NUMPY ASSIGNMENT

1. Explain the purpose and advantages of NumPy in scientific computing and data analysis. How does it enhance Python's capabilities for numerical operations?

A. NumPy is a foundational library in Python for numerical computing, offering:

- **Fast Numerical Operations**: Efficiently handles multi-dimensional arrays and matrices with operations that are optimized in C.
- **Array Support**: Provides `ndarray`, a versatile and memory-efficient array object.
- **Rich Functionality**: Includes mathematical functions, linear algebra routines, random number generators, and more.
- **Integration**: Works seamlessly with other Python libraries like Pandas, SciPy, and Matplotlib.

Enhancement of Python's Capabilities:

- **Speed**: NumPy's vectorized operations are faster than Python loops for large datasets.
- **Memory Efficiency**: Uses less memory compared to Python lists due to homogeneous data storage.
- **Flexibility**: Supports advanced indexing and broadcasting for elegant, compact code

2. Compare and contrast np.mean() and np.average() functions in NumPy. When would you use one over the other?

A. **`np.mean()`**: Computes the arithmetic mean of an array along a specified axis.

Ex: - arr = np.array([1, 2, 3, 4,])

Print(np.mean(arr)) #2.5

`np.average()`: Computes a weighted average. If no weights are specified, it behaves like `np.mean()`

Ex: - wt = [1, 2, 1, 1]

print(np.wt(arr, weight = wt)) # 2.6

NOTE: -

- Use `np.mean()` for simple averages.
- Use `np.average()` when weights are involved

3. Describe the methods for reversing a NumPy array along different axes. Provide examples for 1D and 2D arrays.

A. 1d array: -

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

rev_arr = arr[::-1]

Print(rev_arr) # [ 4 3 2 1]
```

2d array: -

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

rev_row = arr[::-1]

rev_col = arr[:, ::-1]

Print(rev_row)  #[[3 4] [1 2]]

Print(rev_cols) #[[2 1] [4 3]]
```

4. How can you determine the data type of elements in a NumPy array? Discuss the importance of data types in memory management and performance.

A.

```
arr = np.array([1, 2, 3], dtype = np.int32)

        Print(aee.dtype) # int32
```

Importance of Data Types:

- **Memory Management**: Optimized memory usage due to fixed-size data types.
- **Performance**: Faster computations with lower-level type handling compared to Python's dynamic typing

5. Define ndarrays in NumPy and explain their key features. How do they differ from standard Python lists?

A. **Definition**: `ndarray` is the core data structure of NumPy, representing N-dimensional arrays.

- **Key Features**:
  - Homogeneous data.
  - Optimized for numerical operations.
  - Supports advanced slicing, indexing, and broadcasting.
- **Differences from Python Lists**:

- Fixed-size and homogeneous vs. variable-size and heterogeneous.
- Faster operations due to C-optimized functions

6. Analyze the performance benefits of NumPy arrays over Python lists for large-scale numerical operations.

A.

- **Speed**: NumPy arrays use vectorized operations implemented in C, which are faster than Python's loops.
- **Memory Efficiency**: Store elements of the same type compactly, unlike Python lists with their object overhead.

Ex: - arr = np.arange(1e6)

lst = list(range(int(1e6)))

7. Compare vstack() and hstack() functions in NumPy. Provide examples demonstrating their usage and output.

A.

`vstack()`: Stacks arrays vertically.

Ex: -

a = np.array([1, 2])

b = np.array([3, 4])

print(np.vstack((a, b))) # Output: [[1 2] [3 4]]

`hstack()`: Stacks arrays horizontally

Ex: -

print(np.hstack((a, b))) # Output: [1 2 3 4]

8. Explain the differences between fliplr() and flipud() methods in NumPy, including their effects on various array dimensions.

A.

`fliplr()`: Reverses the order of columns (left to right)

Ex: -

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

print(np.fliplr(arr)) # Output: [[2 1] [4 3]]

`flipud()`: Reverses the order of rows (up to down).

Ex: -

print(np.flipud(arr)) # Output: [[3 4] [1 2]]

9. Discuss the functionality of the array_split() method in NumPy. How does it handle uneven splits?

A. `array_split()` Functionality:

Splits an array into sub-arrays, handling uneven splits by creating smaller-sized arrays

Ex: - arr = np.array([1, 2, 3, 4, 5])

parts = np.array_split(arr, 3)

print(parts) # Output: [array([1, 2]), array([3, 4]), array([5])]

10. Explain the concepts of vectorization and broadcasting in NumPy. How do they contribute to efficient array operations?

A. **Vectorization**: Replaces explicit loops with array operations, leveraging low-level optimizations.

Ex: -

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

print(arr + 10) # Output: [11 12 13]

Broadcasting: Automatically expands smaller arrays to perform operations on arrays of different shapes

Ex: -

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

b = np.array([10, 20])

print(a + b) # Output: [[11 22] [13 24]]

Practical Questions: -

```
# 1. Create a 3x3 NumPy array with random integers between 1 and 100.
Interchange its rows and columns.
import numpy as np
array_3x3 = np.random.randint(1, 101, size=(3, 3))
transposed_array = array_3x3.T
```

```
print("1. Original Array:\n", array_3x3)
print("Transposed Array:\n", transposed_array)

# 2. Generate a 1D NumPy array with 10 elements. Reshape it into a 2x5 array,
then into a 5x2 array.
array_1d = np.arange(10)
array_2x5 = array_1d.reshape(2, 5)
array_5x2 = array_1d.reshape(5, 2)
print("\n2. 1D Array:", array_1d)
print("Reshaped to 2x5:\n", array_2x5)
print("Reshaped to 5x2:\n", array_5x2)

# 3. Create a 4x4 NumPy array with random float values. Add a border of zeros
around it, resulting in a 6x6 array.
array_4x4 = np.random.rand(4, 4)
array_6x6 = np.pad(array_4x4, pad_width=1, mode='constant', constant_values=0)
print("\n3. Original 4x4 Array:\n", array_4x4)
print("6x6 Array with Border:\n", array_6x6)

# 4. Using NumPy, create an array of integers from 10 to 60 with a step of 5.
array_step = np.arange(10, 61, 5)
print("\n4. Array from 10 to 60 with step 5:", array_step)

# 5. Create a NumPy array of strings ['python', 'numpy', 'pandas'] and apply
case transformations.
string_array = np.array(['python', 'numpy', 'pandas'])
upper_case = np.char.upper(string_array)
lower_case = np.char.lower(string_array)
title_case = np.char.title(string_array)
print("\n5. String Array:", string_array)
print("Uppercase:", upper_case)
print("Lowercase:", lower_case)
print("Title Case:", title_case)

# 6. Generate a NumPy array of words and insert a space between each character
of every word in the array.
spaced_array = np.char.join(' ', string_array)
print("\n6. Spaced Characters Array:", spaced_array)

# 7. Create two 2D NumPy arrays and perform element-wise addition,
subtraction, multiplication, and division.
array1 = np.random.randint(1, 10, size=(2, 2))
array2 = np.random.randint(1, 10, size=(2, 2))
addition = np.add(array1, array2)
subtraction = np.subtract(array1, array2)
multiplication = np.multiply(array1, array2)
```

```
division = np.divide(array1, array2)
print("\n7. Array 1:\n", array1)
print("Array 2:\n", array2)
print("Addition:\n", addition)
print("Subtraction:\n", subtraction)
print("Multiplication:\n", multiplication)
print("Division:\n", division)

# 8. Create a 5x5 identity matrix and extract its diagonal elements.
identity_matrix = np.eye(5)
diagonal_elements = np.diag(identity_matrix)
print("\n8. Identity Matrix:\n", identity_matrix)
print("Diagonal Elements:", diagonal_elements)

# 9. Generate a NumPy array of 100 random integers between 0 and 1000. Find
and display all prime numbers.
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(np.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True

random_integers = np.random.randint(0, 1000, size=100)
primes = [num for num in random_integers if is_prime(num)]
print("\n9. Random Integers:\n", random_integers)
print("Prime Numbers:", primes)

# 10. Create a NumPy array representing daily temperatures for a month.
Calculate and display the weekly averages.
daily_temperatures = np.random.randint(20, 40, size=30)
weekly_averages = np.mean(daily_temperatures.reshape(5, 6), axis=1)
print("\n10. Daily Temperatures:\n", daily_temperatures)
print("Weekly Averages:", weekly_averages)
```