NumPy

In [91]:
```
import numpy as np
```

1. DataTypes & Attributes

In [92]:
```
# NumPy's main datatype is ndarray (n-dimentional-array)
a1 = np.array([1,2,3])
a1
```

Out[92]:
```
array([1, 2, 3])
```

In [93]:
```
type(a1)
```

Out[93]:
```
numpy.ndarray
```

In [94]:
```
a2 = np.array([[1,2.0,3.3],
               [4,5,6.5]])
a3 = np.array([[[1,2,3],
                [4,5,6],
                [7,8,9]],
               [[10,11,12],
                [13,14,15],
                [16,17,18]]
              ])
```

In [95]:
```
a2
```

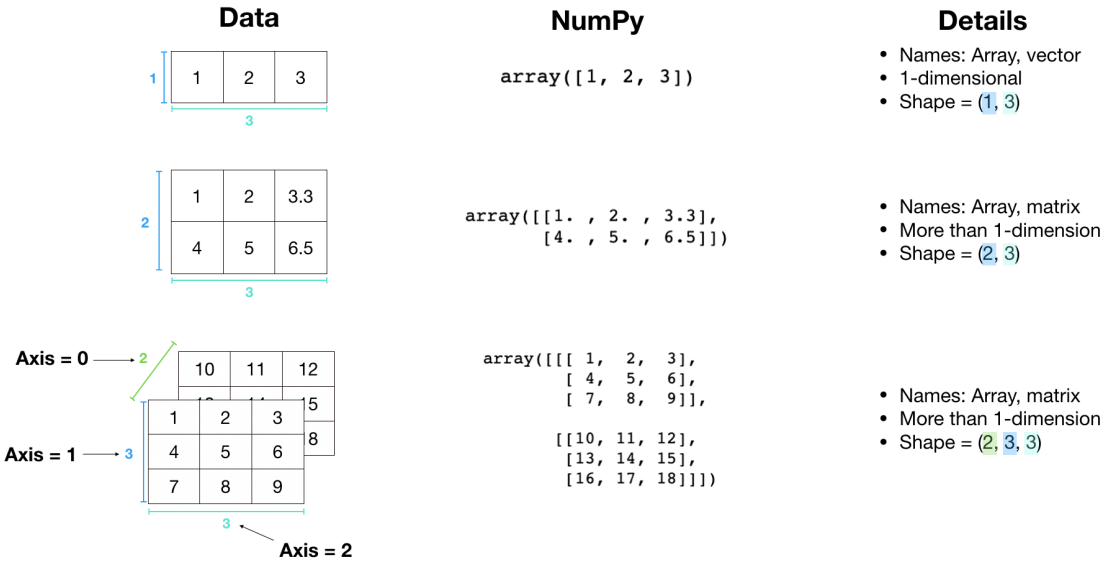
Out[95]:
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [96]:
```
a3
```

Out[96]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],

       [[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]]])
```

Anatomy of a NumPy array



In [97]:
```
a1.shape
```

Out[97]:
```
(3,)
```

In [98]:
```
a2.shape
```

Out[98]:
```
(2, 3)
```

In [99]:
```
a3.shape
```

Out[99]:
```
(2, 3, 3)
```

In [100]:
```
a1.ndim,a2.ndim,a3.ndim
```

Out[100]:
```
(1, 2, 3)
```

In [101]:
```
a1.dtype,a2.dtype,a3.dtype
```

Out[101]:
```
(dtype('int32'), dtype('float64'), dtype('int32'))
```

In [102]:
```
a1.size,a2.size,a3.size
```

Out[102]:
```
(3, 6, 18)
```

In [103]:
```
# Crate a Dataframe from NumPy array
import pandas as pd
df = pd.DataFrame(a2)
df
```

Out[103]:

	0	1	2
0	1.0	2.0	3.3
1	4.0	5.0	6.5

2. Creating Arrays

In [104]:
```
sample_array = np.array([1,2,3])
sample_array
```

Out[104]:
```
array([1, 2, 3])
```

In [105]:
```
# Prefilled array
ones = np.ones((2,3)) # Default dtype float
ones
```

Out[105]:
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

In [106]:
```
zeros = np.zeros((2,3))
zeros
```

Out[106]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [107]:
```
range_array = np.arange(0, 10, 2) # Start, Stop, Step
range_array
```

Out[107]:
```
array([0, 2, 4, 6, 8])
```

In [108]:
```
random_array = np.random.randint(0, 10, (3,5)) # Low, High, Size
random_array
```

Out[108]:
```
array([[2, 3, 8, 1, 3],
       [3, 3, 7, 0, 1],
       [9, 9, 0, 4, 7]])
```

In [109]:
```
random_array_2 = np.random.random((5,3)) #Shape/SIze
# Gives float values within interval 0-1
random_array_2
```

Out[109]:
```
array([[0.26455561, 0.77423369, 0.45615033],
       [0.56843395, 0.0187898 , 0.6176355 ],
       [0.61209572, 0.616934  , 0.94374808],
       [0.6818203 , 0.3595079 , 0.43703195],
       [0.6976312 , 0.06022547, 0.66676672]])
```

In [110]:
```
random_array_3 = np.random.rand(5,3) # Random vales in given shape
random_array_3
```

Out[110]:
```
array([[0.67063787, 0.21038256, 0.1289263 ],
       [0.31542835, 0.36371077, 0.57019677],
       [0.43860151, 0.98837384, 0.10204481],
       [0.20887676, 0.16130952, 0.65310833],
       [0.2532916 , 0.46631077, 0.24442559]])
```

In [111]:
```
# Pseudo-random numbers : When NumPy generates random numbers they are actully Ps
# np.random.seed() helps in generating random numbers that are reproducable
# Used when we want exact results on reiteration
np.random.seed(0) # Works with any seed value
random_array_4 = np.random.randint(0,10, (5,3))
random_array_4
```

Out[111]:
```
array([[5, 0, 3],
       [3, 7, 9],
       [3, 5, 2],
       [4, 7, 6],
       [8, 8, 1]])
```

3. Viewing array and matrices

In [112]:
```
# Find unique elements
np.unique(random_array_4)
```

Out[112]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [113]:
```
a1
```

Out[113]:
```
array([1, 2, 3])
```

In [114]:
```
a2
```

Out[114]:
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [115]:
```
a3
```

Out[115]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],

       [[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]]])
```

In [116]:
```
a1[0],a2[0],a3[0]
```

Out[116]:
```
(1,
 array([1. , 2. , 3.3]),
 array([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]))
```

In [117]:
```
a2[0,0], a3[0,0,0]
```

Out[117]:
```
(1.0, 1)
```

In [118]:
```
a3[:2, :2, :2] # Slicing
```

Out[118]:
```
array([[[ 1,  2],
        [ 4,  5]],

       [[10, 11],
        [13, 14]]])
```

In [119]:
```
a4 = np.random.randint(0, 10, (2,3,4,5))
a4
```

Out[119]:
```
array([[[[6, 7, 7, 8, 1],
         [5, 9, 8, 9, 4],
         [3, 0, 3, 5, 0],
         [2, 3, 8, 1, 3]],

        [[3, 3, 7, 0, 1],
         [9, 9, 0, 4, 7],
         [3, 2, 7, 2, 0],
         [0, 4, 5, 5, 6]],

        [[8, 4, 1, 4, 9],
         [8, 1, 1, 7, 9],
         [9, 3, 6, 7, 2],
         [0, 3, 5, 9, 4]]],


       [[[4, 6, 4, 4, 3],
         [4, 4, 8, 4, 3],
         [7, 5, 5, 0, 1],
         [5, 9, 3, 0, 5]],

        [[0, 1, 2, 4, 2],
         [0, 3, 2, 0, 7],
         [5, 9, 0, 2, 7],
         [2, 9, 2, 3, 3]],

        [[2, 3, 4, 1, 2],
         [9, 1, 4, 6, 8],
         [2, 3, 0, 0, 6],
         [0, 6, 3, 3, 8]]]])
```

In [120]:
```
# Get first n numbers of the immer most arrays
a4[:, :, :, :3]
```

Out[120]:
```
array([[[[6, 7, 7],
         [5, 9, 8],
         [3, 0, 3],
         [2, 3, 8]],

        [[3, 3, 7],
         [9, 9, 0],
         [3, 2, 7],
         [0, 4, 5]],

        [[8, 4, 1],
         [8, 1, 1],
         [9, 3, 6],
         [0, 3, 5]]],


       [[[4, 6, 4],
         [4, 4, 8],
         [7, 5, 5],
         [5, 9, 3]],

        [[0, 1, 2],
         [0, 3, 2],
         [5, 9, 0],
         [2, 9, 2]],

        [[2, 3, 4],
         [9, 1, 4],
         [2, 3, 0],
         [0, 6, 3]]]])
```

4. Manipulating & comparing arrays

Arithmatic

In [121]:
```
a1
```

Out[121]:
```
array([1, 2, 3])
```

In [122]:
```
ones = np.ones(3)
ones
```

Out[122]:
```
array([1., 1., 1.])
```

In [123]:
```
a1 + ones, a1 - ones, a1 * ones * 2, a1 / ones
```

Out[123]:
```
(array([2., 3., 4.]),
 array([0., 1., 2.]),
 array([2., 4., 6.]),
 array([1., 2., 3.]))
```

In [124]:
```
a2, a2.shape
```

Out[124]:
```
(array([[1. , 2. , 3.3],
        [4. , 5. , 6.5]]),
 (2, 3))
```

In [125]:
```
a1 * a2, a1 / a2, a2 ** 2, a1%2
```

Out[125]:
```
(array([[ 1. ,  4. ,  9.9],
        [ 4. , 10. , 19.5]]),
 array([[1.        , 1.        , 0.90909091],
        [0.25      , 0.4       , 0.46153846]]),
 array([[ 1.  ,  4.  , 10.89],
        [16.  , 25.  , 42.25]]),
 array([1, 0, 1], dtype=int32))
```

In [126]:
```
a3, a3.shape
```

Out[126]:
```
(array([[[ 1,  2,  3],
         [ 4,  5,  6],
         [ 7,  8,  9]],

        [[10, 11, 12],
         [13, 14, 15],
         [16, 17, 18]]]),
 (2, 3, 3))
```

In [127]:
```
a2//a1 # Floor division rounds down decimals
```

Out[127]:
```
array([[1., 1., 1.],
       [4., 2., 2.]])
```

In [128]:
```
np.exp(a1), np.log(a1)
```

Out[128]:
```
(array([ 2.71828183,  7.3890561 , 20.08553692]),
 array([0.        , 0.69314718, 1.09861229]))
```

Aggregation

Performing the same operation on a number of things

In [129]:
```
listy_list = [1,2,3]
type(listy_list), sum(listy_list)
```

Out[129]:
```
(list, 6)
```

In [130]:
```
type(a1), sum(a1), np.sum(a1) # Many Ways to do one operation
```

Out[130]:
```
(numpy.ndarray, 6, 6)
```

Use Python's methods ('sum()') on Python datatypes and use NumPy's methods ('np.sum()') on NumPy arrays.

In [131]:
```
massive_array = np.random.random(100000)
massive_array.size
```

Out[131]:
```
100000
```

In [132]:
```
massive_array[:10]
```

Out[132]:
```
array([0.16494046, 0.36980809, 0.14644176, 0.56961841, 0.70373728,
       0.28847644, 0.43328806, 0.75610669, 0.39609828, 0.89603839])
```

In [133]:
```
%timeit sum(massive_array) # Time how long a perticular line of code takes to rur
%timeit np.sum(massive_array) # Faster as it is numpy array
```

```
11 ms ± 83.3 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
68.5 µs ± 806 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [134]:
```
a2
```

Out[134]:
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [135]:
```
np.mean(a2), np.max(a2), np.min(a2), np.std(a2), np.var(a2) # High varience = wid
# Std Deviation = Sqrt of Var
```

Out[135]:
```
(3.6333333333333333, 6.5, 1.0, 1.8226964152656422, 3.3222222222222224)
```

In [136]:
```
high_var_array = np.array([1, 100, 200, 300, 4000, 5000])
low_var_array = np.array([2,4,6,8, 10])
np.var(high_var_array), np.var(low_var_array), np.std(high_var_array), np.std(lov
```

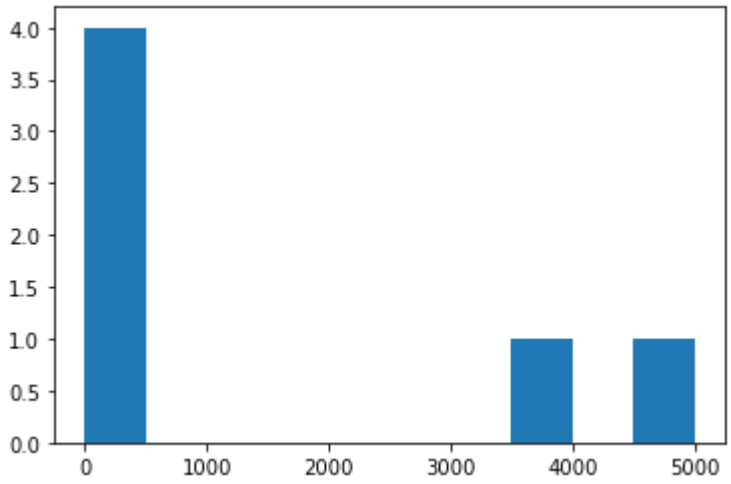
Out[136]:
```
(4296133.472222221, 8.0, 2072.711623024829, 2.8284271247461903)
```

In [137]:
```
 np.mean(high_var_array), np.mean(low_var_array)
```

Out[137]:
```
(1600.1666666666667, 6.0)
```

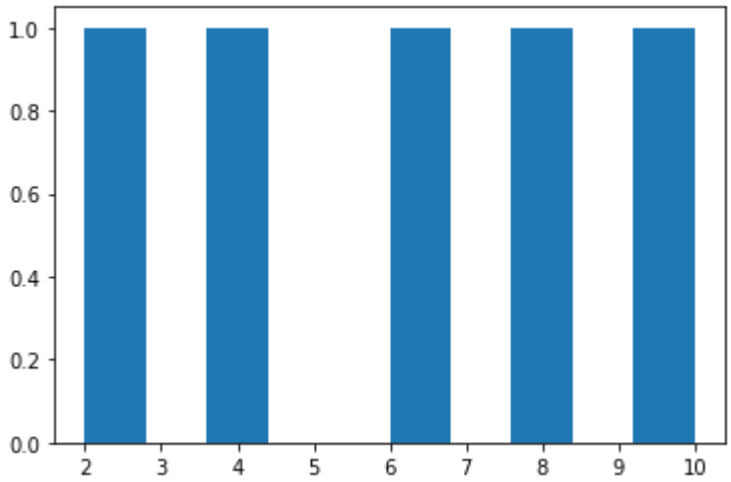
In [138]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.hist(high_var_array)
plt.show
```

Out[138]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [139]:
```
plt.hist(low_var_array)
plt.show
```

Out[139]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Reshaping & transposing

In [140]:
```
a2, a2.shape
```

Out[140]:
```
(array([[1. , 2. , 3.3],
        [4. , 5. , 6.5]]),
 (2, 3))
```

In [141]:
```
a3, a3.shape
```

Out[141]:
```
(array([[[ 1,  2,  3],
         [ 4,  5,  6],
         [ 7,  8,  9]],

        [[10, 11, 12],
         [13, 14, 15],
         [16, 17, 18]]]),
 (2, 3, 3))
```

In [142]:
```
# Broadcasting rule states smaller array are brodcast over larger array for multi
# Two dimentions are compatible when A. They are equal B. One of them is 1
a2_reshape = a2.reshape(2,3,1) # Returns an array containing same data with new s
a2_reshape
```

Out[142]:
```
array([[[1. ],
        [2. ],
        [3.3]],

       [[4. ],
        [5. ],
        [6.5]]])
```

In [143]:
```
a2_reshape * a3 # Works now
```

Out[143]:
```
array([[[  1. ,   2. ,   3. ],
        [  8. ,  10. ,  12. ],
        [ 23.1,  26.4,  29.7]],

       [[ 40. ,  44. ,  48. ],
        [ 65. ,  70. ,  75. ],
        [104. , 110.5, 117. ]]])
```

In [144]:
```
#Transpose = switch axes
a2, a2.shape
```

Out[144]:
```
(array([[1. , 2. , 3.3],
        [4. , 5. , 6.5]]),
 (2, 3))
```

In [145]:
```
a2.T, a2.T.shape
```

Out[145]:
```
(array([[1. , 4. ],
        [2. , 5. ],
        [3.3, 6.5]]),
 (3, 2))
```

Dot product vs Element wise

In [146]:
```
np.random.seed(0)
mat1 = np.random.randint(0, 10, (5,3))
mat2 = np.random.randint(0, 10, (5,3))
mat1, mat2
```

Out[146]:
```
(array([[5, 0, 3],
        [3, 7, 9],
        [3, 5, 2],
        [4, 7, 6],
        [8, 8, 1]]),
 array([[6, 7, 7],
        [8, 1, 5],
        [9, 8, 9],
        [4, 3, 0],
        [3, 5, 0]]))
```

In [147]:
```
# Element-wise multiplication (Hadamard Product)
mat1 * mat2
```

Out[147]:
```
array([[30,  0, 21],
       [24,  7, 45],
       [27, 40, 18],
       [16, 21,  0],
       [24, 40,  0]])
```

In [148]:
```
# Dot product ; Condition is number on the inside must match (ex. like: 2x3 & 3x2
# Result has dimention of outside numbers
mat3 = np.dot(mat1, mat2.T) # 5x3 & 3x5
mat3, mat3.shape
```

Out[148]:
```
(array([[ 51,  55,  72,  20,  15],
        [130,  76, 164,  33,  44],
        [ 67,  39,  85,  27,  34],
        [115,  69, 146,  37,  47],
        [111,  77, 145,  56,  64]]),
 (5, 5))
```

Comparison Operators

In [149]:
```
a1, a2
```

Out[149]:
```
(array([1, 2, 3]),
 array([[1. , 2. , 3.3],
        [4. , 5. , 6.5]]))
```

In [150]:
```
a1 < a2
```

Out[150]:
```
array([[False, False,  True],
       [ True,  True,  True]])
```

In [151]:
```
a1 < 3
```

Out[151]:
```
array([ True,  True, False])
```

In [152]:
```
a1 == a2
```

Out[152]:
```
array([[ True,  True, False],
       [False, False, False]])
```

5. Sorting Arrays

In [153]:
```
random_array
```

Out[153]:
```
array([[2, 3, 8, 1, 3],
       [3, 3, 7, 0, 1],
       [9, 9, 0, 4, 7]])
```

In [154]:
```
np.sort(random_array)
```

Out[154]:
```
array([[1, 2, 3, 3, 8],
       [0, 1, 3, 3, 7],
       [0, 4, 7, 9, 9]])
```

In [155]:
```
np.argsort(random_array) # Returns indicies that would sort an array.
```

Out[155]:
```
array([[3, 0, 1, 4, 2],
       [3, 4, 0, 1, 2],
       [2, 3, 4, 0, 1]], dtype=int64)
```

In [156]:
```
a1
```

Out[156]:
```
array([1, 2, 3])
```

In [157]:
```
np.argmin(a1), np.argmax(a1) #returms imdex of value
```

Out[157]:
```
(0, 2)
```

In [158]:
```
np.argmax(random_array, axis = 0) # Going along columns and returning position
```

Out[158]:
```
array([2, 2, 0, 2, 2], dtype=int64)
```

6. Turn images into numpy arrays

In [159]:
```
from matplotlib.image import imread
panda = imread("panda.png")
print(type(panda))
```

```
<class 'numpy.ndarray'>
```

In [160]:
```
panda[:5]
```

Out[160]:
```
array([[[0.05490196, 0.10588235, 0.06666667],
        [0.05490196, 0.10588235, 0.06666667],
        [0.05490196, 0.10588235, 0.06666667],
        ...,
        [0.16470589, 0.12941177, 0.09411765],
        [0.16470589, 0.12941177, 0.09411765],
        [0.16470589, 0.12941177, 0.09411765]],

       [[0.05490196, 0.10588235, 0.06666667],
        [0.05490196, 0.10588235, 0.06666667],
        [0.05490196, 0.10588235, 0.06666667],
        ...,
        [0.16470589, 0.12941177, 0.09411765],
        [0.16470589, 0.12941177, 0.09411765],
        [0.16470589, 0.12941177, 0.09411765]],

       [[0.05490196, 0.10588235, 0.06666667],
        [0.05490196, 0.10588235, 0.06666667],
        [0.05490196, 0.10588235, 0.06666667],
        ...,
        [0.16470589, 0.12941177, 0.09411765],
        [0.16470589, 0.12941177, 0.09411765],
        [0.16470589, 0.12941177, 0.09411765]],

       [[0.05490196, 0.10588235, 0.06666667],
        [0.05490196, 0.10588235, 0.06666667],
        [0.05490196, 0.10588235, 0.06666667],
        ...,
        [0.16862746, 0.13333334, 0.09803922],
        [0.16862746, 0.13333334, 0.09803922],
        [0.16862746, 0.13333334, 0.09803922]],

       [[0.05490196, 0.10588235, 0.06666667],
        [0.05490196, 0.10588235, 0.06666667],
        [0.05490196, 0.10588235, 0.06666667],
        ...,
        [0.16862746, 0.13333334, 0.09803922],
        [0.16862746, 0.13333334, 0.09803922],
        [0.16862746, 0.13333334, 0.09803922]]], dtype=float32)
```

In [161]:
```
panda.size, panda.shape, panda.ndim
```

Out[161]:
```
(24465000, (2330, 3500, 3), 3)
```