|  |
| --- |
| Experiment No. 9 |
| Program to manipulate arrays using NumPy |
| Date of Performance:18/3/2024 |
| Date of Submission:25/3/2024 |

**Experiment No. 9**

**Title:** Program to manipulate arrays using NumPy

**Aim:** To study and implement arrays manipulation using NumPy

**Objective:** To introduce NumPy package **Theory:**

**Numpy** is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

*Arrays in Numpy*

Array in Numpy is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In Numpy, number of dimensions of the array is called rank of the array.A tuple of integers giving the size of the array along each dimension is known as shape of the array. An array class in Numpy is called as **ndarray**. Elements in Numpy arrays are accessed by using square brackets and can be initialized by using nested Python Lists.

**Creating a Numpy Array**

Arrays in Numpy can be created by multiple ways, with various number of Ranks, defining the size of the Array. Arrays can also be created with the use of various data types such as lists, tuples, etc. The type of the resultant array is deduced from the type of the elements in the sequences.

**Note:** Type of array can be explicitly defined while creating the array.

**Code :**

import numpy as np

arr = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]]) print("Original array:") print(arr)

print("\nAccessing elements:") print("Element at index (1, 2):", arr[1, 2]) print("First row:", arr[0]) print("Second column:", arr[:, 1])

print("\nSlicing arrays:") print("First two rows:") print(arr[:2]) # Slicing rows print("Last two columns:") print(arr[:, -2:]) # Slicing columns

print("\nReshaping array:") reshaped\_arr = arr.reshape(1, 9) print(reshaped\_arr) print("\nTransposing array:") transposed\_arr = arr.T print(transposed\_arr)

print("\nElement-wise operations:") arr\_add = arr + 10 print("Adding 10 to each element:") print(arr\_add)

arr\_mul = arr \* 2 print("Multiplying each element by 2:") print(arr\_mul)

print("\nArray concatenation:") arr\_concat = np.concatenate((arr, arr), axis=0) print("Concatenated along rows:") print(arr\_concat)

**Output:**

Original array:

[[1 2 3]

[4 5 6]

[7 8 9]]

Accessing elements:

Element at index (1, 2): 6

First row: [1 2 3]

Second column: [2 5 8]

Slicing arrays:

First two rows:

[[1 2 3]

1. 5 6]]

Last two columns:

[[2 3]

1. 6] [8 9]]

Reshaping array:

[[1 2 3 4 5 6 7 8 9]]

Transposing array:

[[1 4 7]

1. 5 8]
2. 6 9]]

Element-wise operations:

Adding 10 to each element:

[[11 12 13]

[14 15 16]

[17 18 19]]

Multiplying each element by 2:

[[ 2 4 6]

[ 8 10 12]

[14 16 18]]

Array concatenation:

Concatenated along rows:

[[1 2 3]

[4 5 6]

[7 8 9]

[1 2 3]

[4 5 6]

[7 8 9]]

**Conclusion:** The NumPy package was utilized to create and manipulate arrays. An array `arr` was initialized with values from 1 to 9 arranged in a 3x3 matrix. Various operations were performed on `arr`, including accessing specific elements, slicing rows and columns, reshaping the array, transposing it, performing element-wise operations such as addition and multiplication, and concatenating arrays along the row axis. Each operation yielded the expected results, demonstrating the versatility and efficiency of NumPy for array manipulation tasks.