Numpy

Indexing, Slicing, Arithmetic and Statistical operations, Reshaping, Iterating, Broadcasting, Sorting.

Indexing

- Numpy arrays can be indexed, sliced and iterated over.

```
a = np.array([1, 2 ,3, 4, 5])
print(a[2])  # prints 3 as indexing starts from 0
```

- We use two indices for indexing a two dimensional array

```
a = np.array([ [1, 2 ,3], [4, 5, 6] ])
print(a[0,0])  # prints 1
print(a[1,2])  # prints 6
print(a[2,0])  # Index Error as there is no 3rd row.
```

```
    0 1 2
0  [[1 2 3]
1   [4 5 6]]
```

Slicing

- We can extract some part of an array through slicing.
- We can slice an array using start and end index values [start:end:step]

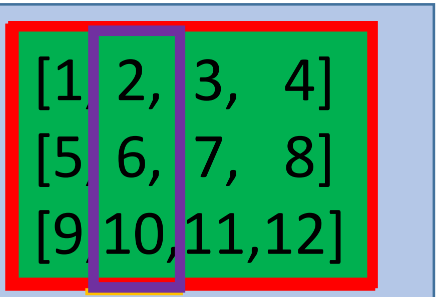
```
a = np.array([1,2,3,4,5,6,7])
print(a[2:5]) # prints [3 4 5]
```

- We can reverse the array

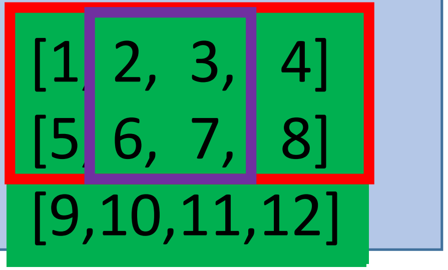
```
a = np.array([1,2,3,4,5,6,7])
print(a[::-1]) # prints [7 6 5 4 3 2 1]
```

Slicing

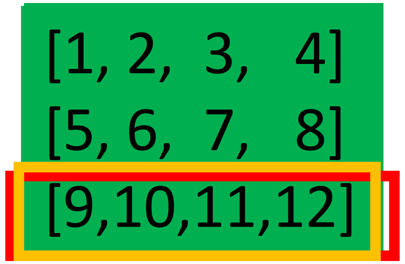
```
a = np.array( [ [1,2,3,4], [5,6,7,8], [9,10,11,12] ])
print(a[0:3,1])    # prints [ 2  6 10]
print(a[0:2, 1:3])  #prints [[2 3]
                             [6 7]]
print(a[2,:])  # prints [ 9 10 11 12]
```



```
[1, 2, 3,  4]
[5, 6, 7,  8]
[9,10,11,12]
```



```
[1, 2, 3,  4]
[5, 6, 7,  8]
[9,10,11,12]
```



```
[1, 2, 3,  4]
[5, 6, 7,  8]
[9,10,11,12]
```

Arithmetic operations

- Arithmetic operations on Numpy arrays are fast and simple.
- Basic arithmetic operations like addition, subtraction, etc. on two arrays, are done element-by-element.
- Examples

```
a1 = np.array([1,2],[3,4])
a2 = np.array([5,6],[7,8])
print(a1+a2)   # [[ 6  8] [10 12]]
print(a1-a2)    #[[-4 -4] [-4 -4]]
```

```
# c = np.add(a1, a2,..)
# c = np.subtract(a1,a2,…)
# c = np.multiply(a1,a2,…)
# c = np.divide(a1,a2)
# c = np.negative(a)
# c = np.power(a1,a2)
```

Arithmetic operations

```
a1 = np.array([1,2],[3,4])
a2 = np.array([5,6],[7,8])
print(a1*a2)   # [[ 5 12] [21 32]]
print(a1/a2)   # [[0.2 0.33333333] [0.42857143 0.5]]
print(a1**a2) # [[1   64] [2187 65536]]
```

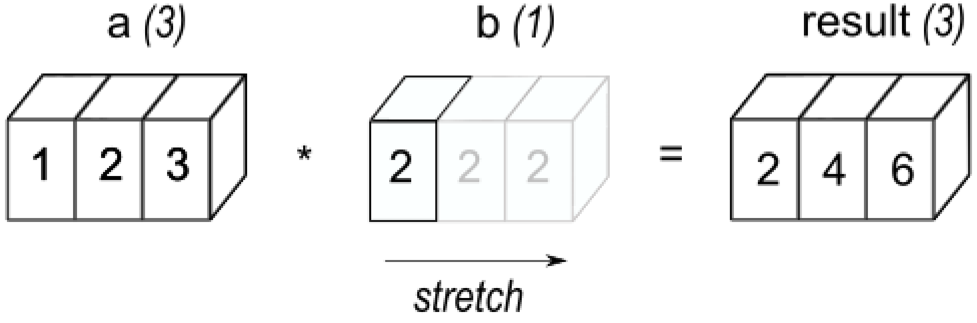
- Note that for element-wise operations, a1.shape must be equal to a2.shape. But numpy's *broadcasting* rule relaxes this constraint when the shapes of arrays meet certain constraints.

Broadcasting

- Example:

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([2, 2, 2])
c = a * b
print(c)    # [ 2 4 6]
```

```
a = np.array([1, 2, 3])
c = a * 2
print(c) # [ 2 4 6]
```



https://numpy.org/doc/stable/user/basics.broadcasting.html

Broadcasting

- General Broadcasting Rules:
- When operating on two arrays, Numpy compares their shapes element-wise. It starts with the trailing (i.e. rightmost) dimensions and works its way left. Two dimensions are compatible when:
  - they are equal, or
  - one of them is 1
- If these conditions are not met, a ValueError exception is thrown.
- The size of the resulting array is the size that is not 1 along each axis of the inputs.

https://numpy.org/doc/stable/user/basics.broadcasting.html

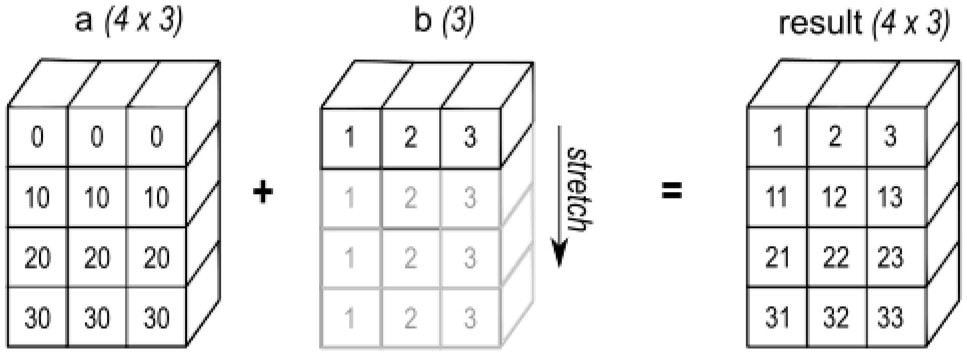
Broadcasting

- Examples ✓ ✓ ✓ ✓
- A is a 4d array: 8 x 1 x 6 x 1
- B is a 3d array: 7 x 1 x 5
- Result is a 4d array: 8 x 7 x 6 x 5

✓ ✗ ✓

A is a 2d array: 2 x 1

B is a 3d array: 8 x 4 x 3



0	0	0
10	10	10
20	20	20
30	30	30

1	2	3
1	2	3
1	2	3
1	2	3

1	2	3
11	12	13
21	22	23
31	32	33

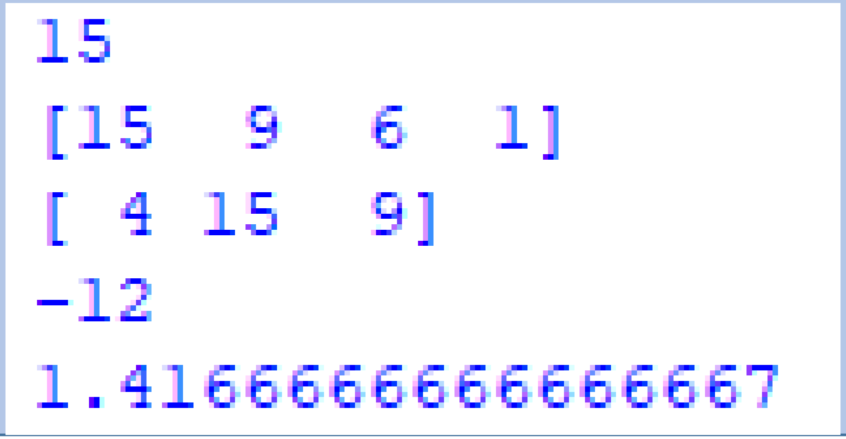
A is a 2d array: 4 x 3

B is a 1d array: 1 x 3

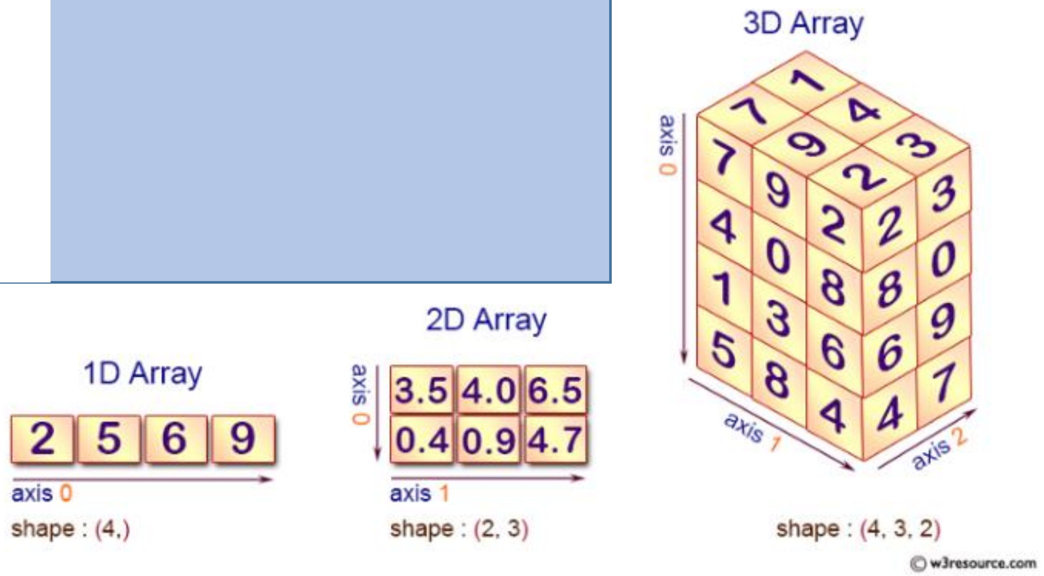
https://numpy.org/doc/stable/user/basics.broadcasting.html

Statistical Operations

```
a = np.array( [ [4,3,2,1], [15,-6,0,-8], [3,9,6,-12] ])
print(a.max())
print(a.max(axis=0))
print(a.max(axis=1))
print(a.min())
print(a.mean())
```



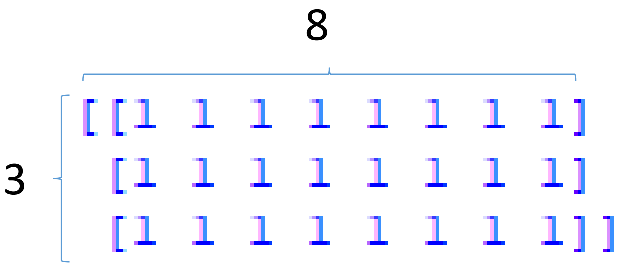
```
15
[15   9   6   1]
[ 4  15   9]
-12
1.4166666666666667
```



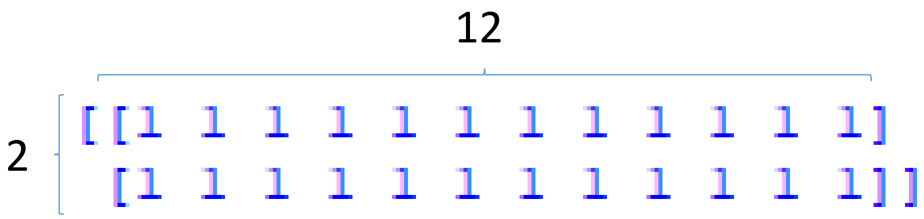
Reshaping Arrays

- We can use the reshape() method to change the shape of an array.
  - Recall that the shape of an array is the number of elements in each dimension.

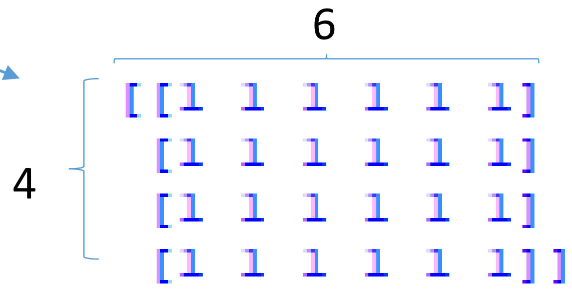
```
a= np.ones((3,8), dtype=int)
print(a)
b= a.reshape(2,12)
print(b)
c= b.reshape(4,6)
print(c)
c.shape = (1,24)
```



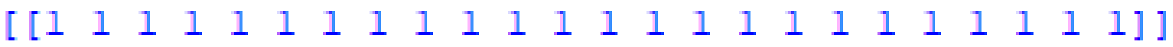
```
[[1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1]]
```



```
[[1 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 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]]
```

3*8 = 2*12 = 4*6 = 1*24

Iterating Arrays

- Iterating means going through elements one by one.

```
a = np.array([1,2,3,4,5])
for x in a:
    print(x, end=" ")  # prints 1 2 3 4 5
a = np.array([ [1,2,3], [4,5,6] ])
for x in a:
    print(x, end=" ")  # prints [1 2 3] [4 5 6]
a = np.array([ [[1,2], [4,5]], [[6,7],[8,9]] ])
for x in a:
    print(x)  # prints [[1 2]
                         [4 5]]
                        [[6 7]
                          [8 9]]
```

Iterating Arrays

- We can use the nditer() function to iterate through every element in an N-dimensional array

```
a = np.array([ [[1,2], [4,5]], [[6,7],[8,9]] ])
for x in np.nditer(a):
    print(x, end=" ")    #prints 1 2 4 5 6 7 8 9
```

```
for x in a.flat:
    print(i, end=“ ”)     # prints 1 2 4 5 6 7 8 9
```

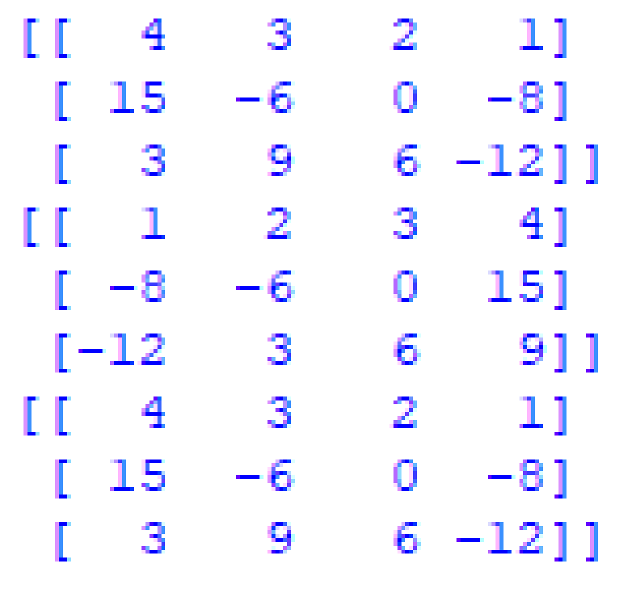
Sorting Arrays

- numpy.sort() function returns a sorted copy of an array

```
a = np.array( [ [4,3,2,1], [15,-6,0,-8], [3,9,6,-12] ])
print(a)

print(np.sort(a))

print(a)
```



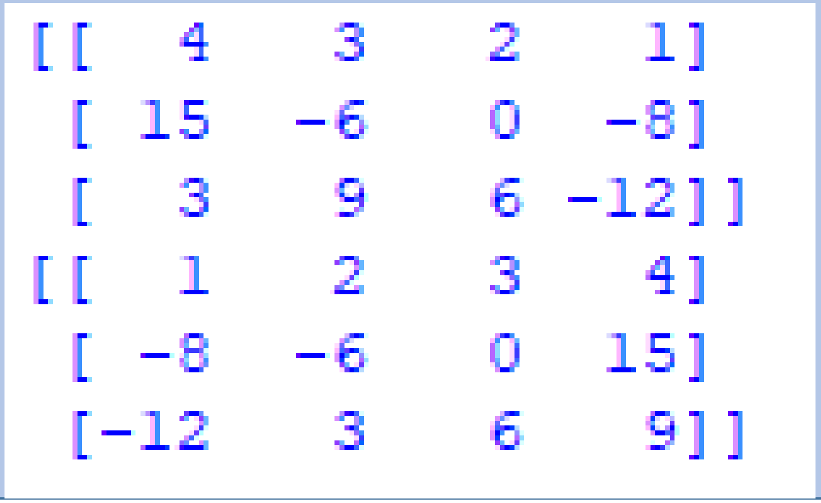
```
[[  4   3   2   1]
 [ 15  -6   0  -8]
 [  3   9   6 -12]]
[[  1   2   3   4]
 [ -8  -6   0  15]
 [-12   3   6   9]]
[[  4   3   2   1]
 [ 15  -6   0  -8]
 [  3   9   6 -12]]
```

Sorting Arrays

- ndarray.sort() method sorts an array in-place.

```
a = np.array( [ [4,3,2,1], [15,-6,0,-8], [3,9,6,-12] ])


print(a)
a.sort()
print(a)
```



```
[[  4   3   2   1]
 [ 15  -6   0  -8]
 [  3   9   6 -12]]
[[  1   2   3   4]
 [ -8  -6   0  15]
 [-12   3   6   9]]
```