```python
# matrix.py

import numpy as np

class Matrix:

    mat = [[]]
    nb_line = 0
    nb_column = 0

    def __init__(self,n=1,m=0):
        """Creer une matrice initialisée à 0:

            Paramètres:
            n (int) : nombre de lignes de la matrice
            m (int) : nombre de colonnes de la matrice"""

        self.nb_line = n
        self.nb_column = m
        self.setToNull()

    @staticmethod
    def fromArray(array):
        """Créé une matrice à partir d'un tableau de tableau.
            Les tableaux internes seront les lignes de la matrice,
            ils doivent donc tous avoir la même taille"""
        n = len(array)
        m = len(array[0])
        for line in array:
            if len(line) != m:
                raise Exception("All the lines do not have the same size")
        mat = Matrix(n,m)
        for i,line in enumerate(array):
                mat.changeLine(i,line[:])
        return mat

    def __getitem__(self, key):
        """Surcharge l'opérateur []"""
        return self.mat[key]

    def __str__(self):
        """Surcharge l'opérateur str()
            Utilisé par la fonction print()"""
        return "Nombre de lignes = {}, colonnes={}, \nmatrice={}".format(
        self.nb_line,self.nb_column,self.mat)

    def __eq__(self,M):
        return self.mat == M.mat

    def setToNull(self):
        """Rend la matrice nulle en gardant ses dimensions"""
        self.mat = [[]]*self.nb_line
        line = [0]*self.nb_column
        for i in range(0,self.nb_line):
            self.mat[i] = line[:]

    def getTransposed(self):
        """Renvoie la transposée de la matrice """
        transposed = Matrix(self.nb_column,self.nb_line)
        for i in range(0,self.nb_line):
            for j in range(0,self.nb_column):
                transposed[j][i] = self.mat[i][j]
        return transposed

    def transpose(self):
        """Transpose la matrice """
        self.mat = self.getTransposed().mat
        self.nb_line,self.nb_column = self.nb_column, self.nb_line

    def getHeight(self):
        """Renvoie le nombre de lignes de la matrice"""
        return self.nb_line

    def getWidth(self):
        """Renvoie le nombre de colonne de la matrice"""
        return self.nb_column

```

```
076|     @staticmethod
077|     def multiply(A,B):
078|         """Multiplie deux matrices entre elles
079|
080|             A = Matrix.fromArray([[0,0],[1,0],[0,1],[1,1]])
081|             B = Matrix.fromArray([[1,1,1,1],[0,0,0,0]])
082|             print(Matrix.multiply(A,B))
083|             #print(multiplyMatrixes([[0]],[[0]]))
084|         """
085|         n = A.getHeight()
086|         m = B.getWidth()
087|         if A.getWidth() != B.getHeight():
088|             raise Exception("Matrixes have the wrong dimensions: A : l={},c={} B:
l={},c={}".format(A.getHeight(),A.getWidth(),B.getHeight(),B.getWidth()))
089|         t_B = B.getTransposed()
090|         q = A.getWidth()
091|         C = Matrix(n,m)
092|         for i in range(0,n):
093|             for j in range(0,m):
094|                 #On traite des matrices de bits, on est donc dans le corps
095|                 #Z/2Z où 1+1 = 0
096|                 C.mat[i][j] = np.dot(A[i],t_B[j]) % 2
097|         return C
098|
099|     @staticmethod
100|     def toColumn(array):
101|         n = len(array)
102|         m = Matrix(n,1)
103|         for i in range(0,n):
104|             m.mat[i][0] = array[i]
105|         return  m #Matrix.fromArray([array]).getTransposed()
106|
107|     def changeLine(self,line_index,new_line):
108|         self.mat[line_index] = new_line[:]
109|
110|     def searchColumn(self,column):
111|         if  self.nb_line != column.getHeight():
112|             raise Exception("Wrong dimensions, len_column: A={},
B={}".format(self.nb_line,len(column)))
113|         for i in range(0,self.nb_column):
114|             same = True
115|             for j in range(0,self.nb_line):
116|                 if self.mat[j][i] != column.mat[j][0]:
117|                     same = False
118|                     break
119|             if same:
120|                 return i
121|         return -1
122|
123|     def multiplyLine(self,line_index,column):
124|         n = column.getHeight()
125|         C = 0
126|         for i in range(0,n):
127|             C ^= self.mat[line_index][i]*column.mat[i][0]
128|         return [[C]]
129|
130|     @staticmethod
131|     def multiplyLineWithColumn(line,column):
132|         #On traite des matrices de bits, on est donc dans le corps
133|         #Z/2Z où 1+1 = 0
134|         return np.dot(line,column) % 2
135|
136| def testMatrix():
137|     m = Matrix(4,5)
138|     m[0][3] = 1
139|     t = m.getTransposed()
140|     print(m)
141|     print(t)
142|     c= Matrix.multiply(m,t)
143|
144|     print(Matrix.toColumn([0,1,2,3,4,5]))
145|     print(Matrix(2,1) == Matrix.fromArray([[0],[0]]))
```