# Python Exercises for Homework-1

- Do not provide your name on the assignment.
- Convert your python assignment to a pdf and submit with your mathematical part.

In [247…

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

**Question 1** Use or modify the code provided for image manipulation in Module 1 for the following.

> - (A) Create a single image as a collage of at least 10 cartoon characters of your choice.
> - (B) Read any image and display it after flipping horizontally. Left becomes right.
> - (C) Read any image and display it after flipping vertically. Top becomes bottom.

In [248…

```python
#Create a single image as a collage of at least 10 cartoon characters of your choic
#Creating a plot.figure object with size 20X25
fig = plt.figure(figsize=(20,25))
# 3X3 Rows and columns
rows = 3
columns = 3
#Looping from 1 to 10 and plotting the images one by one.
for i in range(1,11):
    #If index is 10, Select the 8th grid
    if i == 10:
        fig.add_subplot(rows, columns, 8)
    #Else select the respective grids as i value.
    else:
        fig.add_subplot(rows, columns, i)
    #Read the image using mping
    pix = mpimg.imread("./Character_{}.jpg".format(i))
    #Plot the image
    plt.imshow(pix,cmap='gray')
    #Turn off the axis
    plt.axis('off')
    #Adjust the spacing between the images.
    plt.subplots_adjust(wspace=0, hspace=-0.3)
```

```
In [249…  #Original Image
          pix = mpimg.imread("./Cats.jpg")
          plt.axis('off')
          plt.imshow(pix,cmap='gray')
```

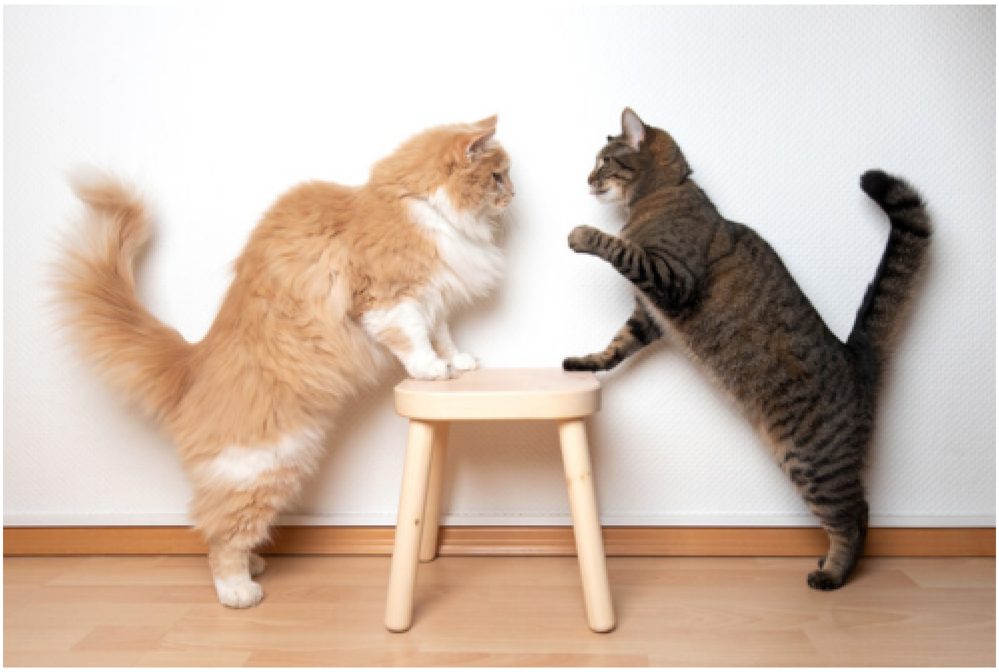
```
Out[249]: <matplotlib.image.AxesImage at 0x1b787d8e7d0>
```

In [250…

```
#Read any image and display it after flipping horizontally. Left becomes right.
pix = mpimg.imread("./Cats.jpg")
#Flipping the 2d array horizontally
pix_horizaontal = pix[:, ::-1, :]
plt.axis('off')
plt.imshow(pix_horizaontal,cmap='gray')
```

Out[250]:

```
<matplotlib.image.AxesImage at 0x1b7a823bb50>
```



In [251…

```
#Read any image and display it after flipping vertically. Top becomes bottom
pix = mpimg.imread("./Cats.jpg")
# print(pix.shape)
#Flipping the 2d array horizontally
pix_vertical = pix[::-1, :, :]
plt.axis('off')
plt.imshow(pix_vertical,cmap='gray')
```

Out[251]:

```
<matplotlib.image.AxesImage at 0x1b7a4c6ea50>
```

**Question 2** Use the python code provided in Module 2 for the following

(A) Modify the code to write a function for finding inverse. Use this to find the inverse of the following. Show the output.

In [252…

```
#Code copied from the reference module provided with the assignment.
def findRREF(A):
    '''
    Input: a general rectangular matrix
    Output: the row reduced echelon form of the matrix A
    by using Gauss-Jordan elimination
    '''
    augA = np.copy(A)
    m,n=augA.shape
    print("Augmented matrix: ",augA)
    for k in range(m):
        # Check that the matrix is not rank deficient
        if np.abs(augA[k,k]) < 10**(-15):
            exit("The given matrix is singular or requires row swapping")
        # Convert the pivot element to 1
        augA[k,:] = augA[k,:] /augA[k,k]
        for i in range(m):
            if i==k:
                continue
            z = -augA[i,k]
            # Change the entire rows (k+1)st onward
            augA[i,:] = augA[i,:] + z*augA[k,:]
        print("Pass {}:\n".format(k+1))
        print(augA)
    return augA
```

In [253…

```
def invByRREF(A):
    m,n=A.shape
    if m!=n:
        exit("To find inverse, please provide a square matrix only.")
    #Declaring a identity matrix in the form of 2d array.
    identityMatrix = np.array([[1,0,0],
                               [0,1,0],
                               [0,0,1]],dtype=float)
    #Concatenating identity matrix with original matrix provided by the user.
    augA = np.concatenate((A,identityMatrix),1)
    #Passing the concatenated matrix to findRREF function.
    modified_augA = findRREF(augA)
    #Take appropriate slice of modified_augA for inverse
    invA = modified_augA[:,3:]
    print("Inverse Matrix")
    return invA


#Creating a matrix to find the inverse.
A = np.array([[1,1,4],
             [3,2,4],
             [1,1,6]],dtype=float)
print(invByRREF(A))
```

```
Augmented matrix:  [[1. 1. 4. 1. 0. 0.]
 [3. 2. 4. 0. 1. 0.]
 [1. 1. 6. 0. 0. 1.]]
Pass 1:

[[ 1.  1.  4.  1.  0.  0.]
 [ 0. -1. -8. -3.  1.  0.]
 [ 0.  0.  2. -1.  0.  1.]]
Pass 2:

[[ 1.  0. -4. -2.  1.  0.]
 [-0.  1.  8.  3. -1. -0.]
 [ 0.  0.  2. -1.  0.  1.]]
Pass 3:

[[ 1.   0.   0.  -4.   1.   2. ]
 [-0.   1.   0.   7.  -1.  -4. ]
 [ 0.   0.   1.  -0.5  0.   0.5]]
Inverse Matrix
[[-4.   1.   2. ]
 [ 7.  -1.  -4. ]
 [-0.5  0.   0.5]]
```

(B) Modify the code to write a function for finding determinant of a general square matrix. Show the output for the matrix A given in part (A).

In [254…

```
#Modifying the code to find the determinant of the matrix
def findRREF_determinant(A):
    augA = np.copy(A)
    m,n=augA.shape
    #initialize a variable for storing the determinant value with 1.0
    determinant = 1.0
    for k in range(m):
        # Convert the pivot element to 1
        #update the determinant value
        determinant *= augA[k, k]
        augA[k,:] = augA[k,:] /augA[k,k]
        for i in range(m):
            if i==k:
                continue
            z = -augA[i,k]
            # Change the entire rows (k+1)st onward
            augA[i,:] = augA[i,:] + z*augA[k,:]
    return determinant
```

In [255…

```
print("Determinant of the Matrix\n {}\n \n is {}\n".format(A,findRREF_determinant(
Determinant of the Matrix
 [[1. 1. 4.]
 [3. 2. 4.]
 [1. 1. 6.]]

 is -2.0
```

(C) Modify the code so that it also allows row swaps. Solve the following system by using this.

$$\begin{bmatrix} 0 & 1 & 2 \\ 5 & -2 & 6 \\ 3 & 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 9 \\ 2 \end{bmatrix}$$

In [256…

```
#Modified code for solving the system of equations
def findRREF_solve(A, b):
    # Augment A with vector b
    augA = np.hstack((A, b[:, np.newaxis]))
    m, n = augA.shape
    print("Augmented matrix: \n", augA)
    for k in range(m):
        # Find the pivot row with the maximum absolute value in the current column
        pivot_row = np.argmax(np.abs(augA[k:, k])) + k
        if np.abs(augA[pivot_row, k]) < 10**(-15):
            return None, None  # Singular matrix, return None for RREF and solutio

        # Swap rows if necessary
        if pivot_row != k:
            augA[[k, pivot_row], :] = augA[[pivot_row, k], :]

        # Convert the pivot element to 1
        augA[k, :] = augA[k, :] / augA[k, k]

        for i in range(m):
            if i == k:
                continue
            z = -augA[i, k]
            # Change the entire rows (k+1)st onward
            augA[i, :] = augA[i, :] + z * augA[k, :]

        print("Pass {}:\n".format(k+1))
        print(augA)

    # Extract the solution from the augmented matrix
    solution = augA[:, -1]

    return solution

# Example usage to solve a system of linear equations:
#Declaring two arrays using numpy.
A = np.array([[0,1,2],
              [5, -2, 6],
              [3, 1, -2]], dtype=float)
b = np.array([3,9,2], dtype=float)
#Passing both the arrays in findRREF_solve function.
solution = findRREF_solve(A, b)
print()
print("Solution:", solution)
print()
print("x1 = {}\nx2 = {}\nx3 = {}".format(solution[0].round(1),solution[1].round(1).
Augmented matrix:
 [[ 0.  1.  2.  3.]
 [ 5. -2.  6.  9.]
 [ 3.  1. -2.  2.]]
Pass 1:

[[ 1.  -0.4  1.2  1.8]
 [ 0.   1.   2.   3. ]
 [ 0.   2.2 -5.6 -3.4]]
Pass 2:

[[ 1.          0.          0.18181818  1.18181818]
 [ 0.          1.         -2.54545455 -1.54545455]
 [ 0.          0.          4.54545455  4.54545455]]
Pass 3:

[[1. 0. 0. 1.]
 [0. 1. 0. 1.]
 [0. 0. 1. 1.]]

Solution: [1. 1. 1.]

x1 = 1.0
x2 = 1.0
x3 = 1.0
```

```
In [ ]:
```