In [5]:

```
import numpy as np
```

In [6]:

```
a= np.array([1,2,3])
b= np.array([10,20,30])
```

In [7]:

```
print(a+b)
```

```
[11 22 33]
```

In [8]:

```
print(b**a)
```

```
[   10   400 27000]
```

In [9]:

```
a.shape
```

Out[9]:

```
(3,)
```

In [10]:

```
a.size
```

Out[10]:

```
3
```

In [11]:

```
a.ndim
```

Out[11]:

```
1
```

In [12]:

```
a+b
```

Out[12]:

```
array([11, 22, 33])
```

here in the background some vectorized operations are going on. a[0] is also replaced by getitem(0). so this square brackets are also called syntactic sugar because it gives us simple and sweet way to write down the code.

In [13]:

```
a
```

Out[13]:

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

In [14]:

```
a*10
```

Out[14]:

Out[14]:

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

In [15]:

```
a
```

Out[15]:

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

In [16]:

```
#universal functions are also called ufuncs
```

In [17]:

```
a.fill(10.1)
a
```

Out[17]:

```
array([10, 10, 10])
```

In [18]:

```
type(a)
```

Out[18]:

```
numpy.ndarray
```

In [19]:

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

Out[19]:

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

In [20]:

```
a.size
```

Out[20]:

```
8
```

In [21]:

```
a.shape
```

Out[21]:

```
(2, 4)
```

In [22]:

```
a.ndim
```

Out[22]:

```
2
```

In [23]:

```
a[1]
```

Out[23]:

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

In [24]:

```
a[0]
```

Out[24]:

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

In [25]:

```
a[0,3]=100
```

In [26]:

```
a
```

Out[26]:

```
array([[  1,   2,   3, 100],
       [  1,   2,   3,   4]])
```

In [27]:

```
a[1, 2:4]
```

Out[27]:

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

In [28]:

```
a[:,3]
```

Out[28]:

```
array([100,   4])
```

In [29]:

```
a= np.arange(25).reshape(5,5)
```

In [30]:

```
a
```

Out[30]:

```
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, 24]])
```

In [31]:

```
timeit(a[1:4:2,:3:2])
```

```
1.1 µs ± 77.2 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

In [32]:

```
a=np.array([12,0,-1,3,5,-5,-6])
a<0
```

Out[32]:

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

In [33]:

```
mask=np.array([0,1,0,1,0,1,0], dtype=bool)
y= a[mask]
print(y)
```

```
[ 0  3 -5]
```

In [34]:

```
a = np.arange(55).reshape(11,5)
```

In [35]:

```
a
```

Out[35]:

```
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, 24],
       [25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34],
       [35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44],
       [45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54]])
```

In [36]:

```
a[[0,1,2,3],[0,2,3,4]]
```

Out[36]:

```
array([ 0,  7, 13, 19])
```

In [37]:

```
a[3:,[1,3,4]]
```

Out[37]:

```
array([[16, 18, 19],
       [21, 23, 24],
       [26, 28, 29],
       [31, 33, 34],
       [36, 38, 39],
       [41, 43, 44],
       [46, 48, 49],
       [51, 53, 54]])
```

In [39]:

```
mask=np.array([1,0,1,0,1,0,1,1,1,1,1],dtype=bool)
a[mask,2]
```

Out[39]:

```
array([ 2, 12, 22, 32, 37, 42, 47, 52])
```

In [43]:

```
a*np.nan #not a number so we use if there is a value which is not a nunber and want get r
id of it.
```

Out[43]:

```
array([[nan, nan, nan, nan, nan],
       [nan, nan, nan, nan, nan],
       [nan, nan, nan, nan, nan],
       [nan, nan, nan, nan, nan],
       [nan, nan, nan, nan, nan],
       [nan, nan, nan, nan, nan],
       [nan, nan, nan, nan, nan],
       [nan, nan, nan, nan, nan],
       [nan, nan, nan, nan, nan],
       [nan, nan, nan, nan, nan],
```

```
       [nan, nan, nan, nan, nan]])
```

In [69]:

```
a= np.ones((5,6))
```

In [72]:

```
timeit a.flatten()
```

```
1.77 µs ± 90.2 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

In [73]:

```
timeit a.ravel()
```

```
404 ns ± 84.1 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

In [61]:

```
a.sum(axis=0)
```

Out[61]:

```
array([5., 5., 5., 5., 5., 5.])
```

In [ ]:

```

```

how to find location of values using where functions, flatten(safe method) , ravel (efficient)

In [66]:

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

In [68]:

```
a.argmin()
```

Out[68]:

```
0
```