Lab 03
NumPy

A. Multiple Choice (10 points, 5 points each question)

1. What is/are the advantage(s) of NumPy arrays over Python lists?

   (a) Operations like addition and multiplication are faster.
   (b) It has better support for mathematical operations.
   (c) They consume less memory.
   (d) All of the above.

2. Suppose the statement `A = np.matrix([10, 20])` is executed, what is the dimension of matrix `A`?

   (a) 0 (b) 1 (c) 2

B. A Piece of Cake (28 points, 4 points each question)

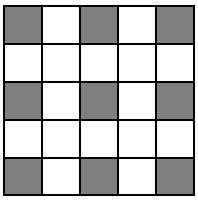
1. Without using loops, create the following 4 variables in Lab03_B.py.

$$A = [5 \quad 4.8 \quad 4.6 \quad \cdots \quad -4.8 \quad -5] \qquad B = [10^{0} \quad 10^{0.01} \quad 10^{0.02} \quad \cdots \quad 10^{0.99} \quad 10^{1}]$$

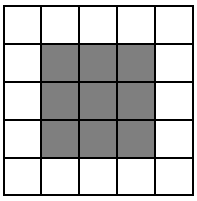
$$C = \begin{bmatrix} 1 & 11 & \cdots & 91 \\ 2 & 12 & \ddots & 92 \\ \vdots & \vdots & \ddots & \vdots \\ 10 & 20 & \cdots & 100 \end{bmatrix} \qquad D = \begin{bmatrix} 1 & 2 & \cdots & 12 \\ 2 & 4 & \ddots & 24 \\ \vdots & \vdots & \ddots & \vdots \\ 12 & 24 & \cdots & 144 \end{bmatrix}$$

2. Suppose an M × M matrix (M ≥ 3) `X` is given as an argument, implement functions which extract values from matrix `X` to create the following matrixes as return values. Complete the function template in Lab04_B.py and do NOT use loops inside your function.

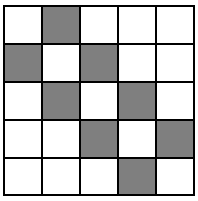
function `E`	return matrix composed of all values in odd columns AND odd rows of `X`.
function `F`	return matrix composed of all entries of `X`, except for the outside rows and columns.
function `G`	return matrix composed of diagonals surrounding the middle diagonal of matrix `X`.



`E`



`F`



`G`

C. Programming Exercise (30 points, 6 points each question)

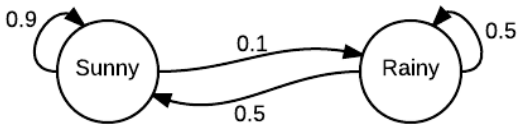
You will implement 5 functions in this part. Please find the function template in Lab04_C.py. For problems 1, 2 and 3, loops are NOT allowed in the function. For problems 4 and 5, be sure to test your function with various inputs.

1. Write a function `swap_rows(x, r1, r2)` that swap two given rows in an 2d array `x`. Do NOT declare any other variable in the function.
2. Write a function `most_value(x)` that find the most frequent value in an 1d integer array `x`. You can assume that there will be only one most frequent value.
3. Write a function `top_n(x, n)` that find the `n` largest values of an 1d array `x`.
4. Write a function `pythagorean(x)` to calculate the hypotenuse of a right-angled triangle, for given the other two sides of the triangle. The input argument is either a 1d or a 2d array of two columns `[a b]`, each of which is a side of the triangle. The function should return a column vector that contains positive values each of which satisfies the Pythagorean Theorem, $c_i = \sqrt{a_i^2 + b_i^2}$, for the corresponding i^{th} row of `[a b]`.
   The function needs to check if the input matrix has exactly two columns. If it does not, the function raise an error.
5. Write a function that takes inputs `(v,a,b,c)`, where `v` is a 1d array, and `a`, `b`, and `c` are all integar. The function replaces every element of `v` that is equal to a with two entries `[b c]`. For example, the command `x = replace_me([1 2 3],2,4,5)` makes `x` equal to `[1 4 5 3]`. If `c` is omitted, it replaces occurrences of a with two copies of `b`, (i.e., `[b b]`). If b is also omitted, it replaces each a with two zeros, (i.e., `[0 0]`)

D. Markov Chains (32 points, 6 / 6 / 4 / 2 / 4 / 6 / 4)

In the following problems, you will be learning about and implementing Markov Chains. A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. Let's check out a simple example directly to understand the concept: (Example from Wikipedia: https://en.wikipedia.org/wiki/Markov_chain)
Figure 1a shows a simple weather model which assume the weather conform to the following rules:

- It is either Sunny or Rainy a day.
- If it is Sunny today, tomorrow will be Sunny with probability 0.9, and Rainy with probability 0.1.
- If it is Rainy today, tomorrow will be sunny or Rainy with equal probability.



CURRENT \ NEXT STATE	Sunny	Rainy
Sunny	0.9	0.1
Rainy	0.5	0.5

Fig. 1 (a) **state diagram** of a simple weather model and (b) it's **transition matrix**

This weather can be modeled with a Markov chain since the weather tomorrow depends solely on the weather today, not yesterday or any other time in the past.

The same information can be represented by a **transition matrix** (Fig. 1b) from time n to time n+1. Every state in the state space is included once as a row and again as a column, and each cell in the matrix tells you the probability of transitioning from its row's state to its column's state. Notice that the rows of a transition matrix sum to 1.

One statistical property that could be calculated is the expected percentage, over a long period, of the days that will be Rainy. To do so, we first assume the weather on day 0 (today) is Sunny. This is represented by a vector in which the "sunny" entry is 100%, and the "rainy" entry is 0%:

$$x^{(0)} = [1 \quad 0]$$

The weather on day 1 (tomorrow) can be predicted by:

$$x^{(1)} = x^{(0)}P = [1 \quad 0]\begin{bmatrix}0.9 & 0.1\\ 0.5 & 0.5\end{bmatrix} = [0.9 \quad 0.1]$$

Thus, there is a 90% chance that day 1 will also be Sunny. The weather on day 2 (the day after tomorrow) can be predicted in the same way:

$$x^{(2)} = x^{(1)}P = [0.9 \quad 0.1]\begin{bmatrix}0.9 & 0.1\\ 0.5 & 0.5\end{bmatrix} = [0.86 \quad 0.14]$$

Finally, the weather on day k represented with the initial state will be:

$$x^{(n)} = x^{(0)}P^k$$

Now that you have understood the concept of Markov Chains (if not, feel free to find other tutorials on the internet, there are plenty of them), let's start to implement the Markov Chains with numpy.

1. Write a function `transition_matrix(n)` to return the transition matrix for the following Markov chain, given the **number of states `n`** as an argument. The rule for the Markov chain is as follows. Again, NO loops are allowed for this question.

   If current state is i *and* $i < n/2$

   With probability 0.60, it goes to state $i = i + 1$

   With probability 0.35, it stays at state $i = i - 1$

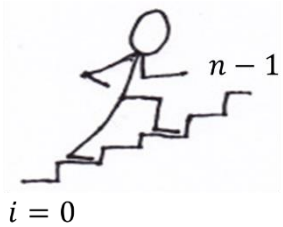
   With probability 0.05, it will fall back to state $i = 0$

   If current state is i *and* $i \geq n/2$

   With probability 0.50, it goes to state $i = i + 1$

   With probability 0.40, it will fall back to state $i = i - 1$

   With probability 0.10, it will fall back to state $i = 0$

   If current state is i *and* $i = 0$

   Since it cannot get any lower, it will stay at state $i = 0$ *with probability 0.40.*

   If current state is i *and* $i = n - 1$

   Since it cannot go higher, it will stay at state $i = n - 1$ *with probability 0.50.*

2. Now that the transition matrix is defined, we want to know what are the probabilities of being in each state after running the Markov chain for a given steps. Implement a function `propagate(x0, P, k)` which given an initial distribution `x0`, a transition matrix `P` and number of steps `k`, it will return the distribution after `k` steps.

3. Evaluate the probabilities at the end of 8 steps:
   1) Create a transition matrix `P = transition_matrix(n=10)`.
   2) Create a vector `x0` which represents the initial state that you are in state 0 with probability 1.
   3) Calculate the probability distribution using `x8 = propagate(x0, P, k=8)`.
   4) Plot the probability distribution using `plot_distribution(x8)`.

   A figure Lab04_D3.png will be saved automatically to your current directory. Look at the figure and answer the following questions:
   (a) Which state has the highest probability in step 8?
   (b) What is the probability to reach the final state ($i = 9$) in step 8?
   (c) Make sure to include the figure in your report.

4. How many steps does it take for the probability of being in the final state ($i = 9$) to be at least 1%?

5. Try initialing `x0` with random numbers and keep $\text{sum}\left(x^{(0)}\right) = 1$, will the probability distribution be different? Give an explanation.

6. So far, we have seen the over all distribution. However, we are also interested in computing single sample evolutions of the Markov chains. Implement the function `create_sample(s0, P, k)` which given an initial state `s0`, a transition matrix `P` and number of steps `k`. It will return a different state sequence every call. Here is one of the samples of `create_sample(s0=0, P=P, k=20)`:

   ```
   [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 9, 8, 9]
   ```

7. Last, we want to verify if our sampling implementation is consistent with our implementation of computing the probability distribution:
   1) Generate 1000 independent samples by calling `create_sample(s0=0, P=P, k=8)` for 1000 times.
   2) Store the state of last step (`sample[-1]`) of each sample in a list `last_steps`.
   3) Plot the histogram using `plot_histogram(last_steps)`.

   A figure Lab04_D7.png will be saved automatically to your current directory. Look at the figure and answer the following questions:
   (a) Which state has the highest probability in step 8?
   (b) What do you observe from the two figures? Does the result meet your expectation?
   (c) Make sure to include the figure in your report.