① Difference between a scalar, vector and matrix in Numpy are:

Numpy support for large, multi-dimensional array and matrices with high level mathematical function.

Property	Scalar	Vector	Matrix
Dimension	0-D	1-D	2-D
Shape	( )	(n,)	(m,n)
Example	5	[1,2,3]	[[1,2], [3,4]]

② Creating evenly spaced values within given range, we use linspace (start, stop, num).

Example,

```
import numpy as np.
arr = np.linspace(0,10,5)
print(arr)

#Output
[0., 2.5, 5, 7.5, 10]
```

```
import numpy as np
arr = np.arange(10)
print(arr).

np.arange([start,] stop,
          [step,], dtype)
```

③ Array Broadcasting:

Broadcasting in numpy is a feature that allows arithmetic operations to be performed on arrays of different shapes. Instead of reshaping explicitly, Numpy automatically "broadcasts" the smaller smaller array to match the shape of larger array.

```
import numpy as np
arr = np.array([1,2,3]) #1-D array
result = arr + 5  #Scalar 5 broadcast to [5,5,5]
print(result)
```

(4)

→ Performing element-wise operations on Numpy arrays:

```
import numpy as np
arr1 = np.array([1,2,3])
arr2 = np.array([4,5,6])

res1 = arr1 + arr2  #arithmetic operation.
print(res1)
res1 = arr2 - arr1 #(-) similarly can do *, /

#with universal function
  c = np.add(a,b)
  d = np.subtract(a,b)
  e = np.multiply(a,b)
  f = np.divide(a,b)
  g = np.sin(a)
  h = np.sqrt(a)
```

Date
Page

Ans5) Purpose of np.newaxis in NumPy

firstly Slicing in Python

sequence[start:stop:step]

Example

```
lisst = [1,2,3,4,5]

#Basic Slicing        (close bracket, Open bracket)
print(lisst[1:4])  # [2,3,4]
print(:4)  # [1,2,3]
print(::2) # [1,3,5]

#negative slicing
print[-4:-1] # [2,3,4].

Then, Slicing in Numpy
import numpy as np
arr = np.arr([[1,2,3],[4,5,6],[7,8,9]])

# basic slicing
print(a[0,:]) # [1,4,7] #Row 0, [1,2,3]
print(a[:,1] #Column 1, [2,5,8]
print(a[1:3, 1:3]) # [[5,6]
                     [8,9]]
```

Rough.

0- [1 2 3
1- 4 5 6
2- 7 8 9
   0 1 2

1:2,1
([5]

Date
Page

np.newaxis increases the dimensionality of array by 1.
It changes the shape without changing the data.

Example:

```
import numpy as np
a = np.array([1,2,3,])   # shape (3,)
b = a[:, np.newaxis]     # shape (3,1) new col ad
c = a[np.newaxis, :]     # shape (1,3), new row ad
```

here (3,) indicates, 1-D array has 3- elements.
(3,1) indicates array hay 3 rows 1 colum.
(1,3) indicates array has 1 row 3 column.

Basically newaxis is also reshaping the matrix.

```
b = [[1]          c = [[1,2,3]].
     [2]                 ↳2-D
     [3]]
      ↳2-D.
```

Initially, a was just 1-D array.

We can do the same using reshape.

```
b = a[:, np.newaxis]
```

or,

```
b = a.reshape(-1,1)
```

Date
Page

Qno 6) Sorting a Numpy along a specific axis:

```
b = np.array([[3,1,9], [4,2,8], [5,6,7]])
sort_b = np.sort(b, axis=1)
```

Output: ~~[[3,1,9~~

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

Qno 7) Difference between np.array and np.asarray function

	np.array	np.asarray
Creates a Copy	Always, unless copy=false	Only if the input is not an array
Input Type	Converts any sequence-like object	Converts any sequence like object.
Performace:	Slower due to copying	faster when input already an array

asarray shallow copies the data if its data is already on array time.

shallow copy vaneko data ko refrence copy (like base index of data).

Python Implementation: CPython Jython IronPython PyPy

8) Advantages of using Numpy over python's built-in lists for numerical operations are:

(i) Speed: Because Numpy is implemented in C.

Proof:

```
import numpy as np
import time
arr = np.arange(1_000_000)
start = time.time()
arr = arr * 2
print("Numpy time :", time.time() - start)

# using python range
lst = list(range(1_000_000))
start = time.time()
lst = [x*2 for x in lst]
print("List time :", time.time() - start)
```

(ii) Memory Efficiency:

Numpy arrays store elements in contiguous memory blocks and require less memory compared to Python lists.

9) Save and load Numpy arrays to/from disk.

```
#Save an array to binary file
a = np.array([1,2,3,4])
np.save('data.npy', a)

#loading
b = np.load('data.npy')
```

10) np.where():

- Return elements chosen from two arrays (or a condition) based on a condition.

Syntax:

```
np.where(condition, x, y)
```

x: value to return when condition True

y: value to return when condition False

Condition: An array like or boolean condition

If x and y are array, their shape should match the shape of condition.

```
import numpy as np
arr = np.array([1,6,3,8,5])
result = np.where(arr > 5, 10, 0)
#output [0 10 0 10 0]
```