```
import numpy as np
```

Array Creation

```
#1-dimensional array of integers from 1 to 10.
array1 = np.arange(1,11)
array1
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
#2-dimensional array with random float values of shape (3, 4).
array2 = np.random.rand(3,4)
array2
```

```
array([[0.56177447, 0.25112734, 0.37403771, 0.31595353],
       [0.33330218, 0.9043981 , 0.43234581, 0.66089986],
       [0.30518854, 0.71518162, 0.57506181, 0.89036983]])
```

```
#3-dimensional array of zeros with shape (2, 3, 4).
arrayzeros = np.zeros((2,3,4))
arrayzeros
```

```
array([[[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]])
```

Array Operations

```
#Given two 1-dimensional arrays, perform element-wise addition, subtraction, multiplication, and division.
array1 = np.array([2,3,4,5])
array2 = np.array([3,2,1,1])

#Addtional
arrayAddtion = array1 + array2
print("Array Addition: ", arrayAddtion)

#Subtraction
arraySubtraction = array1 - array2
print("Array Subtraction: ", arraySubtraction)

#multiplication
arrayMultiplication = array1 * array2
print("Array Multiplication: ", arrayMultiplication)

#division
arrayDivision = array1 / array2
print("Array Division: ", arrayDivision)
```

```
Array Addition:  [5 5 5 6]
Array Subtraction:  [-1  1  3  4]
Array Multiplication:  [6 6 4 5]
Array Division:  [0.66666667 1.5        4.         5.        ]
```

```
#mean, median, and standard deviation of a given 1-dimensional array.
arrayy = [2,3,4,1]

#mean
arrayMean = np.mean(arrayy)
print("Mean: ", arrayMean)

#median
arrayMedian = np.median(arrayy)
print("Median: ", arrayMedian)

arraySD = np.std(arrayy)
print("Standard Deviation: ", arraySD)
```

```
Mean:  2.5
Median:  2.5
Standard Deviation:  1.118033988749895
```

```
# Reshape a 1-dimensional array into a 2-dimensional array of shape (3,4).
oneD = np.random.randint(0,5, size=12)
print(oneD)

oneD_to_twoD = oneD.reshape(3,4)
print(oneD_to_twoD)
```

```
[3 1 4 4 4 4 4 0 2 0 2 2]
[[3 1 4 4]
 [4 4 4 0]
 [2 0 2 2]]
```

Array Indexing and Slicing:

```
# Extract the first row and last column of a 2-dimensional array.
two_D_array = np.array([[10, 20, 30, 40],
                        [50, 60, 70, 80],
                        [90, 100, 110, 120]])
print(two_D_array)

#to extract first row
first_row = two_D_array[0]
print("\nFirst Row: ",first_row)

#to extract last column
last_column = two_D_array[:, -1]
print("\nLast Column: ",last_column)
```

```
[[ 10  20  30  40]
 [ 50  60  70  80]
 [ 90 100 110 120]]

First Row:  [10 20 30 40]

Last Column:  [ 40  80 120]
```

```
#Reverse the order of elements in a 1-dimensional array.
array = np.array([1,2,3,4,5])
reversed_array = array[::-1]
print(reversed_array)
```

```
[5 4 3 2 1]
```

```
# Select elements from a 2-dimensional array that satisfy a specificcondition (e.g., values greater than a certain threshold).
two_D_array = np.array([[4,5,6,3],
                        [9,8,7,6]])

#check even number
two_D_array_even = two_D_array[two_D_array % 2 == 0]
print("Conditional arrays: ",two_D_array_even)
```

```
Conditional arrays:  [4 6 8 6]
```

Array Broadcasting

```
#Add a scalar value to each element of a 2-dimensional array.
arr2_b = np.array([[1, 2, 3],
                   [4, 5, 6]])

array_with_scalar = arr2_b + 5
print(array_with_scalar)
```

```
[[ 6  7  8]
 [ 9 10 11]]
```

```
# Multiply a 1-dimensional array with a 2-dimensional array, leveraging NumPy's broadcasting rules.
one_D = np.array([2, 3],)
two_D = np.array([[1],
                  [4]])
multiply_differnt_dimensions = oneD * two_D
print(multiply_differnt_dimensions)
```

```
[[ 3  1  4  4  4  4  4  0  2  0  2  2]
 [12  4 16 16 16 16 16  0  8  0  8  8]]
```

Linear Algebra

```
# Calculate the dot product of two 1-dimensional arrays.
arr1 = np.array([1,2,3,4])
arr2 = np.array([6,7,2,3])
dot_array = np.dot(arr1, arr2)
print("Dot product is: ", dot_array)
```

```
Dot product is:  38
```

```
# Matrix multiplication (2D)
m1 = np.array([[1, 2],
               [3, 4]])
m2 = np.array([[5, 6],
               [7, 8]])
matrix_multiplication = np.matmul(m1, m2)
print(matrix_multiplication)
```

```
[[19 22]
 [43 50]]
```

```
# Inverse and Determinant of square matrix
matrix = np.array([[4, 7],
                   [2, 6]])

#Inverse
print("Inverse:\n", np.linalg.inv(matrix))

#Determinant
print("\nDeterminant:\n", np.linalg.det(matrix))
```

```
Inverse:
 [[ 0.6 -0.7]
 [-0.2  0.4]]

Determinant:
 10.000000000000002
```

Questions:

1. What is the difference between a scalar, vector, and matrix in NumPy?

```
Scalar - A single value (0-dimensional). Example: 5 or np.array(5).
Vector - A 1-dimensional array of values. Example: [1, 2, 3] or np.array([1, 2, 3]).
Matrix - A 2-dimensional array (rows × columns). Example: [[1, 2], [3, 4]].
```

2. How can you create an array with evenly spaced values within a given range?

```
Use np.linspace(start, end, num_points) for a specific number of evenly spaced points.
Use np.arange(start, end, step) for evenly spaced values with a fixed step size.
```

3. Explain the concept of array broadcasting in NumPy.

```
Broadcasting allows NumPy to perform operations on arrays of different shapes without making copies.
Rules:
*   If shapes differ, NumPy automatically expands the smaller one to match.
*   Works if dimensions are either equal or one of them is 1.
```

4. How can you perform element-wise operations on NumPy arrays?

```
Element-wise means operation is applied to each corresponding element:
Example:
        arr1 + arr2 → adds each element of arr1 to arr2.
Works for addition, subtraction, multiplication, division, etc., if shapes are compatible (or broadcastable).
```

5. What is the purpose of the np.newaxis in NumPy?

```
Used to add a new dimension to an array.
example:
          arr.shape  # (3,)
          arr[:, np.newaxis].shape  # (3, 1)
```

6. How can you sort a NumPy array along a specific axis?

```
Use np.sort(arr, axis=0) (sort column-wise) or axis=1 (row-wise).
np.argsort() returns the indices that would sort the array.
```

7. Explain the difference between np.array and np.asarray functions.

```
np.array() always creates a new array (makes a copy).
np.asarray() does not copy if the input is already a NumPy array (more memory-efficient).
```

8. What are the advantages of using NumPy over Python's built-in lists for numerical operations?
   ans:

```
*   Faster numerical computations (written in C under the hood).
*   Vectorized operations (no need for loops).
*   Less memory usage.
*   Supports multi-dimensional arrays and advanced math operations easily.
```

9. How can you save and load NumPy arrays to/from disk?

```
Save: np.save('file.npy', arr) -> saves in binary .npy format.
Load: arr = np.load('file.npy').
For multiple arrays: np.savez('file.npz', arr1=..., arr2=...).
```

10. What is the purpose of the np.where function in NumPy?

```
Returns indices or values where a condition is True.
Example:
np.where(arr > 5)  # indices where condition is True
np.where(arr > 5, 1, 0)  # replace True with 1, False with 0
```