```
#Attributes
.shape
.size
.ndim
.dtype
```

```
import numpy as np
```

```
x = np.array([5,7,8,2,9,15])
print(x)
```

```
[ 5  7  8  2  9 15]
```

```
#shape  no of rows and column in 1 d
print(x.shape)
```

```
(6,)
```

```
print(x.size)
```

```
6
```

Unsupported cell type. Double-click to inspect/edit the content.

```
print(x.dtype)
```

```
int64
```

```
mat = np.array([[5,4,3],[9,7,6]])
print(mat)
```

```
[[5 4 3]
 [9 7 6]]
```

```
print(mat.shape) # duita row ra duita column diyo
```

```
(2, 3)
```

```
print(mat.size)
```

```
6
```

```
print(mat.ndim)
```

```
2
```

Slicing/Indexing

```
#indexing (particular element array bata nikalna)
x = np.array([4,5,6,7,15])
print(x)
```

```
[ 4  5  6  7 15]
```

```
print(x[1])
```

```
5
```

```
#slicing array ko kei part or portion matra chahiyo
#syntax: x[startindex : stopindex]
##stop indexing exclusive ie nadiye pani hunxa

x = np.array([5,7,8,2,9,15])
print(x[0:3])
```

```
[5 7 8]
```

```
print(x[1:5])
```

```
[7 8 2 9]
```

```
print(x[1:]) # stop index diyena vane jata samma xa array tei samma dinxa
```

```
[ 7  8  2  9 15]
```

```
#slicing syntax: x[startindex: stopindex: step] BY default step is 1
```

```
x = np.array([5,7,8,2,9,15])
print(x[:6: 2])# 2 step le hidyo
```

```
[5 8 9]
```

```
x[::]
```

```
array([ 5,  7,  8,  2,  9, 15])
```

```
print(x[::-1])
```

```
[15  9  2  8  7  5]
```

```
x[1:-1]
```

```
array([7, 8, 2, 9])
```

```
x[-2:1] #by default +1 step size dinxa step lai -1 garaune
```

```
array([], dtype=int64)
```

```
x[-2:1:-1]
```

```
array([9, 2, 8])
```

```
#modification
print(x)
```

```
[ 5  7  8  2  9 15]
```

```
# 2 lai 12 banaune  existing data ma
x[3]= 12
print(x)
```

```
[ 5  7  8 12  9 15]
```

```
#copy
```

```
y = x[0:3]
print(y)
```

```
[5 7 8]
```

```
y[0]= 15
print(y)
```

```
[15  7  8]
```

```
print(x)
```

```
[15  7  8 12  9 15]
```

Start coding or generate with AI.

```
# x  ma pani modify vayo y change garda x bata banako y numpy array dependent xan ie


#copy view jun y banayin tyo hamle x ko view matra banako y
#independ tarika le xuttai banauna xa changes nauna lai vane y i


#y lai independet banaune x bata

x = np.array([5,7,8,2,9,15])

y = x[0:3].copy() # copy garera y lai independ banaune
print(y)
```

```
[5 7 8]
```

```
y[0]= 15
print(y)
```

```
[15  7  8]
```

```
print(x) # aba chai x ma pani change ayena
```

```
[ 5  7  8  2  9 15]
```

```
#negative indexing last ko element -1 bata suru hunxa both direction ma hunxa ie forward and backward
print(x[-1])
```

```
15
```

Start coding or generate with AI.

```
#Functions:
np.sum()
np.prod()

np.min()
np.max()

np.mean()
np.std()
np.median()
np.sort()
np.medium
#numpy array ma garna milne functions haru


x = np.array([5,7,8,9,15])


sum = np.sum(x)
print(f'Sum={sum}')
```

```
Sum=49
```

```
#x.shape xa vane numpy koarray lai call garne



product = np.prod(x)
print(f'Product={product}')
```

```
Product=189000
```

```
minimum = np.min(x)
print(f'Minimum={minimum}')
```

```
Minimum=5
```

```
maximum = np.max(x)
print(f'Maximun={maximum}')
```

```
Maximun=15
```

```
avg = np.mean(x)
print(f'Avg={avg}')
```

```
Avg=8.8
```

```
print(f'Avg={avg:.2f}')
```

```
Avg=8.80
```

```
SD = np.std(x)
print(f'Sum= {sum}')
```

```
Sum= 44
```

```
median = np.median(x)
print(f'Median={median}')
```

```
Median=7.5
```

```
Sort = np.sort(x)[::-1]# ascending order
print(f' Sorting={Sort}')
```

```
 Sorting=[15  9  8  7  5  5]
```

```
Start coding or generate with AI.
```

```
Sort = np.sort(x)# ascending order
print(Sort)
```

```
x = np.array([5,5,7,8,9,15])

unique = np.unique(x)
print(f'Unique={unique}')
```

```
Unique=[ 5  7  8  9 15]
```

```
help(np.sort) #documentation
```

```
Help on _ArrayFunctionDispatcher in module numpy:

sort(a, axis=-1, kind=None, order=None, *, stable=None)
    Return a sorted copy of an array.

    Parameters
    ----------
    a : array_like
        Array to be sorted.
    axis : int or None, optional
        Axis along which to sort. If None, the array is flattened before
        sorting. The default is -1, which sorts along the last axis.
    kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, optional
        Sorting algorithm. The default is 'quicksort'. Note that both 'stable'
        and 'mergesort' use timsort or radix sort under the covers and,
        in general, the actual implementation will vary with data type.
        The 'mergesort' option is retained for backwards compatibility.
    order : str or list of str, optional
        When `a` is an array with fields defined, this argument specifies
        which fields to compare first, second, etc.  A single field can
        be specified as a string, and not all fields need be specified,
        but unspecified fields will still be used, in the order in which
        they come up in the dtype, to break ties.
    stable : bool, optional
        Sort stability. If ``True``, the returned array will maintain
        the relative order of ``a`` values which compare as equal.
        If ``False`` or ``None``, this is not guaranteed. Internally,
        this option selects ``kind='stable'``. Default: ``None``.

        .. versionadded:: 2.0.0

    Returns
    -------
```

```
            sorted_array : ndarray
                Array of the same type and shape as `a`.

            See Also
            --------
            ndarray.sort : Method to sort an array in-place.
            argsort : Indirect sort.
            lexsort : Indirect stable sort on multiple keys.
            searchsorted : Find elements in a sorted array.
            partition : Partial sort.

            Notes
            -----
            The various sorting algorithms are characterized by their average speed,
            worst case performance, work space size, and whether they are stable. A
            stable sort keeps items with the same key in the same relative
            order. The four algorithms implemented in NumPy have the following
            properties:

            =========== ======= ============= ============ ========
               kind      speed   worst case    work space   stable
            =========== ======= ============= ============ ========
            'quicksort'    1     O(n^2)            0          no
            'heapsort'     3     O(n*log(n))       0          no
            'mergesort'    2     O(n*log(n))       n/2        yes
```

```
#mathematical operations duita or dui bhanda badi numpy array ma


#Array Arithmetic  # duita
np.add()      +
np.multiple   *
np.divide     /
np.floor_divide
np.mod()
np.power()
np.sqrt()


x = np.array([5,4,6,1])#corresponding elements ko sum ie 5 +2 ,
y = np.array([2,1,3,4])

z= np.add(x,y)
print(z)
```

```
[7 5 9 5]
```

```
#+ use garna ni sakinxa
z = x + y
print(z)
```

```
[7 5 9 5]
```

```
x = np.array([5,4,6,1])#corresponding elements ko sum ie 5 +2 ,
y = np.array([2,1,3,9,4])
z= np.add(x,y)
print(z)# size same huna paryo
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[78], line 3
      1 x = np.array([5,4,6,1])#corresponding elements ko sum ie 5 +2 ,
      2 y = np.array([2,1,3,9,4])
----> 3 z= np.add(x,y)
      4 print(z)

ValueError: operands could not be broadcast together with shapes (4,) (5,)
```

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

z = x/y
print(z)
```

```
[2.5  4.   2.   0.25]
```

```
x = np.array([5,4,6,1]) # floor divide le point pachadi ko value didaina
y = np.array([2,1,3,4])
```

```
z = np.floor_divide(x,y)
print(z)
```

```
[2 4 2 0]
```

```
z = np.mod(x,y) # remainder matra dinxa
print(z)
```

```
[1 0 0 1]
```

```
z = np.power(x,y)# x ra y lai multiple garera power dinxa  x ** y ie 5**2(2 power ma ho)
print(z)
```

```
[ 25   4 216   1]
```

```
z = np.sqrt(x)
print(z)
```

```
[2.23606798 2.         2.44948974 1.        ]
```

```
z = np.sqrt(y)
print(z)
```

```
[1.41421356 1.         1.73205081 2.        ]
```

```
#cube root lagauna xa bhane herera aune afai
```

```
#Broadcasting
```

```
- np.zeros()
- np.ones()
- np.full()
- np.arrange()
- np.linspace()
- np.random.random()
- np.random.randint()
- np.random.uniform()
#random generation
```