**Aim:**

Different vans at Habib University passes through the same location but leave and different times and take different routes. This means that a new student may find it difficult to choose the right van that reaches their desired location as early as possible. To tackle this problem, We have designed an algorithm, that does three things:

1. If a van or a number of vans already go to the desired location of the new student, it returns them with the list or individual van that goes there. If multiple vans go there. It returns the list of vans in the sorted order, starting from the van that reaches there in the shortest time.
2. If the address needs to be registered, it will take an input of the address and automatically assign it to the van that passes by that location on its route. It then creates a new path for the van based on the newly added location.
3. If the transport in charge of the university wants to delete a specific location from a specific van, the algorithm can delete that location from that van and automatically create the new path for the van based on the updated list.

**DATA:**

* Raw data was obtained from the point services' supervisor , in excel worksheet format with 34 different worksheets consisting destination data and pickup times.
* The raw data is compiled into a single excel file saved as a csv. The data for pickup times had been discarded. It is formatted in a way such that the first column represents the point number followed by the place the point will go to, its distance from HU, it’s distance upon following the route, the order in which it is visited followed by the points and the departure time of the point from HU.
* The data in the destinations’ column consists of names of varying length and a noticeable portion of the data consists of more than one key-word. The Destinations’ data was altered and the blank spaces between the keywords were replaced by hyphens. This was done to make implementation of the data easier and constant in the overall number of data entries. This made the implementation of the code for reading data reduce in size and number of commands being executed.

**CODE:**

* The data-set is produced in the form of a dictionary in which the keys for the dictionary represent the point number and the items for that key hold data about the destination, distance from Habib University, distance from Habib in followed route, drop-off number and departure time in a list. So each destination has a separate list containing is data.

**Algorithm for Data Set:**

**create\_list(document):**

**file = Open (document)**

**file1 = {}**

**for each line in file:**

**if line is the column line:**

**pass**

**else:**

**lst = line[2:len(line)-1].split()**

**if lst[1] is not in file then:**

**file1[lst[1]] = []**

**file1[lst[1]].append(lst[2:])**

**else:**

**file1[lst[1]].append(lst[2:])**

**Algorithm for time calculation:  
 def timer(time):**

**f = float(time) – int(time)**

**if f > 0.59:**

**time = int(time+1.0)**

**f= f - 0.60**

**time = time + f**

**return time**

**Algorithm for obtaining the list of vans:**

**destination = input from the user**

**vans=[]**

**for key in file1:**

**for items in file1[key]:**

**if destination is in items:**

**t = ((distance in items/30)\*60)/100)**

**time = departure-time + t**

**timer(time)**

**result = [key, hold, dropoff-order]**

**vans.append(res)**

**Algorithm for bubble sort:**

**def bubblesort(vans):**

**index = length of vans -1**

**while index >= 0:  
 for j in range(index):**

**if arrival-time for vans[j] > arrival-time for vans[j+1]:**

**vans[j], vans[j+1] = vans[j+1], vans[j]**

**index = index – 1**

**return vans**

**Algorithm for choosing the best van:**

**def chooseBest(vans):**

**if length(vens) == 1:**

**print(('Van: '+str(vans[0][0])+', Estimated time of drop off:’ +str(round(vans[0][1],2))+' p.m, Order: '+vans[0][2])**

**else:**

**bubblesort(vans)**

**Algorithm for obtaining distance between origin and destination node:**

**def dist(origin,destination):**

**nav\_request = 'origin={}&destination={}&key={}'.format(origin,destination,api\_key)**

**request= endpoint + nav\_request**

**response = urllib.request.urlopen(request).read()**

**directons = json.loads(respose)**

**return directions[“routes”][0][“legs”][0][“distance”][“value”]**

**Algorithm for bubble sort #2:**

**def bubblesort(vans):**

**index = length of vans -1**

**while index >= 0:  
 for j in range(index):**

**if distance from habib on route for vans[j] > distance from habib on route vans[j+1]:**

**swap the two value positions**

**index = index – 1**

**return vans**

**Algorithm for align code:**

**def align(a):**

**count=length(a)**

**for i in range(0, length(a)):**

**for each order number, replace it with the count value starting from count = length(a),**

**then decrementing the value for count by 1**

**return a**

**Algorithm for adding a node:**

**def addingAnode(address):**

**van = input()**

**van = int(van)**

**distance = (dist(add,uni))/1000**

**uniDist = distance**

**for i in file1[van]:**

**f = replace hyphens with ‘+’ in i[0]**

**distance1 = (dist(add,f)/1000) to get distance in kilometres**

**if distance < distance1:**

**pass**

**else:**

**pos = 0**

**distance = distance1+distance in i**

**dataInput = [address, uniDist, distance, pos, departure time from file1[van]]**

**file1[van].append(dataInput)**

**bubble\_sort2(file1[van[)**

**print(align(file1[van]))**

**Algorithm for the deletion of nodes:**

**addressD = input from user**

**vanD = input from user**

**def deletingNodes(addressD,vanD):**

**for i in file[vanD]:**

**if destination in I == addressD;**

**remove I from file1[vanD]**

**sort and align the function again**

**bubble\_sort2(file1[vanD]))**

**align(file1[vanD])**

**print(file[vanD])**

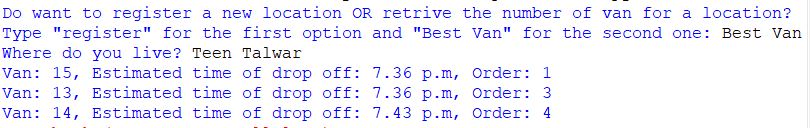
Test cases

## Retrieving the number of the best van:

### Case #1

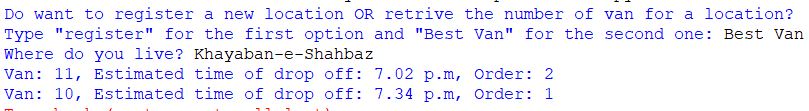
Location: Teen Talwar

Note: The algorithm not only finds the van, it sorts them firstly based on their time and if the time is same, it sorts them according to their order of drop off.



### Case#2

Location: Khayaban-e-Shahbaz

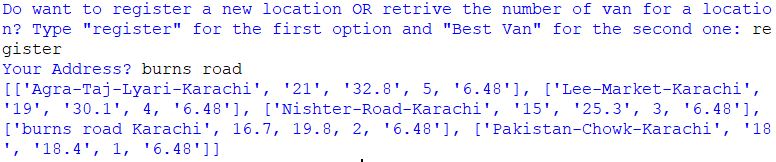


## Registering A new Location:

Note: This algorithm only requires the location and it figure out itself which van goes closest to it and assigns it to the van.

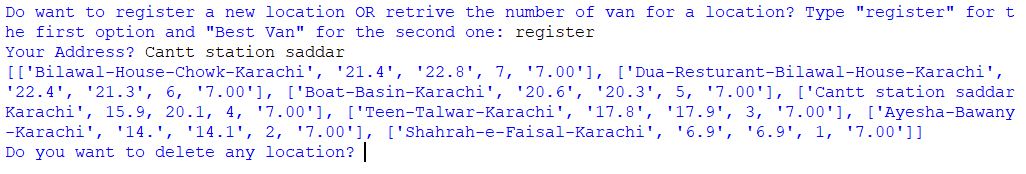
### Case#1

Location: Burns road



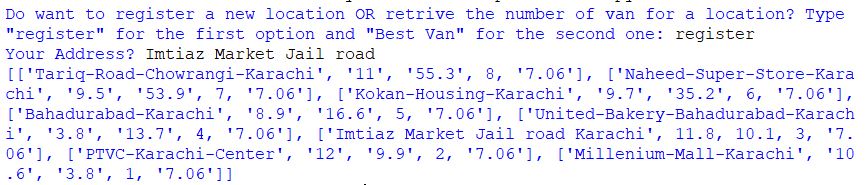
### Case#2

Location: Cantt Station Saddar



### Case#3

Location: Imtiaz Market Jail Road



## Deleting a location:

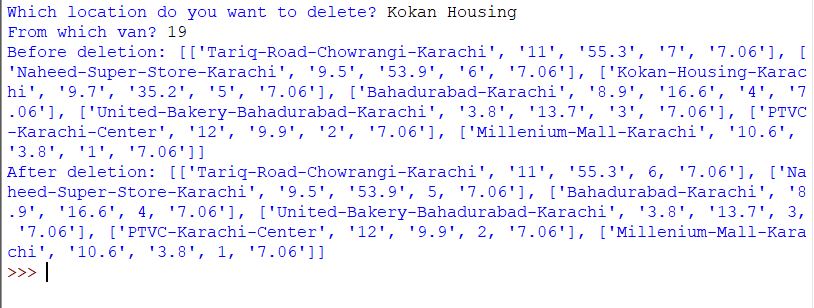
Note:

This algorithm deletes the input location from the input van. This algorithm is made not for the students but the transport in charge himself.

## Case#1

Location: Kokan Housing

Van: 19



## Case2:

Location: Lal Kothi

Van: 12

