Numpy array vs Python lists

```
# speed
# list
a = [i for i in range(10000000)]
b = [i for i in range(10000000,20000000)]

c = []
import time

start = time.time()
for i in range(len(a)):
  c.append(a[i] + b[i])
print(time.time()-start)
```

```
3.2699835300445557
```

```
# numpy
import numpy as np
a = np.arange(10000000)
b = np.arange(10000000,20000000)

start = time.time()
c = a + b
print(time.time()-start)
```

```
0.06481003761291504
```

```
3.26/0.06
```

```
54.33333333333333
```

```
# memory
a = [i for i in range(10000000)]
import sys

sys.getsizeof(a)
```

```
81528048
```

```
a = np.arange(10000000,dtype=np.int8)
sys.getsizeof(a)
```

```
10000104
```

```
# convenience
```

Advanced Indexing

```
# Normal Indexing and slicing

a = np.arange(24).reshape(6,4)
a
```

```
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]])
```

```
a[1,2]
```

```
5
```

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

```
array([[4, 5],
       [7, 8]])
```

```
# Fancy Indexing

a[:,[0,2,3]]
```

```
array([[ 0,  2,  3],
       [ 4,  6,  7],
       [ 8, 10, 11],
       [12, 14, 15],
       [16, 18, 19],
       [20, 22, 23]])
```

```
# Boolean Indexing
a = np.random.randint(1,100,24).reshape(6,4)
a
```

```
array([[76, 98, 99, 39],
       [91, 46, 88, 23],
       [45,  6, 83,  1],
       [37, 43, 78, 85],
       [54, 73, 61, 53],
       [40, 93, 85, 77]])
```

```
# find all numbers greater than 50
a[a > 50]
```

```
array([76, 98, 99, 91, 88, 83, 78, 85, 54, 73, 61, 53, 93, 85, 77])
```

```
# find out even numbers
a[a % 2 == 0]
```

```
array([76, 98, 46, 88,  6, 78, 54, 40])
```

```
# find all numbers greater than 50 and are even

a[(a > 50) & (a % 2 == 0)]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-97-0e69559201d8> in <module>
      1 # find all numbers greater than 50 and are even
      2
----> 3 a[(a > 50) and (a % 2 == 0)]

ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()
```

SEARCH STACK OVERFLOW

```
# find all numbers not divisible by 7
a[~(a % 7 == 0)]
```

```
array([76, 99, 39, 46, 88, 23, 45,  6, 83,  1, 37, 43, 78, 85, 54, 73, 61,
       53, 40, 93, 85])
```

Broadcasting

The term broadcasting describes how NumPy treats arrays with different shapes during arithmetic operations.

The smaller array is “broadcast” across the larger array so that they have compatible shapes.

```
# same shape
a = np.arange(6).reshape(2,3)
b = np.arange(6,12).reshape(2,3)

print(a)
print(b)

print(a+b)
```

```
[[0 1 2]
 [3 4 5]]
[[ 6  7  8]
 [ 9 10 11]]
[[ 6  8 10]
 [12 14 16]]
```

```
# diff shape
a = np.arange(6).reshape(2,3)
b = np.arange(3).reshape(1,3)

print(a)
print(b)

print(a+b)
```

```
[[0 1 2]
 [3 4 5]]
[[0 1 2]]
[[0 2 4]
 [3 5 7]]
```

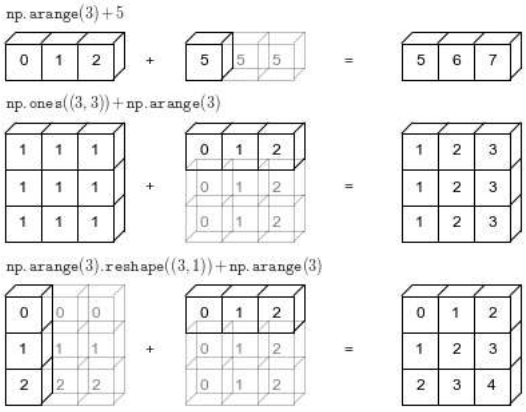
Broadcasting Rules

1. Make the two arrays have the same number of dimensions.

- If the numbers of dimensions of the two arrays are different, add new dimensions with size 1 to the head of the array with the smaller dimension.

2. Make each dimension of the two arrays the same size.

- If the sizes of each dimension of the two arrays do not match, dimensions with size 1 are stretched to the size of the other array.
- If there is a dimension whose size is not 1 in either of the two arrays, it cannot be broadcasted, and an error is raised.




```
# More examples

a = np.arange(12).reshape(4,3)
b = np.arange(3)

print(a)
print(b)

print(a+b)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
[0 1 2]
[[ 0  2  4]
 [ 3  5  7]
 [ 6  8 10]
 [ 9 11 13]]
```

```
a = np.arange(12).reshape(3,4)
b = np.arange(3)

print(a)
print(b)

print(a+b)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[0 1 2]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-104-fa6cbb589166> in <module>
      5 print(b)
      6
----> 7 print(a+b)

ValueError: operands could not be broadcast together with shapes (3,4) (3,)
```

SEARCH STACK OVERFLOW

```
a = np.arange(3).reshape(1,3)
b = np.arange(3).reshape(3,1)

print(a)
print(b)

print(a+b)
```

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

```
a = np.arange(3).reshape(1,3)
b = np.arange(4).reshape(4,1)

print(a)
print(b)

print(a + b)
```

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

```
a = np.array([1])
# shape -> (1,1)
b = np.arange(4).reshape(2,2)
# shape -> (2,2)

print(a)
print(b)

print(a+b)
```

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

```
a = np.arange(12).reshape(3,4)
b = np.arange(12).reshape(4,3)

print(a)
```

```
print(b)

print(a+b)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-109-c590a65467e5> in <module>
      5 print(b)
      6
----> 7 print(a+b)

ValueError: operands could not be broadcast together with shapes (3,4) (4,3)
```

SEARCH STACK OVERFLOW

```
a = np.arange(16).reshape(4,4)
b = np.arange(4).reshape(2,2)

print(a)
print(b)

print(a+b)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
[[0 1]
 [2 3]]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-110-57df50a0058a> in <module>
      5 print(b)
      6
----> 7 print(a+b)

ValueError: operands could not be broadcast together with shapes (4,4) (2,2)
```

SEARCH STACK OVERFLOW

Working with mathematical formulas

```
a = np.arange(10)
np.sin(a)
```

```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

```
# sigmoid
def sigmoid(array):
  return 1/(1 + np.exp(-(array)))


a = np.arange(100)

sigmoid(a)
```

```
array([0.5       , 0.73105858, 0.88079708, 0.95257413, 0.98201379,
       0.99330715, 0.99752738, 0.99908895, 0.99966465, 0.99987661,
       0.9999546 , 0.9999833 , 0.99999386, 0.99999774, 0.99999917,
       0.99999969, 0.99999989, 0.99999996, 0.99999998, 0.99999999,
       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.        ])
```

```
# mean squared error

actual = np.random.randint(1,50,25)
predicted = np.random.randint(1,50,25)


def mse(actual,predicted):
  return np.mean((actual - predicted)**2)

mse(actual,predicted)
```

```
500.12
```

```
# binary cross entropy
np.mean((actual - predicted)**2)
```

```
500.12
```

```
actual
```

```
array([ 5,  3,  9,  7,  3, 36, 49, 28, 20, 40,  2, 23, 29, 18, 30, 23,  7,
       40, 15, 11, 27, 44, 32, 28, 10])
```

Working with missing values

```
# Working with missing values -> np.nan
a = np.array([1,2,3,4,np.nan,6])
a
```

```
array([ 1.,  2.,  3.,  4., nan,  6.])
```

```
a[~np.isnan(a)]
```

```
array([1., 2., 3., 4., 6.])
```

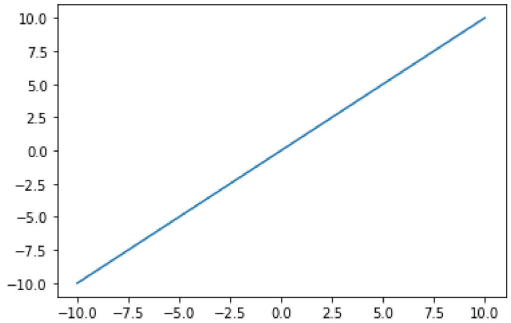
Plotting Graphs

```
# plotting a 2D plot
# x = y
import matplotlib.pyplot as plt

x = np.linspace(-10,10,100)
y = x

plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7f6f78e18f70>]
```

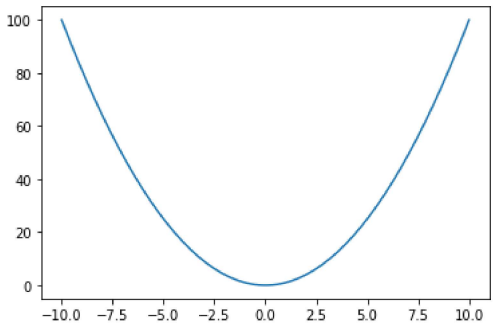



```
# y = x^2
x = np.linspace(-10,10,100)
```

```
y = x**2

plt.plot(x,y)
```

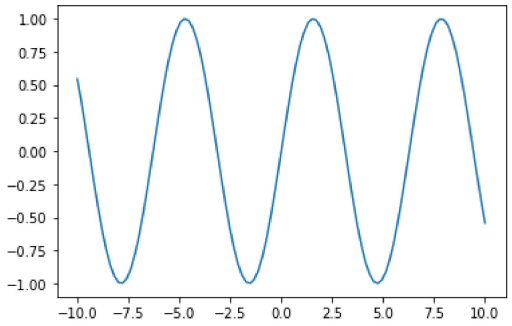
```
[<matplotlib.lines.Line2D at 0x7f6f87acf100>]
```



```
# y = sin(x)
x = np.linspace(-10,10,100)
y = np.sin(x)

plt.plot(x,y)
```

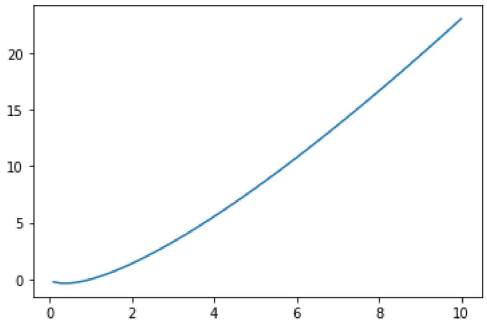
```
[<matplotlib.lines.Line2D at 0x7f6f5d1d0100>]
```



```
# y = xlog(x)
x = np.linspace(-10,10,100)
y = x * np.log(x)

plt.plot(x,y)
```

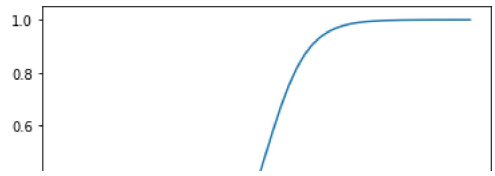
```
<ipython-input-137-4b3958c08378>:3: RuntimeWarning: invalid value encountered in log
  y = x * np.log(x)
[<matplotlib.lines.Line2D at 0x7f6f57ab62e0>]
```



```
# sigmoid
x = np.linspace(-10,10,100)
y = 1/(1+np.exp(-x))

plt.plot(x,y)
```

```
[<matplotlib.lines.Line2D at 0x7f6f5401e100>]
```



Meshgrids

```
# Meshgrids
```