Homework Turnin

Name:	Alexander A Miller
Email:	alexandermiller@email.arizona.edu
Section:	1E
Course:	CS 120 17au
Assignment:	hw5
Receipt ID:	4032d5a88fdd2ec3a3d25bad9161bc44

Turnin Successful!

The following file(s) were received:

bball.py (14275 bytes)

```
"""
File: bball.py
Author: Alexander Miller
Purpose:
* read in name of file from input
* for each non-comment line in file:
    * create Team object from line
    * extract appropriate data from lines
    * compute win ratio for team
    * initialize team objects accordingly
    * add team to list of teams in conference
    * print out results subject to output requirements

"""

class Team:
    def __init__(self, line, team_number, char_index_1):
        """
        Description: initializes team object
        Parameters: line (from file), team_number (organizational purposes),\
        char_index_1 (indicating end of team name)
        Returns: none
        Pre-condition: parameters exist and are of proper types \
        (str, int, int respectively)
        Post-Condition: team object is initialized
        """
        team_name = ''
        # objective: get team names
        # char_index_1 is where conf name starts, so can end before to get team name
        team_name = line[:char_index_1-1]
        assert len(team_name) < len(line)
        assert type(team_name) == str
        # objective: clean up team_name
        x = team_name
        x = whitespace_stripper(x)
        team_name = x
        self._team_name = team_name
        self._win_ratio = 0
        self._conference_name = ''
        self._team_number = team_number
# getters:
    def conf(self):
        """
        Description: returns name of conference
        Parameters: none
        Returns: name of conference
        Pre-condition: conference has a name
        Post-Condition: returns name of conference
        """
        return self._conference_name
    def name(self):
        """
        Description: returns name of team
        Parameters: none
        Returns: name of team
        Pre-condition: team has a name
        Post-Condition: returns name of team
        """
        return self._team_name
    def number(self):
        """
        Description: returns team number
        Parameters: none
        Returns: team number
        Pre-condition: team has a number
        Post-Condition: returns team number
        """
        return self._team_number
    def win_ratio(self):
        """
        Description: returns win ratio
        Parameters: none
        Returns: win ratio
        Pre-condition: team has a win ratio
        Post-Condition: returns win ratio
        """
        return self._win_ratio
# setters:
    def set_ratio(self,ratio):
        """
        Description: sets win ratio
        Parameters: ratio
        Returns: none
        Pre-condition: none
        Post-Condition: set win ratio
        """
        self._win_ratio = ratio
    def update_conference_name(self,conf):
        """
        Description: updates conference name
        Parameters: conf
        Returns: none
        Pre-condition: conf is a string
        Post-Condition: conference name updated
        """
        self._conference_name = conf
# misc:
    def __str__(self):
        """
        Description: returns print description
        Parameters: none
        Returns: self._team_name
        Pre-condition: team has a name
        Post-Condition: returns print description
        """
        return self._team_name


class Conference:
    def __init__(self,conf, line_number):
        """
        Description: initializes conference object
        Parameters: conf, line_number
        Returns: none
        Pre-condition: conf is a string, line_number is an integer
        Post-Condition: conference object initialized
        """
        assert type(conf) == str
        assert type(line_number) == int
        self._conference_name = conf
        self._line_number = [line_number]
        self._team_list = []
        self._win_ratio_avg = 0
        self._length = 0
# getters:
    def name(self):
        """
        Description: returns name of conference
        Parameters: none
        Returns: self._conference_name
        Pre-condition: conference has name
        Post-Condition: returned name of conference
        """
        return self._conference_name
    def line_number(self):
        """
        Description: returns line_number
        Parameters: none
        Returns: self._line_number
        Pre-condition: conference has line_number
        Post-Condition: returned line_number
        """
        return self._line_number
    def get_teams(self):
        """
        Description: returns team list
        Parameters: none
        Returns: self._team_list
        Pre-condition: has team list
        Post-Condition: none
        """
        return self._team_list
    def win_ratio_avg(self):
        """
        Description: returns win ratio average
        Parameters: none
        Returns: self._win_ratio_avg
        Pre-condition: has a win ratio average
        Post-Condition: returned win ratio average
        """
        return self._win_ratio_avg
# setters:
    def update_line_number(self, line_number):
        """
        Description: updates line number
        Parameters: line_number
        Returns: none
        Pre-condition: has line number
        Post-Condition: updated line number
        """
        self._line_number += [line_number]
    def add(self,team):
        """
        Description: adds team
        Parameters: team
        Returns: none
        Pre-condition: team exists
        Post-Condition: added team
        """
        self._team_list = self._team_list + [team]
    def update_conf_length(self):
        """
        Description: updates conf length
        Parameters: none
        Returns: none
        Pre-condition: none
        Post-Condition: conf length updated
        """
        self._length += 1
    def set_win_ratio_avg(self,avg):
        """
        Description: sets win ratio average
        Parameters: avg
        Returns: none
        Pre-condition: avg exists
        Post-Condition: average updated
        """
        self._win_ratio_avg = avg
```

```
# misc:
    def __str__(self):
        """
        Description: gives print description
        Parameters: none
        Returns: self._conference_name
        Pre-condition: none
        Post-Condition: gave print description
        """
        return self._conference_name
    def __eq__(self, conf):
        """
        Description: gives equality description
        Parameters: conf
        Returns: conf == self._conference_name
        Pre-condition: conf exists
        Post-Condition: gave equality description
        """
        return conf == self._conference_name
    def __len__(self):
        """
        Description: returns length
        Parameters: none
        Returns: self._length
        Pre-condition: has length
        Post-Condition: returned length
        """
        return self._length


class ConferenceSet:
    def __init__(self):
        """
        Description: initializes conference set object
        Parameters: none
        Returns: none
        Pre-condition: none
        Post-Condition: initialized conference set object
        """
        self._conference_set = []
        self._conference_name_set = []
# getters:
    def get_objects(self):
        """
        Description: returns objects in conference set
        Parameters: none
        Returns: self._conference_set
        Pre-condition: none
        Post-Condition: returned objects
        """
        return self._conference_set
    def get_names(self):
        """
        Description: returns names of objects
        Parameters: none
        Returns: self._conference_name_set
        Pre-condition: none
        Post-Condition: returned names of objects
        """
        return self._conference_name_set

    def best(self):
        """
        Description: calculates and prints confs with best win ratio
        Parameters: none
        Returns: none
        Pre-condition: conf win ratios fully updated
        Post-Condition: program finished objectives
        """
        max_val = 0
        winners_list = []
        for i in self.get_objects():
            assert type(i.win_ratio_avg()) == float or int
            if i.win_ratio_avg() > max_val:
                max_val = i.win_ratio_avg()
        for i in self.get_objects():
            if i.win_ratio_avg() >= max_val:
                winners_list += [i]
        for x in winners_list:
            print("{} : {}".format(x.name(),x.win_ratio_avg()))
# setters:
    def add(self,conf):
        """
        Description: adds conf
        Parameters: conf
        Returns: none
        Pre-condition: conf exists
        Post-Condition: added conf object and name
        """
        self._conference_set += [conf]
        self._conference_name_set += [conf._conference_name]
# misc:
    def __str__(self):
        """
        Description: returns print description
        Parameters: none
        Returns: print_word
        Pre-condition: none
        Post-Condition: returned print description
        """
        print_word = ''
        for i in self._conference_set:
            print_word = print_word + '\n' + i._conference_name
        return print_word
    def __contains__(self,conf):
        """
        Description: returns contains description
        Parameters: conf
        Returns: conf in self._conference_name_set
        Pre-condition: none
        Post-Condition: returned contains description
        """
        return conf in self._conference_name_set


def main():
    conference_list = ConferenceSet()
    assert type(conference_list.get_names()) == list
    conference_processing(conference_list)
    ### INVARIANT: elements of conference_list are objects
    find_best_teams(conference_list)

def whitespace_stripper(x):
    """
    Description: strips whitespace from argument
    Parameters: x
    Returns: x
    Pre-condition: x is string of length 1 or greater
    Post-Condition: stripped whitespace from argument
    """
    assert type(x) == str
    while x[-1] == ' ':
        x = x.rstrip()
    while x[0] == ' ':
        x = x.lstrip()
    assert x[-1] != ' ' and x[0] != ' '
    return x

def conference_processing(conference_list):
    """
    Description: processes file, gets conferences, adds to conf list,
    \gets wins and losses, gets and places teams in confs, sets win records
    Parameters: conference_list
    Returns: none
    Pre-condition: conference_list exists
    Post-Condition: objectives accomplished (see above)
    """
    filename = str(input())
    ### ASSUMPTION: file is readable and data is in correct position
    openfile = open(filename)
    # objective: get conferences
    wins_and_losses = []
    line_number = 0
    team_number = 0
    for line in openfile:
        if line[0] != '#':
            line_number += 1
            # objective: get conferences
            accum = 0
            for char in line:
                assert type(char) == str
# for the char_indices: will want lattermost in any case so it's ok
# if it overwrites in event of multiple parenthesis sets
                if char == '(':
                    char_index_1 = accum+1
                    #we don't want parenthesis itself included in range
                if char == ')':
                    char_index_2 = accum
                accum = accum + 1
            conf = line[char_index_1:char_index_2]
            assert len(conf) < len(line)
            # objective: clean conf
            x = conf
            x = whitespace_stripper(x)
            conf = x
            # objective: create and add unique conf to conference_list
            if conf not in conference_list.get_names():
                conf = Conference(conf, line_number)
                conference_list.add(conf)
                assert conf in conference_list.get_names()
            else:
                for i in conference_list.get_objects():
                    if conf == i:
                        i.update_line_number(line_number)
                        assert type(i.line_number()) == list
            # objective: get wins & losses
            rest_of_line = line[char_index_2+1:]
            rest_of_line_list = rest_of_line.split()
            assert len(rest_of_line_list) < len(line)
            cleaned_list = []
            #cleaning results
            for i in rest_of_line_list:
                if i != (' ' or ''):
                    cleaned_list += [i]
            for z in cleaned_list:
                x = z
                x = whitespace_stripper(x)
                z = x
            wins_and_losses += [cleaned_list]
            # objective: get teams and place in conferences; set data
            team_number += 1 # initializes at "team number 1"
            team = Team(line, team_number, char_index_1)
            for i in conference_list.get_objects():
                if team.number() in i.line_number():
                    i.add(team)
                    i.update_conf_length()
                    conf = i.name()
                    team.update_conference_name(conf)
                    assert team in i.get_teams()
            # objective: get their win record
            ### ASSUMPTION: wins come before losses in data file
            sublist = wins_and_losses[team.number()-1] # want team 1 at pos. 0 etc.
            sublist[0] = int(sublist[0])
            sublist[1] = int(sublist[1])
            if sublist[1]+sublist[0] != 0:
                ### INVARIANT: team has played games this season
                ratio = sublist[0]/(sublist[1]+sublist[0])
            else:
                ratio = 0
            team.set_ratio(ratio)
    openfile.close()



def find_best_teams(conference_list):
    """
    Description: finds best conferences and prints them out
    Parameters: conference_list
    Returns: none
    Pre-condition: conference_list is complete and updated
    Post-Condition: none
    """
    # objective: set average win ratio for each conference
    for i in conference_list.get_objects():
        if len(i) > 0:
            ratio_sum = 0
            for x in i.get_teams():
                assert type(x.win_ratio()) == float or int
                ratio_sum += x.win_ratio()
            avg = ratio_sum/len(i)
            i.set_win_ratio_avg(avg)
    # objective: find best conferences
    conference_list.best()



main()
```

ngrams.py (5443 bytes)

```
"""
File: ngrams.py
Author: Alexander Miller
Purpose:
* read file and n from input without prompt
    * create list of words, split at whitespace
    * strip punctuation from ends of words; then discard empty strings
* construct n-grams and count occurences (case insensitive)
    * find maximum occurring n-grams
* print one per line using format statement
    * will need to separate words in n-gram by whitespace
* assertions:
    * any preconditions for all methods
    * one assert per substantive loop (compute value; transform data)
* use class system detailed in specs

"""

import string

class Input:
    def __init__(self):
        """
        Description: initializes file object
        Parameters: none
        Returns: a file object
        Pre-condition: file exists
        Post-Condition: file object will be object
        """
        ### ASSUMPTION: file is readable
        filename = str(input())
        self._openfile = open(filename)

    def close(self):
        """
        Description: closes file object
        Parameters: none
        Returns: none
        Pre-condition: file object is open
        Post-Condition: closes file object
        """
        self._openfile.close()

    def wordlist(self):
        """
        Description: processes document to acquire data
        Parameters: none
        Returns: none
        Pre-condition: file is readable
        Post-Condition: self._list will be a list containing necessary data
        """
        ### ASSUMPTION: file is readable
        self._openfile
        file_list = []
        for line in self._openfile:
            line_list = line.split()
            accum = 0
            while accum != len(line_list):
                try:
                    while line_list[accum][-1] in string.punctuation:
                        line_list[accum] = \
                                         line_list[accum].rstrip(line_list[accum][-1])
                    while line_list[accum][0] in string.punctuation:
                        line_list[accum] = \
                                         line_list[accum].lstrip(line_list[accum][0])
                    # assert to check that strips worked correctly
                    assert line_list[accum][-1] and \
                           line_list[accum][0] not in string.punctuation
                except IndexError:
                    pass
                line_list[accum] = line_list[accum].lower()
                if line_list[accum] != '':
                    file_list = file_list + [line_list[accum]]
                # assert to check that no list elements are blankspace
                for i in file_list:
                    assert i != ''
                accum = accum + 1
        self.close()
        self._list = file_list # new list : file._list
        assert len(self._list) == len(file_list)


class Ngrams:
    def __init__(self):
        """
        Description: initializes ngrams object
        Parameters: none
        Returns: none
        Pre-condition: file._list exists
        Post-Condition: ngrams object will be initialized
        """
        n = int(input())
        assert type(n)==int
        self._count = 0
        self._n = n
        self._dict = {}
        self._winnerslist = []
    def update(self,i):
        """
        Description: updates count
        Parameters: i, element of self._dict
        Returns: none
        Pre-condition: i can be indexed
        Post-Condition: count will be updated
        """
        self._count = self._dict[i][1]
    def __str__(self):
        """
        Description: gives print instructions
        Parameters: none
        Returns: self._dict
        Pre-condition: none
        Post-Condition: none
        """
        return self._dict
    def process_wordlist(self, other):
        """
        Description: processes list of data to count ngrams
        Parameters: other, an object
        Returns: none
        Pre-condition: other contains a list with necessary data
        Post-Condition: will have printed most numerous ngrams
        """
        assert type(other._list) == list
        accum = 0
        while accum != len(other._list):
            cgram = other._list[accum:accum + self._n]
            assert type(cgram) == list
            cgram_phrase = ' '.join(cgram)
            if len(cgram) == self._n: #removes shorter cgrams from very end of file
                if cgram_phrase not in self._dict:
                    self._dict[cgram_phrase] = [cgram_phrase,1]
                else:
                    self._dict[cgram_phrase][1] += 1
            assert len(self._dict) <= len(other._list)
            accum = accum + 1
        # establishing max. occurrences
        for i in self._dict:
            assert type(i) == str
            if self._dict[i][1] > self._count:
                self.update(i)
        for x in self._dict:
            assert self._count >= self._dict[x][1]
            if self._dict[x][1] == self._count:
                self._winnerslist = self._winnerslist + [self._dict[x][0]]
    def print_max_ngrams(self):
        if self._n != 0:
            for i in self._winnerslist:
                print("{:d} -- {}".format(self._count, i))
        else:
            pass


def main():
    file = Input()
    file.wordlist()
    ngram = Ngrams()
    ngram.process_wordlist(file)
    ngram.print_max_ngrams()


main()
```