**Week 5 practice tasks**

Note: All of these tasks should be done by writing functions in a separate file having extension “py”. E.g

%run try.py

**Numpy**

import array

# Creating an array of integers

int\_array = array.array('i', [1, 2, 3, 4, 5])

# Accessing elements of the array

print(int\_array[0]) # Output: 1

print(int\_array[2]) # Output: 3

# Modifying elements of the array

int\_array[1] = 10

print(int\_array) # Output: array('i', [1, 10, 3, 4, 5])

# Appending elements to the array

int\_array.append(6)

print(int\_array) # Output: array('i', [1, 10, 3, 4, 5, 6])

# Removing elements from the array

int\_array.pop(2)

print(int\_array) # Output: array('i', [1, 10, 4, 5, 6])

Note: By specifying the type code 'i', you are indicating that the elements of the array will be signed integers. This allows Python to allocate memory efficiently for storing integers in the array.

Different type codes represent different data types, and they determine how the array stores and interprets the data. For example:

'i': Signed integer

'f': Floating-point number

'd': Double-precision floating-point number

'l': Signed integer (deprecated, similar to 'i')

'u': Unsigned integer

'c': Character (interpreted as a single byte)

While NumPy provides more extensive functionality and better performance for numerical computations, using Python's built-in arrays can be advantageous in situations where you want a lightweight solution without additional dependencies. However, keep in mind that Python arrays are less versatile compared to NumPy arrays and are restricted to a single data type.

**Question1:** Matrix Multiplication:

Multiply two matrices using NumPy's np.dot() function.

import numpy as np

# Define two matrices

A = np.array([[1, 2], [3, 4]])

B = np.array([[5, 6], [7, 8]])

# Multiply matrices

C = np.dot(A, B)

print(C)

**output:**

**Question2 : Element-wise Operations:**

Perform element-wise addition, subtraction, multiplication, and division on arrays.

import numpy as np

# Define two arrays

x = np.array([1, 2, 3])

y = np.array([4, 5, 6])

# Element-wise operations

sum\_array = x + y

diff\_array = x - y

product\_array = x \* y

quotient\_array = x / y

print("Sum:", sum\_array)

print("Difference:", diff\_array)

print("Product:", product\_array)

print("Quotient:", quotient\_array)

**Output:**

np.arange() is a NumPy function that returns an array with evenly spaced values within a specified range.

In this case, np.arange(1, 10) creates an array containing integers from 1 to 9 (inclusive of 1, exclusive of 10). The array will look like this: [1, 2, 3, 4, 5, 6, 7, 8, 9].

x.reshape(3, 3):

x is the array created by np.arange(1, 10).

reshape() is a method of NumPy arrays used to change the shape of an array without changing its data. It returns a new array with a specified shape.

In this case, reshape(3, 3) reshapes the array x into a 3x3 matrix. The elements of the original array are filled into the new shape row by row. If the original array does not have enough elements to fill the new shape, a ValueError will be raised.

**Question 3: Array Reshaping:**

Reshape an array into a different shape.

import numpy as np

# Define an array

x = np.arange(1, 10)

# Reshape array

y = x.reshape(3, 3)

print(y)

**Output:**

**Question 4: Statistical Operations:**

Calculate statistics such as mean, median, standard deviation, etc., for an array.

import numpy as np

# Define an array

x = np.array([1, 2, 3, 4, 5])

# Calculate statistics

mean\_value = np.mean(x)

median\_value = np.median(x)

std\_deviation = np.std(x)

print("Mean:", mean\_value)

print("Median:", median\_value)

print("Standard Deviation:", std\_deviation)

**Output:**

**Question 5: Random Number Generation:**

import numpy as np

# Generate random array

random\_array = np.random.rand(3, 3)

print(random\_array)

**Output:**

**Question 6:** Iterate over the elements of a 2D array row by row to produce following output

**1 2 3**

**4 5 6**

**7 8 9**