# Arrays

Here we show how to use basic arrays, how to concatenate, add, change type, call type,etc

In [ ]:
```
import numpy as np
a=np.array([1, 2, 3, 4])
b=np.array([0, 0.5, 1, 1.5, 2])
print(a[0]+b[1])
print(a[[0,3]])
print(a.dtype)
c=np.array([1,4,5], dtype=np.float64)
print(c)
```

```
1.5
[1 4]
int32
[1. 4. 5.]
```

## Array Dimensions and shapes

Basically stuff like matrices and how to work with them

In [ ]:
```
A=np.array([
    [1, 2, 3],
    [4, 5, 6]
])
A.shape #basic shape of matrix
```

Out[ ]:
```
(2, 3)
```

In [ ]:
```
A.ndim #number of dimensions
```

Out[ ]:
```
2
```

In [ ]:
```
A.size #number of elements
```

Out[ ]:
```
6
```

In [ ]:
```
B=np.array([
    [
        [1, 2, 13],
        [21, 2, 3]
    ],
    [
        [32, 23, 12],
        [11, 1, 2]
    ]
])
print(B)
```

```
[[[ 1  2 13]
  [21  2  3]]

 [[32 23 12]
  [11  1  2]]]
```

In [ ]:
```
B.shape
```

Out[ ]:
```
(2, 2, 3)
```

In [ ]:
```
B.ndim
```

Out[ ]:
```
3
```

In [ ]:
```
B.size
```

Out[ ]:
```
12
```

## Indexing and Slicing of Matrices

Now all we need to do is account for the varying size and dimensions

In [ ]:
```
A= np.array([
    # 0 1 2
    [1, 2, 3], #0
    [4, 5, 6], #1
    [7, 8, 9]  #2
])
```

In [ ]:
```
A[1]
```

Out[ ]:
```
array([4, 5, 6])
```

In [ ]:
```
A[0][1]
```

Out[ ]:
```
2
```

In [ ]:
```
A[0,1] #this is the same as before, but it will allow some sort of splicing advant
```

Out[ ]:
```
2
```

In [ ]:
```
A[0:2]
```

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

In [ ]:
```
A[:, :2] #I want every row, but only the 0th and 1st numbers
```

Out[ ]:
```
array([[1, 2],
       [4, 5],
       [7, 8]])
```

In [ ]:
```
A[1]=np.array([10,10,10]) #modifying array
A
```

Out[ ]:
```
array([[ 1,  2,  3],
       [10, 10, 10],
       [ 7,  8,  9]])
```

## Summary and Statistics

Some Mathematical tools

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

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

In [ ]:
```
x.sum()
```

Out[ ]:
```
21
```

In [ ]:
```
x.mean()
```

Out[ ]:
```
3.5
```

In [ ]:
```
x.std()
```

Out[ ]:
```
1.707825127659933
```

In [ ]:
```
x.var()
```

Out[ ]:
```
2.9166666666666665
```

In [ ]:
```
A.sum() #working with a 2D array
```

Out[ ]:
```
60
```

In [ ]:
```
A.mean()
```

Out[ ]:
```
6.666666666666667
```

Now, we can also do a sum along the axis. Like if we have an array like *A* then we can say that it is a 2D array, wherein **axis=0** means columns and **axis=1** means rows

In [ ]:
```
A.sum(axis=0) #sum of the 1st,2nd and 3rd columns
```

Out[ ]:
```
array([18, 20, 22])
```

# NUMPY Operations

## BROADCASTING AND VECTORIZED OPERATIONS

In [ ]:
```
y=np.arange(4)
y
```

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

See whatever we do to *y* gets applied to all the elements within it. Also, an important point to remember is that we are creating a new array and not modifying the already existing one!

In [ ]:
```
y+10
```

Out[ ]:
```
array([10, 11, 12, 13])
```

In [ ]:
```
y*10
```

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

In [ ]:
```
y #see how it didn't change
```
Out[ ]:
```
array([0, 1, 2, 3])
```
In [ ]:
```
y+=100 #but now we're modifying the actual thing
y
```
Out[ ]:
```
array([100, 101, 102, 103])
```

## Basic Addition, Subtraction, multiplication,etc can also be done

In [ ]:
```
z=np.array([10,12,13,14])
```
In [ ]:
```
z+y
```
Out[ ]:
```
array([110, 113, 115, 117])
```
In [ ]:
```
z-y
```
Out[ ]:
```
array([-90, -89, -89, -89])
```
In [ ]:
```
z*y
```
Out[ ]:
```
array([1000, 1212, 1326, 1442])
```
In [ ]:
```
z/y
```
Out[ ]:
```
array([0.1       , 0.11881188, 0.12745098, 0.13592233])
```

# NUMPY Boolean Operations

Also called *Masks*

In [ ]:
```
p=np.arange(4)
p
```
Out[ ]:
```
array([0, 1, 2, 3])
```
In [ ]:
```
p[[True, False, True, True]] #basically another way of selecting elements
```
Out[ ]:
```
array([0, 2, 3])
```
In [ ]:
```
p>=2 #this is something very powerful!
```
Out[ ]:
```
array([False, False,  True,  True])
```

Filtering or quering

In [ ]:
```
p[p>=2]
```
Out[ ]:
```
array([2, 3])
```
In [ ]:
```
p[p> p.mean()]
```

Out[ ]:
```
array([2, 3])
```

In [ ]:
```
p[~(p>p.mean())] #less than the mean
```

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

In [ ]:
```
p[(p==1)| (p==0)] #Boolen OR operator
```

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

In [ ]:
```
p[(p<=2) & (p%2==0)] #Boolean AND
```

Out[ ]:
```
array([0, 2])
```

In [ ]:
```
Q=np.random.randint(100, size=(3,3)) #with a 2D array
Q
```

Out[ ]:
```
array([[36, 34, 85],
       [ 7, 60, 13],
       [60, 45, 93]])
```

In [ ]:
```
Q[Q>30]
```

Out[ ]:
```
array([36, 34, 85, 60, 60, 45, 93])
```

# NUMPY ALGEBRA AND SIZE

In [ ]:
```
R=np.array([
    [1, 2, 3],
    [4,5,6],
    [7,8,9]
])
S=np.array([
    [6,5],
    [4,3],
    [2,1]
])
```

In [ ]:
```
R.dot(S)
```

Out[ ]:
```
array([[20, 14],
       [56, 41],
       [92, 68]])
```

In [ ]:
```
R @ S #another way to show dot product
```

Out[ ]:
```
array([[20, 14],
       [56, 41],
       [92, 68]])
```

In [ ]:
```
S.T #transposing
```

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

In [ ]:
```
S.T @ R #multiplicating after transposing
```

Out[ ]:
```
array([[36, 48, 60],
       [24, 33, 42]])
```

# Size of Objects in Memory

In [ ]:
```
import sys
```

In [ ]:
```
sys.getsizeof(1) #an integer always has a size > 24 bytes
```

Out[ ]:
```
28
```

In [ ]:
```
sys.getsizeof(10**100) #longs are even longer!!
```

Out[ ]:
```
72
```

system sizes are much longer, but numpy ones are much much smaller

In [ ]:
```
np.dtype(int).itemsize
```

Out[ ]:
```
4
```

In [ ]:
```
np.dtype(np.int8).itemsize
```

Out[ ]:
```
1
```

In [ ]:
```
np.dtype(float).itemsize
```

Out[ ]:
```
8
```

Lists are larger! but even here numpy is smaller

In [ ]:
```
sys.getsizeof(1)
```

Out[ ]:
```
28
```

In [ ]:
```
np.array([1]).nbytes
```

Out[ ]:
```
4
```

## Performance is also important

Here also we can see the superiority that numpy offers over raw python

In [ ]:
```
l=list(range(1000))
```

In [ ]:
```
u=np.array(1000)
```

In [ ]:
```
%time sum([i**2 for i in l])
```

```
Wall time: 997 µs
```

Out[ ]:
```
332833500
```

In [ ]:
```
%time np.sum(u**2)
```

```
Wall time: 0 ns
```

Out[ ]:
```
1000000
```

# SOME useful NUMPY Functions

## Random

In [ ]:
```
np.random.random(size=2)
```
Out[ ]:
```
array([0.97212076, 0.97044102])
```

In [ ]:
```
np.random.normal(size=3)
```
Out[ ]:
```
array([ 0.5012844 ,  0.20960137, -0.77556289])
```

## Arange

In [ ]:
```
np.arange(10)
```
Out[ ]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [ ]:
```
np.arange(5,10,0.1)
```
Out[ ]:
```
array([5. , 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. , 6.1, 6.2,
       6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 7. , 7.1, 7.2, 7.3, 7.4, 7.5,
       7.6, 7.7, 7.8, 7.9, 8. , 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8,
       8.9, 9. , 9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9])
```

## Reshape

In [ ]:
```
np.arange(10).reshape(2,5)
```
Out[ ]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

In [ ]:
```
np.arange(10).reshape(5,2) # reshaping is only allowed in integral factors of the s
```
Out[ ]:
```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

## Linspace

In [ ]:
```
np.linspace(0,1,5)
```
Out[ ]:
```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

In [ ]:
```
np.linspace(0,1,20,False)
```
Out[ ]:
```
array([0.  , 0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 ,
       0.55, 0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95])
```

## Zeros, Ones, Empty

In [ ]:
```
np.zeros((3,2))
```

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

In [ ]:
```
np.ones(5)
```

Out[ ]:
```
array([1., 1., 1., 1., 1.])
```

In [ ]:
```
np.ones((2,100))
```

Out[ ]:
```
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
        1., 1., 1., 1.]])
```

In [ ]:
```
np.empty((2,3))
```

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

# Identity and Eye

In [ ]:
```
np.identity(3) #identity matrix
```

Out[ ]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [ ]:
```
np.eye(8,4)
```

Out[ ]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [ ]:
```
np.eye(3,3)
```

Out[ ]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [ ]:
```
np.eye(8,4,k=-3)
```

Out[ ]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.]])
```

In [ ]:
```
np.eye(8,4,k=1)
```

Out[ ]:
```
array([[0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```