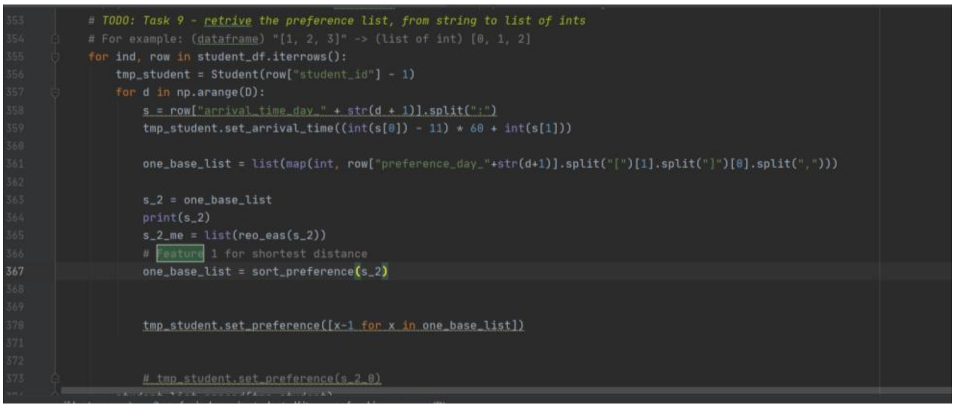
Name: Chan Kai Yin

SID: 1155124983

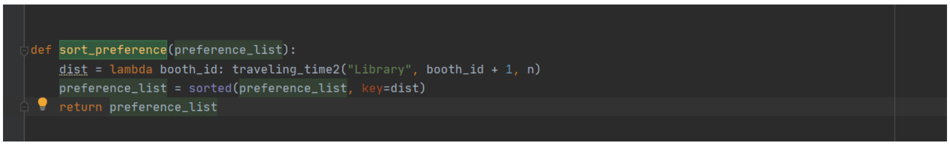
Selected task: 1, 2, 5

Task 1:

System1:



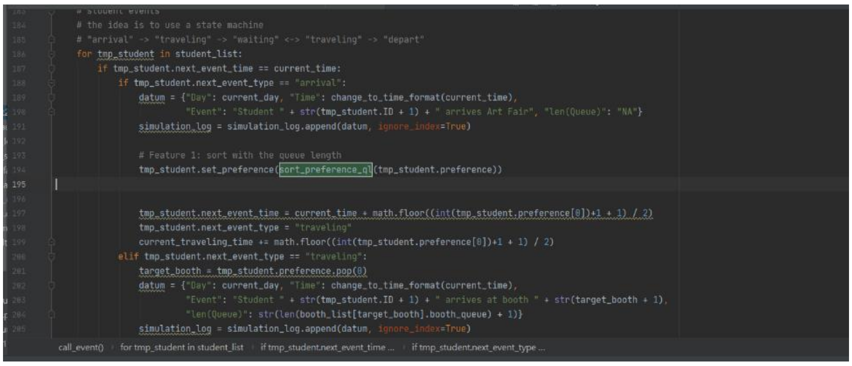
```
# TODO: Task 9 - retrive the preference list, from string to list of ints
# For example: (dataframe) "[1, 2, 3]" -> (list of int) [0, 1, 2]
for ind, row in student_df.iterrows():
    tmp_student = Student(row["student_id"] - 1)
    for d in np.arange(D):
        s = row["arrival_time_day_" + str(d + 1)].split(":")
        tmp_student.set_arrival_time((int(s[0]) - 11) * 60 + int(s[1]))

        one_base_list = list(map(int, row["preference_day_"+str(d+1)].split("[")[1].split("]")[0].split(",")))

        s_2 = one_base_list
        print(s_2)
        s_2_me = list(reo_eas(s_2))
        # Feature 1 for shortest distance
        one_base_list = sort_preference(s_2)


        tmp_student.set_preference([x-1 for x in one_base_list])


        # tmp_student.set_preference(s_2_0)
```



```
def sort_preference(preference_list):
    dist = lambda booth_id: traveling_time2("Library", booth_id + 1, n)
    preference_list = sorted(preference_list, key=dist)
    return preference_list
```

System1 sorts the preference list of student in the beginning based on the distance of the booth and library.

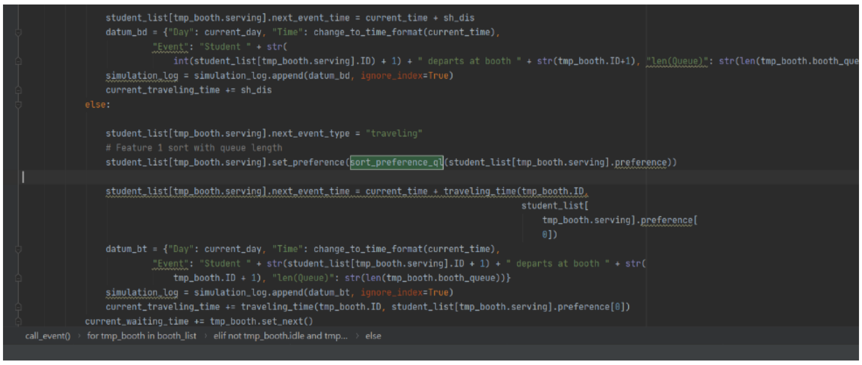
System2:



```
# student events
# the idea is to use a state machine
# "arrival" -> "traveling" -> "waiting" <-> "traveling" -> "depart"
for tmp_student in student_list:
    if tmp_student.next_event_time == current_time:
        if tmp_student.next_event_type == "arrival":
            datum = {"Day": current_day, "Time": change_to_time_format(current_time),
                     "Event": "Student " + str(tmp_student.ID + 1) + " arrives Art Fair", "len(Queue)": "NA"}
            simulation_log = simulation_log.append(datum, ignore_index=True)

            # Feature 1: sort with the queue length
            tmp_student.set_preference(sort_preference_ql(tmp_student.preference))


            tmp_student.next_event_time = current_time + math.floor((int(tmp_student.preference[0])+1 + 1) / 2)
            tmp_student.next_event_type = "traveling"
            current_traveling_time += math.floor((int(tmp_student.preference[0])+1 + 1) / 2)
        elif tmp_student.next_event_type == "traveling":
            target_booth = tmp_student.preference.pop(0)
            datum = {"Day": current_day, "Time": change_to_time_format(current_time),
                     "Event": "Student " + str(tmp_student.ID + 1) + " arrives at booth " + str(target_booth + 1),
                     "len(Queue)": str(len(booth_list[target_booth].booth_queue) + 1)}
            simulation_log = simulation_log.append(datum, ignore_index=True)
```

call_event()
for tmp_student in student_list
if tmp_student.next_event_time ...
if tmp_student.next_event_type ...

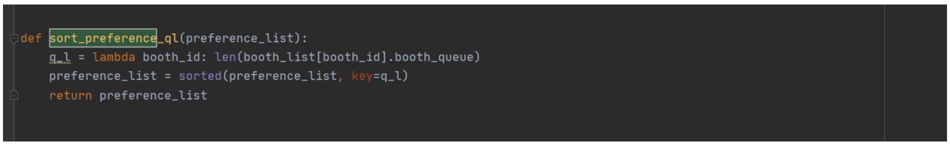
On arrival



```
            student_list[tmp_booth.serving].next_event_time = current_time + sh_dis
            datum_bd = {"Day": current_day, "Time": change_to_time_format(current_time),
                        "Event": "Student " + str(
                            int(student_list[tmp_booth.serving].ID) + 1) + " departs at booth " + str(tmp_booth.ID+1), "len(Queue)": str(len(tmp_booth.booth_que
            simulation_log = simulation_log.append(datum_bd, ignore_index=True)
            current_traveling_time += sh_dis
        else:

            student_list[tmp_booth.serving].next_event_type = "traveling"
            # Feature 1 sort with queue length
            student_list[tmp_booth.serving].set_preference(sort_preference_ql(student_list[tmp_booth.serving].preference))

            student_list[tmp_booth.serving].next_event_time = current_time + traveling_time(tmp_booth.ID,
                                                                                            student_list[
                                                                                                tmp_booth.serving].preference[
                                                                                                0])
            datum_bt = {"Day": current_day, "Time": change_to_time_format(current_time),
                        "Event": "Student " + str(student_list[tmp_booth.serving].ID + 1) + " departs at booth " + str(
                            tmp_booth.ID + 1), "len(Queue)": str(len(tmp_booth.booth_queue))}
            simulation_log = simulation_log.append(datum_bt, ignore_index=True)
            current_traveling_time += traveling_time(tmp_booth.ID, student_list[tmp_booth.serving].preference[0])
        current_waiting_time += tmp_booth.set_next()
```

call_event() › for tmp_booth in booth_list › elif not tmp_booth.idle and tmp... › else

On traveling



```
def sort_preference_ql(preference_list):
    q_l = lambda booth_id: len(booth_list[booth_id].booth_queue)
    preference_list = sorted(preference_list, key=q_l)
    return preference_list
```

The system 2 sort the preference list of student based on the queue length in that time. Stduent will decide to go the shortest queue length first when they just arrive to the fair and departure from booth.

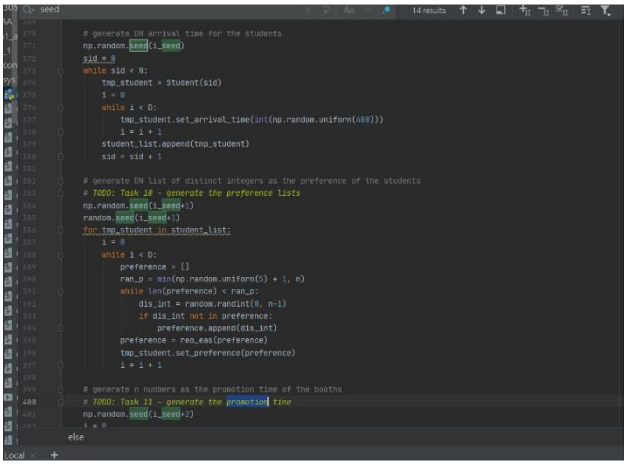
The system 2 don’t consider the distance. For example, if a student departure from booth 3 and realize that booth 4 and booth 18 are both empty, he will radomly choose the booth.

Two-stage sampling:

With the same setting: 2 days, 20 booth and 100 student. I use a bash file to run the program. Each time will inputing differrent seed to create differetn simulation and output the data to a text file as record. In sys1 and sys2 file are used in making data for comparison.

First stage: I run the program for 20 times with differernt seed and record the total waiting time for 2 days. It makes 20 different dataset including stduent and both promotion time. I am using d = 5, P = 0.95. From the h_1 value table, the value of h_1 = 2.453.

System 1 - First stage:



```
        # generate DN arrival time for the students
        np.random.seed(i_seed)
        sid = 0
        while sid < N:
            tmp_student = Student(sid)
            i = 0
            while i < D:
                tmp_student.set_arrival_time(int(np.random.uniform(400)))
                i = i + 1
            student_list.append(tmp_student)
            sid = sid + 1

        # generate DN list of distinct integers as the preference of the students
        # TODO: Task 10 - generate the preference lists
        np.random.seed(i_seed+1)
        random.seed(i_seed+1)
        for tmp_student in student_list:
            i = 0
            while i < D:
                preference = []
                ran_p = min(np.random.uniform(5) + 1, n)
                while len(preference) < ran_p:
                    dis_int = random.randint(0, n-1)
                    if dis_int not in preference:
                        preference.append(dis_int)
                preference = reo_eas(preference)
                tmp_student.set_preference(preference)
                i = i + 1

        # generate n numbers as the promotion time of the booths
        # TODO: Task 11 - generate the promotion time
        np.random.seed(i_seed+2)
        i = 0
```

The output data of total waiting time for 2 days:

1151	2065	894	1404	1108	2061	1593	1494	1453	1488	1140	1546	1366	768	1508	1490	622	1697	1046	1154

Mean = 1352.4

S.D = 375.735

For the second-stage:

total sample size N needed for system 1:

N_1 = max(21, (h_1^2 * 375.735)/25)

= 91

71 replications needed in the second stage for the system 1

The first 23 output in the second stage:

1090	810	1252	1254	402	1283	822	1270	1803	1819	722	1126	462	1128	1643	1900	582	389	1102	1088	1040	2123	750

Mean = 1354.414286

S.D = 491.9705863

W11 = 0.252

W12 = 0.748

For System 2:

First stage output:

639	1334	655	1083	933	1018	1040	1093	1070	998	852	924	915	494	939	932	436	1146	859	805

Mean = 908.25

S.D = 219.568

total sample size N needed for system 1:

N = max(21, (h_1^2 * 219.568)/25)

= 53

33 replications needed in the second stage for the system 2

The first 8 Output:

805	1350	1218	710	1061	1379	1255	1073

Mean = 975.969697

S.D = 335.4852349

W21 = 0.40

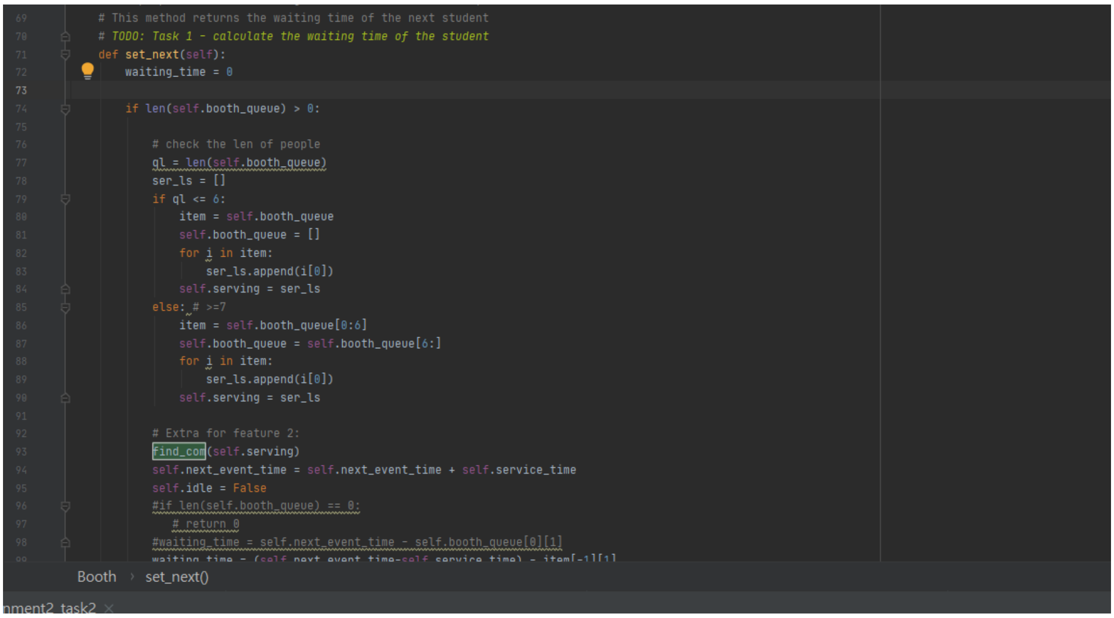
W22 = 0.60

I	$X^1(20)$	$S_i^2(20)$	N_i	$X^2(N_i - 20)$	W_{i1}	W_{i2}	$X_i(N_i)$
1	1352.40	375.74	90	1354.41	0.25	0.75	1353.91
2	908.25	219.57	53	975.96	0.40	0.60	948.876

From the above table. Using 2-stage sampling at 5% significance level, we can conclude that the system 2 has a shorter tatal waiting time compare to system 1.

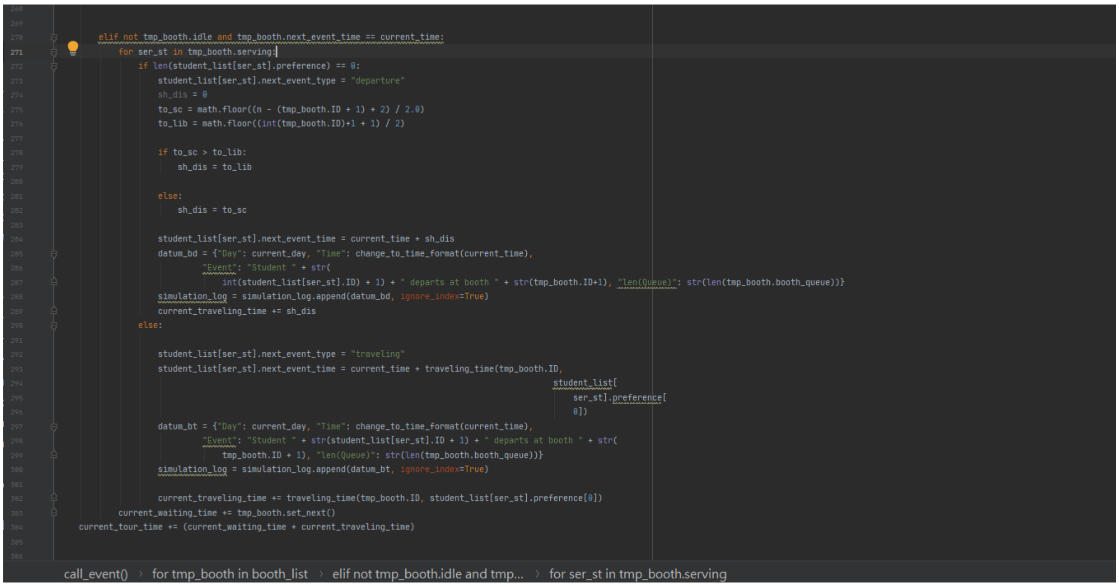
Task 2:

Group promotion:



```
69      # This method returns the waiting time of the next student
70      # TODO: Task 1 - calculate the waiting time of the student
71      def set_next(self):
72          waiting_time = 0
73
74          if len(self.booth_queue) > 0:
75
76              # check the len of people
77              ql = len(self.booth_queue)
78              ser_ls = []
79              if ql <= 6:
80                  item = self.booth_queue
81                  self.booth_queue = []
82                  for i in item:
83                      ser_ls.append(i[0])
84                  self.serving = ser_ls
85              else: # >=7
86                  item = self.booth_queue[0:6]
87                  self.booth_queue = self.booth_queue[6:]
88                  for i in item:
89                      ser_ls.append(i[0])
90                  self.serving = ser_ls
91
92              # Extra for feature 2:
93              find_com(self.serving)
94              self.next_event_time = self.next_event_time + self.service_time
95              self.idle = False
96              #if len(self.booth_queue) == 0:
97                 # return 0
98              #waiting_time = self.next_event_time - self.booth_queue[0][1]
99              waiting_time = (self.next_event_time-self.service_time) - item[-1][1]
```

By modify the set_next function from line 76 to 91, it takes up to 6 student for group promotion.



```
268
269
270         elif not tmp_booth.idle and tmp_booth.next_event_time == current_time:
271             for ser_st in tmp_booth.serving:
272                 if len(student_list[ser_st].preference) == 0:
273                     student_list[ser_st].next_event_type = "departure"
274                     sh_dis = 0
275                     to_sc = math.floor((n - (tmp_booth.ID + 1) + 2) / 2.0)
276                     to_lib = math.floor((int(tmp_booth.ID)+1 + 1) / 2)
277
278                     if to_sc > to_lib:
279                         sh_dis = to_lib
280
281                     else:
282                         sh_dis = to_sc
283
284                     student_list[ser_st].next_event_time = current_time + sh_dis
285                     datum_bd = {"Day": current_day, "Time": change_to_time_format(current_time),
286                                 "Event": "Student " + str(
287                                     int(student_list[ser_st].ID) + 1) + " departs at booth " + str(tmp_booth.ID+1), "len(Queue)": str(len(tmp_booth.booth_queue))}
288                     simulation_log = simulation_log.append(datum_bd, ignore_index=True)
289                     current_traveling_time += sh_dis
290                 else:
291
292                     student_list[ser_st].next_event_type = "traveling"
293                     student_list[ser_st].next_event_time = current_time + traveling_time(tmp_booth.ID,
294                                                                                          student_list[
295                                                                                              ser_st].preference[
296                                                                                              0])
297                     datum_bt = {"Day": current_day, "Time": change_to_time_format(current_time),
298                                 "Event": "Student " + str(student_list[ser_st].ID + 1) + " departs at booth " + str(
299                                     tmp_booth.ID + 1), "len(Queue)": str(len(tmp_booth.booth_queue))}
300                     simulation_log = simulation_log.append(datum_bt, ignore_index=True)
301
302                     current_traveling_time += traveling_time(tmp_booth.ID, student_list[ser_st].preference[0])
303             current_waiting_time += tmp_booth.set_next()
304         current_tour_time += (current_waiting_time + current_traveling_time)
305
306
```

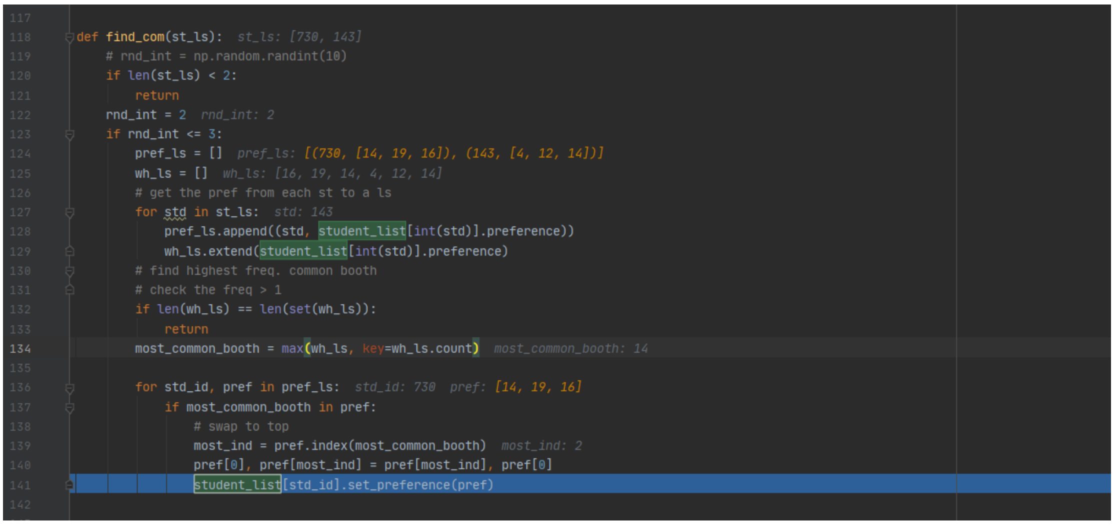
call_event() › for tmp_booth in booth_list › elif not tmp_booth.idle and tmp... › for ser_st in tmp_booth.serving

In booth event also need to iterate the serving list of the booth (line 271) for perform operations for each student in group promotion.

2	13:53	Student 733 departs at booth 13	0
2	13:55	Student 145 arrives at booth 13	1
2	13:56	Student 63 arrives at booth 13	1
2	13:56	Student 120 arrives at booth 13	2
2	13:56	Student 401 arrives at booth 13	3
2	13:56	Student 576 arrives at booth 13	4
2	13:56	Student 145 departs at booth 13	4
2	13:57	Student 63 departs at booth 13	0
2	13:57	Student 120 departs at booth 13	0
2	13:57	Student 401 departs at booth 13	0
2	13:57	Student 576 departs at booth 13	0

For illustration, the program set 1000 student and 20 booth in the art fair. The student 145 comes and having promotion for one. Then 4 student comes and wait for next promotion. After the student 145 left, the booth is having a group promotion for student 145, 63, 120, 401 and 576. They leave at the same time after having group promotion.

Extra feature: make friends and visit together



```
def find_com(st_ls):  st_ls: [730, 143]
    # rnd_int = np.random.randint(10)
    if len(st_ls) < 2:
        return
    rnd_int = 2  rnd_int: 2
    if rnd_int <= 3:
        pref_ls = []  pref_ls: [(730, [14, 19, 16]), (143, [4, 12, 14])]
        wh_ls = []  wh_ls: [16, 19, 14, 4, 12, 14]
        # get the pref from each st to a ls
        for std in st_ls:  std: 143
            pref_ls.append((std, student_list[int(std)].preference))
            wh_ls.extend(student_list[int(std)].preference)
        # find highest freq. common booth
        # check the freq > 1
        if len(wh_ls) == len(set(wh_ls)):
            return
        most_common_booth = max(wh_ls, key=wh_ls.count)  most_common_booth: 14

        for std_id, pref in pref_ls:  std_id: 730  pref: [14, 19, 16]
            if most_common_booth in pref:
                # swap to top
                most_ind = pref.index(most_common_booth)  most_ind: 2
                pref[0], pref[most_ind] = pref[most_ind], pref[0]
                student_list[std_id].set_preference(pref)
```

In set_next function, while giving student promotion. It sends the serving list to the function find_com to find the most common shared booth in the serving list. If the booth is serving 6 student and 3 of them are having a same interested booth x in later, then the function will swap the booth x to the front of the preference list of those 3-student. Given that most likely students will change their mind and give top priority to the shared booth. The q set as 0.4 which if they find match student in group promotion then they will visit the common booth together with 0.3

probability.

An example for illustration. The student ID and booth ID are 0-base. It needs to +1 on both ID to match the log file. The q set to 1 for illustrate the idea.
Student:

student_id	arrival_time_day_1	arrival_time_day_2	preference_day_1	preference_day_2
144	11:08	14:29	[6, 5, 13, 15]	[13, 4, 20]
731	11:07	11:40	[6, 17, 20, 15]	[20, 15, 2]

Log file:

Day	Time	Event	len(Queue)
1	11:08	Student 144 arrives Art Fair	NA
1	11:11	Student 144 arrives at booth 6	2
1	11:14	Student 144 departs at booth 6	2
1	11:19	Student 144 arrives at booth 15	1
1	11:22	Student 144 departs at booth 15	0

1	11:07	Student 731 arrives Art Fair	NA
1	11:10	Student 731 arrives at booth 6	1
1	11:14	Student 731 departs at booth 6	2
1	11:19	Student 731 arrives at booth 15	2
1	11:22	Student 731 departs at booth 15	0

On day1, student 144 and 731 are having a group promotion on booth 6. The booth 6 are serving 144 and 731. The function detected that they have a common interesting booth which is 15. And reorder their preference list. As shown on the table below, they visit the booth 15 together after departure from booth 6.
Setting the q = 0.4 but there are actually more people which sharing a group promotion and go the same booth by coincidence.

With the same setting:

D	2
n	20
N	300
Booth_promotion_time	[3, 1, 4, 3, 3, 3, 2, 5, 2, 2, 5, 4, 1, 4, 2, 8, 10, 4, 10, 1]

Without grouping promotion and visiting together features:

Day	TotalWaitingTime	TotalTravelingTime	TotalTourTime
1	21557	5737	31977
2	26855	5518	37124

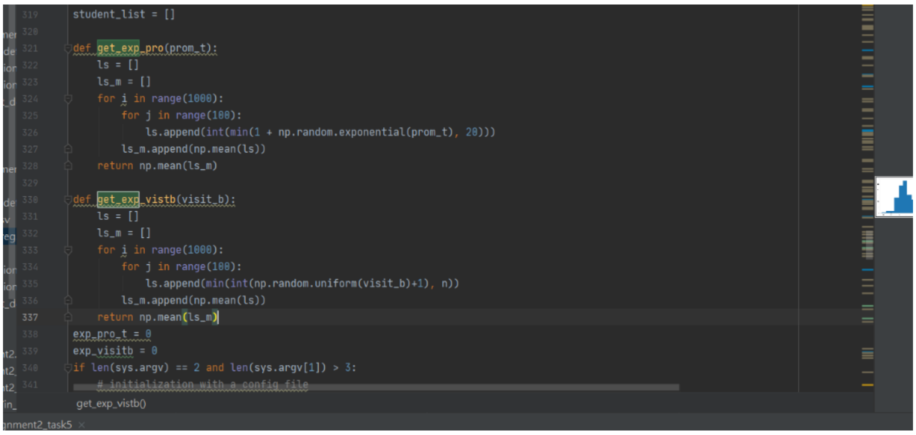
With the grouping promotion and visiting together features:

Day	TotalWaitingTime	TotalTravelingTime	TotalTourTime
1	569	6170	11829
2	689	6161	12198

As a conclusion, the grouping promotion and visiting together features can greatly reduce the waiting time by making the promotion more efficiently.

Task 5

In task 5, I use two function to compute the expected promotion time and expected number of booths visited by a student.



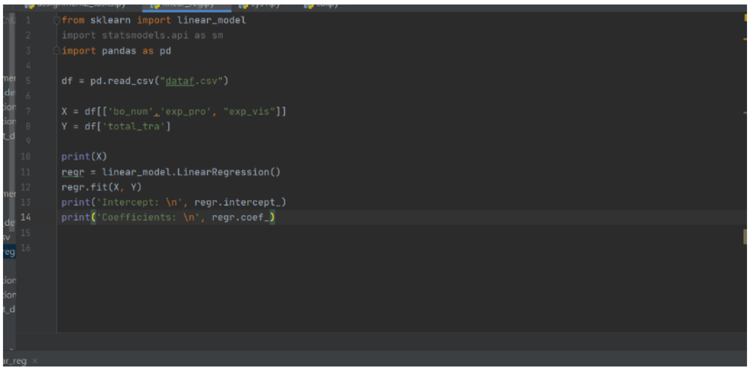
```python
student_list = []

def get_exp_pro(prom_t):
    ls = []
    ls_m = []
    for i in range(1000):
        for j in range(100):
            ls.append(int(min(1 + np.random.exponential(prom_t), 20)))
        ls_m.append(np.mean(ls))
    return np.mean(ls_m)

def get_exp_vistb(visit_b):
    ls = []
    ls_m = []
    for i in range(1000):
        for j in range(100):
            ls.append(min(int(np.random.uniform(visit_b)+1), n))
        ls_m.append(np.mean(ls))
    return np.mean(ls_m)
exp_pro_t = 0
exp_visitb = 0
if len(sys.argv) == 2 and len(sys.argv[1]) > 3:
    # initialization with a config file
```

For making different simulation, I use a bath file same as task 1 and produce different data and output it to a excel file (“dataf.csv”) for collecting the data.

Data collected:

bo_num	exp_pro	exp_vis	total_tra
186	9.112735	8.517473	35804
194	11.02343	31.50432	38838
9	3.531119	2.998864	1484
34	9.057583	9.468668	9638
38	11.46147	19.49666	11288
13	3.515267	4.007206	2534
124	1.580352	16.06814	40102

I collect 100 data with different data for doing regression.



```python
from sklearn import linear_model
import statsmodels.api as sm
import pandas as pd

df = pd.read_csv("dataf.csv")

X = df[['bo_num','exp_pro', "exp_vis"]]
Y = df['total_tra']

print(X)
regr = linear_model.LinearRegression()
regr.fit(X, Y)
print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)
```

Use a built-in function in Sklearn for linear regression.

Output:

```
Intercept:
8294.86543604935
Coefficients:
[ 198.43489868 -764.63977606    150.35732128]
```

We can conclude that the promotion time have highest weighting which shows it has a large significant effect on the traveling time. Increase the time on the promotion can reduce the time of traveling time of student. And the booth number is the second high weighting.