**Hashing and Collision Resolve [CO3]**

Instructions for students:

● Complete the following methods on Hashing.

● You may use any language to complete the tasks.

● All your methods must be written in one single .java or .py or .pynb file. DO NOT CREATE separate files for each task.

● If you are using JAVA, you must include the main method as well which should test your other methods and print the outputs according to the tasks.

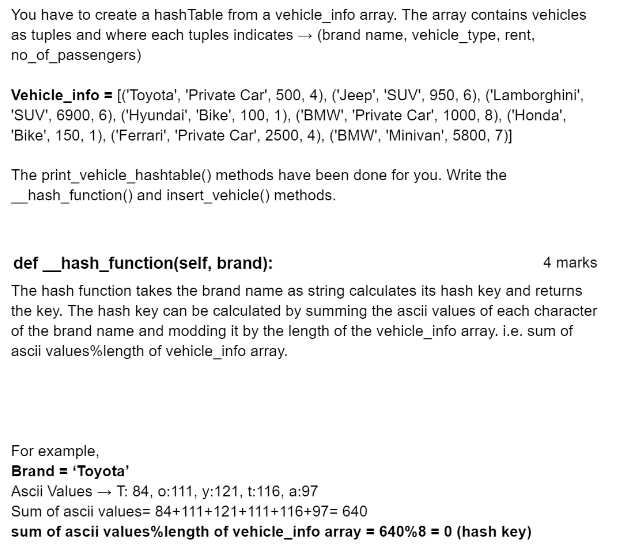
● If you are using PYTHON, then follow the coding templates shared in this folder.

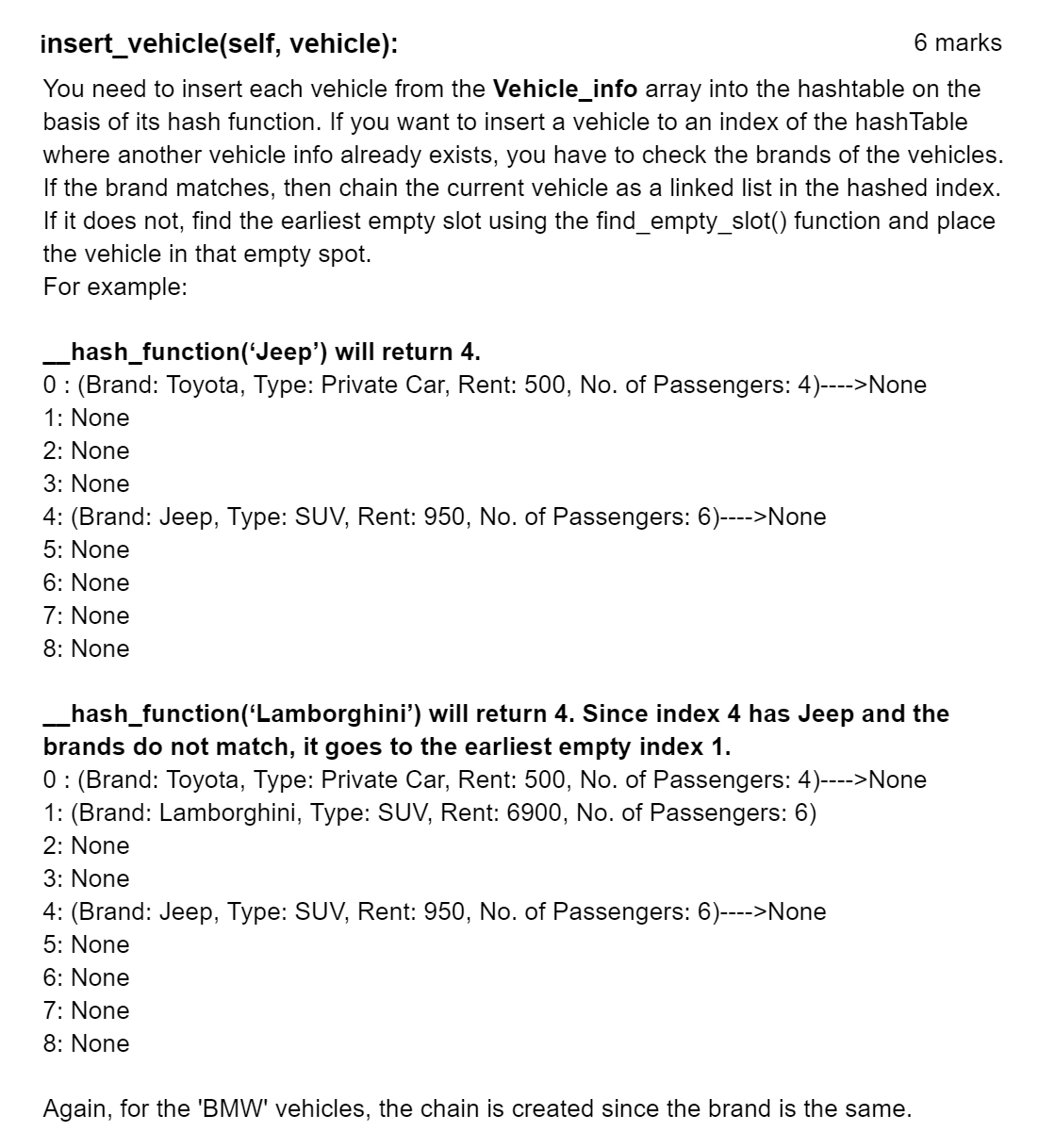
NOTE:

**● YOU CANNOT USE ANY BUILT-IN FUNCTION EXCEPT** len **IN PYTHON. [negative indexing, append is prohibited]**

**● YOU HAVE TO MENTION SIZE OF ARRAY WHILE INITIALIZATION**

## Hashtable with Forward Chaining:





## Searching in hashtable:

Complete the \_\_hash\_function() and search\_hashtable() methods in the given colab file . **Do not** change the given code; implement only the required methods. Creating and Inserting into a hash table using forward chaining is already done in the class. Do not initialize any other instance variable other than the given ones.

1. search\_hashtable(self,s) → this instance method takes a **key value pair** in the parameter then uses the string key to search for the string in the instance variable **ht**. If the string s is found, this method returns **‘Found’**, else returns **‘Not Found’**. (**Do not** implement sequential search, implement the hash based search.)
2. \_\_hash\_function(self,key) → this instance method takes only the string key, calculates the hashed index on key and returns the index. This hash function takes consecutive two letters of the key string, concatenates their ascii values into an integer and sums all the concatenated integers. Then it finds out the modulus of the summation (**think for yourself with which number should we mod the summation**) as the hashed index.

For instance, for a string ‘Mortis’, the consecutive two letters are Mo, rt, is. The concatenated integer for

Mo is 77111 (Ascii of M is 77, o is 111);

rt is 114116 (Ascii of r is 114, t is 116);

‘is’ is 105115 (Ascii of i is 105, s is 115).

The summation is = 77111+114116+105115

Mod the summation with \_\_\_\_\_\_\_\_\_\_ (**fill in the gap**) and return the answer as the hashed index.

(As for an odd length string, add the letter 'N' at the end of it. Thus, ‘Morti’ becomes ‘MortiN’ and the consecutive two letters are Mo, rt, iN)

## 

## 3. Hashtable with Forward Chaining 2.0

Write the insert() function and hash\_function() of the HashTable class which uses an array to store a key-value pair, where the key is a string representing a fruit, and the value is an int representing its respective price.

**hash\_function(key)**

Takes a key which is a string, and calculates its length. If length is even, the sum of the ascii values of the even characters is calculated, otherwise, the sum of the ascii values of odd characters is calculated.

Finally, it returns the sum modded by length of the array

If we call \_\_hash\_function(“apple”):

As len(“apple”) is odd, characters in odd indexes (p, l) are taken:

sum = ord(p) + ord(l) = 112 + 108 = 220

If the Hashtable object is initialized with length 5, then:

sum % length of Hashtable array = 220 % 5 = 0

So, the function returns 0

**insert(key, value)**

Creates a node that contains a tuple which stores the key-value pair as (key, value). Finds index by passing key to hash\_function(). If there is no collision, the node is placed in the index.

If there is a collision, forward chaining is applied. Here, the chain should be arranged in **descending order**, i.e, if the node being inserted has the highest value (price), it will be the head of the chain. Otherwise, you should iterate the chain and insert it in the appropriate position.

4. Deletion from hashtable

You are given the hash function, **h(key) = (key + 3) % 6** for a hash-table of length 6. In this

hashing, forward chaining is used for resolving conflict and a 6-length array of singly linked lists

is used as the hash-table. In the singly linked list, each node has a next pointer, an Integer key

and a string value, for example: (4 (key) , “Rafi” (value)). The hash-table stores this key-value

pair.

Implement a function **remove(hashTable, key)** that takes a key and a hash-table as

parameters. The function removes the key-value pair from the aforementioned hashtable if such a key-value pair (whose key matches the key passed as argument) exists in the hashtable.

Consider,Node class and hashTable are given to you. You just need to complete the

remove(hashTable, key) function.

class Node:

def \_\_init\_\_(self, key, value, next=None):

self.key, self.value, self.next = key, value, next

Sample Input and Output:

Some Key-value pairs:

(34, “Abid”) , (4, “Rafi”), (6, “Karim”), (3, “Chitra”), (22, “Nilu”)

HashTable is given in the following:

0: (3, “Chitra”)

1: (22, “Nilu”) → (4, “Rafi”) → (34, “Abid”)

2: None

3: (6, “Karim”)

4: None

5: None

**remove(hashTable, key=4)** returns the changed hashTable where (4, “Rafi”) is removed.

New HashTable Output:

0: (3, “Chitra”)

1: (22, “Nilu”) → (34, “Abid”)

2: None

3: (6, “Karim”)

4: None

5: None

**remove(hashTable, key=9)** returns the same given hashTable since 9 doesn’t exist in the table.