```
################################################################################
# Kenneth Drew Gonzales
# Professor Bao Wang
# Math151A Hw8
#
# Here is a simple python script that will solve a linear system. This uses
# gaussian substitution (back substitution) withouth any pivoting optimization.
#
# TODO: I need to add more user intuitive features and more I/O checking
# because rn this will totally break if you don't know what to do.
################################################################################

class Matrix:
    def __init__(self):
        print("How many rows do you have: ")
        self.rows = int(input())
        self.matrix = []
        for i in range(0, int(self.rows)):
            strarr = input().split(' ')
            row = [int(num) for num in strarr]
            # if len(row) != (self.rows + 1):
            #     print("Error: Not a linear system.\n")
            #     exit(1)
            self.matrix.append(row)

    def __swapRow(self, i, j):
        temp = self.matrix[i]
        self.matrix[i] = self.matrix[j]
        self.matrix[j] = temp
        print("Swapped ", i,  " and ", j)

    def __scaleRow(self, scalar, i ):
        for x in range(0, len(self.matrix[i])):
            self.matrix[i][x] = float(scalar) * float(self.matrix[i][x])

    def display(self):
        for i in range(0, len(self.matrix)):
            print(self.matrix[i])

    def __addRow(self, i, j):
        for x in range(0, len(self.matrix[i])):
            self.matrix[j][x] += self.matrix[i][x]

    def __upperTri(self):
        for c in range(0, len(self.matrix[0]) - 1):
            for r in range(c+1, len(self.matrix)):
                s = -(float(self.matrix[r][c]) / float(self.matrix[c][c]))
                self.__scaleRow(s, c)
```

```
                self.__addRow(c, r)

    def __backSub(self):
        for c in range(len(self.matrix[0]) - 2, -1, -1):
            for r in range( c, -1, -1):
                s = -(float(self.matrix[r][c]) / float(self.matrix[c][c]))
                self.__scaleRow(s, c)
                self.__addRow(c, r)

    def __scaleBack(self):
        for c in range(0, int(self.rows)):
            self.__scaleRow( float(1/self.matrix[c][c]), c)

    def solve(self):
        self.__upperTri()
        self.__backSub()
        self.__scaleBack()
#### End Matrix

if __name__ == "__main__":
    matrix = Matrix()
    matrix.solve()
    matrix.display()
```