In this lab, you will implement Gaussian Elimination and test your implementation by solving a linear system. **Sample runs of the Forward and Backward steps are available for download form the course webpage.** Recall that in Python, you can use a *list of lists* to store a matrix, with each inner list representing a row. For example, you can store the matrix

$$\begin{pmatrix} 5 & 6 & 7 \\ 0 & -3 & 5 \end{pmatrix}$$

by storing each row as a list: `M = [[5,  6, 7], [0, -3, 5]]`. For ease of reading, since Python allows for line breaks inside brackets, you can write it as follows:

```
M = [[5,  6, 7],
     [0, -3, 5]]
```

You can use `M[1]` to access the second row of `M` (i.e., `[0, -3, 5]`), and you can use `M[1][2]` to access the third entry in the second row (i.e., `5`).

You will first implement some helper functions, then use them to implement the *forward step*, then implement the *backward step*, and finally you will use your functions to solve a linear system. This lab is longer than usual –

Problem 1.

Download `numpy.py` and understand how to print lists of lists as matrices by converting lists of lists to arrays (lines 1-11). Write a function `print_matrix(M_lol)` which prints the nested list `M_lol` as a matrix. You will be able to use your function to debug your implementation of Gaussian Elimination.

Problem 2.

Write a function with the signature `get_lead_ind(row)` which takes in a list of numbers `row`, and returns the index of the first non-zero element of `row`. The function should return `len(row)` if the row contains no non-zero elements.

Problem 3.

Write a function with the signature `get_row_to_swap(M, start_i)` which takes in a matrix `M` (represented as a list of lists) and an integer `start_i`, and returns the row that needs to be swapped (permuted/interchanged) with the row `M[start_i]`. The row that needs to be swapped with the row `M[start_i]` is the row at index larger or equal to `start_i` which has the leading non-zero coefficient that is as far to the left as possible.

For example, for

```
M = [[5, 6, 7, 8],
     [0, 0, 0, 1],
     [0, 0, 5, 2],
     [0, 1, 0, 0]]

start_i = 1,
```

`[0,0, 0, 1]` needs to be swapped with `[0, 1, 0, 0]`, so that the function should return `3`.

Problem 4.

Write a function with the signature `add_rows_coefs(r1, c1, r2, c2)` which takes in rows (represented as lists of equal lengths) `r1` and `r2` and coefficients (floats) `c1` and `c2`, and returns a *new* list that contains the row `c1*r1 + c2r2`. (N.B.: to create a new list of 10 zeros, you can use `[0]*10`.)

Problem 5.

Write a function with the signature `eliminate(M, row_to_sub, best_lead_ind)` which takes in a matrix `M` (represented as a list of lists), `row_to_sub`, an index of the row to subtract from other rows to eliminate, and `best_lead_ind`, the index of the coefficient to be eliminated in rows below index `best_lead_ind`. Assume that `M[row_to_sub]` is all zeros before `M[row_to_sub][best_lead_ind]`

For example, an input might be

```
M = [[5,  6, 7,  8],
     [0,0, 1, 1],
     [0, 0, 5, 2],
     [0, 0, 7, 0]]

row_to_sub = 1
best_lead_ind = 2
```

The function should change `M` to become

```
M = [[5, 6, 7, 8],
     [0, 0, 1, 1],
     [0, 0, 0, -3],
     [0, 0, 0, -7]]
```

by adding `-5*M[1]` to `M[2]` and `-7*M[1]` to `M[3]`.

Problem 6.

Write a function with the signature `forward_step(M)` which takes in an arbitrary matrix `M` (as a list of lists), applies the forward step of Gaussian Elimination to it, and modifies M to be the matrix obtained after the forward step is applied. This can be done by repeatedly calling `get_row_to_swap`, swapping rows, and calling `eliminate`. Unlike with ESC103, I recommend that you keep the entire matrix rather than extracting submatrices. The process of performing the forward step remains essentially the same. The process is illustrated on the next page. As you write `forward_step()`, add print() statements to `forward_step()` to produce output similar to the examples provided on this handout (i.e, print out the matrix transformation process, and comments on what's happening at every step).

Problem 7.

Write a function with the signature `backward_step(M)` which takes in an arbitrary matrix `M` and applies the backward step of Gaussian Elimination to it. A sample run of the backward step (with automatically-generated explanations of what's going on) is illustrated at the end of this handout.

Problem 8.

Now, write a function that solves the equation $Mx = b$ for the vector x. The idea is to first build the augmented matrix $(M|b)$, then apply Gaussian Elimination to the augmented matrix, and then solve for x. Test your `solve()` function. In `numpy.py`, we provide code to quickly perform matrix multiplication in Python so that you can pick arbitrary `M` and `x`, obtain a `b` by multiplying `M` and `x`, and then verify that your algorithm can recover the `x`.