CS215 Chapter7A Lab – Numpy Arrays and DataFrames Name:___________________

***Create a Python file called Lab7A_Array_DataFrames.py.

1. Numpy is a powerful package for scientific computing and is used heavily in computer science. We'll use it today to understand arrays.

```
import numpy as np
```

2. Let's create our first array. We have to create a list of items into the array method of numpy.

```
nums1 = np.array([1,3,5,7,99])
print(nums1)

nums2 = np.array([[1,2,3], [4,5,6]])
print(nums2)
```

   Notice how nums2 is printed with 2 rows and 3 columns because it is essentially what we would call a 2 by 3 two dimensional array.

3. Now, arrays will seem like lists to you in some sense, but there are important differences. For one thing, arrays optimized to be WAY speedier than lists. (Watch video 07_05 if you are curious.)

4. Besides being quicker, there are some other differences too. Let's explore.

   - Try these lines of code.

```
array = np.array([1, 2, 3, 'hi'])
print(array)
```

   What happens to the elements 1, 2, 3? They are converted from integers to strings. This is because arrays require that all elements have the same type. A list lets you mix types. Arrays do not.

   - Try these lines of code.

```
array = np.array([1,2,3], [1,2,3,4])
print(array)
```

   You should have gotten an error. This is because a multidimensional array rows have the same number of elements in each row. A list would let the rows have different sizes. An array will not.

   Comment out the error-provoking code before moving on.

5. You can find the dimension (number of rows), the shape (X by Y), and the size (num of elements in the whole array) via the following commands. What is printed out by each?

```
print(nums2.ndim)   2
print(nums2.shape)  (2,3)
print(nums2.size)   6
```

6. We can form arrays like using ranges like we did before hand, but now we use the “a-range” function.

```
array = np.arange(1,21)
print(array)  #Should see 1-20 in a 1 dimensional list
array = array.reshape(4,5) #This reshapes the list into a 4 by 5 array
print(array)
```

7. Write code to create a 100 by 1000 array of the numbers containing the numbers 200000-299999. Here’s a hint.

```
array = np.arange(200_000______________, 300_000______________)
array = array.reshape(100__________, 1000__________)
print(array)
```

8. And as with lists, we can look at the mean/min/etc. There are built-in np.array functions to do this so we can call functions like array.sum() and array.min() Try the following code.

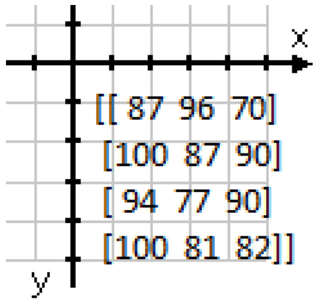
```
#create an array of 4 student’s exam1-exam 3 grades
grades = np.array([[87, 96, 70],
                   [100, 87, 90],
                   [94, 77, 90],
                   [100, 81, 82]])

print(grades)

#Notice that the below code prints the sum/min/etc of ALL the grades
#across all rows and all columns.
print(grades.sum())
print(grades.min())
print(grades.max())
print(grades.mean())
print(grades.std())
```

9. It might be the case that you just want the sum/min/etc of each row or the sum/min/etc of each column.

   Well, if you imagine that a 2D array is placed on the X-Y plane then the columns (labeled 0-2) line up with the x values and the rows (labeled 0-3) line up with the y axis. The x-axis is considered axis 0 and the y-axis is considered axis 1.



   Hence, the functions sum/min/etc let you specifigy the right axis.

```
#perform operations ON the columns (x-axis is axis 0)
print("\nstats for cols")
print(grades.sum(axis = 0))  #This gives a 1 by 3 array
print(grades.min(axis = 0))
print(grades.max(axis = 0))
print(grades.mean(axis = 0))
print(grades.std(axis = 0))
```

```
#perform operations ON the rows (y-axis is axis 1)
print("\nstats for rows:")
print(grades.sum(axis = 1))  #This gives a 1 by 4 array
print(grades.min(axis = 1))
print(grades.max(axis = 1))
print(grades.mean(axis = 1))
print(grades.std(axis = 1))
```

Why does axis 0 give you a 1 by 3 array of numbers back and axis 1 gives you a 1 by 4 array back?

0 looks at the cols, 1 looks at rows

10. And we can also slice.

```
print("\nSlicing and Dicing")
print(grades)

#Recall:  In bracket notation, we do row first, column 2nd.
print(grades[0,1])  #grade at row 0 col 1
print(grades[1]) #grades in row 1
print(grades[0:2]) #grades in first two rows (excludes row 2)
print(grades[[1,3]]) #how to get nonconsecutive rows:
                     #here you need 2 brackets because
                     #you are passing #in the list of rows (so [1,3])
                     #that you want to print
print(grades[:,0]) #gives all rows but just col 0 in each
print(grades[:,1:3]) #gives all rows but just cols 1-2 in each
print(grades[:,[0,2]]) #gives all rows ut just cols 0 and 2 in each
```

Write code to print all columns of the last two rows.

print(grades[:, [-1, -2]])

Write code to print all columns of the first and the last row.

print(grades[:, [1, -1]])

11. Should you ever need to do so, you can flatten an array into a 1D array.

What does this code print out? [87 96 70 100 87 90 94 77 90 100 81 82]

print(grades.flatten())

12. You can also gain access to the transpose of the array (so the output when you swap rows/columns. Essentially, you flip the array across its diagonal).

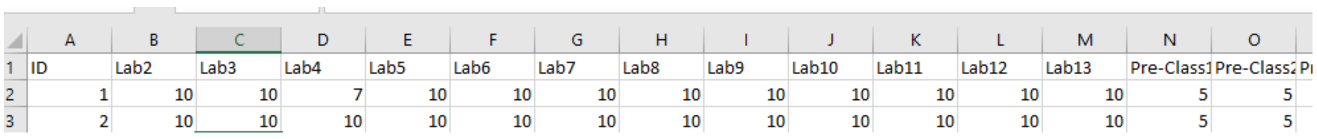
```
transpose = grades.T #This one is weird - No parentheses - The transpose
                     #is an attribute of an array, not a method.
print(transpose)     #The 4 by 3 became a 3 by 4
```

13. Now we are going to use a DataFrame which is built on arrays and are essential to data science work. DataFrame lives in the pandas library. Put this line at the top of your code.

    import pandas as pd

14. We will populate our dataframe by importing a .csv file. (Don't know what a .csv file is and want to know? Watch this: https://www.youtube.com/watch?v=n8qz0wZ8Z0c.)

    Save the finalgrades.csv in excel or notepad to see what is inside. It contains all grades from one of my previous programming classes. Below is a portion of the file when open in excel. In total there are 21 students but you can see the first two students (anonymized with ID 1 and 2) below.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	ID	Lab2	Lab3	Lab4	Lab5	Lab6	Lab7	Lab8	Lab9	Lab10	Lab11	Lab12	Lab13	Pre-Class1	Pre-Class2 Pr
2	1	10	10	7	10	10	10	10	10	10	10	10	10	5	5
3	2	10	10	10	10	10	10	10	10	10	10	10	10	5	5

    Notice that the .csv file is arranged with all lab grades followed by all pre-class work grades, then all HW grades, all exam Grades, and the Final Project grade.

    Save the finalgrades.csv (posted with this lab) at the same level as your Lab7A_Array_DataFrames.py file. For example, if your lab file is saved at C:\Users\ryank\CS215, then your csv file should be at C:\Users\ryank\CS215 as well.

    Then add this code to your .py file. This is how you read in from a csv into a DataFrame.

```
df = pd.read_csv('finalgrades.csv')
```

15. The csv file had row headers and now these row headers are associated with the data frame. The following will print the first few rows of the DataFrame and the headers of the DataFrame and the shape of the DataFrame (21 rows/students by 53 columns/grade categories).

```
print("Printing List of the Columns")
print(df.columns)

print("\nPrinting First Few Rows")
print(df.head())

print("\nPrint How Many Rows and Cols")
print(df.shape)
```

16. When we print the DataFrame information, I want to see the ID of the students (1-21) and not the row number (0-20), so I will make this happen by setting the row index information to the student ID column as follows.

```
df.index = df["ID"]
```

17. To get the element in the 5th row and 10th column of the DF (short for "DataFrame"), you may expect to do something like this: df[4][9] but this is not how it works you need to use one of these two functions.

- iloc stands for "integer location" and you can use df.iloc to get access to a specific locations using integer indexing. Try this code.

```
     print("\nPracticing Getting Data from a DF")
    x = df.iloc[4,9] #x is the element in the 5th row, column 10
                     #because we start counting at 0.
    print(x)


    labs = df.iloc[:,1:13] #labs contains ALL rows and columns 1:13,
                           #which corresponds to the lab grades in
                          #the csv file.
    print(labs)
```

- We can also use loc which lets us index by the string header names which is very helpful. Try this.

```
labs = df.loc[:,'Lab2':'Lab12'] #We start at Lab2 column and go up to
                                 #AND including the Lab12 column.
print(labs)

hwks = df.loc[:,'HW1':'HW13'] #We start at HW1 column and go up to
                               #AND including the HW13 column.
print(hwks)
```

  Be sure to notice: with iloc, to get to column 12, we need to end at column 13. However, with loc, the last column is included in the results.

18. Let's get some practice by finding which students did the best on Exam1.

    We will use the sort_values method to first sort the students in descending order. We use the "by" parameter to tell Python which column to sort by. Since we want descending, we set the ascending parameter to false. You should see that student 1, 19, and 6 were the top 3 scorers on exam 1.

```
print("\nWho did Best on the Exams")
gradesExam1 = df.sort_values(by='Exam1', ascending = False)
gradesExam1 = gradesExam1.loc[:, "Exam1"]
print(gradesExam1.head()) #Shows the first few values
```

19. Write similar code to determine who the top 3 scorers on exam 2, 3, and the Final Exam are.

    Exam 2: 1 98.4
    Exam 3: 1 97.93
    Final Exam: 19 94.6

20. Now let's compare the disributions of the exam grades. We use a bar chart to look at grade data in a previous lab. However, in that case, the possible grades were 0, 5, 10, etc. In this situation, the exam grades could be any floating point value between 0 and 100, like 94.29. So we will create a histogram, not a barchart. This just means that instead of counting how many people got a 0 or a 5 … or a 95 or a 100, we will instead count how many people got in the range 0-5, 5-10, 10-15, …, 95-100.

    Numpy makes this very easy for us. To start though, add this import which helps with plotting.

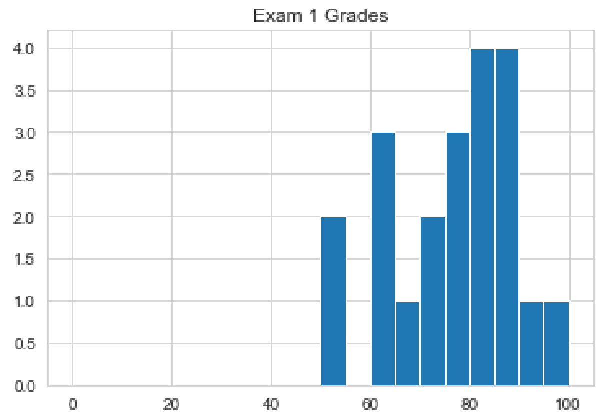
```
import matplotlib.pyplot as plt
```

    We have to tell Python to collect together 0-5, 5-10, 10-15, …, 95-100. We just do this by creating the list below. The histogram functions will know that the list 0, 5, 10, etc means categories 0-5, 5-10, etc.

```
xBins = np.arange(0,105,5)
```

    Now we actually show the histogram and some of the statistics on the exam1 values.

```
#create the histogram
ex1Histogram = gradesExam1.hist(grid = True, bins = xBins)
plt.title("Exam 1 Grades") #Set its title
plt.show() #Show the histogram
print(gradesExam1.describe())  #You have not seen this before! This
                               #shows you all the descriptive stats
                               #you would want to see on a dataframe
```



21. Now use similar code to find the histogram and descriptive stat data for exam 2 and exam 3 and the final. What does this information tell you about the distribution of the exams? Which did the students seem to do best on?

Looking through the graphs we can see that Exam 2 had the highest mean of all tests, we also see that exam 2 had a somewhat standard looking curve, hence exam 2 is the looking exam score

22. Now let’s create a new column called ExamAvg and find the average of all the exams.

```
exams = df.loc[:,'Exam1':'Exam3']  #Zoom in on just the exam cols
df["ExamAvg"] = exams.mean(axis=1)  #take the mean of the 3 exams
                                    #for each student.  Since we are
                                    #doing an average for each student,
                                    #we are focusing in on the y axis,
                                    #so axis 1.
print(df) #You’ll see a new column at the end.
```

Since there is no “ExamAvg” column, the above code creates one in the DF.

23. Now let’s create a new column called LabAvg but each lab is out of 10 ponts and we want it on a 0-100 scale.

```
labs = df.loc[:,'Lab2':'Lab12']
df["LabAvg"] =labs.mean(axis=1)*10
print(df)
```

24. Finally, you create a PreClassAvg and HWAvg column by doing the following:

- Each PreClassX Column is out of 5 points, so you can just find the average and multiple by 20 to put it on a 0-100 scale.

- Each HW grade is out of a different number of points. The total possible points is 190 so sum up all the values in the HW columns for each student. Then divide by 190 and multiple by 100 to get it onto a 0-100 scale.

When done, you should see these values.

```
    ID  FinalProject  LabAvg  PreClassAvg   HWAvg

ID
1    1        97.69   97.27      100.00  100.00
2    2        90.77  100.00      100.00   97.89
3    3        72.31   99.09      100.00   95.26
4    4        81.54   99.09      100.00   98.42
5    5        93.85   90.91      100.00   97.89
6    6        70.00   99.09      100.00   98.95
7    7        80.00  100.00      100.00   97.89
8    8        88.46  100.00      100.00   96.32
```

25. Finally let’s calculate the final grade for each student. Students’ grades were calculated as such:
    - 8% labs
    - 2% preclass videos
    - 15% for the final exam
    - 15% for the final project
    - 20% for HWs
    - 45% for the exam 1-3 average

    We can do this via the code below. If you get an error, then keep the code on one long line.

```
df["FinalGrade"]  = df.loc[:,"LabAvg"] * 0.08 + df.loc[:,"PreClassAvg"] *
0.02 + df.loc[:,"HWAvg"] * 0.15 + df.loc[:,"ExamAvg"] * 0.45 + df.loc[:,
"FinalProject"] * 0.15 + df.loc[:, "FinalExam"] * 0.15
```

SUBMISSION INFO

TO GET CREDIT FOR THIS LAB, UPLOAD THE FOLLOWING TO THE SUBMISSION AREA.

- Your pdf doc with your summaries/answers
- Your code