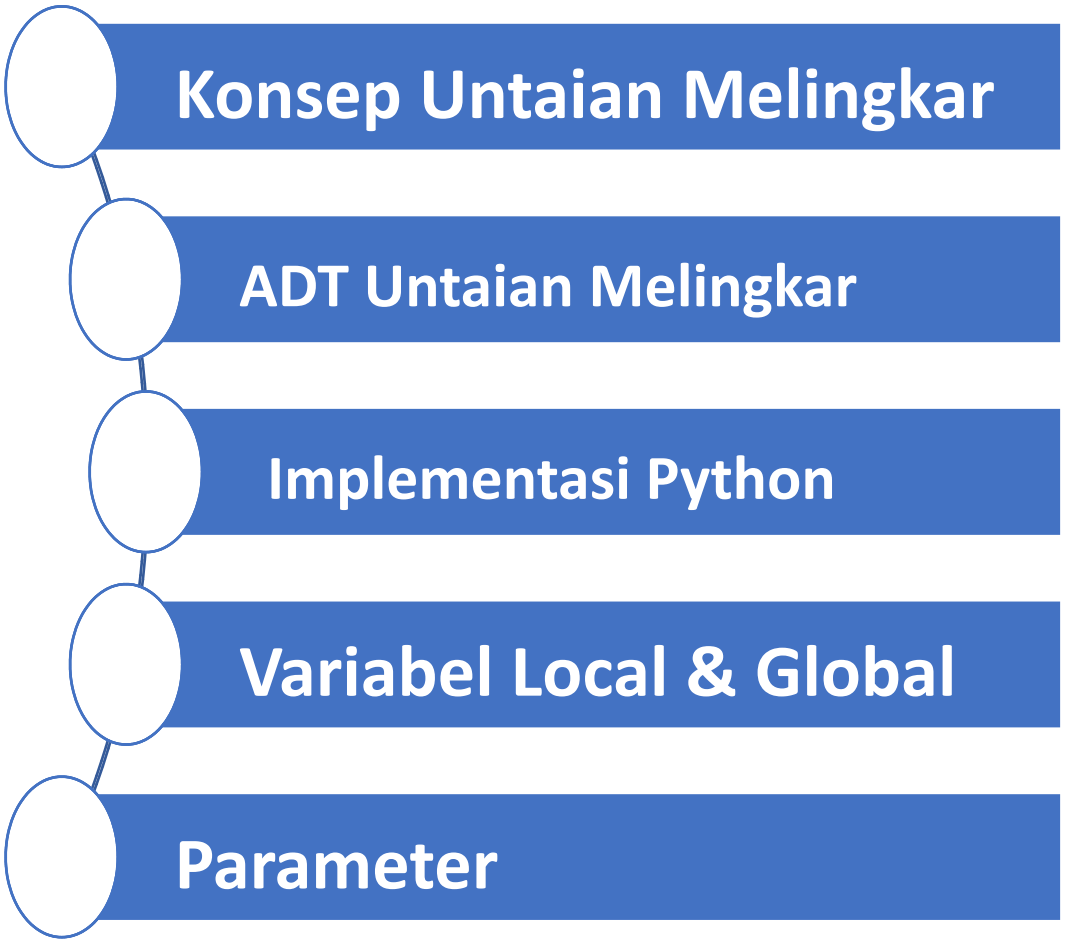
#KampusKuningGading

STRUKTUR DATA (PYTHON)

"Untaian Melingkar"

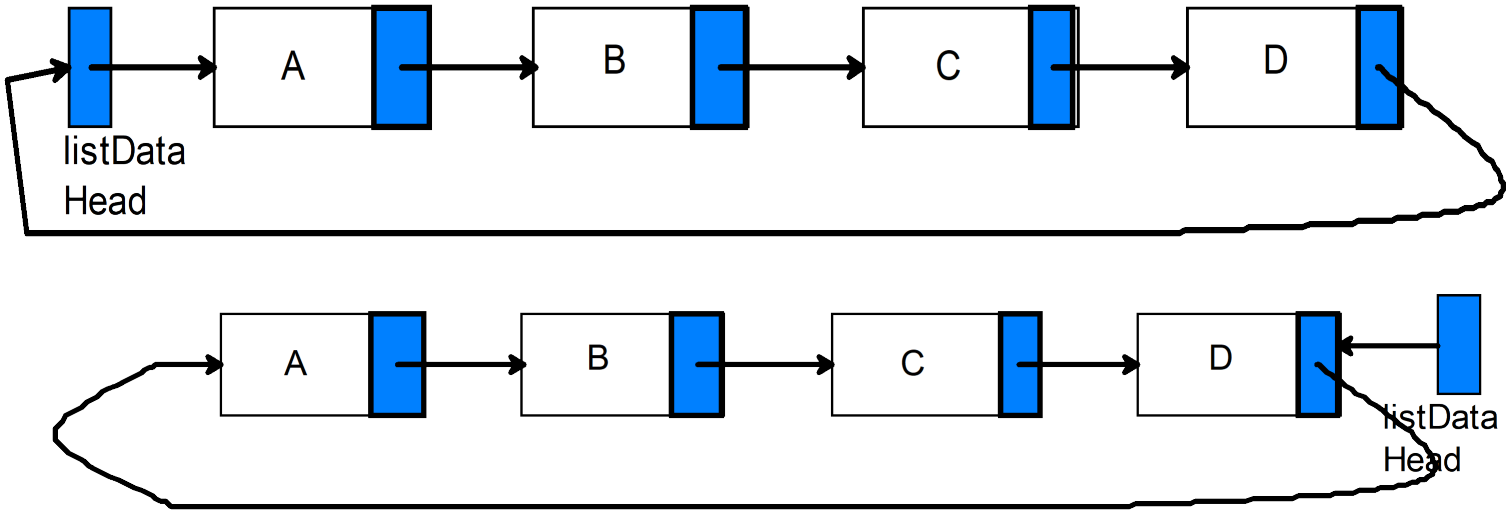
[@SUARGA| [Pertemuan 10]

OutLine

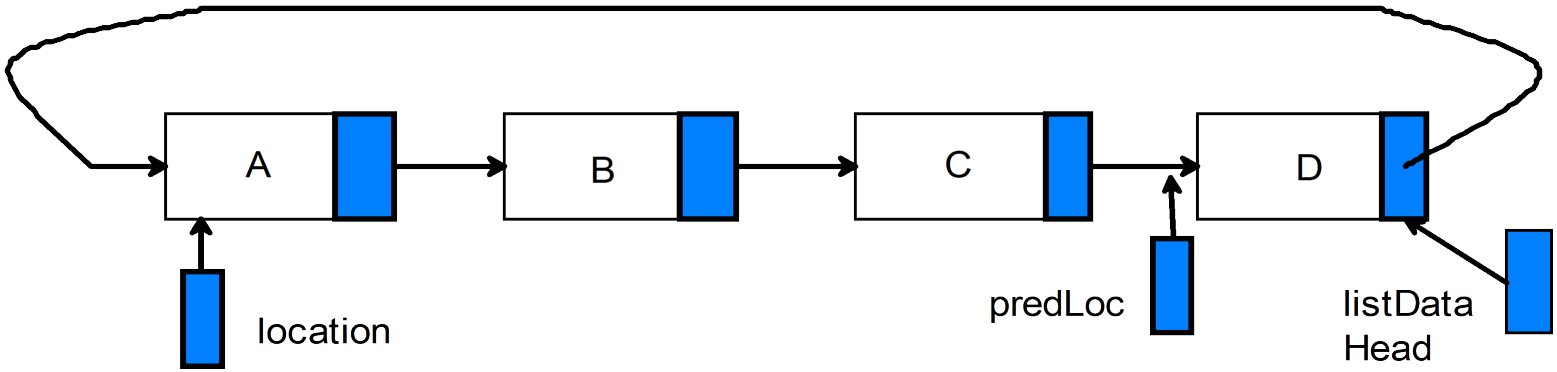

Konsep Untaian Melingkar
ADT Untaian Melingkar
Implementasi Python
Variabel Local & Global
Parameter

Konsep Untaian Melingkar

- Struktur untaian tunggal dapat dimodifikasi menjadi struktur untaian melingkar dimana pointer next pada elemen terakhir yang menunjuk null diubah sehingga menunjuk ke elemen pertama yang ditunjuk oleh Head. Bentuk struktur untaian melingkar adalah sebagai berikut:



- Salah satu proses yang paling penting pada untaian melingkar ini adalah proses mencari elemen (**searchClist**), karena proses ini diperlukan ketika membaca satu elemen tertentu, menyisipkan elemen, dan juga ketika menghapus elemen.
- Prosedur searchClist menggunakan tiga pointer, yaitu: ***Head*** yang menunjuk elemen pertama, ***location*** yang menunjuk node yang sedang dibaca, dan ***predLoc*** yang menunjuk node sebelum location.



```python
#my_Circular_LL.py
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class CircularLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def add_node(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            self.tail = new_node
        else:
            self.tail.next = new_node
            self.tail = new_node

    def print_list(self):
        current_node = self.head
        while current_node is not None:
            print(current_node.data)
            current_node = current_node.next
```

```
def insert_node_in_middle(self, list, thisdata, data):
    #insert data after thisdata
    if list.head is None:
        list.head = Node(data)
        list.tail = list.head
    else:
        current_node = list.head
        while current_node.data != thisdata:
            current_node = current_node.next
        new_node = Node(data)
        new_node.next = current_node.next
        current_node.next = new_node

def delete_node(self,list, data):
    if list.head is None:
        return

    current_node = list.head
    previous_node = None

    while current_node is not None:
        if current_node.data == data:
            if previous_node is None:
                list.head = current_node.next
            else:
                previous_node.next = current_node.next
        break

    previous_node = current_node
    current_node = current_node.next
```

```
if __name__ == "__main__":
    my_list = CircularLinkedList()
    print("add 3 nodes")
    my_list.add_node(1)
    my_list.add_node(2)
    my_list.add_node(3)
    print("Insert a node in the middle")
    my_list.insert_node_in_middle(my_list, 2, 4)
    my_list.print_list()
    print("delete node 2")
    my_list.delete_node(my_list, 2)
    my_list.print_list()
```

ADT CircularList

- **__init__()** : inisialisasi CList
- **__len__()** : menghitung cacah elemen CList
- **is_empty()** : memeriksa apakah kosong?
- **first()** : melihat elemen terdepan
- **delete()** : menghapus elemen pertama
- **add_last(e)** : menyisip e di posisi akhir
- **insert_order(e)** : menyisip e pada urutan-nya
- **search_item(target)**: mencari elemen target
- **traverse()** : menelusuri semua elemen
- **rotate()** : memutar untaian kepala <-> ekor

inisialisasi

```
def __init__(self):
      self._tail = None
      self._size = 0
```

- prosedur ini menciptakan untaian melingkar yang kosong, jadi size = 0

cacah elemen

```
def __len__(self):
      return self._size
```

- atribute _size digunakan untuk menyimpan cacah elemen dalam untain melingkar

Apakah untaian Kosong?

```
def is_empty(self):
      return self._size == 0
```

- fungsi ini memeriksa apakah cacah _size = 0, bila ya (true) berarti untaian kosong, bila tidak (false) berarti untaian tidak kosong.

elemen pertama: first()

- karena untaian melingkar maka elemen yang ditunjuk oleh tail.next (ekor untaian) pasti head (kepala untaian), elemen pertama adalah: head._element

```
def first(self):
    if self.is_empty():
       raise Empty('List is empty')
    head = self._tail._next
    return head._element
```

hapus yang pertama: delete()

- Proses penghapusan pada contoh ini adalah elemen pertama dalam untaian

```
    def delete(self):
        #Remove, return the 1ST element of the list .
        #Raise Empty exception if the list is empty.

        if self.is_empty():
            raise Empty('List is empty')
        answer = self._tail._element
        oldhead = self._tail
        if self._size == 1: # removing only element
            self._tail = None # queue becomes empty
        else:
            self._tail = oldhead._next # bypass the old head

        self._size -= 1
        return answer
```

Sisip diposisi akhir: add_last()

```
    def add_last(self, e):
        #Add an element to the back of the list.
        newest = self._Node(e, None) # node will be new tail node
        if self.is_empty():
            newest._next = newest # initialize circularly
        else:
            newest._next = self._tail._next # new node points to head
            self._tail._next = newest # old tail points to new node
        self._tail = newest # new node becomes the tail
        self._size += 1
```

Sisip pada urutan: insert_order()

```
    def insert_order(self, e):
        # add an element in order
        newest = self._Node(e, None)
        if self.is_empty():                          # empty list
            #newest._next = self._tail
            newest._next = newest
            self._tail = newest
        elif e < self._tail._element:                # insert before
            newest._next = self._tail._next
            self._tail._next = newest
            self._tail = newest
        else:
            found = False
            prevP = None
            p = self._tail
            x=self._tail._element
            n = self._size
            for i in range(n):
```

```
            for i in range(n):
                if (e > x and p._next is not None):
                    prevP = p
                    p = p._next
                    x = p._element
                elif p is None:
                    newest._next = self._tail._next
                    self._tail._next = newest
                    found = True
                    break
                else:
                    found = True
                    newest._next = prevP._next
                    prevP._next = newest
                    break

        self._size += 1
```

Mencari node: search_item(target)

```
100     def search_item(self, target):
            pos=0
            found=False
            curNode = self._tail
            n = self._size
            for i in range(n):
                curNode = curNode._next
                pos += 1
                if curNode._element == target:
                    found = True
110                 break

            if found:
                print(target, ' diposisi: ',pos)
            else:
                print(target, ' tidak ada dalam list !')
```

Menelusuri: traverse()

```
120     def traverse(self):
            curNode = self._tail
            n=self._size
            for i in range(n):
                print(curNode._element)
                curNode = curNode._next
123
```

Memutar arah : rotate()

```
def rotate(self):
    #Rotate front element to the back of the queue.”””
    if self._size > 0:
        self._tail = self._tail._next # old head becomes new tail
```

Prosedur searchClist(Head, item, location, predLoc, found):

```
1. location <- Head.next;
2. predLoc <- Head;
3. found <- false;
4. masihCari <- true;
5. while (masihCari && !found) do
            if ( item < location.isi)
            then masihCari ßfalse;
            else if (item = location.isi)
                      then found <- true;
                 else {
                    predLoc <- location;
                    location <- location.next;
                    masihCari <- (location != Head.next)
                 }
                 endif;
              endif;
       endwhile;
```

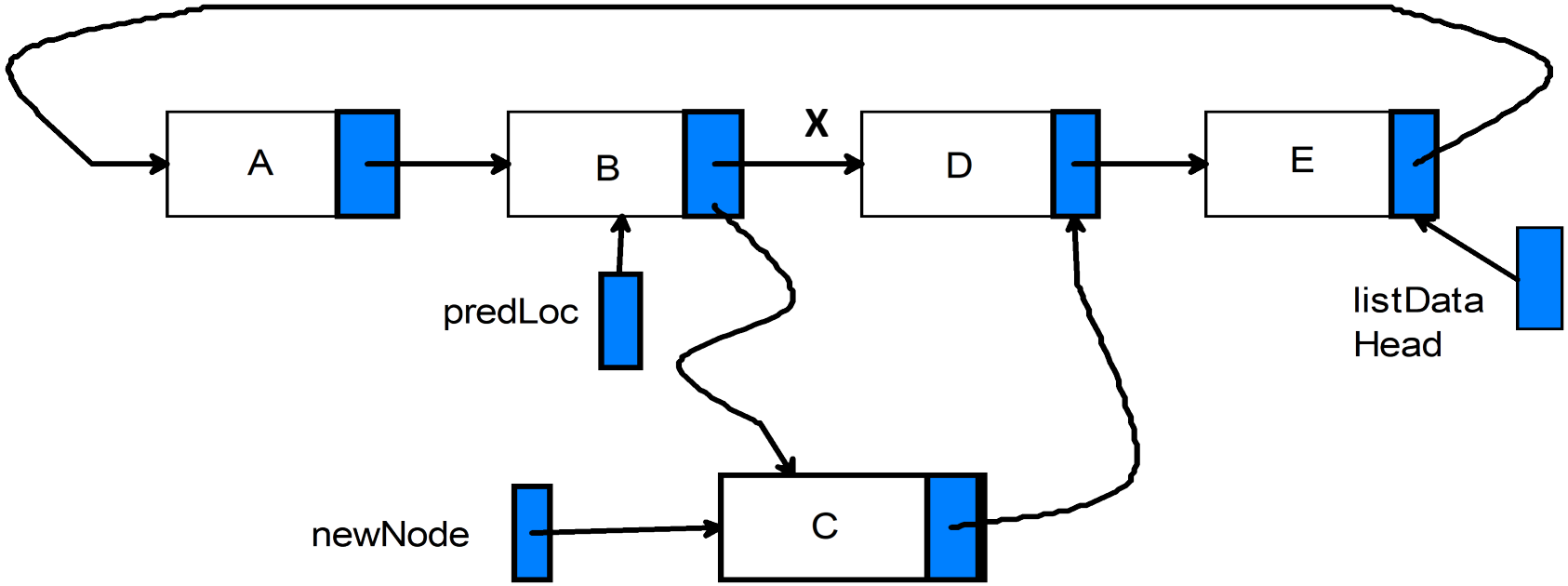
Beberapa fungsi tambahan

- insertClist() : menyisipkan elemen diantara dua elemen yang sudah ada
- deleteNode() : menghapus node tertentu

Menyisipkan data (insertClist)

- Proses menyisipkan satu elemen ke dalam untaian melingkar adalah sebagai berikut.

```
Prosedur insertClist(item)
1. ciptakan sebuah node baru,  New(node);
2. masukkan item kedalam isi node:   node.isi <- item;
3. if (Head != Null)
        //gunakan prosedur searchClist untu mencari posisi
        searchClist(Head, item, location, predLoc, found);
        // sisip diantara dua elemen
        node.next <- predLoc.next;
        predLoc.next <- node;
        if (Head.isi < item)
            Head <- node;
         else {
                  // untaian kosong
                  Head <- node;
                  node.next <- node;
          }
         endif;
    endif;
```



Menyisipkan C dalam untaian melingkar antara B dan C
node <- newNode(C) ; predLoc <- B;
node.next <- predLoc.next
predLoc.next <- node

menghapus node (deleteNode)

- **Prosedur deleteNode()**

1. **searchClist(**Head, item, location, predLoc, found);
2. if (predLoc = location) // hanya ada satu elemen
   then Head ← Null;
   else {
         predLoc.next ← location.next;
         if (location = Head) // elemen terakhir dihapus
         then Head ← predLoc;
         endif;
        }
   endif;
3. dispose(location);

Python implementation

- Dengan meng-edit semua fungsi sebelumnya menjadi satu maka diperoleh suatu implementasi dari untaian melingkar.

CircularList.py

```
#CircularList.py - implementasi untaian melingkar @Suarga
from Empty import Empty
class CircularList:

    #-------------------- nested Node class --------------------
    class _Node:
        #Lightweight, nonpublic class for storing a singly linked node.
        __slots__ = '_element', '_next'     # streamline memory usage

        def __init__(self, element, next): # initialize node's fields
            self._element = element        # reference to user's element
            self._next = next              # reference to next node

    #-------------------- methods ------------------------------

    def __init__(self):
        #Create an empty list."""
            self._tail = None # will represent tail of queue
            self._size = 0 # number of queue elements

    def __len__(self):
        #Return the number of elements in the queue."""
        return self._size

    def is_empty(self):
        #Return True if the queue is empty."""
        return self._size == 0
```

```
    def first(self):
        #Return,do not remove the element at the front of the list.
        #Raise Empty exception if the list is empty.

        if self.is_empty():
            raise Empty('List is empty')
        head = self._tail._next
        return head._element

    def delete(self):
        #Remove, return the 1ST element of the list .
        #Raise Empty exception if the list is empty.

        if self.is_empty():
            raise Empty('List is empty')
        answer = self._tail._element
        oldhead = self._tail
        if self._size == 1: # removing only element
            self._tail = None # queue becomes empty
        else:
            self._tail = oldhead._next # bypass the old head

        self._size -= 1
        return answer
```

```
    def add_last(self, e):
        #Add an element to the back of the list.
        newest = self._Node(e, None) # node will be new tail node
        if self.is_empty():
            newest._next = newest # initialize circularly
        else:
            newest._next = self._tail._next # new node points to head
            self._tail._next = newest # old tail points to new node
        self._tail = newest # new node becomes the tail
        self._size += 1

    def insert_order(self, e):
        # add an element in order
        newest = self._Node(e, None)
        if self.is_empty():                    # empty list
            #newest._next = self._tail
            newest._next = newest
            self._tail = newest
        elif e < self._tail._element:          # insert before
            newest._next = self._tail._next
            self._tail._next = newest
            self._tail = newest
        else:
            found = False
            prevP = None
            p = self._tail
            x=self._tail._element
            n = self._size
```

```
            for i in range(n):
                if (e > x and p._next is not None):
                    prevP = p
                    p = p._next
                    x = p._element
                elif p is None:
                    newest._next = self._tail._next
                    self._tail._next = newest
                    found = True
                    break
                else:
                    found = True
                    newest._next = prevP._next
                    prevP._next = newest
                    break

        self._size += 1

    def search_item(self, target):
        pos=0
        found=False
        curNode = self._tail
        n = self._size
        for i in range(n):
            curNode = curNode._next
            pos += 1
            if curNode._element == target:
                found = True
                break
```

```
        if found:
            print(target, ' diposisi: ',pos)
        else:
            print(target, ' tidak ada dalam list !')

    def traverse(self):
        curNode = self._tail
        n=self._size
        for i in range(n):
            print(curNode._element)
            curNode = curNode._next

    def rotate(self):
        #Rotate front element to the back of the queue."""
        if self._size > 0:
            self._tail = self._tail._next # old head becomes new tail
```

Contoh pemakaian sebagai berikut:

```
>>> c=CircularList()
>>> c.add_last('A')
>>> c.add_last('B')
>>> c.add_last('D')
>>> c.traverse()
D
A
B
>>> c.add_last('C')
>>> c.traverse()
C
A
B
D
>>> c.search_item('C')
C  diposisi:  4
>>> c.search_item('A')
A  diposisi:  1
>>> c.rotate()
>>> c.traverse()
A
B
D
C
>>> c.search_item('A')
A  diposisi:  4
>>>
```

CircularQueue.py

- Sebagai contoh pemakaian dari circular-linked-list, berikut ini disajikan implementasi antrian (queue) memakai circular-linked-list sebagai struktur data dasar

```
#CircularQueue.py ==> @Suarga
#implementasi antrian memakai Circular-Linked_List
class CircularQueue:
    #Queue implementation using circularly linked list

    #-------------------- nested Node class --------------------
    class _Node:
        #Lightweight, nonpublic class for storing a singly linked node."""
        __slots__ = '_element', '_next'     # streamline memory usage

        def __init__(self, element, next): # initialize node's fields
            self._element = element        # reference to user's element
            self._next = next              # reference to next node
```

```
#-------------------- methods -----------------------------

def __init__(self):
    #Create an empty queue."""
        self._tail = None # will represent tail of queue
        self._size = 0 # number of queue elements

def __len__(self):
    #Return the number of elements in the queue."""
    return self._size

def is_empty(self):
    #Return True if the queue is empty."""
    return self._size == 0

def first(self):
    #Return,do not remove the element at the front of the queue.
    #Raise Empty exception if the queue is empty.

    if self.is_empty():
        raise Empty('Queue is empty')
    head = self._tail._next
    return head._element
```

```
def dequeue(self):
    #Remove, return the 1ST element of the queue (i.e., FIFO).
    #Raise Empty exception if the queue is empty.

    if self.is_empty():
        raise Empty('Queue is empty')
    oldhead = self._tail._next
    if self._size == 1: # removing only element
        self._tail = None # queue becomes empty
    else:
        self._tail._next = oldhead._next # bypass the old head
    self._size -= 1
    return oldhead._element

def enqueue(self, e):
    #Add an element to the back of queue."""
    newest = self._Node(e, None) # node will be new tail node
    if self.is_empty():
        newest._next = newest # initialize circularly
    else:
        newest._next = self._tail._next # new node points to head
        self._tail._next = newest # old tail points to new node
    self._tail = newest # new node becomes the tail
    self._size += 1
```

```
def rotate(self):
    #Rotate front element to the back of the queue.”””
    if self._size > 0:
        self._tail = self._tail._next # old head becomes new tail
```

test CircularQueue

```
#CircularQueue_test.py
#contoh pemakaian Circular Queue
from CircularQueue import *

def main():
    print("menciptakan antrian Q:")
    Q = CircularQueue()
    print("memasukkan 5 data:")
    Q.enqueue(5)
    Q.enqueue(6)
    Q.enqueue(7)
    Q.enqueue(8)
    Q.enqueue(9)
    #mengitung elemen dalam Q
    print("Panjang antrian:")
    print(len(Q))
    print("elemen pertama :")
    print(Q.first())
    print("melakukan pelayanan 2 kali")
    print(Q.dequeue())
    print(Q.dequeue())
    print("Apakan antrian Kosong?:")
    print(Q.is_empty())
    print("melayani semua isi antrian")
    for x in range(len(Q)):
        print(Q.dequeue())

    print("Apakah antrian kosong?")
    print(Q.is_empty())

main()
```

Hasil Test

```
========== RESTART: D:/USER/Python/CircularQueue_test.py
menciptakan antrian Q:
memasukkan 5 data:
Panjang antrian:
5
elemen pertama :
5
melakukan pelayanan 2 kali
5
6
Apakan antrian Kosong?:
False
melayani semua isi antrian
7
8
9
Apakah antrian kosong?
True
```