```
# Code écrit par Morgane Farez L1 CMI Informatique USMB
# Hash Table version 1
#liste de listes

collision = [0]
tab_nb_elements = [0]

#taille des listes de collisions
tab_liste_collisions = [0]


### DEBUT ###

### FONCTION HACHAGE ###
def hashFunction(cle):
    """
    Associe une case a un element donne.
    Parameters
    ----------
    cle : entier ou chaine de caracteres
    table : table de hachage (tableau de tableaux de tuples)
    Returns
    -------
    pos : entier
        indice de l'element dans la table de hashage.

    """
    #fonction de hachage 2
    if type(cle) == int:
        pos = cle*3
    else :
        pos = 0
        for i in range(len(cle)):
            pos = pos*17 + ord(cle[i])*97#i #ord() permet de recuperer le numero ascii du caractere
    return pos%(2**64)


### TABLE DE HACHAGE ###
class HashTable:
    def __init__(self, size):
        '''
        creer un tableau de tableaux de la taille demandee a l'initialisation
        Parameters
        ----------
        size : entier
        Returns
        -------
        None.
```

```
    '''
    self.table = []
    self.nb_collisions = 0
    self.nb_case_remplie = 0 # compte le nombre de case remplie dans la table
    self.nb_elements = 0 # compte le nombre de couples cle/valeur inseres dans la table
    self.taille = size
    for i in range(self.taille):
        self.table.append([])

def __str__(self):
    '''
    permet un affichage comme les dictionnaires python
    Returns
    -------
    affichage : chaine de caracteres
    '''
    affichage = '{'
    for i in range(len(self.table)):
        if len(self.table[i]) != 0 :
            for j in range(len(self.table[i])):
                affichage = affichage +  f'{self.table[i][j][0]} : {self.table[i][j][1]}, \n '
    affichage += '}'
    return affichage

def ajouter_valeur(self, cle, valeur):
    """
    Ajoute l'element a la position indique de la table
    Parameters
    ----------
    cle : chaine de caracteres ou entier
    valeur : pas de type particulier
    Returns
    -------
    None.
    """
    position = (hashFunction(cle)) %  len(self.table)
    recherche = self.recherche_valeur(cle)
    if recherche != None: # la cle est deja dans la table
        for i in range(len(self.table[position])):
            if self.table[position][i][0] == cle:
                self.table[position][i] = (cle, valeur) #on met a jour la valeur
                break
    else:
        if len(self.table[position]) == 0:
            self.nb_case_remplie += 1
        else: #si la liste n'est pas vide
            self.nb_collisions +=1
            #permet de connaitre la taille des listes de collisions
```

```
            tab_liste_collisions.append(len(self.table[position]))

            # permet de comptabiliser les collisions dans un tableau (cf.nuage de points)
            tab_nb_elements.append(self.nb_elements)
            collision.append(self.nb_collisions)


        self.table[position].append((cle, valeur))
        self.nb_elements += 1

    self.agrandir_table() # agrandit la table si necessaire

def __setitem__(self, cle, valeur):
    '''
    permet d'ajouter un couple cle/valeur a la table avec de la facon
    suivante:
        t['r']='t'
    '''
    self.ajouter_valeur(cle, valeur)

def recherche_valeur(self, cle):
    '''
    Renvoie la valeur correspondante à la clé
    Parameters
    ----------
    cle : entier ou chaine de caracteres
    Returns
    -------
    valeur : pas de type particulier
    '''
    position = (hashFunction(cle))%len(self.table)
    valeur = None
    for i in range(len(self.table[position])):
        if self.table[position][i][0] == cle:
            return self.table[position][i][1]
    return valeur

def __getitem__(self, cle):
    '''
    permet de faire une recherche d'une valeur dans une table T de la facon
    suivante:
        T[cle]
    Parameters
    ----------
    cle : eniter ou chaine de caracteres
    Returns
    -------
    renvoie la valeur associee a la cle
```

```
    '''
    return self.recherche_valeur(cle)

def __get__(self, cle):
    '''
    but : rechercher valeurs : t.get(cle)
    Mais fonctionne seulement comme : t.__get__(cle)
    Parameters
    ----------
    cle : entier ou chaine de caracteres
    Returns
    -------
    renvoie la valeur associee a la cle
    '''
    return self.recherche_valeur(cle)

def supprimer_tuple(self, cle):
    '''
    Supprime le couple cle-valeur de la cle indiquee
    Parameters
    ----------
    cle : entier ou chaine de caracteres
    Returns
    -------
    None.
    '''
    position = (hashFunction(cle))%len(self.table)
    self.nb_elements -=1
    for i in range(len(self.table[position])):
        if self.table[position][i][0] == cle:
            del self.table[position][i]
            if len(self.table[position])==0:
                self.nb_case_remplie -= 1
            elif len(self.table[position][i])==1:
                self.nb_collisions -=1

def __delitem__(self, cle):
    '''
    permet de supprimer un element d'une table t de la façon suivante :
        del t[cle]
    '''
    return self.supprimer_tuple(cle)

def agrandir_table(self):
    """
    test si la table est remplie a plus de 2/3 de sa taille et double sa
    taille si c'est le cas
    Returns
```

```
-------
None.
"""
if self.nb_case_remplie >= ((2/3)*self.taille):
    ancienneTable = self.table
    self.__init__(self.taille*2)
    for i in range(len(ancienneTable)):
        for j in range(len(ancienneTable[i])):
            #recalcule la position de chaque elements presents dans ancienneTable
            self.ajouter_valeur(ancienneTable[i][j][0], ancienneTable[i][j][1])
```