numpy_introduction

December 23, 2025

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

0.1 Datatypes & Attributes

[2]:
```
# Numpy main datatype is ndarray -> n dimensional array
a1 = np.array([1,2,3])
a1
```

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

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

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

[4]:
```
a2 = np.array([[1,2,3],[4,5.4,6.4]])

a3 = np.array([[1,2,3],[4,5.4,6.4],[11,22,33]])
```

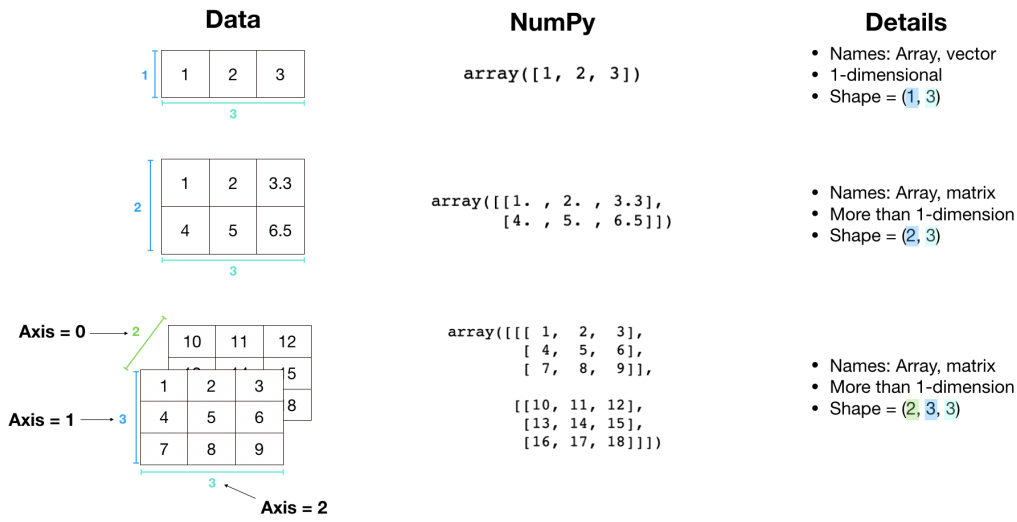
[5]:
```
a2
```

[5]:
```
array([[1. , 2. , 3. ],
       [4. , 5.4, 6.4]])
```

[6]:
```
a3
```

[6]:
```
array([[ 1. ,  2. ,  3. ],
       [ 4. ,  5.4,  6.4],
       [11. , 22. , 33. ]])
```

Anatomy of a NumPy array



[7]:
```
# Attributes

a1.shape
```

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

[8]:
```
a2.shape #2 rows and 3 columns
```

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

[9]:
```
a3.shape
```

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

[10]:
```
a1.ndim, a2.ndim, a3.ndim # number of dimensions
```

[10]:
```
(1, 2, 2)
```

[11]:
```
a1.dtype, a2.dtype, a3.dtype
```

[11]:
```
(dtype('int64'), dtype('float64'), dtype('float64'))
```

[12]:
```
a1.size, a2.size, a3.size # number of elements
```

[12]:
```
(3, 6, 9)
```

[13]:
```
type(a2),type(a1)
```

[13]:
```
(numpy.ndarray, numpy.ndarray)
```

[14]:
```
# Create a dataframe from numPy array
import pandas as pd

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

[14]:

	0	1	2
0	1.0	2.0	3.0
1	4.0	5.4	6.4

0.2 2.Creating Arrays

[15]:
```
sample_array = np.array([1,2,3])
sample_array
```

[15]:
```
array([1, 2, 3])
```

[16]:
```
sample_array.dtype
```

[16]:
```
dtype('int64')
```

[17]:
```
ones = np.ones((2,3)) #Return a new array of given shape and type, filled with␣
↪ones.
ones
```

[17]:
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

[18]:
```
ones.dtype
```

[18]:
```
dtype('float64')
```

[19]:
```
type(ones)
```

[19]:
```
numpy.ndarray
```

[20]:
```
zeros = np.zeros((2,3)) ##Return a new array of given shape and type, filled␣
↪with zeros.
zeros
```

[20]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

[21]:
```
range_array = np.arange(0,10,2)
range_array
```

[21]:
```
array([0, 2, 4, 6, 8])
```

[22]:
```
random_array = np.random.randint(0,10,size=(3,5))
random_array
```

[22]:
```
array([[5, 9, 0, 9, 0],
       [5, 8, 9, 2, 9],
       [9, 9, 6, 0, 4]], dtype=int32)
```

[23]:
```
random_array.size
```

[23]:
```
15
```

[24]:
```
random_array.shape
```

[24]:
```
(3, 5)
```

[25]:
```
random_array_2 =np.random.random((5,3))
```

[26]:
```
random_array_2
```

[26]:
```
array([[0.61596038, 0.3983683 , 0.93923186],
       [0.84746258, 0.1496947 , 0.32426165],
       [0.42687376, 0.91447911, 0.51702518],
       [0.76914842, 0.90265542, 0.7927374 ],
       [0.63775467, 0.54112805, 0.32653233]])
```

[27]:
```
random_array_2.shape
```

[27]:
```
(5, 3)
```

[28]:
```
random_array_3 = np.random.rand(5,3)
```

[29]:
```
random_array_3
```

[29]:
```
array([[0.10695915, 0.22946684, 0.08465808],
       [0.55546917, 0.33517269, 0.61499288],
       [0.82264613, 0.72205008, 0.81184655],
       [0.07587898, 0.2204324 , 0.19228433],
       [0.31720758, 0.2418619 , 0.15043071]])
```

[30]:
```
# Pseudo-random numbers
np.random.seed(seed=0) #create random numbers but take them along the seed
random_array_4 = np.random.randint(10, size=(5,3))
random_array_4
```

[30]:
```
array([[5, 0, 3],
       [3, 7, 9],
       [3, 5, 2],
       [4, 7, 6],
```

```
       [8, 8, 1]], dtype=int32)
```

[31]:
```
random_array_4.shape
```

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

[32]:
```
np.random.seed(seed=0)
random_array_5 = np.random.random((5,3))
random_array_5
```

[32]:
```
array([[0.5488135 , 0.71518937, 0.60276338],
       [0.54488318, 0.4236548 , 0.64589411],
       [0.43758721, 0.891773  , 0.96366276],
       [0.38344152, 0.79172504, 0.52889492],
       [0.56804456, 0.92559664, 0.07103606]])
```

[33]:
```
random_array_5 = np.random.random((5,3))
random_array_5
```

[33]:
```
array([[0.0871293 , 0.0202184 , 0.83261985],
       [0.77815675, 0.87001215, 0.97861834],
       [0.79915856, 0.46147936, 0.78052918],
       [0.11827443, 0.63992102, 0.14335329],
       [0.94466892, 0.52184832, 0.41466194]])
```

[34]:
```
np.unique(random_array_4)
```

[34]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int32)
```

0.3 3.Viewing arrays and matrices

[35]:
```
a1
```

[35]:
```
array([1, 2, 3])
```

[36]:
```
a2
```

[36]:
```
array([[1. , 2. , 3. ],
       [4. , 5.4, 6.4]])
```

[37]:
```
a1[0]
```

[37]:
```
np.int64(1)
```

[38]:
```
a2.shape
```

[38]:
```
(2, 3)
```

[39]:
```
a2[0]
```

[39]:
```
array([1., 2., 3.])
```

[40]:
```
a2[1]
```

[40]:
```
array([4. , 5.4, 6.4])
```

[41]:
```
a2[:2, :2]
```

[41]:
```
array([[1. , 2. ],
       [4. , 5.4]])
```

[42]:
```
a4 = np.random.randint(10, size = (2,3,4,5)) # fromm inner to outer
a4.shape
```

[42]:
```
(2, 3, 4, 5)
```

[43]:
```
a4
```

[43]:
```
array([[[[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],
         [8, 8, 2, 3, 2],
         [0, 8, 8, 3, 8]],

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

```
            [0, 8, 5, 9, 0],
            [9, 6, 5, 3, 1]]]], dtype=int32)
```

[44]:
```
#Get the first 4 numbers of the inner most array
```

[45]:
```
a4[:,:,:,:3]
```

[45]:
```
array([[[[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],
         [8, 8, 2],
         [0, 8, 8]],

        [[2, 8, 4],
         [4, 3, 6],
         [0, 8, 5],
         [9, 6, 5]]]], dtype=int32)
```

0.4 4. Manipulating & comparing Arrays and Matrices

0.4.1 Arithmetic

[46]:
```
a1
```

[46]:
```
array([1, 2, 3])
```

[47]:
```
ones
```

[47]:
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

[48]:
```
ones = np.ones(3)
```

[49]:
```
ones
```

[49]:
```
array([1., 1., 1.])
```

[50]:
```
a1 + ones # add together the elements of both arrays at the same position
```

[50]:
```
array([2., 3., 4.])
```

[51]:
```
a1-ones
```

[51]:
```
array([0., 1., 2.])
```

[52]:
```
a1*ones
```

[52]:
```
array([1., 2., 3.])
```

[53]:
```
a2
```

[53]:
```
array([[1. , 2. , 3. ],
       [4. , 5.4, 6.4]])
```

[54]:
```
a1*a2 # only possible when the dim is equal or one of them 1
```

[54]:
```
array([[ 1. ,  4. ,  9. ],
       [ 4. , 10.8, 19.2]])
```

[55]:
```
a1 / ones
```

[55]:
```
array([1., 2., 3.])
```

[56]:
```
ones / a1
```

[56]:
```
array([1.        , 0.5       , 0.33333333])
```

[57]:
```
a2 // a1 # Floor divison removes the decimals
```

[57]:
```
array([[1., 1., 1.],
       [4., 2., 2.]])
```

[58]:
```
a2 ** 2 # pow
```

[58]:
```
array([[ 1.  ,  4.  ,  9.  ],
       [16.  , 29.16, 40.96]])
```

[59]:
```
np.square(a2)
```

[59]:
```
array([[ 1.  ,  4.  ,  9.  ],
       [16.  , 29.16, 40.96]])
```

[60]:
```
np.add(a1,ones)
```

[60]:
```
array([2., 3., 4.])
```

[61]:
```
a1%2
```

[61]:
```
array([1, 0, 1])
```

[62]:
```
np.exp(a1)
```

[62]:
```
array([ 2.71828183,  7.3890561 , 20.08553692])
```

[63]:
```
np.log(a1)
```

[63]:
```
array([0.        , 0.69314718, 1.09861229])
```

0.4.2 Aggregation

Aggregation = performing the same operation on a number of things

[64]:
```
listy_list = [1,2,3]
```

[65]:
```
type(listy_list)
```

[65]:
```
list
```

[66]:
```
sum(listy_list)
```

[66]:
```
6
```

[67]:
```
a1
```

[67]:
```
array([1, 2, 3])
```

[68]:
```
type(a1)
```

[68]:
```
numpy.ndarray
```

[69]:
```
np.sum(a1)
```

[69]:
```
np.int64(6)
```

[70]:
```
sum(a1)
```

[70]:
```
np.int64(6)
```

[71]:
```
# Use Python methods on Python DT and Numpy Methods on Numpy DT
```

[72]:
```
#create a massive np array
```

[73]:
```
massive_array = np.random.random(100000)
massive_array.size
```

[73]:
```
100000
```

[74]:
```
massive_array[:10]
```

[74]:
```
array([0.39486929, 0.61880856, 0.47486752, 0.47013219, 0.71607453,
       0.287991  , 0.38346223, 0.74916984, 0.87845219, 0.10286336])
```

[75]:
```
%timeit sum(massive_array) # Python
%timeit np.sum(massive_array) # NP # Performance is faster
```

```
17.6 ms ± 1.71 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
79.9 µs ± 6.99 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

[76]:
```
np.mean(a2)
```

[76]:
```
np.float64(3.6333333333333333)
```

[77]:
```
np.max(a2)
```

[77]:
```
np.float64(6.4)
```

[78]:
```
np.min(a2)
```

[78]:
```
np.float64(1.0)
```

[79]:
```
np.std(a2) # Deviation from the mean of each element [1,2,3] mean = 2 ....
```

[79]:
```
np.float64(1.8669642619920597)
```

[80]:
```
np.var(a2) # Variance = Quadratic Deviation
# Higher variance = wider range of numbers
# Lower variance = lower range of numbers
```

[80]:
```
np.float64(3.4855555555555564)
```

1 Demo of std and var

[81]:
```
high_var_array = np.array([1,100,200,300,4000,5000])
low_var_array = np.array([2,4,6,8,10])
```

[82]:
```
np.var(high_var_array), np.var(low_var_array)
```

[82]:
```
(np.float64(4296133.472222221), np.float64(8.0))
```

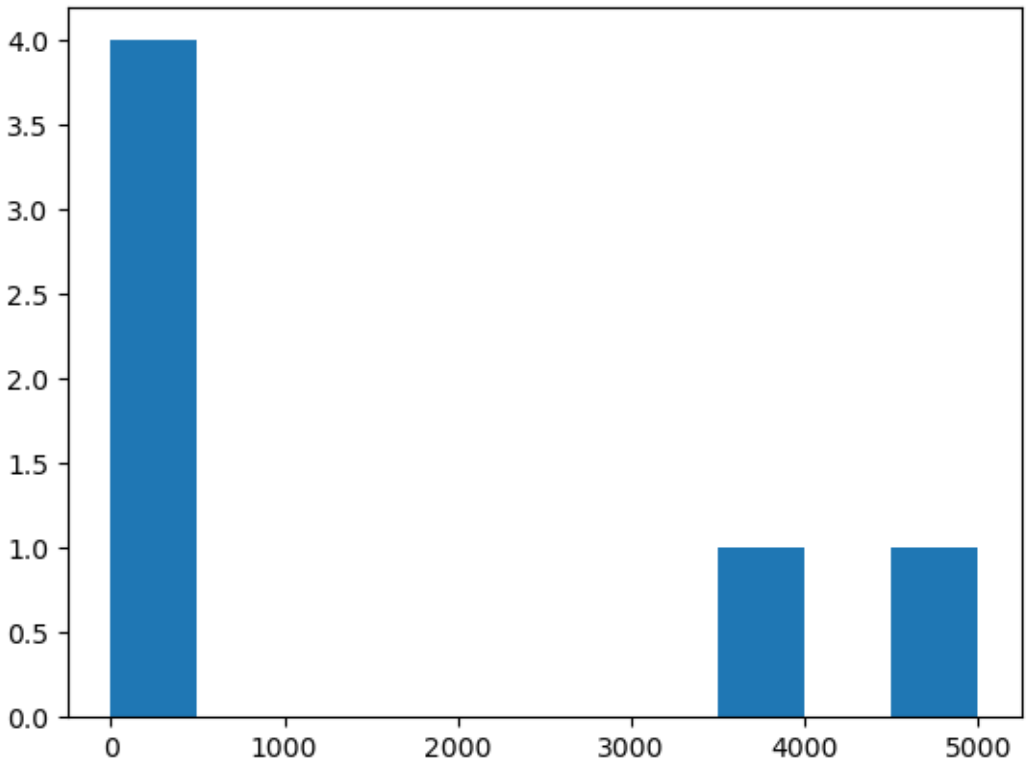
[83]:
```
np.std(high_var_array), np.std(low_var_array)
```

[83]:
```
(np.float64(2072.711623024829), np.float64(2.8284271247461903))
```

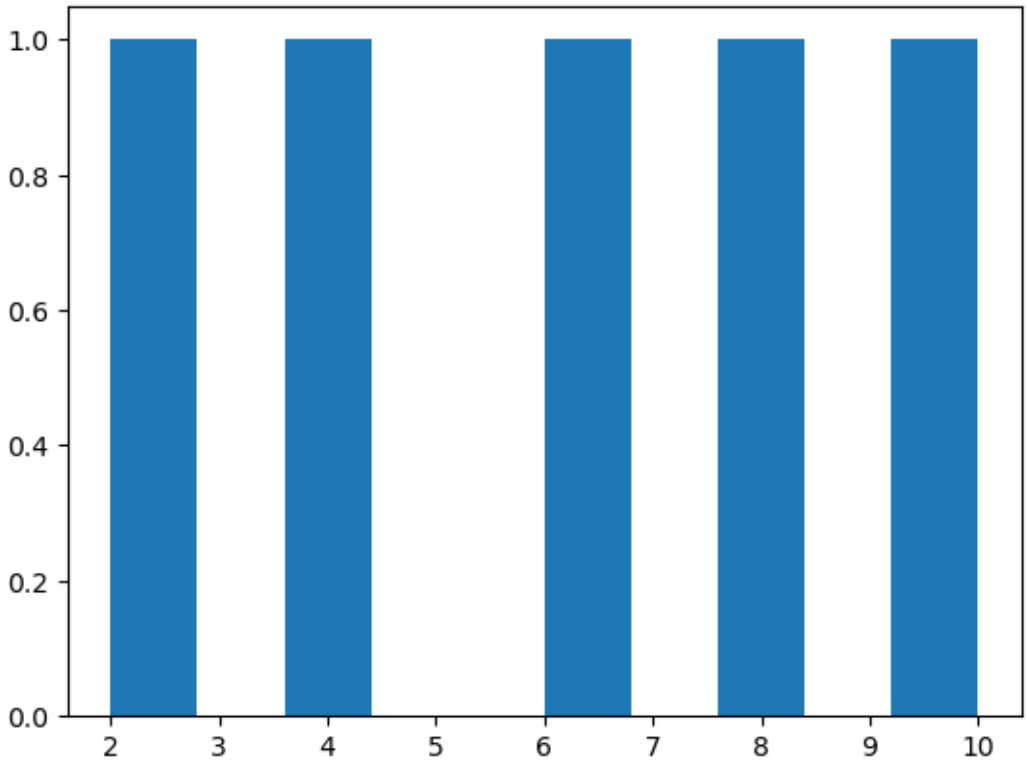
[84]:
```
np.mean(high_var_array), np.mean(low_var_array)
```

[84]:
```
(np.float64(1600.1666666666667), np.float64(6.0))
```

[85]:
```
import matplotlib.pyplot as plt
plt.hist(high_var_array)
plt.show()
```



[86]:
```
plt.hist(low_var_array)
plt.show()
```



1.0.1 Reshaping & Transposing

[87]:
```
a2.shape
```

[87]:
```
(2, 3)
```

[88]:
```
a3.shape
```

[88]:
```
(3, 3)
```

[89]:
```
a2.reshape(2,3,1).shape
```

[89]:
```
(2, 3, 1)
```

[91]:
```
a2_reshape = a2.reshape(2,3,1)
```

[92]:
```
a2_reshape.shape
```

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

[94]:
```
a2_reshape
```

[94]:
```
array([[[1. ],
        [2. ],
        [3. ]],

       [[4. ],
        [5.4],
        [6.4]]])
```

[95]:
```
a2
```

[95]:
```
array([[1. , 2. , 3. ],
       [4. , 5.4, 6.4]])
```

[96]:
```
#Transpose = switches the axis
a2.T
```

[96]:
```
array([[1. , 4. ],
       [2. , 5.4],
       [3. , 6.4]])
```

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

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

1.0.2 DOT PRODUCT

[99]:
```
np.random.seed(0)

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

mat1
```

[99]:
```
array([[5, 0, 3],
       [3, 7, 9],
       [3, 5, 2],
       [4, 7, 6],
       [8, 8, 1]], dtype=int32)
```

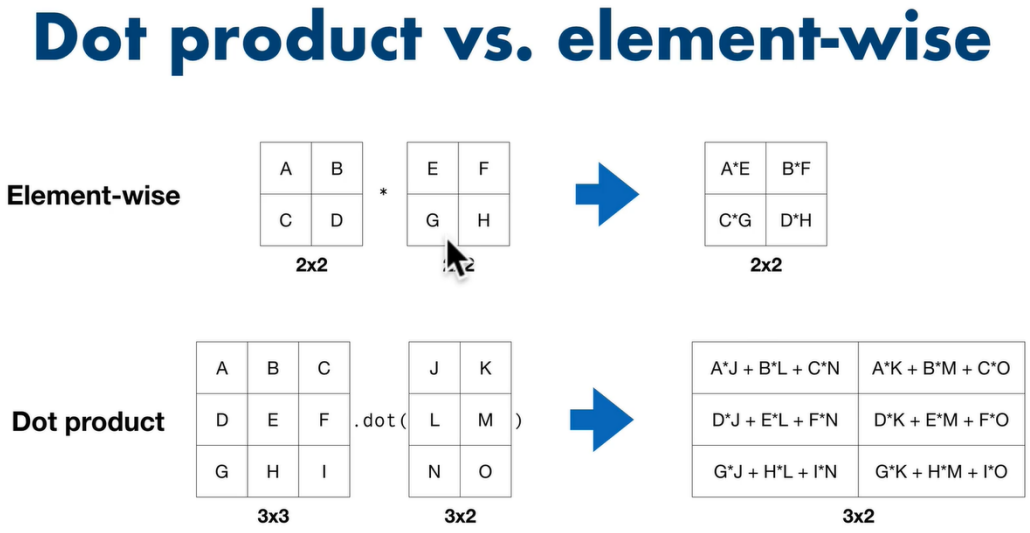
[100]:
```
mat2
```

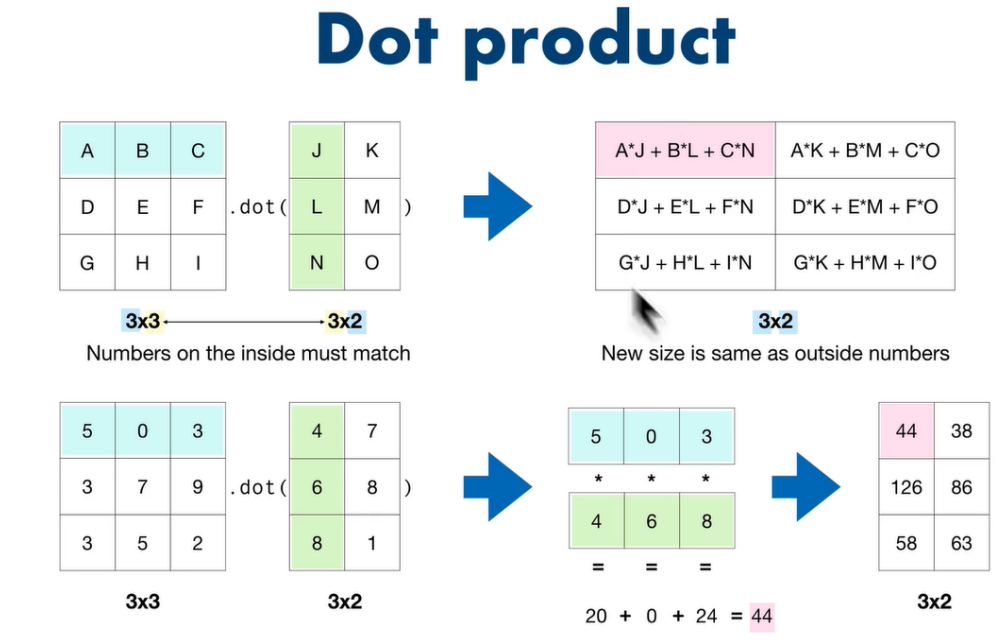
[100]:
```
array([[6, 7, 7],
       [8, 1, 5],
       [9, 8, 9],
       [4, 3, 0],
       [3, 5, 0]], dtype=int32)
```

[101]:
```
mat1 * mat2 #Element Wise multiplication (hadamard product)
```

[101]:
```
array([[30,  0, 21],
       [24,  7, 45],
       [27, 40, 18],
       [16, 21,  0],
       [24, 40,  0]], dtype=int32)
```

[102]:
```
# DOT Product -> Viele Werte × feste Regeln → Ergebnis
```





[104]:
```
mat3 = np.dot(mat1,mat2.T)
```

[105]:
```
mat3
```

[105]:
```
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]], dtype=int32)
```

1.1 Dot Product example (nut butter sales)

[108]:
```
np.random.seed(0)
# Number of jars sold
sales_amounts = np.random.randint(20,size=(5,3))
sales_amounts
```

[108]:
```
array([[12, 15,  0],
       [ 3,  3,  7],
       [ 9, 19, 18],
       [ 4,  6, 12],
       [ 1,  6,  7]], dtype=int32)
```

[110]:
```
# Create weekly_sales DataFrame
weekly_sales = pd.DataFrame(sales_amounts, index =␣
↪["mon","tues","Wed","Thurs","Fri"], columns=["Almond butter","Peanut␣
↪Butter","Cashew Butter"])
weekly_sales
```

[110]:

	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

[111]:
```
# Create the prices Array
prices = np.array([10,8,12])
prices
```

[111]:
```
array([10,  8, 12])
```

[113]:
```
# Create Prices DF
butter_prices = pd.DataFrame(prices.reshape(1,3), index =["Prices"], columns␣
↪=["Almond butter","Peanut Butter","Cashew Butter"])
butter_prices
```

[113]:

	Almond butter	Peanut Butter	Cashew Butter
Prices	10	8	12

[114]:
```
total_sales = prices.dot(sales_amounts)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[114], line 1
----> 1 total_sales = prices.dot(sales_amounts)

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

[115]:
```
# Shapes arent aligned lets transpose
```

[116]:
```
total_sales = prices.dot(sales_amounts.T)
total_sales
```

[116]:
```
array([240, 138, 458, 232, 142])
```

[117]:
```
# Create daily_sales
daily_sales = butter_prices.dot(weekly_sales.T)
daily_sales
```

[117]:

	mon	tues	Wed	Thurs	Fri
Prices	240	138	458	232	142

[119]:
```
weekly_sales["Total"] = daily_sales.T
weekly_sales
```

[119]:

	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

1.1.1 Comparison Operators

[120]:
```
a1
```

[120]:
```
array([1, 2, 3])
```

[121]:
```
a2
```

[121]:
```
array([[1. , 2. , 3. ],
       [4. , 5.4, 6.4]])
```

[122]:
```
a1>a2
```

[122]:
```
array([[False, False, False],
       [False, False, False]])
```

[123]:
```
bool_array = a1>=a2
bool_array
```

[123]:
```
array([[ True,  True,  True],
       [False, False, False]])
```

[124]:
```
#return always a bool Array
```

1.2 5. Sorting Arrays

[125]:
```
random_array
```

[125]:
```
array([[5, 9, 0, 9, 0],
       [5, 8, 9, 2, 9],
       [9, 9, 6, 0, 4]], dtype=int32)
```

[126]:
```
random_array = np.random.randint(10,size=(3,5))
```

[127]:
```
random_array
```

[127]:
```
array([[7, 8, 1, 5, 9],
       [8, 9, 4, 3, 0],
       [3, 5, 0, 2, 3]], dtype=int32)
```

[130]:
```
random_array.shape
```

[130]:
```
(3, 5)
```

[131]:
```
np.sort(random_array)
```

[131]:
```
array([[1, 5, 7, 8, 9],
       [0, 3, 4, 8, 9],
       [0, 2, 3, 3, 5]], dtype=int32)
```

[132]:
```
np.argsort(random_array) #-> which index has a high Value
```

[132]:
```
array([[2, 3, 0, 1, 4],
       [4, 3, 2, 0, 1],
       [2, 3, 0, 4, 1]])
```

[133]:
```
a1
```

[133]:
```
array([1, 2, 3])
```

[134]:
```
np.argsort(a1)
```

[134]:
```
array([0, 1, 2])
```

[135]:
```
np.argmin(a1) # minimum Value at which index
```

[135]:
```
np.int64(0)
```

[136]:
```
np.argmax(a1)
```

[136]:
```
np.int64(2)
```

[139]:
```
np.argmax(random_array, axis=1) # axis 1 is vertically and 0 ist horizontally
```

[139]:
```
array([4, 1, 1])
```

[138]:
```
random_array
```

[138]:
```
array([[7, 8, 1, 5, 9],
       [8, 9, 4, 3, 0],
       [3, 5, 0, 2, 3]], dtype=int32)
```

1.3 6.Practical Example

[142]:
```
# Turn an image into a NumPy Array
from matplotlib.image import imread

panda = imread("./panda.png")
type(panda)
```

[142]:
```
numpy.ndarray
```

[145]:
```
panda.size, panda.shape, panda.ndim
```

[145]:
```
(24465000, (2330, 3500, 3), 3)
```

[146]:
```
panda[:5]
```

[146]:
```
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]]],
shape=(5, 3500, 3), dtype=float32)
```

[148]:
```
car = imread("./car-photo.png")
```

[149]:
```
car.shape, car.size, car.ndim
```

[149]:
```
((431, 575, 4), 991300, 3)
```

[ ]:
```

```