NUMPY

1. What is numpy?

   It stands for Numerical Python.

   It is an open source python library which is used for numerical and statistical computing.

2. Syntax to install numpy.

   pip install numpy

3. Syntax to import numpy.

   Import numpy as np

4. What is Array?

   An Array is a data structure that stores homogeneous elements in a contiguous memory location.

5. Syntax to create an Array?

```
Variable=np.array(object,dtype=None,ndmin=0,copy=True)
```

object: Any array-like object (list, tuple, etc.) or (primitive for scalar array)
dtype: (Optional) Desired data type of array elements.
ndmin: (Optional) Specifies minimum dimensions.
copy: (Optional) If True, a copy of the object is made.

6. What are the features of ARRAY?

1. stores homogeneous elements.
2. supports indexing and slicing (+ve and -ve).
3. supports duplicate elements
4. It is ordered collection.
5. supports element wise operation.
6. It is faster than python List for numerical computations.
7. Partially mutable:
   - Elements can be updated.
   - Insertion or deletion is not supported directly.
8. It is fixed size.
9. supports large collection of high-level mathematical functions to operate on these arrays.
10. array supports broadcasting.

7. What are the types of Array?

Scalar array	(0-D array)
Uni-Dimensional array	(1-D array)
Matrix	(2-D array)

Multi-Dimensional array (nd array)

8. Array Attributes

ndim:
Returns the Number of dimensions of the array.
Syntax: array.ndim

dtype:
Returns the Data type of array elements.
Syntax: array.dtype

size:
Returns the total number of elements in the array.
Syntax: array.size

shape:
Returns the Dimensions of the array (no_of_elements in each dimension).
Syntax: array.shape

itemsize:
Returns the memory (in bytes) consumed by the one array element.
Syntax: array.itemsize

nbytes:
Returns the total memory (in bytes) consumed by the array.
Syntax: array.nbytes

9. What is indexing , slicing ?

Indexing: Accessing a single element from an array.
Slicing: Accessing a range or subset of elements.

10. Syntax for Indexing and slicing .

ndarray

array_name[index1, index2, ..., indexN]

array_name[start:stop:step, ...]

Note: indexing and slicing doesn't support for scalar array.

11. What is broadcasting?

automatic stretching of smaller arrays to match bigger ones, so element-wise operations can be done.
Every corresponding dimension must match or be **1** (from the right to left when comparing shapes).

12. Array Operations.

1. Array with constant
2. Array with Array

```
import numpy as np

array1 = np.array([2, 1, 5, 3, 4])

cons = 3


print(array1 + cons)   # Add constant → [5 4 8 6 7]

print(array1 - cons)   # Subtract constant → [-1 -2 2 0 1]

print(array1 * cons)   # Multiply by constant → [6 3 15 9 12]

print(array1 / cons)   # Divide by constant → [0.666... 0.333... 1.666... 1.0 1.333...]

print(array1 % cons)   # Modulus → [2 1 2 0 1]


print(array1 > cons)   # Greater than → [False False  True  True  True]

print(array1 < cons)   # Less than → [ True  True False False False]

print(array1 == cons)  # Equal to → [False False False  True False]

print(array1 != cons)  # Not equal → [True True True False True]


print(array1 & cons)   # Bitwise AND → [2 1 1 3 0]

print(array1 | cons)   # Bitwise OR → [3 3 7 3 7]

print(array1 ^ cons)   # Bitwise XOR → [1 2 6 0 7]

print(~array1)         # Bitwise NOT → [-3 -2 -6 -4 -5]
```

```
import numpy as np

array1 = np.array([2, 1, 5, 3, 4])

array2 = np.array([1, 5, 5, 3, 3])


print(array1 + array2)    # [3 6 10 6 7]

print(array1 - array2)    # [ 1 -4  0  0  1]

print(array1 * array2)   # [ 2  5 25  9 12]

print(array1 / array2)   # [2.        0.2        1.        1.        1.33333333]

print(array1 % array2)   # [0 1 0 0 1]


print(array1 > array2)   # [ True False False False  True]

print(array1 < array2)   # [False  True False False False]

print(array1 == array2)  # [False False  True  True False]

print(array1 != array2)  # [ True  True False False  True]


print(array1 & array2)   # [0 1 5 3 0]

print(array1 | array2)   # [3 5 5 3 7]

print(array1 ^ array2)   # [3 4 0 0 7]

print(~array1)            # [-3 -2 -6 -4 -5]
```

Array Creation Methods

1. np.zeros(): Creates an array filled with zeros.

   Syntax: np.zeros(shape, dtype=float, order='C')

   Return Type: ndarray

2. np.ones(): Creates an array filled with ones.

   Syntax: np.ones(shape, dtype=float, order='C')

   Return Type: ndarray

3. np.full(): Creates an array filled with a specified constant value.

   Syntax: np.full(shape, fill_value, dtype=None, order='C')

   Return Type: ndarray

4. np.eye(): Creates a 2D identity matrix with ones on the diagonal.

   Syntax: np.eye(N, M=None, k=0, dtype=float, order='C')

   Return Type: ndarray

5. np.arange(): Returns evenly spaced values within a given interval.

   Syntax: np.arange(start, stop, step=1, dtype=None)

   Return Type: ndarray

6. np.linspace(): Returns evenly spaced numbers over a specified interval.

   Syntax: np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)

   Return Type: ndarray (or tuple if retstep=True)

```
a = np.zeros((2, 3), dtype=int)

b = np.ones((2, 3), dtype=float)

c = np.full((2, 3), fill_value=7)

d = np.eye(3, dtype=int)

e = np.arange(1, 10, 2)

f = np.linspace(0, 1, num=5)
```

RANDOM MODULE METHODS

1. np.random.rand(): Generates random floats in the range [0.0, 1.0).

    Syntax: np.random.rand(d0, d1, ..., dn)
    Return Type: ndarray

---

2. np.random.randn(): Generates random floats from the standard normal distribution (mean=0, std=1).

    Syntax: np.random.randn(d0, d1, ..., dn)
    Return Type: ndarray

---

3. np.random.randint(): Generates random integers from low (inclusive) to high (exclusive).

    Syntax: np.random.randint(low, high=None, size=None, dtype=int)
    Return Type: ndarray

---

4. np.random.choice(): Generates a random sample from a given 1D array or iterable(list).

    Syntax: np.random.choice(a, size=None, replace=True, p=None)
    Return Type: ndarray

---

5. np.random.seed(): Sets the seed for reproducibility of random results.

    Syntax: np.random.seed(seed_value)
    Return Type: None

```
np.random.seed(42)

a = np.random.rand(2, 3)

b = np.random.randn(2, 3)

c = np.random.randint(1, 10, size=(2, 3))

d = np.random.choice([10, 20, 30, 40], size=3)
```

AGGREGATE / STATISTICAL METHODS

1. sum(): Returns the sum of array elements over a given axis.

    Syntax: array.sum(axis=None, dtype=None, keepdims=False)

    Return Type: scalar or ndarray

---

2. mean(): Returns the mean of array elements over a given axis.

    Syntax: array.mean(axis=None, dtype=None, keepdims=False)

    Return Type: scalar or ndarray

---

3. max(): Returns the maximum element along a given axis.

    Syntax: array.max(axis=None, keepdims=False)

    Return Type: scalar or ndarray

---

4. min(): Returns the minimum element along a given axis.

    Syntax: array.min(axis=None, keepdims=False)

    Return Type: scalar or ndarray

---

5. prod(): Returns the product of array elements over a given axis.

    Syntax: array.prod(axis=None, dtype=None, keepdims=False)

    Return Type: scalar or ndarray

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

s = a.sum()
m = a.mean()
mx = a.max()
mn = a.min()
p = a.prod()
st = a.std()
v = a.var()
```

6. std(): Returns the standard deviation of array elements.

Syntax: array.std(axis=None, dtype=None, keepdims=False)

Return Type: scalar or ndarray

---

7. var(): Returns the variance of array elements.

Syntax: array.var(axis=None, dtype=None, keepdims=False)

Return Type: scalar or ndarray

---

8. percentile(): Calculates the q-th percentile of the data along the specified axis.

Syntax: np.percentile(a, q, axis=None, interpolation='linear', keepdims=False)

Return Type: scalar or ndarray

```
a = np.array([10, 20, 30, 40, 50])

st = a.std()
v = a.var()


result = np.percentile(a, 25)  # 20.0
```

Array Manipulation Methods in NumPy

1. reshape(): Changes the shape of an array without changing its data.

Syntax: array.reshape(new_shape)

Return Type: ndarray

2. flatten(): Flattens a multi-dimensional array to a 1D array (returns a copy).

Syntax: array.flatten()

Return Type: ndarray

3. ravel(): Returns a flattened 1D array (returns a view when possible).

Syntax: array.ravel()

Return Type: ndarray

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


b = a.reshape(2, 3)

c = a.flatten()

d = a.ravel()
```

CONDITIONAL Methods in NumPy

1. where(): Selects values or positions based on a condition.

    Syntax: np.where(condition[, x, y])

    Return Type: ndarray or tuple of ndarrays

2. any(): Checks if at least one element is True.

    Syntax: array.any(axis=None, keepdims=False)

    Return Type: bool or ndarray

3. all(): Checks if all elements are True.

    Syntax: array.all(axis=None, keepdims=False)

    Return Type: bool or ndarray

4. isin(): Checks if elements are present in a given list.

    Syntax: np.isin(element, test_elements, assume_unique=False, invert=False)

    Return Type: ndarray of bool

```
a = np.array([[10, 15], [20, 25]])
w = np.where(a > 15, 100, -1)                                # [ [ -1  -1]   [100 100]  ]
an = (a > 15).any(axis=1, keepdims=True)                      #  [[False]  [ True]]
al = (a > 5).all(axis=0, keepdims=False)                      #  [ True  True]
i = np.isin(a, [10, 25], assume_unique=True, invert=True)     # [[False  True]  [ True False]]
```