

Koç University

COMP 125: Programming with Python

Homework #4

Deadline: May 26, Wednesday at 23:59pm

Submission through: Blackboard

Make sure you read and understand every part of this document

This homework assignment contains 3 programming questions. Each question may contain multiple parts.

Download [Hw4.zip](#) from Blackboard and unzip the contents to a convenient location on your computer. The [.ipynb](#) files contain starter codes for the programming questions. **You should open them with Jupyter Notebook (not Spyder).** Remaining files are sample text/data files. Solve each question in its own file.

This time, you won't be working with functions. Jupyter Notebook files will be leading you about the steps you should implement. If you want, you can add new cells while writing your codes. When you are finished, compress your Hw4 folder containing all of your answers. The result should be a SINGLE compressed file (file extension must be .zip or .rar). Upload this compressed file to Blackboard.

Q1: Mini Exercises - 30 pts (10 + 10 + 10)

Open [MiniExercises.ipynb](#) with Jupyter Notebook.

Part A:

In the first cell, you are given an **n** value. You can assume $n \geq 3$, no need for error checking.

- Your code should construct an **n** by **n** Numpy matrix that has an outer frame of 0s and all inner values are 1s.
- Important:** Do not construct a 3 by 3 Numpy matrix, your code should be working for any **n** value.

Result when n=4:

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

Result when n=5:

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

Part B:

In the first cell, you are given a list of lists:

- First, convert the given list of lists into a Numpy matrix.
- Then, swap the rows and columns of the given matrix in reverse order (*Hint: Recall slicing with step size*).
- The result should be a Numpy matrix.

When `lst = [[1., 6., 7., 4.], [8., 0., 3., 4.], [9., 1., 6., 5.], [6., 7., 0., 2.]]`, the result should be:

```
array([[2., 0., 7., 6.],
       [5., 6., 1., 9.],
       [4., 3., 0., 8.],
       [4., 7., 6., 1.]])
```

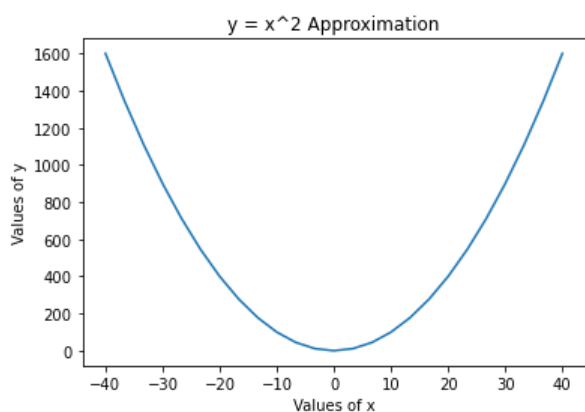
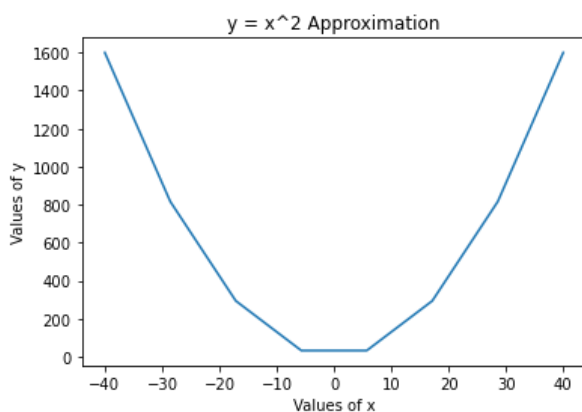
Part C:

We would like to approximate the function $y = x^2$ by evaluating it at multiple x values and plotting the corresponding values of the function.

In the first cell, you are given an integer `m`:

- First, create `m` equally spaced x values between $[-40, +40]$ and compute $y = x^2$ at each of these x values. x values should start with -40 and end with $+40$ (inclusive).
- Then plot a line graph of the x values and y values found in the previous step
 - Title of plot should be: “ $y = x^2$ Approximation”
 - X axis label should be: “Values of x ”
 - Y axis label should be: “Values of y ”
 - You do not need to modify any other details of the plot

Here are the plots generated when `m=8` and `m=25` respectively. As `m` becomes larger, you should be able to visually verify that the line graph is a better (smoother) approximation of the function $y = x^2$.



Q2: Lab Experiment - 35 pts (15 + 10 + 10)

The file associated with this question is [experiment.ipynb](#).

You are working in a research lab and you need to report the most recent experiment results to your advisor. You are given two txt files: **actual.txt** and **prediction.txt**, which contain the actual (ground truth) values and the results you obtained from your experiments, respectively.

Part A: Your first task is to write a Python code to open “**actual.txt**” and “**prediction.txt**”, and read their contents.

- First, open “**actual.txt**” and construct a list called *lst_a* which includes the values you read from “**actual.txt**”. Do not forget to close the file.
- Then, open “**prediction.txt**” and construct a list called *lst_p* which includes the values you read from “**prediction.txt**”. Do not forget to close the file.
- The values in both lists should be **floats**, do not round them.

lst_a should look like this (partially shown):

```
[25.107490514283192,  
26.7300667904594,  
23.56189644188721,  
24.576008062068034,  
20.389447649907464,  
26.472507028241232,  
20.22902326150204,  
26.43984222479451,  
29.64399594818827,  
20.539253521955825,  
28.42888911383025,  
27.703737853652715,  
22.408780604716586,  
26.34124467321852.]
```

lst_p should look like this (partially shown):

```
[25.29698087437678,  
26.97812772555742,  
23.814685436917586,  
24.871575073851254,  
20.803175297350066,  
26.440489416285697,  
20.420445484255634,  
26.362188562000114,  
30.059428858952387,  
20.47356988384456,  
28.608121808702933,  
27.704703481179966,  
22.759727278441105,  
26.403913996702236.]
```

Part B : You would like to calculate two different error metrics for your experiment: mean square error (MSE) and mean absolute error (MAE). Here are the formulas you will be using:

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 \quad \quad MAE = \frac{1}{n} \sum_{i=1}^n |Y_i - \hat{Y}_i|$$

where Y_i corresponds to actual values and \hat{Y}_i corresponds to predicted values.

You will be using `lst_a` and `lst_p` you constructed in Part A.

- First, convert these lists to Numpy arrays.
- Then calculate the MSE and MAE values using these two arrays.
 - **Hint:** Lecture 22 will be really helpful.

MSE result you should be getting for the given files: 0.08593074847863753

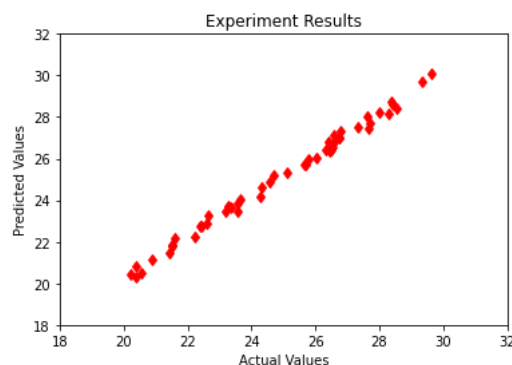
MAE result you should be getting for the given files: 0.2434272472101283

Part C :

You will be using `lst_a` and `lst_p` you constructed in Part A.

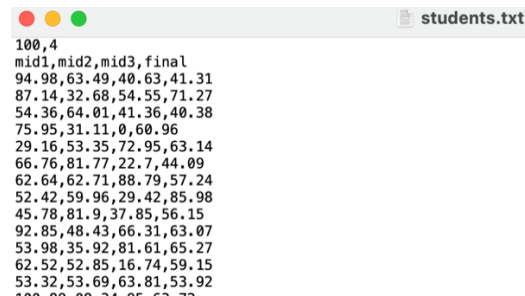
- Your output should be a plot where:
 - X axis contains the actual values (`lst_a`)
 - Y axis contains the predicted values (`lst_p`)
 - X label must be: “Actual Values”
 - Y label must be: “Predicted Values”
 - Plot title must be: “Experiment Results”
 - Your markers should be red diamonds
 - X and Y axes limits should both start from 18 and end at 32.

Here is the plot you should be getting:



Q3: Student Grades - 35 pts (15 + 5 + 5 + 10)

You are given a file containing student grades: students.txt.



```
100,4
mid1,mid2,mid3,final
94.98,63.49,40.63,41.31
87.14,32.68,54.55,71.27
54.36,64.01,41.36,40.38
75.95,31.11,0,60.96
29.16,53.35,72.95,63.14
66.76,81.77,22.7,44.09
62.64,62.71,88.79,57.24
52.42,59.96,29.42,85.98
45.78,81.9,37.85,56.15
92.85,48.43,66.31,63.07
53.98,35.92,81.61,65.27
62.52,52.85,16.74,59.15
53.32,53.69,63.81,53.92
...
```

The first line contains two integers separated by a comma. The first integer corresponds to how many students exist (in the above example, 100). The second integer corresponds to how many columns exist (in the above example, 4).

The second line contains the headers of the grades table: **mid1** is Midterm 1, **mid2** is Midterm 2, **mid3** is Midterm 3, and **final** is the final exam grade.

Then, each remaining line in the text file corresponds to the grades of one student separated by commas. You may assume that there are no errors in the file, e.g., no erroneous or empty grades, all grades are legitimate (between 0 and 100).

Open [StudentGrades.ipynb](#) and implement the following tasks.

Part A:

- Your first task is to write a Python code to open “students.txt” and read its contents.
- Your result should be a Numpy ndarray that contains the grades of all students (all floats).

For example, for the given file, the result is a 100x4 array that produces the following output (this is a *partial* output, the full array is quite large):

```
array([[ 94.98,   63.49,   40.63,   41.31],
       [ 87.14,   32.68,   54.55,   71.27],
       [ 54.36,   64.01,   41.36,   40.38],
       [ 75.95,   31.11,    0. ,   60.96],
       [ 29.16,   53.35,   72.95,   63.14],
       [ 66.76,   81.77,   22.7 ,   44.09],
       [ 62.64,   62.71,   88.79,   57.24],
       ...])
```

Part B: You will be using the array you constructed in Part A.

- Calculate the variance of the final exam grades.

The result you should be getting for the given file: 381.986665

Part C: You will be using the array you constructed in Part A.

- Find the maximum midterm grade for each student (not including the final grade). Your result should be an array including the maximum midterm grades of all students.

The result you should be getting:

```
array([ 94.98,  87.14,  64.01,  75.95,  72.95,  81.77,  88.79,  59.96,
        81.9 ,  92.85,  81.61,  62.52,  63.81, 100. ,  68.56,  81.61,
        80.09,  94.3 ,  90.67,  63.65,  77.5 ,  77.42,  69.46,  85.31,
        78.71,  89.18,  66.61,  69.07,  68. ,  75.94,  62.91,  84.82,
        74.96,  72.26,  86.69,  70.99,  58.08,  86.87,  66.2 ,  84.48,
        74.73,  60.01,  92.82,  48.03,  69.77,  92.88,  78.9 ,  52.11,
        67.26,  84.06,  85.51,  40.16,  78.45,  67.87,  79.38,  82.13,
       100. ,  71.97, 100. ,  63.37,  72. ,  53.79, 100. ,  78.25,
        89.31,  91.59,  72.36,  86.71,  66.42,  58.97,  90.62,  78.3 ,
        66.84,  83. ,  69.59,  86.39,  67.81,  65.74,  64.92,  47.41,
        65.16,  86.98,  56.75,  56.4 ,  70.18,  53.42,  42.48, 100. ,
        69.39,  77.06,  81.5 ,  75.49,  66.75,  96.66,  55.09,  96.41,
       100. ,  94.05,  85.93,  81.06])
```

Part D: *You will be using the array you constructed in Part A.*

- When calculating each student's course grade, each midterm weighs 20% and final exam weighs 40%. Based on these weights, first compute each student's course grade.
- Then, calculate the average (mean) course grade of all students.

The result you should be getting: 54.015840000000004

(Do not round the answer you are getting – your last decimal digits might be different than the one above because of Python's way of representing floats using binary system)

Submission and Grading

When you are finished, compress your Hw4 folder containing all of your answers. The result should be a SINGLE compressed file (file extension must be .zip, or .rar). Upload this compressed file to Blackboard.

Make sure that you are saving your .ipynb files periodically. After you submit your Hw4, download it from Blackboard to make sure it is not corrupted and it has the latest version of your code (check the last edit dates/times). You are only going to be graded based on your Blackboard submission. **We will not accept homework via e-mail or other means.**

Happy coding! ☺