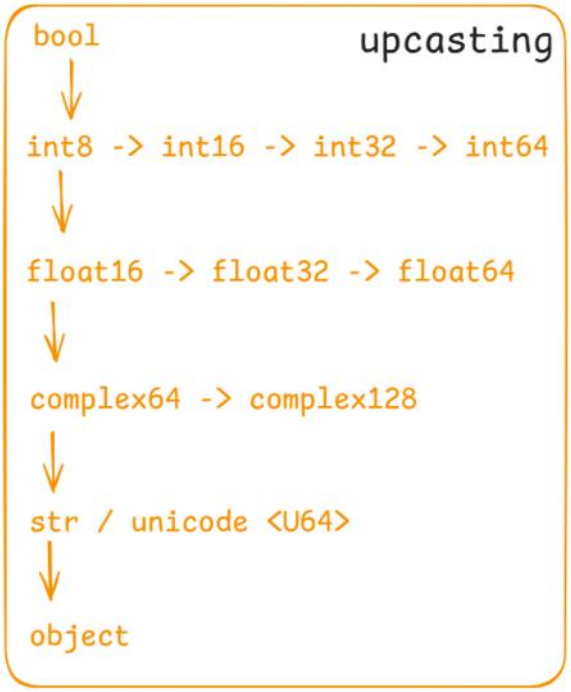
NumPy-II

Session Objectives:

- Apply indexing and slicing on arrays
- Understand how to perform common array operations using NumPy.
- Perform sorting, reshaping, and concatenating arrays.
- Distinguish between lists and arrays.
- Learn statistical and transformation operations on arrays.




```
arr_2d = np.array([
    [1,2,3,4,5],
    [1,3,5,7,9],
    [2,4,6,8,0],
    [5,4,3,2,1],
])
print(arr_2d[1:3, 2:4]) # [[5 7] [6  8]]
```

```
[[5 7]
 [6 8]]
```

```
print(arr_2d[-3::1, 3::-1]) #
```

```
[7 5 3 1]
[8 6 4 2]
[2 3 4 5]
```

```
Top -> Bottom [+ ve]
Bottom -> Top [- ve]

Left -> Right [+ ve]
Right -> Left [- ve]
```

```python
import numpy as np
# Specifying the Data Types [Upcasting]
nested_list = np.array([
    [11,22],
    [33,44],
    [55,66],
    [77,88],
    [99,99.99],
]) # upcast -> float
print(nested_list)
```

```
[[11.   22.  ]
 [33.   44.  ]
 [55.   66.  ]
 [77.   88.  ]
 [99.   99.99]]
```

```python
nested_list.dtype
```

```
dtype('float64')
```

```python
nested_list = np.array([
    [11,22],
    [33,44],
    [55,66],
    [77,88],
    [99,121],
]) # dtype attribute -> float
float_arr = np.array(nested_list , dtype = float)
float_arr
```

```
array([[ 11.,  22.],
       [ 33.,  44.],
       [ 55.,  66.],
       [ 77.,  88.],
       [ 99., 121.]])
```

```
# upcast ['str'] Vs upcast['float']
nested_list = np.array([
    [11,22],
    [33,44],
    [55,66],
    [77,88],
    [99.99,'121'],
]) # upcast['121'] us winning
float_arr = np.array(nested_list , dtype = float) # external force
float_arr
```

```
array([[ 11.  ,  22.  ],
       [ 33.  ,  44.  ],
       [ 55.  ,  66.  ],
       [ 77.  ,  88.  ],
       [ 99.99, 121.  ]])
```

```
nested_list
```

```
array([['11', '22'],
       ['33', '44'],
       ['55', '66'],
       ['77', '88'],
       ['99.99', '121']], dtype='<U32')
```

```
# upcast ['str'] Vs upcast['float']
nested_list = np.array([
    [11,22],
    [33,44],
    [55,66],
    [77,88],
    [99.99,'k'], # ValueError: could not convert string to float: 'k'
]) # upcast['121'] us winning
float_arr = np.array(nested_list , dtype = float) # Error 'k' won't typecast into float
float_arr
```

```
nested_list
```

```
array([['11', '22'],
       ['33', '44'],
       ['55', '66'],
       ['77', '88'],
       ['99.99', 'k']], dtype='<U32')
```

```
# upcast ['int'] Vs upcast['bool']
nested_list = np.array([
    [11,22],
    [33,44],
    [55,66],
    [77,88],
    [True,False],
]) # upcast [int]
float_arr = np.array(nested_list , dtype = float) # external force
float_arr
```

```
array([[11., 22.],
       [33., 44.],
       [55., 66.],
       [77., 88.],
       [ 1.,  0.]])
```

```
nested_list
```

```
array([[11, 22],
       [33, 44],
       [55, 66],
       [77, 88],
       [ 1,  0]])
```

```
nested_list.dtype
```

```
dtype('int32')
```

```
nested_list.ndim # 2
```

```
2
```

```
nested_list.shape
```

```
(5, 2)
```

```
# size -> no. of elements within the n dimensional array
nested_list.size
```

```
10
```

```
arr_1d = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16])
print(arr_1d)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
```

```
arr_1d.ndim
```

```
1
```

```
arr_1d.size
```

```
16
```

```
arr_1d.shape
```

```
(16,)
```

```
arr_4d = arr_1d.reshape(2,2,2,2)
arr_4d
```

```
array([[[[ 1,  2],
         [ 3,  4]],

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


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

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

```
arr_4d.ndim
```

```
4
```

```
arr_4d.shape
```

```
(2, 2, 2, 2)
```

```
arr_4d.dtype
```

```
dtype('int32')
```

```
arr_4d.size
```

```
16
```

```
# itemsize() -> ndarray.itemsize attribute -> Memory Size of each elements (in Bytes)
# Homogenous Numpy Array
arr_int = np.array([1,2,3,4,5,6,7,8,9,11,15,21,77,99] , dtype = int)
print(arr_int.itemsize, "bytes")
```

```
4 bytes
```

```
arr_float = np.array([1,2,3,4,5,6,7,8,9,11,15,21,77,99] , dtype = float)
print(arr_float)
print(arr_float.itemsize, "bytes")
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 11. 15. 21. 77. 99.]
8 bytes
```

```
arr_str = np.array(['Coding','Python','Programming','Numpy','Array'] , dtype = str)
print(arr_str)
print(arr_str.itemsize , "bytes")
```

```
['Coding' 'Python' 'Programming' 'Numpy' 'Array']
44 bytes
```

```
arr_str = np.array(['Coding','Python','Java','Array'] , dtype = str)
print(arr_str)
print(arr_str.itemsize , "bytes") # 6 * 4 bytes ['each char'] = 24
```

```
['Coding' 'Python' 'Java' 'Array']
24 bytes
```

```
# ndarray.data # 'Memory Location' -> memory View
arr = np.array([11,22,33,44,55,66,77])
print("Data Buffer: " , arr.data) # Memory Address
print("Type of Buffer" , type(arr.data)) # 'Memory View'
```

```
Data Buffer:  <memory at 0x0000024340738700>
Type of Buffer <class 'memoryview'>
```

```python
# Indexing , slicing , Operations on numpy array
arr_1d = np.array([11,22,33,44,55])
print(arr_1d[2]) # 33
print(arr_1d[-2]) # 44
print(arr_1d[-5]) # 11
# print(arr_1d[5]) # Index Error: Out of Range
print(arr_1d[0]) # 11
```

```
33
44
11
11
```

```python
# Shape of 3D Array (2,3,4) -> 2(depth),3(rows),4(cols)
arr_3d = np.array([
    [[11,22,33,44],
     [22,44,66,88],
     [11,33,55,77],],

    [[66,77,88,99],
     [11,55,77,99],
     [22,44,66,88],]
])
# indexing[depth,row,col]
print(arr_3d[-1,2,3]) # 88
print(arr_3d[-2,-3,-4]) # 11
# print(arr_3d[0,3,2]) # Error
# print(arr_3d[-2,0,4]) # Error
print(arr_3d[1,2,-3]) # 44
print(arr_3d[0,-1,-1]) # 77
print(arr_3d[-2,1,-4]) # 22
print(arr_3d[0,-3,3]) # 44
```

```
88
11
44
77
22
44
```

```python
# Slicing[Start = 0 , Stop = length [Exclusive], Step = 1 [default]]
arr_1d = np.array([11,22,33,44,55,66,77,88,99])
print(arr_1d[-3:]) # [77,88,99]
print(arr_1d[-3::-1]) # [77,66...11]
print(arr_1d[-3:9:-1]) # []
print(arr_1d[-3:-6:-1]) # [77,66,55]
print(arr_1d[-3:0:-1]) # [77,66....22]
print(arr_1d[5:9:2]) # [66,88]
print(arr_1d[5:4:2]) # []
print(arr_1d[5:4:-2]) # [66]
print(arr_1d[8:0:-2]) # [99,77,55,33]
print(arr_1d[-4::-2]) # [66,44,22]
print(arr_1d[3:8:-3]) # []
print(arr_1d[5:11:2]) # [66,88]
```

```
[77 88 99]
[77 66 55 44 33 22 11]
[]
[77 66 55]
[77 66 55 44 33 22]
[66 88]
[]
[66]
[99 77 55 33]
[66 44 22]
[]
[66 88]
```

```python
# Slicing 2D [Start,Stop,Step]-> arr_2d [row_slicing , col_slicing] (4,5)
arr_2d = np.array([
    [1,2,3,4,5],
    [1,3,5,7,9],
    [2,4,6,8,0],
    [5,4,3,2,1],
])
print(arr_2d[1:3, 2:4]) # [[5 7] [6  8]]
```

```
[[5 7]
 [6 8]]
```

```python
print(arr_2d[-3::1, 3::-1]) #
```

```
[[7 5 3 1]
 [8 6 4 2]
 [2 3 4 5]]
```

```python
arr_2d = np.array([
    [1,2,3,4,5],
    [1,3,5,7,9],
    [2,4,6,8,0],
    [5,4,3,2,1],
])
print(arr_2d[3::-2 , 1:4:3]) # [[4][3]]
```

```
[[4]
 [3]]
```

```python
arr_2d = np.array([
    [1,2,3,4,5],
    [1,3,5,7,9],
    [2,4,6,8,0],
    [5,4,3,2,1],
])
print(arr_2d[2:5 , 2:5:-2]) # []
```

```
[]
```

```python
arr_2d = np.array([
    [1,2,3,4,5],
    [1,3,5,7,9],
    [2,4,6,8,0],
    [5,4,3,2,1],
])
print(arr_2d[2:5 , 2:5:2]) # [[6 0] [3 1]]
```

```
[[6 0]
 [3 1]]
```

```python
arr_2d = np.array([
    [1,2,3,4,5],
    [1,3,5,7,9],
    [2,4,6,8,0],
    [5,4,3,2,1],
])
print(arr_2d[: , ::-2])
```

```
[[5 3 1]
 [9 5 1]
 [0 6 2]
 [1 3 5]]
```

```python
arr_2d = np.array([
    [1,2,3,4,5],
    [1,3,5,7,9],
    [2,4,6,8,0],
    [5,4,3,2,1],
])
print(arr_2d[::-1 , ::-1]) # row in reverse , and col in reverse
# row in reverse
# [5,4,3,2,1]
# [2,4,6,8,0]
# [1,3,5,7,9]
# [1,2,3,4,5]
# col reverse wrt to row in reverse [right to left]
# [1,2,3,4,5]
# [0,8,6,4,2]
# [9,7,5,3,1]
# [5,4,3,2,1]
```

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

```
# slicing in 3D Array [depth , row , col]
arr_3d = np.array([
    [[11,22,33,44],
     [22,44,66,88],
     [11,33,55,77],],

    [[66,77,88,99],
     [11,55,77,99],
     [22,44,66,88],]
])
arr_3d[:,:,::-1] # Both depth including all rows but cols reversed
```

```
array([[[44, 33, 22, 11],
        [88, 66, 44, 22],
        [77, 55, 33, 11]],

       [[99, 88, 77, 66],
        [99, 77, 55, 11],
        [88, 66, 44, 22]]])
```

```
arr_3d = np.array([
    [[11,22,33,44],
     [22,44,66,88],
     [11,33,55,77],],

    [[66,77,88,99],
     [11,55,77,99],
     [22,44,66,88],]
])
arr_3d[1::2, 0:3:2, ::-3]
```

```
array([[[99, 66],
        [88, 22]]])
```

```
arr_3d = np.array([
    [[11,22,33,44],
     [22,44,66,88],
     [11,33,55,77],],

    [[66,77,88,99],
     [11,55,77,99],
     [22,44,66,88],]
])
print(arr_3d[-2,2,0]) # 11
arr_3d[-2: , -1: , 0:3:2] # last row of both depth
```

```
11
array([[[11, 55]],

       [[22, 66]]])
```

```
# SELECT * FROM TABLENAME WHERE = <Apply Filter>
# Masking -> Uses a boolean array of the same shape to filter the original array
arr_1d = np.array([11,22,33,44,55,66,77,88,99])
mask_arr = np.array([True,True,False,False,True,False,True,False,True])
print(arr_1d[mask_arr]) # Filtering on the numpy array
```

```
[11 22 55 77 99]
```

```
arr_1d.shape
```

```
(9,)
```

```
mask_arr.shape
```

```
(9,)
```

```
arr_1d = np.array([11,22,33,44,55,66,77,88,99])
mask_arr = np.array([True,True,False,False,True,False,True])
print(arr_1d.shape)
print(mask_arr.shape)
```

```
(9,)
(7,)
```

```
print(arr_1d[mask_arr]) # Error
# IndexError: boolean index did not match indexed array along dimension 0;
# dimension is 9 but corresponding boolean dimension is 7
```

```
# arr_2d [Masking]
arr_2d = np.array([
    [1,2,3,4,5],
    [1,3,5,7,9],
    [2,4,6,8,0],
    [5,4,3,2,1]
])
mask_2d = np.array([
    [True,True,False,False,True],
    [True,False,True,True,True],
    [True,False,False,False,False],
    [True,False,False,True,True]
])
# Final Result -> 1D [Flatten Up]
arr_2d[mask_2d]
```

```
array([1, 2, 5, 1, 5, 7, 9, 2, 5, 2, 1])
```

```
arr_3d = np.array([
    [[11,22,33,44],
     [22,44,66,88],
     [11,33,55,77],],

    [[66,77,88,99],
     [11,55,77,99],
     [22,44,66,88],]
])
mask_3d = np.array([
    [[True,True,False,False],
     [True,False,False,False],
     [True,False,True,True],],

    [[False,True,False,True],
     [True,True,True,True],
     [False,False,False,False],]
])
arr_3d[mask_3d] # flatten up
```

```
array([11, 22, 22, 11, 55, 77, 77, 99, 11, 55, 77, 99])
```

```
# Ellipsis [...] -> Used Only Once in a slicing -> 3D Matrix
# 3D [2,5,2]
arr_3d = np.array([
    [[11,22],
    [33,44],
    [55,66],
    [77,88],
    [99,110],],

    [[1,2],
    [3,4],
    [5,6],
    [7,8],
    [9,10],]
])
arr_3d[...] # arr_3d[:,:,:]
```

```
array([[[ 11,  22],
        [ 33,  44],
        [ 55,  66],
        [ 77,  88],
        [ 99, 110]],

       [[  1,   2],
        [  3,   4],
        [  5,   6],
        [  7,   8],
        [  9,  10]]])
```

```
# Ellipsis [...] -> Used Only Once in a slicing -> 3D Matrix
# 3D [2,5,2]
arr_3d = np.array([
    [[11,22],
    [33,44],
    [55,66],
    [77,88],
    [99,110],],

    [[1,2],
    [3,4],
    [5,6],
    [7,8],
    [9,10],]
])
arr_3d[...,1] # (depth,rows,cols)
```

```
array([[ 22,  44,  66,  88, 110],
       [  2,   4,   6,   8,  10]])
```

```
arr_3d[:,:,1] # (depth,rows,cols)
```

```
array([[ 22,  44,  66,  88, 110],
       [  2,   4,   6,   8,  10]])
```

```
arr_3d.shape
```

```
(2, 5, 2)
```

```
arr_3d = np.array([
    [[11,22],
    [33,44],
    [55,66],
    [77,88],
    [99,110],],

    [[1,2],
    [3,4],
    [5,6],
    [7,8],
    [9,10],]
])
arr_3d[:,0,...] # (depth,rows,cols)
```

```
array([[11, 22],
       [ 1,  2]])
```

```
arr_3d[:,0,:] # (depth,rows,cols)
```

```
array([[11, 22],
       [ 1,  2]])
```

```
arr_3d = np.array([
    [[11,22],
    [33,44],
    [55,66],
    [77,88],
    [99,110],],

    [[1,2],
    [3,4],
    [5,6],
    [7,8],
    [9,10],]
])
arr_3d[...,-1] # (depth,rows,cols)
```

```
array([[ 22,  44,  66,  88, 110],
       [  2,   4,   6,   8,  10]])
```

```
arr_3d[:,:,-1] # (depth,rows,cols)
```

```
array([[ 22,  44,  66,  88, 110],
       [  2,   4,   6,   8,  10]])
```

```
arr_3d = np.array([
    [[11,22],
    [33,44],
    [55,66],
    [77,88],
    [99,110],],

    [[1,2],
    [3,4],
    [5,6],
    [7,8],
    [9,10],]
])
arr_3d[0,...] # (depth,rows,cols)
```

```
array([[ 11,  22],
       [ 33,  44],
       [ 55,  66],
       [ 77,  88],
       [ 99, 110]])
```

```
arr_3d[0,:,:] # (depth,rows,cols)
```

```
array([[ 11,  22],
       [ 33,  44],
       [ 55,  66],
       [ 77,  88],
       [ 99, 110]])
```

```
# Operations on Array
a = np.array([11,22,33,44,55,66,77,88,99])
b = np.array([7,9,11,17,21,29,41,77,91])

print(a.shape)
print(b.shape)
```

```
(9,)
(9,)
```

```
# Addition
print("Numpy Array Addition: ")
a+b
```

```
Numpy Array Addition:
array([ 18,  31,  44,  61,  76,  95, 118, 165, 190])
```

```
# Subtractions
print("Numpy Array Subtraction: ")
a-b
```

```
Numpy Array Subtraction:
array([ 4, 13, 22, 27, 34, 37, 36, 11,  8])
```

```
# Multiplications
print("Numpy Array Multiplication: ")
a*b
```

```
Numpy Array Multiplication:
array([  77,  198,  363,  748, 1155, 1914, 3157, 6776, 9009])
```

```
# Divisions
print("Numpy Array Divisions: ")
a/b
```

```
Numpy Array Divisions:
array([1.57142857, 2.44444444, 3.        , 2.58823529, 2.61904762,
       2.27586207, 1.87804878, 1.14285714, 1.08791209])
```

```
# Floor Divisions
print("Numpy Array Floor Division: ")
a//b
```

```
Numpy Array Floor Division:
array([1, 2, 3, 2, 2, 2, 1, 1, 1])
```

```
# Modulus
print("Numpy Array Modulus: ")
a%b
```

```
Numpy Array Modulus:
array([ 4,  4,  0, 10, 13,  8, 36, 11,  8])
```

```
# Exponentiations
print("Numpy Array Exponent: ")
a ** 2
```

```
Numpy Array Exponent:
array([ 121,  484, 1089, 1936, 3025, 4356, 5929, 7744, 9801])
```

```
# Exponentiations
print("Numpy Array Exponent: ")
b ** 2
```

```
Numpy Array Exponent:
array([  49,   81,  121,  289,  441,  841, 1681, 5929, 8281])
```

```
# Operations on Array [Unequal shape] -> Won't perform the operations
a = np.array([11,22,33,44,55,66,77,88,99])
b = np.array([7,9,11,17,21,29,41])

print(a.shape)
print(b.shape)
```

```
(9,)
(7,)
```

```
# Addition
print("Numpy Array Addition: ")
a+b
ValueError: operands could not be broadcast together with shapes (9,) (7,)
```

```python
# # Operations on 2D Array
A = np.array([
    [11,22],
    [33,44],
    [55,66],
    [77,88],
    [99,110]
])
B = np.array([
    [1,2],
    [3,4],
    [5,6],
    [7,8],
    [9,10]
])
print(A.shape)
print(B.shape)
```

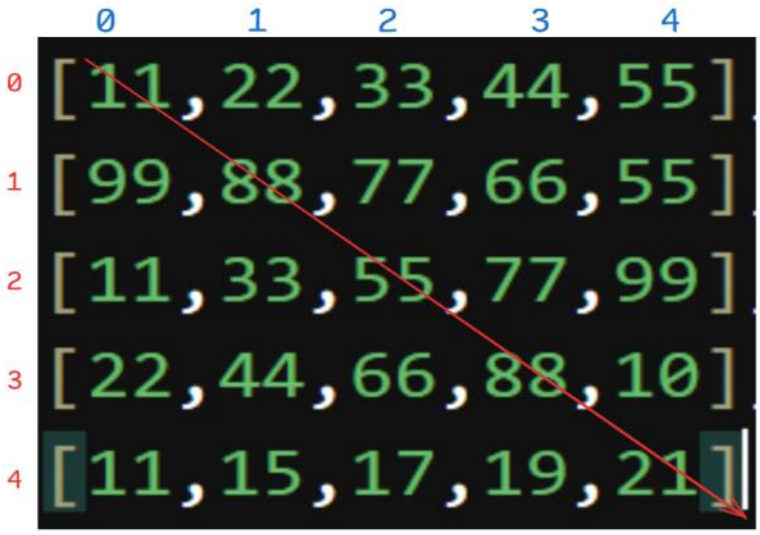
```
(5, 2)
(5, 2)
```

A+B

```
array([[ 12,  24],
       [ 36,  48],
       [ 60,  72],
       [ 84,  96],
       [108, 120]])
```

A-B

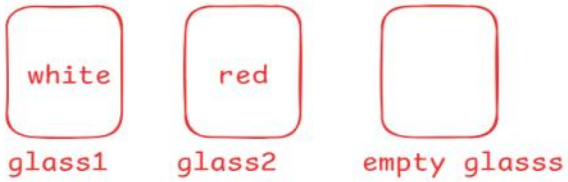
```
array([[ 10,  20],
       [ 30,  40],
       [ 50,  60],
       [ 70,  80],
       [ 90, 100]])
```

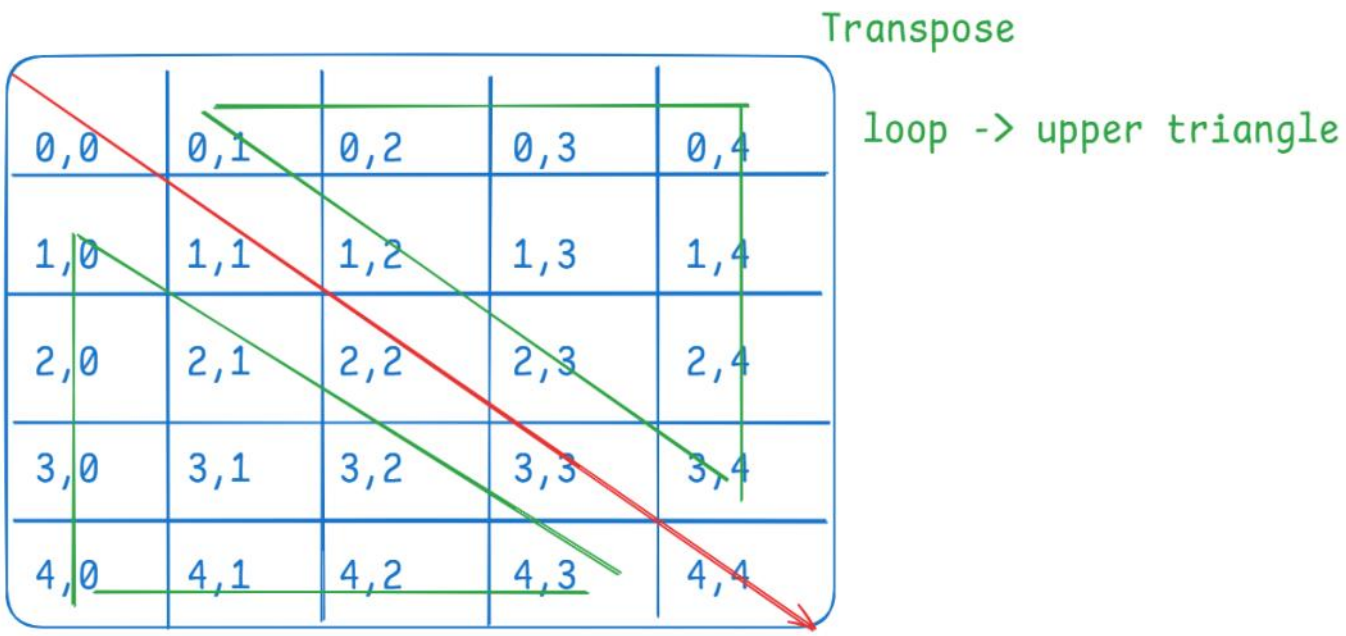


swap(i,j) => (j,i)

diagonal [i==j]

Swapping a number
a = ~~10~~ 20
b = ~~20~~ 10
temp = ~~None~~ 10






```
# Transposing an Array : Converting rows into coloumns and vice-versa. [Square Matrix]
arr_2d = np.array([
    [11,22,33,44,55],
    [99,88,77,66,55],
    [11,33,55,77,99],
    [22,44,66,88,10],
    [11,15,17,19,21]
])
print("Original Array: \n", arr_2d)
```

```
Original Array:
 [[11 22 33 44 55]
 [99 88 77 66 55]
 [11 33 55 77 99]
 [22 44 66 88 10]
 [11 15 17 19 21]]
```

```
print("Transposed Array: \n", arr_2d.transpose())
```

```
Transposed Array:
 [[11 99 11 22 11]
 [22 88 33 44 15]
 [33 77 55 66 17]
 [44 66 77 88 19]
 [55 55 99 10 21]]
```

```
print("Transposed Array: \n", arr_2d.T)
```

```
Transposed Array:
 [[11 99 11 22 11]
 [22 88 33 44 15]
 [33 77 55 66 17]
 [44 66 77 88 19]
 [55 55 99 10 21]]
```