What is Numpy.

NumPy refers to Numerical Python and it is an open source library in Python that aids in mathematical ,numerical calculations and computations , scientific, engineering, and data science programming.

NumPy is an most essential library to perform mathematical and statistical operations. It works perfectly well for multi-dimensional arrays and matrices multiplication.

Numpy is memory efficiency, meaning it can handle the vast amount of data more accessible than any other library. Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping and more helpful in machine learning.

Numpy makes the work simpler by doing the complex numerical computations in background.

What is an Array.

An array is a special variable, which can hold more than one value at a time.

Sequences of elements.

In [1]:
```
# Importing numpy librry in python
import numpy as np
```

In [2]:
```
list1 = [111,222,333]
ar = np.array(list1)
```

In [3]:
```
print(ar)
```
```
[111 222 333]
```

In [4]:
```
type(ar)
```
Out[4]:
```
numpy.ndarray
```

Note : Arrays can store elements of the same `type`.

In [5]:
```
ar.shape     # 1- Dim Array
```
Out[5]:
```
(3,)
```

In [6]:
```
ar.itemsize   # Each item of array takes 4-bytes
```
Out[6]:
```
4
```

In [7]:
```
ar.ndim # To check array dimension
```
Out[7]:
```
1
```

In [8]:
```
ar.size     #no. of elements in array
```
Out[8]:
```
3
```

In [9]:
```
ar.itemsize*ar.size # Total memory occupied in the system in this array
```
Out[9]:
```
12
```

In [10]:
```
ar
```
Out[10]:
```
array([111, 222, 333])
```

In [11]:
```
print(ar[1])
```
```
222
```

In [12]:
```
list1=[2,5,10]
list2=[3,6,4]
ar1=np.array(list1)
ar2=np.array(list2)
ar3=ar1+ar2                 # arithmetic addition
```

In [13]:
```
print(ar3)
```
```
[ 5 11 14]
```

In [14]:
```
ar4=ar1*ar2
print(ar4)
```
```
[ 6 30 40]
```

In [15]:
```
ar4.ndim # 1-D array
```
Out[15]:
```
1
```

Two dimensional arrays

In [16]:
```
list1=[[111,222,333],[444,555,666],[777,888,999]] # Creating the list of list
print(list1)
```
```
[[111, 222, 333], [444, 555, 666], [777, 888, 999]]
```

In [17]:
```
len(list1)
```
Out[17]:
```
3
```

In [18]:
```
list_arry = np.array(list1)
```

In [19]:
```
list_arry
```
Out[19]:
```
array([[111, 222, 333],
       [444, 555, 666],
       [777, 888, 999]])
```

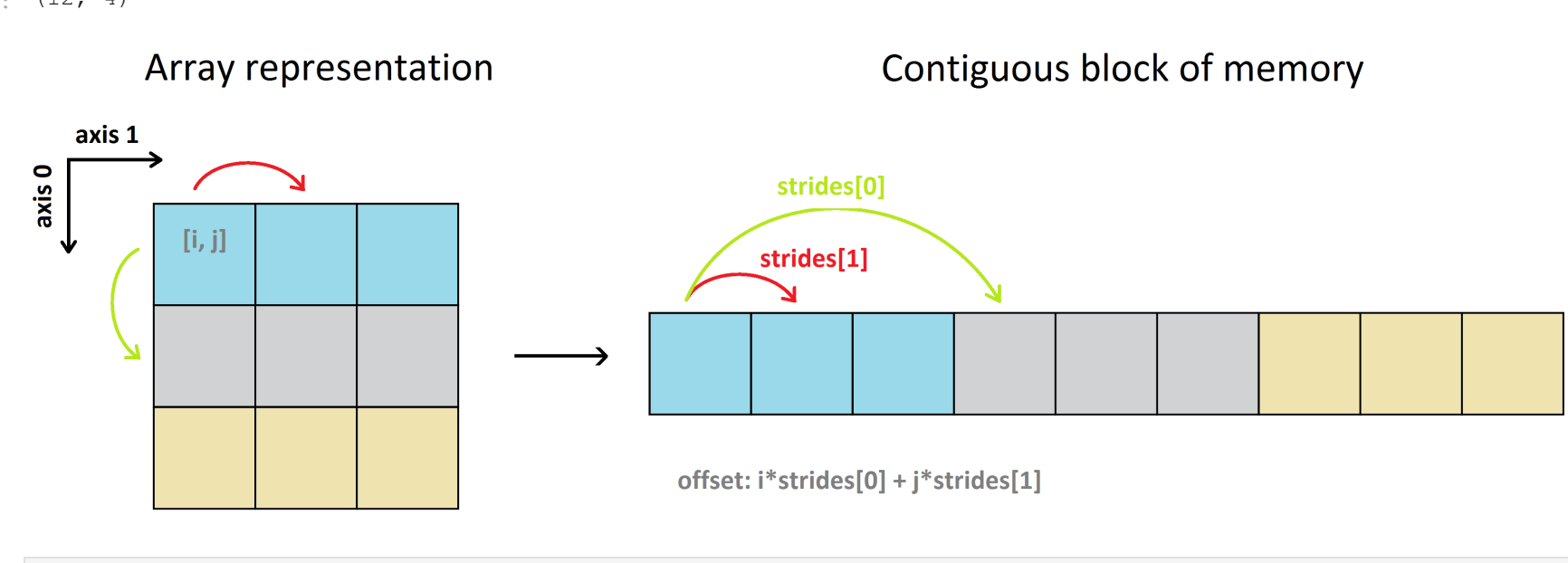
In [20]:
```
print(" Output Array is: ", list_arry)
```
```
 Output Array is:  [[111 222 333]
 [444 555 666]
 [777 888 999]]
```

In [21]:
```
list_arry.shape  # 3 rows and 3 column
```
Out[21]:
```
(3, 3)
```

In [22]:
```
list_arry.itemsize      #bytes taken by the element
```
Out[22]:
```
4
```

In [23]:
```
list_arry.size         #total no. of elements
```
Out[23]:
```
9
```

In [24]:
```
list_arry.strides # How many bytes occupied by the prog to move from 1 row to another row
```
Out[24]:
```
(12, 4)
```



In [25]:
```
list_arry.ndim        #dimensions of array
```
Out[25]:
```
2
```

In [26]:
```
len(list_arry)
```
Out[26]:
```
3
```

In [27]:
```
help(print)
```
```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Array Indexing

    Slicing the array for doing data impuations, filtering and tranformations.

In [28]:
```
# Slicing
arry = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(arry)
```
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [29]:
```
print(arry[0,0]) # This would print the element from 1st row and 1st column
print(arry[1,1])  # This would print the element from 2nd row and 2nd column
```
```
1
5
```

In [30]:
```
print(arry[:,0:2]) # Print all the row elements from col 1 and col 2
```
```
[[1 2]
 [4 5]
 [7 8]]
```

In [31]:
```
print(arry[0:2,2])
# [0:2,2] would translate to [(0,1),2] i.e to print the row 1 and row 2 elements of column 2 which is 3,6
```
```
[3 6]
```

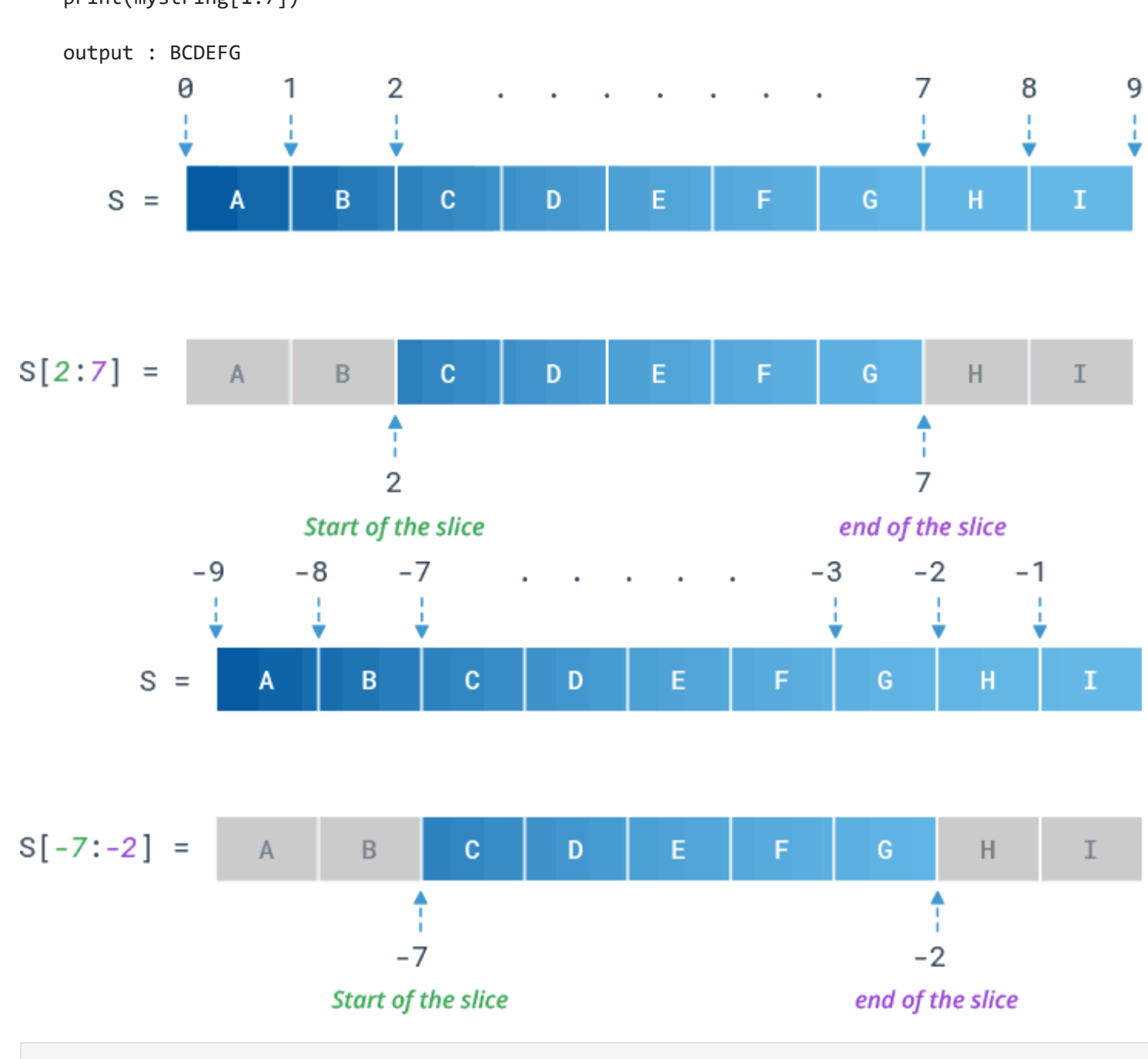
In [32]:
```
z=arry[[1,2,0],[0,1,1]]       # this is not slicing but simple co ordinates for elements and read as that are (
```

In [33]:
```
z
```
Out[33]:
```
array([4, 8, 2])
```

Accessing values of String through Slicing

mystring[N:M:STEP] # starting point=N, end point=M(excluding M) ,Step =+1 by default

```
 eg:

mystring="ABCDEFGHI"
print(mystring[1:7])

output : BCDEFG
```



In [34]:
```
mystr = "ABCDEFGHI"
```

In [35]:
```
print(mystr[1:7])
```
```
BCDEFG
```

In [36]:
```
print(mystr[-8:-2])
```
```
BCDEFG
```

In [37]:
```
print(mystr[4:-2])
```
```
EFG
```

In [42]:
```
print(mystr[-1:4:-1])
```
```
IHGF
```

In [43]:
```
thred_arry = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12],[13,14,15],[16,17,18]])
```

In [44]:
```
thred_arry
```
Out[44]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

In [45]:
```
thred_arry.shape
```
Out[45]:
```
(6, 3)
```

In [46]:
```
thred_arry.ndim
```
Out[46]:
```
2
```

In [47]:
```
reshp_arr = thred_arry.reshape(3,6)
```

In [48]:
```
reshp_arr
```
Out[48]:
```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12],
       [13, 14, 15, 16, 17, 18]])
```

In [49]:
```
reshp_arr.shape
```
Out[49]:
```
(3, 6)
```

In [54]:
```
ar8 = thred_arry.reshape(9,2)
print(ar8)
```
```
[[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]
 [11 12]
 [13 14]
 [15 16]
 [17 18]]
```

In [55]:
```
ar8.shape
```
Out[55]:
```
(9, 2)
```

In [56]:
```
trp_arr = np.transpose(thred_arry)          #transpose function
```

In [57]:
```
trp_arr
```
Out[57]:
```
array([[ 1,  4,  7, 10, 13, 16],
       [ 2,  5,  8, 11, 14, 17],
       [ 3,  6,  9, 12, 15, 18]])
```

In [58]:
```
trp_arr.shape
```
Out[58]:
```
(3, 6)
```

In [ ]:
```

```

In [59]:
```
thred_arry
```
Out[59]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

3- D Array

In [60]:
```
three_D_Arry = thred_arry.reshape(3,2,3) ## Create 3 arrays of 2 rows and 3 cols
```

In [61]:
```
three_D_Arry
```
Out[61]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]],

       [[13, 14, 15],
        [16, 17, 18]]])
```

In [62]:
```
three_D_Arry.ndim
```
Out[62]:
```
3
```

In [63]:
```
three_D_Arry.size
```
Out[63]:
```
18
```

In [64]:
```
three_D_Arry.shape     # 3 = depth, 2 = rws, 3 = cols
```
Out[64]:
```
(3, 2, 3)
```

In [65]:
```
# Slice first col of each row from the 3 d array
three_D_Arry[0:3, 0:3, 0:1]
```
Out[65]:
```
array([[[ 1],
        [ 4]],

       [[ 7],
        [10]],

       [[13],
        [16]]])
```

In [66]:
```
list1=[
    [
        [1,2],[3,4],[5,6],[7,8],[9,10]
    ],
    [
        [11,12],[13,14],[15,16],[17,18],[19,20]
    ],
    [
        [21,22],[23,24],[25,26],[27,28],[29,30]
                                                                  ]
]
```

In [67]:
```
list1
```
Out[67]:
```
[[[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]],
 [[11, 12], [13, 14], [15, 16], [17, 18], [19, 20]],
 [[21, 22], [23, 24], [25, 26], [27, 28], [29, 30]]]
```

In [68]:
```
ar11=np.array(list1)
```

In [69]:
```
ar11
```
Out[69]:
```
array([[[ 1,  2],
        [ 3,  4],
        [ 5,  6],
        [ 7,  8],
        [ 9, 10]],

       [[11, 12],
        [13, 14],
        [15, 16],
        [17, 18],
        [19, 20]],

       [[21, 22],
        [23, 24],
        [25, 26],
        [27, 28],
        [29, 30]]])
```

In [70]:
```
ar11.ndim
```
Out[70]:
```
3
```

In [71]:
```
ar11.shape
```
Out[71]:
```
(3, 5, 2)
```

In [72]:
```
print(ar11[1,2,1])
```
```
16
```

In [73]:
```
list2=[
    [
        [111,112,113],[121,122,123],[131,132,133],[141,142,143]
    ],
    [
        [211,212,213],[221,222,223],[231,232,233],[241,242,243]
    ],
    [
        [311,312,313],[321,322,323],[331,332,333],[341,342,343]
    ]
    ,[
        [411,412,413],[421,422,423],[431,432,433],[441,442,443]
    ],
    [
        [511,512,513],[521,522,523],[531,532,533],[541,542,543]
    ]
]
```

In [74]:
```
list2
```
Out[74]:
```
[[[111, 112, 113], [121, 122, 123], [131, 132, 133], [141, 142, 143]],
 [[211, 212, 213], [221, 222, 223], [231, 232, 233], [241, 242, 243]],
 [[311, 312, 313], [321, 322, 323], [331, 332, 333], [341, 342, 343]],
 [[411, 412, 413], [421, 422, 423], [431, 432, 433], [441, 442, 443]],
 [[511, 512, 513], [521, 522, 523], [531, 532, 533], [541, 542, 543]]]
```

In [75]:
```
arr12=np.array(list2)
```

In [76]:
```
arr12
```
Out[76]:
```
array([[[111, 112, 113],
        [121, 122, 123],
        [131, 132, 133],
        [141, 142, 143]],

       [[211, 212, 213],
        [221, 222, 223],
        [231, 232, 233],
        [241, 242, 243]],

       [[311, 312, 313],
        [321, 322, 323],
        [331, 332, 333],
        [341, 342, 343]],

       [[411, 412, 413],
        [421, 422, 423],
        [431, 432, 433],
        [441, 442, 443]],

       [[511, 512, 513],
        [521, 522, 523],
        [531, 532, 533],
        [541, 542, 543]]])
```

In [77]:
```
arr12.shape
```
Out[77]:
```
(5, 4, 3)
```

In [78]:
```
len(arr12)
```
Out[78]:
```
5
```

In [79]:
```
for d in range(len(arr12)):
    for r in range(len(arr12[0])):
        for c in range(len(arr12[0][0])):
            print("depth, row, col : ",d,r,c,":", arr12[d][r][c])
```
```
depth, row, col :  0 0 0 : 111
depth, row, col :  0 0 1 : 112
depth, row, col :  0 0 2 : 113
depth, row, col :  0 1 0 : 121
depth, row, col :  0 1 1 : 122
depth, row, col :  0 1 2 : 123
depth, row, col :  0 2 0 : 131
depth, row, col :  0 2 1 : 132
depth, row, col :  0 2 2 : 133
depth, row, col :  0 3 0 : 141
depth, row, col :  0 3 1 : 142
depth, row, col :  0 3 2 : 143
depth, row, col :  1 0 0 : 211
depth, row, col :  1 0 1 : 212
depth, row, col :  1 0 2 : 213
depth, row, col :  1 1 0 : 221
depth, row, col :  1 1 1 : 222
depth, row, col :  1 1 2 : 223
depth, row, col :  1 2 0 : 231
depth, row, col :  1 2 1 : 232
depth, row, col :  1 2 2 : 233
depth, row, col :  1 3 0 : 241
depth, row, col :  1 3 1 : 242
depth, row, col :  1 3 2 : 243
depth, row, col :  2 0 0 : 311
depth, row, col :  2 0 1 : 312
depth, row, col :  2 0 2 : 313
depth, row, col :  2 1 0 : 321
depth, row, col :  2 1 1 : 322
depth, row, col :  2 1 2 : 323
depth, row, col :  2 2 0 : 331
depth, row, col :  2 2 1 : 332
depth, row, col :  2 2 2 : 333
depth, row, col :  2 3 0 : 341
depth, row, col :  2 3 1 : 342
depth, row, col :  2 3 2 : 343
depth, row, col :  3 0 0 : 411
depth, row, col :  3 0 1 : 412
depth, row, col :  3 0 2 : 413
depth, row, col :  3 1 0 : 421
depth, row, col :  3 1 1 : 422
depth, row, col :  3 1 2 : 423
depth, row, col :  3 2 0 : 431
depth, row, col :  3 2 1 : 432
depth, row, col :  3 2 2 : 433
depth, row, col :  3 3 0 : 441
depth, row, col :  3 3 1 : 442
depth, row, col :  3 3 2 : 443
depth, row, col :  4 0 0 : 511
depth, row, col :  4 0 1 : 512
depth, row, col :  4 0 2 : 513
depth, row, col :  4 1 0 : 521
depth, row, col :  4 1 1 : 522
depth, row, col :  4 1 2 : 523
depth, row, col :  4 2 0 : 531
depth, row, col :  4 2 1 : 532
depth, row, col :  4 2 2 : 533
depth, row, col :  4 3 0 : 541
depth, row, col :  4 3 1 : 542
depth, row, col :  4 3 2 : 543
```

In [ ]:
```
### Converting/Storing arrays into pandas dataframe of 2-D shape
```

In [80]:
```
import pandas as pd
```

In [81]:
```
arr13 = arr12.reshape(12,5)
```

In [82]:
```
arr13
```
Out[82]:
```
array([[111, 112, 113, 121, 122],
       [123, 131, 132, 133, 141],
       [142, 143, 211, 212, 213],
       [221, 222, 223, 231, 232],
       [233, 241, 242, 243, 311],
       [312, 313, 321, 322, 323],
       [331, 332, 333, 341, 342],
       [343, 411, 412, 413, 421],
       [422, 423, 431, 432, 433],
       [441, 442, 443, 511, 512],
       [513, 521, 522, 523, 531],
       [532, 533, 541, 542, 543]])
```

In [83]:
```
arr13.ndim
```
Out[83]:
```
2
```

In [84]:
```
arr13.shape
```
Out[84]:
```
(12, 5)
```

In [85]:
```
arr13.size
```
Out[85]:
```
60
```

In [86]:
```
df_arry = pd.DataFrame(arr13)
```

In [87]:
```
df_arry
```
Out[87]:

	0	1	2	3	4
0	111	112	113	121	122
1	123	131	132	133	141
2	142	143	211	212	213
3	221	222	223	231	232
4	233	241	242	243	311
5	312	313	321	322	323
6	331	332	333	341	342
7	343	411	412	413	421
8	422	423	431	432	433
9	441	442	443	511	512
10	513	521	522	523	531
11	532	533	541	542	543

Broadcasting in Arrays

In [88]:
```
a = np.arange(0,24,dtype=np.int32).reshape(6,4) # arange will take a series of nos
```

In [89]:
```
a
```
Out[89]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

In [90]:
```
a+2      #broadcasting
```
Out[90]:
```
array([[ 2,  3,  4,  5],
       [ 6,  7,  8,  9],
       [10, 11, 12, 13],
       [14, 15, 16, 17],
       [18, 19, 20, 21],
       [22, 23, 24, 25]])
```

In [91]:
```
a=np.arange(1,25).reshape(6,2,2)
```

In [92]:
```
a
```
Out[92]:
```
array([[[ 1,  2],
        [ 3,  4]],

       [[ 5,  6],
        [ 7,  8]],

       [[ 9, 10],
        [11, 12]],

       [[13, 14],
        [15, 16]],

       [[17, 18],
        [19, 20]],

       [[21, 22],
        [23, 24]]])
```

In [93]:
```
a=a+[1,2] # Add 1,2 to each rows
```

In [94]:
```
a
```
Out[94]:
```
array([[[ 2,  4],
        [ 4,  6]],

       [[ 6,  8],
        [ 8, 10]],

       [[10, 12],
        [12, 14]],

       [[14, 16],
        [16, 18]],

       [[18, 20],
        [20, 22]],

       [[22, 24],
        [24, 26]]])
```

In [95]:
```
a=np.arange(1,25).reshape(6,4)
```

In [96]:
```
a
```
Out[96]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24]])
```

In [97]:
```
a+[2,3,4,5] #Broadcast 2,3,4,5 into each row of a array
```
Out[97]:
```
array([[ 3,  5,  7,  9],
       [ 7,  9, 11, 13],
       [11, 13, 15, 17],
       [15, 17, 19, 21],
       [19, 21, 23, 25],
       [23, 25, 27, 29]])
```

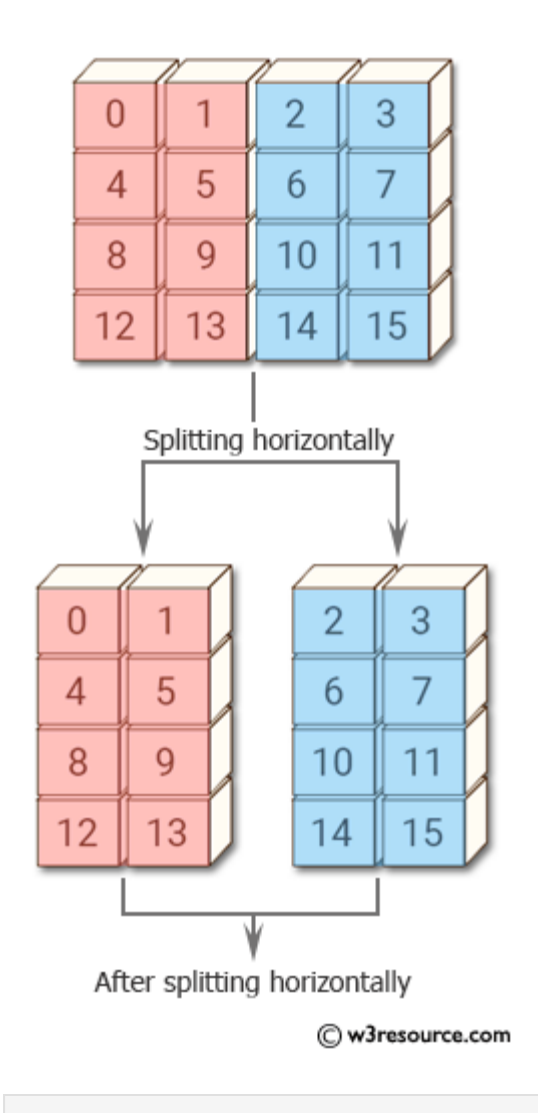
In [98]:
```
a[1,1]+[1,9,10] #Add 1 and 9 to the index value of a[1,1]
```
Out[98]:
```
array([ 7, 15, 16])
```

In [99]:
```
a
```
Out[99]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24]])
```

In [100…
```
a[1,2]+[1,1]
```
Out[100…
```
array([8, 8])
```

Lets see how can we `split` a large arrays into smaller arrays

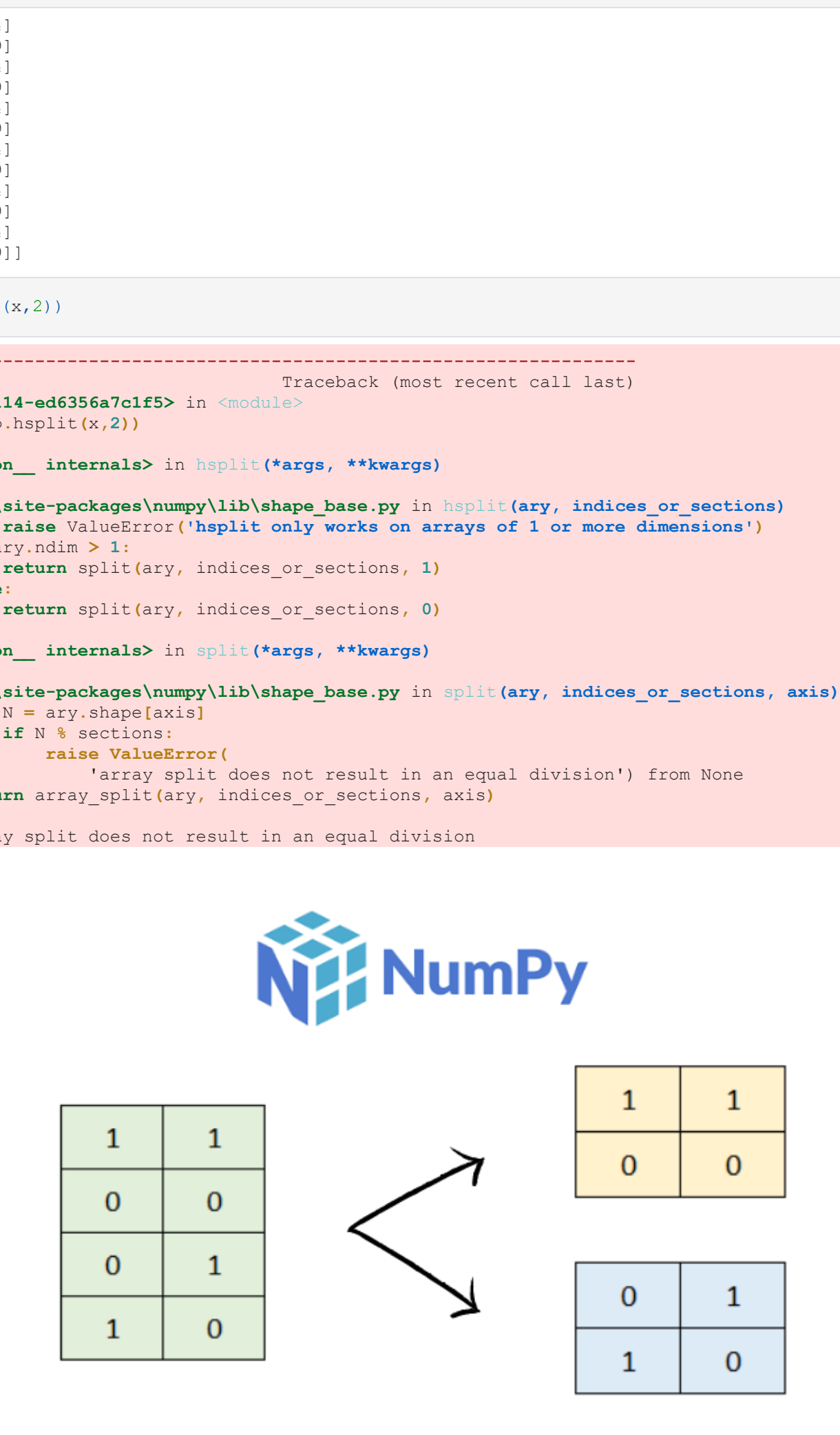
In [101…
```
biga = np.arange(16).reshape(4,4)
print(biga)
biga.shape
```
```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```
Out[101…
```
(4, 4)
```



In [102…
```
sma = np.hsplit(biga,2)  #  sma is a list consisting of 2 smaller arrays  #numpy.hsplit(ary, indices_or_sectio
sma
```
Out[102…
```
[array([[ 0,  1],
        [ 4,  5],
        [ 8,  9],
        [12, 13]]),
 array([[ 2,  3],
        [ 6,  7],
        [10, 11],
        [14, 15]])]
```

In [103…
```
arr13.shape
```
Out[103…
```
(12, 5)
```

In [104…
```
arr13
```
Out[104…
```
array([[111, 112, 113, 121, 122],
       [123, 131, 132, 133, 141],
       [142, 143, 211, 212, 213],
       [221, 222, 223, 231, 232],
       [233, 241, 242, 243, 311],
       [312, 313, 321, 322, 323],
       [331, 332, 333, 341, 342],
       [343, 411, 412, 413, 421],
       [422, 423, 431, 432, 433],
       [441, 442, 443, 511, 512],
       [513, 521, 522, 523, 531],
       [532, 533, 541, 542, 543]])
```
In [105…
```
arr13.ndim
```
Out[105…
```
2
```
In [106…
```
a=np.arange(1,25).reshape(6,4)
```
In [107…
```
a
```
Out[107…
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24]])
```
In [108…
```
a.shape
```
Out[108…
```
(6, 4)
```
In [109…
```
type(np.hsplit(a,2))
```
Out[109…
```
list
```
In [112…
```
print(np.hsplit(a,4))
```
```
[array([[ 1],
       [ 5],
       [ 9],
       [13],
       [17],
       [21]]), array([[ 2],
       [ 6],
       [10],
       [14],
       [18],
       [22]]), array([[ 3],
       [ 7],
       [11],
       [15],
       [19],
       [23]]), array([[ 4],
       [ 8],
       [12],
       [16],
       [20],
       [24]])]
```
In [113…
```
x=np.arange(60).reshape(12,5)
print(x)
```
```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]
 [30 31 32 33 34]
 [35 36 37 38 39]
 [40 41 42 43 44]
 [45 46 47 48 49]
 [50 51 52 53 54]
 [55 56 57 58 59]]
```
In [114…
```
print(np.hsplit(x,2))
```
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-114-ed6356a7c1f5> in <module>
----> 1 print(np.hsplit(x,2))

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

~\anaconda3\lib\site-packages\numpy\lib\shape_base.py in hsplit(ary, indices_or_sections)
    938         raise ValueError('hsplit only works on arrays of 1 or more dimensions')
    939     if ary.ndim > 1:
--> 940         return split(ary, indices_or_sections, 1)
    941     else:
    942         return split(ary, indices_or_sections, 0)

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

~\anaconda3\lib\site-packages\numpy\lib\shape_base.py in split(ary, indices_or_sections, axis)
    870         N = ary.shape[axis]
    871         if N % sections:
--> 872             raise ValueError(
    873                 'array split does not result in an equal division') from None
    874     return array_split(ary, indices_or_sections, axis)

ValueError: array split does not result in an equal division
```



Vertically split

In [115…
```
print(np.vsplit(x,2))
```
```
[array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29]]), array([[30, 31, 32, 33, 34],
       [35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44],
       [45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54],
       [55, 56, 57, 58, 59]])]
```
In [116…
```
z=np.vsplit(x,4)
print(z)
```
```
[array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]]), array([[15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29]]), array([[30, 31, 32, 33, 34],
       [35, 36, 37, 38, 39],
       [40, 41, 42, 43, 44]]), array([[45, 46, 47, 48, 49],
       [50, 51, 52, 53, 54],
       [55, 56, 57, 58, 59]])]
```
In [117…
```
x=np.arange(16).reshape(4,4)
print(x)
```
```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```
In [120…
```
z=np.vsplit(x,np.array([1,2]))      #:1,1:2,2:3,3:
print(z)
```
```
[array([[0, 1, 2, 3]]), array([[4, 5, 6, 7]]), array([[ 8,  9, 10, 11],
       [12, 13, 14, 15]])]
```
In [121…
```
x=np.arange(32).reshape(2,4,4)
print(x)
```
```
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]
  [12 13 14 15]]

 [[16 17 18 19]
  [20 21 22 23]
  [24 25 26 27]
  [28 29 30 31]]]
```

```
### numpy.dsplit(ary, indices_or_sections)
#### Split array into multiple sub-arrays along the 3rd axis (depth)
```

In [124…
```
np.dsplit(x,2)
```
Out[124…
```
[array([[[ 0,  1],
         [ 4,  5],
         [ 8,  9],
         [12, 13]],

        [[16, 17],
         [20, 21],
         [24, 25],
         [28, 29]]]),
 array([[[ 2,  3],
         [ 6,  7],
         [10, 11],
         [14, 15]],

        [[18, 19],
         [22, 23],
         [26, 27],
         [30, 31]]])]
```
In [125…
```
zero_arry = np.zeros((3,3,3))
```
In [126…
```
zero_arry
```
Out[126…
```
array([[[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]]])
```

Let us Find How to verify that numpy arrays are `faster` than the python lists

In [127…
```
import sys
import time
import numpy as np

def list_add(l1,l2):
 i=0
 for a in l2:
  l1[i]=l1[i]+a
  i=i+1


count = 10000
l1=list(range(count))
l2=list(range(count))

start=time.time()
list_add(l1,l2)
end=time.time()

elapsed = 1000*(end-start)
print("List addition time is ",elapsed)

a1 = np.arange(count)
a2 = np.arange(count)

start=time.time()
a3=a1+a2
end=time.time()

elapsed = 1000*(end-start)
print("Array addition time is ",elapsed)
```
```
List addition time is  1.9953250885009766
Array addition time is  1.0061264038085938
```

Let us Understand about the `ravel function`,its used to convert an `N dimensional array` into a `1 dimensional array`.

In [128…
```
b=np.array([[1,2],[3,4]])
a = b.ravel()
print(a)
type(a)
```
```
[1 2 3 4]
```
Out[128…
```
numpy.ndarray
```
In [129…
```
x=np.arange(16).reshape(4,2,2)
print(x)
```
```
[[[ 0  1]
  [ 2  3]]

 [[ 4  5]
  [ 6  7]]

 [[ 8  9]
  [10 11]]

 [[12 13]
  [14 15]]]
```
In [134…
```
x.shape
```
Out[134…
```
(4, 2, 2)
```
In [135…
```
s=x.ravel()
s
```
Out[135…
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```
In [136…
```
s.shape
```
Out[136…
```
(16,)
```
In [131…
```
print(x.max())     #maximum value
```
```
15
```
In [132…
```
print(x.min())     #minimmum value
```
```
0
```
In [133…
```
print(x.mean())        #mean value
```
```
7.5
```

What is Variance.

Variance measures how far a data is spread out. The technical definition is "The average of the squared differences from the mean," but all it really does is to give you a very general idea of the spread of your data. A value of zero means that there is no variability; All the numbers in the data set are the same.

For Example :-

The data set 12, 12, 12, 12, 12 has a var. of zero (the numbers are identical).

The data set 12, 12, 12, 12, 13 has a var. of 0.167; a small change in the numbers equals a very small var.

The data set 12, 12, 12, 12, 13,013 has a var. of 28171000; a large change in the numbers equals a very large number.

In [137…
```
print(x.var())    #Variance
```
```
21.25
```

what is standard deviation.

The square root of the variance is the standard deviation. While var. gives you a rough idea of spread, the standard deviation is more concrete, giving you exact distances from the mean.

In [138…
```
print(x.std())      #standard deviation
```
```
4.6097722286464435
```
In [139…
```
x=np.arange(16).reshape(4,2,2)
print(x)
```
```
[[[ 0  1]
  [ 2  3]]

 [[ 4  5]
  [ 6  7]]

 [[ 8  9]
  [10 11]]

 [[12 13]
  [14 15]]]
```

** HAPPY LEARNING *