```
import numpy as np

# Assignment-1
# Q1: Questions on Basic NumPy Array

# (a) Reverse the NumPy array: arr = np.array([1, 2, 3, 6, 4, 5])
arr = np.array([1, 2, 3, 6, 4, 5])
reversed_arr = arr[::-1]
print("Reversed Array:", reversed_arr)

# (b) Flatten the NumPy arr: array1 = np.array([[1, 2, 3], [2, 4, 5], [1, 2, 3]]) using any two
NumPy in-built methods
array1 = np.array([[1, 2, 3], [2, 4, 5], [1, 2, 3]])
flattened1 = array1.flatten()  # Method 1
flattened2 = array1.ravel()    # Method 2
print("Flattened Array (Method 1):", flattened1)
print("Flattened Array (Method 2):", flattened2)

# (c) Compare the following numpy arrays:
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[1, 2], [3, 4]])
are_equal = np.array_equal(arr1, arr2)
print("Are arrays equal?", are_equal)

# (d) Find the most frequent value and their indice(s) in the following arrays:
x = np.array([1, 2, 3, 4, 5, 1, 2, 1, 1, 1])
y = np.array([1, 1, 1, 2, 3, 4, 2, 4, 3, 3])

def most_frequent(arr):
    values, counts = np.unique(arr, return_counts=True)
    max_count = np.max(counts)
    most_frequent_value = values[counts == max_count]
    indices = np.where(arr == most_frequent_value[0])[0]
    return most_frequent_value[0], indices

freq_value_x, indices_x = most_frequent(x)
freq_value_y, indices_y = most_frequent(y)
print(f"Most frequent value in x: {freq_value_x} at indices {indices_x}")
print(f"Most frequent value in y: {freq_value_y} at indices {indices_y}")

# (e) For the array gfg = np.matrix('[4, 1, 9; 12, 3, 1; 4, 5, 6]'), find
gfg = np.matrix('[4, 1, 9; 12, 3, 1; 4, 5, 6]')

# i. Sum of all elements
sum_all_elements = gfg.sum()
print("Sum of all elements:", sum_all_elements)

# ii. Sum of all elements row-wise
```

```
sum_row_wise = gfg.sum(axis=1)
print("Sum of all elements row-wise:", sum_row_wise)

# iii. Sum of all elements column-wise
sum_col_wise = gfg.sum(axis=0)
print("Sum of all elements column-wise:", sum_col_wise)

# (f) For the matrix: n_array = np.array([[55, 25, 15],[30, 44, 2],[11, 45, 77]]), find
n_array = np.array([[55, 25, 15], [30, 44, 2], [11, 45, 77]])

# i. Sum of diagonal elements
sum_diagonal = np.trace(n_array)
print("Sum of diagonal elements:", sum_diagonal)

# ii. Eigen values of matrix
eigen_values = np.linalg.eigvals(n_array)
print("Eigen values of matrix:", eigen_values)

# iii. Eigen vectors of matrix
eigen_values, eigen_vectors = np.linalg.eig(n_array)
print("Eigen vectors of matrix:", eigen_vectors)

# iv. Inverse of matrix
inverse_matrix = np.linalg.inv(n_array)
print("Inverse of matrix:\n", inverse_matrix)

# v. Determinant of matrix
determinant = np.linalg.det(n_array)
print("Determinant of matrix:", determinant)

# (g) Multiply the following matrices and also find covariance between matrices using
NumPy:

# i. p = [[1, 2], [2, 3]], q = [[4, 5], [6, 7]]
p1 = np.array([[1, 2], [2, 3]])
q1 = np.array([[4, 5], [6, 7]])
product1 = np.dot(p1, q1)
covariance1 = np.cov(p1.T, q1.T)
print("Product of matrices (1):\n", product1)
print("Covariance between matrices (1):\n", covariance1)

# ii. p = [[1, 2], [2, 3], [4, 5]], q = [[4, 5, 1], [6, 7, 2]]
p2 = np.array([[1, 2], [2, 3], [4, 5]])
q2 = np.array([[4, 5, 1], [6, 7, 2]])
product2 = np.dot(p2, q2)
covariance2 = np.cov(p2.T, q2.T)
print("Product of matrices (2):\n", product2)
print("Covariance between matrices (2):\n", covariance2)
```

```
# (h) For the matrices: x = np.array([[2, 3, 4], [3, 2, 9]]); y = np.array([[1, 5, 0], [5, 10, 3]]),
x = np.array([[2, 3, 4], [3, 2, 9]])
y = np.array([[1, 5, 0], [5, 10, 3]])

# Find inner, outer, and cartesian product
inner_product = np.inner(x, y)
outer_product = np.outer(x, y)
cartesian_product = np.dot(x, y.T)
print("Inner product:\n", inner_product)
print("Outer product:\n", outer_product)
print("Cartesian product:\n", cartesian_product)

# Q2: Based on NumPy Mathematics and Statistics

# (a) For the array: array = np.array([[1, -2, 3],[-4, 5, -6]])

array = np.array([[1, -2, 3], [-4, 5, -6]])

# i. Find element-wise absolute value
abs_values = np.abs(array)
print("Element-wise absolute values:\n", abs_values)

# ii. Find the 25th, 50th, and 75th percentile of flattened array, for each column, for each row.
percentiles_flattened = np.percentile(array, [25, 50, 75])
percentiles_columns = np.percentile(array, [25, 50, 75], axis=0)
percentiles_rows = np.percentile(array, [25, 50, 75], axis=1)
print("Percentiles of flattened array:", percentiles_flattened)
print("Percentiles of each column:\n", percentiles_columns)
print("Percentiles of each row:\n", percentiles_rows)

# iii. Mean, Median and Standard Deviation of flattened array, of each column, and each row
mean_flattened = np.mean(array)
median_flattened = np.median(array)
std_flattened = np.std(array)
mean_columns = np.mean(array, axis=0)
median_columns = np.median(array, axis=0)
std_columns = np.std(array, axis=0)
mean_rows = np.mean(array, axis=1)
median_rows = np.median(array, axis=1)
std_rows = np.std(array, axis=1)
print("Mean, Median, Std of flattened array:", mean_flattened, median_flattened,
std_flattened)
print("Mean, Median, Std of each column:\n", mean_columns, median_columns,
std_columns)
print("Mean, Median, Std of each row:\n", mean_rows, median_rows, std_rows)

# (b) For the array: a = np.array([-1.8, -1.6, -0.5, 0.5,1.6, 1.8, 3.0])
```

```
a = np.array([-1.8, -1.6, -0.5, 0.5, 1.6, 1.8, 3.0])

# Find floor, ceiling, truncated, and rounded values
floor_values = np.floor(a)
ceiling_values = np.ceil(a)
truncated_values = np.trunc(a)
rounded_values = np.round(a)
print("Floor values:", floor_values)
print("Ceiling values:", ceiling_values)
print("Truncated values:", truncated_values)
print("Rounded values:", rounded_values)

# Q3: Based on Searching and Sorting

# (a) For the array: array = np.array([10, 52, 62, 16, 16, 54, 453])

array = np.array([10, 52, 62, 16, 16, 54, 453])

# i. Sorted array
sorted_array = np.sort(array)
```