```
import numpy as np #importing numpy as alias name np
a = np.array([1,5,4,2])
a
```

```
     array([1, 5, 4, 2])
```

```
b = np.arange(10) #arange will distribute number in range of given value with defualt step
b
```

```
     array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
b.shape #shape ,dimension of array
```

```
     (10,)
```

```
b.ndim
```

```
     1
```

```
b = np.array([[1,2,3],[5,6,7]])
b
```

```
     array([[1, 2, 3],
            [5, 6, 7]])
```

```
b = np.array([[[1,2],[3,4]],[[4,5],[6,7]]]) #list within list within list
b
```

```
     array([[[1, 2],
             [3, 4]],

            [[4, 5],
             [6, 7]]])
```

```
b.shape
```

```
     (2, 2, 2)
```

```
len(b)
```

```
     2
```

```
b.ndim
```

```
     3
```

```
b = np.arange(1,10) #start,end,stepsize(deafult step size is 1)
b
```

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

There is another way to have number of points (start,end,no.of points)

```
c = np.linspace(0,1,10)
c
```

```
array([0.        , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
       0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.        ])
```

```
d = np.ones((3,2),dtype=int) #function argument is tuples
d
```

```
array([[1, 1],
       [1, 1],
       [1, 1]])
```

```
e = np.zeros((3,3))
e
```

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

Returning 2-D array with ones on diagonals and zeros else Where.

```
s = np.eye(3,dtype = int)
s
```

```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
```

```
s = np.eye(3,2,dtype=int)
```

```
s
```

```
array([[1, 0],
       [0, 1],
       [0, 0]])
```

```
f = np.diag([2,3,4,5])
f
```

```
array([[2, 0, 0, 0],
       [0, 3, 0, 0],
       [0, 0, 4, 0],
       [0, 0, 0, 5]])
```

```
np.diag(f)
```

```
array([2, 3, 4, 5])
```

```
np.random.rand(4) #samples from normal distribution
```

```
array([0.40166948, 0.10761877, 0.39642003, 0.54246297])
```

Data type

```
g = np.arange(10,dtype = float)
g.dtype
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-4c8813c1eaa0> in <module>()
----> 1 g = np.arange(10,dtype = float)
      2 g.dtype

NameError: name 'np' is not defined
```

SEARCH STACK OVERFLOW

```
c = np.array([1+2j,3j])
c.dtype
```

Indexing and Slicing

```
a = np.arange(10)
print(a[5])
```

```
b=a[:5:2]
b
```

```
a
```

```
n=a[::-1]
n
```

Slicing Operations Creates a View on Original array which is just a way of Accessing Array Data,If we modify array using slicing it will be reflected in original array also.

```
b[2]=10
a
```

If we use copy two location will be created

```
c = a[::2].copy()
c
```

```
array([ 0,  2, 10,  6,  8])
```

```
np.shares_memory(a,c)
```

```
False
```

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

```
array([2, 3, 4, 5])
```

```
a ** 2
```

```
array([ 1,  4,  9, 16])
```

Matrix Multiplication

```
import numpy as np
c = np.diag([1,2,3,4])
print(c.dot(c))
```

```
[[ 1  0  0  0]
 [ 0  4  0  0]
 [ 0  0  9  0]
 [ 0  0  0 16]]
```

Comparison Operator

```
a = np.array([1,2,3,4])
b = np.array([1,6,7,8])

a == b
```

```
array([ True, False, False, False])
```

```
a > b
```

```
array([False, False, False, False])
```

Array-Wise Comparison

```
a = np.array([1,2,3,4])
b = np.array([5,6,7,8])
c = np.array([1,2,3,4])

print(np.array_equal(a,b))
print(np.array_equal(a,c))
```

```
False
True
```

Logical Operators

```
a = np.array([1,2,3,4])
b = np.array([5,0,0,8])

np.logical_and(a,b)
```

```
array([ True, False, False,  True])
```

Transcendental Functions

```
a = np.arange(5)
np.sin(a)
```

```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ])
```

```
np.log(a)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarning: divi
  """Entry point for launching an IPython kernel.
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436])
```

```
np.exp(a)
```

```
array([ 1.        ,  2.71828183,  7.3890561 , 20.08553692, 54.59815003])
```

Shape Mismatch

```
#a = np.arange(4)
#a + np.array([1,2])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-cf0a6e60a4f5> in <module>()
      1 a = np.arange(4)
----> 2 a + np.array([1,2])

ValueError: operands could not be broadcast together with shapes (4,) (2,)
```

SEARCH STACK OVERFLOW

Basic Reductions

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

```
10
```

```
x = np.array([[1,0],[2,2]])
np.sum(x,axis=1)
```

```
array([1, 4])
```

```
x.min()
```

```
0
```

```
x.min(axis=0)
```

```
array([1, 0])
```

```
x = np.array([1,2,3,4])
x.argmin() #return index of minimum element
```

```
0
```

Logical Operations

```
np.all([True,True,False,True])
```

```
False
```

```
#Can be used for array Comparison

a = np.zeros((50,50))
np.any(a==0)
```

```
    True
```

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

```
    2.5
```

```
y.mean(axis =1)
```

```
    array([1.5, 3.5])
```

```
np.median(y,axis=1) #calculting median row wise
```

```
    array([1.5, 3.5])
```

```
x.std() #calculating Standard deviations
```

```
    1.118033988749895
```

Load data into numpy array object

It Describe Populations of hare and lynxes and carrots in north canada.

```
data = np.loadtxt('/content/Populations.txt')
data
```

```
    array([[ 1900., 30000.,  4000., 48300.],
           [ 1901., 47200.,  6100., 48200.],
           [ 1902., 70200.,  9800., 41500.],
           [ 1903., 77400., 35200., 38200.],
           [ 1904., 36300., 59400., 40600.],
           [ 1905., 20600., 41700., 39800.],
           [ 1906., 18100., 19000., 38600.],
           [ 1907., 21400., 13000., 42300.],
           [ 1908., 22000.,  8300., 44500.],
           [ 1909., 25400.,  9100., 42100.],
           [ 1910., 27100.,  7400., 46000.],
           [ 1911., 40300.,  8000., 46800.],
           [ 1912., 57000., 12300., 43800.],
           [ 1913., 76600., 19500., 40900.],
           [ 1914., 52300., 45700., 39400.],
           [ 1915., 19500., 51100., 39000.],
           [ 1916., 11200., 29700., 36700.],
           [ 1917.,  7600., 15800., 41800.],
           [ 1918., 14600.,  9700., 43300.],
           [ 1919., 16200., 10100., 41300.],
           [ 1920., 24700.,  8600., 47300.]])
```

```
type(data)
```

```
numpy.ndarray
```

*When Ever we see number like 77.4e3 i.e is equal to 77.4 * 10^3 *

Transpose of matrix

```
year,hares,lynxex,carrots = data.T #columns to variables
```

Printing data in row wise

```
print(year)
```

```
[1900. 1901. 1902. 1903. 1904. 1905. 1906. 1907. 1908. 1909. 1910. 1911.
 1912. 1913. 1914. 1915. 1916. 1917. 1918. 1919. 1920.]
```

```
pop = data[:,1:]
pop
```

```
array([[30000.,  4000., 48300.],
       [47200.,  6100., 48200.],
       [70200.,  9800., 41500.],
       [77400., 35200., 38200.],
       [36300., 59400., 40600.],
       [20600., 41700., 39800.],
       [18100., 19000., 38600.],
       [21400., 13000., 42300.],
       [22000.,  8300., 44500.],
       [25400.,  9100., 42100.],
       [27100.,  7400., 46000.],
       [40300.,  8000., 46800.],
       [57000., 12300., 43800.],
       [76600., 19500., 40900.],
       [52300., 45700., 39400.],
       [19500., 51100., 39000.],
       [11200., 29700., 36700.],
       [ 7600., 15800., 41800.],
       [14600.,  9700., 43300.],
       [16200., 10100., 41300.],
       [24700.,  8600., 47300.]])
```

```
np.argmax(pop,axis=0)
```

```
     array([3, 4, 0])
```

```
pop.std(axis=1)
```

```
     array([18176.23601177, 19614.67704439, 24668.33327703, 19225.21492439,
            10030.73055941,  9530.41913501,  9458.79954798, 12319.18106946,
            14923.434219  , 13472.52347888, 15759.51211879, 16967.22330456,
            18751.53327064, 23553.39088586,  5266.87763291, 13018.02169644,
            10757.4263754 , 14578.82787546, 14819.88154099, 13501.68713737,
            15873.31793363])
```

Broadcasting

Basic Operation on numpy arrays are element wise. This work on arrays of the same size.Its also possible to do arrays of different size if numpy can transform these array ,this conversion is called BroadCasting.

```
a = np.tile(np.arange(0,40,10),(3,1))
a
```

```
     array([[ 0, 10, 20, 30],
            [ 0, 10, 20, 30],
            [ 0, 10, 20, 30]])
```

```
a=a.T
a
```

```
     array([[ 0,  0,  0],
            [10, 10, 10],
            [20, 20, 20],
            [30, 30, 30]])
```

```
b = np.array([0,1,2])
```

```
a+b
```

```
     array([[ 0,  1,  2],
            [10, 11, 12],
            [20, 21, 22],
            [30, 31, 32]])
```

```
a = np.arange(0,40,10)
a.shape
```

```
     (4,)
```

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

```
(4, 1)
```

```
a
```

```
array([[ 0],
       [10],
       [20],
       [30]])
```

```
a+b
```

```
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22],
       [30, 31, 32]])
```

Array Shape Manipulation

```
#flattening

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

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
a.T
```

```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

```
a.ravel()
```

```
array([1, 2, 3, 4, 5, 6])
```

```
a
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
a.shape
```

```
(2, 3)
```

```
b = a.ravel()
```

```
b
```

```
array([1, 2, 3, 4, 5, 6])
```

```
b=b.reshape((2,3))
```

```
b
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
b[0][0]=100
b
```

```
array([[100,   2,   3],
       [  4,   5,   6]])
```

```
a
```

```
array([[100,   2,   3],
       [  4,   5,   6]])
```

It means They are at same memory location

Be aware reshape can also return copy

```
a = np.zeros((3,2))
b=a.T.reshape(6)
b
```

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

```
b[0]=100
b
```

```
array([100.,   0.,   0.,   0.,   0.,   0.])
```

```
a
```

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

In this part it return copy

Dimension Shifting

```
a = np.arange(4*3*2).reshape(4,3,2)
a
```

```
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5]],

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

       [[12, 13],
        [14, 15],
        [16, 17]],

       [[18, 19],
        [20, 21],
        [22, 23]]])
```

```
a[0][2][1]
```

```
5
```

Resizing

```
a = np.arange(4)
a.resize((8,))
```

```
a
```

```
array([0, 1, 2, 3, 0, 0, 0, 0])
```

```
#b = a
a.resize((4,))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-90-91f6a9f90ff2> in <module>()
      1 #b = a
```

Sorting Data

```
by another array in this way.
Use the np.resize function or refcheck False
```

```
a = np.array([[1,5,4,3],[8,4,5,2]])
a
```

```
array([[1, 5, 4, 3],
       [8, 4, 5, 2]])
```

```
np.sort(a,axis=1)
```

```
array([[1, 3, 4, 5],
       [2, 4, 5, 8]])
```

```
a
```

```
array([[1, 5, 4, 3],
       [8, 4, 5, 2]])
```

```
a.sort(axis=1) #inplace sorting
```

```
a
```

```
array([[1, 3, 4, 5],
       [2, 4, 5, 8]])
```

Fancy Indexing

```
a = np.array([4,2,3,7])
j = np.argsort(a)
j
```

```
array([1, 2, 0, 3])
```

```
a[j]
```

```
array([2, 3, 4, 7])
```

0s completed at 8:58 AM