
# Repeatedly asks the user for a number between min & max (inclusive)
def getNumberBetween(prompt, min, max):
  userinp = input(prompt) # ask the first time
  while True:
    try:
      n = int(userinp) # try casting to an integer
      if n < min:
        errmessage = 'Must be at least {}'.format(min)
      elif n > max:
        errmessage = 'Must be at most {}'.format(max)
      else:
        return n
    except ValueError: # The user didn't enter a number
      errmessage = '{} is not a number.'.format(userinp)
```

```
        # If we haven't gotten a number yet, add the error message
        # and ask again
        userinp = input('{}\n{}'.format(errmessage, prompt))
# Spins the wheel of fortune wheel to give a random prize
# Examples:
#    { "type": "cash", "text": "$950", "value": 950, "prize": "A trip to Ann Arbor!" },
#    { "type": "bankrupt", "text": "Bankrupt", "prize": false },
#    { "type": "loseturn", "text": "Lose a turn", "prize": false }
def spinWheel():
    with open("wheel.json", 'r') as f:
        wheel = json.loads(f.read())
        return random.choice(wheel)
# Returns a category & phrase (as a tuple) to guess
# Example:
#     ("Artist & Song", "Whitney Houston's I Will Always Love You")
def getRandomCategoryAndPhrase():
    with open("phrases.json", 'r') as f:
        phrases = json.loads(f.read())
        category = random.choice(list(phrases.keys()))
        phrase   = random.choice(phrases[category])
        return (category, phrase.upper())
# Given a phrase and a list of guessed letters, returns an obscured version
# Example:
#     guessed: ['L', 'B', 'E', 'R', 'N', 'P', 'K', 'X', 'Z']
#     phrase:  "GLACIER NATIONAL PARK"
#     returns> "_L___ER N____N_L P_RK"
def obscurePhrase(phrase, guessed):
    rv = ''
    for s in phrase:
        if (s in LETTERS) and (s not in guessed):
            rv = rv+'_'
```

```
        else:
            rv = rv+s
    return rv
# Returns a string representing the current state of the game
def showBoard(category, obscuredPhrase, guessed):
    return """
Category: {}
Phrase:   {}
Guessed:  {}""".format(category, obscuredPhrase, ', '.join(sorted(guessed)))
# GAME LOGIC CODE
print('='*15)
print('WHEEL OF PYTHON')
print('='*15)
print('')
num_human = getNumberBetween('How many human players?', 0, 10)
# Create the human player instances
human_players = [WOFHumanPlayer(input('Enter the name for human player #{}'.format(i+1))) for i
in range(num_human)]
num_computer = getNumberBetween('How many computer players?', 0, 10)
# If there are computer players, ask how difficult they should be
if num_computer >= 1:
    difficulty = getNumberBetween('What difficulty for the computers? (1-10)', 1, 10)
# Create the computer player instances
computer_players = [WOFComputerPlayer('Computer {}'.format(i+1), difficulty) for i in
range(num_computer)]
players = human_players + computer_players
# No players, no game :(
if len(players) == 0:
    print('We need players to play!')
    raise Exception('Not enough players')
# category and phrase are strings.
category, phrase = getRandomCategoryAndPhrase()
```

```
# guessed is a list of the letters that have been guessed
guessed = []
# playerIndex keeps track of the index (0 to len(players)-1) of the player whose turn it is
playerIndex = 0
# will be set to the player instance when/if someone wins
winner = False
def requestPlayerMove(player, category, guessed):
    while True: # we're going to keep asking the player for a move until they give a valid one
        time.sleep(0.1) # added so that any feedback is printed out before the next prompt
        move = player.getMove(category, obscurePhrase(phrase, guessed), guessed)
        move = move.upper() # convert whatever the player entered to UPPERCASE
        if move == 'EXIT' or move == 'PASS':
            return move
        elif len(move) == 1: # they guessed a character
            if move not in LETTERS: # the user entered an invalid letter (such as @, #, or $)
                print('Guesses should be letters. Try again.')
                continue
            elif move in guessed: # this letter has already been guessed
                print('{} has already been guessed. Try again.'.format(move))
                continue
            elif move in VOWELS and player.prizeMoney < VOWEL_COST: # if it's a vowel, we need to be
sure the player has enough
                print('Need ${} to guess a vowel. Try again.'.format(VOWEL_COST))
                continue
            else:
                return move
        else: # they guessed the phrase
            return move
while True:
    player = players[playerIndex]
    wheelPrize = spinWheel()
```

```
print('')
print('-'*15)
print(showBoard(category, obscurePhrase(phrase, guessed), guessed))
print('')
print('{} spins...'.format(player.name))
time.sleep(2) # pause for dramatic effect!
print('{}!'.format(wheelPrize['text']))
time.sleep(1) # pause again for more dramatic effect!
if wheelPrize['type'] == 'bankrupt':
    player.goBankrupt()
elif wheelPrize['type'] == 'loseturn':
    pass # do nothing; just move on to the next player
elif wheelPrize['type'] == 'cash':
    move = requestPlayerMove(player, category, guessed)
    if move == 'EXIT': # leave the game
        print('Until next time!')
        break
    elif move == 'PASS': # will just move on to next player
        print('{} passes'.format(player.name))
    elif len(move) == 1: # they guessed a letter
        guessed.append(move)
        print('{} guesses "{}"'.format(player.name, move))
        if move in VOWELS:
            player.prizeMoney -= VOWEL_COST
        count = phrase.count(move) # returns an integer with how many times this letter appears
        if count > 0:
            if count == 1:
                print("There is one {}".format(move))
            else:
                print("There are {} {}'s".format(count, move))
```

```
                # Give them the money and the prizes
                player.addMoney(count * wheelPrize['value'])
                if wheelPrize['prize']:
                    player.addPrize(wheelPrize['prize'])
                # all of the letters have been guessed
                if obscurePhrase(phrase, guessed) == phrase:
                    winner = player
                    break
                continue # this player gets to go again
            elif count == 0:
                print("There is no {}".format(move))
        else: # they guessed the whole phrase
            if move == phrase: # they guessed the full phrase correctly
                winner = player
                # Give them the money and the prizes
                player.addMoney(wheelPrize['value'])
                if wheelPrize['prize']:
                    player.addPrize(wheelPrize['prize'])
                break
            else:
                print('{} was not the phrase'.format(move))
    # Move on to the next player (or go back to player[0] if we reached the end)
    playerIndex = (playerIndex + 1) % len(players)
if winner:
    # In your head, you should hear this as being announced by a game show host
    print('{} wins! The phrase was {}'.format(winner.name, phrase))
    print('{} won ${}'.format(winner.name, winner.prizeMoney))
    if len(winner.prizes) > 0:
        print('{} also won:'.format(winner.name))
        for prize in winner.prizes:
            print('    - {}'.format(prize))
```

```
else:
    print('Nobody won. The phrase was {}'.format(phrase))
#####################################################################
```