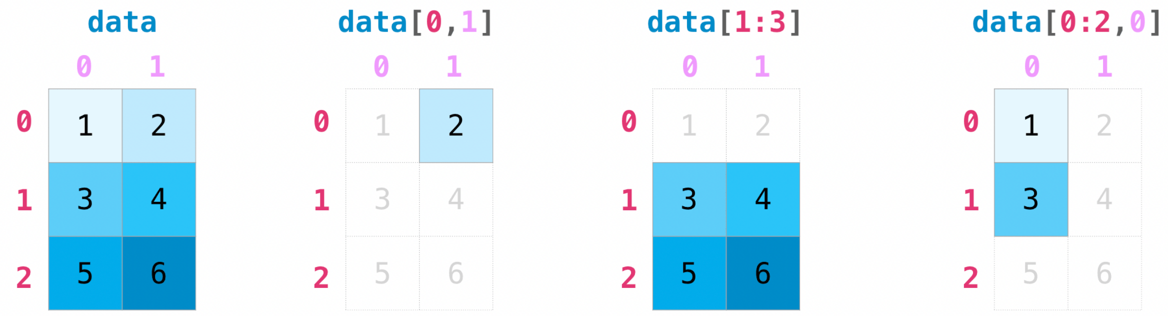
Matrix operations

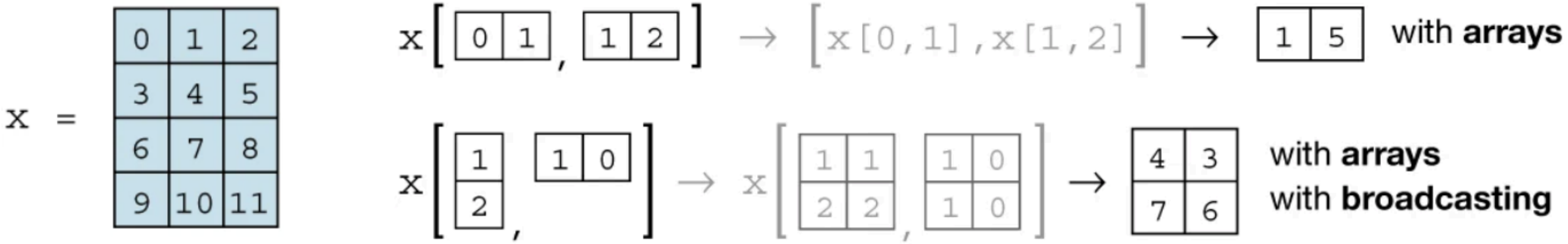
In NumPy, a matrix is simply a two-dimensional array. Operations on matrix are similar to those for 1-dimensional operations.

Creating (initializing) and indexing matrices

```
data = np.array([[1,2],[3,4],[5,6]])
b = data[0,1]
c = data[1:3]        # along the 1st dimension
d = data[0:2, 0]
e = data[[1,2,0], [1,0,1]]
```




```
x = np.array([[0,1,2],[3,4,5],[6,7,8],[9,10,11]])
y = x[     [0,1]    , [1,2]  ]
z = x[  [ [1],[2] ], [1,0]  ]
cf)  q = x[  [   [0,1]   ], [1,2]   ]  ???
```



Information about a matrix

```
ndarray.ndim          # number of axis (number of dimensions)

ndarray.shape         # number of elements in each dimension

ndarray.size          # total number of elements

np.unique()
  a_2d = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [1, 2, 3, 4]])
  unique_rows = np.unique(a_2d, axis=0)
  unique_rows, indices, occurrence_count = np.unique(
                         a_2d, axis=0, return_counts=True, return_index=True)
```

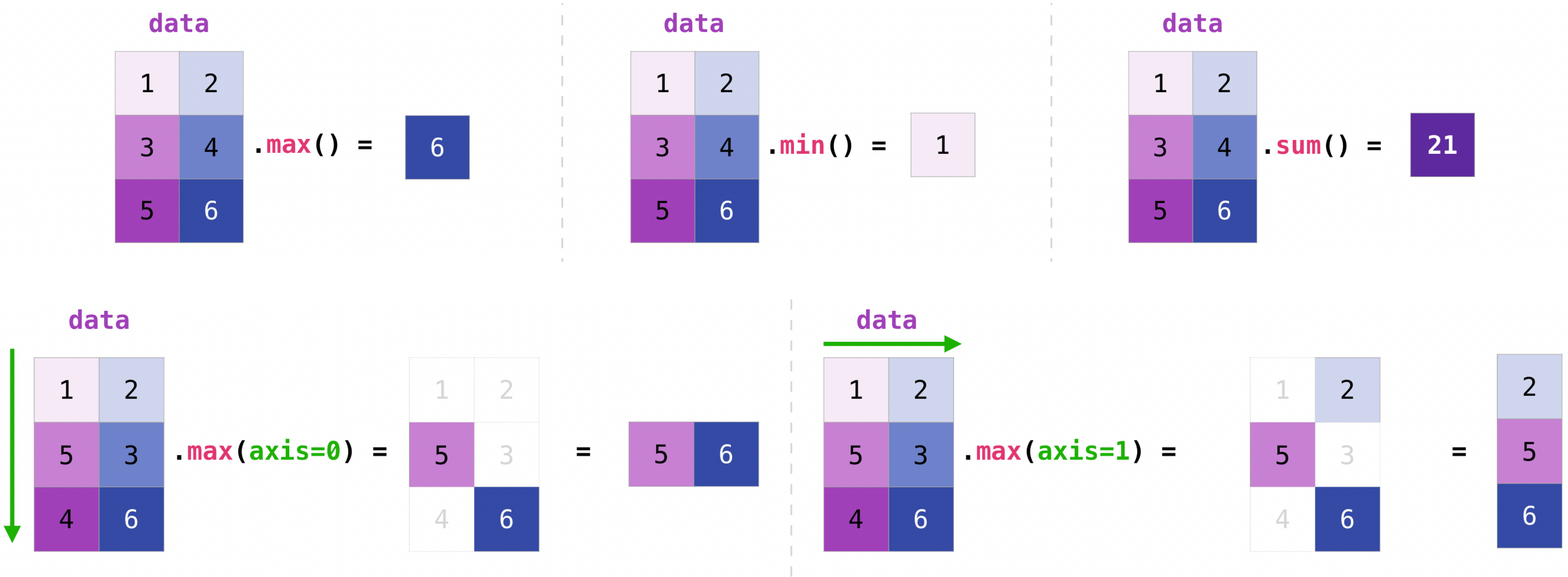
Changing the shape of an array

```
ndarray.reshape(m, n)     # change the dimension to m x n

np.newaxis                b = a[np.newaxis, :]
                          b = a[:, np.newaxis]

np.expand_dims            b = np.expand_dims(a, axis = 1)
                          b = np.expand_dims(a, axis = 0)

np.squeeze()              b = np.squeeze(a)

np.sort()                 b = np.sort(a)

np.flip()                 b = np.flip(a)  # reverse
```

Creating (initializing) a new array from existing arrays

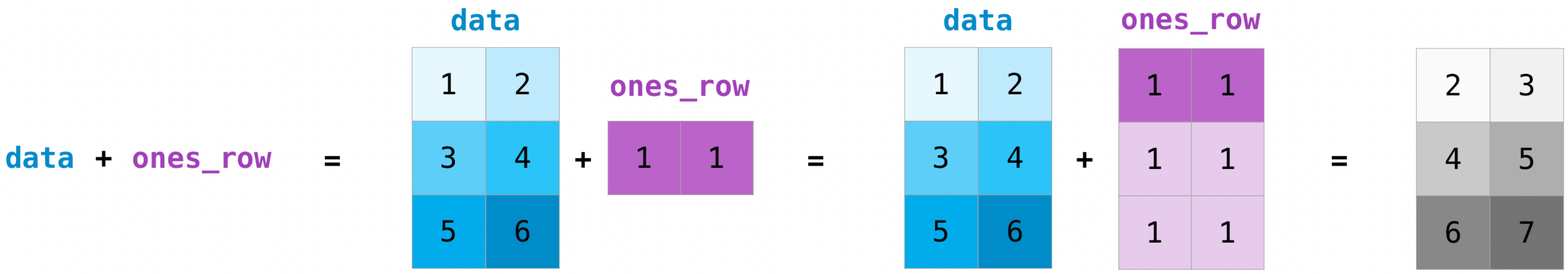
```
a = a[3:]       # removing first 3 elements

a1 = np.array([[1,1],[2,2]])
a2 = np.array([[3,3],[4,4]])
np.vstack((a1,a2))
np.hstack((a1,a2))

x = np.arange(1,25).reshape(2,12)
a,b,c = np.hsplit(x,3)
```

Operations on matrices: max(), min(), sum()
also: cumsum(), mean(), median(), std(), etc…



Operations on matrices (broadcasting)

```
data = np.array([[1,2],[3,4],[5,6]])
ones_row = np.array([[1,1]])   # Note: a nested list
b = data + ones_row
```



Operations on matrices: transposing, multiplication

```
data = np.array([[1,2],[3,4],[5,6]])
b = data.T      # equivalent to  →   b = data.transpose()
c = data @ data.T     # Matrix multiplication
```

data

1	2
3	4
5	6

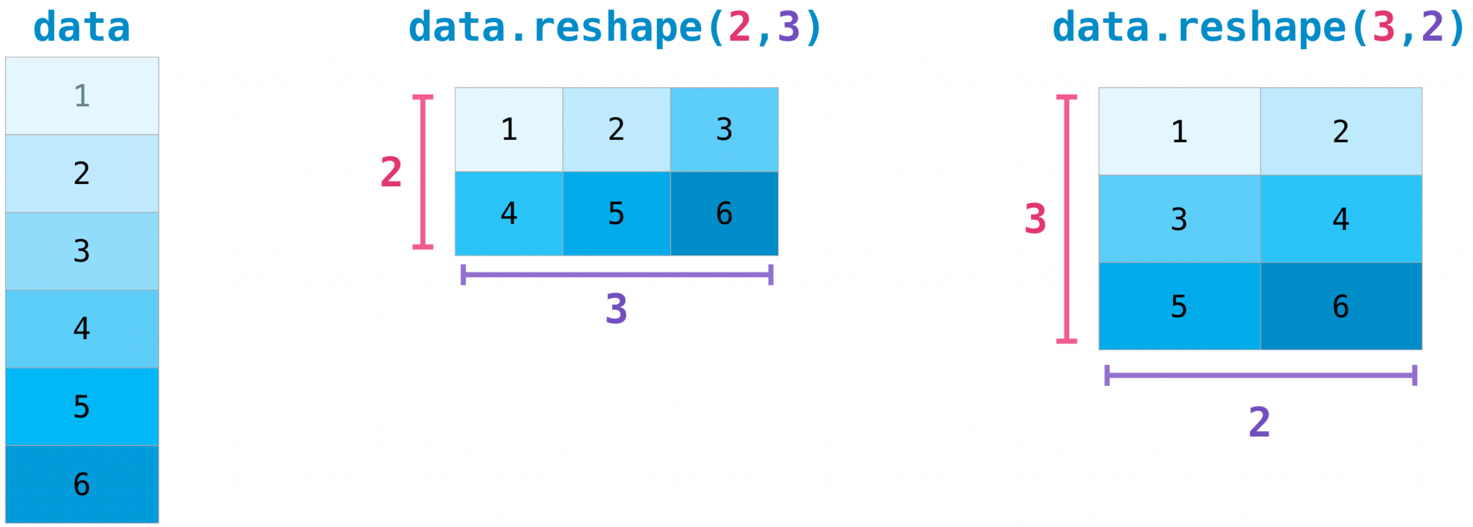
data.T

1	3	5
2	4	6

```
# Advanced transpose
x_test = np.arange(30).reshape(3, 2, 5)
print(x_test)
print(x_test.shape)
y = np.transpose(x_test, (0, 2, 1))  # axis change
print(y)
print(y.shape)
```

Operations on matrices: reshaping

```
data = np.arange(1,7)
b = data.reshape(2,3)
c = data.reshape(3,2)
e = data.reshape(3,-1)    # automatic # of cols
f = data.reshape(-1,3)    # automatic # of rows
```



Operations on matrices: reverse

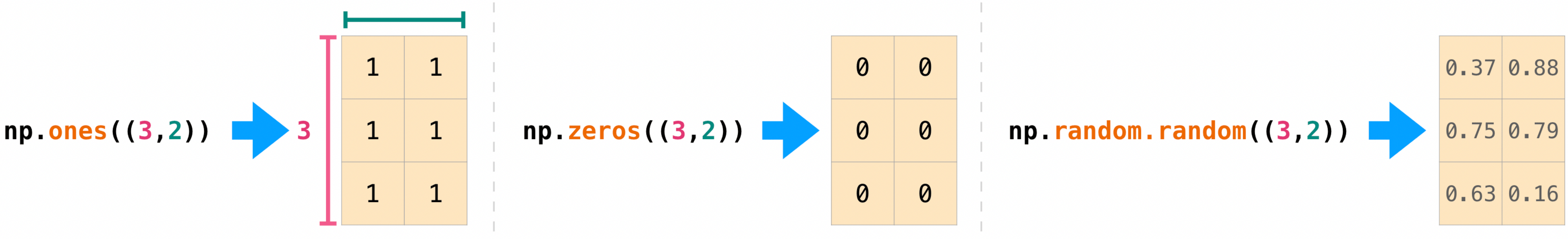
```
data = np.arange(1,7).reshape(2,3)

b = np.flip(data)

c = np.flip(data, axis = 1)  # reverses only the given dim

data[1] = np.flip(data[1])   # reverses a single row

data[:,1] = np.flip(data[:,1])   # reverses a single column
```

Tiling

```
data = np.arange(1,7).reshape(2,3)

b = np.tile(data, (3,1))

c = np.tile(data.reshape(6,1), (1,3))
```

Other ways of initializing matrices

```
a = np.ones((3,2))

a = np.zeros((3,2))

a = np.random.random((3,2))
```




```
a = np.random.randint(3, 10, size=(5,10))
a = np.random.normal(size=(6,6))
```

Let's use NumPy to evaluate a math formula

$$MeanSquareError = \frac{1}{n}\sum_{i=1}^{n}(Y\_prediction_i - Y_i)^2$$

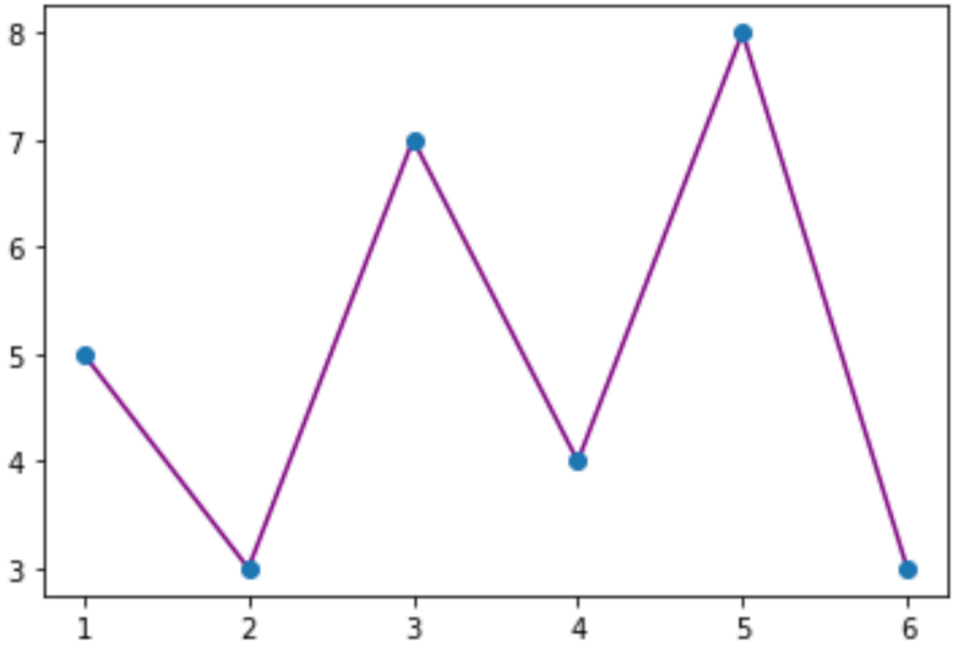
```
Y_prediction    →  y_pred
Y_i             →  yi

Then, Y_prediction – Y_i  can be written as → y_pred – yi
The square term can be broadcasted →  (y_pred-yi)**2
Summation     → ((y_pred-yi)**2).sum()
1/n           → ((y_pred-yi)**2).sum()/yi.size
Therefore →   MSE = ((y_pred-yi)**2).sum()/yi.size
```

Save and load NumPy arrays

```
a = np.array([1,2,3,4,5,6])
np.save('filename', a)
m = np.load('filename.npy')

a = np.array([1,2,3,4,5,6])
b = np.array([5,3,7,4,8,3, 1.0])
np.savez('filename', abc=a, qqq=b)
m = np.load('filename.npz')
m['abc']
m['qqq']

# Use 'savez_compressed' to compress the data
# (my take more time to save)
np.savez_compressed('filename', abc=a, qqq=b)
m = np.load('filename.npz')
```

Plot

```
%matplotlib widget  # use this for interactive plots
import matplotlib.pyplot as plt
a = np.array([1,2,3,4,5,6])
b = np.array([5,3,7,4,8,3])
plt.plot(a,b, ‘purple’)  # line color
plt.plot(a,b, ‘o’)       # dots
```



Practice, practice practice!!!

The concepts and basic operations of Numpy we covered in the last and this week is essential. You HAVE to practice every single operations I mentioned in the lectures by yourself. Otherwise, you will quickly forget them.