Midterm 2

This test is open-book. You may use your scripts, textbook, notes, homework assignments, Python. You may NOT use anything else including search engines, on-line chatting, phones, etc. You may NOT discuss with anyone.

Problem 1 (15 points)

Script: P1.py
Data: P1.txt
Type your name in the script to agree with the UNC Honor Code Pledge.

Write a function called `countVal(x,minVal,maxVal)` that takes an 1D Numpy array `x` and returns the number of elements that satisfy the following requirements:

1. Elements are multiples of 5;
2. Elements are in range [`minVal`, `maxVal`], inclusive.

Test your function with the data in P1.txt. The testing code is available in the template. Do NOT modify the test code. Your job is to complete the function definition.

Problem 2 (20 points)

Script: P2.py
Type your name in the script to agree with the UNC Honor Code Pledge.

In 1748, Euler published *Introductio in Analysin infinitorum* and showed that the natural base *e* can be defined by the limit:

$$e = \lim_{k\to\infty}\left(1+\frac{1}{k}\right)^k$$

or by the infinite series:

$$e = \sum_{k=0}^{\infty}\frac{1}{k!}$$

We can approximate *e* by the summation using the first *n+1* terms of the series

$$\hat{e} \approx \sum_{k=0}^{n}\frac{1}{k!}$$

To control the approximation accuracy, one can choose n to be the first number when $\frac{1}{n!}$ becomes less than a given convergence tolerance ϵ.

Write a function called `estimate_e` that takes a prescribed convergence tolerance ϵ and returns

1. the numerical approximation of e: $\hat{e}$ and
2. the number of terms used for the approximation convergence: *n+1*

You can use function `math.factorial()` in math module to calculate the factorials.

Problem 3 (20 points)

Script: P3.py
Type your name in the script to agree with the UNC Honor Code Pledge.

Write a function called `remove_duplicates` that takes a list of immutable elements and returns a new list with only the unique elements from the original.

Hint: The unique elements in the output list do NOT have to be in the same order as in the input list. You may find the dictionary method `dict.key()` useful where it returns a list of dictionary's keys.

Problem 4 (20 points)

Script: P4.py
Type your name in the script to agree with the UNC Honor Code Pledge.

A simple letter encryption involves cyclically shift each letter in a string. Cyclically shifting a letter means to shift it through the alphabet, wrapping around to the beginning or to the end if necessary. For example, '*A*' shifted to the **right** by 3 is '*D*', '*d*' shifted to the **left** by 3 is '*a*', and '*A*' shifted to the **left** by 3 is '*X*'.

Write a function called `decode_string(s, shift)` that takes an encrypted string `s` and an integer of the shifting amount `shift` and return a decoded string. To decode, shift the encrypted characters to the **left** by the given amount.

For example, "*jolly*" shifted to the left by 7 is "*cheer*", "'*Khoor Zruog!*'" shifted to the left by 3 is "*Hello World!*".

Hint: You may want to implement a function `decode_char(char, shift)` to shift a letter to the left by the given amount. Call function `decode_char` in your definition of `decode_string`.

Note: Only upper- and lower-case letters will be shifted. Other characters, e.g., spaces, periods, commas, exclamation marks, etc. will remain the same.

Useful functions: `ord()`, `chr()`, `str.isupper()`, `str.islower()`, `str.isalpha()`. You may also find operator '+' on strings (string concatenation) useful.

Problem 5 (25 points)

Script: P5.py
Data: P5.png and P5.txt
Type your name in the script to agree with the UNC Honor Code Pledge.

Image denoising recovers a digital image contaminated by noise. A common image denoising method is to convolve the corrupt image with a smoothing filter (aka smoothing kernel). One of the widely used smoothing filter is weighted averaging filter.

In this problem, you will implement a function called `applyFilter(data, kernel)` to apply a 5-by-5 smoothing `kernel` (i.e., a 5-by-5 Numpy array) to a 2D image `data` (also a 2D Numpy array). The return of the function is a 2D Numpy array with the same shape as `data`.

Assume variable `data` is M-by-N, the element of the resulting array `result[i,j]` `(0<=i<M, 0<=j<N)` is defined by

$$result[i,j] = \begin{cases} 0, & if\ i \notin [2, M-3]\ or\ j \notin [2, N-3] \\ \sum_{u=i-2}^{i+2} \sum_{v=j-2}^{j+2} data[u,v] * kernel[u,v], & otherwise \end{cases}$$

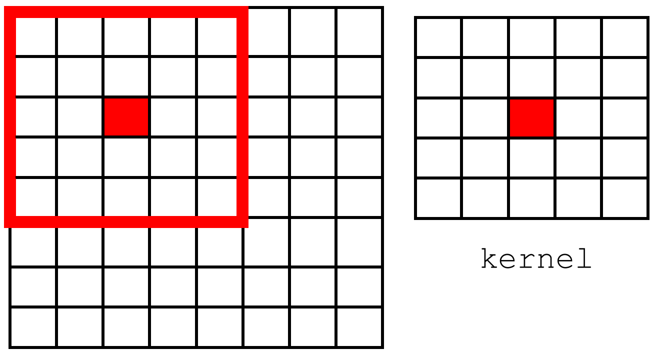
The pixels on the boundary (2-pixel wide) are set to be 0 in the resulting array. The interior pixels of the resulting array are set to be the summation of **element-wise** product of the neighborhood (5-by-5) and the kernel centered at that pixel.

Assume the 2D array `data` is 8-by-8, then the resulting array `result` is also 8-by-8. In `result`, pixels on the boundary (2-pixel wide) are set to be 0. For each interior pixel in `result`, the pixel value can be calculated by a weighted average of its 5-by-5 neighbors. The weights for a pixel's neighbors are given by the kernel centered at that pixel.

For example, to compute the pixel value at the red pixel, we use the 5-by-5 neighbors surrounding that pixel. The contribution of each neighbor is specified by the corresponding values in the kernel. The resulting pixel value at the red pixel is the weighted average of its neighboring pixel values.

You will use nested loops to iterate through all the interior pixels. Note the boundary is defined as 2-pixel wide in this problem. For each interior pixel, you will calculate the weighted average of its neighbors and store the result in the corresponding pixel in the resulting array.

Your function will be tested on a noisy image. The expected result is a smooth image.



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	🟥				0	0
0	0					0	0
0	0					0	0
0	0					0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

result