Lab 02 - numpy

Submission: `lab02.ipynb`, pushed to your Git repo on `master`.
Points: 10
Due: Wednesday, January 23, 9:59am

Objectives

- `numpy`. More `numpy`.

Introduction

You are learning about `numpy`. And now, we'll learn more. And, that is what we're going to do, yes?

Create a **`lab02.ipynb`** file. Create your header cell, then your first cell to set up your imports. Remember, answer each question with [P] with Python code that produces your answer. No answer should be hard coded! However, if the answer is going to require many lines of output, then please do NOT print it. There is a good chance a later question will ask you something additional about that object.

For example, if a line states, "Create a 2x3 matrix of zeros, single precision floating point format, stored as X", then your answer should be something like:

```
X = np.zeros((2,3), dtype=np.float32)
print(X)
```

However, if the line states, "Create a 10000x100 matrix", then please don't show the entire array!

Exercises

Some of you did not fill out your **`README.md`**. Please do that and push that file to your remote repo.

1) [P] Create an 52500 x 75 matrix of zeros, stored as `X`. Then print out the *shape* of the matrix, the base data type, the total size of the array in bytes (as an integer), and the total size in megabytes with 3 places of significance.

2) [P] Resize X to have the same number of elements, but with 100 rows. Show the shape. Show the number of bytes (it should be the same as the previous answer)

3) [P] Redo #1, but use a base datatype of 16-bit integers.

You should have a reduction in memory by a factor of 4. There are times when this type of change is simple, but highly effective in speeding up your algorithm. Integer computations will always outperform floating point computations. Use the simplest data type available that can accurately store your data.

4) [P] How many dimensions does X have? Answer using the appropriate property of `np.ndarray` objects.

5) [P] Enter the following list in your cell:

```
str_nums = ['2.14', '-9.3', '42']
```

Convert this to a `numpy` array named `X`. What is the base type?

Then, convert `X` to an array of single precision floating point numbers. (HINT: use `astype`)

Let's assume you have two week's worth of quiz scores. Quizzes are out of 10 points, and are given every day. Enter the following in your cell:

```
days = ["Mon","Tue","Wed","Thu","Fri"]
scores_1 = [9.5, 8.75, 8, 10, 7.75]
scores_2 = [9, 8, 10, 8.75, 7.25]
```

The array `scores_1` represents quiz scores from week 1, and `scores_2` is week 2. The `days` array will be used for data selection purposes.

6) [P] Create a `numpy` array called `scores`, that has `scores_1` as the first row and `scores_2` as the second row, using `np.concatenate`. Then, change `days` into a `np.array` from the list days. Verify the shape of `scores` and `days`.

7) [P] Repeat the previous problem, but repeat it with `np.vstack`. The array should be identical.

For the next several questions, you are going to use advanced data selection techniques using the days array to assist with looking up data. For example, if you are asked to reveal the scores from only Friday, then you must answer in the following way:

```
scores[:, days == "Fri"]
```

8) [M] Compare the result of the expression `days == "Fri"` if the variable `days` was a Python list, vs. `days` being a `numpy` array. What is the difference? In general, how does `numpy` deal with standard comparison operators?

9) [P] Select the scores that fell on Monday

10) [P] Select all of the scores that are NOT on Monday (Hint – look up the ~ operator)

11) [P] Select the scores that were on Tuesday or Thursday

12) [P] Show the minimum and maximum scores for the entire array of scores

13) [P] Show the minimum scores for each week as a new array with the same dimensions (hint – use the `keepdims` parameter). Your result should be something like:

```
array([[7.75],
       [7.25]])
```

14) [P] Report the day that the maximum score occurred each week. (HINT: use `argmax` and use that result to index `days`.)

15) [P] Report the mean of the scores of each week

16) [P] Suppose the lowest score was dropped from each week. Report the mean of each week, but without the minimum score for that week.

17) [P] Convert the scores to fall on a scale from 0-100 instead of 0-10.

Copy and paste this code into a cell:

```
np.random.seed(1234)
X = np.random.randint(1,100,50).reshape((10,5))
X
```

You may use standard selection techniques with integers and slicing for the following exercises

18) [P] Select the first row of X

19) [P] Select the last column of X

20) [P] Select the first AND last column of X

21) [P] Select every other row of X

22) [P] Show the transpose of X, but don't change X itself

23) [P] Select the first column of X and set the result to Y.

24) [P] Increment the first value of Y, then show the corresponding value of X. Did both values in X and Y change?

25) [P] Repeat exercise 23, but ensure that Y is assigned a **copy** of the selected data. Increment the first value of Y and ensure that the corresponding value of X did not change.

26) [P] Create an array that contains the sequence of numbers 0, 0.1, 0.2, … 9.8, 9.9 using `arange`, as a 10x10 matrix, stored as `X`.

27) [P] Set the RNG seed to 1234. Then create an array X of 100 uniformly distributed numbers, with all values between 1.0 and 10.0. Then, show the mean, the median, the minimum and maximum values of X.

28) [M] Define what is meant by a **normal distribution**. What are the parameters of a normal distribution?

29) [M] In simple terms, using a normal distribution, what does the Law of Large Numbers tell us?

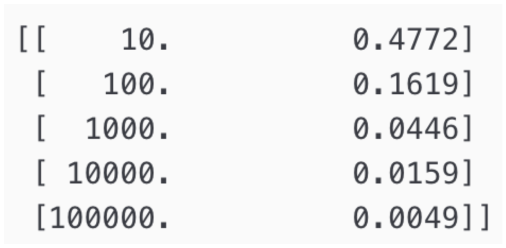
30) [P] Write a function called `test_normal_dist`. The purpose of this function is to evaluate the law of large numbers. It should have four parameters:

    `mu` = mean of distribution
    `sd` = standard deviation
    `vec_length` = length of the vector to generate randomly from a normal distribution, with `mu` and `sd` as parameters
    `num_trials` = number of times to repeat the experiment

    The function should behave as follows. First, initialize the seed value to 1234, before your loop. Then, loop for `num_trials`, generating `vec_length` numbers from a normal distribution. Compute the mean of that vector, then compute its deviation (absolute value of the observed mean - expected mean). This should be repeated for all trials, and then return the *average* deviation over all trials.

31) [P] Use `test_normal_dist` to obtain the deviation for vector lengths of 10, 100, 1000, 10000, and 100000. Use a fixed number of trials of 100 for each experiment. Report the results as a `numpy` array with two dimensions. the first being the vector length, and the second being the average deviation resulting from your

`test_normal_dist` function. Your resulting array should look like:



```
[[    10.          0.4772]
 [   100.          0.1619]
 [  1000.          0.0446]
 [ 10000.          0.0159]
 [100000.          0.0049]]
```

In other words, your results should confirm the Law of Large Numbers.

Deliverables

Commit and push `lab02.ipynb`. Be sure you have every cell run, and output generated. Verify that your file is pushed properly on Gitlab.