Data structure in python

Searching Technique

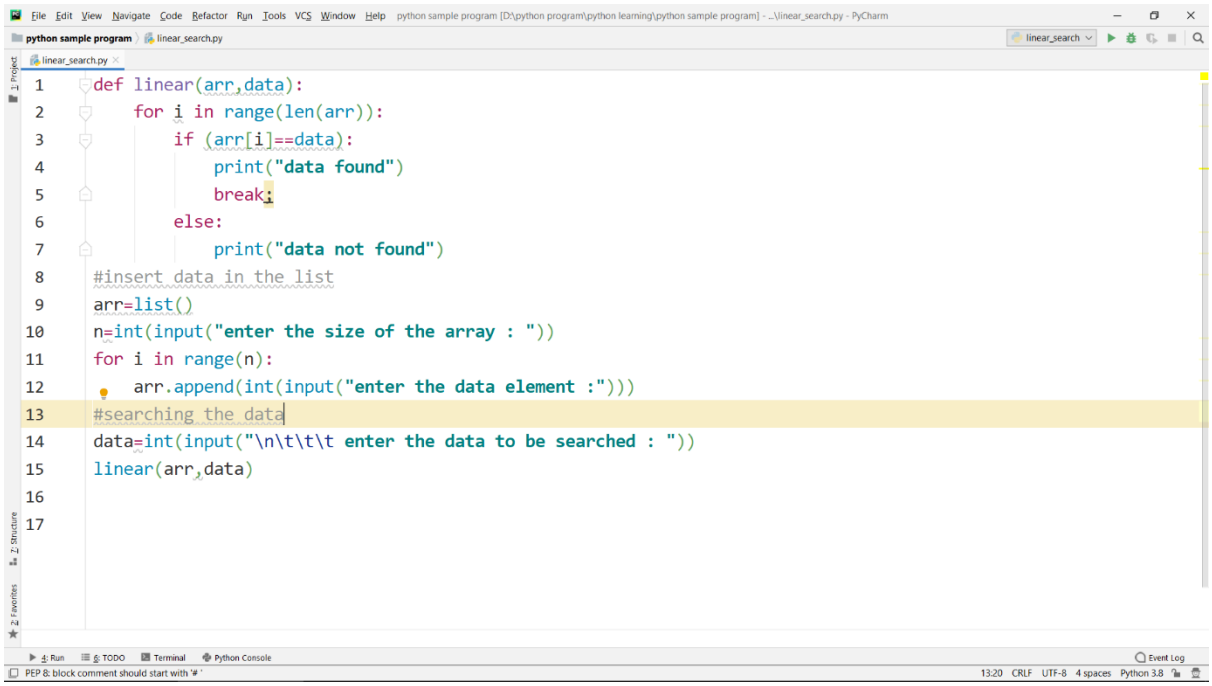
Linear search:

it is a process of searching the data in an array for a particular value. It works by comparing the value to be searched with every element of the array one by one until match is found.

Time complexity:

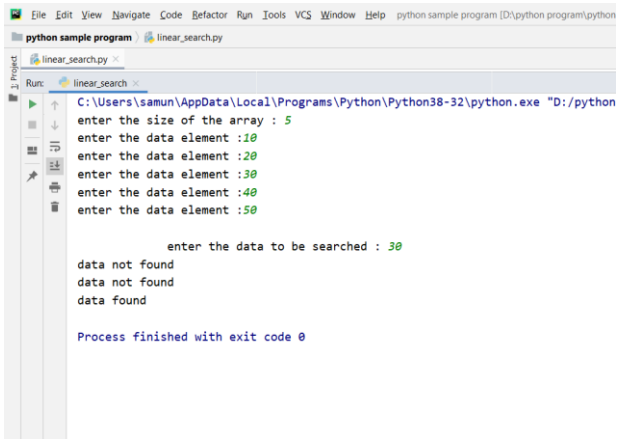
Case	Best case	Worst case	Average case
If item is present	**1**	**n**	**n/2**
If item is not present	**n**	**n**	**n**

Program to implement linear search:



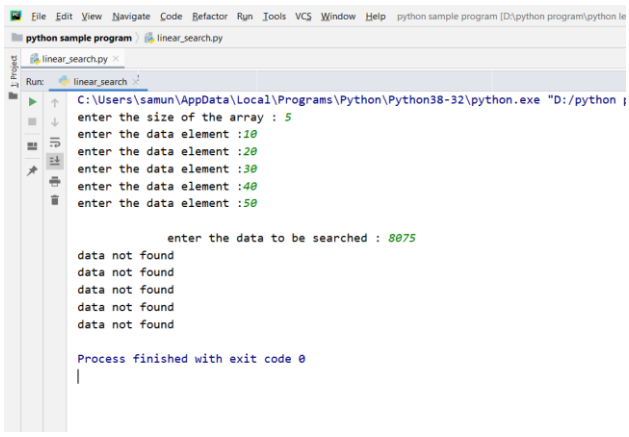
```
def linear(arr,data):
    for i in range(len(arr)):
        if (arr[i]==data):
            print("data found")
            break;
        else:
            print("data not found")
#insert data in the list
arr=list()
n=int(input("enter the size of the array : "))
for i in range(n):
    arr.append(int(input("enter the data element :")))
#searching the data
data=int(input("\n\t\t\t enter the data to be searched : "))
linear(arr,data)
```

When data is found then no need to execute the whole loop



```
C:\Users\samun\AppData\Local\Programs\Python\Python38-32\python.exe "D:/python
enter the size of the array : 5
enter the data element :10
enter the data element :20
enter the data element :30
enter the data element :40
enter the data element :50

            enter the data to be searched : 30
data not found
data not found
data found

Process finished with exit code 0
```

When data is not in array although have to execute the whole loop



```
C:\Users\samun\AppData\Local\Programs\Python\Python38-32\python.exe "D:/python
enter the size of the array : 5
enter the data element :10
enter the data element :20
enter the data element :30
enter the data element :40
enter the data element :50

            enter the data to be searched : 8075
data not found
data not found
data not found
data not found
data not found

Process finished with exit code 0
```

Binary search:

- To initiate binary search first we have to arrange the data in an ordered manner (either increasing or decreasing manner).
- This search starts from the middle term of array.

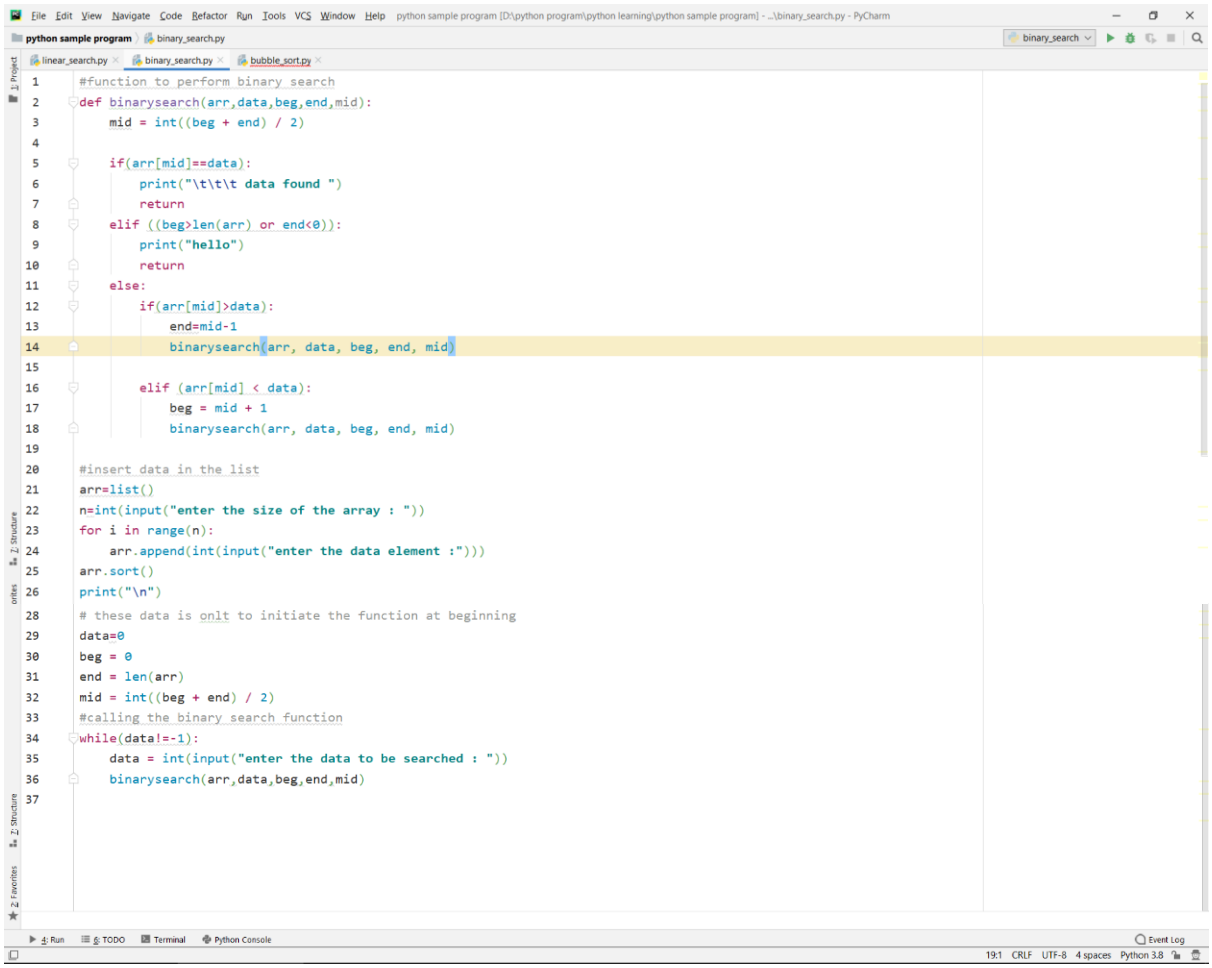
Logic:

1. If the **arr[middle term] == target value** then it simply returns true.
2. If our **target_item >arr[middle item]** the we have to search it in the right section and forget the left section. By doing so once we know that the data in the right section of array then we have to change the value of beginning and middle, end will remain the same.
   beg= mid+1 and mid= (beg +end)/2
3. If our **target_item<arr[middle item]** then we have to search in the left section and forget the left section. By doing so once we know that the data in the right section of array then we have to change the value of end and middle, beg will remain the same.
   end= mid-1 and mid= (beg +end)/2

Time complexity:

Case	Best case	Worst case	Average case
If item is present	**1**	**O(log n)**	**O(log n)**
If item is not present	**O(log n)**	**O(log n)**	**O(log n)**

Program to implement the binary search operation:



```
#function to perform binary search
def binarysearch(arr,data,beg,end,mid):
    mid = int((beg + end) / 2)

    if(arr[mid]==data):
        print("\t\t\t data found ")
        return
    elif ((beg>len(arr) or end<0)):
        print("hello")
        return
    else:
        if(arr[mid]>data):
            end=mid-1
            binarysearch(arr, data, beg, end, mid)

        elif (arr[mid] < data):
            beg = mid + 1
            binarysearch(arr, data, beg, end, mid)

#insert data in the list
arr=list()
n=int(input("enter the size of the array : "))
for i in range(n):
    arr.append(int(input("enter the data element :")))
arr.sort()
print("\n")
# these data is onlt to initiate the function at beginning
data=0
beg = 0
end = len(arr)
mid = int((beg + end) / 2)
#calling the binary search function
while(data!=-1):
    data = int(input("enter the data to be searched : "))
    binarysearch(arr,data,beg,end,mid)
```

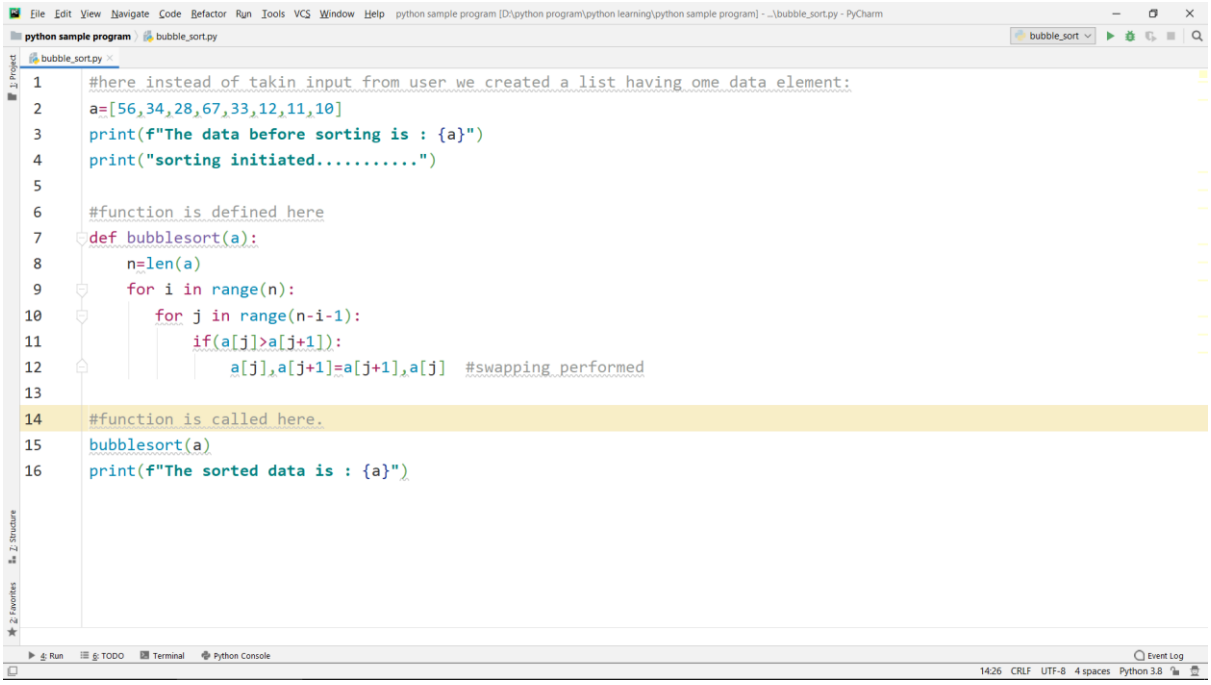
Sorting Technique

1. Bubble Sort:

- In bubble sort consecutive adjacent pair of elements in the array are compared with each other using loops (only two loops are required to perform bubble sort).
- It simply put one element from start of array and make comparison to all the element of the array if it is largest from all element then it places it to the last position (for ascending order).
- It means that during first pass the largest value is placed on the last index of array and in second pass the second largest value place in the second largest index of array and this process continues until all the element is not sorted.
- Now why we call it bubble sort? Because the highest value raises like a bubble from array and put on its designated place.

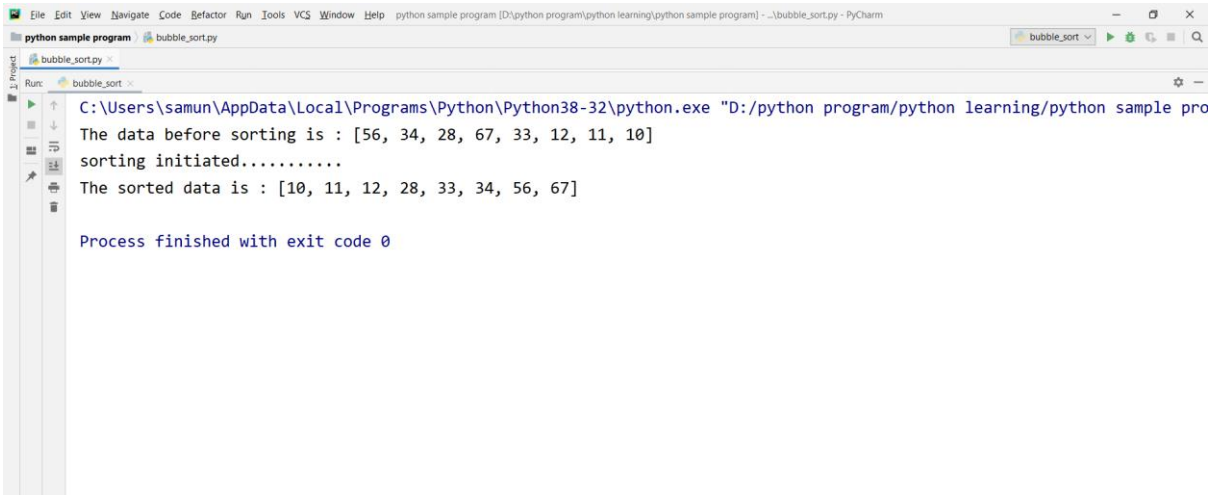
Time complexity: $O(n^2)$ for all the best, worst, average case .

Program to implement the binary search operation:



```
#here instead of takin input from user we created a list having ome data element:
a=[56,34,28,67,33,12,11,10]
print(f"The data before sorting is : {a}")
print("sorting initiated..........")

#function is defined here
def bubblesort(a):
    n=len(a)
    for i in range(n):
        for j in range(n-i-1):
            if(a[j]>a[j+1]):
                a[j],a[j+1]=a[j+1],a[j]  #swapping performed

#function is called here.
bubblesort(a)
print(f"The sorted data is : {a}")
```

Output:



```
C:\Users\samun\AppData\Local\Programs\Python\Python38-32\python.exe "D:/python program/python learning/python sample pro
The data before sorting is : [56, 34, 28, 67, 33, 12, 11, 10]
sorting initiated..........
The sorted data is : [10, 11, 12, 28, 33, 34, 56, 67]

Process finished with exit code 0
```

For better understanding we take an example and see the flow of execution:

Let us consider an (unsorted) array arr = [8, 5, 3, 1, 2, 6] here n=6(size of array)

Pass 1: **our array is arr = [8, 5, 3, 1, 2, 6]**

i = 0 and (j =0 ; j < n-i-1; j++) it means that loop in j will execute up to j <n – i -1➔ j < 5

j=0	arr[0] > arr[1]	true	swapping done	[**5**,**8**, 3, 1, 2, 6]
j=1	arr[1] > arr[2]	true	swapping done	[5 ,**3**,**8**, 1, 2, 6]
j=2	arr[2] > arr[3]	true	swapping done	[5 ,3 ,**1**,**8**, 2, 6]
j=3	arr[3] > arr[4]	true	swapping done	[5 ,3 , 1,**2**,**8**, 6]
j=4	arr[1] > arr[2]	true	swapping done	[5 ,3 , 1, 2,**6**,**8**]

in first pass the highest element is placed at the last index of array.

Pass 2: **our array is arr = [5 ,3 , 1, 2, 6, 8]**

i = 1 and (j =0 ; j < n-i-1; j++) it means that loop in j will execute up to j <n – i -1➔ j < 4

j=0	arr[0] > arr[1]	true	swapping done	[**3**,**5**, 1, 2, 6, 8]
j=1	arr[1] > arr[2]	true	swapping done	[3,**1**,**5**, 2, 6, 8]
j=2	arr[2] > arr[3]	true	swapping done	[3, 1,**2**,**5**, 6, 8]
j=3	arr[3] > arr[4]	false	NOTHING	

in second pass the second highest element is placed at the second largest index of array,

Pass 3: **our array is arr = [3, 1, 2, 5, 6, 8]**

i = 2 and (j =0 ; j < n-i-1; j++) it means that loop in j will execute up to j <n – i -1➔ j < 3

j=0	arr[0] > arr[1]	true	swapping done	[**1**,**3**, 2, 5, 6, 8]
j=1	arr[1] > arr[2]	true	swapping done	[1,**2**,**3**, 5, 6, 8]
j=2	arr[2] > arr[3]	false	NOTHING	

in third pass the third highest element is placed at the third largest index of array,

Now although the array is sorted but still the loop will executed that is the big demerit of bubble sort

Pass 4: **our array is arr = [1, 2, 3, 5, 6, 8]**

i = 3 and (j =0 ; j < n-i-1; j++) it means that loop in j will execute up to j <n – i -1➔ j < 2

j=0	arr[0] > arr[1]	false	NOTHING
j=1	arr[1] > arr[2]	false	NOTHING

Pass 5: **our array is arr = [1, 2, 3, 5, 6, 8]**

i = 4 and (j =0 ; j < n-i-1; j++) it means that loop in j will execute up to j <n – i -1➔ j < 1

j=0	arr[0] > arr[1]	false	NOTHING

Sorted data

Pass 5: **our array is arr = [1, 2, 3, 5, 6, 8]**

i = 5 and (j =0 ; j < n-i-1; j++) it means that loop in j will execute up to j <n – i -1➔ j < 0

further no execution

2. Insertion Sort:

- In insertion sort simply one data is picked up and at appropriate place it is inserted
- The array is divided into two set sorted and unsorted data , then we select a data and traversed the sorted set to insert it.
- As we are moving from left to right one element has picked and then a appropriate place is searched after that we insert the data.
- There is no need to search the whole element only we have to searched from the sorted set due to which the time complexity quite reduced.

Time complexity:	
Best case : O (n)	Average case & worst case :O (n^2)

- When the array is already sorted then there is best case occur and when the data is in reverse order then the worst case occur.

Program to implement Insertion sort

```
#here instead of takin input from user we created a list having ome data element:
a=[56,34,28,67,33,12,11,10]
print(f"The data before sorting is : {a}")
print("sorting initiated..........")

#function is defined here
def insertionsort(a):
    n=len(a)
    for i in range(1,n):
        j = i-1
        temp=a[i]
        while(temp<a[j] and j>=0):
            a[j+1]=a[j]
            j=j-1
        a[j+1]=temp

#function is called here.
insertionsort(a)
print(f"The sorted data is : {a}")
```

output:

```
C:\Users\samun\AppData\Local\Programs\Python\Python38-32\python.exe "D:/python program/python learning/python sample pro
The data before sorting is : [56, 34, 28, 67, 33, 12, 11, 10]
sorting initiated..........

The sorted data is : [10, 11, 12, 28, 33, 34, 56, 67]

Process finished with exit code 0
```

For better understanding we take an example and see the flow of execution:

Let us consider an (unsorted) array arr = [3, 8, 5, 2, 4, 1] here n=6(size of array)

Pass 1: **our array is arr = [3, 8, 5, 2, 4, 1]**

i = 1 temp=arr[1] = 8 j = i - 1= 0

j=0	temp<arr[0](8<0)	false	NOTHING	[3 , 8, 5, 2, 4, 1]

Pass 2: **our array is arr = [3, 8, 5, 2, 4, 1]**

i = 2 temp=arr[2] = 5 j = 2 - 1= 1

j=1	temp<arr[1](5<8)	true	data picked and shift	[3 , b,8 , 2, 4, 1]
j=0	temp<arr[0](5<3)	false	data inserted	[3, 5, 8, 2, 4, 1]

Pass 3: **our array is arr = [3, 5, 8, 2, 4, 1]**

i = 3 temp=arr[3] =2 j = 3 - 1= 2

j=2	temp<arr[2](2<8)	true	data picked and shift	[3 ,5 ,b ,8 , 4, 1]
j=1	temp<arr[1](2<5)	true	space shift	[3 , b, 5 , 8, 4, 1]
j=0	temp<arr[0](2<3)	true	space shift	[b , 3, 5, 8, 4, 1]
j=-1	while loop failed (j<=0)		data inserted	[2, 3, 5, 8, 4, 1]

Pass 4: **our array is arr = [2, 3, 5, 8, 4, 1]**

i = 4 temp=arr[4] =4 j = 4 - 1= 3

j=3	temp<arr[3](4<8)	true	data picked	[2, 3, 5, b , 8, 1]
j=2	temp<arr[2](4<5)	true	space shift	[2 ,3 ,b ,5, 8, 1]
j=1	temp<arr[1](4<3)	false	data inserted	[2 ,3 ,4 ,5, 8, 1]

Pass 4: **our array is arr = [2 ,3 ,4 ,5, 8, 1]**

i = 5 temp=arr[5] =1 j = 5 - 1= 4

j=4	temp<arr[4](1<8)	true	data picked and shift	[2 ,3 ,4 ,5, b, 8]
j=3	temp<arr[3](1<5)	true	space shift	[2 ,3 ,4 ,b, 5, 8]
j=2	temp<arr[2](1<4)	true	space shift	[2 ,3 ,b ,4, 5, 8]
j=1	temp<arr[1](1<3)	true	space shift	[2 ,b ,3 ,4, 5, 8]
j=0	temp<arr[0](1<2)	true	space shift	[b ,2 ,3 ,4, 5, 8]
j=-1	while loop failed (j<=0)		data inserted	[1 ,2 ,3 ,4, 5, 8]

for loop ended and data is sorted…….

Sorted data

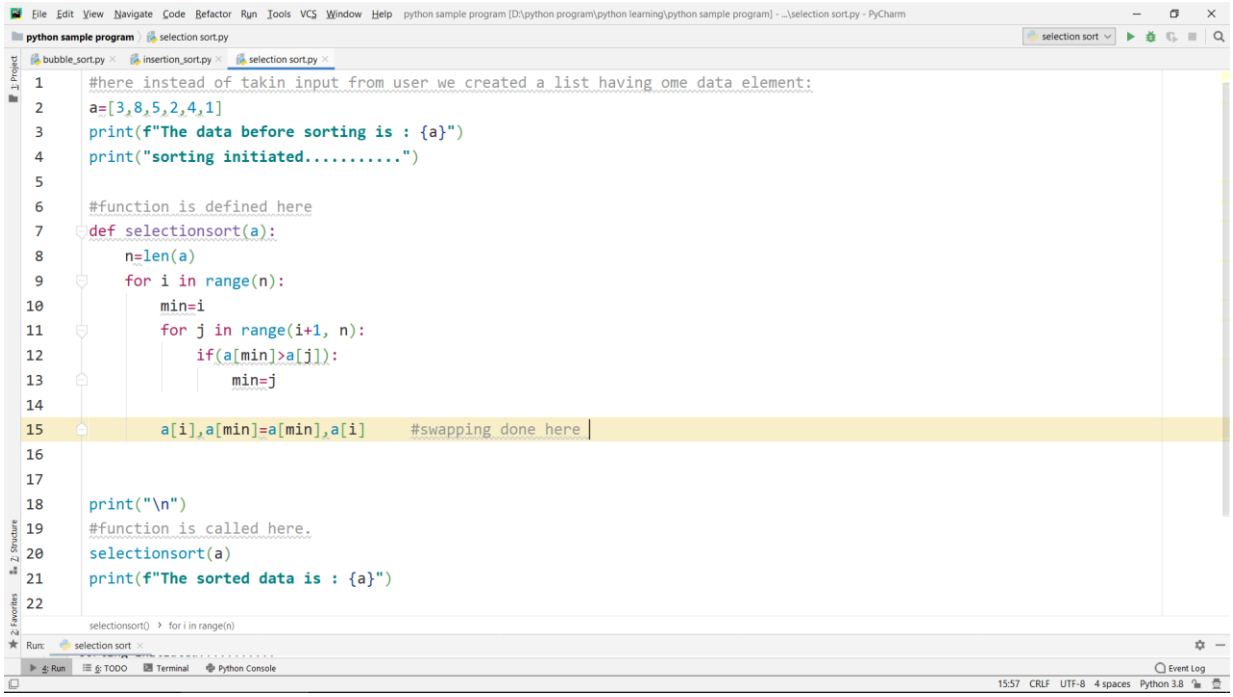
arr = [1 ,2 ,3 ,4, 5, 8]

3. Selection Sort:

- In selection sort again we create two set i.e.: sorted and unsorted set of the array
- First we traverse the array and find the minimum element (for ascending order).
- For traversing we assume that the first element of the unsorted set is small and using linear search we find that whether there is any small element is present or not.
- If there is small element is found then that element is swapped with the small element
- After swapping the sorted set is updated having that element and the unsorted list start with the next element, this process continues until the last element get sorted.
- Why we call it section sort? because it selects a element from the unsorted set compare it to the other element for finding the smallest one and then it swapped it.

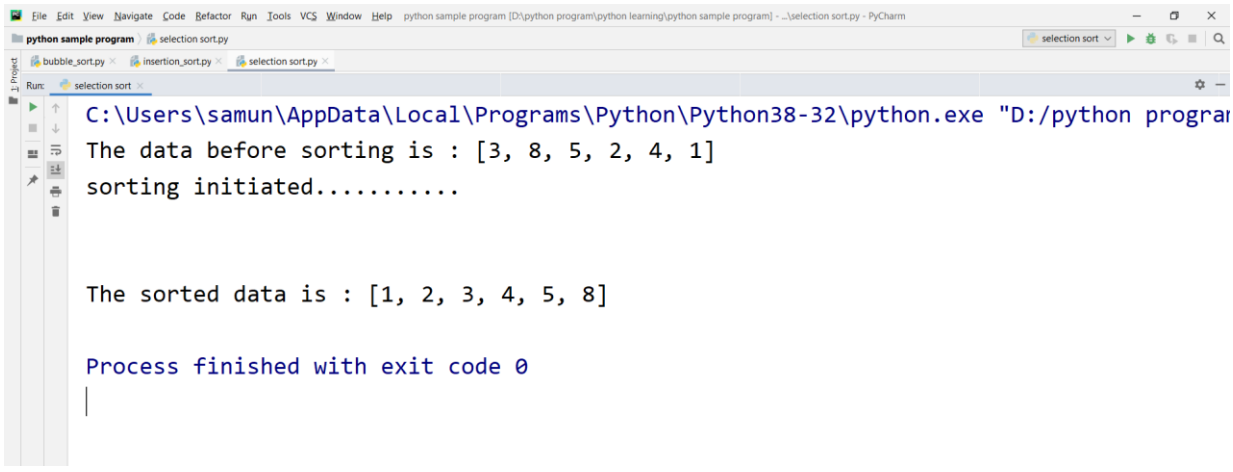
Time complexity:
For Best case, worst case, average case $O(n^2)$

Program to implement the selection sort



```
#here instead of takin input from user we created a list having ome data element:
a=[3,8,5,2,4,1]
print(f"The data before sorting is : {a}")
print("sorting initiated.....")

#function is defined here
def selectionsort(a):
    n=len(a)
    for i in range(n):
        min=i
        for j in range(i+1, n):
            if(a[min]>a[j]):
                min=j

        a[i],a[min]=a[min],a[i]     #swapping done here


print("\n")
#function is called here.
selectionsort(a)
print(f"The sorted data is : {a}")
```

Output:

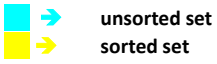


```
C:\Users\samun\AppData\Local\Programs\Python\Python38-32\python.exe "D:/python prograr
The data before sorting is : [3, 8, 5, 2, 4, 1]
sorting initiated.....


The sorted data is : [1, 2, 3, 4, 5, 8]

Process finished with exit code 0
```

For better understanding we take an example and see the flow of execution:

Let us consider an (unsorted) array arr = [3, 8, 5, 2, 4, 1] here n=6(size of array)

Pass 1: **our array is arr = [3, 8, 5, 2, 4, 1]**



i = 0 min = i and j = i + 1 hence j starts with '1'

min = 0	j =1	arr[min] > arr[j] (3>8)	false	no change in min
min = 0	j =2	arr[min] > arr[j] (3>5)	false	no change in min
min = 0	j =3	arr[min] > arr[j] (3>2)	true	change min = 3
min = 3	j =4	arr[min] > arr[j] (2>4)	false	no change in min
min = 3	j =5	arr[min] > arr[j] (2>1)	true	change min = 5

loop in j ends and we get the min value at place 5 of this array

swap arr[i] with arr[min] such that arr[0] with arr[5] now **arr = [1 , 8, 5, 2, 4, 3]**

Pass 2: **our array is arr = [1 , 8, 5, 2, 4, 3]**

i = 1 min = i and j = i + 1 hence j starts with '2'

min = 1	j =2	arr[min] > arr[j] (8>5)	true	change min = 2
min = 2	j =3	arr[min] > arr[j] (5>2)	true	change min = 3
min = 3	j =4	arr[min] > arr[j] (2>4)	false	no change in min
min = 3	j =5	arr[min] > arr[j] (2>1)	false	no change in min

loop in j ends and we get the min value at place 3 of this array

swap arr[i] with arr[min] such that arr[1] with arr[3] now **arr = [1 , 2, 5, 8, 4, 3]**

Pass 3: **our array is arr = [1 , 2, 5, 8, 4, 3]**

i = 2 min = i and j = i + 1 hence j starts with '3'

min = 2	j =3	arr[min] > arr[j] (5>8)	false	no change in min
min = 2	j =4	arr[min] > arr[j] (5>4)	true	change min = 4
min = 4	j =5	arr[min] > arr[j] (4>3)	true	change min = 5

loop in j ends and we get the min value at place 5 of this array

swap arr[i] with arr[min] such that arr[2] with arr[5] now **arr = [1 , 2, 3, 8, 4, 5]**

Pass 4: **our array is arr = [1 , 2, 3, 8, 4, 5]**

i = 3 min = i and j = i + 1 hence j starts with '4'

min = 3	j =4	arr[min] > arr[j] (8>4)	true	change min = 4
min = 4	j =5	arr[min] > arr[j] (4>5)	false	no change in min

loop in j ends and we get the min value at place 4 of this array

swap arr[i] with arr[min] such that arr[3] with arr[4] now **arr = [1 , 2, 3, 4, 8, 5]**

Pass 4: **our array is arr = [1 , 2, 3, 4, 8, 5]**

i = 4 min = i and j = i + 1 hence j starts with '5'

min = 4	j =5	arr[min] > arr[j] (8>5)	true	change min = 5

loop in j ends and we get the min value at place 5 of this array

swap arr[i] with arr[min] such that arr[4] with arr[5] now **arr = [1 , 2, 3, 4, 5, 8]**

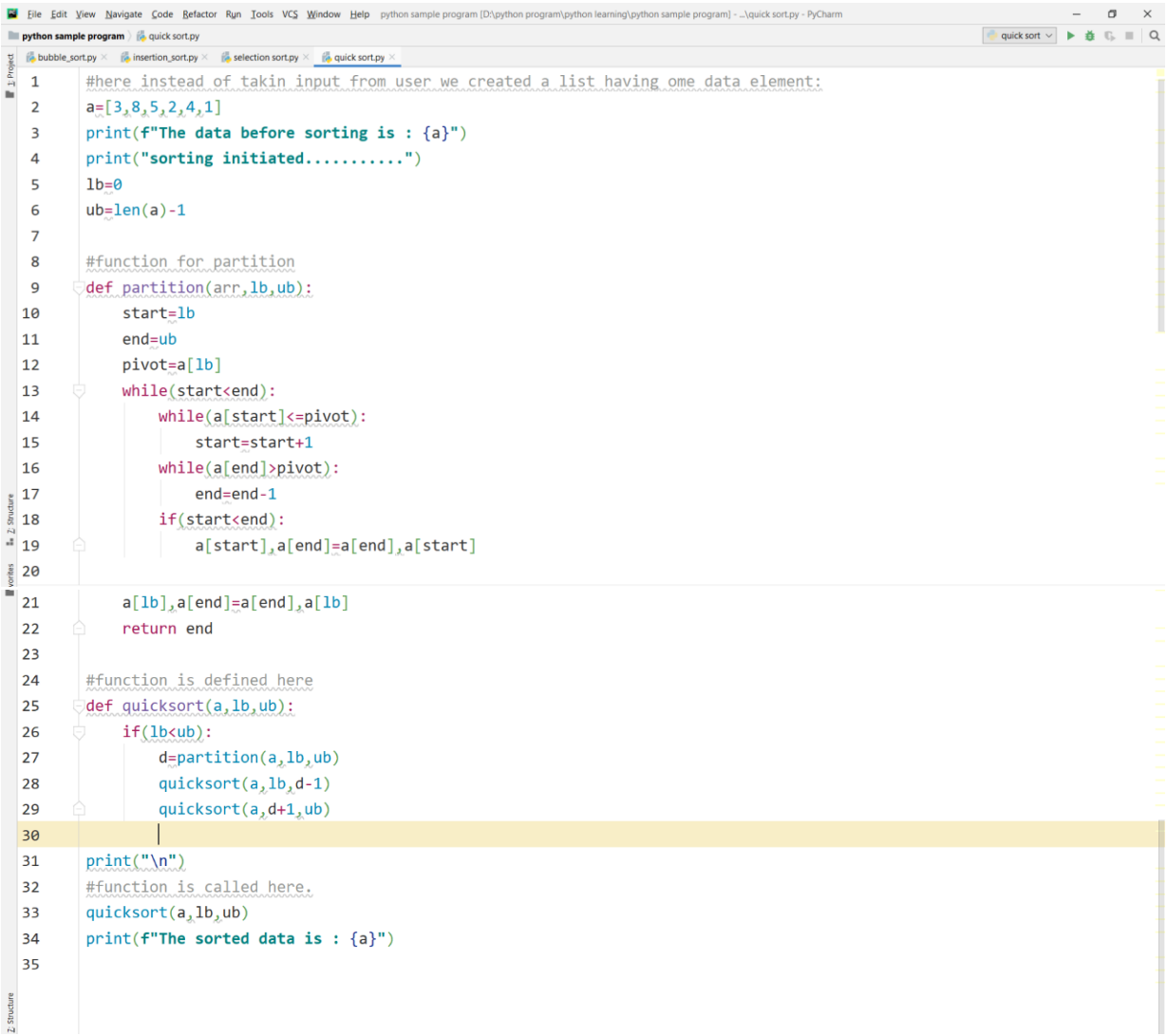
finally the i loop exits with having the array **arr = [1 , 2, 3, 4, 5, 8]** which is sorted.

4. Quick Sort:

- It is based on divide and conquer mechanism.
- First, we have to set the upper bound i.e. 0 and lower bound i.e. size_of_array - 1 and choose a pivot point any of your choice (first, mid or last element). here I choose the first element as my pivot point
- After that we have create a **partition function which is backbone** of this sorting technique. In partition function we can arrange our data in such a manner that the left side of pivot is smaller data and right side of pivot is greater, this function return that value of that index from where we have to divide our array.
- The index returned from the partition function is feed into the quicksort with some logic for recursive call of quicksort
- The quicksort function recursively call itself until each element became single. The *base case* of the recursion occurs when the array has zero or one element because in that case the array is already sorted.

Time complexity:	
Best case ,Average case : *O* (n logn)	**Worst case :*O* (n^2)**

Program to implement the Quick sort



```
#here instead of takin input from user we created a list having ome data element:
a=[3,8,5,2,4,1]
print(f"The data before sorting is : {a}")
print("sorting initiated.....")
lb=0
ub=len(a)-1

#function for partition
def partition(arr,lb,ub):
    start=lb
    end=ub
    pivot=a[lb]
    while(start<end):
        while(a[start]<=pivot):
            start=start+1
        while(a[end]>pivot):
            end=end-1
        if(start<end):
            a[start],a[end]=a[end],a[start]

    a[lb],a[end]=a[end],a[lb]
    return end

#function is defined here
def quicksort(a,lb,ub):
    if(lb<ub):
        d=partition(a,lb,ub)
        quicksort(a,lb,d-1)
        quicksort(a,d+1,ub)

print("\n")
#function is called here.
quicksort(a,lb,ub)
print(f"The sorted data is : {a}")
```

For better understanding we take an example and see the flow of execution:

Let us consider an (unsorted) array arr = [3, 8, 5, 2, 4, 1] here n=6(size of array)

Here we consider:

start = lb = 0 end = ub(size–1) pivot = a[lb](here I am assuming the first element as pivot)

Pass 1: our array is a = [3, 8, 5, 2, 4, 1] start = 0 end = 5 pivot = 3

Main while (start < end) (0<5) true

Inside while (a[start]<=pivot)

Start = 0	a[0]<=pivot (3<=3)	true	start ++
Start = 1	a[1]<=pivot(8<=3)	false	loop exits

Inside while(a[end]>pivot)

end = 5	a[5]>pivot(1>3)	false	loop exits

if(start<end) i.e. (1<5) true (swap data of start with end)

Swap(a[start] with a[end]) i.e. swap a[1] with a[5]

NOW [3, 1, 5, 2, 4, 8]

start end

Pass 2: our array is a = [3, 1, 5, 2, 4, 8] start = 1 end = 5 pivot = 3

Main while (start < end) (1<5) true

Inside while (a[start]<=pivot)

Start = 1	a[1]<=pivot (1<=3)	true	start++
Start = 2	a[2]<=pivot(5<=3)	false	loop exits

Inside while(a[end]>pivot)

end = 5	a[5]>pivot(8>3)	true	end –
end = 4	a[4]>pivot(4>3)	true	end –
end = 3	a[3]>pivot(2>3)	false	loop exits

if(start<end) i.e. (2<3) true (swap data of start with end)

Swap(a[start] with a[end]) i.e. swap a[2] with a[3]

NOW [3, 1, 2, 5, 4, 8]

start end

Pass 3: our array is a = [3, 1, 2, 5, 4, 8] start = 2 end = 3 pivot = 3

Main while (start < end) (1<5) true

Inside while (a[start]<=pivot)

Start = 2	a[2]<=pivot (2<=3)	true	start++
Start = 3	a[3]<=pivot(5<=3)	false	loop exits

Inside while(a[end]>pivot)

end = 3	a[3]>pivot(5>3)	true	end –
end = 2	a[2]>pivot(2>3)	false	loop exits

if(start<end) i.e. (3<2) false (No swapping performed)

Pass 4: **our array is a = [3, 1, 2, 5, 4, 8]** **start = 3** **end = 2** **pivot = 3**

Main while (start < end) (3 < 2) False

Now the pivot is swapped with the end and function exits returning the value of the end
Swap(a[lb] with a[end]) i.e. swap(a[0] with a[2])

NOW [2, 1, 3, 5, 4, 8]
Return 2(end value)

a = [2, 1] a = [5, 4, 8]

for sub array a=[2,1] the end point modify end = return value from partition fun -1

Here we consider:
start = lb = 0 end = 2 – 1 = 1 pivot = a[lb](here I am assuming the first element as pivot)

Pass 5: **our array is a = [2, 1]** **start = 0** **end = 1** **pivot = 2**

Main while (start < end) (0<1) true
Inside while (a[start]<=pivot)
Start = 0 a[0]<=pivot (2<=2) true start ++
Start = 1 a[1]<=pivot(1<=2) false loop exits
Inside while(a[end]>pivot)
end = 1 a[1]>pivot(1>2) false loop exits

if(start<end) i.e. (1<1) false (No swapping performed)

Pass 6: **our array is a = [2, 1]** **start = 1** **end = 1** **pivot = 2**

Main while (start < end) (1 < 1) False

Now the pivot is swapped with the end and function exits returning the value of the end
Swap(a[lb] with a[end]) i.e. swap(a[0] with a[1])

NOW [1, 2]
Return 1(end value)

a = [1] a[2]
(since in quick sort function ub == lb hence recursion stops)

Similarly for sub array a=[5, 4, 8] the start point modified **start = return value +1**

Here we consider:

start = lb = 2 + 1 = 3 end = 0 pivot = a[lb] (here I am assuming the first element as pivot)

Pass 7: **our array is a = [5, 4, 8]** **start = 3** **end = 5** **pivot = 5**

Main while (start < end) (3<5) true

Inside while (a[start]<=pivot)

Start = 3	a[3]<=pivot (5<=5)	true	start ++
Start = 4	a[4]<=pivot(4<=5)	true	start ++
Start = 5	a[5]<=pivot(8<=5)	false	loop exits

Inside while(a[end]>pivot)

end = 5	a[5]>pivot(8>5)	true	end --
end = 4	a[4]>pivot(4>5)	false	loop exits

if(start<end) i.e. (5<4) false (No swapping performed)

Pass 8: **our array is a = [5, 4, 8]** **start = 5** **end = 4** **pivot = 2**

Main while (start < end) (5 < 4) False

Now the pivot is swapped with the end and function exits returning the value of the end

Swap(a[lb] with a[end]) i.e. swap(a[3] with a[4])

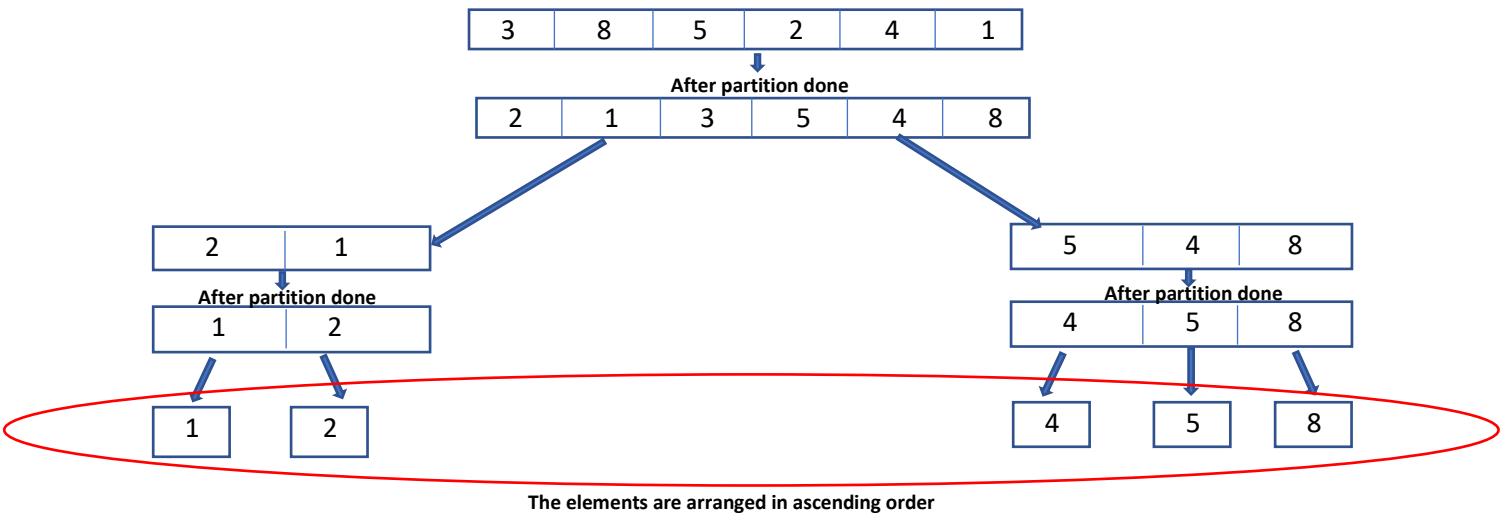
NOW [4, 5 ,8]

Return 4 (end value)

a[4] a[8]

(since in quick sort function ub == lb hence recursion stops)

Generalising the result from the recursion:



The elements are arranged in ascending order

5.Merge Sort:

- It is base on divide and conquer method to sort the data.
- It take a data from the left sub array compare it with the right sub array the data which is smaller is put in another array
- Here the merge function will be the backbone of this sorting whose work is to compare the sub array and store appropriate data in the temporary array.
- After the sorting is completed then we copy the data of temporary array to the original array.

Time complexity:
Best case ,Average case, Worst case : *O* (n logn)

Program to implement the Merge sort

```
def mergesort(a,lb,ub):
    if(lb<ub):
        mid=int((ub+lb)/2)
        mergesort(a,lb,mid)
        mergesort(a,mid+1,ub)
        merge(a, lb, mid, ub)

def merge(a,lb,mid,ub):
    i=lb
    j=mid+1
    k=lb

    while(i<=mid and j<=ub):
        if(a[i]<=a[j]):
            b[k]=a[i]
            i=i+1
        else:
            b[k]=a[j]
            j=j+1
        k=k+1
    if(i>mid):
        while(j<=ub):
            b[k] = a[j]
            j=j+1
            k=k+1
    else:
        while(i<=mid):
            b[k]=a[i]
            k = k+1
            i = i+1
    for i in range (lb,k):
        a[i]=b[i]


#here instead of takin input from user we created a list having ome data element:
a=[3,8,5,2,4,1]
b = [0] * 6     #this will fix the size of the array ans set the default value 0 in it
print(f"The data before sorting is : {a}")
print("sorting initiated.....")
lb=0
ub=len(a)-1
print("\n")
#function is called here
mergesort(a,lb,ub)
print(f"The sorted data is : {a}")
```

Output:

```
C:\Users\samun\AppData\Local\Programs\Python\Python3
The data before sorting is : [3, 8, 5, 2, 4, 1]
sorting initiated.....


The sorted data is : [1, 2, 3, 4, 5, 8]
```

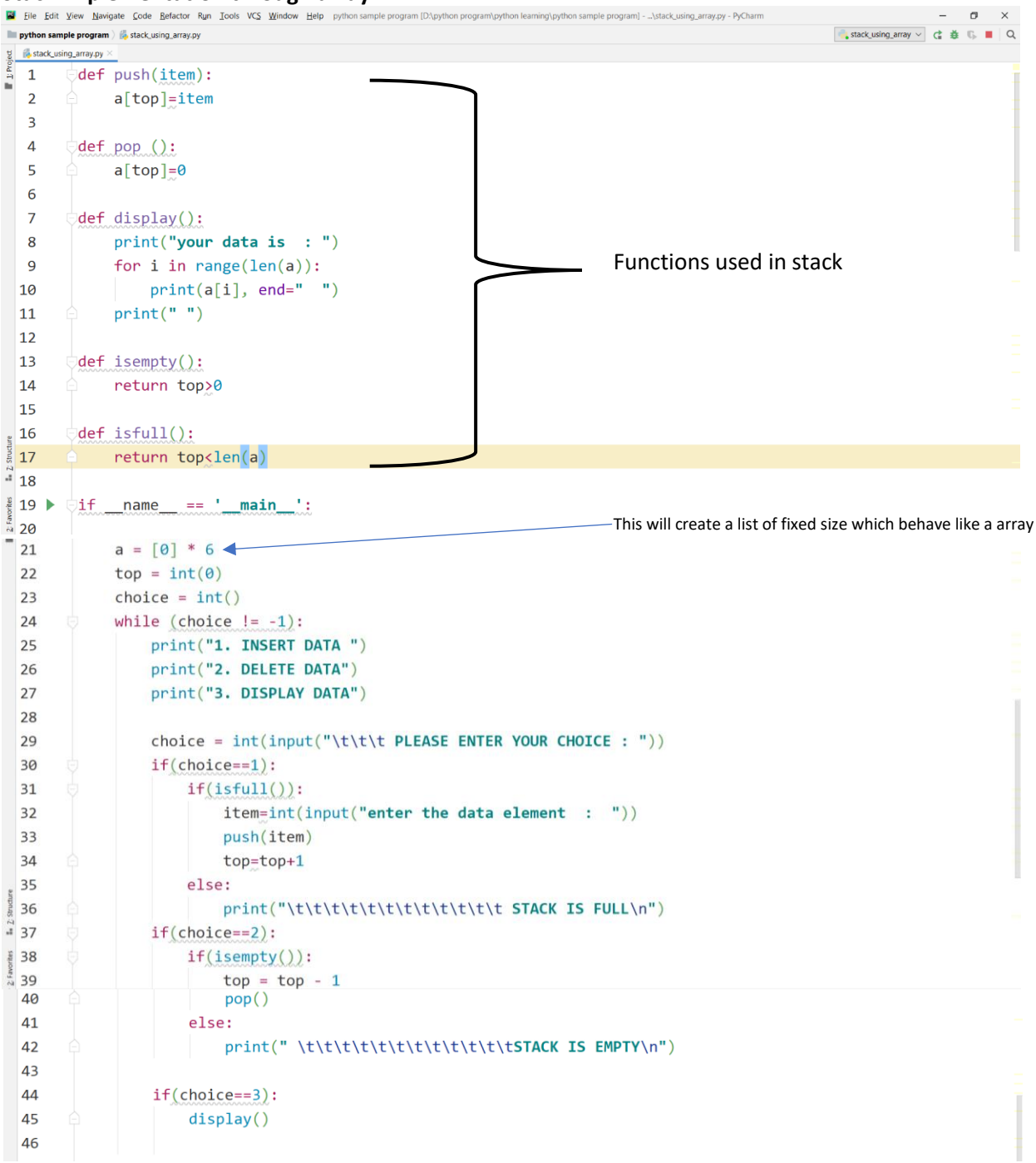
Data structure

1. Stack

Stack is a linear data structure that work in LIFO (last in first out) mechanism.
It has two major function:

- **Push:** To insert data in the last of list
- **Pop:** To remove data from the first position

Stack implementation through array:



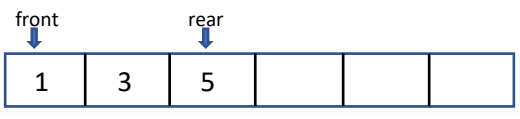

```
1   def push(item):
2       a[top]=item
3
4   def pop ():
5       a[top]=0
6
7   def display():
8       print("your data is  : ")
9       for i in range(len(a)):
10          print(a[i], end="  ")
11      print(" ")
12
13  def isempty():
14      return top>0
15
16  def isfull():
17      return top<len(a)
18
19  if __name__ == '__main__':
20
21      a = [0] * 6
22      top = int(0)
23      choice = int()
24      while (choice != -1):
25          print("1. INSERT DATA ")
26          print("2. DELETE DATA")
27          print("3. DISPLAY DATA")
28
29          choice = int(input("\t\t\t PLEASE ENTER YOUR CHOICE : "))
30          if(choice==1):
31              if(isfull()):
32                  item=int(input("enter the data element  :  "))
33                  push(item)
34                  top=top+1
35              else:
36                  print("\t\t\t\t\t\t\t\t\t\t\t\t STACK IS FULL\n")
37          if(choice==2):
38              if(isempty()):
39                  top = top - 1
40                  pop()
41              else:
42                  print(" \t\t\t\t\t\t\t\t\t\t\t\tSTACK IS EMPTY\n")
43
44          if(choice==3):
45              display()
46
```

2. Queue

Queue is a linear data structure that work in FIFO (first in first out) mechanism. The data is inserted at one end called rear and the data that is deleted at other end called front.

Queue implementation through array:

```
def enqueue(item):
    a[rear]=item
    display()

def  dequeue():
   a[front]=0
   display()

def display():
    print("your data is  : ")
    for i in range(len(a)):
        print(a[i], end="  ")
    print(" ")

def isempty():
    return (front==0 or front<rear)

def isfull():
    return (rear<len(a)-1)

if __name__ == '__main__':

    a = [0] * 6
    rear = -1
    front=-1
    choice = int()
    while (choice != -1):
        print("1. INSERT DATA ")
        print("2. DELETE DATA")
        print("3. DISPLAY DATA")

        choice = int(input("\t\t\t PLEASE ENTER YOUR CHOICE : "))
        if(choice==1):
            if(isfull()):
                item=int(input("enter the data element  :  "))
                rear = rear + 1
                enqueue(item)

            else:
                print("\t\t\t\t Queue  IS FULL\n")

        if(choice==2):
            if(isempty()):
                front = front + 1
                dequeue()
            else:
                print(" \t\t\t\t\tQueue IS EMPTY\n")

        if(choice==3):
            display()
```



NOTE : When data inserted the rear is incremented and when the data is deleted the front is incremented

3. Circular Queue

The circular Queue is same as queue except one thing that when data is deleted from Queue that block cannot be used means the rear cannot be shifted in backward direction but in circular queue the rear can be shifted in back ward direction, it means that the deleted block can be reused.

Program to implement the circular Queue.

```
def enqueue(item):
    a[rear]=item
    display()

def  dequeue():
   a[front]="NULL"
   display()

def display():
    print("your data is  : ")
    for i in range(len(a)):
        print(a[i], end="  ")
    print(" ")

def isempty():
    return (front!=-1 and rear!=-1)

def isfull():
    return (front!=(rear+1)%len(a))

if __name__ == '__main__':

    a = [0] * 6
    rear = -1
    front=-1
    choice = int()
    while (choice != -1):
        print("1. INSERT DATA ")
        print("2. DELETE DATA")
        print("3. DISPLAY DATA")
        print(f"\t\t\t\t\t\t front:{front}  \t\t\t\t\t  rear :{rear}  ")

        choice = int(input("\t\t\t PLEASE ENTER YOUR CHOICE : "))
        if(choice==1):
            if(isfull()):
                item = int(input("enter the data element  :  "))
                if (front == -1 and rear == -1):
                    front = rear = 0
                    a[rear]=item
                else:
                    rear = (rear + 1)%len(a)
                    enqueue(item)

            else:
                print("\t\t\t\t Queue  IS FULL\n")

        if(choice==2):
            if(isempty()):
                dequeue()
                front = (front + 1) % len(a)
            else:
                print(" \t\t\t\t\tQueue IS EMPTY\n")

        if(choice==3):
            display()
```

Here the only difference is that we cannot increment the value of the front and rear ordinary since we have to reuse the block if there is no element in the block

Front=(front + 1)%len(a) and rear=(rear + 1)%len(a)

4. Linked List

A linked list is a collection of data that is non sequential it means that the data is not arranged in consecutive manner as in the array. The memory is allocated at the runtime not in compile time.

Structure of link list: it contain some data element and a pointer which point to the next node.

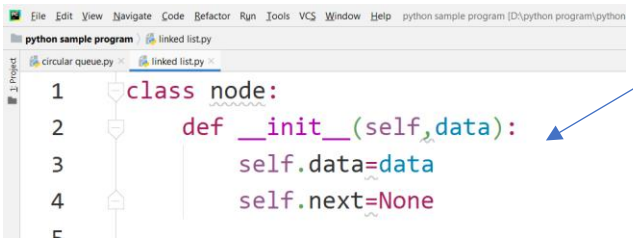
Types of linked list:

a.) Single linked list
b.) Double linked list
c.) Circular linked list

a) Single Linked List

- In a single linked list there is only one pointer that points the next node.
- We cannot traversed in the backward direction of the list.
- Here we create a dynamic memory by using class
- The logic is that when the object of any class is created then using a constructor we can allocate the memory dynamically
- Here the use of constructor is compulsory because the constructor is automatically called whenever the object of class is created.

Program to implement Single linked list



This is class having constructor which is automatically invoked whenever the object is created

```
127 if __name__ == '__main__':
128     head = None
129     while(choice!=-1):
130         print("1. INSERT DATA ")
131         print("2. DELETE DATA")
132         print("3. DISPLAY DATA")
133         choice=int(input("\t\t\t\t\t\t\t ENTER YOUR CHOICE : "))
134
135         if(choice==1):
136             h=insert(head)
137             head=h
138             display(head)
139         if (choice==2):
140             h=delete(head)
141             head=h
142             display(head)
143         if(choice==3):
144             display(head)
145
```

This is menu driven program which call the various function and important thing is that whenever we call the function it return the head value which is required for other function to work since the function may modify the head value.

Data insertion in Single linked list

```
def insert(head):
    print("**" * 80)
    print("\t\t\t\t1 . Insert data in the beginning", end="")
    print("\t\t\t\t2 . Insert data in  the end", end="")
    print("\t\t\t\t3 . Insert data at  any position")
    print("**" * 80)
    print("\n")
    ch = int(input("\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t Enter your Choice : "))
    data = int(input("Enter the data : "))
    nod = node(data)
    if(ch==1):
        nod.next = head
        head=nod
        print("DATA AT BEGINNING INSERTED")
        return head
    if(ch==2):
        p=head
        while(p.next!=None):
            p=p.next
        p.next=nod
        print("DATA AT END INSERTED")
        return head
    if(ch==3):
        p=head
        pos=int(input("ENTER THE POS : "))
        #you cannot able to insert the data at te begining by this method
        for i in range(1,pos-1 or p!=None):#bahut mehnat laga ye karne me
            if(p==None):
                print("out of index")
                break
            else:
                p=p.next
        if(p!=None):        #bahut mehnat laga ye karne me
            nod.next=p.next
            p.next=nod
            print(f"your data at {pos} position inserted sucessfully")
        return head

```

Insertion at beginning

Insertion at end

Insertion in Mid

Output:

```
C:\Users\samun\AppData\Local\Programs\Python\Python38-32\python.exe "D:/python program/python learning/python sample program/linked list.py"
1. INSERT DATA
2. DELETE DATA
3. DISPLAY DATA
                ENTER YOUR CHOICE : 1
********************************************************************************************************************************************************
                1 . Insert data in the beginning                2 . Insert data in  the end             3 . Insert data at  any position
********************************************************************************************************************************************************
                                                        Enter your Choice : 1
Enter the data : 10
DATA AT BEGINNING INSERTED
YOUR LINKED LIST IS :  10

1. INSERT DATA
2. DELETE DATA
3. DISPLAY DATA
                ENTER YOUR CHOICE : 1
********************************************************************************************************************************************************
                1 . Insert data in the beginning                2 . Insert data in  the end             3 . Insert data at  any position
********************************************************************************************************************************************************
                                                        Enter your Choice : 2
Enter the data : 20
DATA AT END INSERTED
YOUR LINKED LIST IS :  10  20

1. INSERT DATA
2. DELETE DATA
3. DISPLAY DATA
                ENTER YOUR CHOICE : 1
********************************************************************************************************************************************************
                1 . Insert data in the beginning                2 . Insert data in  the end             3 . Insert data at  any position
********************************************************************************************************************************************************
                                                        Enter your Choice : 3
Enter the data : 15
ENTER THE POS : 2
your data at 2 position inserted sucessfully
YOUR LINKED LIST IS :  10  15  20
```

Data Deletion from single linked list

```
def delete(head):
    print("**"*80)
    print("\t\t\t\t1 . delete data from the beginning",end="")
    print("\t\t\t\t2 . delete data from the end",end="")
    print("\t\t\t\t3 . delete data from the any position")
    print("**" * 80)
    ch = int(input("\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t Enter your Choice : "))
    if(ch==1):
        if(head==None):
            print("NO ELEMENT TO DELETE ")
        else:
            p=head
            head=p.next
            p=None
            print("Data deleted from Begining ")
            return head
    if (ch==2):
        if(head==None):
            print("NO ELEMENT TO DELETE ")
        else:
            p=head
            q=head
            if(p.next==None):
                print("the only",end="")
                head= None  # this will delete the single last element also   ***********
            else:
                while(p.next!=None):
                    p=p.next
                while(q.next!=p):
                    q=q.next
                q.next=None   # or you can write here  None
                p=None
            print("Data deleted from end")
            return head

    if(ch==3):
        if (head == None):
            print("NO ELEMENT TO DELETE ")
        else:
            flag=1
            p=head
            item=int(input("Enter the data item to be deleted : "))
            while(p!=None):
                if(p.data==item):
                    flag=1
                    break
                else:
                    flag=0
                q = p
                p = p.next

            if(flag==0):
                print("data not found")
                return head
            if (p.next == None):
                print("the only", end="")
                head = None              # this will delete the single last element also   ***********
                return head
            elif(p==head):              # if someone want to delete the first data then
                head=p.next
                p=None
                return head
            else:
                q.next = p.next
                p=None                          # kaam chal raha hai koi last element ko udana chahe toh
            return head
```

Deletion from beginning

Deletion from End

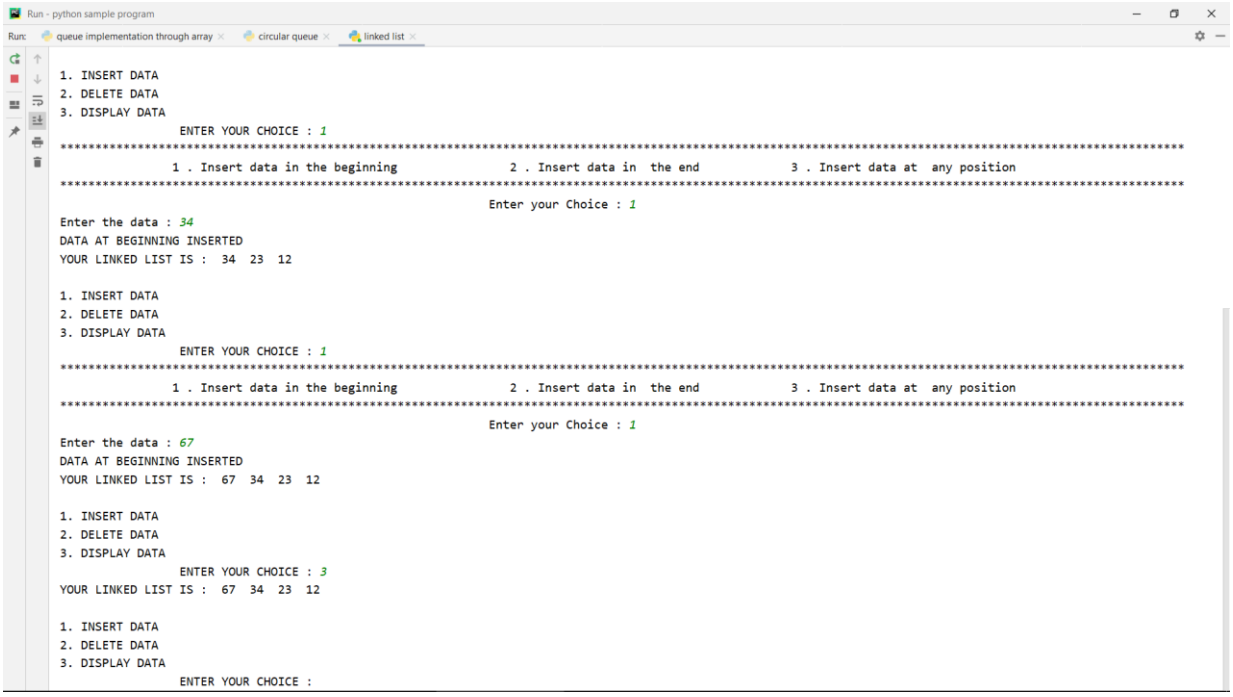
Deletion from Anywhere

Output:

Display the data of single linked list (traversing):

```
113  def display(head):
114      if(head==None):
115          print("YOUR LIST IS EMPTY")
116      else:
117          p = head
118          print("YOUR LINKED LIST IS : ",end=" ")
119          while (p != None):
120              print(p.data,end="  ")
121              p = p.next
122          print("\n")
```

Output



b.) Doubly Linked List

- In a Doubly linked list there is two pointer that points the next node and prev node.
- We can traversed in the backward direction of the list.
- The time complexity has reduced in some places since we do not traverse the whole list in some places as we are doing in case of singly linked list

Program to implement Doubly linked list :

Data Insertion in doubly linked list

```
class node:
    def __init__(self,data):
        self.data=data
        self.next=None
        self.prev=None

def insert(head):
    print("**" * 80)
    print("\t\t\t\t1 . Insert data in the beginning", end="")
    print("\t\t\t\t2 . Insert data in  the end", end="")
    print("\t\t\t\t3 . Insert data at  any position")
    print("**" * 80)
    ch = int(input("\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t Enter your Choice : "))
    data = int(input("Enter the data : "))
    nod = node(data)
    if(ch==1):
        if(head==None):
            nod.next=head
            nod.prev=None
            head = nod
        else:
            nod.next = head
            head.prev=nod
            head=nod
        print("DATA AT BEGINNING INSERTED")
        return head
    if(ch==2):
        p=head
        while(p.next!=None):
            p=p.next
        p.next=nod
        nod.prev=p
        print("DATA AT END INSERTED")
        return head
    if(ch==3):
        p=head                                  #you cannot able to insert the data at te begining  qnd at the end by this method
        pos=int(input("ENTER THE POS : "))      #but giving some condition will do this but i am skipping here to make it simple to understan
        for i in range(1,pos-1 or p!=None):#bahut mehnat laga ye karne me
            if(p==None):
                print("out of index")
                break
            else:
                p=p.next
        if(p!=None):        #bahut mehnat laga ye karne me
            nod.next=p.next
            nod.prev=p
            p.next.prev=nod
            p.next=nod
            p.next=nod
            print(f"your data at {pos} position inserted sucessfully")
        return head
```

Here one extra element is added

Output:

```
C:\Users\samun\AppData\Local\Programs\Python\Python38-32\python.exe "D:/python program/python learning/python sample program/doubly linked list.py"
1. INSERT DATA
2. DELETE DATA
3. DISPLAY DATA
                ENTER YOUR CHOICE : 1
********************************************************************************************************************************************************
            1 . Insert data in the beginning            2 . Insert data in  the end            3 . Insert data at  any position
********************************************************************************************************************************************************
                                                    Enter your Choice : 1
Enter the data : 10
DATA AT BEGINNING INSERTED
................................................
 YOUR LINKED LIST IS :  10
................................................
 BACKWARD TRAVERSING :  10
................................................
```

```
1. INSERT DATA
2. DELETE DATA
3. DISPLAY DATA
                ENTER YOUR CHOICE : 1
*****************************************************************************************************************************************
               1 . Insert data in the beginning              2 . Insert data in  the end            3 . Insert data at  any position
*****************************************************************************************************************************************
                                                        Enter your Choice : 2

Enter the data : 20
DATA AT END INSERTED
.................................................
 YOUR LINKED LIST IS :  10  20
.................................................
 BACKWARD TRAVERSING :  20 10
.................................................
1. INSERT DATA
2. DELETE DATA
3. DISPLAY DATA
                ENTER YOUR CHOICE : 1
*****************************************************************************************************************************************
               1 . Insert data in the beginning              2 . Insert data in  the end            3 . Insert data at  any position
*****************************************************************************************************************************************
                                                        Enter your Choice : 3

Enter the data : 15
ENTER THE POS : 2
your data at 2 position inserted sucessfully
.................................................
 YOUR LINKED LIST IS :  10  15  20
.................................................
 BACKWARD TRAVERSING :  20 15 10
.................................................
1. INSERT DATA
2. DELETE DATA
3. DISPLAY DATA
                ENTER YOUR CHOICE :
```

Data Deletion in doubly linked list

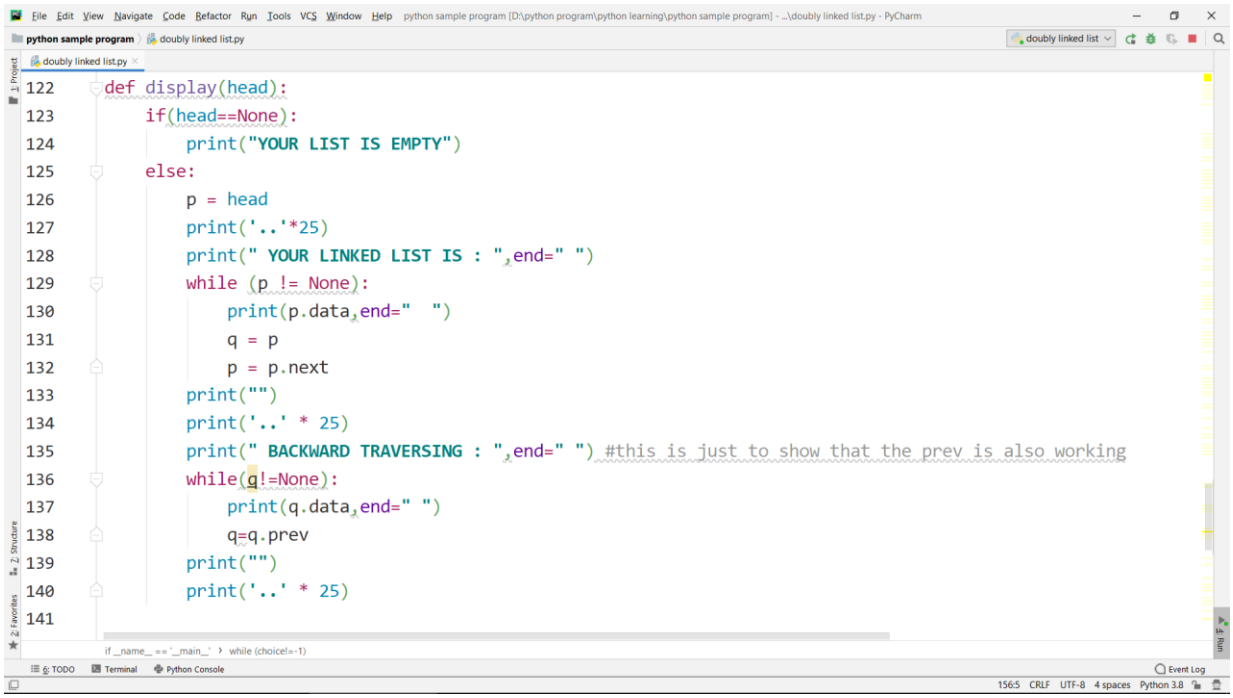
```
54  def delete(head):
55      print("**"*80)
56      print("\t\t\t\t1 . delete data from the beginning",end="")
57      print("\t\t\t\t2 . delete data from the end",end="")
58      print("\t\t\t\t3 . delete data from the any position")
59      print("**" * 80)
60      ch = int(input("\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t Enter your Choice : "))
61      if(ch==1):
62          if(head==None):
63              print("NO ELEMENT TO DELETE ")
64          else:
65              p=head
66              head=p.next
67              if(head!=None):
68                  head.prev=None  #this condition will check whether head is not None means if we want to delete the single left element
69              p=None
70              print("Data deleted from Begining ")
71          return head
72      if (ch==2):
73          if(head==None):
74              print("NO ELEMENT TO DELETE ")
75          else:
76              p=head
77              while(p.next!=None):
78                  p=p.next
79              if(p.prev!=None):           #this condition is for if we want to delete the single left element
80                  p.prev.next=None
81              else:
82                  head=None
83              print("Data deleted from end")
84              return head
85
86      if(ch==3):
87          if (head == None):
88              print("NO ELEMENT TO DELETE ")
89          else:
90              flag=1
91              p=head
92              item=int(input("Enter the data item to be deleted : "))
93              while(p!=None):
94                  if(p.data==item):
95                      flag=1
96                      break
97                  else:
98                      flag=0
99                  p=p.next
100
101
102             if(flag==0):
103                 print("data not found")
104                 return head
```

```
            if (p.next == None):
                print("the only", end="")
                p=None
                head = p             # this will delete the single last element also   ***********
                return head
            elif(p==head):              # if someone want to delete the first data then
                head=p.next
                head.prev=None
                p=None
                return head
            else:                           # kaam chal raha hai koi last element ko udana chahe toh
                q = p.prev
                r = p.next
                q.next = r
                r.prev = q
                return head

def display(head):
    if(head==None):
        print("YOUR LIST IS EMPTY")
    else:
        p = head
        print('..'*25)
        print(" YOUR LINKED LIST IS : ",end=" ")
        while (p != None):
            print(p.data,end="  ")
            q = p
            p = p.next
        print("")
        print('..' * 25)
        print(" BACKWARD TRAVERSING : ",end=" ") #this is just to show that the prev is also working
        while(q!=None):
            print(q.data,end=" ")
            q=q.prev
        print("")
        print('..' * 25)
```

Output

```
Run - python sample program
Run: doubly linked list | circular queue | linked list
..................................................
 YOUR LINKED LIST IS :  70  60  50  40  30  20  10
..................................................
 BACKWARD TRAVERSING :  10 20 30 40 50 60 70
..................................................
1. INSERT DATA
2. DELETE DATA
3. DISPLAY DATA
                ENTER YOUR CHOICE : 2
**************************************************************************************************************************************************************
              1 . delete data from the beginning             2 . delete data from the end               3 . delete data from the any position
**************************************************************************************************************************************************************
                                                    Enter your Choice : 1
Data deleted from Begining
..................................................
 YOUR LINKED LIST IS :  60  50  40  30  20  10
..................................................
 BACKWARD TRAVERSING :  10 20 30 40 50 60
..................................................
1. INSERT DATA
2. DELETE DATA
3. DISPLAY DATA
                ENTER YOUR CHOICE : 2
**************************************************************************************************************************************************************
              1 . delete data from the beginning             2 . delete data from the end               3 . delete data from the any position
**************************************************************************************************************************************************************
                                                    Enter your Choice : 2
Data deleted from end
..................................................
 YOUR LINKED LIST IS :  60  50  40  30  20
..................................................
 BACKWARD TRAVERSING :  20 30 40 50 60
..................................................
1. INSERT DATA
2. DELETE DATA
3. DISPLAY DATA
                ENTER YOUR CHOICE : 2
**************************************************************************************************************************************************************
              1 . delete data from the beginning             2 . delete data from the end               3 . delete data from the any position
**************************************************************************************************************************************************************
                                                    Enter your Choice : 2
Data deleted from end
..................................................
 YOUR LINKED LIST IS :  60  50  40  30  20
..................................................
 BACKWARD TRAVERSING :  20 30 40 50 60
..................................................
1. INSERT DATA
2. DELETE DATA
3. DISPLAY DATA
                ENTER YOUR CHOICE : 2
**************************************************************************************************************************************************************
              1 . delete data from the beginning             2 . delete data from the end               3 . delete data from the any position
**************************************************************************************************************************************************************
                                                    Enter your Choice : 3
Enter the data item to be deleted : 40
..................................................
 YOUR LINKED LIST IS :  60  50  30  20
..................................................
 BACKWARD TRAVERSING :  20 30 50 60
..................................................
```

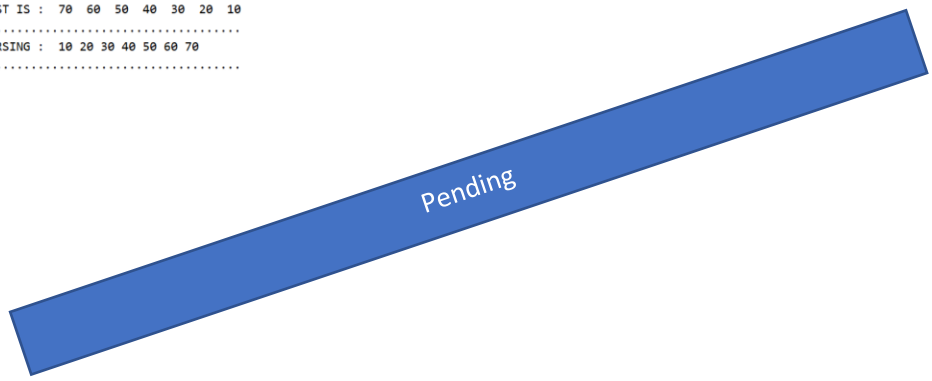
Traversing in doubly linked list



```
def display(head):
    if(head==None):
        print("YOUR LIST IS EMPTY")
    else:
        p = head
        print('..'*25)
        print(" YOUR LINKED LIST IS : ",end=" ")
        while (p != None):
            print(p.data,end="  ")
            q = p
            p = p.next
        print("")
        print('..' * 25)
        print(" BACKWARD TRAVERSING : ",end=" ") #this is just to show that the prev is also working
        while(q!=None):
            print(q.data,end=" ")
            q=q.prev
        print("")
        print('..' * 25)
```

Output

```
..................................................
1. INSERT DATA
2. DELETE DATA
3. DISPLAY DATA
                ENTER YOUR CHOICE : 3
..................................................
 YOUR LINKED LIST IS :  70  60  50  40  30  20  10
..................................................
 BACKWARD TRAVERSING :  10 20 30 40 50 60 70
..................................................
```



To be continue.....