



MODUL PERKULIAHAN

TFC251 – Praktikum Struktur Data

Linked List Dalam Python

Penyusun Modul	:	Suamanda Ika Novichasari, M.Kom
Minggu/Pertemuan	:	2
Sub-CPMK/Tujuan Pembelajaran	:	1. Mahasiswa mampu menerapkan konsep Linked List pada bahasa pemrograman python
Pokok Bahasan	:	1. Linked List

**Program Studi Teknologi Informasi (S1)
Fakultas Teknik
Universitas Tidar
Tahun 2024**



Materi 2

LINKED LIST DALAM PYTHON

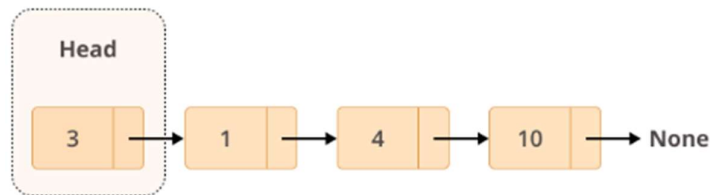
Petunjuk Praktikum :

- Cobalah semua contoh kode program yang terdapat pada semua sub bab dalam modul ini menggunakan google colab.
- Dokumentasikan kegiatan dalam bentuk laporan praktikum sesuai template yang telah ditentukan.
- Setelah selesai mempraktekan semua materi, silakan kerjakan tugas untuk persiapan praktikum pertemuan selanjutnya.

Linked list merupakan sekumpulan objek yang terurut. Perbedaan dengan *list* dapat terlihat dari cara mereka menyimpan elemen di dalam memori. *List* menyimpan referensi data mereka dengan menggunakan blok memori yang berdekatan, sedangkan referensi dalam *linked list* menjadi bagian dari elemen mereka sendiri. Ada 3 macam linked list yang harus diketahui yaitu :

a. Single linked list

Karakteristik dari jenis ini adalah masing-masing node memiliki 2 elemen yaitu data untuk menyimpan nilai dan next untuk menyimpan informasi node selanjutnya.



b. Double Linked list

Karakteristik dari jenis ini adalah masing-masing node memiliki 3 elemen yaitu data untuk menyimpan nilai, prev untuk menyimpan informasi node sebelumnya dan next untuk menyimpan informasi node selanjutnya. Penelusuran node jenis ini dapat dilakukan 2 arah.

Class node memiliki 2 elemen penting yaitu **value** dan **next**. Value digunakan untuk menyimpan sebuah nilai dan next untuk menghubungkan dengan node selanjutnya. Dalam penerapannya elemen value dan next dapat direpresentasikan melalui penggunaan variable dengan nama bebas. Dalam contoh ini menggunakan nama **dataval** dan **nextval**. Melalui code diatas sudah terbentuk 3 node (n1, n2, dan n3).



Untuk menghubungkan node-node tersebut menjadi sebuah *linked list* diperlukan sebuah class lain, dalam contoh ini **class LinkedList**.

▼ Membuat class linked list

```
class LinkedList:
    def __init__(self):
        self.headval = None

Li = LinkedList()
Li.headval = n1

# Link first Node to second node
Li.headval.nextval = n2

# Link second Node to third node
n2.nextval = n3
```

Variable headval digunakan untuk menyimpan node pertama / head. Penamaan variabel headval dapat diganti. Melalui code diatas sudah terbentuk *linked list* yang memiliki 3 node dengan n1 sebagai head, n2 merupakan node setelah head dan n3 merupakan node setelah n2.



2.2. Menelusuri Node dalam Linked List

Linked list yang sudah tercipta sebelumnya merupakan *single linked list*. Menelusuri node pada single linked list hanya dapat dilakukan satu arah yaitu arah maju. Penelusuran dilakukan mulai dari head sampai node terakhir yaitu node yang tidak memiliki nilai next. Penelusuran dapat menggunakan konsep iterasi, dimana elemen next digunakan sebagai acuan menemukan node selanjutnya. Contoh penerapan penelusuran linked list adalah membuat fungsi untuk mencetak semua elemen dalam linked list tersebut. Fungsi `listprint()` pada code dibawah ini merupakan contoh untuk mencetak semua node yang terhubung dalam linked list. Ketika fungsi dijalankan maka akan tercetak dataaval dari masing-masing node secara urut dari head dalam hal ini n1 sampai node terakhir yaitu n3.

```
Menelusuri Linked List

class Node:
    def __init__(self, dataaval=None):
        self.dataaval = dataaval
        self.nextval = None

class LinkedList:
    def __init__(self):
        self.headval = None

    def listprint(self):
        printval = self.headval
        while printval is not None:
            print (printval.dataaval)
            printval = printval.nextval

Li = LinkedList()
Li.headval = n1

# Link first Node to second node
Li.headval.nextval = n2

# Link second Node to third node
n2.nextval = n3

Li.listprint()
```

Januari
Februari
Maret

2.3. Penyisipan node pada linked list

Node dalam linked list sudah tersimpan informasi node selanjutnya, ketika dilakukan penyisipan maka akan mengubah informasi tersebut. Penyisipan dapat dilakukan di awal, tengah dan akhir.

a. Penyisipan di awal

Konsep penyisipan di awal adalah dengan membuat node baru kemudian mengganti nilai nextval node tersebut dengan nilai nextval dari head dan mengganti nilai headval dengan nilai node baru tersebut.

```

  Penyelesaian di awal

class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None

# create node
n1 = Node("Januari")
n2 = Node("Februari")
n3 = Node("Maret")

class LinkedList:
    def __init__(self):
        self.headval = None

# Print the linked list
def listprint(self):
    printval = self.headval
    while printval is not None:
        print (printval.dataval)
        printval = printval.nextval

# create atribut add beginning to Update the new nodes next val to existing node
def AddBeginning(self, newdata):
    NewNode = Node(newdata)
    NewNode.nextval = self.headval
    self.headval = NewNode

l1 = LinkedList()
l1.headval = n1

# Link first Node to second node
l1.headval.nextval = n2

# Link second Node to third node
n2.nextval = n3

l1.AddBeginning("Start")
l1.listprint()

```

Start
Januari
Februari
Maret

b. Penyisipan di tengah

Konsep penyisipan di tengah adalah dengan membuat node baru kemudian mengganti nilai nextval node baru dengan nilai nextval dari node sebelumnya dan mengganti nilai nextval node sebelumnya dengan nilai node baru tersebut.

```

v Penyisipan di tengah
class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None

# create node
n1 = Node("Januari")
n2 = Node("Februari")

class LinkedList:
    def __init__(self):
        self.headval = None

# Print the linked list
def listprint(self):
    printval = self.headval
    while printval is not None:
        print (printval.dataval)
        printval = printval.nextval

# Function to add node in the middle
def AddInbetween(self, mid_node, newdata):
    if mid_node is None:
        print("The mentioned node is absent")
        return
    NewNode = Node(newdata)
    NewNode.nextval = mid_node.nextval
    mid_node.nextval = NewNode

li = LinkedList()
li.headval = n1

# Link first Node to second node
li.headval.nextval = n2

# add node after n1
li.AddInbetween(n1, "Middle")
li.listprint()

```

```

Januari
Middle
Februari

```

c. Penyisipan di akhir

Konsep penyisipan di akhir adalah dengan membuat node baru kemudian mengganti nilai nextval node sebelumnya dengan nilai node baru tersebut. Nilai nextval dari node yang baru dibiarkan kosong sebagai tanda bahwa node tersebut merupakan node terakhir.

```
class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None

# create node
n1 = Node("Januari")
n2 = Node("Februari")

class LinkedList:
    def __init__(self):
        self.headval = None

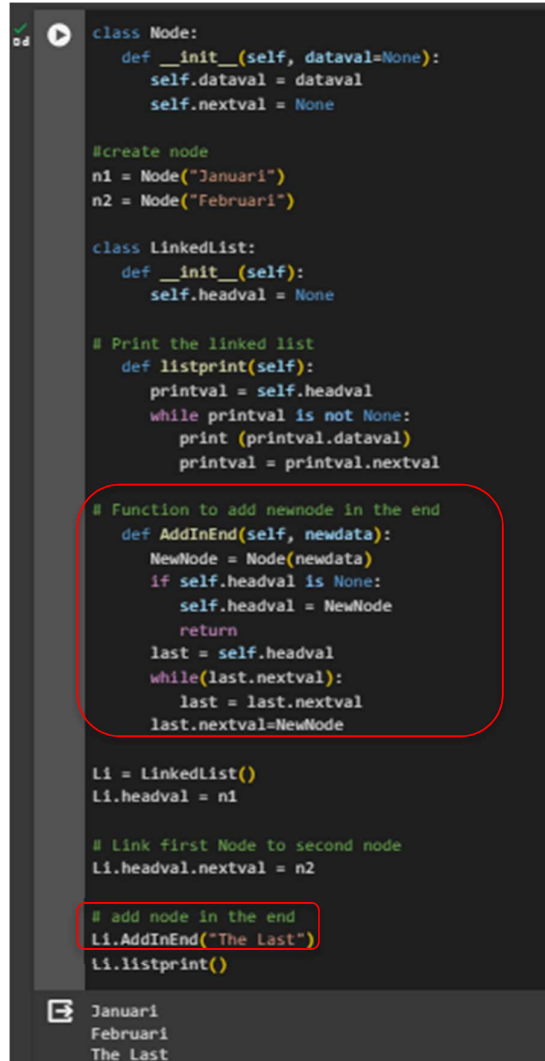
# Print the linked list
def listprint(self):
    printval = self.headval
    while printval is not None:
        print (printval.dataval)
        printval = printval.nextval

# Function to add newnode in the end
def AddInEnd(self, newdata):
    NewNode = Node(newdata)
    if self.headval is None:
        self.headval = NewNode
        return
    last = self.headval
    while (last.nextval):
        last = last.nextval
    last.nextval = NewNode

l1 = LinkedList()
l1.headval = n1

# Link first Node to second node
l1.headval.nextval = n2

# add node in the end
l1.AddInEnd("The Last")
l1.listprint()
```



2.4. Menghapus node pada linked list

Konsep hapus node dilakukan dengan menghapus isi node dan keterkaitan node dengan linked list. Dapat dilakukan dengan 3 cara seperti penyisipan.

1. Hapus node dengan kata kunci

Menghapus node dapat dilakukan dengan menggunakan kata kunci, sehingga node yang terhapus adalah yang nilai dataval sama dengan kata kunci. Untuk melakukan hal tersebut harus melakukan penelusuran untuk mencari dataval yang sama dengan kata kunci. Jika sudah ditemukan maka nextval dari node sebelumnya diganti dengan nextval dari node yang memiliki dataval sama dengan kata kunci.


```

Hapus node

class Node:
    def __init__(self, dataaval=None):
        self.dataaval = dataaval
        self.nextval = None

class LinkedList:
    def __init__(self):
        self.headval = None

# Print the linked list
def listprint(self):
    printval = self.headval
    while printval is not None:
        print (printval.dataaval)
        printval = printval.nextval

# Function to add node in the beginning
def AddBeginning(self, newdata):
    NewNode = Node(newdata)
    # Update the new nodes next val to existing node
    NewNode.nextval = self.headval
    self.headval = NewNode

# Function to remove node by key
def RemoveNode(self, Removekey):
    HeadVal = self.headval
    if (HeadVal is not None):
        if (HeadVal.dataaval == Removekey):
            self.headval = HeadVal.next
            HeadVal = None
            return
        while (HeadVal is not None):
            if HeadVal.dataaval == Removekey:
                break
            prev = HeadVal
            HeadVal = HeadVal.nextval
        if (HeadVal == None):
            return
        prev.nextval = HeadVal.nextval
        HeadVal = None

# create linked list named Li
Li = LinkedList()

# add node
Li.AddBeginning("April")
Li.AddBeginning("Maret")
Li.AddBeginning("Februari")
print("Sebelum dilakukan remove")
Li.listprint()
print("=====")

# remove node by key = "Maret"
Li.RemoveNode("Maret")
print("Setelah dilakukan remove dengan kunci 'Maret'")
Li.listprint()

Sebelum dilakukan remove
Februari
Maret
April
=====
Setelah dilakukan remove dengan kunci 'Maret'
Februari
April

```

2. Hapus node di awal

Hapus node awal dengan cara menghapus headval dan menjadikan node setelah headval menjadi headval seperti contoh berikut.

```

Hapus di awal

[ ] class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None

class LinkedList:
    def __init__(self):
        self.headval = None

# Print the linked list
def listprint(self):
    printval = self.headval
    while printval is not None:
        print (printval.dataval)
        printval = printval.nextval

# Function to add node in the begining
def AddBeginning(self,newdata):
    NewNode = Node(newdata)
    # Update the new nodes next val to existing node
    NewNode.nextval = self.headval
    self.headval = NewNode

# Function to remove first node
def RemoveFirst(self):
    afterhead = self.headval
    self.headval = afterhead.nextval

# create linked list named Li
Li = LinkedList()

# add node
Li.AddBeginning("April")
Li.AddBeginning("Maret")
Li.AddBeginning("Februari")
print("Sebelum dilakukan remove")
Li.listprint()
print("=====")

# remove first node"
Li.RemoveFirst()
print("Setelah dilakukan remove node pertama")
Li.listprint()

Sebelum dilakukan remove
Februari
Maret
April
=====
Setelah dilakukan remove node pertama
Maret
April

```

Dari kode diatas dapat terlihat kemiripan penyisipan di awal dan hapus di awal. Mereka sama-sama berfokus pada headval dan terjadi perubahan headval. Pada penyisipan awal headval diganti dengan node baru, sedangkan di hapus awal headval diganti dengan node selanjutnya.

3. Hapus node di akhir

Sedangkan untuk hapus node akhir dengan cara menghapus nilai nextval pada node sebelum node terakhir, untuk melaksanakan hal tersebut maka harus dilakukan penelusuran untuk mengetahui node terakhir.

Perhatikan contoh dibawah ini, terlihat kemiripan antara penyisipan akhir dan hapus akhir. Keduanya sama-sama diawali dengan penelusuran node-node untuk mencari node terakhir yang nilai nextval nya kosong. Setelah ditemukan node terakhir, pada penyisipan akhir dilakukan dengan mengisi nilai nextval node terakhir dengan node yang baru. Sedangkan untuk hapus akhir, menggunakan variabel prev untuk membantu menyimpan node sebelum node terakhir kemudian nilai nextval pada prev dikosongkan dan nilai node terakhir juga dikosongkan.

▼ Hapus Di akhir

```
class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None

class LinkedList:
    def __init__(self):
        self.headval = None

# Print the linked list
def listprint(self):
    printval = self.headval
    while printval is not None:
        print (printval.dataval)
        printval = printval.nextval

# Function to add newnode in the end
def AddInEnd(self, newdata):
    NewNode = Node(newdata)
    if self.headval is None:
        self.headval = NewNode
        return
    last = self.headval
    while(last.nextval):
        last = last.nextval
    last.nextval=NewNode

# Function to remove first node
def RemoveEnd(self):
    last = self.headval
    while (last is not None):
        if last.nextval == None:
            break
        prev = last
        last = last.nextval
    if (last == None):
        return
    prev.nextval = last.nextval
    last = None

# create linked list named Li
Li = LinkedList()

# add node
Li.AddInEnd("Januari")
Li.AddInEnd("Februari")
Li.AddInEnd("Maret")
print("Sebelum dilakukan remove")
Li.listprint()
print("=====")

# remove last node "
Li.RemoveEnd()
print("Setelah dilakukan remove node terakhir")
Li.listprint()
```

Sebelum dilakukan remove
Januari
Februari
Maret
=====

Setelah dilakukan remove node terakhir
Januari
Februari

LATIHAN 2

1. Buatlah double linked list dengan 3 node didalamnya, kemudian lakukan penyisipan tengah dan hapus dengan kata kunci !
2. Buatlah circular linked list dengan 3 node didalamnya, kemudian lakukan penyisipan akhir dan hapus akhir !

TUGAS 2

Rangkumlah materi tentang penerapan stack dalam python!

Tugas dikumpulkan pada elita maksimal 1 jam sebelum pertemuan selanjutnya.