Numpy

- It consist of tricks mostly used

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

In [2]: sales = [4543,2233,4335,3344,4543,6544,3565,3344,6543,6644,8765,3323,5654,3344,5654,7766]
        profit = [454,233,443,566,777,677,555,766,455,554,665,455,345,766,444,567]

In [3]: #to check the length
        len(sales)
        len(profit)

Out[3]: 16

In [4]: #we cannot divide list/list because sales and profit are in list
        #to divide this list we have to convert it into 'numpy array'

        profit/sales

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-49ed75946523> in <module>
      2 #to divide this list we have to convert it into 'numpy array'
      3
----> 4 profit/sales

TypeError: unsupported operand type(s) for /: 'list' and 'list'

In [5]: # to make sales array
        sales_array = np.array(sales)

In [6]: type(sales_array)

Out[6]: numpy.ndarray

In [12]: #to make profit array
         profit_array = np.array(profit)

In [13]: type(profit_array)

Out[13]: numpy.ndarray

In [14]: #to make sales array
         sales_array

Out[14]: array([4543, 2233, 4335, 3344, 4543, 6544, 3565, 3344, 6543, 6644, 8765,
                3323, 5654, 3344, 5654, 7766])

In [15]: profit_array

Out[15]: array([454, 233, 443, 566, 777, 677, 555, 766, 455, 554, 665, 455, 345,
                766, 444, 567])

In [16]: #now we can divid
         profit_array/sales_array

Out[16]: array([0.09993396, 0.10434393, 0.10219146, 0.16925837, 0.17103236,
                0.10345355, 0.15568022, 0.22906699, 0.06953997, 0.0833835 ,
                0.07586994, 0.13692447, 0.06101875, 0.22906699, 0.07852848,
                0.07301056])

In [17]: list1 = [10, 'A', 10.5, True]

In [18]: type(list1[2])

Out[18]: float

In [19]: array1 = np.array(list1)

In [20]: # in an array where string is present string takes priority
         type(array1[2])

Out[20]: numpy.str_

In [21]: list2 = [10, 10.5, True]

In [22]: array2 = np.array(list2)

In [23]: array2

Out[23]: array([10. , 10.5,  1. ])

In [24]: # in this float become priority --> (10.5)
         type(array2[2])

Out[24]: numpy.float64

In [25]: list3 = [10, True]

In [26]: array3 = np.array(list3)

In [27]: array3

Out[27]: array([10,  1])

In [28]: #integer is priority
         #if integer in not present then bool is the priority
         type(array3[0])

Out[28]: numpy.int32

In [2]: list1 = [32,55,99,69,95]

In [3]: #if we multiply by list*2 the same number will be doubled we can see below
        list1*2

Out[3]: [32, 55, 99, 69, 95, 32, 55, 99, 69, 95]

In [4]: array1 = np.array(list1)

In [5]: #we can multiply the array for Ex:- 32*2 = 64
        array1*2

Out[5]: array([ 64, 110, 198, 138, 190])

In [6]: #this is the copy of list1
        list2 = list1.copy()

In [7]: list2

Out[7]: [32, 55, 99, 69, 95]

In [8]: list3 = list1[0:3]

In [9]: list3

Out[9]: [32, 55, 99]

In [35]: #parent list
         list1

Out[35]: [32, 55, 99, 69, 95]

In [36]: list2[0] = 200

In [37]: list2

Out[37]: [200, 55, 99, 69, 95]

In [10]: #parent list is not changed by copied list2
         list1

Out[10]: [32, 55, 99, 69, 95]

In [11]: list3[0] = 100

In [12]: #it will not update parent list
         list3

Out[12]: [100, 55, 99]

In [13]: list1

Out[13]: [32, 55, 99, 69, 95]

In [40]: #converting list1 to array
         a1 = np.array(list1)

In [41]: a1

Out[41]: array([32, 55, 99, 69, 95])

In [42]: a2 = a1.copy()

In [43]: a2

Out[43]: array([32, 55, 99, 69, 95])

In [44]: #sliced list
         a3 = a1[0:3]

In [45]: a3

Out[45]: array([32, 55, 99])

In [46]: a1

Out[46]: array([32, 55, 99, 69, 95])

In [47]: a2

Out[47]: array([32, 55, 99, 69, 95])

In [48]: a3[0]=1000

In [49]: #array has updated parent also --> [0]
         a1

Out[49]: array([1000,   55,   99,   69,   95])

In [50]: #coppied list is not changing
         a2

Out[50]: array([32, 55, 99, 69, 95])

In [51]: a3

Out[51]: array([1000,   55,   99])

In [52]: #when we make change in coppied list a2
         a2[0]=5000

In [53]: #a1 is not changing
         a1

Out[53]: array([1000,   55,   99,   69,   95])

In [54]: #a3 is not changing
         a3

Out[54]: array([1000,   55,   99])

In [55]: a1.dtype

Out[55]: dtype('int32')

In [56]: #one diamension array
         a1.ndim

Out[56]: 1

In [57]: a1.shape

Out[57]: (5,)

In [58]: #[0] --> Scalar
         #[236,258,369] --> vector
         '''[[365,235,699,362]
           [123,369,258,963]] --> 2D vector'''

         sales_us = [2659,6266,3685,2235]
         sales_eu = [3265,8955,6977,5721]
         sales_jp = [2365,5688,3255,2365]

In [59]: profit_us = [365,235,699,362]
         profit_eu = [123,369,258,963]
         profit_jp = [213,365,796,653]

In [60]: sales_matrix = np.array([sales_us,sales_eu,sales_jp])

In [61]: #two diamension array or matrix
         sales_matrix

Out[61]: array([[2659, 6266, 3685, 2235],
                [3265, 8955, 6977, 5721],
                [2365, 5688, 3255, 2365]])

In [62]: sales_matrix.dtype

Out[62]: dtype('int32')

In [63]: sales_matrix.shape

Out[63]: (3, 4)

In [64]: sales_matrix.ndim

Out[64]: 2

In [65]: type(sales_matrix)

Out[65]: numpy.ndarray

In [66]: profit_matrix = np.array([profit_us,profit_eu,profit_jp])

In [67]: profit_matrix

Out[67]: array([[365, 235, 699, 362],
                [123, 369, 258, 963],
                [213, 365, 796, 653]])

In [68]: profit_matrix/sales_matrix

Out[68]: array([[0.13726965, 0.03750399, 0.18968792, 0.16196868],
                [0.03767228, 0.04120603, 0.03697864, 0.16832722],
                [0.09006342, 0.06417018, 0.24454685, 0.27610994]])

In [69]: #in this the profit_jp has missing one value because the values are inconsistance
         profit_us_A = [365,235,699,362]
         profit_eu_B = [123,369,258,963]
         profit_jp_C = [213,365,796]

In [70]: profit_matrix_1 = np.array([profit_us_A,profit_eu_B,profit_jp_C])

<ipython-input-70-f2dadf038aa6>:1: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with differe
nt lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
  profit_matrix_1 = np.array([profit_us_A,profit_eu_B,profit_jp_C])

In [71]: type(profit_matrix_1)
         profit_matrix_1.ndim

Out[71]: 1

In [72]: profit_matrix

Out[72]: array([[365, 235, 699, 362],
                [123, 369, 258, 963],
                [213, 365, 796, 653]])

In [73]: #index start from 0 and we want "258" by slicing
         #slicing
         profit_matrix[1,2]

Out[73]: 258

In [74]: profit_matrix[2,3]

Out[74]: 653

In [75]: profit_matrix[0,0]

Out[75]: 365

In [76]: profit_matrix[0:2, 2:5]

Out[76]: array([[699, 362],
                [258, 963]])

In [77]: profit_matrix[:, 2:5]

Out[77]: array([[699, 362],
                [258, 963],
                [796, 653]])

In [78]: profit_matrix

Out[78]: array([[365, 235, 699, 362],
                [123, 369, 258, 963],
                [213, 365, 796, 653]])

In [79]: profit_matrix[0,2]

Out[79]: 699

In [80]: profit_matrix[0][2]

Out[80]: 699

In [96]: profit_matrix[rdict['US']][qdict['Q3']]

Out[96]: 699

In [92]: rdict = {'US':0, 'EU':1, 'JP':2}
         qdict = {'Q1':0, 'Q2':1, 'Q3':2, 'Q4':3}

In [93]: rdict['US']

Out[93]: 0

In [94]: qdict['Q3']

Out[94]: 2

In [95]: rdict['US'],qdict['Q3']

Out[95]: (0, 2)

In [86]: a = np.arange(1,21)

In [87]: a

Out[87]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
                18, 19, 20])

In [88]: #'C' means inherit from C-Programming
         np.reshape(a, (10,2), 'C')

Out[88]: array([[ 1,  2],
                [ 3,  4],
                [ 5,  6],
                [ 7,  8],
                [ 9, 10],
                [11, 12],
                [13, 14],
                [15, 16],
                [17, 18],
                [19, 20]])

In [89]: #'F' means fortran-programming
         #in fortran programming language the values are populated column by column
         np.reshape(a, (10,2), 'F')

Out[89]: array([[ 1, 11],
                [ 2, 12],
                [ 3, 13],
                [ 4, 14],
                [ 5, 15],
                [ 6, 16],
                [ 7, 17],
                [ 8, 18],
                [ 9, 19],
                [10, 20]])

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```