Numpy Function

In [1]:
```
import numpy as np
```

In [2]:
```
# zeros

np.zeros(4)
```

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

In [4]:
```
np.zeros((2,4))
```

Out[4]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [12]:
```
np.zeros((2,4),dtype ="int")
```

Out[12]:
```
array([[0, 0, 0, 0],
       [0, 0, 0, 0]])
```

In [14]:
```
# ones

np.ones(4,dtype="int")
```

Out[14]:
```
array([1, 1, 1, 1])
```

In [16]:
```
np.ones((4,2),dtype="int")
```

Out[16]:
```
array([[1, 1],
       [1, 1],
       [1, 1],
       [1, 1]])
```

In [28]:
```
np.ones((4,2),dtype="int") * 2 * 5 *5
```

Out[28]:
```
array([[50, 50],
       [50, 50],
       [50, 50],
       [50, 50]])
```

In [33]:
```
# full
np.full((3,3),2)
```

Out[33]:
```
array([[2, 2, 2],
       [2, 2, 2],
       [2, 2, 2]])
```

In [39]:
```
z = np.identity(4,dtype = "int")
z
```

Out[39]:
```
array([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]])
```

In [40]:
```
np.diag(z)
```

Out[40]:
```
array([1, 1, 1, 1])
```

Array Indexing

In [42]:
```
a = np.array(range(1,11))
a
```

Out[42]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [45]:
```
print(a[0])
print(a[4])
```

```
1
5
```

In [49]:
```
print(a[1:2])
print(a[:-1])
print(a[-1])
```

```
[2]
[1 2 3 4 5 6 7 8 9]
10
```

In [ ]:
```
# deep copy and swallow copy
```

Deep copy

In [50]:
```
a
```

Out[50]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [51]:
```
b = a.copy()
b
```

Out[51]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [52]:
```
b[:] =100
```

In [53]:
```
b
```

Out[53]:
```
array([100, 100, 100, 100, 100, 100, 100, 100, 100, 100])
```

In [54]:
```
a
```

Out[54]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

Shallow copy

In [55]:
```
c = a
```

```
In [55]: c
```

```
Out[55]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [56]: c[:] = 100
```

```
In [57]: c
```

```
Out[57]: array([100, 100, 100, 100, 100, 100, 100, 100, 100, 100])
```

```
In [58]: a
```

```
Out[58]: array([100, 100, 100, 100, 100, 100, 100, 100, 100, 100])
```

Array Math

```
In [59]: a
```

```
Out[59]: array([100, 100, 100, 100, 100, 100, 100, 100, 100, 100])
```

```
In [61]: a - 99
```

```
Out[61]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
In [62]: a + 10
```

```
Out[62]: array([110, 110, 110, 110, 110, 110, 110, 110, 110, 110])
```

```
In [63]: a * 10
```

```
Out[63]: array([1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000])
```

```
In [64]: a /10
```

```
Out[64]: array([10., 10., 10., 10., 10., 10., 10., 10., 10., 10.])
```

```
In [65]: c
```

```
Out[65]: array([100, 100, 100, 100, 100, 100, 100, 100, 100, 100])
```

```
In [67]: # adding two array
         a + c
```

```
Out[67]: array([200, 200, 200, 200, 200, 200, 200, 200, 200, 200])
```

```
In [68]: np.sqrt(a)
```

```
Out[68]: array([10., 10., 10., 10., 10., 10., 10., 10., 10., 10.])
```

```
In [69]: np.square(a)
```

```
Out[69]: array([10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
                10000])
```

Matrix multiplication

table -- row , colum ...matrix

In NumPy, a matrix is essentially a two-dimensional array, specifically represented by the numpy.ndarray object.

In [70]:
```
m1 = np.array([[1,2],[3,4]])
m2 = np.array([[5,6],[7,8]])
```

In [71]:
```
m1.shape
```

Out[71]:
```
(2, 2)
```

In [72]:
```
m1.ndim
```

Out[72]:
```
2
```

In [73]:
```
m1 * m2  # you can use all type of arithmatic operation on the matrices
```

Out[73]:
```
array([[ 5, 12],
       [21, 32]])
```

In [74]:
```
m1 +  m2
```

Out[74]:
```
array([[ 6,  8],
       [10, 12]])
```

In [75]:
```
m1 -   m2
```

Out[75]:
```
array([[-4, -4],
       [-4, -4]])
```

In [77]:
```
m1 / m2
```

Out[77]:
```
array([[0.2       , 0.33333333],
       [0.42857143, 0.5       ]])
```

dot matrix multiplication

In [76]:
```
np.dot(m1,m2)
```

Out[76]:
```
array([[19, 22],
       [43, 50]])
```

In [ ]:
```
1  2          5    6

3  4          7    8


=  19   22
   43   50
```

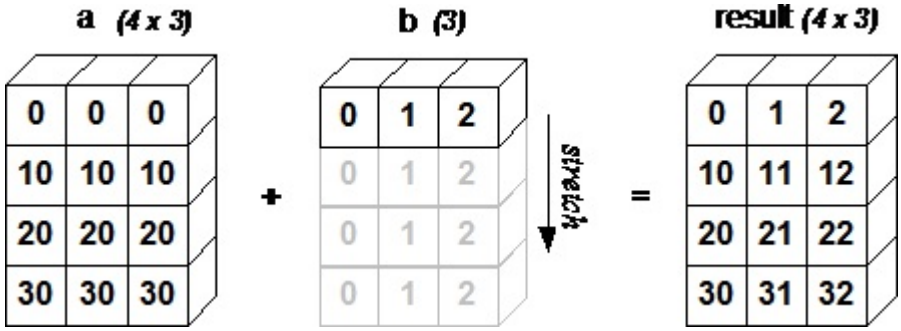
```
1 * 5 + 2 * 7  = 19
1 * 6 + 2 * 8   = 22

3 * 5  + 4 * 7 = 43
3 * 6 + 4 * 8 =  50
```

Broadcasting

In [ ]:
```
# Rules

The dimensions have the same size.
One of the dimensions has a size of 1.
```



In [78]:
```
a  = np.array([1,2,3])

b  = np.array([4,5,6])

a * b
```

Out[78]:
```
array([ 4, 10, 18])
```

In [79]:
```
c = np.array([[2,3,5],[4,5,7]]) # 2d
d = np.array([2,3,4]) # 1
c * d
```

Out[79]:
```
array([[ 4,  9, 20],
       [ 8, 15, 28]])
```

In [ ]:
```

```