Question 4: 2d Binary Matrices Class (OOP)

We are going to develop a class for 2d binary matrices. We simply call that class 'bmatrix'. let's limit the class to only binary numerical matrices (elements are either 0 or 1).

```
class bmatrix:
    pass
```

When an instance of the class bmatrix is defined (__init__), declaration of the number of rows and columns is obligatory. The 3rd parameter is the initial value of the matrix (See the examples and test cases below).

Attributes/ Data members:

- MTX: the binary matrix
- rows: number of rows
- cols: number of columns

Methods/ Member functions:

- in your constructor, __init__(), you need the fill parameter too. Fill is either 0 or 1 to initiate your matrix.
- print(self, sprt): to print the matrix, nice and ordered, sprt is the separator character which separates the columns. Default is space (' ').
- How to define a default parameter?
- add(self, N): to binary add to matrices of the same size. It must check the sizes. N is the other matrix. if N has got different size, an error message should be printed and self shouldn't be changed. By binary addition we mean a kind of modula-2 addition: 0 ← 0+0 , 1← 0+1 , 1 ← 1+0 , 0 ← 1+1
- set_elem(self,r,c,stuff): set the self's element at [r][c] to stuff. Stuff only can be 0 or 1. Anything else means printing an error message and doing nothing.
- get_elem(self,r,c): returns the self[r][c]
- mul(self, N): element-wise multiplication of self and N, self[i][j] *= N[i][j], size check is a must, just like the add method.
- b_or(self, N): bit or of self and N, element-wise. Size check is a must, just like the add method.
- list(self): to convert/ cast the object to an ordinary 2d list. Useful when you want to return the results for checking.

Test Cases

You need to add the test cases below to your module for testing and evaluation purposes:

```
#
# bmatrix class
#

class bmatrix:
   pass

#----------- TEST CASES ----------- def
bm_test1():
aa          = bmatrix(3,4,0)     bb =
bmatrix(3,4,1)
   aa.print(',')
bb.print(' ')
aa.add(bb)
return aa.list()

#--------------------------------
def bm_test2():
bb          = bmatrix(3,4,1)
bb.set_elem(2,2,0)
   return bb.list()

#--------------------------------
def bm_test3():
   aa = bmatrix(5,5,1)
bb = bmatrix(6,6,1)
bb.set_elem(3,3,0)
bb.set_elem(4,4,5.25)
bb.mul(aa)     return
bb.list()
#-------------------------------
def bm_test4():
   cc = bmatrix(5,5,0)
dd = bmatrix(5,5,1)
dd.set_elem(3,3,0)
dd.set_elem(4,4,0)
dd.b_or(cc)
   return dd.list()
```

Results: After running your program we expect those 4 test cases above to return:

```
>>> bm_test1()
0 ,0 ,0 ,0
0 ,0 ,0 ,0
0  ,0 ,0 ,0
1  1  1  1
1  1  1  1
1  1  1  1
[[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]]
>>> bm_test2()
[[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 0, 1]]
>>> bm_test3()
Error ! wrong value
Error ! different sizes
[[1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1], [1, 1, 1, 0, 1, 1], [1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1]] >>> bm_test4()
[[1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 0, 1], [1, 1, 1, 1, 0]]
```

==========