Contents

1 Basic Test Results

```
Starting tests...
Thu Dec 19 20:27:27 IST 2013
70c90d627db27e03c74a6b7c5574252083aa3976  -


tweet.py
README
nation_mood.py
geo_tweet_tools.py

Testing README...
Done testing README...

Testing tweet.py...
result_code    tweettext    68    1
result_code    tweetwords    68    1
result_code    tweettime    68    1
result_code    tweetesent    68    1
result_code    tweetloc    68    1
result_code    tweetsent    68    1
result_code    tweetmult    85    1
Done testing tweet.py

Testing geo_tweet_tools.py...
result_code    groupcontain    37    1
result_code    center    58    1
result_code    centroid    121    1
result_code    groupclose    37    1
Wrong result:
expected: ['WA', 'WA', 'TX', 'TX', 'NC', 'NY', 'MO', 'MD']
actual:   ['WA', 'WA', 'TX', 'TX', 'NC', 'NY', 'MO', None]
result_code    contains_8    wrong    1
Wrong result:
expected: ['FL', 'RI', 'SC', 'CA', 'DE', 'DE', 'IL', 'CA']
actual:   ['FL', None, 'SC', 'CA', 'DE', 'DE', 'IL', 'CA']
result_code    contains_10    wrong    1
Wrong result:
expected: ['VA', None, 'NJ', 'TX', 'FL', 'TN', 'GA', 'TX']
actual:   [None, None, 'NJ', 'TX', 'FL', 'TN', 'GA', 'TX']
result_code    contains_13    wrong    1
result_code    contains    11    1
result_code    closest    14    1
Done testing geo_tweet_tools.py

Testing nation_mood.py...
result_code    byhour    1830    1
result_code    talkative    12    1
result_code    average    14    1
Done testing nation_mood.py

Grading summary
---------------
****** tweettext:
Number of failed tests: 0
Total number of tests : 68
Penalty: 0.0
****** tweettime:
Number of failed tests: 0
Total number of tests : 68
```

```
Penalty: 0.0
****** tweetloc:
Number of failed tests: 0
Total number of tests : 68
Penalty: 0.0
****** tweetwords:
Number of failed tests: 0
Total number of tests : 68
Penalty: 0.0
****** tweetsent:
Number of failed tests: 0
Total number of tests : 68
Penalty: 0.0
****** tweetesent:
Number of failed tests: 0
Total number of tests : 68
Penalty: 0.0
****** tweetmult:
Number of failed tests: 0
Total number of tests : 85
Penalty: 0.0
****** centroid:
Number of failed tests: 0
Total number of tests : 121
Penalty: 0.0
****** center:
Number of failed tests: 0
Total number of tests : 58
Penalty: 0.0
****** groupclose:
Number of failed tests: 0
Total number of tests : 37
Penalty: 0.0
****** groupcontain:
Number of failed tests: 0
Total number of tests : 37
Penalty: 0.0
****** closest:
Number of failed tests: 0
Total number of tests : 14
Penalty: 0.0
****** talkative:
Number of failed tests: 0
Total number of tests : 12
Penalty: 0.0
****** average:
Number of failed tests: 0
Total number of tests : 14
Penalty: 0.0
****** byhour:
Number of failed tests: 0
Total number of tests : 1830
Penalty: 0.0
****** contains:
Number of passed tests: 11
Total number of tests : 14
Bonus: 7.857142857142857
******
Expected automatic grade: 107.85714285714286
******
Submission passed!
Tests completed
```

2 aaa expected autograde

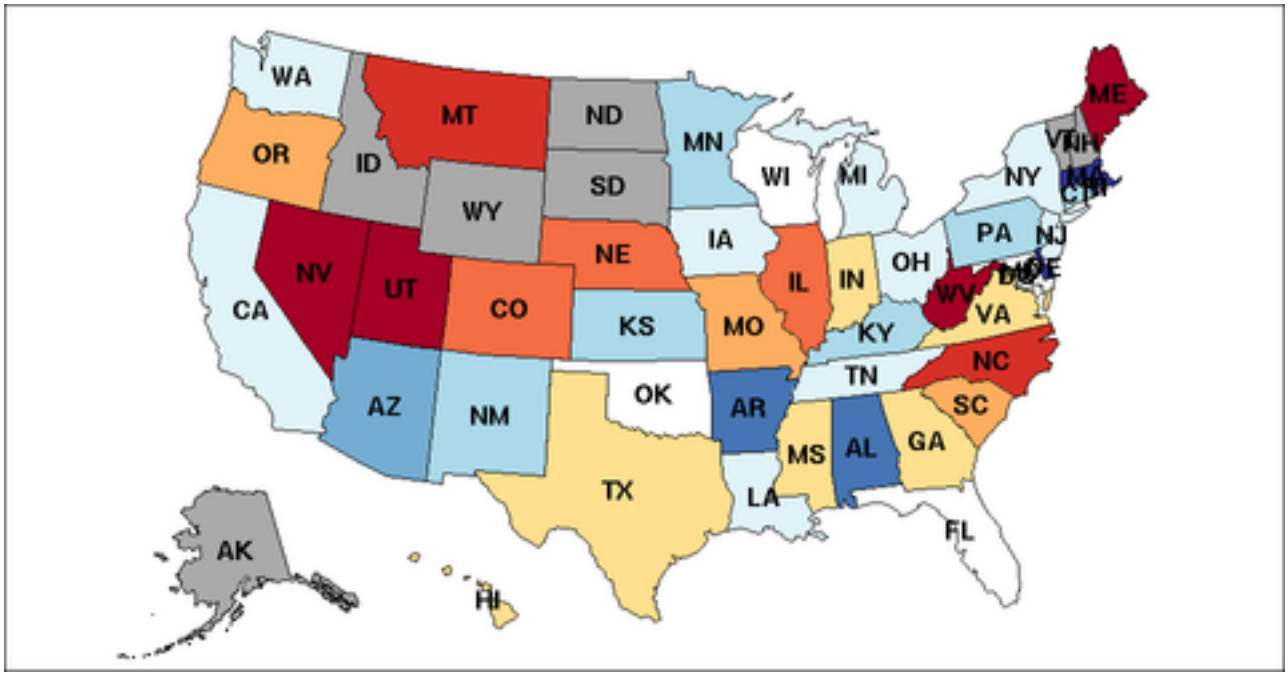
```
Grading summary
---------------
****** tweettext:
Number of failed tests: 0
Total number of tests : 68
Penalty: 0.0
****** tweettime:
Number of failed tests: 0
Total number of tests : 68
Penalty: 0.0
****** tweetloc:
Number of failed tests: 0
Total number of tests : 68
Penalty: 0.0
****** tweetwords:
Number of failed tests: 0
Total number of tests : 68
Penalty: 0.0
****** tweetsent:
Number of failed tests: 0
Total number of tests : 68
Penalty: 0.0
****** tweetesent:
Number of failed tests: 0
Total number of tests : 68
Penalty: 0.0
****** tweetmult:
Number of failed tests: 0
Total number of tests : 85
Penalty: 0.0
****** centroid:
Number of failed tests: 0
Total number of tests : 121
Penalty: 0.0
****** center:
Number of failed tests: 0
Total number of tests : 58
Penalty: 0.0
****** groupclose:
Number of failed tests: 0
Total number of tests : 37
Penalty: 0.0
****** groupcontain:
Number of failed tests: 0
Total number of tests : 37
Penalty: 0.0
****** closest:
Number of failed tests: 0
Total number of tests : 14
Penalty: 0.0
****** talkative:
Number of failed tests: 0
Total number of tests : 12
Penalty: 0.0
****** average:
Number of failed tests: 0
Total number of tests : 14
Penalty: 0.0
****** byhour:
```

```
Number of failed tests: 0
Total number of tests : 1830
Penalty: 0.0
****** contains:
Number of passed tests: 11
Total number of tests : 14
Bonus: 7.857142857142857
******
Expected automatic grade: 107.85714285714286
******
Submission passed!
```

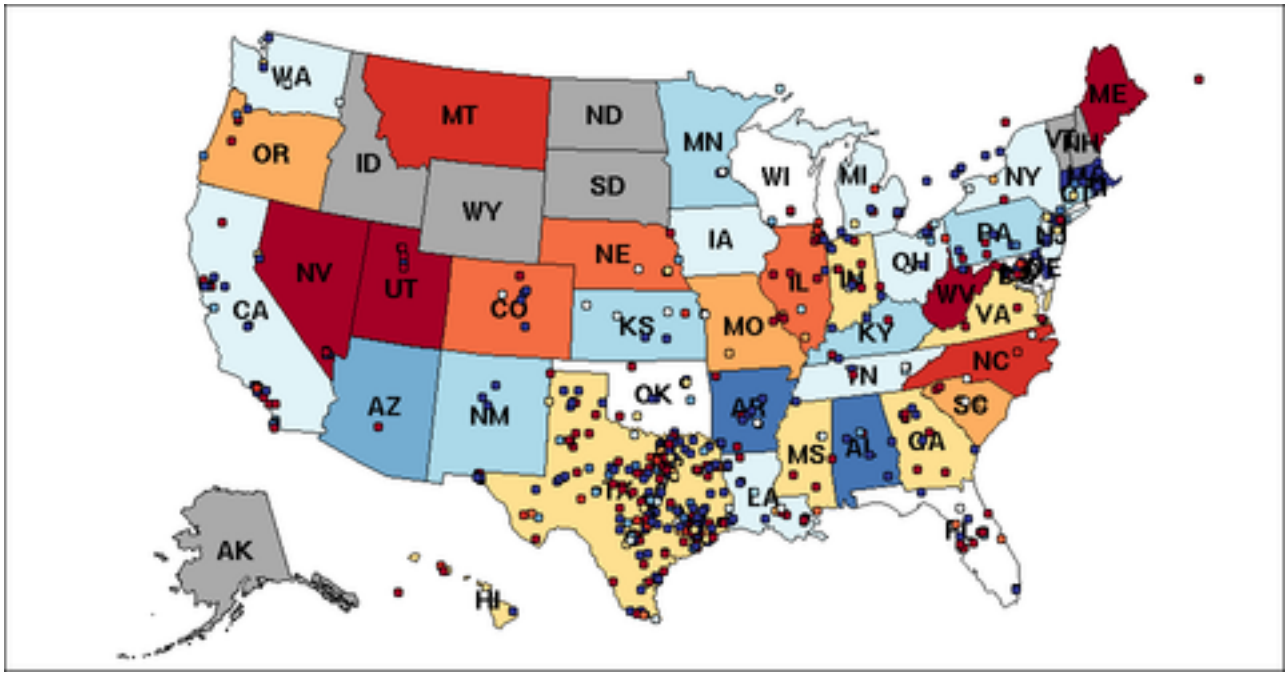
3 aaa hint result.png

4 b texas nearby.png

5 c texas sentiments.png



6 d texas map.png



7 README

```
roigreenberg

305571234

Roi Greenberg


==================================
=  README for ex7: Twitter trends =
==================================


usage:
python3 trends.py -p     -> print sentiment
python3 trends.py -t     -> run doctests
python3 trends.py -d     -> draw centered map
python3 trends.py -s     -> draw state sentiments
python3 trends.py -m     -> draw map for term
python3 trends.py -b     -> draw map by hour
python3 trends.py -c     -> containing state

==================
=  Description:  =
==================




tweet.py: implementation for the class Tweet.
              An instance of the type Tweet is initialized with the following parameters:
              - text: a string, the text of the tweet.
              - time: a datetime object, when the tweet was posted.
              - latitude: a floating-point number, the latitude of the tweet's location.
              - longitude: a floating-point number, the longitude of the tweet's location.
              - get_text() - Returns the text of the tweet.
              - get_time() - Returns a datetime object, when the tweet was posted.
              - get_location() - returns a Position. (Position is a class defined at the top of geo.py)
              - get_words() - Returns an ordered list of all words in the tweet. We define a "word"
                               as any consecutive substring of text that consists
                               ONLY of ASCII letters  converted to lowercase.
              - get_sentiment(word_sentiments) - Returns the sentiment of the tweet, that is,
                              the average sentiment of all the words in the tweet that have a sentiment.
                              Returns None if none of the words have a sentiment.

geo_tweet_tools.py:
    1. find_centroid: calculate the centeroid and area of polygon
        2. find_center:  calculate the center of several polygons
        3. find_closest_state: find the closest state to the tweet
    4. find_containing_state: find the state containing the tweet
        5. group_tweets_by_state: create a dictionary that group tweets by  their state as key

nation_mood.py: implement some function that
        1. most_talkative_state: find the most talkative state
        2. average_sentiments: create a dictionary from state to average of the tweets  sentiments
        3. group_tweets_by_hour: create a list of lists of hours so that every list contain the tweet
                           that sent in the same hour
============================
=  List of submitted files: =
============================
```

there should be another word at the end of each usage . see exercise description

```
60
61  README
62  tweet.py
63  geo_tweet_tools.py
64  nation_mood.py
```

8 geo tweet tools.py

```
1  ######################################################################
2  # FILE: geo_tweet_tools.py
3  # WRITER: Roi Greenberg + roigreenberg + 305571234
4  # EXERCISE : intro2cs ex7 2013-2014
5  # Description: implement some function about polygon and tweets:
6  #     1. find_centroid: calculate the centeroid and area of polygon
7  #     2. find_center:  calculate the center of several polygons
8  #     3. find_closest_state: find the closest state to the tweet
9  #     4. find_containing_state: find the state containing the tweet
10 #     5. group_tweets_by_state: create a dictionary that group
11 #        tweets by  their state as key
12 ######################################################################
13
14 from geo import us_states, Position
15 from tweet import Tweet
16
17 def find_centroid(polygon):
18     """Find the centroid of a polygon.
19
20     http://en.wikipedia.org/wiki/Centroid #Centroid_of_polygon
21
22     polygon -- A list of positions, in which the first and last are the same
23
24     Returns: 3 numbers; centroid latitude, centroid longitude, and polygon area
25
26     Hint: If a polygon has 0 area, return its first position as its centroid
27
28     >>> p1, p2, p3 = Position(1, 2), Position(3, 4), Position(5, 0)
29     >>> triangle = [p1, p2, p3, p1]  # First vertex is also the last vertex
30     >>> find_centroid(triangle)
31     (Position(3.0, 2.0), 6.0)
32     >>> find_centroid([p1, p3, p2, p1])
33     (Position(3.0, 2.0), 6.0)
34     >>> find_centroid([p1, p2, p1])
35     (Position(1.0, 2.0), 0)
36     """
37     # calculate the area
38     area=0.5 * (sum(polygon[i].latitude() * polygon[i+1].longitude()\
39               - polygon[i + 1].latitude() * polygon[i].longitude() \
40                  for i in range(len(polygon) - 1)))
41     # return the first position if area is 0
42     if not area:
43         return Position(polygon[0].latitude(), polygon[0].longitude()),0
44
45     # calclate the centroid latitude
46     latitude=(1 / (6 * area)) * sum((polygon[i].latitude() + \
47                                polygon[i + 1].latitude()) * \
48               (polygon[i].latitude() * polygon[i+1].longitude() - \
49                polygon[i + 1].latitude() * polygon[i].longitude())\
50                  for i in range(len(polygon)-1))
51     # calclate the centroid logitude
52     longitude=(1 / (6 * area)) * sum((polygon[i].longitude() + \
53                                 polygon[i + 1].longitude()) * \
54               (polygon[i].latitude() * polygon[i + 1].longitude() - \
55                polygon[i + 1].latitude() * polygon[i].longitude())\
56                  for i in range(len(polygon)-1))
57
58     return Position(latitude,longitude),abs(area)
59
```

6 is a magic number :-1

it is more efficient to copy the positions in to tuples than to access the position each time

backslash is redundent in parenthesis.

lines 47,53 indented incorrectly see http://pep8online.com /checkresult

```
60  def find_center(polygons):
61      """Compute the geographic center of a state, averaged over its polygons.
62
63      The center is the average position of centroids of the polygons in polygons,
64      weighted by the area of those polygons.
65
66      Arguments:
67      polygons -- a list of polygons
68
69      >>> ca = find_center(us_states['CA'])  # California
70      >>> round(ca.latitude(), 5)
71      37.25389
72      >>> round(ca.longitude(), 5)
73      -119.61439
74
75      >>> hi = find_center(us_states['HI'])  # Hawaii
76      >>> round(hi.latitude(), 5)
77      20.1489
78      >>> round(hi.longitude(), 5)
79      -156.21763
80      """
81
82      # calclate the latitude
83      latitude=sum(find_centroid(polygon)[0].latitude() * \
84                   find_centroid(polygon)[1] for polygon in polygons)\
85             /(sum(find_centroid(polygon)[1] for polygon in polygons))
86      # calclate the longitude
87      longitude=sum(find_centroid(polygon)[0].longitude() * \
88                    find_centroid(polygon)[1] for polygon in polygons) / \
89                    sum(find_centroid(polygon)[1] for polygon in polygons)
90
91      return Position(latitude,longitude)
92
93
94  def find_closest_state(state_centers):
95      import geo
96      """Returns a function that takes a tweet and returns the name of the state
97      closest to the given tweet's location.
98
99      Use the geo_distance function (already provided) to calculate distance
100     in miles between two latitude-longitude positions.
101
102     Arguments:
103     tweet -- a tweet abstract data type
104     state_centers -- a dictionary from state names to positions.
105
106     >>> state_centers = {n: find_center(s) for n, s in us_states.items()}
107     >>> sf = Tweet("Welcome to San Francisco", None, 38, -122)
108     >>> nj = Tweet("Welcome to New Jersey", None, 41.1, -74)
109     >>> find_state = find_closest_state(state_centers)
110     >>> find_state(sf)
111     'CA'
112     >>> find_state(nj)
113     'NJ'
114     """
115
116     def find_state(tweet):
117         """the function takes a tweet and returns the name of the state
118         closest to the given tweet's location.
119
120         Argumente:
121         tweet- a tweet abstract data type"""
122         # calculate the distances between the tweet position and the states
123         # centers and take the shortest distance
124         closest_state=min([(geo.geo_distance(tweet.get_location(),\
125                                 state_centers[state]),state)\
126                for state in state_centers])
127         # return the name of tShe state
```

lines 83,87 there should be spaces on bot sides of "=".

```
128             return closest_state[1]
129
130         return find_state
131
132
133
134     def find_containing_state(states):
135         """Returns a function that takes a tweet and returns the name of the state
136         containing the given tweet's location.
137
138         Use the geo_distance function (already provided) to calculate distance
139         in miles between two latitude-longitude positions.
140
141         Arguments:
142         tweet -- a tweet abstract data type
143         us_states -- a dictionary from state names to positions.
144
145         >>> sf = Tweet("Welcome to San Francisco", None, 38, -122)
146         >>> ny = Tweet("Welcome to New York", None, 41.1, -74)
147         >>> find_state = find_containing_state(us_states)
148         >>> find_state(sf)
149         'CA'
150         >>> find_state(ny)
151         'NY'
152         """
153         def find_state(tweet):
154             """the function takes a tweet and returns the name of the state
155             contain the given tweet's location.
156
157             Argumente:
158             tweet- a tweet abstract data type"""
159
160             # get latitude and longitude of the tweet
161             t_lat=tweet.get_location().latitude()
162             t_lon=tweet.get_location().longitude()
163
164             # run for every state
165             for state in states:
166                 # set inside to False
167                 inside=False
168                 # set polygon as the list of state positions
169                 polygon=states[state][0]
170                 # number of positions in state
171                 num=len(polygon)
172                 # take the first position
173                 lat_1,lon_1=polygon[0].latitude(),polygon[0].longitude()
174                 # run for every position
175                 for pol in range(num):
176                     # set the next position
177                     lat_2=polygon[pol].latitude()
178                     lon_2=polygon[pol].longitude()
179                     # check if the tweet position cross the line between
180                     # the positions we check
181                     if min(lon_1,lon_2) < t_lon <= max(lon_1,lon_2):
182
183                         if t_lat <= max(lat_1,lat_2):
184                             if lon_1 != lon_2:
185                                 lat_in_pol = (t_lon-lon_1) * (lat_2-lat_1) / \
186                                         (lon_2-lon_1) + lat_1
187                             # if cross, change the state of inside
188                             if lat_1 == lat_2 or t_lat <= lat_in_pol:
189                                 inside = not inside
190                     # set the first position as the next one
191                     lat_1,lon_1 = lat_2,lon_2
192                 # if tweet inside state return the state name
193                 if inside == True:
194                     return state
195
```

lines 167,169,171,173,177,178: "=" should have spaces on the sides:-0.5 . see http://pep8online.com/

```
196     return find_state
197
198 def group_tweets_by_state(tweets,find_state):
199     """Return a dictionary that aggregates tweets by their nearest state center.
200
201     The keys of the returned dictionary are state names, and the values are
202     lists of tweets that appear closer to that state center than any other.
203
204     tweets -- a sequence of tweet abstract data types
205
206     >>> state_centers = {n: find_center(s) for n, s in us_states.items()}
207     >>> find_state = find_closest_state(state_centers);
208     >>> sf = Tweet("Welcome to San Francisco", None, 38, -122)
209     >>> ny = Tweet("Welcome to New York", None, 41, -74)
210     >>> ca_tweets = group_tweets_by_state([sf, ny],find_state)['CA']
211     >>> ca_tweets
212     [Tweet('Welcome to San Francisco', None, 38, -122)]
213     """
214     # create an empty dictionary
215     tweets_by_state={}
216     # run for every tweet
217     for tweet in tweets:
218         # recieve the tweet state
219         state=find_state(tweet)
220         # add new state to dictionary if not exist
221         if state not in tweets_by_state.keys():
222             # set the value as the tweet
223             tweets_by_state[state]=[tweet]
224         # add the tweet to the state key if alredy exist
225         else:
226             tweets_by_state[state].append(tweet)
227
228     return tweets_by_state
```

9 nation mood.py

```
1  ###############################################################
2  # FILE: nation_mood.py
3  # WRITER: Roi Greenberg + roigreenberg + 305571234
4  # EXERCISE : intro2cs ex7 2013-2014
5  # Description: implement some function that
6  # 1. most_talkative_state: find the most talkative state
7  # 2. average_sentiments: create a dictionary from state to
8  #    average of the tweets  sentiments
9  # 3. group_tweets_by_hour: create a list of lists of hours
10 #    so that every list contain the tweet that sent in the
11 #    same hour
12 ###############################################################
13
14 from data import load_tweets
15 from geo_tweet_tools import group_tweets_by_state,find_closest_state,find_center
16 from geo import us_states, Position
17 from tweet import Tweet
18
19 def most_talkative_state(tweets,find_state):
20     """Return the state that has the largest number of tweets containing term.
21     >>> state_centers = {n: find_center(s) for n, s in us_states.items()}
22     >>> tweets = load_tweets('texas')
23     >>> find_state = find_closest_state(state_centers);
24     >>> most_talkative_state(tweets,find_state)
25     'TX'
26     >>> tweets = load_tweets('sandwich')
27     >>> most_talkative_state(tweets,find_state)
28     'NJ'
29     """
30     # create a dictionary of states to tweets
31     tweets_by_state=group_tweets_by_state(tweets,find_state)
32
33     # if no tweets with sentiment value
34     if not tweets_by_state:
35         return None
36     # create an empty list
37     states=[]
38     # run for every tweet
39     for tweet_with_state in tweets_by_state.items():
40         # ammount of tweets in the state
41         ammount_tweets=len(tweet_with_state[1])
42         # the state code name
43         state_name=tweet_with_state[0]
44         # add the ammount and name as a tuple to the list
45         states.append((ammount_tweets, state_name))
46     # take the name of the state with the highest ammount of tweets
47     state=max(states)[1]
48
49     return state
50
51 def average_sentiments(tweets_by_state,word_sentiments):
52     """Calculate the average sentiment of the states by averaging over all
53     the tweets from each state. Return the result as a dictionary from state
54     names to average sentiment values (numbers).
55
56     If a state has no tweets with sentiment values, leave it out of the
57     dictionary entirely.  Do NOT include states with no tweets, or with tweets
58     that have no sentiment, as 0.  0 represents neutral sentiment, not unknown
59     sentiment.
```

lines :37,41,43,47 : there should be spaces around "=". see http://pep8online.com /:-0.5

```
60
61      tweets_by_state -- A dictionary from state names to lists of tweets
62      """
63      # create an empty dictionary
64      state_average={}
65
66      # run for every state
67      for state in tweets_by_state.items():
68          # set the variables
69          total_sentiment=0
70          count=0
71          tweet_sentiment=None
72
73          # the state code name
74          state_name=state[0]
75          # the tweets list
76          tweets=state[1]
77
78          # run for every tweet
79          for tweet in tweets:
80              # recieve the sentiment of the tweet
81              tweet_sentiment=tweet.get_sentiment(word_sentiments)
82              # add the tweet sentiment to the total sentiment if
83              # it not None and raise the counter
84              if tweet_sentiment!=None:
85                  count +=1
86                  total_sentiment+=tweet_sentiment
87          # add the state to the dictionary if any tweet had sentiment value
88          if count != 0:
89              state_average[state_name]=total_sentiment/count
90      return state_average
91
92
93
94  def group_tweets_by_hour(tweets):
95      """Return a list of lists of tweets that are gouped by the hour
96      they were posted.
97
98      The indexes of the returned list represent the hour when they were posted
99      - the integers 0 through 23.
100
101     tweets_by_hour[i] is the list of all
102     tweets that were posted between hour i and hour i + 1. Hour 0 refers to
103     midnight, while hour 23 refers to 11:00PM.
104
105     To get started, read the Python Library documentation for datetime
106     objects:
107     http://docs.python.org/py3k/library/datetime.html#datetime.datetime
108
109     tweets -- A list of tweets to be grouped
110     """
111     # create list of 24 empty lists
112     HOURS=24
113     hours_list=[[] for hour in range(HOURS)]
114
115     # run for every tweet
116     for tweet in tweets:
117         # put the tweet in the list of hours in the
118         hours_list[tweet.get_time().hour].append(tweet)
119     return hours_list
```

lines :64,74,76,81,84,81,85 ,86 : there should be spaces around "!=","+=","=". see http://pep8online.com /:-0.5

if hours is a constant t should be at the beginning of the file

10 tweet.py

```
1  ###############################################################
2  # FILE: tweet.py
3  # WRITER: Roi Greenberg + roigreenberg + 305571234
4  # EXERCISE : intro2cs ex7 2013-2014
5  # Description: implement a new class of Tweets.
6  #              tweet contain text, time and position
7  # coordinate(latitude and longitude)
8  # the method of the tweet are get.word - return a list
9  # containing only the words converting to lower-case char
10 # get_location - return the position of the tweet
11 # get_sentiment - return the sentiment value of the tweet
12 ###############################################################
13
14 from doctest import run_docstring_examples
15 from geo import Position
16
17 class Tweet:
18     def __init__(self, text, time, lat, lon):
19         self.__text=text
20         self.__time=time
21         self.__lat=lat
22         self.__lon=lon
23
24     def get_words(self):
25         """Return the words in a tweet, not including punctuation.
26         """
27         import re
28         # create a list containing only the words converting to lower-case char
29         word_list=re.sub("[^a-zA-Z]", " ", self.__text.lower()).split()
30         return word_list
31
32     def get_text(self):
33         """Return the text of the tweet."""
34         return self.__text
35
36     def get_time(self):
37         """Return the datetime that represents when the tweet was posted."""
38         return self.__time
39
40     def get_location(self):
41         """Return a position (see geo.py) that represents the tweet's location."""
42         return Position(self.__lat,self.__lon)
43
44     def __eq__(self, other):
45         if isinstance(other, self.__class__):
46             return (self.get_text() == other.get_text() and
47                     self.get_location() == other.get_location() and
48                     self.get_time() == other.get_time())
49         else:
50             return False
51
52     def __str__(self):
53         """Return a string representing the tweet."""
54         return '"{0}" @ {1} : {2}'.format(self.get_text(),
55                                           self.get_location(),
56                                           self.get_time())
57
58     def __repr__(self):
59         """Return a string representing the tweet."""
```

no documentation for __init__:-1

```
60          return 'Tweet({0}, {1}, {2}, {3})'\
61                 .format(*map(repr,(self.get_text(),
62                                    self.get_time(),
63                                    self.get_location().latitude(),
64                                    self.get_location().longitude())))
65
66      def get_sentiment(self,word_sentiments):
67          """ Return a sentiment representing the degree of positive or negative
68          sentiment in the given tweet, averaging over all the words in the tweet
69          that have a sentiment value.
70          """
71          # get only the words from the tweet
72          words=self.get_words()
73          # set the variables
74          total_sentiment=0.
75          count=0
76          # run for every word
77          for word in words:
78              # if word had sentiment value raise the counter and add the
79              # the sentiment value to total_sentimet
80              if word in word_sentiments.keys():
81                  count += 1
82                  total_sentiment += word_sentiments[word]
83
84          return None if count == 0 else total_sentiment/count
```

lines :72,74,75: there should be spaces around "=". see http://pep8online.com/:-0.5