Importing numpy

In [3]:
```
import numpy as np
```

Here is an array with rank 1, and the length of the axis is 3:

In [4]:
```
[1,2,3]
```

Out[4]:
```
[1, 2, 3]
```

Below is an array with rank 2, and the length of the axis is 3 too:

In [5]:
```
[[ 1, 2, 3],[ 4, 5, 6]]
```

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

We can create an array of NumPy through the array function, for example:

In [7]:
```
a = np.array([1, 2, 3])
b = np.array([(1,2,3), (4,5,6)])

print("a: ",a)
print("b: ",b)
```

```
a:  [1 2 3]
b:  [[1 2 3]
 [4 5 6]]
```

In [8]:
```
a = np.array([1, 2, 3])
b = np.array([(1,2,3), (4,5,6)])

print('a=')
print(a)
print("a's ndim {}".format(a.ndim))
print("a's shape {}".format(a.shape))
print("a's size {}".format(a.size))
print("a's dtype {}".format(a.dtype))
print("a's itemsize {}".format(a.itemsize))

print('')

print('b=')
print(b)
print("b's ndim {}".format(b.ndim))
print("b's shape {}".format(b.shape))
print("b's size {}".format(b.size))
print("b's dtype {}".format(b.dtype))
print("b's itemsize {}".format(b.itemsize))
```

```
a=
[1 2 3]
a's ndim 1
a's shape (3,)
a's size 3
a's dtype int32
a's itemsize 4
```

```
b=
[[1 2 3]
 [4 5 6]]
b's ndim 2
b's shape (2, 3)
b's size 6
b's dtype int32
b's itemsize 4
```

We can also specify the type of the element when creating the array, for example:

In [10]:
```
c = np.array( [ [1,2], [3,4] ], dtype=complex )
c
```

Out[10]:
```
array([[1.+0.j, 2.+0.j],
       [3.+0.j, 4.+0.j]])
```

Create a specific array

In [12]:
```
a = np.zeros((2,3))
print('np.zeros((2,3))= \n{}\n'.format(a))

b = np.ones((2,3))
print('np.ones((2,3))= \n{}\n'.format(b))

c = np.empty((2,3))
print('np.empty((2,3))= \n{}\n'.format(c))

d = np.arange(1, 2, 0.3)
print('np.arange(1, 2, 0.3)= \n{}\n'.format(d))

f = np.random.random((2,3))
print('np.random.random((2,3))= \n{}\n'.format(f))
```

```
np.zeros((2,3))=
[[0. 0. 0.]
 [0. 0. 0.]]

np.ones((2,3))=
[[1. 1. 1.]
 [1. 1. 1.]]

np.empty((2,3))=
[[1. 1. 1.]
 [1. 1. 1.]]

np.arange(1, 2, 0.3)=
[1.  1.3 1.6 1.9]

np.random.random((2,3))=
[[0.33016184 0.01274884 0.6798199 ]
 [0.14854762 0.56535887 0.63948355]]
```

Shape and operation

In [13]:
```
zero_line = np.zeros((1,3))
one_column = np.ones((3,1))
print("zero_line = \n{}\n".format(zero_line))
print("one_column = \n{}\n".format(one_column))

a = np.array([(1,2,3), (4,5,6)])
b = np.arange(11, 20)
```

```
print("a = \n{}\n".format(a))
print("b = \n{}\n".format(b))
```

```
zero_line =
[[0. 0. 0.]]

one_column =
[[1.]
 [1.]
 [1.]]

a =
[[1 2 3]
 [4 5 6]]

b =
[11 12 13 14 15 16 17 18 19]
```

The array b is a one-dimensional array originally, and we resize it into a matrix of 3 rows and 3 columns by the reshape method:

In [28]:
```
b = np.array([11, 12, 13, 14, 15, 16, 17, 18])

b = b.reshape(2, -1)
print("b.reshape(3) = \n{}\n".format(b))
```

```
b.reshape(3) =
[[11 12 13 14]
 [15 16 17 18]]
```

Index

Basically, we can specify the subscripts by array[index] to access the elements of the array.

In [31]:
```
base_data = np.arange(100, 200)
print("base_data\n={}\n".format(base_data))

print("base_data[11] = {}\n".format(base_data[11]))
```

```
base_data
=[100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135
 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171
 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189
 190 191 192 193 194 195 196 197 198 199]

base_data[11] = 111
```

In NumPy, we can create an array containing several subscripts to get the elements in the target array. For example:

In [38]:
```
every_five = np.arange(0, 100, 5)
print("base_data[every_five] = \n{}\n".format(
    base_data[every_five]))
```

```
base_data[every_five] =
[100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185
 190 195]
```

In [42]:
```
base_data
```

Out[42]:
```
array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
       113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125,
       126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138,
       139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151,
       152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164,
       165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177,
       178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
       191, 192, 193, 194, 195, 196, 197, 198, 199])
```

In [40]:
```
a = np.array([(1,2), (10,20)])
print("a = \n{}\n".format(a))
print("base_data[a] = \n{}\n".format(base_data[a]))
```

```
a =
[[ 1  2]
 [10 20]]

base_data[a] =
[[101 102]
 [110 120]]
```

For a two-dimensional array,

- if we only specify one subscript, the result of the access is still an array.
- if we specify two subscripts, the result of the access is the elements inside.
- we can also specify the last element by "-1".

In [46]:
```
base_data2 = base_data.reshape(10,-1)
print(base_data2)
```

```
[[100 101 102 103 104 105 106 107 108 109]
 [110 111 112 113 114 115 116 117 118 119]
 [120 121 122 123 124 125 126 127 128 129]
 [130 131 132 133 134 135 136 137 138 139]
 [140 141 142 143 144 145 146 147 148 149]
 [150 151 152 153 154 155 156 157 158 159]
 [160 161 162 163 164 165 166 167 168 169]
 [170 171 172 173 174 175 176 177 178 179]
 [180 181 182 183 184 185 186 187 188 189]
 [190 191 192 193 194 195 196 197 198 199]]
```

In [45]:
```
print("base_data2[2] = \n{}\n".format(base_data2[2]))
print("base_data2[2, 3] = \n{}\n".format(base_data2[2, 3]))
print("base_data2[-1, -1] = \n{}\n".format(base_data2[-1, -1]))
```

```
base_data2[2] =
[120 121 122 123 124 125 126 127 128 129]

base_data2[2, 3] =
123

base_data2[-1, -1] =
199
```

Mathematics

There are also a lot of mathematical functions in NumPy. Here are some examples.

In [52]:
```
base_data = (np.random.random((5, 5)))*100
print("base_data = \n{}\n".format(base_data))

print("np.amin(base_data) = {}".format(np.amin(base_data)))
print("np.amax(base_data) = {}".format(np.amax(base_data)))
print("np.average(base_data) = {}".format(np.average(base_data)))
print("np.sum(base_data) = {}".format(np.sum(base_data)))
```

```
base_data =
[[ 4.85184285 63.68887591 98.45008763 30.76738546 28.80352113]
 [78.89343673 54.44684881 22.18357707 43.50308788 22.48882174]
 [ 4.98006447 20.22864129 30.79958108 57.65444409 80.75119427]
 [12.01304837 36.26225506 56.65352441  3.39283353 38.55351544]
 [25.87856673 86.16558916 63.37490057 70.9845742  56.53150353]]

np.amin(base_data) = 3.3928335338888838
np.amax(base_data) = 98.45008762867045
np.average(base_data) = 43.692068857177986
np.sum(base_data) = 1092.3017214294496
```

In [56]:
```
arr = np.arange(1,20)
print(arr)
arr = arr * arr              #Multiplies each element by itself
print("Multpiles: ",arr)
arr = arr - arr              #Subtracts each element from itself
print("Substracts: ",arr)
arr = np.arange(1,20)
arr = arr + arr              #Adds each element to itself
print("Add: ",arr)
arr = arr / arr              #Divides each element by itself
print("Divide: ",arr)
arr = np.arange(1,20)
# arr = arr + 50
# print("Add +50: ",arr)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
Multpiles:  [  1   4   9  16  25  36  49  64  81 100 121 144 169 196 225 256 289 324
 361]
Substracts:  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Add:  [ 2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38]
Divide:  [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [57]:
```
print("Sqrt: ",np.sqrt(arr))#Returns the square root of each element
print("Exp: ",np.exp(arr))     #Returns the exponentials of each element (e=2.718)
print("Sin: ",np.sin(arr))     #Returns the sin of each element
print("Cos: ",np.cos(arr))     #Returns the cosine of each element
print("Log: ",np.log(arr))     #Returns the logarithm of each element
print("Sum: ",np.sum(arr))     #Returns the sum total of elements in the array
print("Std: ",np.std(arr))     #Returns the standard deviation of in the array
```

```
Sqrt:  [1.         1.41421356 1.73205081 2.         2.23606798 2.44948974
 2.64575131 2.82842712 3.         3.16227766 3.31662479 3.46410162
 3.60555128 3.74165739 3.87298335 4.         4.12310563 4.24264069
 4.35889894]
Exp:  [2.71828183e+00 7.38905610e+00 2.00855369e+01 5.45981500e+01
 1.48413159e+02 4.03428793e+02 1.09663316e+03 2.98095799e+03
 8.10308393e+03 2.20264658e+04 5.98741417e+04 1.62754791e+05
 4.42413392e+05 1.20260428e+06 3.26901737e+06 8.88611052e+06
 2.41549528e+07 6.56599691e+07 1.78482301e+08]
Sin:  [ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427 -0.2794155
  0.6569866   0.98935825  0.41211849 -0.54402111 -0.99999021 -0.53657292
  0.42016704  0.99060736  0.65028784 -0.28790332 -0.96139749 -0.75098725
  0.14987721]
Cos:  [ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219  0.96017029
  0.75390225 -0.14550003 -0.91113026 -0.83907153  0.0044257   0.84385396
```

```
  0.90744678  0.13673722 -0.75968791 -0.95765948 -0.27516334  0.66031671
  0.98870462]
Log:  [0.         0.69314718 1.09861229 1.38629436 1.60943791 1.79175947
 1.94591015 2.07944154 2.19722458 2.30258509 2.39789527 2.48490665
 2.56494936 2.63905733 2.7080502  2.77258872 2.83321334 2.89037176
 2.94443898]
Sum:  190
Std:  5.477225575051661
```

Matrix

In [59]:

```
base_data = np.floor((np.random.random((5, 5)) - 0.5) * 100)
print("base_data = \n{}\n".format(base_data))

print("base_data.T = \n{}\n".format(base_data.T))
print("base_data.transpose() = \n{}\n".format(base_data.transpose()))

matrix_one = np.ones((5, 5))
print("matrix_one = \n{}\n".format(matrix_one))

minus_one = np.dot(matrix_one, -1)
print("minus_one = \n{}\n".format(minus_one))
print(base_data)

print("np.dot(base_data, minus_one) = \n{}\n".format(
    np.dot(base_data, minus_one)))
```

```
base_data =
[[ 37.  25.  16.  13.  22.]
 [ 33. -35. -16.  -6. -46.]
 [-35.   2.  49.  42. -10.]
 [-49.  -4.  -9.  -9.  47.]
 [ 46.  -5. -44. -45.  14.]]

base_data.T =
[[ 37.  33. -35. -49.  46.]
 [ 25. -35.   2.  -4.  -5.]
 [ 16. -16.  49.  -9. -44.]
 [ 13.  -6.  42.  -9. -45.]
 [ 22. -46. -10.  47.  14.]]

base_data.transpose() =
[[ 37.  33. -35. -49.  46.]
 [ 25. -35.   2.  -4.  -5.]
 [ 16. -16.  49.  -9. -44.]
 [ 13.  -6.  42.  -9. -45.]
 [ 22. -46. -10.  47.  14.]]

matrix_one =
[[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.]]

minus_one =
[[-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.]]

[[ 37.  25.  16.  13.  22.]
 [ 33. -35. -16.  -6. -46.]
 [-35.   2.  49.  42. -10.]
 [-49.  -4.  -9.  -9.  47.]
```

```
 [ 46.  -5. -44. -45.  14.]]
np.dot(base_data, minus_one) =
[[-113. -113. -113. -113. -113.]
 [  70.   70.   70.   70.   70.]
 [ -48.  -48.  -48.  -48.  -48.]
 [  24.   24.   24.   24.   24.]
 [  34.   34.   34.   34.   34.]]
```

Random Number

In [60]:

```
print("random: {}\n".format(np.random.random(20)));

print("rand: {}\n".format(np.random.rand(3, 4)));

print("randint: {}\n".format(np.random.randint(0, 100, 20)));

print("permutation: {}\n".format(np.random.permutation(np.arange(20))));
```

```
random: [0.46055867 0.77743642 0.01540901 0.02848748 0.49968614 0.60765879
 0.41729919 0.7167813  0.1098415  0.68637934 0.46925282 0.95760573
 0.82322318 0.79797601 0.29905994 0.58608099 0.66852858 0.80379115
 0.32612049 0.30639526]

rand: [[0.95133552 0.59222942 0.01935234 0.17646002]
 [0.69314883 0.49127927 0.40917416 0.60279785]
 [0.24544088 0.07759221 0.68174582 0.77833323]]

randint: [94 47  5 53  9 40 15 36  5 39 62 20 41  4 12 24 30 26 68 47]

permutation: [13 12  8 11  6  2  3  5  1  0 10  7 18 17 14 15 19  9 16  4]
```