Question 1:

Define the following metrics and perform the following operations:

i) Write a Python program using Python Lists
ii) Write a Python program using NumPy

Matrices:

- **Matrix A:** [[3.7827, 3.3454, 3.2341], [2.2122, 3.5678, 3.9087], [1.1234, 2.8934, 5.9087]]
- **Matrix B:** [[3.1234, 3.0987, 3.1234], [2.1111, 3.2222, 3.3333], [1.0987, 1.3456, 5.1234]]
- **Matrix C:** [[3.1243, 3.0989, 3.1256], [2.6721, 3.6785, 3.9017], [1.1254, 2.8956, 5.9187]]

Answer:

i): Write a Python program using Python Lists

Code:

Considering two matrices, matrix_a and matrix_b, using Python lists.

Calculating the sum of these matrices element-wise and storing the result in matrix_c and it prints.

```
# Matrix A
matrix_a = [
   [3.7827, 3.3454, 3.2341],
   [2.2122, 3.5678, 3.9087],
   [1.1234, 2.8934, 5.9087]
]

# Matrix B
matrix_b = [
   [3.1234, 3.0987, 3.1234],
   [2.1111, 3.2222, 3.3333],
   [1.0987, 1.3456, 5.1234]
]

# Matrix C (Addition of Matrix A and B)
matrix_c = [[matrix_a[i][j] + matrix_b[i][j] for j in range(len(matrix_a[0]))] for i in range(len(matrix_a))]
```

```
# Displaying the result
print("Matrix C (A + B):")
for row in matrix_c:
    print(row)
```

Output:

```
Matrix C (A + B):
[6.9061, 6.4441, 6.3575]
[4.3233, 6.79, 7.242]
[2.2221, 4.239, 11.0321]
```

---

ii): Write a Python program using NumPy

Code:

Taking the NumPy library to define matrices matrix_a and matrix_b.

The sum of these matrices is computed element-wise using NumPy's **'+'** operator and stored in matrix_c.

```
import numpy as np

# Matrix A
matrix_a = np.array([
    [3.7827, 3.3454, 3.2341],
    [2.2122, 3.5678, 3.9087],
    [1.1234, 2.8934, 5.9087]
])

# Matrix B
matrix_b = np.array([
    [3.1234, 3.0987, 3.1234],
    [2.1111, 3.2222, 3.3333],
    [1.0987, 1.3456, 5.1234]
])

# Matrix C (Addition of Matrix A and B)
matrix_c = matrix_a + matrix_b
```

```
# Displaying the result
print("Matrix C (A + B):")
print(matrix_c)
```

Output:

```
Matrix C (A + B):
[[ 6.9061  6.4441  6.3575]
 [ 4.3233  6.79    7.242 ]
 [ 2.2221  4.239  11.0321]]
```

Question 2:

Write a Python Program and perform the following operations:

I. Create a List {1225, 4986, 6789, 7890, 2345, 6783, 0987, 1234, 8765, 3456}
II. Iterate using a for loop
III. Iterate using a for loop and range
IV. List Comprehension
V. Enumerate
VI. Iter function and next function
VII. Map function
VIII. Using zip
IX. Using NumPy Module

Answer:

I: Create a List

Code:

Creating a list of integers and prints it.

```
# List creation
my_list = [1225, 4986, 6789, 7890, 2345, 6783, 987, 1234, 8765, 3456]
print("List:", my_list)
```

Output:

```
List: [1225, 4986, 6789, 7890, 2345, 6783, 987, 1234, 8765, 3456]
```

II: Iterate using a for loop

Code:

The below loop iterates over each element in the list and prints it.

```
for item in my_list:
    print(item)
```

Output:

```
1225
4986
6789
7890
2345
6783
987
1234
8765
3456
```

III: Iterate using a for loop and range

Code:

Below loop uses the range function to iterate over the index positions of the list and prints each element.

```
for i in range(len(my_list)):
    print(my_list[i])
```

Output:

```
1225
4986
6789
7890
2345
6783
987
1234
8765
3456
```

IV: List Comprehension

Code:

Using list comprehension to create a new list where each element is the square of the corresponding element in the original list.

```
squared_list = [x**2 for x in my_list]
print("Squared List:", squared_list)
```

Output:

```
Squared List: [1500625, 24860296, 46187221, 62252100, 5509025, 46007289, 974169, 1522756, 76895325, 11929536]
```

V: Enumerate

Code:

Provides both the index and the value while iterating over the list.

```
for index, item in enumerate(my_list):
    print(f"Index {index}: {item}")
```

Output:

```
Index 0: 1225
Index 1: 4986
Index 2: 6789
Index 3: 7890
Index 4: 2345
Index 5: 6783
Index 6: 987
Index 7: 1234
Index 8: 8765
Index 9: 3456
```

VI: Iter function and next function

Code:

This function is used to create an iterator for the list, and next is used to get the next item in the sequence.

```
list_iter = iter(my_list)
print(next(list_iter))  # Output: 1225
print(next(list_iter))  # Output: 4986
```

Output:

```
1225
4986
```

VII: Map function

Code:

This function applies the lambda function to each element of the list, effectively doubling each value.

```
mapped_list = list(map(lambda x: x * 2, my_list))
```

```
print("Mapped List:", mapped_list)
```

Output:

```
Mapped List: [2450, 9972, 13578, 15780, 4690, 13566, 1974, 2468, 17530, 6912]
```

VIII: Using zip

Code:

Zipping pairs elements from two lists into tuples. If the lists are of different lengths, the shortest one dictates the number of pairs.

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
zipped_list = list(zip(list1, list2))
print("Zipped List:", zipped_list)
```

Output:

```
Zipped List: [(1, 4), (2, 5), (3, 6)]
```

IX: Using NumPy Module

Code:

Creating an array from the original list, allowing for more advanced numerical operations.

```
import numpy as np

np_array = np.array(my_list)
print("NumPy Array:", np_array)
```

Output:

```
NumPy Array: [1225 4986 6789 7890 2345 6783  987 1234 8765 3456]
```

Question 3:

For a List A, B, C, D, E write a Python program to compute all the combinations and permutations

Answer:

Code:

→ itertools.permutations generates all possible arrangements of the elements

→ itertools.combinations generates all possible selections of a specified length.

```
from itertools import permutations, combinations

# List A, B, C, D, E
elements = ['A', 'B', 'C', 'D', 'E']

# Computing permutations
perm = list(permutations(elements))
print("Permutations:")
for p in perm:
    print(p)

# Computing combinations
comb = list(combinations(elements, 2))

# Changing 2 to the desired length
print("\nCombinations:")
for c in comb:
    print(c)
```

Output:

```
Permutations:
('A', 'B', 'C', 'D', 'E')
```

```
('A', 'B', 'C', 'E', 'D')
...

Combinations:
('A', 'B')
('A', 'C')
...
```

Question 4:

Using the same list, use itertools to compute permutations and combinations

Answer:

Code:

The below code explicitly uses itertools for both permutations and combinations, by explaining how to compute these in Python.

```
from itertools import permutations, combinations

# List A, B, C, D, E
elements = ['A', 'B', 'C', 'D', 'E']

# Computing permutations
perm = list(permutations(elements))
print("Permutations:")
for p in perm:
    print(p)

# Computing combinations
comb = list(combinations(elements, 3))

# Changing 3 to the desired length
print("\nCombinations:")
for c in comb:
    print(c)
```

Output:

```
Permutations:
('A', 'B', 'C', 'D', 'E')
('A', 'B', 'C', 'E', 'D')
...

Combinations:
('A', 'B', 'C')
('A', 'B', 'D')
...
```