Friday, April 26, 2024 9:20 PM

Array:

```
a = [10, 20, 30, 40, 50]
print(a)
```

Suppose, we have array, a = | 10 | 20 | 30 | 40 | 50 |

0 1 2 3 4

Let's say we're starting from 0.

```
i = 0
while i < len(a):
    print(a[i])
    i = i+1
```

↳ element accessing

output:

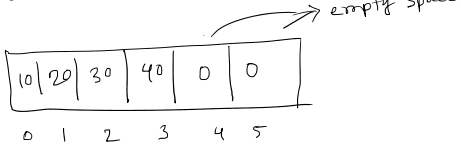
```
a[0] = 10
a[1] = 20
.
.
i
```

doing it in function:

```
def print_array(a):
    i = 0
    while i < len(a):
        print(a[i])
        i = i+1

a = [10, 20, 30, 40, 50]
print_array(a)
```

array size & array len:



Here, array len is 6; size is 4

* if I wanna start from the end, I'll have to consider it as i = len(a) - 1

printing array reversely:

```
a = [0] * 5
i = len(a) - 1
while i >= 0:
    print(a[i])
    i = i - 1
```

| 0 | 0 | 0 | 0 | 0 |

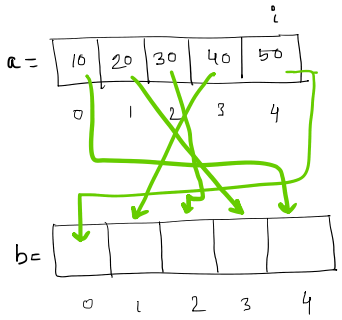
↳ here, we iterated the array reversely

If we wanna reverse an array:

means, if we have | 10 | 20 | 30 | 40 | ; it'll be as | 40 | 30 | 20 | 10 |

So, this can be done in 2 ways:

i) out of place:




```
def reverse_out_of_place(arr):
    arr2= np.zeros(len(arr), dtype=int)
    i= 0
    while(i<=len(arr)-1):
        arr2[i]= arr[len(arr)-1-i]
        i+=1
    return arr2
arr= np.array([1,2,3,4])
print(reverse_out_of_place(arr))
arr= np.array([1,2,3,4,5])
print(reverse_out_of_place(arr))
```

In a, I'm starting my 'i' from 50. So, b array এর j index এ we can copy i

∴ b[j] = a[i]

j + = 1

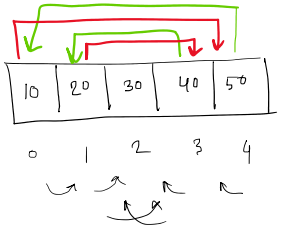
i - = 1

এভাবে, নতুন array create করে reversely copy করা হয়।

ii) in place:

first & last index swap করে করে

```
def reverse_in_place(arr):
    x, j = 0, len(arr)-1
    for i in range(len(arr)//2):
        arr[i], arr[j]= arr[j], arr[i]
        x+=0
        j-=1
    return arr
arr= np.array([1,2,3,4])
print(reverse_in_place(arr))
arr= np.array([1,2,3,4,5])
print(reverse_in_place(arr))
```

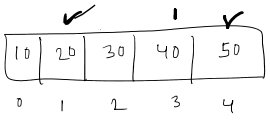


i < j

3 1

↳ won't fulfil anymore. তাহলে off হয়ে যাবে,

↳ tracing this → suppose, we are not creating any new array. just working on...



→ swaping

i	j
0	4
1	3
2	2
3	1 → এভাবে stop হয়ে যাবে।

len (arr) - 1 - i

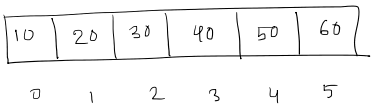
& len(arr) - 2

Array Deleting / Shifting:

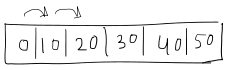
3 1 ...

Array Rotating & Shifting:

Right shift :



1 block right shift →



* Right shift must be start from last block. ডানের 60 থেকে start হবে।

Rotating এ, temp এ first এলিমেন্ট নিবে।
Then, a[0] = temp.

Some Operations:

Creating 1D array:

```
array_1D = [None] *5
```

Creating 2D array [4x5]:

```
array_2D = [None] * 4
for i in range(len(array_2D)):
    array_2D[i] = [None] * 5
```

How to access elements:

```
print(array_1D[3])
print(array_2D[2][3])
```

→ 3rd element of 2nd row

Creating array with Numpy:

numpy.empty(shape, data_type) → creates array filled with None
numpy.zeros(shape, data_type) → " " " zeros

```
import numpy as np
arr = np.empty(2, dtype = int)
```

when creating an array, you must specify the length of its dimension that you can't change later.

```
# Let's look at the length property clearly
array = [None]*10
```

```python
# Let's look at the length property clearly
array = [None]*10
#We just declared an array of 10 length. Let's fill it up with something.

Names = ['Emon','Nusrat','Joy','Abrar','Ifty','Ashik','Raisa','Suba','Sazid','Jawad']
for i in range(10):
  array[i] = Names[i]
print(array)
```

```
['Emon', 'Nusrat', 'Joy', 'Abrar', 'Ifty', 'Ashik', 'Raisa', 'Suba', 'Sazid', 'Jawad']
```

Iteration : means checking value index by index.

```
def iteration (source):
    for i in range ( len(source) ):
        print ( source[i] )

def reverse_iteration (source):
    for i in range( len(source) - 1, -1, -1):
        print( source [i] )
```

Copy Array : [Pass by value]

↳ initializing new array with the same length given and then copying the value one by one.

as just memory location store হয়, just copying value is not enough for array copying.

```
def copy_array( arr ):
    new_arr = np.array ([0] * len(arr))
    for i in range( len( source )):
        new_arr[i] = arr [i]

    return new_arr
arr1 = np.array ([1,2,3,4])
```

Resize Array : fixed memory location থাকায় কারণে we cannot resize array.

But resize করতে চাইলে →

1. creating new array with new length
2. copying value from original array
3. if we have [10|20|30|40|50] → len (5)

↓ resizing

[10|20|30|40|50|None|None|None]

len (8)

Code :

```
import numpy as np
def resize_array (old_array, new_capacity) :
    new_array = np.array( [0] * new_capacity )
    for i in range (len ( old_array)):
```

```
    new_array = np.array([0] * new_capacity)
    for i in range(len(old_array)):
        new_array[i] = old_array[i]
    return new_array

arr1 = np.array([1,2,3])
x = resize_array(arr1, 6)
```

Reversing an array:

↳ out of place operation:

- Creating new array with same size
- then copying value in reverse order

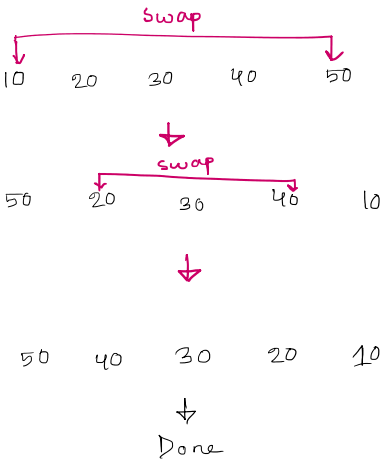
Code:

```
def rev_arr_op(arr):
    rev_arr = [None] * len(arr)
    i = 0
    j = len(arr) - 1
    while i < len(arr):
        rev_arr[i] = arr[j]
        i += 1
        j -= 1
    return rev_arr
```

↳ In place operation:

- Swaping starting value with the end value
- it's reversing the array in the original array

Example:



Code:

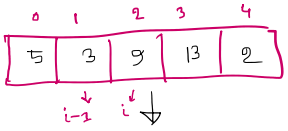
```
def rev_arr_inplace(arr):
    i = 0
```

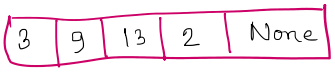
```
j = len(arrr) - 1
while i < j:
    temp = arrr[i]
    arrr[i] = arrr[j]
    arrr[j] = temp
    i += 1
    j -= 1
```

Shifting:

Left shift: shift amount wise element সবগুলো বামদিকে সরে যাবে।

ex:





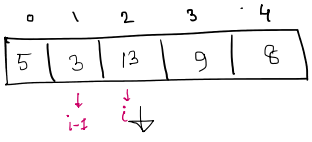
we'll lose 5।

Code:

```
def left_shift(arrr):
    for i in range(1, len(arrr), 1):
        arrr[i-1] = arrr[i]
    arrr[len(arrr)-1] = None
    return arrr
arrr = np.array([1, 2, 3])
```

 # Right Shift: shift amount wise element সবগুলো ডানে দিব।
and first এর দিকের box blank থাকবে।

last element will be lost forever.



Last theke start হবে।
* আগে left থেকে right এ shifting
right থেকে left এ iterating

None, 5, 3, 13, 9

Code:

```
def right_shift(arrr):
    for i in range(len(arrr)-1, 0, -1):
        arrr[i] = arrr[i-1]
```

```
arr[0] = None
return arr
```

Rotating:

Rotating array left:

- ↳ eqv. to shifting circular array left
- ↳ 1st element won't be lost as it'll move to last slot

3 9 5 13 2

↓

9 5 13 2 3

← - - - -

Code:

```
def rotate_left(arr):
    temp = arr[0]
    for i in range(1, len(arr)):
        arr[i-1] = arr[i]
    arr[len(arr) - 1] = temp
    return arr
```

Rotating array right:

- ↳ last element will move to the 1st slot
- ↳ element গুলো right এ সরতে থাকবে।

2 5 3 9 13

↓

13 2 5 3 9

Code:

```
def rotate_right(arr):
    temp = arr[len(arr) - 1]
    for i in range(len(arr) - 1, 0, -1):
        arr[i] = arr[i-1]
    arr[0] = temp
```

return arr

arr = np.array([1,2,3])
print(rotate_right(arr))

Insertion :

* Inserting an element into an array *

a. making an empty slot, insert the value in array

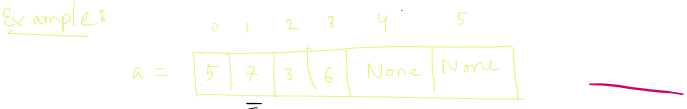
2. to make empty slot, first check any slots are available. If not then we need to resize the array.

b. Using idea of right shift to have an empty slot.

[Initialization point of right shift would be index where we wanna insert value]

8. we need to check if asked index is valid or not

Example:



suppose, you wanna insert 10 at idx 1

Code :

def insert_element(arr, size, elem, index):

if size == len(arr):

print("No space. No insertion")

else:

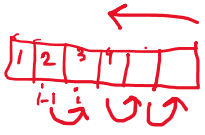
for i in range(size, index, -1):

arr[i] = arr[i-1] # shifting right till the index

arr[index] = elem # Inserting element

return arr

arr = np.zeros(4, dtype=int)



* Inserting at the end of the array, no shifting is needed.

```
#Inserting anywhere

def insert_anywhere(arr, size, index, elem):
    if index<0 or index>size:
        return "Insertion Not Possible"

    if size >= len(arr):
        arr= resizeArray(arr, len(arr)+3)

    for i in range(size, index, -1): #Right Shift
        arr[i]= arr[i-1]
```

```
#Inserting at the end


def insert_at_the_end(arr, size, elem):
    if size >= len(arr):
        arr= resizeArray(arr, len(arr)+5)
    arr[size]= elem
    return arr

arr= np.zeros(4, dtype=int)
arr[0], arr[1], arr[2] = 4, 6, 5
print(arr)
```

```
        arr= resizeArray(arr, len(arr)+3)

    for i in range(size, index, -1): #Right Shift
        arr[i]= arr[i-1]

    arr[index]= elem
    return arr

arr= np.zeros(4, dtype=int)
arr[0], arr[1], arr[2] = 4, 6, 5
print(arr)
print(insert_anywhere(arr, 3, 1, 15)) #array, size, index, elem
print(insert_anywhere(arr, 4, 4, 25)) #array, size, index, elem, the size increased


# If you want to insert at the end with this function too,
# all you have to do is provide the index value equal to the size
```

```
    return arr

arr= np.zeros(4, dtype=int)
arr[0], arr[1], arr[2] = 4, 6, 5
print(arr)
print(insert_at_the_end(arr, 3, 15)) #array, size, elem
print(insert_at_the_end(arr, 4, 25)) #array, size, elem
```

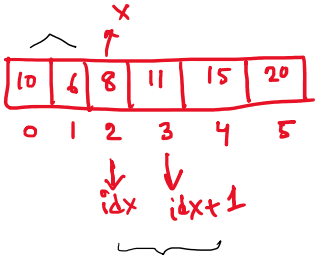
Removing Element from Array:

start left shifting from the index which value we wanna remove.

a = [10 6 8 11 15 20]

↓ will rmv 8 [left shift]

[10 6 11 15 20 None]



Code:

```
def rmv_element( arrr, index, size):
    if size==0 or (index<0 and index=>size):
        return "deletion not possible"
    for i in range (index+1, size):
        arrr[i-1] = arrr[i] # shifting left from rmving index
    arrr[size-1] = None  # making last space empty
    return arrr

arrr= np.zeros(5, dtype=int)
arrr[0], arrr[1], arrr[2], arrr[3] = 4, 6, 5, 2
print(arrr)
print(rmv_element(arrr, 4, 0)) # arrray, size, index
```

Multidimentional Array

~~Ram~~ RAM এর মধ্যে multid. array 1D হিসাবে store করে রাখা হয়।

How to determine the translated linear index of a multidimensional idx from original array?

↳ suppose 4D array:

dim: W x N x O x P

elem of box [w][x][y][z]

∴ index of that element in the linear array is:

$W \times (N \times O \times P) + X \times (O \times P) + Y \times P + Z$

Q: linear array length : 128

dim : 3D ; 4x4x8

what are the multidimensional indexes of the element stored at location 111 ?

Soln:

The dimension of the given array is [4][4][8] and length is 128. We need to map the dimension of linear index 111.

The equation we get is:

X * (4*8) + Y*8 + Z = 111 where 0 <= X <= 3, 0 <= Y <= 3 and 0 <= Z <= 7.

Now to find the value of X, Y and Z we need to repeatedly divide the remainder of the linear index with the multiplication of the lengths of lower dimensions till you reach the last dimension.

So first 111/(4*8) then the quotient is x and then you use the remainder R to do R/8 to get y index. The remainder of that is the z index

Following the process we get,

X = 111//(4*8) = 3 and 111 % (4*8) = 15

Y = 15 // 8 = 1 and 15 % 8 = 7

Z = 7

So, the dimension of linear index 111 is [3][1][7].

Initialization of a Multidimensional Arrray:

```
import numpy as np
m = np.zeros((2,3), dtype = int)
# m = np.array([[4,3,8], [2,5,1]])
```

row, col — Creating empty array → [[0 0 0] [0 0 0]]

creating m.arr. by using array() function → [[4 3 8] [2 5 1]]

Creating empty 2D arrray taking input from user:

```
def create_array():
    m = np.zeros((2,3), dtype= int)
    for i in range(2):
        for j in range(3):
            print(f'element of [{i}][{j}] index : ')
            m[i][j] = int(input())
```

```
return m
```

Iteration of a 2D Matrix:

2. Row wise iteration:

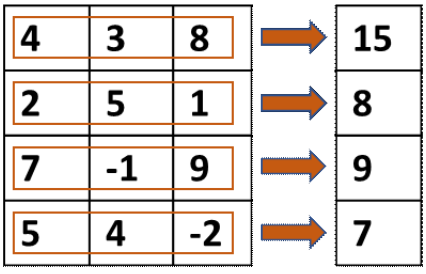
```
def print_row(m):
    row, col = m.shape
    for i in range(row):
        for j in range(cd):
            print(m[i][j], end = ' ')
        print()

m = np.zeros((2,3), dtype = int)
print(m.shape)
```

Sum of all elements in 2D matrix:

```
def array_sum(m):
    sum = 0
    row, column = m.shape
    for i in range(row):
        for j in range(col):
            sumt = m[i][j]
    return sum
```

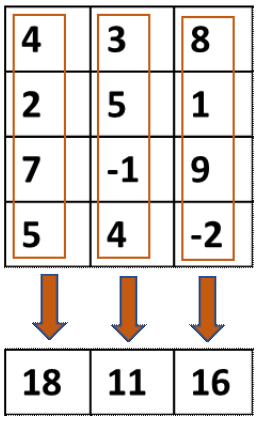
Sum of every row in a given matrix



Thus, the resulting matrix will always have 1 column to store row wise sum and equal number of rows of the main matrix.

```
def row_wise_sum(m):
  row, col = m.shape
  result = np.zeros((row,1), dtype = int)
  for i in range(row):
    for j in range(col):
      result[i][0] += m[i][j]
  return result
```

Sum of every column in a given matrix

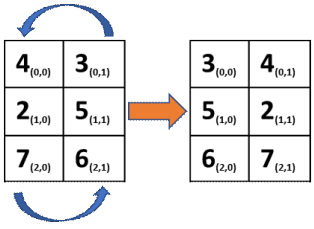


Thus, the resulting matrix will always have 1 row to store column wise sum and equal number of columns of the main matrix.

```
def column_wise_sum(m):
  sum = 0
  row, col = m.shape
  result = np.zeros((1,col), dtype = int)
  for i in range(col):
    for j in range(row):
      result[0][i] += m[j][i]
  return result
```

Swapping:

Swap the two columns of a m x 2 matrix




```
def swap_2columns(m):
  row,col = m.shape
  for i in range(row):
    m[i][0], m[i][1] = m[i][1], m[i][0]
  return m
```

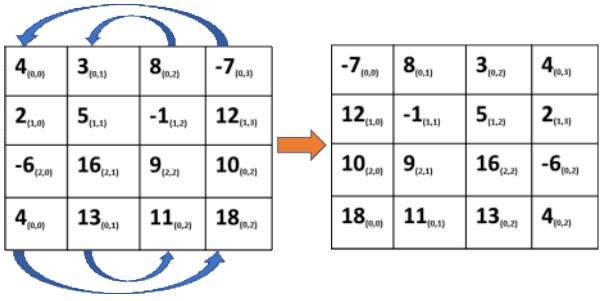
Swap the columns of a m x n matrix

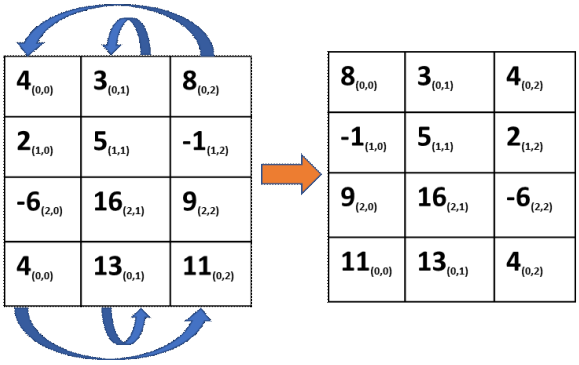
0^{th} column ⇌ $(n-1)^{th}$ column

1^{st} column ⇌ $(n-2)^{th}$ column

2^{nd} column ⇌ $(n-3)^{th}$ column

and so on and so forth

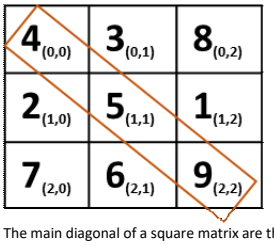





```
def swap_columns(m):
  row,col = m.shape
  for i in range(row):
    for j in range(col//2):
      m[i][j],m[i][col-1-j] = m[i][col-1-j],m[i][j]
  return m
```

Addition:

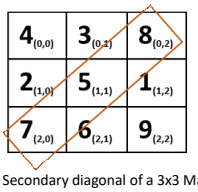
Add the elements of the primary diagonal in a square matrix



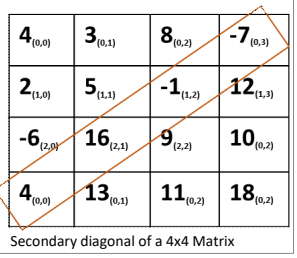
The main diagonal of a square matrix are the elements who's row number and column number are equal, a_{ii}

```
def sum_primary_diagonal(m):
  row,col = m.shape
  assert (row==col), 'Not a square matrix'
  sum = 0
  for i in range(row):
    sum += m[i][i]
  return sum
```

Add the elements of the secondary diagonal in a square matrix



Secondary diagonal of a 3x3 Matrix



Secondary diagonal of a 4x4 Matrix

For an element a_{ij} in the secondary diagonal, can you find out the j for a particular i?

Hint: try i+j for a_{ij} in the secondary diagonal and find out the relation.

Add two matrices of same dimension

```
def add_matrix(m,n):
  r_m, c_m = m.shape
  r_n, c_n = n.shape
  assert (r_m == r_n and c_m == c_n), 'Dimension mismatch'
  result = np.zeros((r_m,c_m), dtype=int)
  for i in range(r_m):
    for j in range(c_m):
      result[i][j] = m[i][j]+n[i][j]
  return result
```

Multiply two matrices

```python
def mulitply(m,n):
  r_m, c_m = m.shape
  r_n, c_n = n.shape
  assert c_m == r_n, 'Cannot multiply'
  result = np.zeros((r_m,c_n), dtype=int)
  for i in range(r_m):
    for j in range(c_n):
      for k in range(c_m):
        result[i][j] += m[i][k]*n[k][j]
  return result
```