In [2]:
```
import numpy as np
```

Data Types and Attributes

In [3]:
```
#numpy main data types is ndarray
```

In [4]:
```
a1 = np.array([2,4,6,7])
```

In [5]:
```
print(a1)
```

```
[2 4 6 7]
```

In [6]:
```
type(a1)
```

Out[6]:
```
numpy.ndarray
```

In [7]:
```
a2 = np.array([[13,4,5,6],[7,8,9,2]])
```

In [8]:
```
a3 = np.array([[[1,2,3,4],[5,6,7,8],[1,4,6,7]],[[8,7,3,9],[3,9,1,0],[7,5,8,4]]])
```

In [9]:
```
a2
```

Out[9]:
```
array([[13,  4,  5,  6],
       [ 7,  8,  9,  2]])
```

In [10]:
```
a2.shape
```

Out[10]:
```
(2, 4)
```

In [11]:
```
a1.ndim, a2.ndim
```

Out[11]:
```
(1, 2)
```

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

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

       [[8, 7, 3, 9],
        [3, 9, 1, 0],
        [7, 5, 8, 4]]])
```

In [13]:
```
a3.shape
```

Out[13]:
```
(2, 3, 4)
```

In [14]:
```
a1.ndim, a2.ndim, a3.ndim
```

Out[14]:
```
(1, 2, 3)
```

In [15]:
```
a1.dtype, a2.dtype, a3.dtype
```

Out[15]:
```
(dtype('int32'), dtype('int32'), dtype('int32'))
```

In [16]:
```
a1.size, a2.size, a3.size
```

Out[16]:
```
(4, 8, 24)
```

In [17]:
```
# craeting dataframe from NumPy
```

In [18]:
```
import pandas as pd
df = pd.DataFrame(a2)
```

In [19]:
```
df
```

Out[19]:

	0	1	2	3
0	13	4	5	6
1	7	8	9	2

In [20]:
```
for_one = np.ones((2,3),dtype=int)
```

In [21]:
```
for_one
```

Out[21]:
```
array([[1, 1, 1],
       [1, 1, 1]])
```

In [22]:
```
for_zero = np.zeros((2,3), dtype=int)
```

In [23]:
```
for_zero
```

Out[23]:
```
array([[0, 0, 0],
       [0, 0, 0]])
```

In [24]:
```
# here we have to define range and interval
range_array = np.arange(2,10,2)
```

In [25]:
```
range_array
```

Out[25]:
```
array([2, 4, 6, 8])
```

In [26]:
```
# here we define the parameter low and high (high is exclusive while low is inclus
random_array = np.random.randint(2,10,size=(3,3))
random_array
```

Out[26]:
```
array([[4, 6, 7],
       [2, 3, 4],
       [7, 2, 2]])
```

In [27]:
```
# give random numbers between 0 and 1
random_array_2 = np.random.random((3,4))
random_array_2
```

Out[27]:
```
array([[0.11351456, 0.07132583, 0.70304168, 0.4773937 ],
       [0.42524661, 0.91561328, 0.98511126, 0.51936808],
       [0.15333297, 0.87597207, 0.55445736, 0.38961997]])
```

In [28]:
```
random_array_3 = np.random.rand(3,4)
random_array
```

Out[28]:
```
array([[4, 6, 7],
       [2, 3, 4],
       [7, 2, 2]])
```

In [29]:
```
# Pseudo random number
np.random.seed(5)
random_array_4 = np.random.randint(8,size=(3,4))
random_array_4
```

Out[29]:
```
array([[3, 6, 7, 5],
       [6, 6, 0, 1],
       [0, 4, 7, 6]])
```

In [30]:
```
np.random.seed(5)
random_array_5 = np.random.random((4,5))
random_array_5
```

Out[30]:
```
array([[0.22199317, 0.87073231, 0.20671916, 0.91861091, 0.48841119],
       [0.61174386, 0.76590786, 0.51841799, 0.2968005 , 0.18772123],
       [0.08074127, 0.7384403 , 0.44130922, 0.15830987, 0.87993703],
       [0.27408646, 0.41423502, 0.29607993, 0.62878791, 0.57983781]])
```

viewing array and matrices

In [31]:
```
np.unique(random_array_4)
```

Out[31]:
```
array([0, 1, 3, 4, 5, 6, 7])
```

In [32]:
```
a1[0]
```

Out[32]:
```
2
```

In [33]:
```
a2[1]
```

Out[33]:
```
array([7, 8, 9, 2])
```

In [34]:
```
a3
```

Out[34]:
```
array([[[1, 2, 3, 4],
        [5, 6, 7, 8],
        [1, 4, 6, 7]],

       [[8, 7, 3, 9],
        [3, 9, 1, 0],
        [7, 5, 8, 4]]])
```

In [35]:
```
a3[:3,:2,:3]
```

Out[35]:
```
array([[[1, 2, 3],
        [5, 6, 7]],

       [[8, 7, 3],
        [3, 9, 1]]])
```

In [36]:
```
a4 = np.random.randint(10,size=(2,3,4,5))
a4
```

Out[36]:
```
array([[[[1, 2, 7, 0, 5],
         [0, 0, 4, 4, 9],
         [3, 2, 4, 6, 9],
         [3, 3, 2, 1, 5]],

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

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


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

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

        [[0, 2, 2, 1, 1],
         [7, 1, 7, 2, 6],
         [3, 6, 8, 0, 9],
         [1, 9, 0, 8, 7]]]])
```

Manipulating &comparing array

In [37]:
```
a1
```

Out[37]:
```
array([2, 4, 6, 7])
```

In [38]:
```
ones = np.ones(4)
ones
```

Out[38]:
```
array([1., 1., 1., 1.])
```

In [39]:
```
a1 + ones
```

Out[39]:
```
array([3., 5., 7., 8.])
```

In [40]:
```
zeros = np.zeros(4)
zeros
```

Out[40]:
```
array([0., 0., 0., 0.])
```

In [41]:
```
a1 + zeros
```

Out[41]:
```
array([2., 4., 6., 7.])
```

In [42]:
```
a2
```

Out[42]:
```
array([[13,  4,  5,  6],
       [ 7,  8,  9,  2]])
```

In [43]:
```
a1*a2
```

Out[43]:
```
array([[26, 16, 30, 42],
       [14, 32, 54, 14]])
```

In [44]:
```
a2*a1
```

Out[44]:
```
array([[26, 16, 30, 42],
       [14, 32, 54, 14]])
```

In [45]:
```
a1+ones
```

Out[45]:
```
array([3., 5., 7., 8.])
```

In [46]:
```
np.add(a1,ones)
```

Out[46]:
```
array([3., 5., 7., 8.])
```

In [47]:
```
a1%2
```

Out[47]:
```
array([0, 0, 0, 1], dtype=int32)
```

In [48]:
```
a1 //2
```

Out[48]:
```
array([1, 2, 3, 3], dtype=int32)
```

In [49]:
```
a1 **2
```

Out[49]:
```
array([ 4, 16, 36, 49])
```

In [50]:
```
np.square(a1)
```

Out[50]:
```
array([ 4, 16, 36, 49])
```

In [51]:
```
np.exp(a1)
```

Out[51]:
```
array([   7.3890561 ,   54.59815003,  403.42879349, 1096.63315843])
```

In [52]:
```
np.log(a1)
```

Out[52]:
```
array([0.69314718, 1.38629436, 1.79175947, 1.94591015])
```

In [53]:
```
np.sin(a1)
```

Out[53]:
```
array([ 0.90929743, -0.7568025 , -0.2794155 ,  0.6569866 ])
```

In [54]:
```
#aggregation
#aggregation  - performing the same operation on number of things
```

In [55]:
```
listy_list = [1,2,3,4]
type(listy_list)
```

Out[55]:
```
list
```

In [56]:
```
sum(listy_list)
```

Out[56]:
```
10
```

In [57]:
```
sum(a1)
```

Out[57]:
```
19
```

In [58]:
```
np.sum(a1)
```

Out[58]:
```
19
```

In [59]:
```
# create a massive array
massive_array = np.random.random(10000)
```

In [60]:
```
massive_array.size
```

Out[60]:
```
10000
```

In [61]:
```
massive_array[:100]
```

Out[61]:
```
array([0.96515755, 0.76993268, 0.75909912, 0.71004905, 0.70155283,
       0.76733138, 0.97434948, 0.37371452, 0.08305444, 0.23964044,
       0.22148276, 0.3635998 , 0.81031423, 0.06009617, 0.44974356,
       0.81317053, 0.26423837, 0.06340098, 0.24210767, 0.08507046,
       0.80777744, 0.17025008, 0.19534463, 0.81464201, 0.81028553,
       0.58937388, 0.91473434, 0.05982164, 0.96499664, 0.57097522,
       0.30251811, 0.82570583, 0.65941728, 0.98650144, 0.10748953,
       0.58091853, 0.47282816, 0.65227079, 0.24185905, 0.03113006,
       0.54423235, 0.36471018, 0.89233328, 0.45934559, 0.4186541 ,
       0.63290554, 0.52717883, 0.96121114, 0.78901011, 0.49669551,
       0.211141  , 0.60398414, 0.74857581, 0.75586383, 0.99350401,
       0.21849778, 0.42563733, 0.4238496 , 0.31864644, 0.36533302,
       0.47794131, 0.54209041, 0.26523025, 0.13436373, 0.3018861 ,
       0.13648009, 0.3177002 , 0.67918268, 0.60134116, 0.99722547,
       0.56091459, 0.5487262 , 0.64234954, 0.72678937, 0.61581354,
       0.58850006, 0.60500431, 0.35478546, 0.77738199, 0.60460331,
       0.30923077, 0.245631  , 0.07126598, 0.34772917, 0.01284807,
       0.16564183, 0.06036643, 0.27815593, 0.34833199, 0.59122231,
       0.77501313, 0.62475447, 0.15968093, 0.89376265, 0.73059127,
       0.49776802, 0.64364022, 0.50947509, 0.16820816, 0.64146477])
```

In [62]:
```
# finding out difference between python's '.sum' and NumPy's 'np.sum'
```

In [63]:
```
%timeit sum(massive_array)    #Pandas sum
%timeit np.sum(massive_array) #NumPy's sum
```

```
1.03 ms ± 4.56 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
13.5 µs ± 693 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

In [64]:
```
np.sum(a2)
```

Out[64]:
```
54
```

In [65]:
```
np.mean(a2)
```

Out[65]:
```
6.75
```

In [66]:
```
np.max(a2)
```

Out[66]:
```
13
```

In [67]:
```
np.min(a2)
```

Out[67]:
```
2
```

In [68]:
```
# standard deviation - how spread out a group of numbers is from the mean
np.std(a2)
```

Out[68]:
```
3.152380053229623
```

In [69]:
```
# variance - measure of the average degree to which each number is different to mea
# higher variance - wider range of numbers
# lower variance - lower range of numbers
np.var(a2)
```

Out[69]:
```
9.9375
```

In [70]:
```
#standard deviation is square root of variance
np.sqrt(np.var(a2))
```

Out[70]:
```
3.152380053229623
```

In [71]:
```
# demo of std and var
```

In [72]:
```
high_var_array = [1,100,200,300,4000,5000]
low_var_array = [2,4,6,8,10,12]
```

In [73]:
```
np.var(high_var_array), np.var(low_var_array)
```

Out[73]:
```
(4296133.472222221, 11.666666666666666)
```

In [74]:
```
np.std(high_var_array), np.std(low_var_array)
```

Out[74]:
```
(2072.711623024829, 3.415650255319866)
```

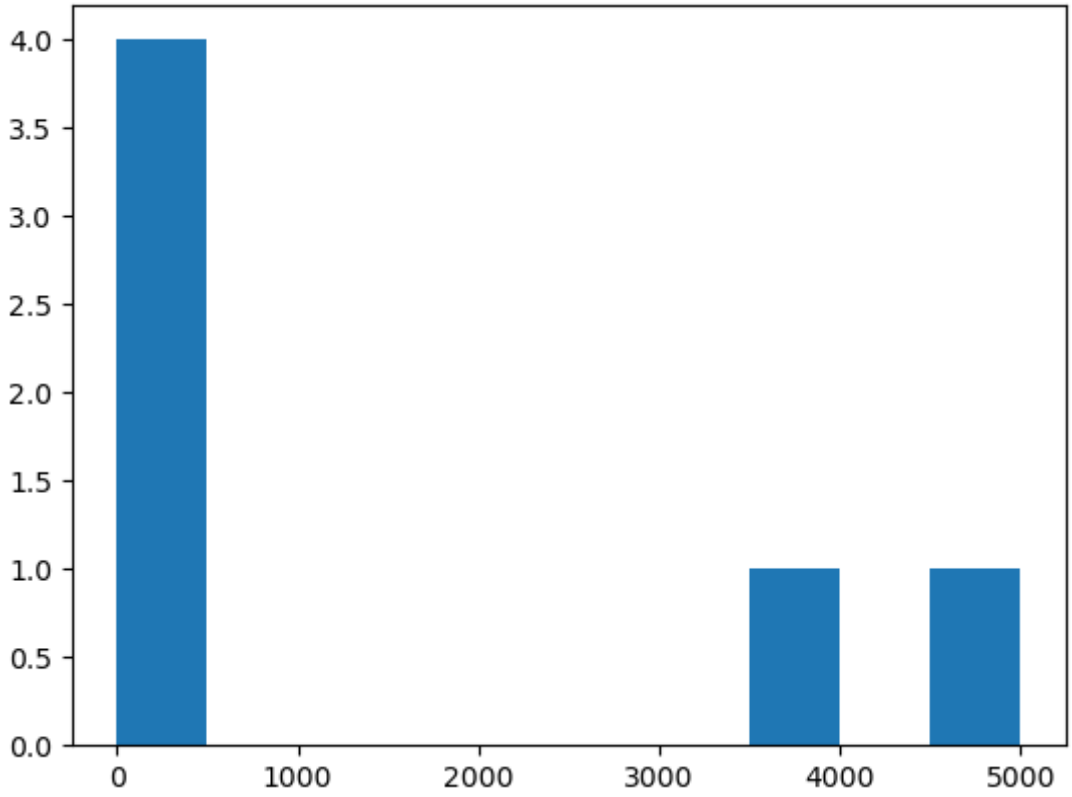
In [75]:
```
np.mean(high_var_array), np.mean(low_var_array)
```

Out[75]:
```
(1600.1666666666667, 7.0)
```

In [76]:
```
%matplotlib inline
import matplotlib.pyplot as plt
```

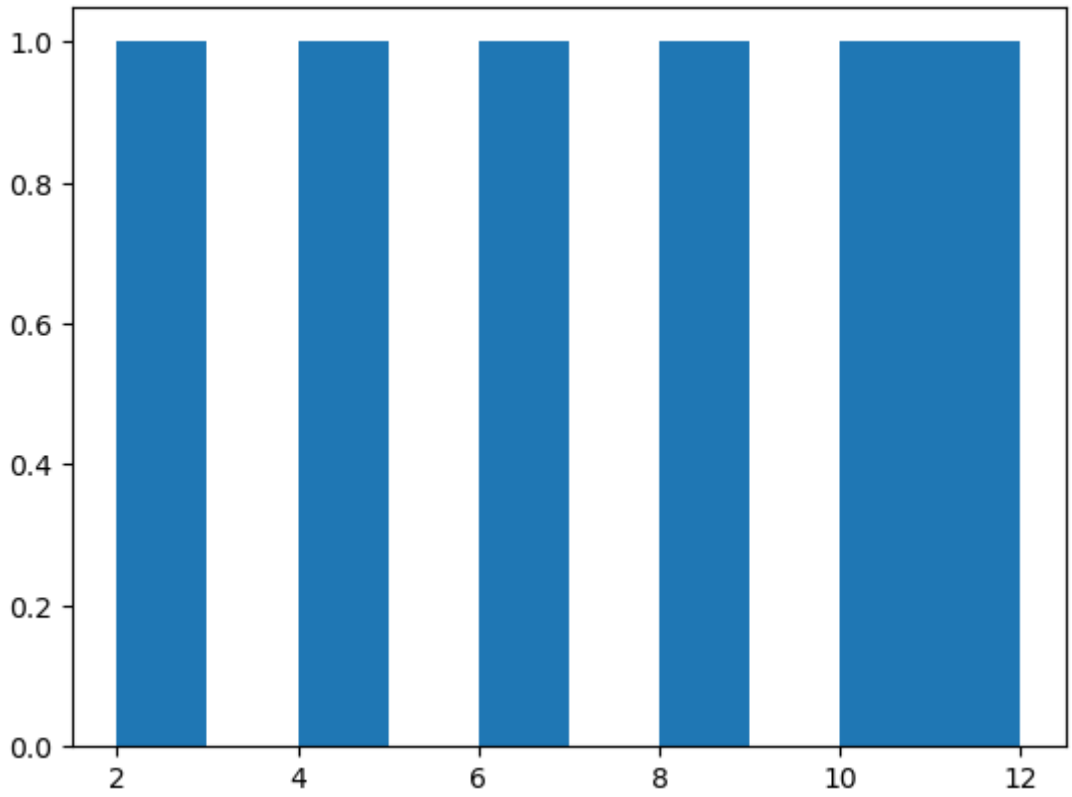
In [77]:
```
plt.hist(high_var_array)
```

Out[77]:
```
(array([4., 0., 0., 0., 0., 0., 0., 1., 0., 1.]),
 array([1.0000e+00, 5.0090e+02, 1.0008e+03, 1.5007e+03, 2.0006e+03,
        2.5005e+03, 3.0004e+03, 3.5003e+03, 4.0002e+03, 4.5001e+03,
        5.0000e+03]),
 <BarContainer object of 10 artists>)
```



In [78]:
```
plt.show()
```

In [79]:
```
plt.hist(low_var_array)
plt.show()
```



Reshape and Transpose

In [80]:
```
a2lite = np.random.randint(10,size=(2,3))
```

In [81]:
```
a2lite * a3
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[81], line 1
----> 1 a2lite * a3

ValueError: operands could not be broadcast together with shapes (2,3) (2,3,4)
```

In [82]:
```
a2lite.shape, a3.shape
```

Out[82]:
```
((2, 3), (2, 3, 4))
```

In [83]:
```
a2lite_reshape = a2lite.reshape(2,3,1)
```

In [84]:
```
a2lite_reshape * a3
```

Out[84]:
```
array([[[ 7, 14, 21, 28],
        [30, 36, 42, 48],
        [ 4, 16, 24, 28]],

       [[56, 49, 21, 63],
        [ 6, 18,  2,  0],
        [49, 35, 56, 28]]])
```

In [85]:
```
# transpose
```

In [86]:
```
a2.shape
```

Out[86]:
```
(2, 4)
```

In [87]:
```
a2
```

Out[87]:
```
array([[13,  4,  5,  6],
       [ 7,  8,  9,  2]])
```

In [88]:
```
a2.T
```

Out[88]:
```
array([[13,  7],
       [ 4,  8],
       [ 5,  9],
       [ 6,  2]])
```

In [89]:
```
a2.T.shape
```

Out[89]:
```
(4, 2)
```

dot Product

In [99]:
```
np.random.seed(7)
m1 = np.random.randint(10,size=(3,4))
m2 = np.random.randint(10,size=(3,4))
```

In [100…
```
m1,m2
```

Out[100]:
```
(array([[4, 9, 6, 3],
        [3, 7, 7, 9],
        [7, 8, 9, 8]]),
 array([[7, 6, 4, 0],
        [7, 0, 7, 6],
        [3, 5, 8, 8]]))
```

In [102…
```
#Hadamard Product (Element wise multiplication)
m1*m2
```

Out[102]:
```
array([[28, 54, 24,  0],
       [21,  0, 49, 54],
       [21, 40, 72, 64]])
```

In [103…
```
np.dot(m1,m2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[103], line 1
----> 1 np.dot(m1,m2)

File <__array_function__ internals>:180, in dot(*args, **kwargs)

ValueError: shapes (3,4) and (3,4) not aligned: 4 (dim 1) != 3 (dim 0)
```

In [106…
```
# m1_dotmatrix is shape of(3,3) and m2_dot matrix is shape of (3,2)
m1_dot = np.random.randint(10,size=(3,3))
m2_dot = np.random.randint(10,size=(3,2))
```

In [107…
```
#dot product
np.dot(m1_dot,m2_dot)
```

Out[107]:
```
array([[  7,   7],
       [ 42,  77],
       [ 63, 127]])
```

In [108…
```
# or we can also use previous (m1,m2 ) matrix to perform dot product via Transposi
# shape is (3,4) so if we will transpose m2 then m1 column and transposed m2 rows

m2_transpose = m1.T
np.dot(m1,m2_transpose)
```

Out[108]:
```
array([[142, 144, 178],
       [144, 188, 212],
       [178, 212, 258]])
```

Dot product example (nut butter sale)

In [144…
```
np.random.seed(0)
butter_sale = np.random.randint(2,20,size=(7,3))
butter_sale
```

Out[144]:
```
array([[14, 17,  2],
       [ 5,  5,  9],
       [11,  6,  8],
       [14,  3,  8],
       [ 9, 16, 19],
       [ 7, 15, 10],
       [11, 18,  7]])
```

In [145…
```
# creating dataframe
weekly_sales = pd.DataFrame(butter_sale,index=["Monday","Tuesday","Wednesday","Thu
```

In [146…
```
weekly_sales
```

Out[146]:

	Peanut Butter	Almond Butter	Cashew Butter
Monday	14	17	2
Tuesday	5	5	9
Wednesday	11	6	8
Thursday	14	3	8
Friday	9	16	19
Saturday	7	15	10
Sunday	11	18	7

In [147…
```
price = np.array([8,10,13])
```

In [148…
```
butter_price = pd.DataFrame(price.reshape(1,3),index=["Price"],columns=["Peanut Bu
```

In [149…
```
butter_price
```

Out[149]:

	Peanut Butter	Almond Butter	Cashew Butter
Price	8	10	13

In [150…
```
butter_price.shape
```

Out[150]:
```
(1, 3)
```

In [151…
```
weekly_sales.shape
```

Out[151]:
```
(7, 3)
```

In [159…
```
weekly_sales['Total Price($)'] = np.dot(butter_sale,price.T)
weekly_sales
```

Out[159]:

	Peanut Butter	Almond Butter	Cashew Butter	Total Price($)
Monday	14	17	2	308
Tuesday	5	5	9	207
Wednesday	11	6	8	252
Thursday	14	3	8	246
Friday	9	16	19	479
Saturday	7	15	10	336
Sunday	11	18	7	359

In [163…
```
combined = pd.concat([weekly_sales,butter_price])
combined
```

Out[163]:

	Peanut Butter	Almond Butter	Cashew Butter	Total Price($)
Monday	14	17	2	308.0
Tuesday	5	5	9	207.0
Wednesday	11	6	8	252.0
Thursday	14	3	8	246.0
Friday	9	16	19	479.0
Saturday	7	15	10	336.0
Sunday	11	18	7	359.0
Price	8	10	13	NaN

Comparison operator

In [165…
```
a1
```

Out[165]:
```
array([2, 4, 6, 7])
```

In [166…
```
a2
```

Out[166]:
```
array([[13,  4,  5,  6],
       [ 7,  8,  9,  2]])
```

In [167…
```
a1<a2
```

Out[167]:
```
array([[ True, False, False, False],
       [ True,  True,  True, False]])
```

In [168…
```
a1 <= a2
```

Out[168]:
```
array([[ True,  True, False, False],
       [ True,  True,  True, False]])
```

In [169…
```
a1<=a2
```

Out[169]:
```
array([[ True,  True, False, False],
       [ True,  True,  True, False]])
```

In [170…
```
a1 == a2
```

Out[170]:
```
array([[False,  True, False, False],
       [False, False, False, False]])
```

In [172…
```
bool_array = a1<=a2
bool_array
```

Out[172]:
```
array([[ True,  True, False, False],
       [ True,  True,  True, False]])
```

In [175…
```
type(bool_array), bool_array.dtype
```

Out[175]:
```
(numpy.ndarray, dtype('bool'))
```

Sorting

In [177…
```
np.random.seed(0)
unsorted_array = np.random.randint(10,size=(3,5))
unsorted_array
```

Out[177]:
```
array([[5, 0, 3, 3, 7],
       [9, 3, 5, 2, 4],
       [7, 6, 8, 8, 1]])
```

In [178…
```
np.sort(unsorted_array)
```

Out[178]:
```
array([[0, 3, 3, 5, 7],
       [2, 3, 4, 5, 9],
       [1, 6, 7, 8, 8]])
```

In [179…
```
np.argsort(unsorted_array)
```

Out[179]:
```
array([[1, 2, 3, 0, 4],
       [3, 1, 4, 2, 0],
       [4, 1, 0, 2, 3]], dtype=int64)
```

In [181…
```
np.argmax(unsorted_array, axis=1)
```

Out[181]:
```
array([4, 0, 2], dtype=int64)
```

In [182…
```
np.argmax(unsorted_array, axis=0)
```

Out[182]:
```
array([1, 2, 2, 2, 0], dtype=int64)
```

In [183…
```
np.argmin(unsorted_array, axis=1)
```

Out[183]:
```
array([1, 3, 4], dtype=int64)
```

In [184…
```
np.argmin(unsorted_array,axis=0)
```

Out[184]:
```
array([0, 0, 0, 1, 2], dtype=int64)
```

converting images into array

In [200…
```
from matplotlib.image import imread
```

In [201…
```
panda_array = imread("NumPy Project images (for conversion into array)/panda.png")
```

In [202…
```
panda_array
```

Out[202]:
```
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]],

       ...,

       [[0.13333334, 0.07450981, 0.05490196],
        [0.12156863, 0.0627451 , 0.04313726],
        [0.10980392, 0.05098039, 0.03137255],
        ...,
        [0.02745098, 0.02745098, 0.03529412],
        [0.02745098, 0.02745098, 0.03529412],
        [0.02745098, 0.02745098, 0.03529412]],

       [[0.13333334, 0.07450981, 0.05490196],
        [0.12156863, 0.0627451 , 0.04313726],
        [0.12156863, 0.0627451 , 0.04313726],
        ...,
        [0.02352941, 0.02352941, 0.03137255],
        [0.02352941, 0.02352941, 0.03137255],
        [0.02352941, 0.02352941, 0.03137255]],

       [[0.13333334, 0.07450981, 0.05490196],
        [0.12156863, 0.0627451 , 0.04313726],
        [0.12156863, 0.0627451 , 0.04313726],
        ...,
        [0.02352941, 0.02352941, 0.03137255],
        [0.02352941, 0.02352941, 0.03137255],
        [0.02352941, 0.02352941, 0.03137255]]], dtype=float32)
```

In [203…
```
panda_array.size, panda_array.ndim, panda_array.dtype
```

Out[203]:
```
(24465000, 3, dtype('float32'))
```

In [204…
```
dog_img_array = imread("NumPy Project images (for conversion into array)/dog-photo
```

In [205…
```
dog_img_array
```

Out[205]:
```
array([[[0.70980394, 0.80784315, 0.88235295, 1.        ],
        [0.72156864, 0.8117647 , 0.8862745 , 1.        ],
        [0.7411765 , 0.8156863 , 0.8862745 , 1.        ],
        ...,
        [0.49803922, 0.6862745 , 0.8392157 , 1.        ],
        [0.49411765, 0.68235296, 0.8392157 , 1.        ],
        [0.49411765, 0.68235296, 0.8352941 , 1.        ]],

       [[0.69411767, 0.8039216 , 0.8862745 , 1.        ],
        [0.7019608 , 0.8039216 , 0.88235295, 1.        ],
        [0.7058824 , 0.80784315, 0.88235295, 1.        ],
        ...,
        [0.5019608 , 0.6862745 , 0.84705883, 1.        ],
        [0.49411765, 0.68235296, 0.84313726, 1.        ],
        [0.49411765, 0.68235296, 0.8392157 , 1.        ]],

       [[0.6901961 , 0.8       , 0.88235295, 1.        ],
        [0.69803923, 0.8039216 , 0.88235295, 1.        ],
        [0.7058824 , 0.80784315, 0.88235295, 1.        ],
        ...,
        [0.5019608 , 0.6862745 , 0.84705883, 1.        ],
        [0.49803922, 0.6862745 , 0.84313726, 1.        ],
        [0.49803922, 0.6862745 , 0.84313726, 1.        ]],

       ...,

       [[0.9098039 , 0.81960785, 0.654902  , 1.        ],
        [0.8352941 , 0.7490196 , 0.6509804 , 1.        ],
        [0.72156864, 0.6313726 , 0.5372549 , 1.        ],
        ...,
        [0.01568628, 0.07058824, 0.02352941, 1.        ],
        [0.03921569, 0.09411765, 0.03529412, 1.        ],
        [0.03921569, 0.09019608, 0.05490196, 1.        ]],

       [[0.9137255 , 0.83137256, 0.6784314 , 1.        ],
        [0.8117647 , 0.7294118 , 0.627451  , 1.        ],
        [0.65882355, 0.5686275 , 0.47843137, 1.        ],
        ...,
        [0.00392157, 0.05490196, 0.03529412, 1.        ],
        [0.03137255, 0.09019608, 0.05490196, 1.        ],
        [0.04705882, 0.10588235, 0.06666667, 1.        ]],

       [[0.9137255 , 0.83137256, 0.68235296, 1.        ],
        [0.76862746, 0.68235296, 0.5882353 , 1.        ],
        [0.59607846, 0.5058824 , 0.44313726, 1.        ],
        ...,
        [0.03921569, 0.10196079, 0.07058824, 1.        ],
        [0.02745098, 0.08235294, 0.05882353, 1.        ],
        [0.05098039, 0.11372549, 0.07058824, 1.        ]]], dtype=float32)
```

In [206…
```
dog_img_array.size, dog_img_array.shape, dog_img_array.ndim, dog_img_array.dtype
```

Out[206]:
```
(993600, (432, 575, 4), 3, dtype('float32'))
```

In [207…
```
car_img_array = imread("NumPy Project images (for conversion into array)/car-photo
```

In [208…
```
car_img_array
```

Out[208]:
```
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.        ]],

       [[0.44313726, 0.43529412, 0.40392157, 1.        ],
        [0.3137255 , 0.31764707, 0.27450982, 1.        ],
        [0.2       , 0.21176471, 0.14117648, 1.        ],
        ...,
        [0.5058824 , 0.5372549 , 0.4117647 , 1.        ],
        [0.49803922, 0.52156866, 0.39607844, 1.        ],
        [0.4       , 0.42745098, 0.34117648, 1.        ]],

       [[0.39607844, 0.38039216, 0.34117648, 1.        ],
        [0.31764707, 0.3137255 , 0.27450982, 1.        ],
        [0.28627452, 0.29411766, 0.24705882, 1.        ],
        ...,
        [0.44705883, 0.45882353, 0.32156864, 1.        ],
        [0.45882353, 0.48235294, 0.3529412 , 1.        ],
        [0.4509804 , 0.49019608, 0.38039216, 1.        ]],

       ...,

       [[0.47058824, 0.57254905, 0.6313726 , 1.        ],
        [0.4392157 , 0.53333336, 0.5882353 , 1.        ],
        [0.48235294, 0.5803922 , 0.6392157 , 1.        ],
        ...,
        [0.6156863 , 0.7529412 , 0.827451  , 1.        ],
        [0.61960787, 0.7607843 , 0.83137256, 1.        ],
        [0.5921569 , 0.73333335, 0.7921569 , 1.        ]],

       [[0.4745098 , 0.5803922 , 0.6392157 , 1.        ],
        [0.47058824, 0.58431375, 0.63529414, 1.        ],
        [0.4117647 , 0.5019608 , 0.5529412 , 1.        ],
        ...,
        [0.54901963, 0.6784314 , 0.74509805, 1.        ],
        [0.65882355, 0.8117647 , 0.8862745 , 1.        ],
        [0.60784316, 0.74509805, 0.8117647 , 1.        ]],

       [[0.4745098 , 0.57254905, 0.627451  , 1.        ],
        [0.49411765, 0.6       , 0.65882355, 1.        ],
        [0.49019608, 0.5921569 , 0.64705884, 1.        ],
        ...,
        [0.5294118 , 0.63529414, 0.69803923, 1.        ],
        [0.5529412 , 0.67058825, 0.7372549 , 1.        ],
        [0.6156863 , 0.73333335, 0.8       , 1.        ]]], dtype=float32)
```

In [ ]: