Basic Operations

- Arithmetic operators on arrays apply elementwise

```
>>> a = np.array([20, 30, 40, 50])
>>> b = np.arange(4)
>>> b
array([0, 1, 2, 3])
>>> c = a - b
>>> c
array([20, 29, 38, 47])
>>> b ** 2
array([0, 1, 4, 9])
>>> 10 * np.sin(a)
array([9.1294, -9.8803, 7.4511, -2.6237])
>>> a <= 35
array([True, True, False, False], dtype=bool)
```

Basic Operations

- Compare with lists

```
>>> a = [20, 30, 40, 50]
>>> b = list(range(4))
>>> b
[0, 1, 2, 3]
>>> a - b
???
>>> a + b
???
>>> a * 2
???
```

Basic Operations

- Multiplication

```
>>> A = np.array([[1,1],
...               [0,1]])
>>> B = np.array([[2,0],
...               [3,4]])

>>> A * B           # elementwise product
array([[2, 0],
       [0, 4]])
>>> np.dot(A, B)    # matrix product
array([[5, 4],
       [3, 4]])
```

Basic Operations

- Statistics

```
>>> a = np.array([[0,1], [2,3]])
>>> a
array([[0, 1],
       [2, 3]])
>>> np.sum(a)
6
>>> np.min(a)
0
>>> np.max(a)
3
>>> np.mean(a)
1.5
>>> np.median(a)
1.5
```

Basic Operations

- Statistical operations along one dimension

```
>>> a = np.array([[0,1], [2,3]])
>>> a
array([[0, 1],
       [2, 3]])

>>> np.sum(a, axis=0) #sum of each column
array([2, 4])
>>> np.sum(a, axis=1) #sum of each row
array([1, 5])
```

Basic Operations

```
>>> a = np.array([[0,1],
...               [2,3]])

>>> np.exp(a) #Euler exponential
array([[  1.      ,   2.7182],
       [  7.3890 ,  20.0855]])

>>> np.sqrt(a)
array([[ 0.    ,  1.    ],
       [ 1.4142,  1.7320]])
```

Basic Operations

```
>>> a = np.array([[0,1],
...               [2,3]])

>>> np.std(a)
1.118033988749895

>>> np.var(a)
1.25

>>> np.transpose(a)
array([[0, 2],
       [1, 3]])
```

Linear Algebra Operations

```
>>> a = np.array([[0,1],
...               [2,3]])

>>> np.linalg.inv(a) #Find the inverse matrix
array([[-1.5,  0.5],
       [ 1. ,  0. ]])
```

Linear Algebra Operations

```
>>> a = np.array([[0,1],
...               [2,3]])

>>> y = np.array([[5], [7]])
>>> y
array([[5],
       [7]])

>>> np.linalg.solve(a,y) #Solve ax = y for x
array([[-4.],
       [ 5.]])
```

$$\boxed{\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \end{bmatrix}}$$

Linear Algebra Operations

```
>>> a = np.array([[0,1],
...               [2,3]])

>>> np.linalg.det(a) #Find the determinant
-2
```

$$\begin{vmatrix}0 & 1\\ 2 & 3\end{vmatrix}$$

Linear Algebra Operations

```
>>> a = np.array([[0,1],
...               [2,3]])

>>> np.diag(a) #Find the diagonal elements
array([0, 3])

>>> np.trace(a) #Sum of the diagonal elements
3
```

Other Linear Algebra Operations

eig	Compute the eigenvalues and eigenvectors of a square matrix
pinv	Compute the Moore-Penrose pseudo-inverse of a matrix
qr	Compute the QR decomposition
svd	Compute the singular value decomposition (SVD)
lstsq	Compute the least-squares solution to Ax = b

Random Number Generator

```
>>> import numpy as np
>>> samples = np.random.normal(size=(4, 4))
>>> samples
array([[ 0.1241,  0.3026,  0.5238,  0.0009],
       [ 1.3438, -0.7135, -0.8312, -2.3702],
       [-1.8608, -0.8608,  0.5601, -1.2659],
       [ 0.1198, -1.0635,  0.3329, -2.3594]])
```

Random Number Generator

```
>>> import numpy as np
>>> samples = np.random.randn(4, 4)
>>> samples
array([[-0.2075, -0.5609,  0.9158,  0.3240],
       [-0.8801,  0.2645,  0.6001,  1.5696],
       [ 0.6339, -1.2218, -1.0477,  0.0378],
       [-0.8984,  0.5321,  0.3640, -2.3160]])
```

Random Number Generator

```
>>> import numpy as np
>>> samples = np.random.randn(100, 100)

>>> samples.mean()
-0.0066885831082848764
>>> samples.std()
0.99315125287072004
```

Random Number Generator

```
>>> a = list(range(5))
>>> a
[0, 1, 2, 3, 4]

>>> np.random.permutation(a)
array([2, 4, 0, 1, 3])
>>> np.random.permutation(a)
array([0, 2, 3, 4, 1])
>>> np.random.permutation(a)
array([2, 1, 3, 0, 4])
```

Other Random Number Generator

seed	Seed the random number generator
shuffle	Randomly permute a sequence in place
rand	Draw samples from a uniform distribution
randint	Draw random integers from a given low-to-high range
binomial	Draw samples from a binomial distribution
beta	Draw samples from a beta distribution
chisquare	Draw samples from a chi-square distribution
gamma	Draw samples from a gamma distribution
uniform	Draw samples from a uniform [0, 1) distribution

Elementwise Logic Operations

```
>>> np.all([True, True, False])  #all true?
False

>>> np.any([True, True, False])  #any true?
True
```

Elementwise Logic Operations

```
>>> a = np.zeros((10, 10))

>>> np.all(a == 0)
True

>>> np.any(a != 0)
False
```

Sorting

```
>>> a = np.random.randn(5)
>>> a
array([ 1.1120,  1.1199, -0.7130,  0.6764,
0.4493])

>>> a.sort()
>>> a
array([-0.7130,  0.4493,  0.6764,  1.1120,
1.1199])
```

- Sorting is done in-place

Sorting

```
>>> b = np.random.randn(3, 3)
>>> b
array([[-1.1703, -1.0639, -1.2858],
       [-1.0607, -0.1168, -1.0546],
       [ 0.0695, -0.2336, -0.4219]])
>>> b.sort(axix=0)  #along each column
>>> b
array([[-1.1703, -1.0639, -1.2858],
       [-1.0607, -0.2336, -1.0546],
       [ 0.0695, -0.1168, -0.4219]])
>>> b.sort(axis=1)  #along each row
>>> b
array([[-1.2858, -1.1703, -1.0639],
       [-1.0607, -1.0546, -0.2336],
       [-0.4219, -0.1168,  0.0695]])
```

Row and Column Vectors

- Numpy does not differentiate between 1D row and column vectors

```
>>> np.array([1,2,3])
array([1, 2, 3])

>>> np.array([1,2,3]).transpose()
array([1, 2, 3])
```

Row and Column Vectors

- A column vector can only be represented in 2D

```
>>> b = np.array([[1], [2], [3]])
>>> b
array([[1],
       [2],
       [3]])
>>> b.transpose()
array([[1, 2, 3]])
>>> b
array([[1],
       [2],
       [3]])
```

- Transposing is not done in-place

Row and Column Vectors

- Can also use `reshape` to make a 2D array

```
>>> np.array([1,2,3]).reshape(3,1)
array([[1],
       [2],
       [3]])
```

Row and Column Vectors

- `r_` and `c_` create arrays by stacking numbers along one axis

```
>>> a = np.r_[-2, -1, 1:4]
>>> a
array([-1, -2, 1, 2, 3])
>>> np.c_[a]
array([[-2],
       [-1],
       [1],
       [2],
       [3]])
```

Indexing

```
>>> a = np.arange(6).reshape(2,3)
array([[0, 1, 2],
       [3, 4, 5]])
>>> a[1]
array([3, 4, 5])
>>> a[1][1]
4
>>> a[1,1]    # preferred
4
```

Indexing

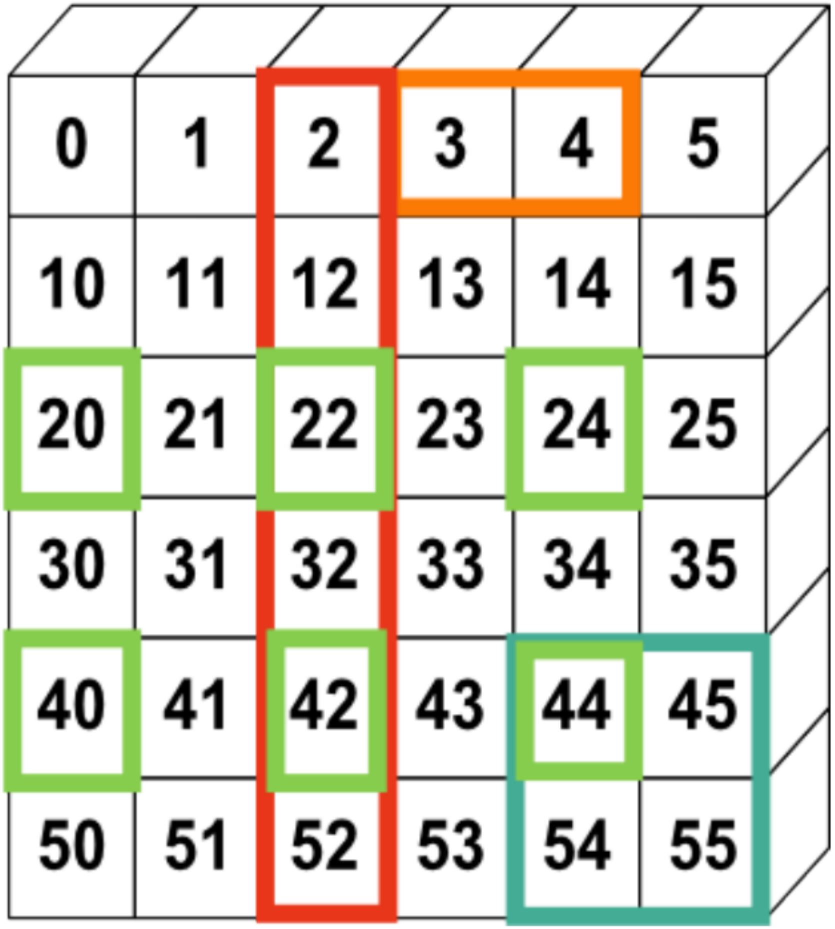
```
>>> b=[[0,1,2],[3,4,5]]
>>> b[1]
[3, 4, 5]
>>> b[1][1]
4
>>> b[1,1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers or 
slices, not tuple
```

Indexing

```
>>> a = np.arange(5) ** 2
>>> a
array([ 0,  1,  4,  9, 16])
>>> a[2]
4
>>> a[2:5]
array([ 4,  9, 16])
>>> #from start to 4, set every 2nd element to -100
>>> a[:4:2] = -100
>>> a
array([-100,    1, -100,    9,   16])
>>> a[::-1]    # reverse
array([  16,    9, -100,    1, -100])
```

Indexing

```
>>> a[0,3:5]
array([3,4])


>>> a[4:,4:]
array([[44, 45],
       [54, 55]])


>>> a[:,2]
array([2,12,22,32,42,52])


>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Fancy Indexing

- Indexed by arrays of integers and arrays of Booleans

```
>>> a = np.arange(5)**2     # data array
>>> a
array([0,  1,  4,  9, 16])
>>> i = np.array([3, 1, 3, 0]) # index array
>>> a[i]
array([9,  1,  9, 0])
```

Fancy Indexing

```
>>> a = np.arange(5)**2    # data array
>>> a
array([0,  1,  4,  9, 16])
>>> j = np.array([[0,3], [2,1]])  # index array
>>> j
array([[0, 3],
       [2, 1]])
>>> a[j]
array([[0, 9],
       [4, 1]])
```

Fancy Indexing

```
>>> a = np.arange(12).reshape(3,4) # data array
>>> a
array([[0,  1,  2,  3],
       [4,  5,  6,  7],
       [8,  9, 10, 11]])
>>> i = np.array([[0,1],   # index array
...               [1,2]])
>>> j = np.array([[2,1],   # index array
...               [3,3]])
>>> a[i, j]
array([[2,  5],
       [7, 11]])
>>> a[i, 2]
array([[2,  6],
       [6, 10]]
```

Fancy Indexing

- Use indexing to assign values

```
>>> a = np.arange(5)       #data array
>>> a
array([0, 1, 2, 3, 4])
>>> a[[1,3,4]] = 10     #[1,3,4] is index array
>>> a
array([0, 10, 2, 10, 10])
>>> a[1,3,4] = 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: too many indices for array
```

Fancy Indexing

- Indexing with Boolean Arrays

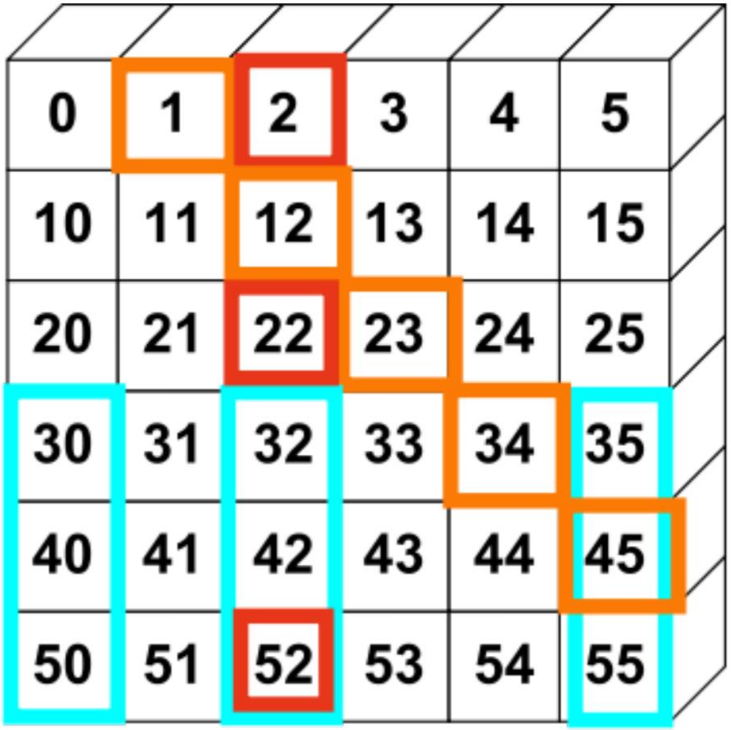
```
>>> a = np.arange(9).reshape(3,3)   #data array
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> b = a > 4                       #index array
>>> b
array([[False, False, False],
       [False, False,  True],
       [ True,  True,  True]], dtype=bool)
>>> a[b]
array([5,  6,  7,  8])
>>> a[b] = -1
>>> a
array([[ 0,  1,  2],
       [ 3,  4, -1],
       [-1, -1, -1]])
```

Fancy Indexing

```
>>> a = np.arange(9).reshape(3,3)  #data array
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])

>>> b1 = np.array([False, True, True]) #index array
>>> b2 = np.array([True, False, True]) #index array

>>> a[b1,:]
array([[3, 4, 5],
       [6, 7, 8]])
>>> a[:,b2]
array([[0, 2],
       [3, 5],
       [6, 8]])
```

Fancy Indexing

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])

>>> a[3:,[0, 2, 5]]
array([[30, 32, 35],
        [40, 42, 45]])
        [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1],
                 dtype=bool)
>>> a[mask,2]
array([2,22,52])
```



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Stacking

```
>>> a = np.array([[0,1], [2,3]])
>>> b = np.array([[0,-1], [-2,-3]])

>>> np.vstack((a,b))
array([[ 0,  1],
       [ 2,  3],
       [ 0, -1],
       [-2, -3]])

>>> np.hstack((a,b))
array([[ 0,  1,  0, -1],
       [ 2,  3, -2, -3]])
```

Splitting

```
>>> a = np.array([[0,1,2],
...               [3,4,5],
...               [6,7,8]])
>>> np.hsplit(a,3)
[array([[0],
        [3],
        [6]]),
 array([[1],
        [4],
        [7]]),
 array([[2],
        [5],
        [8]])]
```

Splitting

```
>>> a = np.array([[0,1,2],
...               [3,4,5],
...               [6,7,8]])
>>> np.vsplit(a,3)
[array([[0, 1, 2]]), array([[3, 4, 5]]),
array([[6, 7, 8]])]
```