In [ ]:
```
import numpy as np
import cv2
import time
import matplotlib.pyplot as plt
```

1. Write a program to load a .csv file as a NumPy 1-D array. Find the maximum and minimum elements in the array. Hint: For the data, use the .csv file “Book1.csv”

Output:

- Maximum element: 100.0
- Minimum element: 1.0

In [ ]:
```
# Load the data from the .csv file
data = np.loadtxt('book1.csv', delimiter='\t', dtype=np.float32, skiprows=1)[:, 1]
# The [:, 1] at the end of the line is a slicing operation that selects all rows (:) and the second column (1) of the data array.
# The result is a 1D array containing the data from the second column of the file.

# Find the maximum and minimum elements in the array
max_element = np.max(data)
min_element = np.min(data)
print('Maximum element:', max_element)
print('Minimum element:', min_element)
```

2. For the Numpy 1-D array as obtained in Q.1, sort the elements in ascending order.

In [ ]:
```
sorted_data = np.sort(data)
# print('Sorted data:', sorted_data)
```

3. For the sorted Numpy 1-D array as obtained in Q.2, reverse the array and print.

In [ ]:
```
reversed_data = sorted_data[::-1]
print('Reversed data:\n', reversed_data)
```

4. Write a program to load three .csv files (Book1.csv, Book2.csv, and Book3.csv) as a list of Numpy 1-D arrays. Print the means of all arrays as a list.

Output:

- Means: [48.566, 51.088444, 513.326]

In [ ]:
```
data1 = np.loadtxt('book1.csv', delimiter='\t', dtype=np.float32, skiprows=1)[:, 1]
data2 = np.loadtxt('book2.csv', delimiter='\t', dtype=np.float32, skiprows=1)[:, 1]
```

```
data3 = np.loadtxt('book3.csv', delimiter='\t', dtype=np.float32, skiprows=1)[:, 1]
# Print the means of all arrays as a list
means = [np.mean(data1), np.mean(data2), np.mean(data3)]
print('Means:', means)
```

5. Write a program to read an image, store the image in NumPy 3-D array. For the image, consider a.PNG. Display the image. Let the image stored in the NumPy array be X.

In [ ]:
```
image = cv2.imread('a.png', cv2.IMREAD_COLOR)
# This line reads the image file 'a.png' in color mode and stores it in the variable image.
# The image data is stored as a 3D NumPy array, with dimensions corresponding to the height, width, and color channels of the image.

cv2.imshow('Image', image)
# This line creates a window named 'Image' and displays the image in it.

cv2.waitKey(0)
# This line waits for a key press in the 'Image' window.
# The argument 0 means it will wait indefinitely until a key is pressed.

cv2.destroyAllWindows()
```

6. Write a program to convert a color image (say a.PNG) into a grescale image. Let the greysacle image stored in the Numpy 2-D array be X. Display the grayscale iamge on the screen

In [ ]:
```
# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Display the grayscale image
cv2.imshow('Grayscale Image', gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

7. Let Y be the transpose matrix of X. Write a program to obtain Z = X×Y.

In [ ]:
```
X = gray_image.copy()
Y = X.T
# Obtain Z
start_time_with = time.time()
Z = np.dot(X, Y)
# This line performs matrix multiplication of X and Y and assigns the result to Z.
# The np.dot function computes the dot product of two arrays. For 2-D arrays, it is equivalent to matrix multiplication.
end_time_with = time.time()
```

8. For the problem in Q. 7, write your program without using NumPy library. Compare the computation times doing the same with NumPy and basic programming in Python

- Time taken with numpy: 0.9223132133483887: This is the time taken to perform matrix multiplication using NumPy's built-in functionality. It's significantly faster because NumPy operations are implemented in C, which is a lower-level language than Python and can therefore execute operations more quickly.
- Time taken without numpy: 71.29215264320374: This is the time taken to perform matrix multiplication without using NumPy, likely using nested for loops in Python. This is significantly slower because Python is an interpreted language, and operations like loops are much slower in Python than in lower-level languages like C.
- The time taken for matrix multiplication without using NumPy can vary significantly from machine to machine. This is because the performance is dependent on several factors, including:
    1. Processor Speed: Faster processors can execute operations more quickly.
    2. Memory: More available memory can allow for larger matrices to be stored and manipulated in memory, which can improve performance.
    3. Python Interpreter: Different Python interpreters (like CPython, PyPy, etc.) can have different performance characteristics.
    4. System Load: If the system is running other resource-intensive tasks at the same time, it can slow down the matrix multiplication.

In contrast, NumPy operations are highly optimized and use low-level languages (like C) to perform operations, which can result in more consistent performance across different machines. However, even with NumPy, the performance can still vary based on the factors listed above.

Output:

- Time taken with numpy: 0.39718008041381836
- Time taken without numpy: 71.29215264320374

In [ ]:
```
X2 = X.tolist()
Y2 = Y.tolist()

start_time_without = time.time()

Z = [[sum(a*b for a, b in zip(X2_row, Y2_col)) for Y2_col in zip(*Y2)] for X2_row in X2]
# This performs matrix multiplication of X2 and Y2 and assigns the result to Z.
# It does by iterating over each row in X2 and each column in Y2, multiplying the corresponding elements together, and summing the results.
# The zip function is used to pair up the corresponding elements from X2_row and Y2_col.

end_time_without = time.time()

print('Time taken with numpy:', end_time_with - start_time_with)
print('Time taken without numpy:', end_time_without - start_time_without)
```

9. Plot the pixel intensity histogram of the grescale image stored in X.

In [ ]:
```
fig = plt.figure("Pixel Intensity Histogram")
plt.hist(gray_image.ravel(), 256, [0, 256])
fig.set_size_inches(10, 5)
# This line creates a histogram. gray_image.ravel() flattens the image into a 1D array.
# 256 is the number of bins in the histogram, and [0, 256] is the range of values.
```

```
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.title('Pixel Intensity Histogram')
plt.show()
```

10. Create a black rectangle at the position [(40,100) top left, (70, 200) bottom right] in the grayscale image. Display the image

In [ ]:
```
rectangle_image = gray_image.copy()
cv2.rectangle(rectangle_image, (40, 100), (70, 200), 0, -1)
# The rectangle's top left corner is at (40, 100) and its bottom right corner is at (70, 200).
# The color of the rectangle is 0 (black in grayscale), and the thickness is -1, which means the rectangle is filled.
cv2.imshow('Rectangle Image', rectangle_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

11. Using the grayscale image stored in X, transform it into the binarized image with thresholds: [50, 70, 100, 150]. Let the binarized images are stored in Z50, Z70, Z100, and Z150, respectively.

- So, this code creates a 2x2 plot of binarized images, each with a different threshold, and displays the plot. The binarized images are created by setting all pixels in the grayscale image that are above the threshold to white, and all other pixels to black.

In [ ]:
```
# Thresholds
thresholds = [50, 70, 100, 150]
# Create a 2x2 plot
fig, axs = plt.subplots(2, 2)
fig.set_size_inches(10, 10)
fig.suptitle('Binarized Images with Thresholds')

# Transform the grayscale image into the binarized image with thresholds

# store in Z50, Z70, Z100, and Z150,respectively
# create 2d arrays Z50, Z70, Z100, Z150
Z50 = np.where(gray_image > thresholds[0], 255, 0)
Z70 = np.where(gray_image > thresholds[1], 255, 0)
Z100 = np.where(gray_image > thresholds[2], 255, 0)
Z150 = np.where(gray_image > thresholds[3], 255, 0)

# Display the binarized images with thresholds
axs[0, 0].imshow(Z50, cmap='gray')
axs[0, 0].set_title('Threshold 50')

axs[0, 1].imshow(Z70, cmap='gray')
axs[0, 1].set_title('Threshold 70')

axs[1, 0].imshow(Z100, cmap='gray')
```

```
axs[1, 0].set_title('Threshold 100')

axs[1, 1].imshow(Z150, cmap='gray')
axs[1, 1].set_title('Threshold 150')

for ax in axs.flat:
    ax.axis('off')

plt.show()
```

12. Consider the color image stored in a.png. Create a filter of [[-1,-1,-1][0,0,0][1,1,1]], and multiply this filter to each pixel value in the image. Display the image after filtering.

- This particular filter is a simple edge detection filter. The filter used here is a simple edge detection filter, which will highlight the horizontal edges in the image.

In [ ]:
```
filter = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])
filtered_image = cv2.filter2D(image, -1, filter)
# This line applies the filter to the image. The cv2.filter2D function convolves the image with the filter.
# The -1 argument means the output image will have the same depth as the input image.

# Display the image after filtering
cv2.imshow('Filtered Image', filtered_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```