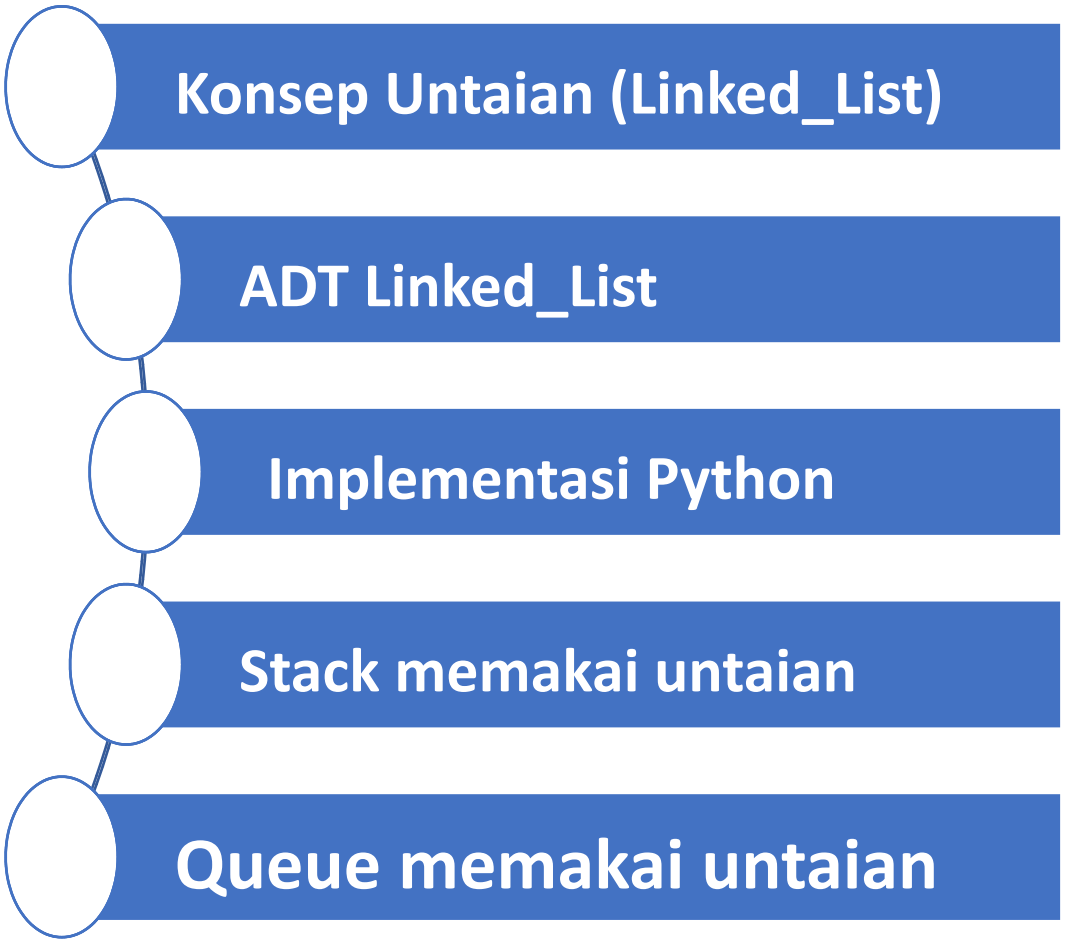
#KampusKuningGading

STRUKTUR DATA (PYTHON)

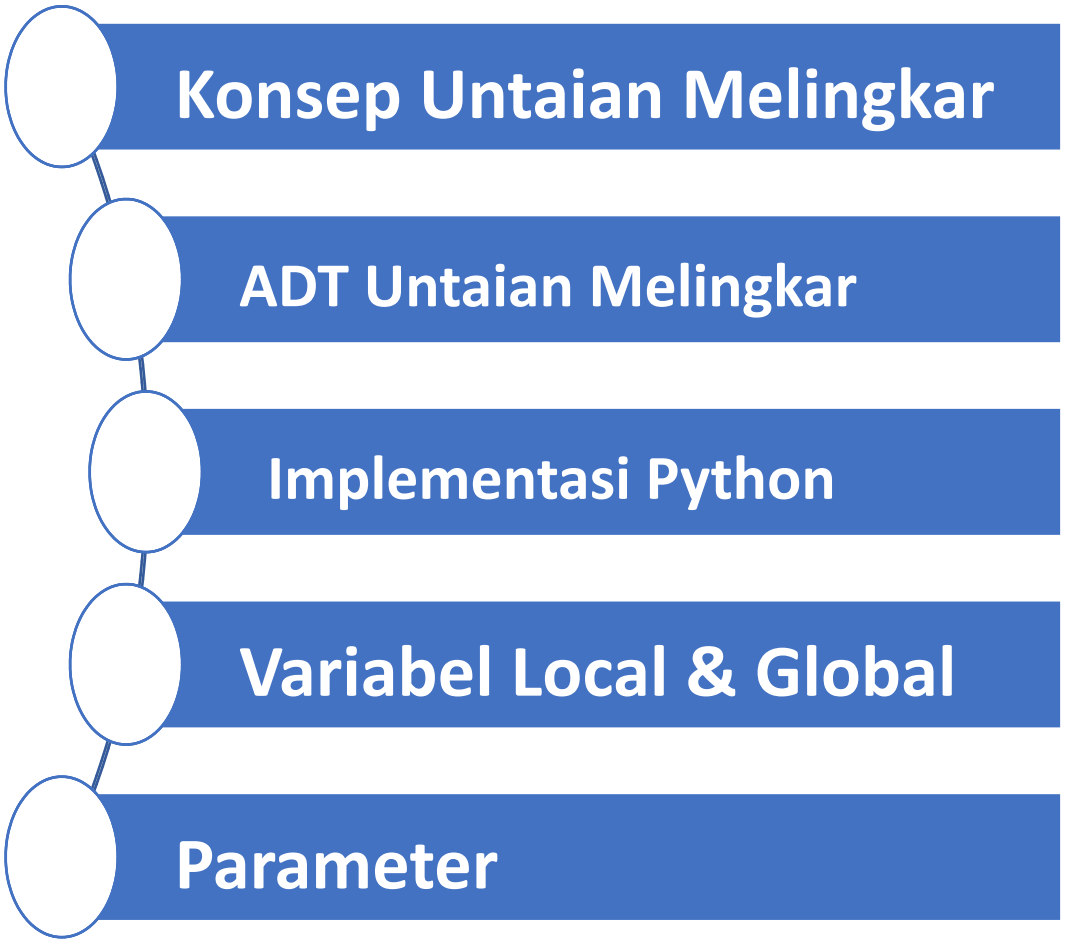
"Struktur Untaian Tunggal + Melingkar"

[@SUARGA| [Pertemuan 09 + 10]

OutLine

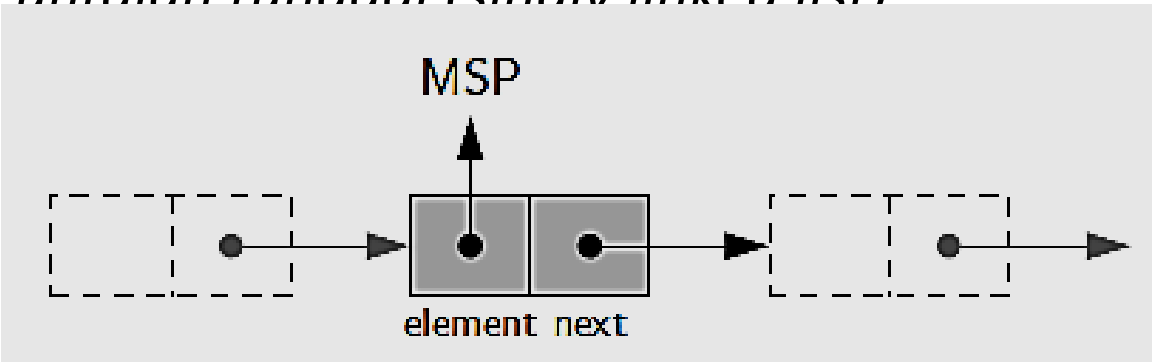

Konsep Untaian (Linked_List)
ADT Linked_List
Implementasi Python
Stack memakai untaian
Queue memakai untaian

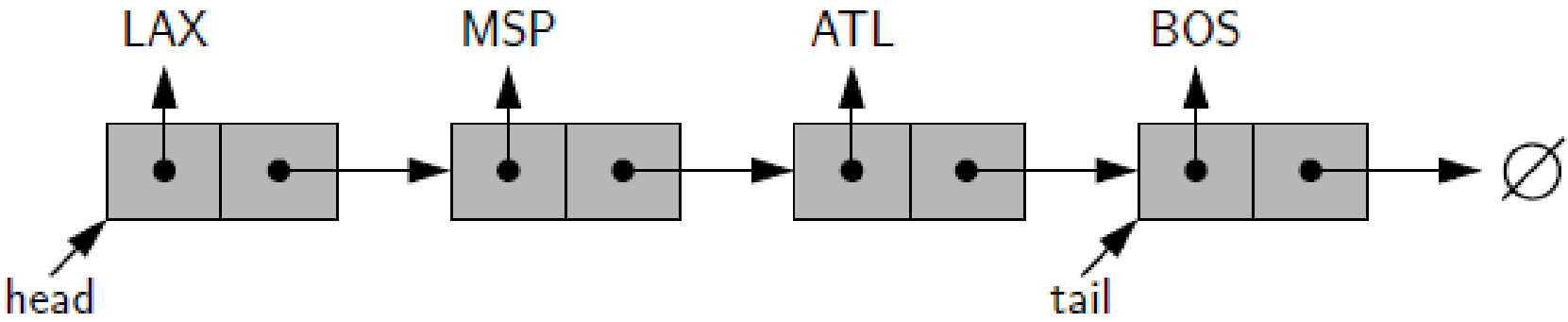
OutLine


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

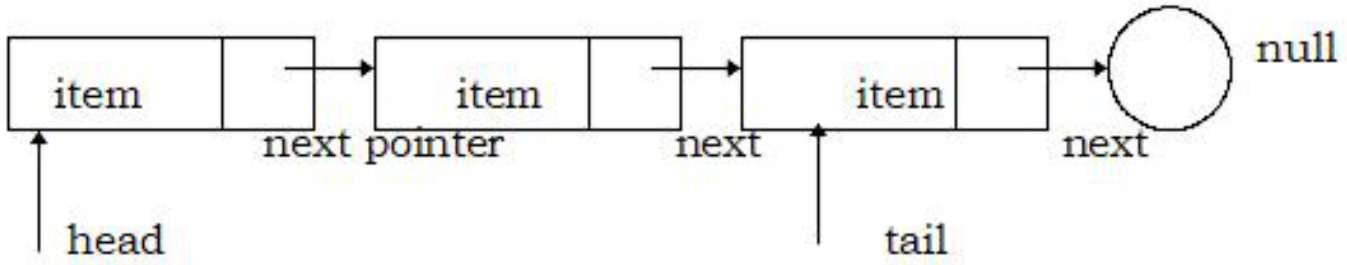
Konsep Untaian

- Struktur untaian adalah suatu struktur data, dengan elemen yang disebut “node” yang terdiri atas **wadah untuk data** (item/element) dan suatu **penunjuk** (pointer/next) ke elemen untaian berikutnya.
- Secara khusus pada bagian ini akan disajikan suatu struktur untaian yang hanya memiliki satu pointer dan biasa disebut sebagai *untaian tunggal (singly linked list)*





Suatu pointer khusus di-sediakan untuk menunjuk ke awal untaian atau node pertama yaitu pointer ***head***, dan di akhir untaian suatu pointer lain biasa diadakan untuk menunjuk item kosong atau ***null***. Suatu pointer tambahan ***tail*** bisa ditambahkan untuk menunjuk node terakhir, sebelum null.



class ListNode

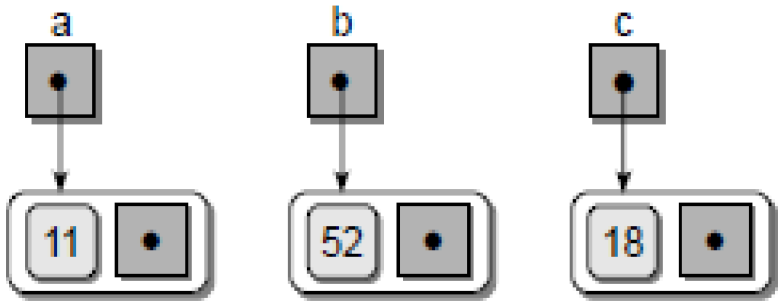
- Dalam Python, struktur dasar untaian tunggal ini dapat ditulis dalam bentuk “class ListNode”, terdiri atas item (wadah) dan next (penunjuk) sebagai berikut (Python tidak mengenal pointer) :

```
class Node :
def __init__(self, data) :
      self.item = data
      self.next = None
```

instruksi: a = Node(11)

b = Node(52)

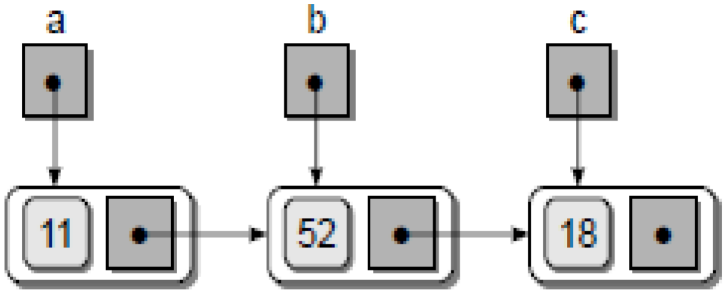
c = Node(18)



apabila ditambahkan instruksi:

a.next = b

b.next = c



ADT Singly Linked List

- Beberapa fungsi utama dalam ADT untaian antara lain adalah sebagai berikut:

- **init** : inisialisasi struktur untaian, pointer head menunjuk (→) null.
- **add_first(e) :** memasukkan elemen baru e di-awal list
- **add_last(e) :** memasukkan elemen baru e di-akhir list
- **delete_first()** : menghapus elemen pertama dari untaian
- **delete_item(x)** :menghapus elemen tertentu dari untaian
- **search_list(e)** : mencari satu elemen dalam untaian
- **first()** : menampilkan elemen pertama
- **last()** : menampilkan elemen terakhir
- **is_empty()** : memeriksa apakah list kosong
- **insert_after()** : menyisipkan elemen baru setelah suatu node tertentu
- **traverse()** : menelusuri semua node dan menampilkan isi-nya

Fungsi inisialisasi: __init__

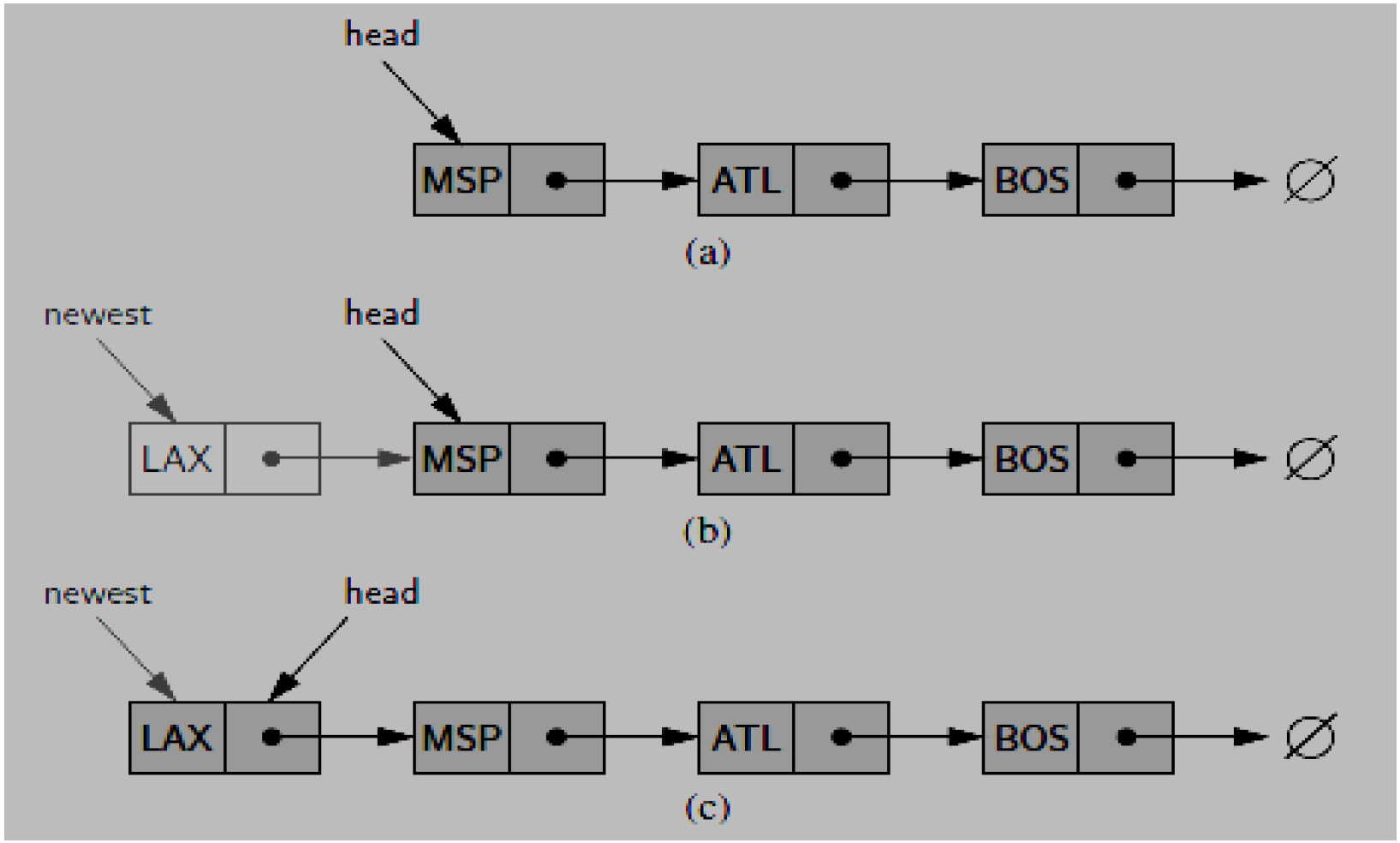
```
class _LinkedList:
    #Base class untuk representasi linked list
    class _Node:
    #class non-public untuk node untaian
        __slots__ = '_element' , '_next'   # streamline memory

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

    def __init__(self):
        #menciptakan untaian kosong
        self._head = None
        self._tail = None
        self._size = 0 # number of elements
```

Penyisipan data / insert

- Setelah untaian di-bentuk maka langkah berikut-nya adalah langkah pemasukkan atau penyisipkan data/elemen. Sebenarnya ada beberapa teknik untuk memasukkan elemen ke dalam untaian, antara lain: **sisip depan (add_first())** atau insert_at_head(), **sisip akhir (add_last())** atau insert_at_tail(), **sisip setelah node tertentu (insert_after) dan insert_item().**

- Prosedur sisip-depan (**add_first**) berarti menempatkan elemen baru selalu didepan, langkahnya adalah sebagai berikut:

  - Ciptakan node baru, newest= Node(x);
  - buatlah pointer next node baru ini menujuk ke elemen yang sedang ditunjuk Head,
    newest.next <- head
  - pointer Head menujuk ke node baru ini,
    head <- newest;



illustrasi fungsi add_first()

Fungsi : add_first()

- **Python coding:**

```
add_first(self,x):
  # ciptakan node untuk nilai x
  newest = Node(x)
  # next node baru menunjuk ke node yg ditunjuk head
  newest.next = self.head
  # head menunjuk node baru
  self.head = newest
  self.size += 1  # dan size ditambah 1
```

- Prosedur sisip-akhir (**add_last**) berarti menempatkan elemen baru selalu dibelakang, langkahnya bisa sebagai berikut:

  - Ciptakan node baru, newest=Node(x)
  - pointer node baru selalu menunjuk akhir (null),
    newest.next ← null;
  - Telusuri untaian mulai dari Head hingga elemen terakhir (Last):

```
thisNode <- self.head;
Last <- null;
while (thisNode.next <> null) do
            Last <- thisNode;
            thisNode <- thisNode.next;
endwhile;
```

  - pointer terakhir mnunjuk node baru, `Last.next ←newest;`

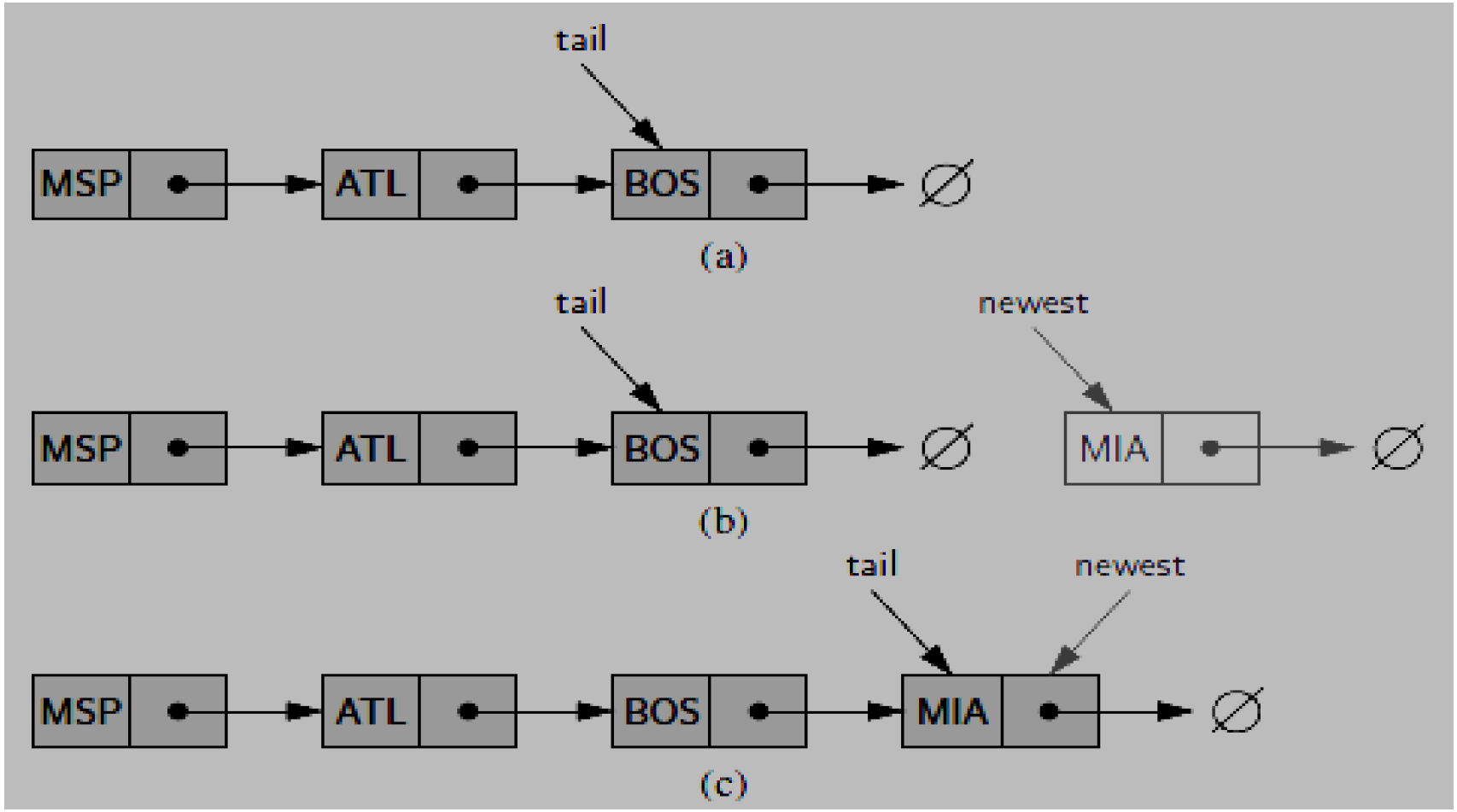
- Cara lain untuk sisip-akhir adalah dengan menciptakan suatu pointer akhir (tail), kemudian memakai pointer ini untuk selalu menujuk elemen terakhir.

- Prosedur add_last bisa dilakukan sebagai berikut:
  - Ciptakan satu elemen (node) baru
  - Buatlah pointer next dari elemen baru ini menunjuk null
  - Bila pointer tail menunjuk null (empty), maka buatlah tail menujuk elemen baru

Fungsi: add_last()

- **Python coding**

```
add_last(self,e):
        # ciptakan node baru untuk e
        newest = Node(e)
       # next node baru menunjuk null
        newest.next = Null
       # next dari tail menunjuk node baru
        self.tail.next = newest
       # pointer tail pindah ke node baru
        self.tail = newest
        self.size = self.size + 1
        # size ditambah 1
```



Illustrasi fungsi add_last()

- Menyisipkan node baru e setelah node tertentu (**insert_after**), bisa dilakukan dengan langkah sebagai berikut:

  – Cari posisi x dari node d dalam list (lihat search_list)
  – andaikan y adalah pointer dari x.next
  – ciptakan node baru newest=Node(e)
  – buat pointer next node baru sama dengan y, newest.next ← y
  – buat pointer x.next menunjuk node baru,
    x.next ← newest

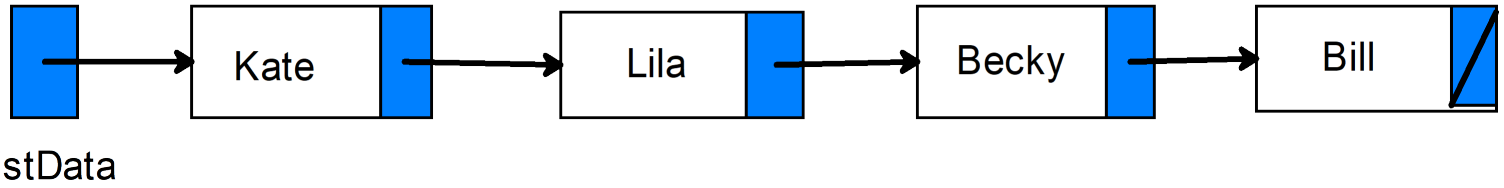
Fungsi: insert_after()

```
def insert_after(self, d, e):
    #masukkan element e setelah node d
    #dan return new node.
    x = self._head
    y = x._element
    found = False
    while (not found and x != None):
        if (y == d):
            found = True
            z = x._next
            newest = self._Node(e,z)
            x._next = newest
            self._size += 1
            break
```

```
    else:
            x = x._next
            if (x != None):
                y = x._element

if (found):
    return newest._element
else:
    print('Node : ',d, ' tdk ada dalam list')
```

Pencarian node (search)

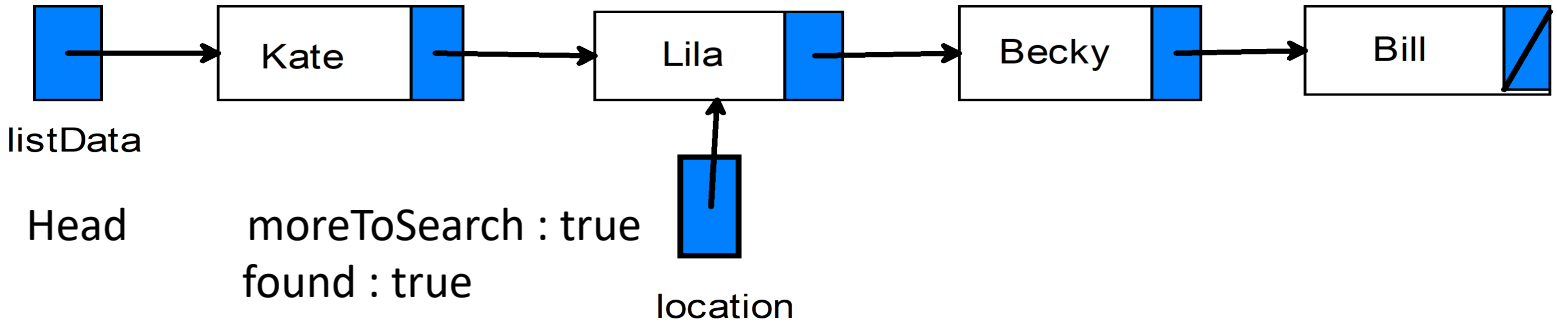
- Prosedur pencarian elemen **search_List** dalam suatu untaian dapat dilakukan sebagai berikut:
  - telusuri untaian dengan memeriksa isi elemen apakah sama dengan elemen yang dicari
  - bila sama maka variabel found = true,
  - bila tidak found = false



Head

moreToSearch : false
found: false
location: false

Mencari “Charlie” dalam untaian menghasilkan found ← false;



Mencari “Lila” dalam untaian menghasilkan found ← true.

Fungsi search_list()

```
def search_list(self,e):
        if self.is_empty():
            raise Empty('List is empty')
        x = self._head
        y= x._element
        i=1
        found = False
        while (not found and x != None):
            print(y)
            if (y == e):
                found = True
                break
```

```
    else:
            x = x._next
            if (x != None):
                y = x._element
            i+=1
  if (found):
      print('ditemukan di posisi ke-',i)
  else:
      print(e, 'tidak ditemukan dalam list')
return found
```

menghapus node (delete)

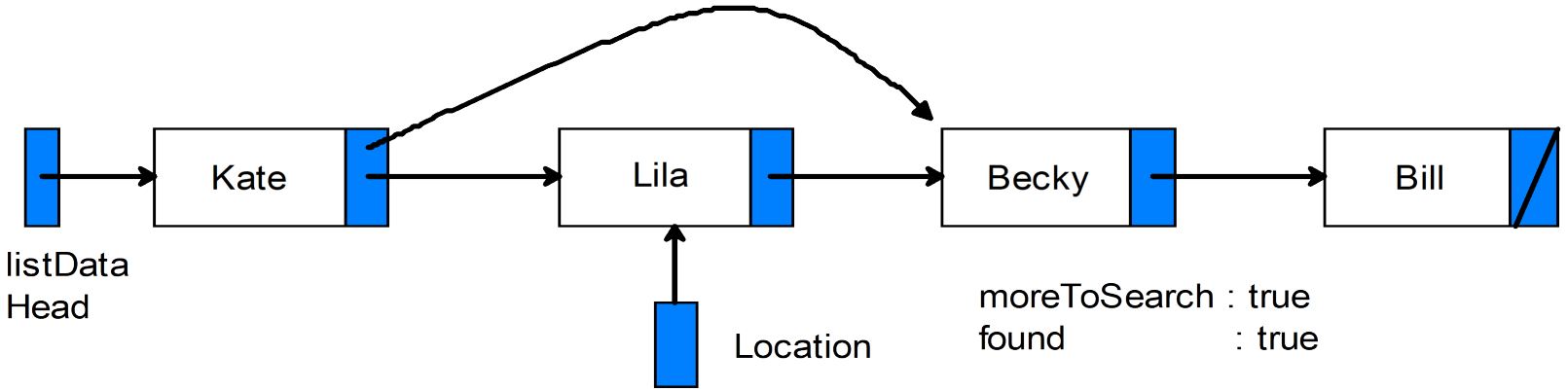
- Proses penghapusan satu elemen dari untaian (delete_Item) dapat dilakukan sebagai berikut:
  - **cari elemen yang akan dihapus (location, found=true)**
  - **bila ketemu maka hapus (dispose)**

Rincian Langkah

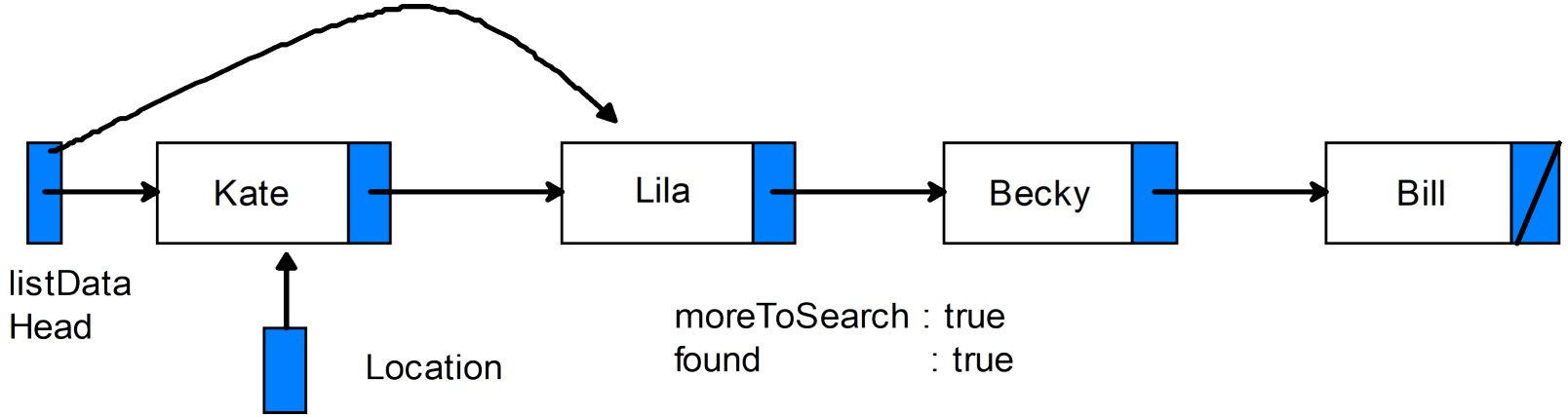
```
searchList(Head, thisptr, lastptr, x, done);
if (done)
then  if (lastptr = Null)
        then lastptr <- Head.next;
                dispose(Head);
           Head <- lastptr;
        else  lastptr.next <- thisptr.next;
                dispose(thisptr);
        endif
endif.
```

Proses pencarian elemen untuk dihapus

```
bila  (x = Head.isi)
{ berarti elemen pertama yang akan dihapus }
maka:   lokasi <- Head;
        Head <- lokasi.next;
        dispose(lokasi);
bila tidak maka :
lokasi <- Head;
{ cari posisi elemen }
         selama ( x <> (lokasi.next).isi):
                lokasi <- lokasi.next;
                 hapus <- lokasi.next;
                 lokasi.next <- (lokasi.next).next;
                 dispose(hapus);
```



Apabila yang dihapus (delete) adalah "Lila", maka pointer ke Lila dipindahkan ke Becky



Apabila yang dihapus (delete) adalah "Kate", maka Head pointer menunjuk Lila.

Fungsi delete_item()

```
def delete_item( self, item ):
    predNode = None
    curNode = self._head
    while curNode is not None and
        curNode._element != item :
        predNode = curNode
        curNode = curNode.next
        # item harus ada agar bisa diambil.
        assert curNode is not None,
              "Item harusnya ada dalam list !!!"
```

```
# Lepaskan node dan berikan nilai itemnya.
self._size -= 1
if curNode is self._head :
      self._head = curNode.next
else :
      predNode.next = curNode.next
return curNode.item
```

- Salah satu cara menghapus elemen termudah adalah dengan menghapus elemen paling depan **(delete_first)**, prosedurnya sebagai berikut:

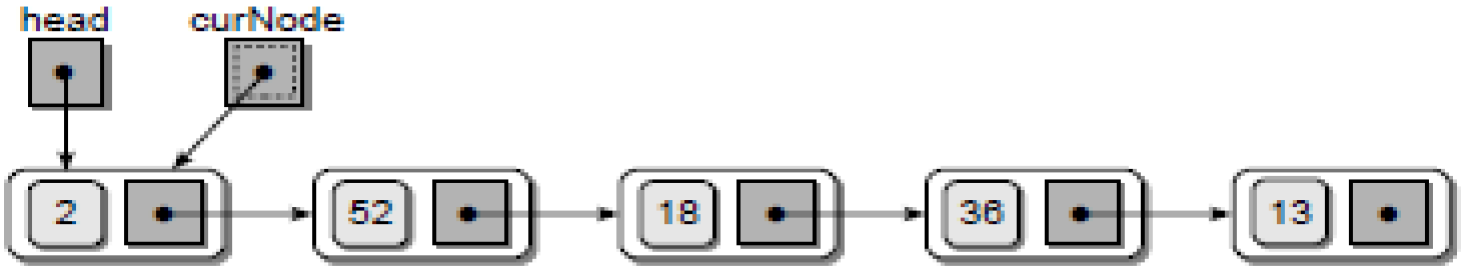
```
def delete_first(self):
    if self.is_empty():
        raise Empty('List is empty')
    answer = self._head._element
    self._head = self._head._next
    self._size -= 1
    return answer
```

Penelusuran (traverse)

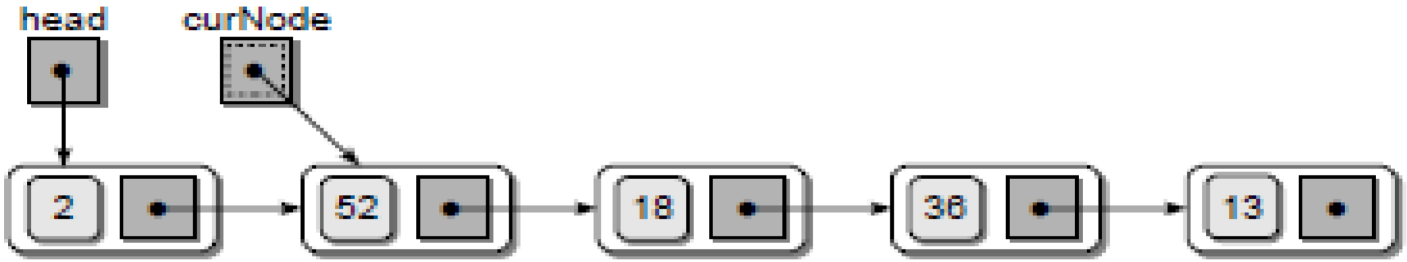
- Menelusuri semua elemen dari linked list dapat dimulai dari pointer head kemudian selanjutnya menelusuri pointer next ke node berikutnya hingga node terakhir dengan isi None ditemukan.

```
def traversal( head ):
    curNode = head
    while curNode is not None
        print(curNode.item)
        curNode = curNode.next
```

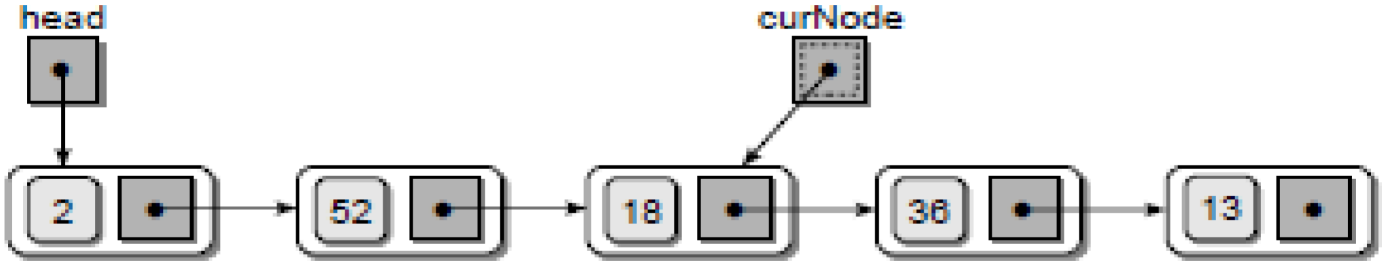
penelusuran mulai dari head



mengikuti pointer next ke node berikutnya



dan ke node selanjutnya



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

class Untaian:
    def __init__(self):
        self.head = None

    def is_empty(self):
        return self.head is None

    #Sisip data di depan
    def insert_at_head(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    #Sisip data di belakang
    def insert_at_tail(self, data):
        new_node = Node(data)
        if self.is_empty():
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
```

```
#Hapus di depan
def delete_at_head(self):
    if self.is_empty():
        return None
    else:
        temp = self.head
        self.head = self.head.next
        temp.next = None
        return temp.data

#Cari data
def search(self, data):
    current = self.head
    while current:
        if current.data == data:
            return True
        current = current.next
    return False

#Menampilkan isi untaian
def traverse(self):
    current = self.head
    while current:
        print(current.data, end=" -> ")
        current = current.next
    print("None")
```

```python
#my_untaian_test.py
from my_untaian import *

def main():
    untaian = Untaian()

    print("periksa apakah untaian kosong?")
    print(untaian.is_empty())  # Output: True

    print("masukkan 3 data di depan")
    untaian.insert_at_head(3)
    untaian.insert_at_head(2)
    untaian.insert_at_head(1)

    print("masukkan 2 data di belakang")
    untaian.insert_at_tail(4)
    untaian.insert_at_tail(5)

    print( "Isi untaian: ")
    untaian.traverse()

    print(" apakah data 3 ada dalam untaian?")
    print(untaian.search(3))  # Output: True
    print(" apakah data 6 ada dalam untaian?")
    print(untaian.search(6))  # Output: False

    print("Menghapus data terdepan ")
    untaian.delete_at_head()
    print("Isi untaian setelah 1 data dihapus")
    untaian.traverse()  # Output: 2 -> 3 -> 4 -> 5 -> None

    print("Menghapus semuanya:")
    while (untaian.is_empty() == False):
        x = untaian.delete_at_head()
        print(x, ' sudah di hapus')

    print("Apakah sudah kosong?:")
    print(untaian.is_empty())

main()
```

Implementasi lebih lengkap

```
#LinkedList.py === @SUARGA
class _LinkedList:
    #A base class untuk representasi linked list.”””
    class _Node:
        #Lightweight, class nonpublic class untuk node untaian.
        __slots__ = '_element' , '_next' # streamline memory

        def __init__(self, element, next): # constructor
            self._element = element # user’s element
            self._next = next       # pointer ke node berikutnya

    def __init__(self):
        #Create an empty list.”””
        self._head = None
        self._tail = None
        self._size = 0 # jumlah awal elemen

    def __len__(self):
```

```
    def __len__(self):
        #memberikan jumlah elemen dalam list.”””
        return self._size

    def is_empty(self):
        #Apabila list kosong mengembalikan True ”””
        return self._size == 0

    def add_first(self,e):
        #Add element at the front
        newest = self._Node(e,self._head)
        self._head = newest
        if self._size == 0:
            self._tail = newest
        self._size += 1

    def add_last(self,e):
        #Add element at the back
```

```
    def add_last(self,e):
        #Add element at the back
        newest = self._Node(e,None)
        self._tail._next = newest
        self._tail = newest
        self._size +=1

    def delete_first(self):
        #delete first element
        if self.is_empty():
            raise Empty('List is empty')
        answer = self._head._element
        self._head = self._head._next
        self._size -= 1
        return answer

    def delete_last(self):
        print('Sorry, we can not remove the last node !')

    def delete_item( self, item ):
```

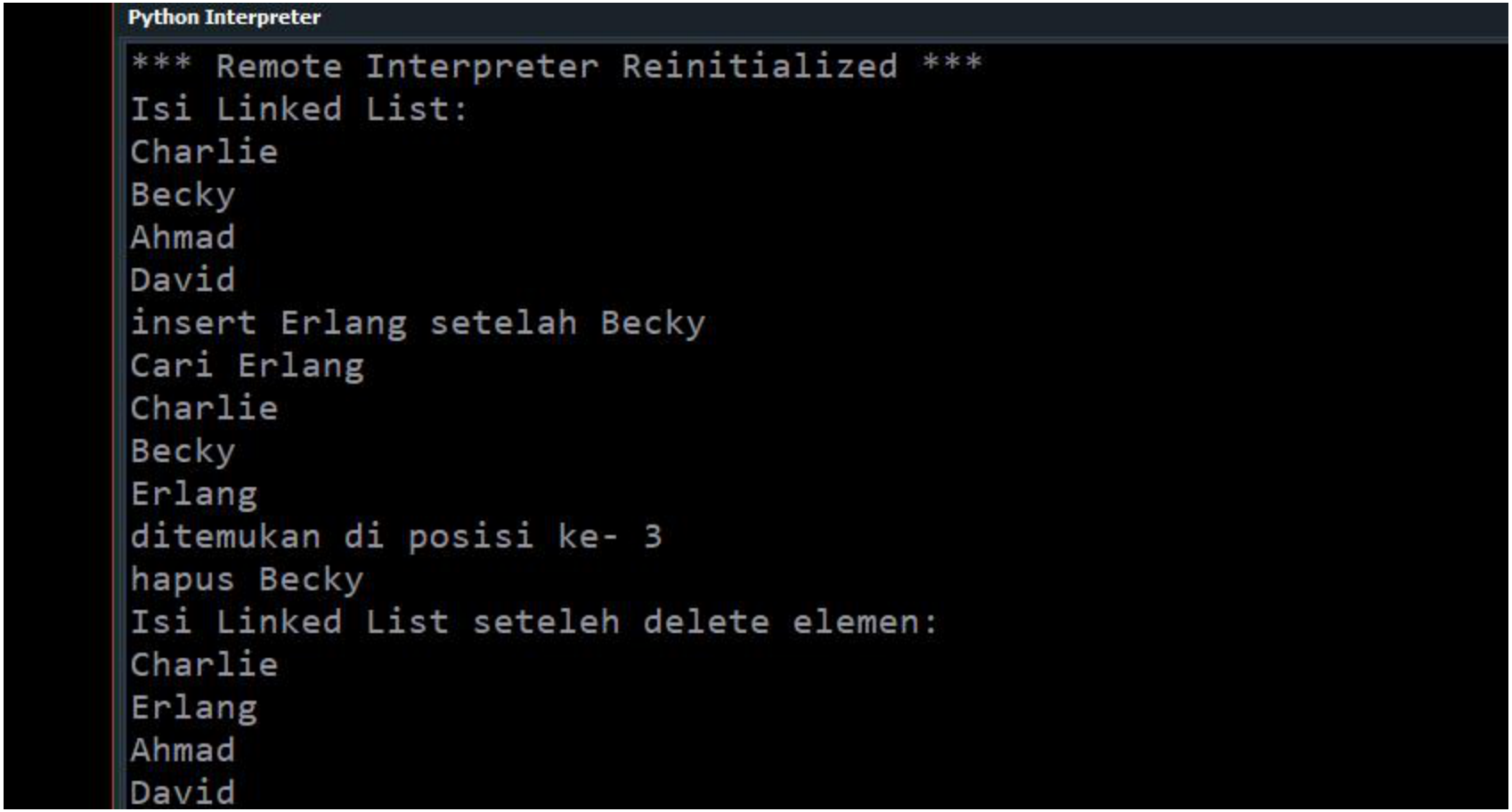
```
    def delete_item( self, item ):
        #delete item from the list
        predNode = None
        curNode = self._head
        while curNode is not None and curNode._element != item :
            predNode = curNode
            curNode = curNode._next

        # item harus ada agar bisa dihapus.
        assert curNode is not None, "Item tidak ada dalam list !!!"

        # Lepaskan node dan berikan nilai itemnya.
        self._size -= 1
        if curNode is self._head :
            self._head = curNode._next
        else :
            predNode._next = curNode._next
        return curNode._element

    def first(self):
        if self.is_empty():
```

```
    def first(self):
        if self.is_empty():
            raise Empty('List is empty')
        return self._head._element

    def last(self):
        if self.is_empty():
            raise Empty('List is empty')
        return self._tail._element

    def search_list(self,e):
        if self.is_empty():
            raise Empty('List is empty')
        x = self._head
        y= x._element
        i=1
        found = False
        while (not found and x != None):
            print(y)
            if (y == e):
                found = True
                break
            else:
```

```
                break
            else:
                x = x._next
                if (x != None):
                    y = x._element
                i+=1

        if (found):
            print('ditemukan di posisi ke-',i)
        else:
            print(e, ' tidak ditemukan dalam list')

    # traversing the list
    def traverse( self ):
        curnode = self._head
        while curnode is not None:
            print(curnode._element)
            curnode = curnode._next

    def insert_after(self, d, e):
        #Add element e after node d and return new node.”””
        x = self._head
        y = x._element
        found = False
        while (not found and x != None):
```

```
        found = False
        while (not found and x != None):
            if (y == d):
                found = True
                z = x._next
                newest = self._Node(e,z)
                x._next = newest
                self._size += 1
                break
            else:
                x = x._next
                if (x != None):
                    y = x._element

        if (found):
            return newest._element
        else:
            print('Node : ',d, ' tdk ada dalam list')
```

```python
#LListUsage.py == pemakaian Linked_List @Suarga
from LinkedList import _LinkedList
def main():
    L = _LinkedList()
    L.add_first("Ahmad")
    L.add_first("Becky")
    L.add_first("Charlie")
    L.add_last("David")
    print("Isi Linked List: ")
    L.traverse()
    print("insert Erlang setelah Becky")
    L.insert_after("Becky", "Erlang")
    print("Cari Erlang")
    L.search_list("Erlang")
    print("hapus Becky")
    L.delete_item("Becky")
    print("Isi Linked List seteleh delete elemen: ")
    L.traverse()
    print("cari Becky")
    L.search_list("Becky")
    print("Coba hapus elemen terakhir")
    L.delete_last()
    print("Isi Linked List")
    L.traverse()
    print("apakah LList kosong? : ", L.is_empty())

main()
```


Python Interpreter
*** Remote Interpreter Reinitialized ***
Isi Linked List:
Charlie
Becky
Ahmad
David
insert Erlang setelah Becky
Cari Erlang
Charlie
Becky
Erlang
ditemukan di posisi ke- 3
hapus Becky
Isi Linked List seteleh delete elemen:
Charlie
Erlang
Ahmad
David

```
cari Becky
Charlie
Erlang
Ahmad
David
Becky  tidak ditemukan dalam list
Coba hapus elemen terakhir
Sorry, we can not remove the last node !
Isi Linked List
Charlie
Erlang
Ahmad
David
apakah LList kosong? :  False
>>>
```

Stack memakai Untaian

- Struktur tumpukan (Stack) dapat di-implementasi menggunakan LinkedList sebagai basisnya.
- Karena penambahan node baru di depan (insert_at_head) menyerupai operasi stack
- Berikut ini implementasi dari “LinkedStack.py”

```python
#LinkedStack.py == @Suarga
class LinkedStack:
    #LIFO Stack implementation using a singly linked list for storage.”””

    #------------------------- 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

    #------------------ stack methods -------------------------
    def __init__(self):
        #”””Create an empty stack.”””
        self._head = None           # reference to the head node
        self._size = 0              # number of stack elements

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

    def is_empty(self):
```

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

    def push(self, e):
        #"""Add element e to the top of the stack."""
        self._head = self._Node(e, self._head) # create and link a new nod
        self._size += 1

    def top(self):
        #"""Return (but do not remove) the element at the top of the stack

        #Raise Empty exception if the stack is empty.

        if self.is_empty():
            raise Empty('Stack is empty')
        return self._head._element

    def pop(self):
```

```
def pop(self):
    #Remove and return the element from the top of the stack (i.e., LIFO).
    #Raise Empty exception if the stack is empty.

    if self.is_empty():
        raise Empty('Stack is empty')
    answer = self._head._element
    self._head = self._head._next         # bypass the former top node
    self._size -= 1
    return answer
```

Contoh Penggunaan

```
>>> s=LinkedStack()
>>> s.push(5)
>>> s.push(3)
>>> print(len(s))
2
>>> print(s.top())
3
>>> s.pop()
3
>>> s.is_empty()
False
>>> s.pop()
5
>>> print(len(s))
0
>>> s.is_empty()
True
>>>
```

Implementasi Queue memakai Untaian

- LinkedList dapat dapat diaplikasikan untuk membuat antrian (queue)
- Karena penambahan di belakang (insert_at_tail) menyerupai antrian.
- Berikut ini listing: LinkedQueue.py

```python
#LinkedQueue.py == @Suarga
class LinkedQueue:
    #FIFO queue implementation using
    #a singly linked list for storage.”””

    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

    # ------------------- method -------------------------------

    def __init__(self):
        #Create an empty queue.”””
        self._head = None
        self._tail = None
        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,but do not remove the element at the front.
        if self.is_empty():
            raise Exception('Queue is empty')
        return self._head._element # front aligned with head of list

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

        if self.is_empty():
            raise Exception('Queue is empty')
        answer = self._head._element
        self._head = self._head._next
        self._size -= 1
        if self.is_empty(): # special case as queue is empty
            self._tail = None # removed head had been the tail
        return answer

    def enqueue(self, e):
```

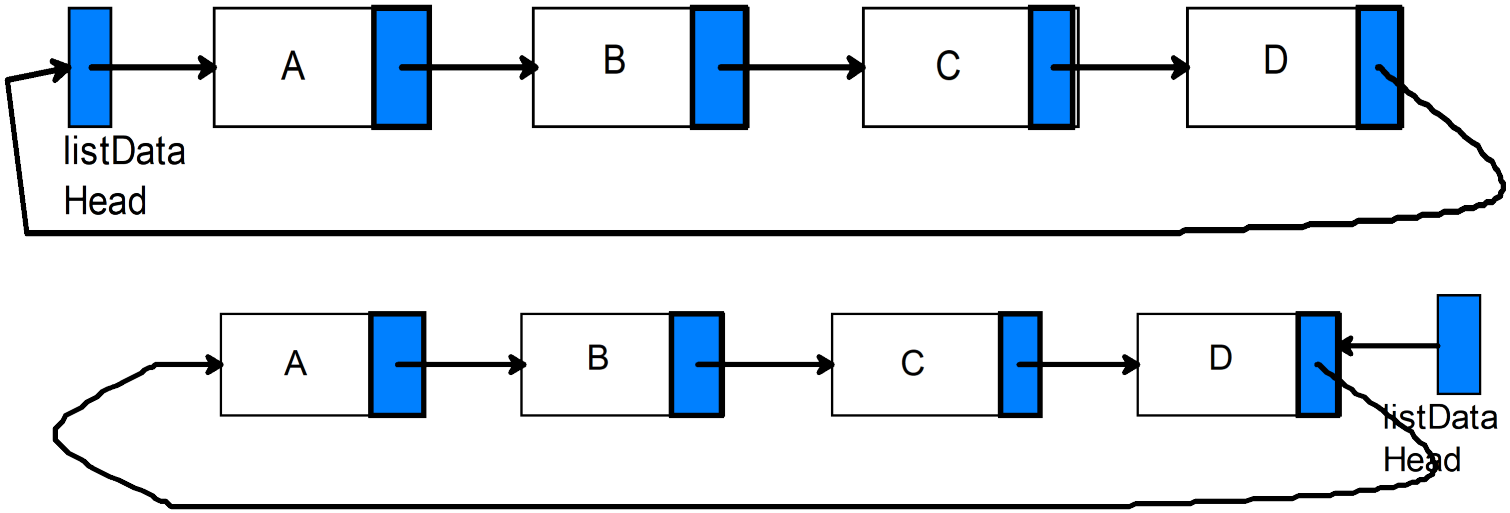
```
    def enqueue(self, e):
        #Add an element to the back of queue.”””
        newest = self._Node(e, None) # node will be new tail node
        if self.is_empty():
            self._head = newest # special case: previously empty
        else:
            self._tail._next = newest
        self._tail = newest # update reference to tail node
        self._size += 1

    def traverse( self ):
        curnode = self._head
        while curnode is not None:
            print(curnode._element)
            curnode = curnode._next
```

Contoh Penggunaan

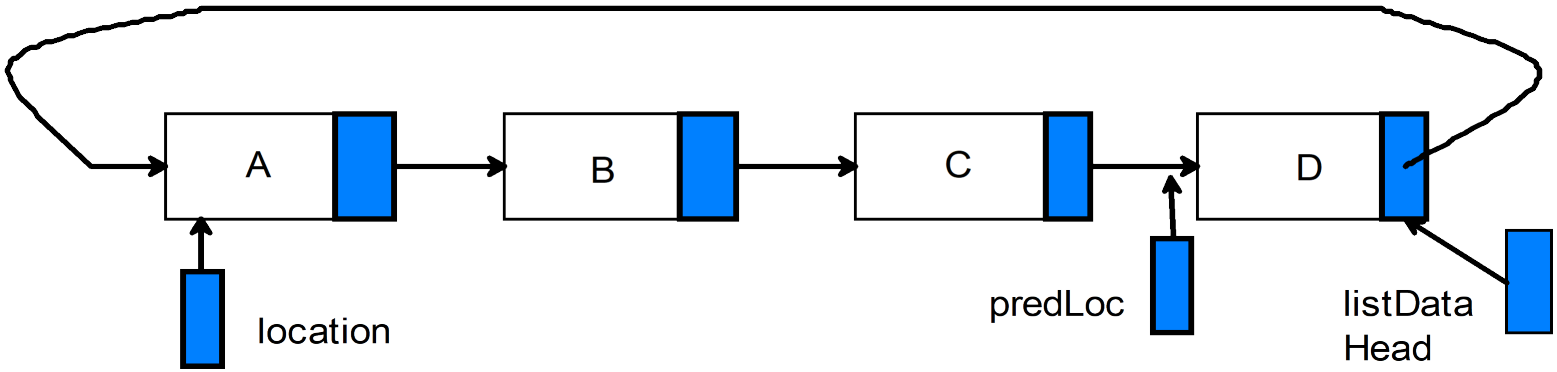
```
>>> q=LinkedQueue()
>>> q.enqueue(10)
>>> q.enqueue(9)
>>> q.enqueue(8)
>>> print(len(q))
3
>>> print(q.first())
10
>>> q.is_empty()
False
>>> q.dequeue()
10
>>> print(len(q))
2
>>> q.dequeue()
9
>>> q.dequeue()
8
>>> q.is_empty()
True
>>>
```

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;
```

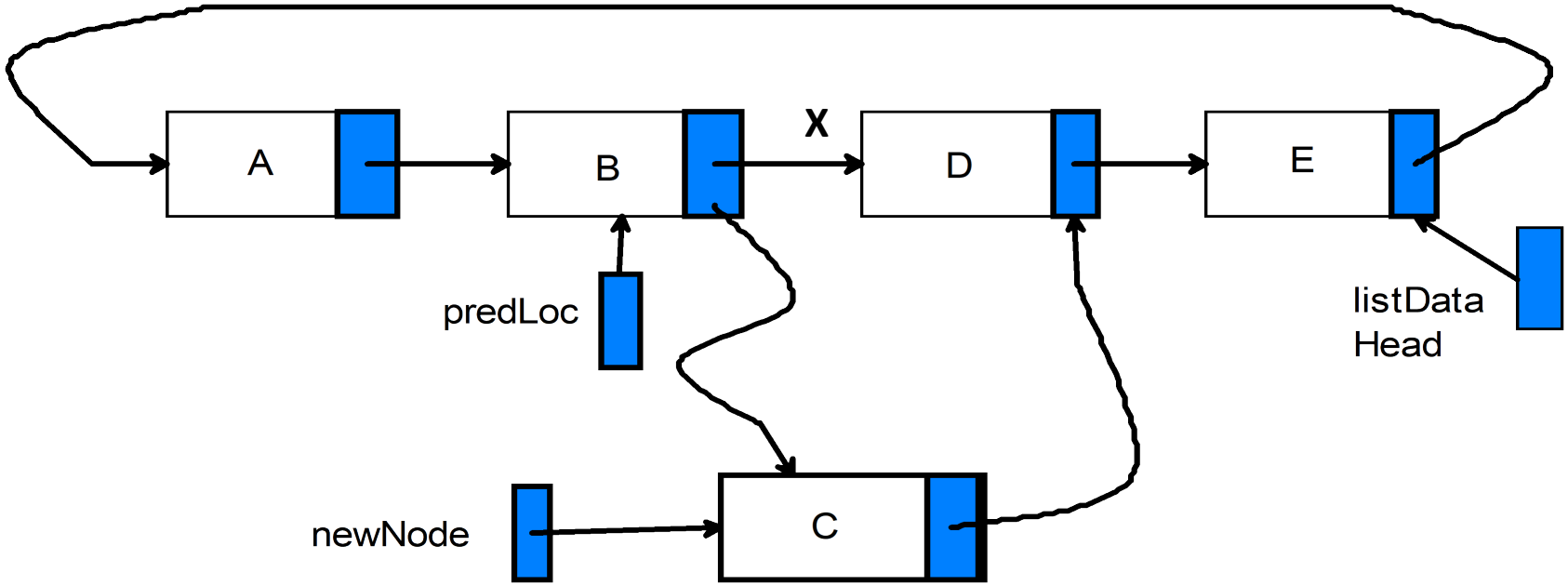
Beberapa fungsi tambahan

- 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
```