In [41]:
```
import gspread
from oauth2client.service_account import ServiceAccountCredentials
from collections import OrderedDict
import pprint
```

In [42]:
```
# functions used for adding points when attribute can choose multiple options
# the "littles" parameter refers to all the answers chosen by the specific little
# the "bigs" parameter refers to all the answers chosen by the specific big
# the "point" parameter refers to the point we should allocate if there's a match in the l
def matching_list(littles, bigs, point):
    points = 0
    for little in littles:
        if little != "No Preference":
            for big in bigs:
                if little == big:
                    points += point
    return points
```

In [43]:
```
# functions used for adding points when attribute can choose only single option
# the "littles" parameter refers to the specific answer chosen by the specific little
# the "bigs" parameter refers to the specific answers chosen by the specific big
# the "point" parameter refers to the point we should allocate if there's a match in the l
def matching(littles, bigs, point):
    points = 0
    if littles == bigs:
        points += point
    return points
```

In [44]:
```
# setting up the authorization so we can edit the google spreadsheet using some python lib
scope = ['https://spreadsheets.google.com/feeds' + ' ' +'https://www.googleapis.com/auth/d
creds = ServiceAccountCredentials.from_json_keyfile_name('client_secret.json', scope)
client = gspread.authorize(creds)
```

In [45]:
```
# opens the sheets
# make sure you create the "matching" sheet as well
# make sure all of these spreadsheets have the client_email from client_secret.json shared
little_sheet = client.open('little_sheet').sheet1
big_sheet = client.open('big_sheet').sheet1
matches = client.open('matches')

# importing all the values from the spreadsheet into the program!
littles = little_sheet.get_all_values()
bigs = big_sheet.get_all_values()
```

In [46]:
```
# map to store all bigs who didn't get matched with anyone
bigs_left = []
```

In [47]:
```
# add all bigs to bigs-left set since we haven't matched any bigs yet at the start
# note that each "big" in bigs list contains all the forms information of that partiuclar
# the big[1] allows us to store the email of the big. You may need to use a different inde
# "1" to get the email if the format of the spreadsheet changes
for big in bigs[1:len(bigs)]:
    bigs_left.append(big[1])
```

In [48]:

```
# make sure to check you selected an email of a big in the bigs_left list
# I've commented this out for privacy reasons if I upload this to a github // shared it pu
# but uncomment below and check it yourself!
# bigs[1][1]
```

In [49]:
```
# make sure to check this to ensure you've got all the bigs who signed up covered!!
len(bigs_left)
```

Out[49]:
```
24
```

In [50]:
```
# Setting up the data structures to map the littles to bigs
d = {}

# Using lists so we can take advantage of indexing to make it easier to compare class star
class_standings = ["Freshman", "Sophomore", "Junior", "Senior", "BS/MS"]
mapping = OrderedDict(sorted(d.items(), key=lambda t: t[0]))
```

In [51]:
```
# we are creating an object to store the worksheet that we made earlier
# matches refers to the "matches" spreadsheet that will store the bigs and littles maching
worksheet = matches.get_worksheet(0)
```

In [62]:
```
# this map allows us to map each big's email to the number of littles they want
# this is necessary for us in order to make sure each big gets the number of littles they
bigs_num_littles_wanted = dict()

for big in bigs[1:len(bigs)]:
    bigs_num_littles_wanted[big[1]] = big[5]
```

In [53]:
```
# Here, we basically score each little against all the bigs and rank all the bigs based or
# When you run this ipython cell, you will get all the ranking resuslts on the matching sp
# Note that the very left-most column represents the littles and all the number and email
# the Bigs that were ranked against the little
row = 1

# matching each little against all the bigs
cell_list = []
for little in littles[1:len(littles)]:

    # usualy email to uniquely identify each little
    key = little[1]
    mapping[key] = []

    # IMPORTANT* Make SURE to check all these indexing again to see if it matches up with
    # spreadsheet
    little_class = little[9]
    little_preferred_interaction = little[10]
    little_event_attendance = little[11]
    little_hours = little[12]
    little_classes_taken  = little[13].split(', ')

    little_background = little[15].split(", ")
    little_gender = little[16].split(', ')
    little_lgbtq = little[17].split(', ')
    little_race = little[18].split(', ')
    little_experience = little[19].split(', ')
    little_interest = little[20].split(', ')

    little_q1 = little[21]
    little_q2 = little[22]
```

```
    little_q3 = little[23]
    little_q4 = little[24]
    little_q5 = little[25]
    little_q6 = little[26]
    little_q7 = little[27]
    little_q8 = little[28]
    little_q9 = little[29]
    little_q10 = little[30]

    # make sure to increment this so that we are parsing through each little
    row += 1
    col = 1
    big_list = []

    # this allows us to compare some little to all the bigs
    for big in bigs[1:len(bigs)]:
        # comparing class standing
        points = 0

        # check this again to make sure
        big_class = big[9]
        big_classes_taken = big[13].split(', ')

        # Ensuring Littles are always a class year below
        if ((class_standings.index(big_class) - class_standings.index(little_class)) > 0):
            # Ensuring Bigs always took more classes than the Little
            if len(big_classes_taken) > len(little_classes_taken):

                big_preferred_interaction = big[10]
                big_event_attendance = big[11]
                big_hours = big[12]
                big_background = big[15].split(", ")

                big_gender = big[16].split(', ')
                big_lgbtq = big[17].split(', ')
                big_race = big[18].split(', ')
                big_experience = big[19].split(', ')
                big_interest = big[20].split(', ')

                big_q1 = big[21]
                big_q2 = big[22]
                big_q3 = big[23]
                big_q4 = big[24]
                big_q5 = big[25]
                big_q6 = big[26]
                big_q7 = big[27]
                big_q8 = big[28]
                big_q9 = big[29]
                big_q10 = big[30]


                # this part is important as this determines how much point we give for eac
                points += matching_list(big_background, little_background, 7)
                points += matching(big_gender, little_gender, 7)
                points += matching(big_lgbtq, little_lgbtq, 7)
                points += matching(big_race, little_race, 7)
                points += matching(big_hours, little_hours, 2.5)
                points += matching(big_experience, little_experience, 2)
                points += matching(big_event_attendance, little_event_attendance, 1.5)
                points += matching(big_preferred_interaction, little_preferred_interactior
                points += matching(big_interest, little_interest, 1)

                # just 0.5 points since these are side questions
                points += matching(big_q1, little_q1, 0.5)
                points += matching(big_q2, little_q2, 0.5)
                points += matching(big_q3, little_q3, 0.5)
```

```
                    points += matching(big_q4, little_q4, 0.5)
                    points += matching(big_q5, little_q5, 0.5)
                    points += matching(big_q6, little_q6, 0.5)
                    points += matching(big_q7, little_q7, 0.5)
                    points += matching(big_q8, little_q8, 0.5)
                    points += matching(big_q9, little_q9, 0.5)
                    points += matching(big_q10, little_q10, 0.5)

                    # add big to map
                    big_list.append([points, big[1]])

        # sorts bigs based on the number of points they have
        sorted_list = sorted(big_list, key=lambda x: x[0], reverse=True)
        for big in sorted_list:
            mapping[key].append(big[0]) # points
            mapping[key].append(big[1]) # cse email
        values = mapping[key]
        values = [key] + values

        # google sheets goes from A-Z
        if (len(values) > 26):
            values = values[0:25]

        # discard big from bigs left
        for big in values:
            if big in bigs_left:
                bigs_left.remove(big)

        end_col = chr(ord('A') + len(values) - 1)
        col_range = 'A' + str(row) + ':' + end_col + str(row)
        if (len(values) == 0):
            col_range = 'A' + str(row)
        curr_val = {'range': col_range, 'values': [values]}
        cell_list.append(curr_val)

# adding to spreadsheet
worksheet.batch_update(cell_list)

# prints the bigs who didn't get matched to any little
# may need to manually match these people or pool the littles andd bigs again and run the
print(bigs_left)
```

```
['gelo@cs.washington.edu']
```

Based off the match ranking, the code below actually pairs each big to little

In [56]:

```
# opens the sheets
# make sure you create the "matching" sheet as well
# make sure all of these documents have the client_email shared with them as well
match_sheet = client.open('matches').get_worksheet(0)
matches = match_sheet.get_all_values()

pairing = client.open('Big / Little Pairing')
```

In [58]:

```
# creating the sheet object so we can update the spreadsheet later
worksheet = pairing.get_worksheet(0)
```

In [67]:

```
# map to store all bigs who didn't get matched with anyone
bigs_left = []
```

```
# resetting the bigs-left set
for big in bigs[1:len(bigs)]:
    bigs_left.append(big[1])

len(bigs_left)
```

Out[67]:
```
24
```

In [71]:
```
# this maps each big's email (key) to the list of little's email (value)
big_map = {}
little_not_paired = set()
# 1 indicates little was paired, 0 indicates little was not paired
flag_paired = 0
count = 0
row = 1

# based on the matches spreadsheet, we go through each row, matching each little with the
# big in the row. If the big is already up to capacity (i.e. wanted 3 littles and got matc
# to the next best ranked big
for i in range(1, len(matches)):
    match = matches[i]
    little_key = match[0]

    # going through each ranked big
    for j in range(2, len(match), 2):
        big_key = match[j]

        # checking if the big is in the dictionary
        # if so, check if they have any room to add littles
        # If they do have room for more littles, add the little to the big
        if big_key in big_map:
            mapped_littles = big_map[big_key]

            # remember that bigs_num_littles_wanted is a map that maps the email of the bi
            # to the number of littles they want!
            #if len(mapped_littles) < int(bigs_num_littles_wanted[big_key]):

            # update for Spring 2022: since there nearly a 1:1 distribution between Little
            # I (Hayoung) decided to let every Big have exactly 1 Little
            if len(mapped_littles) < 1:
                big_map[big_key].append(little_key)
                flag_paired = 1
                break
        else:    # add big to the dictionary along with the little
            big_map[big_key] = [little_key]
            if big_key in bigs_left:
                bigs_left.remove(big_key)
            flag_paired = 1
            print(count)
            count += 1
            break


    if flag_paired == 0:
        little_not_paired.add(little_key)
    else:
        flag_paired = 0
```

```
0
1
2
3
4
```

In [72]:
```
little_not_paired
```

Out[72]:
```
set()
```

In [75]:
```
#bigs_left
```

In [76]:
```
cell_list = []
row = 1
for key, value in big_map.items():
    values = big_map[key]
    values = [key] + values

    # google sheets goes from A-Z
    if (len(values) > 26):
        values = values[0:25]


    end_col = chr(ord('A') + len(values) - 1)
    col_range = 'A' + str(row) + ':' + end_col + str(row)
    row += 1
    if (len(values) == 0):
        col_range = 'A' + str(row)
    curr_val = {'range': col_range, 'values': [values]}
    cell_list.append(curr_val)

# adding to spreadsheet
worksheet.batch_update(cell_list)
```

Out[76]:

```
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'totalUpdatedRows': 21,
 'totalUpdatedColumns': 2,
 'totalUpdatedCells': 42,
 'totalUpdatedSheets': 1,
 'responses': [{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
   'updatedRange': 'Sheet1!A1:B1',
   'updatedRows': 1,
   'updatedColumns': 2,
   'updatedCells': 2},
  {'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
   'updatedRange': 'Sheet1!A2:B2',
   'updatedRows': 1,
   'updatedColumns': 2,
   'updatedCells': 2},
  {'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
   'updatedRange': 'Sheet1!A3:B3',
```

```
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A4:B4',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A5:B5',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A6:B6',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A7:B7',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A8:B8',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A9:B9',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A10:B10',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A11:B11',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A12:B12',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A13:B13',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A14:B14',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A15:B15',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A16:B16',
 'updatedRows': 1,
```

```
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A17:B17',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A18:B18',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A19:B19',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A20:B20',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2},
{'spreadsheetId': '1IvgG1h9JZogSwbcx1sJyfOIiRpCztl7Ohef5J1JxANc',
 'updatedRange': 'Sheet1!A21:B21',
 'updatedRows': 1,
 'updatedColumns': 2,
 'updatedCells': 2}]}
```


In [ ]: