Numpy to Python

```
import numpy as np
```

create an array from a list of values

```
list_of_values = [20.,2.,5.]
x = np.array(list_of_values)
print(x)
```

```
[20.  2.  5.]
```

```
more_values = [[[20],[2],[5]]]
y = np.array(more_values, dtype=np.int32)
print(y)
```

```
[[[20]
  [ 2]
  [ 5]]]
```

how to create an array

```
X =np.array([5,5])
zeros = np.zeros_like(X) # create array [0,0]
print(zeros)
```

```
[0 0]
```

```
ones = np.ones_like(X, dtype=np.float32) # create array [1. 1.]
print(ones)
```

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

how to create an array

```
# create an array containing [0,1,2,3,4]
x= np.arange(5)
print(x)
# create an array containing [2,3,4]
y = np.arange(2,5)
print(y)
# create an array containing [2,4]
```

```
z = np.arange(2,5,2)
print(z)
```

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

```
X = np.eye(5) # create identity matrix of size 5
print(X)
```

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

```
#create an random matrix of size 3x5
X = np.random.rand(3,5)
print(X)
```

```
[[0.32820144 0.10073119 0.99812916 0.64786679 0.03761148]
 [0.30771765 0.65736688 0.42687854 0.20042723 0.92978988]
 [0.04152951 0.65940289 0.51992967 0.4267471  0.90131586]]
```

```
#create and matrix 2*2 matrix in which each element is 7
X = np.full((2,2), 7)
print(X)
```

```
[[7 7]
 [7 7]]
```

```
#create a 2*2 matrix in which each element is 7
X = np.ones((2,2))* 7
print(X)
```

```
[[7. 7.]
 [7. 7.]]
```

attributes

```
x= np.array([2.,5.,3.])
y= np.array([2.,5.,3.],dtype=np.int32)

print(x.dtype)
print(y.dtype)
```

```
float64
int32
```

```
x= np.array([2.,5.,3.])
print(x.shape)
y = np.ones((2,4,1,2,3))
print(y.shape)
```

```
(3,)
(2, 4, 1, 2, 3)
```

Changing the shape of arrays

```
x = np.full((2,2,3),7)
print(x.shape)
x = np.expand_dims(x, axis=0)
assert x.shape == (1,2,2,3)
x = np.expand_dims(x, axis=-1)
assert x.shape == (1,2,2,3,1)
```

```
(2, 2, 3)
```

```
x = np.arange((10)) # [0,1,2,3,4,5,6,7,8,9]
print(x.shape)
y = np.reshape(x,(2,5)) # [[0,1,2,3,4],[5,6,7,8,9]]
assert y.shape == (2,5)
print(y.shape)
```

```
(10,)
(2, 5)
```

```
x = np.arange((20))
print((x))
y  = np.reshape(x,(2,-1,2))
assert y.shape == (2,5,2)
```

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

```
# x = np.arange((20))
# y = np.reshape(x,(2,-1,-1)) #values Error : can only specify one
unknown dimension
# print(y.shape)
```

```
x = np.full((20,1,1),5)
y = np.squeeze(x)
assert y.shape == (20,)
print(y.shape)
print(x.shape)
```

```
(20,)
(20, 1, 1)
```

```
x = np.full((20,1,1),5)
y = np.squeeze(x,axis=1)
assert y.shape == (20,1)
print(y.shape)
print(x.shape)
```

```
(20, 1)
(20, 1, 1)
```

Elements of arrays

```
# get the 10th element of the array
x = np.arange(20)
elem = x[10] # 10
print(elem)
```

```
10
```

```
# get the 10th element of the array
x = np.arange(20)
elemem = x.item(10) # 10
print(elemem)
```

```
10
```

```
# get elements with index in [10,15]
x = np.arange(20)
elements = x[10:15] # [10,11,12,13,14]
print(elements)

# get elements with index in [10: len(Array)-7]
more_elements = x[10:-7] # [10,11,12,13]
assert np.array_equal(more_elements, x[10:13])
print(more_elements)

# get elements whose index is a multiple of 3, starting from index 0
array = x[::3] # [0,3,6,9,12,15,18]
assert np.array_equal(array, [0,3,6,9,12,15,18])
print(array)
```

```
[10 11 12 13 14]
[10 11 12]
[ 0  3  6  9 12 15 18]
```

```
x = np.full((5,2),3)
y = np.full((5,1),4)
z = np.concatenate([x,y], axis=-1)
print(z) # [[3 3 4], [3 3 4], [3 3 4], [3 3 4], [3 3 4]]
assert z.shape == (5,3)
```

```
[[3 3 4]
 [3 3 4]
 [3 3 4]
 [3 3 4]
 [3 3 4]]
```

NumPy math: sum and subtraction

```
x = np.full((4,2,3),8) # 4*2*3 array, full of 7
y = np.ones_like(x) # 4*2*3 array, full of 1

print(x.shape)
print(y.shape)

# sum cuar 2 a array
z = x + y
print(z)
assert np.array_equal(z,np.ones_like(x)*9)

# subtract two arrays
z = x - y
print(z)
assert np.array_equal(z,np.ones_like(x)*7)
```

```
(4, 2, 3)
(4, 2, 3)
[[[9 9 9]
  [9 9 9]]

 [[9 9 9]
  [9 9 9]]

 [[9 9 9]
  [9 9 9]]

 [[9 9 9]
  [9 9 9]]]
[[[7 7 7]
  [7 7 7]]

 [[7 7 7]
  [7 7 7]]

 [[7 7 7]
  [7 7 7]]

 [[7 7 7]
  [7 7 7]]]
```

NumPy math: multiplication

```
x = np.full((4,2,3),8) # 4*2*3 array, full of 8
y = np.ones_like(x)*2 # 4*2*3 array, full of 2

# multiplication element-wise two arrays
mul = x*y
print(mul.shape)
assert np.array_equal(mul, np.ones_like(x)*16)

#subtract two arrays
mul2 = np.multiply(x,y)
print(mul2.shape)
assert np.array_equal(mul2, np.ones_like(x)*16)
```

```
(4, 2, 3)
(4, 2, 3)
```

```
x1 = np.full((4,2,3),8) # 4*2*3 array, full of 8
x2 = np.full((3,3),7) # 3*3 array, full of 7
y = np.eye(3) # 3*3 diagonalam array

# multiplication element-wise two arrays
mul = np.matmul(x1,y)
print(mul.shape)
assert np.array_equal(mul, x1)

#matrix multiplication
mul2 = np.matmul(x2,y)
print(mul2.shape)
assert np.array_equal(mul2, x2)
```

```
(4, 2, 3)
(3, 3)
```

Conditions

```
x = np.arange(5)
y = np.where(x<2,0,255) # [0,0,255,255,255]

print(x)
print(y)
```

```
[0 1 2 3 4]
[  0   0 255 255 255]
```

```
# Notice that NumPy is quite optimized, so when possible try to used
“native”
import time
```

```
x =  np.random.rand(4,640,480,3)
batch, height, width, channels = x.shape
start = time.time()
y = np.ones_like(x)
for i in range(batch):
    for j in range(height):
        for k in range(width):
            for l in range(channels):
                y[i,j,k,l] = 0 if x[i,j,k,l] < 0.05 else 255

duration = time.time() - start

print("Duration native: ", duration)

# NumPy way instead of other approaches
```

```
Duration native:  1.7774317264556885
```

```
x = np.random.rand(4, 640, 480, 3)
start = time.time()
y = np.where(x < 0.05, 0, 255)
duration = time.time() - start  # 0.0283 sec
print("Duration native: ", duration)
```

```
Duration native:  0.018548011779785156
```

NumPy Input/Output

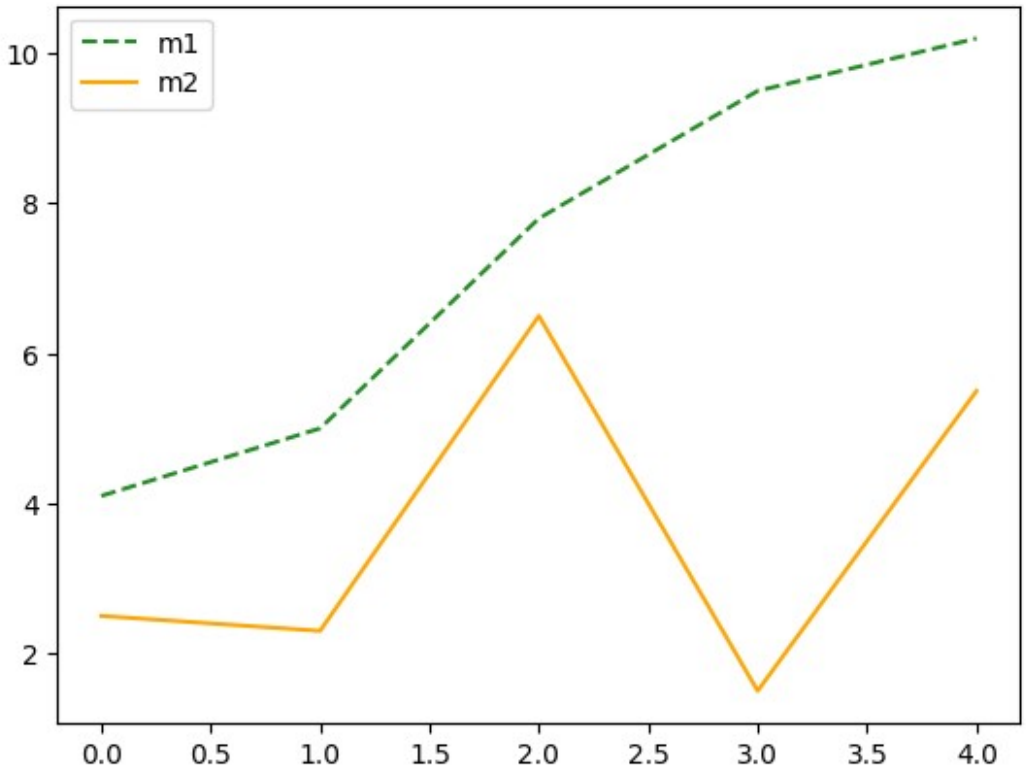
```
# data = np.loadtxt("name.txt",dtype=str,delimiter=",")
```

Matplotlib

```
import numpy as np
import matplotlib.pyplot as plt

labels = ['7', '8', '9', '10', '11']
x = np.arange(5)
m1 = np.array([4.1, 5.0, 7.8, 9.5, 10.2])
m2 = np.array([2.5, 2.3, 6.5, 1.5, 5.5])

plt.plot(x, m1, color='forestgreen', label='m1', linestyle='dashed')
plt.plot(x, m2, color='orange', label='m2')
plt.legend()
plt.savefig('chart.png')
```



#END