In [1]:
```
import numpy as np
```

1. DataTypes & Attributes

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

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

In [3]:
```
type(a1)
```

Out[3]:
```
numpy.ndarray
```

In [4]:
```
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 [5]:
```
a2
```

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

In [6]:
```
a3
```

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

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

In [7]:
```
a1.shape
```

Out[7]:
```
(3,)
```

In [8]:
```
a2.shape
```

Out[8]:
```
(2, 3)
```

In [9]:

```
a3.shape
```

Out[9]:
```
(2, 3, 3)
```

In [10]:
```
a1
```

Out[10]:
```
array([1, 2, 3])
```

In [11]:
```
a2
```

Out[11]:
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [12]:
```
a3
```

Out[12]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],

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

In [13]:
```
# Number of dimensions
a1.ndim, a2.ndim, a3.ndim
```

Out[13]:
```
(1, 2, 3)
```

In [14]:
```
# See the Data type of inside elements
a1.dtype, a2.dtype, a3.dtype
```

Out[14]:
```
(dtype('int32'), dtype('float64'), dtype('int32'))
```

In [15]:
```
a3
```

Out[15]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],

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

In [16]:
```
# See the number of elements
a1.size, a2.size, a3.size
```

Out[16]:
```
(3, 6, 18)
```

In [17]:
```
# See the type of numpy array
type(a1), type(a2), type(a3)
```

Out[17]:
```
(numpy.ndarray, numpy.ndarray, numpy.ndarray)
```

In [18]:
```
a2
```

Out[18]:
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [19]:
```
# Create a DataFrame from a NumPy array
import pandas as pd

df = pd.DataFrame(a2)
df
```

Out[19]:

	0	1	2
0	1.0	2.0	3.3
1	4.0	5.0	6.5

2. Creating arrays

In [20]:
```
sample_array = np.array([1, 2, 3])
sample_array
```

Out[20]:
```
array([1, 2, 3])
```

In [21]:
```
sample_array.dtype
```

Out[21]:
```
dtype('int32')
```

In [22]:
```
# Create array with only 1's
ones = np.ones((2, 3))
```

In [23]:
```
ones
```

Out[23]:
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

In [24]:
```
ones.dtype
```

Out[24]:
```
dtype('float64')
```

In [25]:
```
type(ones)
```

Out[25]:
```
numpy.ndarray
```

In [26]:
```
zeros = np.zeros((2, 3))
```

In [27]:
```
zeros
```

Out[27]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [28]:
```
# Create sequence array
range_array = np.arange(0, 10, 2)
range_array
```

Out[28]:
```
array([0, 2, 4, 6, 8])
```

In [29]:
```
# Create random array between 0 and 10
random_array = np.random.randint(0, 10, size=(3, 5))
random_array
```

Out[29]:
```
array([[3, 5, 0, 8, 1],
       [5, 3, 3, 7, 9],
       [7, 8, 6, 6, 8]])
```

In [30]:
```
random_array.size
```

Out[30]:
```
15
```

In [31]:
```
random_array.shape
```

Out[31]:
```
(3, 5)
```

In [32]:
```
# Create random array between 0 and 1
random_array_2 = np.random.random((5, 3))
random_array_2
```

Out[32]:
```
array([[0.43896165, 0.43698982, 0.76787552],
       [0.65878099, 0.93941913, 0.65547921],
       [0.00532002, 0.67563112, 0.68859225],
       [0.86867191, 0.38447291, 0.14053908],
       [0.93473465, 0.50503627, 0.21339744]])
```

In [33]:
```
random_array_2.shape
```

Out[33]:
```
(5, 3)
```

In [34]:
```
# Another way to create random array between 0 and 1
random_array_3 = np.random.rand(5, 3)
random_array_3
```

Out[34]:
```
array([[0.39171785, 0.80991565, 0.19883763],
       [0.05687711, 0.0306552 , 0.28709319],
       [0.03120219, 0.81174484, 0.00225838],
       [0.32222389, 0.31604699, 0.33876404],
       [0.93212758, 0.30342288, 0.75145588]])
```

In [35]:
```
# Pseudo-random numbers (Seems like random, but actually not random)
# It will generate same random numbers if we use seeding
np.random.seed(seed=99999)
random_array_4 = np.random.randint(10, size=(5, 3))
random_array_4
```

Out[35]:
```
array([[0, 3, 1],
       [8, 1, 3],
       [8, 5, 6],
       [0, 6, 0],
       [4, 0, 9]])
```

In [36]:
```
np.random.seed(7)
random_array_5 = np.random.random((5, 3))
random_array_5
```

Out[36]:
```
array([[0.07630829, 0.77991879, 0.43840923],
       [0.72346518, 0.97798951, 0.53849587],
       [0.50112046, 0.07205113, 0.26843898],
       [0.4998825 , 0.67923   , 0.80373904],
       [0.38094113, 0.06593635, 0.2881456 ]])
```

In [37]:
```
random_array_5 = np.random.random((5, 3))
random_array_5
```

Out[37]:
```
array([[0.90959353, 0.21338535, 0.45212396],
       [0.93120602, 0.02489923, 0.60054892],
       [0.9501295 , 0.23030288, 0.54848992],
       [0.90912837, 0.13316945, 0.52341258],
       [0.75040986, 0.66901324, 0.46775286]])
```

In [38]:
```
random_array_4
```

Out[38]:
```
array([[0, 3, 1],
       [8, 1, 3],
       [8, 5, 6],
       [0, 6, 0],
       [4, 0, 9]])
```

3. Viewing arrays and matrices

In [39]:
```
np.unique(random_array_4)
```

Out[39]:
```
array([0, 1, 3, 4, 5, 6, 8, 9])
```

In [40]:
```
a1
```

Out[40]:
```
array([1, 2, 3])
```

In [41]:
```
a2
```

Out[41]:
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [42]:
```
a3
```

Out[42]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],

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

In [43]:
```
a1[0]
```

Out[43]:
```
1
```

In [44]:
```
a2.shape
```

Out[44]:
```
(2, 3)
```

In [45]:
```
a2[0]
```

Out[45]:
```
array([1. , 2. , 3.3])
```

In [46]:
```
a3.shape
```

Out[46]:
```
(2, 3, 3)
```

In [47]:
```
a3[0]
```

Out[47]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [48]:
```
a3
```

Out[48]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],

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

In [49]:
```
a2
```

Out[49]:
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [50]:
```
a2[1]
```

Out[50]:
```
array([4. , 5. , 6.5])
```

In [51]:
```
a3
```

Out[51]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],

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

In [52]:
```
a3.shape
```

Out[52]:
```
(2, 3, 3)
```

In [53]:
```
a3[:2, :2, :2]
```

Out[53]:
```
array([[[ 1,  2],
        [ 4,  5]],

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

In [54]:
```
# 5 rows, 4 columns, . . .
a4 = np.random.randint(10, size=(2, 3, 4, 5))
a4
```

Out[54]:
```
array([[[[6, 7, 7, 9, 3],
         [0, 7, 7, 7, 0],
         [5, 4, 3, 1, 3],
         [1, 3, 4, 3, 1]],

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

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


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

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

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

In [55]:
```
a4.shape, a4.ndim
```

Out[55]:
```
((2, 3, 4, 5), 4)
```

In [56]:
```
# Get the first 4 numbers of the inner most arrays
# Dont think about too much
a4[:, :, :, :1]
```

Out[56]:
```
array([[[[6],
         [0],
         [5],
         [1]],

        [[9],
         [3],
         [3],
         [4]],

        [[8],
         [3],
         [6],
         [4]]],


       [[[6],
         [4],
         [1],
         [5]],

        [[9],
         [5],
         [9],
         [6]],

        [[0],
         [4],
         [7],
         [5]]]])
```

4. Manipulating & comparing arrays

Arithmetic

In [57]:
```
a1
```

Out[57]:
```
array([1, 2, 3])
```

In [58]:
```
ones = np.ones(3)
ones
```

Out[58]:
```
array([1., 1., 1.])
```

In [59]:
```
a1 + ones
```

Out[59]:
```
array([2., 3., 4.])
```

In [60]:
```
a1 - ones
```

Out[60]:
```
array([0., 1., 2.])
```

In [61]:
```
a1 * ones
```

Out[61]:
```
array([1., 2., 3.])
```

In [62]:
```
a1
```

Out[62]:
```
array([1, 2, 3])
```

In [63]:
```
a2
```

Out[63]:
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [64]:
```
# Not the matrix multiplication (Multiply corresponding elements)
a1 * a2
```

Out[64]:
```
array([[ 1. ,  4. ,  9.9],
       [ 4. , 10. , 19.5]])
```

In [65]:
```
a1
```

Out[65]:
```
array([1, 2, 3])
```

In [66]:
```
a2
```

Out[66]:
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [67]:
```
ones
```

Out[67]:
```
array([1., 1., 1.])
```

In [68]:
```
a1 / ones
```

Out[68]:
```
array([1., 2., 3.])
```

In [69]:
```
a2 / a1
```

Out[69]:
```
array([[1.        , 1.        , 1.1       ],
       [4.        , 2.5       , 2.16666667]])
```

In [70]:
```
# Floor division removes the decimals (rounds down)
a2 // a1
```

Out[70]:
```
array([[1., 1., 1.],
       [4., 2., 2.]])
```

In [71]:
```
a2
```

Out[71]:
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [72]:
```
a2 ** 2
```

Out[72]:
```
array([[ 1.  ,  4.  , 10.89],
       [16.  , 25.  , 42.25]])
```

In [73]:
```
np.square(a2)
```

Out[73]:
```
array([[ 1.  ,  4.  , 10.89],
       [16.  , 25.  , 42.25]])
```

In [74]:
```
# Add 2 arrays
np.add(a1, ones)
```

Out[74]:
```
array([2., 3., 4.])
```

In [75]:
```
a1
```

Out[75]:
```
array([1, 2, 3])
```

In [76]:
```
# Modulus
a1 % 2
```

Out[76]:
```
array([1, 0, 1], dtype=int32)
```

In [77]:
```
a2 % 2
```

Out[77]:
```
array([[1. , 0. , 1.3],
       [0. , 1. , 0.5]])
```

In [78]:
```
# e^(a1)
np.exp(a1)
```

Out[78]:
```
array([ 2.71828183,  7.3890561 , 20.08553692])
```

In [79]:
```
# log10 (a1)
np.log(a1)
```

Out[79]:
```
array([0.        , 0.69314718, 1.09861229])
```

Aggregation

Aggregation = performing the same operation on a number of things

In [80]:
```
# Python list
listy_list = [1, 2, 3]
type(listy_list)
```

Out[80]:
```
list
```

In [81]:
```
sum(listy_list)
```

Out[81]:
```
6
```

In [82]:
```
# Numpy list
type(a1)
```

Out[82]:
```
numpy.ndarray
```

In [83]:
```
# get sum of numpy list
sum(a1)
```

Out[83]:
```
6
```

In [84]:
```
# Another way of getting sum of numpy list
np.sum(a1)
```

Out[84]:
```
6
```

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

In [85]:
```
# Creative a massive NumPy array
massive_array = np.random.random(100000)
massive_array.size
```

Out[85]:
```
100000
```

In [86]:
```
massive_array[:10]
```

Out[86]:
```
array([0.62342345, 0.62955693, 0.9099729 , 0.96251949, 0.5850998 ,
       0.16489774, 0.39159332, 0.94455493, 0.34339118, 0.70507037])
```

In [87]:
```
# '%' => Used for magic functions
# Measure time for python sun and numpy sum
%timeit sum(massive_array) # Python's sum()
%timeit np.sum(massive_array) # NumPy's np.sum()
```

```
5.02 ms ± 93 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
26 µs ± 974 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [88]:
```
# See how much times numpy's sum is fast
4.73 * 1000 / 25.6
```

Out[88]:
```
184.765625
```

In [89]:
```
a2
```

Out[89]:
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [90]:
```
# mean of numpy array
np.mean(a2)
```

Out[90]:
```
3.6333333333333333
```

In [91]:
```
# maximum value
np.max(a2)
```

Out[91]:
```
6.5
```

In [92]:
```
# minimum value
np.min(a2)
```

Out[92]:
```
1.0
```

Standard deviation and **variance** are measures of 'spread' of data.

The higher standard deviation and the higher variance of data, the more spread out the values are.

The lower standard deviation and lower variance, the less spread out the values are.

In [93]:
```
# Standard deviation = a measure of how spread out a group of numbers is from the mean
np.std(a2)
```

Out[93]:
```
1.8226964152656422
```

In [94]:
```
# Variance = measure of the average degree to which each number is different to the mea
# Higher variance = wider range of numbers
# Lower variance = lower range of numbers
np.var(a2)
```

Out[94]:
```
3.3222222222222224
```

In [95]:
```
# Standard deviation = squareroot of variance
np.sqrt(np.var(a2))
```

Out[95]:
```
1.8226964152656422
```

In [96]:
```
# Demo of std and var
high_var_array = np.array([1, 100, 200, 300, 4000, 5000]) # wider range of numbers
low_var_array = np.array([2, 4, 6, 8, 10]) # lower range of numbers
```

In [97]:
```
# See the variance (See the difference)
np.var(high_var_array), np.var(low_var_array)
```

Out[97]:
```
(4296133.472222221, 8.0)
```

In [98]:
```
# See the standard deviation
np.std(high_var_array), np.std(low_var_array)
```

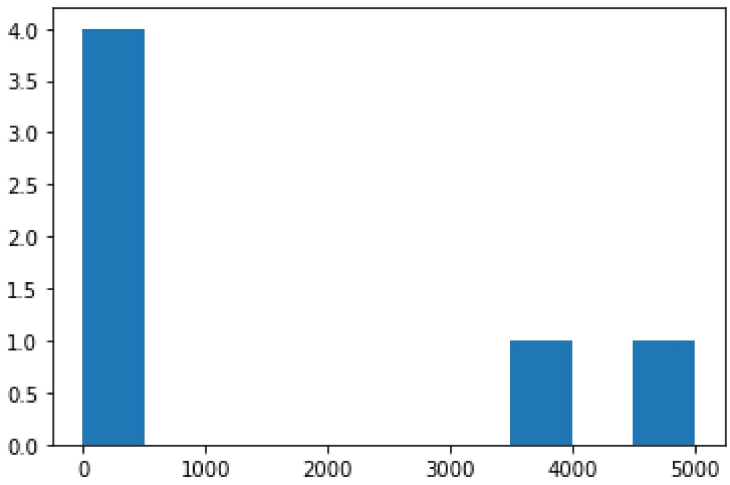
Out[98]:
```
(2072.711623024829, 2.8284271247461903)
```

In [99]:
```
# See the mean
np.mean(high_var_array), np.mean(low_var_array)
```

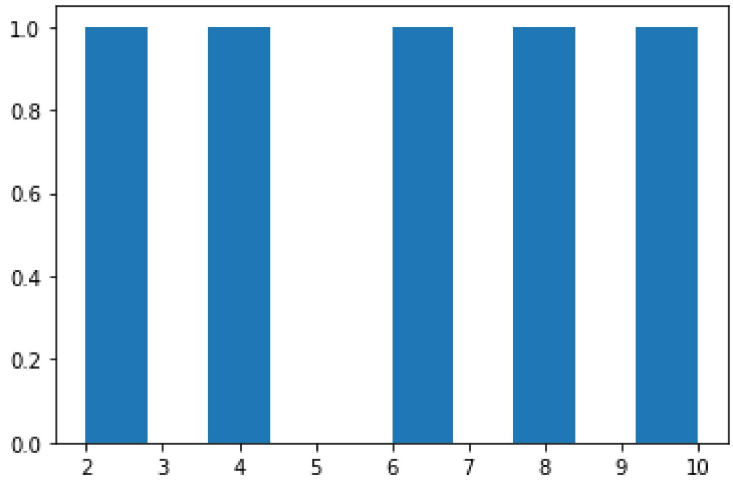
Out[99]:
```
(1600.1666666666667, 6.0)
```

In [100…
```
# See the variance graphically

%matplotlib inline
import matplotlib.pyplot as plt
plt.hist(high_var_array)
plt.show()
```



In [101…
```
plt.hist(low_var_array)
plt.show()
```



Reshaping & transposing

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

1. they are equal, or
2. one of them is 1

In [102…
```
a2
```

Out[102…
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [103…
```
a2.shape
```

Out[103…
```
(2, 3)
```

In [104…
```
a3
```

Out[104…
```
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],

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

In [105…
```
a3.shape
```

Out[105…
```
(2, 3, 3)
```

In [106…
```
# Cannot do operations on (2,3) and (2,3,3) arrays
a2 * a3
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_16256/42393092.py in <module>
```

```
      1 # Cannot do operations on (2,3) and (2,3,3) arrays
----> 2 a2 * a3

ValueError: operands could not be broadcast together with shapes (2,3) (2,3,3)
```

In [107…
```
a2
```

Out[107…
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [108…
```
a2.shape
```

Out[108…
```
(2, 3)
```

In [109…
```
a3.shape
```

Out[109…
```
(2, 3, 3)
```

In [110…
```
# Reshaping array (Convert inner most row to columns)
a2_reshape = a2.reshape(2, 3, 1)
a2_reshape
```

Out[110…
```
array([[[1. ],
        [2. ],
        [3.3]],

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

In [111…
```
a3
```

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

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

In [112…
```
# Do operations on (2,3,1) and (2,3,3) array => Valid
a2_reshape * a3
```

Out[112…
```
array([[[  1. ,   2. ,   3. ],
        [  8. ,  10. ,  12. ],
        [ 23.1,  26.4,  29.7]],

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

In [113…
```
a2
```

Out[113…
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [114…
```
a2.shape
```

Out[114…
```
(2, 3)
```

In [115…
```
# Transpose = switches the axis'
a2.T
```

Out[115…
```
array([[1. , 4. ],
       [2. , 5. ],
       [3.3, 6.5]])
```

In [116…
```
a2.T.shape
```

Out[116…
```
(3, 2)
```

In [117…
```
a3
```

Out[117…
```
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],

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

In [118…
```
a3.shape
```

Out[118…
```
(2, 3, 3)
```

In [119…
```
# Transpose of 3 dimensional array
# Reverse the shape of the array : 2, 3, 3  =>  3, 3, 2
a3.T
```

Out[119…
```
array([[[ 1, 10],
        [ 4, 13],
        [ 7, 16]],

       [[ 2, 11],
        [ 5, 14],
        [ 8, 17]],

       [[ 3, 12],
        [ 6, 15],
        [ 9, 18]]])
```

In [120…
```
a3.T.shape
```

Out[120…
```
(3, 3, 2)
```

Dot product

In [121…
```
np.random.seed(0)

mat1 = np.random.randint(10, size=(5, 3))
mat2 = np.random.randint(10, size=(5, 3))

mat1
```

Out[121…
```
array([[5, 0, 3],
       [3, 7, 9],
       [3, 5, 2],
       [4, 7, 6],
       [8, 8, 1]])
```

In [122…
```
mat2
```

Out[122…
```
array([[6, 7, 7],
       [8, 1, 5],
       [9, 8, 9],
       [4, 3, 0],
       [3, 5, 0]])
```

In [123…
```
mat1.shape, mat2.shape
```

Out[123…
```
((5, 3), (5, 3))
```

In [124…
```
mat1
```

Out[124…
```
array([[5, 0, 3],
       [3, 7, 9],
       [3, 5, 2],
       [4, 7, 6],
       [8, 8, 1]])
```

In [125…
```
mat2
```

Out[125…
```
array([[6, 7, 7],
       [8, 1, 5],
       [9, 8, 9],
       [4, 3, 0],
       [3, 5, 0]])
```

In [126…
```
# Element-wise multiplication (Hadamard product)
mat1 * mat2
```

Out[126…
```
array([[30,  0, 21],
       [24,  7, 45],
       [27, 40, 18],
       [16, 21,  0],
       [24, 40,  0]])
```

In [127…
```
# Dot product (Cannot do matrix multiplication on 5,3 and 5,3 matrices)
np.dot(mat1, mat2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_16256/157527875.py in <module>
      1 # Dot product (Cannot do matrix multiplication on 5,3 and 5,3 matrices)
----> 2 np.dot(mat1, mat2)

<__array_function__ internals> in dot(*args, **kwargs)

ValueError: shapes (5,3) and (5,3) not aligned: 3 (dim 1) != 5 (dim 0)
```

In [128…
```
# Do the matrix multiplication on 5,3 and 3,5 matrices
# Get transpose of mat2
mat3 = np.dot(mat1, mat2.T)
mat3
```

Out[128…
```
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]])
```

In [129…
```
mat3.shape
```

Out[129…
```
(5, 5)
```

Dot product exmaple (nut butter sales)

In [130…
```
np.random.seed(0)
# Number of jars sold
sales_amounts = np.random.randint(20, size=(5,3))
sales_amounts
```

Out[130…
```
array([[12, 15,  0],
       [ 3,  3,  7],
       [ 9, 19, 18],
       [ 4,  6, 12],
       [ 1,  6,  7]])
```

In [131…
```
# Create weekly_sales DataFrame using numpy array
# Give column names and index names
weekly_sales = pd.DataFrame(sales_amounts,
                            index=["Mon", "Tues", "Wed", "Thurs", "Fri"],
                            columns=["Almond butter", "Peanut butter", "Cashew butter"])
weekly_sales
```

Out[131…

	Almond butter	Peanut butter	Cashew butter
Mon	12	15	0
Tues	3	3	7
Wed	9	19	18
Thurs	4	6	12

	Almond butter	Peanut butter	Cashew butter
Fri	1	6	7

In [132…
```
# Create prices array
prices = np.array([10, 8, 12])
prices
```

Out[132…
```
array([10,  8, 12])
```

In [133…
```
# In numpy shape of the 1 array is looks like this
# If we pass this to pandas DataFrame, it is expecting 3 rows and 1 column
# There we need to reshape this
prices.shape
```

Out[133…
```
(3,)
```

In [134…
```
# Create butter_prices DataFrame
# use reshape
butter_prices = pd.DataFrame(prices.reshape(1, 3),
                             index=["Price"],
                             columns=["Almond butter", "Peanut butter", "Cashew butter"
butter_prices
```

Out[134…

	Almond butter	Peanut butter	Cashew butter
Price	10	8	12

In [135…
```
prices.shape
```

Out[135…
```
(3,)
```

In [136…
```
sales_amounts.shape
```

Out[136…
```
(5, 3)
```

In [137…
```
# Calculate total_sales using numpy arrays
# Need to use transpose
total_sales = prices.dot(sales_amounts)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_16256/3796268003.py in <module>
      1 # Calculate total_sales using numpy arrays
      2 # Need to use transpose
----> 3 total_sales = prices.dot(sales_amounts)

ValueError: shapes (3,) and (5,3) not aligned: 3 (dim 0) != 5 (dim 0)
```

In [138…
```
# Shape aren't aligned, let's transpose
```

```
# Calculate total_sales using numpy arrays
total_sales = prices.dot(sales_amounts.T)
total_sales
```

Out[138…
```
array([240, 138, 458, 232, 142])
```

In [139…
```
# Create daily_sales using DataFrames
butter_prices.shape, weekly_sales.shape
```

Out[139…
```
((1, 3), (5, 3))
```

In [140…
```
# Get dot product of butter_prices with transpose of weekly_sales
daily_sales = butter_prices.dot(weekly_sales.T)
daily_sales
```

Out[140…

	Mon	Tues	Wed	Thurs	Fri
Price	240	138	458	232	142

In [141…
```
weekly_sales.shape
```

Out[141…
```
(5, 3)
```

In [142…
```
weekly_sales
```

Out[142…

	Almond butter	Peanut butter	Cashew butter
Mon	12	15	0
Tues	3	3	7
Wed	9	19	18
Thurs	4	6	12
Fri	1	6	7

In [143…
```
# Append daily_sales column to weekly_sales table
# Need to use transpose(shape of weekly_sales is 5,3)
weekly_sales["Total ($)"] = daily_sales.T
weekly_sales
```

Out[143…

	Almond butter	Peanut butter	Cashew butter	Total ($)
Mon	12	15	0	240
Tues	3	3	7	138
Wed	9	19	18	458
Thurs	4	6	12	232
Fri	1	6	7	142

Comparison Operators

In [144…
```
a1
```

Out[144…
```
array([1, 2, 3])
```

In [145…
```
a2
```

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

In [146…
```
# Compare element wise
a1 > a2
```

Out[146…
```
array([[False, False, False],
       [False, False, False]])
```

In [147…
```
# Compare element wise and create a new array
bool_array = a1 >= a2
bool_array
```

Out[147…
```
array([[ True,  True, False],
       [False, False, False]])
```

In [148…
```
# Check the type of bool_array variable and type of the elements in the bool_array
type(bool_array), bool_array.dtype
```

Out[148…
```
(numpy.ndarray, dtype('bool'))
```

In [149…
```
# Compare each element in a1 with 5
a1 > 5
```

Out[149…
```
array([False, False, False])
```

In [150…
```
a1
```

Out[150…
```
array([1, 2, 3])
```

In [151…
```
a2
```

Out[151…
```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

In [152…
```
# Compare element wise (Need to compatible with shapes, like previously)
a1 == a2
```

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

5. Sorting arrays

In [153…
```
random_array = np.random.randint(10, size=(3, 5))
random_array
```

Out[153…
```
array([[7, 8, 1, 5, 9],
       [8, 9, 4, 3, 0],
       [3, 5, 0, 2, 3]])
```

In [154…
```
# Sort array row wise (Sort each row separately)
np.sort(random_array)
```

Out[154…
```
array([[1, 5, 7, 8, 9],
       [0, 3, 4, 8, 9],
       [0, 2, 3, 3, 5]])
```

In [155…
```
random_array
```

Out[155…
```
array([[7, 8, 1, 5, 9],
       [8, 9, 4, 3, 0],
       [3, 5, 0, 2, 3]])
```

In [156…
```
# Get the order of indices after sorting (Row wise)
np.argsort(random_array)
```

Out[156…
```
array([[2, 3, 0, 1, 4],
       [4, 3, 2, 0, 1],
       [2, 3, 0, 4, 1]], dtype=int64)
```

In [157…
```
a1
```

Out[157…
```
array([1, 2, 3])
```

In [158…
```
# Get the order of indices after sorting (Row wise)
np.argsort(a1)
```

Out[158…
```
array([0, 1, 2], dtype=int64)
```

In [159…
```
# Get the index of the minimum value
np.argmin(a1)
```

Out[159…
```
0
```

In [160…
```
# Get the index of the maximum value
np.argmax(a1)
```

Out[160…
```
2
```

In [161…
```
random_array
```

Out[161…
```
array([[7, 8, 1, 5, 9],
       [8, 9, 4, 3, 0],
       [3, 5, 0, 2, 3]])
```

In [162…
```
# Get the maximum indices of random_array in row wise
np.argmax(random_array, axis=0)
```

Out[162…
```
array([1, 1, 1, 0, 0], dtype=int64)
```

In [163…
```
# Get the maximum indices of random_array in column wise
np.argmax(random_array, axis=1)
```

Out[163…
```
array([4, 1, 1], dtype=int64)
```

6. Practical Example - NumPy in Action!!!!

In [164…
```
# Turn an image into a NumPy array
from matplotlib.image import imread

# convert image into a numpy array
panda = imread("images/panda.png")
print(type(panda))
```

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

In [165…
```
# 3 columns (last dimension) is for R, G, B
panda.size, panda.shape, panda.ndim
```

Out[165…
```
(24465000, (2330, 3500, 3), 3)
```

In [166…
```
# See first 5
panda[:5]
```

Out[166…
```
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 [167…
```
car = imread("images/car-photo.png")
print(type(car))
```

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

In [168…
```
# In this picture 4 columns (Last dimension)
car.size, car.shape, car.ndim
```

Out[168…
```
(991300, (431, 575, 4), 3)
```

In [169…
```
car[:1]
```

Out[169…
```
array([[[0.5019608 , 0.50980395, 0.4862745 , 1.        ],
        [0.3372549 , 0.34509805, 0.30588236, 1.        ],
        [0.20392157, 0.21568628, 0.14901961, 1.        ],
        ...,
        [0.64705884, 0.7058824 , 0.54901963, 1.        ],
        [0.59607846, 0.63529414, 0.45882353, 1.        ],
        [0.44705883, 0.47058824, 0.3372549 , 1.        ]]], dtype=float32)
```

In [170…
```
dog = imread("images/dog-photo.jpeg")
print(type(dog))
```

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