Practical Assignment

Dr. Shrihari A

CSE, JIT

Python Basics

Q1: Write a program to find primes in range 2 to 100

Q2: Write a program to reverse a string: **String = 'Data Science'**

Q3: What does the following code output?

```
pp = str(int(7 + 8.0))
print(pp + " happy".upper() + "!")
```

Q4: Define a function to check whether a number is in a range (1000,10000) or not

OOPs & Datatypes in Python

Q1: You are given an initial list [8, 9, 10]. You need to create and initialize a class named "list_operations" and perform a series of operations on this list as class methods. This exercise will help you understand how to work with Python lists, including modifying and removing elements, and sorting and extending lists.

Let's break down the problem:

Setting the second entry: You must change the value at index 1 of the list to 17.

Adding elements to the end: You need to append the values 4, 5, and 6 to the end of the list.

Removing the first entry: You must delete the element at index zero from the list.

Sorting the list: You need to arrange the elements of the list in ascending order.

Doubling the list: You need to create a new list containing all the original list elements twice.

Inserting an element: You need to insert the value 25 at index 3 of the list.

Q2: You are tasked with creating a program that generates a random symmetric matrix. A symmetric matrix is a square matrix where the elements below the main diagonal are mirror images of the elements above the diagonal. This matrix type has important applications in various fields, including linear algebra, statistics, and machine learning.

Let's break down the problem:

Random Number Generation: We need a way to generate random numbers for the matrix elements. We can use the random module in Python.

Matrix Creation: We must create a square matrix of a specified size.

Symmetry: We need to ensure that the matrix is symmetric. This means that the element at position (i, j) should equal that at position (j, i).

Q3: You are tasked with generating a set of 100 random numbers. Using the bubble sort algorithm, you aim to find this set's median and median absolute deviation (MAD).

Let's break down the problem:

Random Number Generation: We need to generate 100 random numbers.

Bubble Sort: We must sort the generated numbers using the bubble sort algorithm.

Median Calculation: We need to find the median of the sorted set.

Median Absolute Deviation (MAD) Calculation: We need to calculate the MAD, which measures the variability of the data around the median.

Q4: You're tasked with creating a program that generates a random password. The password should be eight characters long and include a mix of lowercase letters, uppercase letters, numbers, and special characters. To make it even more interesting, your program should also classify the generated password as "Easy," "Medium," or "Strong" based on its complexity.

Let's break down the problem:

Randomness: We'll need a way to generate random characters.

Password Length: We must ensure the password is exactly eight characters long.

Character Types: We need to include lowercase, uppercase, numbers, and special characters.

Password Strength: We'll define criteria to classify the password as Easy, Medium, or Strong.

Advanced Concepts in Python

Q1: You're building a simple linear regression model using NumPy. You have a dataset with two features (X1 and X2) and a target variable (y). The data is represented as follows:

```
X = np.array([[1, 2, 1], [2, 4, 1], [3, 6, 1], [4, 8, 1]]) # Added a column of ones for the intercept
y = np.array([3, 7, 9, 13])
```

Where the last column of X is a column of ones, representing the intercept term.

Using NumPy, calculate the coefficients (weights) of the linear regression model using the normal equation:

$$\theta = (X^T X)^{-1} X^T y$$

Where:

- θ represents the vector of coefficients (including the intercept).
- X^T is the transpose of the feature matrix X.
- $(X^T X)^{-1}$ is the inverse of the matrix $X^T X$.
- y is the target variable vector.

Q2: You're working on a recommendation system. You have a user-item matrix where rows represent users, columns represent items, and the values represent user ratings for items (e.g., on a scale of 1 to 5). However, the matrix is sparse, meaning most of the entries are missing (represented by NaN or zero).

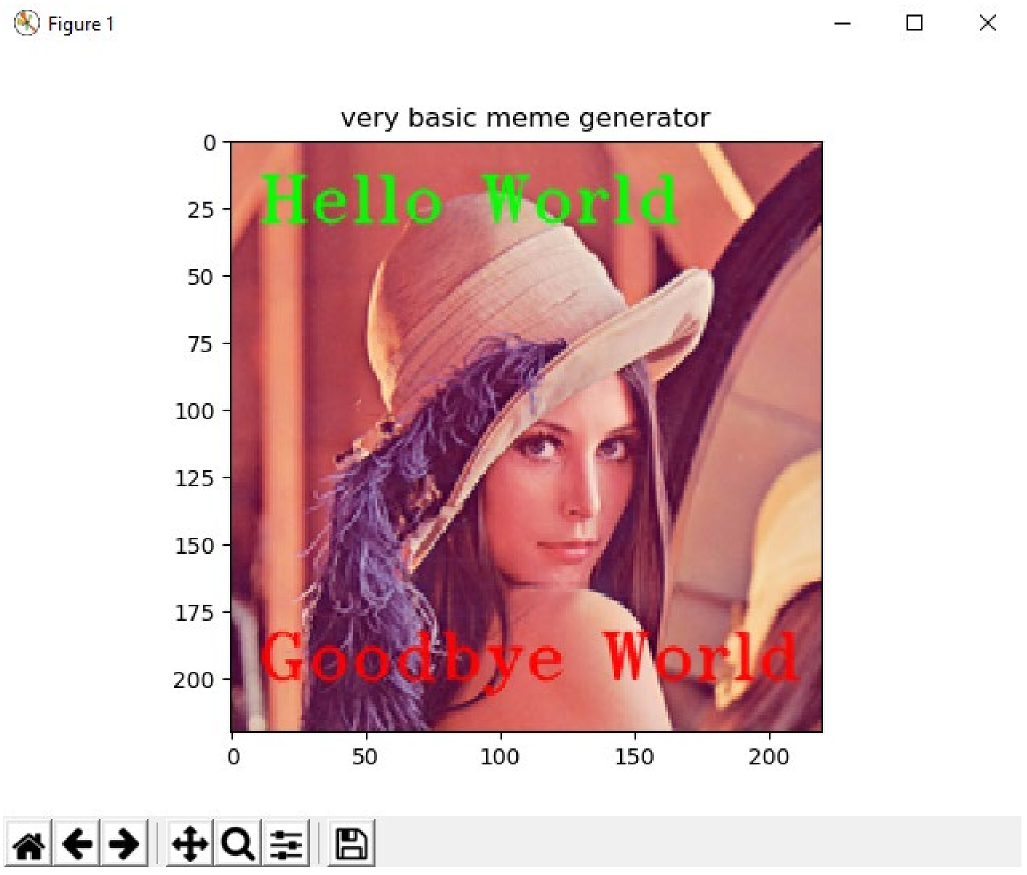
```
import numpy as np

# Sample user-item matrix (rows: users, columns: items)
ratings_matrix = np.array([
    [5, 0, 3, 0, 2],
    [0, 4, 0, 0, 1],
    [3, 0, 0, 5, 0],
    [0, 2, 4, 0, 0],
])
```

Your task is to use **Singular Value Decomposition (SVD)** with NumPy to reduce the dimensionality of this matrix and predict the missing ratings. Specifically:

1. Perform SVD on the `ratings_matrix`.
2. Reduce the dimensionality by keeping only the top *k* singular values and corresponding singular vectors (choose *k* = 2 for this example).
3. Reconstruct the matrix using the reduced SVD components.

Predict the missing ratings by filling in the zeros of the original matrix with the corresponding values from the reconstructed matrix.

Q3: Try to draw this very simple meme generator using the "lenna.png" image as background:
[Hint: Study the "OpenCV" Python library and refer to the "cv2.putText" documentation]



Q4: You're working on image processing, and you've been given a grayscale image represented as a NumPy array. The image has dimensions of 28x28 pixels (like a digit from the MNIST dataset). Your task is to perform a **Principal Component Analysis (PCA)** on the image to reduce its dimensionality while preserving the essential features.

Here's the scenario:

```
import numpy as np

# Simulate a 28x28 grayscale image (replace with your actual image data)
image = np.random.rand(28, 28) * 255  # Generate random pixel values (0-255)
image = image.astype(np.uint8)  # Convert to unsigned 8-bit integer (grayscale)
```

Your goal is to:

1. **Center the data:** Subtract the mean of the image from each pixel value. This is a crucial step in PCA.
2. **Reshape the image:** Flatten the 2D image array into a 1D array (28*28 = 784) so that each pixel becomes a feature.

3. **Calculate the covariance matrix:** Compute the covariance matrix of the flattened data.
4. **Perform Eigen Decomposition:** Find the eigenvectors and eigenvalues of the covariance matrix.
5. **Select Principal Components:** Choose the top k eigenvectors corresponding to the largest eigenvalues (e.g., $k = 10$).
6. **Project the data:** Project the centered, flattened image onto the selected principal components to reduce the dimensionality.

Reconstruct the image: Use the selected principal components and the projected data to reconstruct the image.