Praktek 22

```python
# function untuk membuat node
def buat_node(data):
    return {'data': data, 'next': None}

# menambahkan node di akhir list
def tambah_node(head, data):
    new_node = buat_node(data)
    if head is None:
        return new_node
    current = head
    # traversal ke semua elemen list (node)
    while current['next'] is not None:
        current = current['next']
    current['next'] = new_node
    return head

# menampilkan linked-list
def cetak_linked_list(head):
    current = head
    print('Head', end=' → ')
    while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
    print("NULL")

# Contoh Penerapan
# Head awal dari linked-list
head = None

# Tambah node
head = tambah_node(head, 10)
head = tambah_node(head, 11)
head = tambah_node(head, 12)

# cetak linked-list
print('Linked-List : ')
cetak_linked_list(head)
```

‘buat_node’ digunakan untuk membuat sebuah node baru.

‘data’ (nilai yang ingin disimpan di node).

Mengembalikan sebuah dictionary yang merepresentasikan node dengan data dan pointer next yang awalnya diatur ke None.

‘tambah_node’ digunakan untuk menambahkan node baru ke akhir linked list.

Input:

head: Node pertama dari list.

data: Nilai untuk node baru.

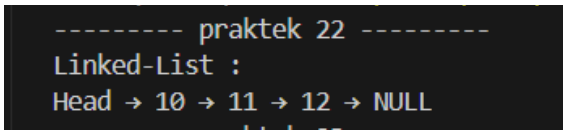
Logika:

Jika head None, berarti list kosong, dan node baru menjadi head.

Jika tidak, traversing dimulai dari head hingga node terakhir, kemudian menghubungkan node terakhir dengan node baru.

Memulai traversal dari head, mencetak setiap **node** hingga mencapai akhir, yang ditunjukkan dengan NULL.

Output



Praktek 23

```python
# function untuk membuat node
def buat_node(data):
    return {'data': data, 'next': None}

# menambahkan node di akhir list
def tambah_node(head, data):
    new_node = buat_node(data)
    if head is None:
        return new_node
    current = head
    while current['next'] is not None:
        current = current['next']
    current['next'] = new_node
    return head

# traversal untuk cetak isi linked-list
def traversal_to_display(head):
    current = head
    print('Head', end=' → ')
    while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
    print("NULL")

# traversal untuk menghitung jumlah elemen dalam linked-list
def traversal_to_count_nodes(head):
    count = 0
    current = head
    while current is not None:
        count += 1
        current = current['next']
    return count

# traversal untuk mencari dimana tail (node terakhir)
def traversal_to_get_tail(head):
    if head is None:
        return None
    current = head
    while current['next'] is not None:
        current = current['next']
    return current

# Penerapan
head = None
head = tambah_node(head, 10)
head = tambah_node(head, 15)
head = tambah_node(head, 117)
head = tambah_node(head, 19)

# cetak isi linked-list
print("Isi Linked-List")
traversal_to_display(head)

# cetak jumlah node
print("Jumlah Nodes = ", traversal_to_count_nodes(head))

# cetak HEAD node
print("HEAD Node : ", head['data'])

# cetak TAIL NODE
print("TAIL Node : ", traversal_to_get_tail(head)['data'])
```

‘def traversal_to_count_nodes(head):

    count = 0

    current = head’

Menghitung jumlah node dalam linked list. Variabel ‘count’ diinisialisasi dengan 0 dan ‘current’ diatur ke ‘head’.

```
‘while current is not None:
    count += 1
    current = current['next']
return count’
```

Mengiterasi list, increment ‘count’ untuk setiap node yang ditemukan, lalu kembalikan ‘count’.

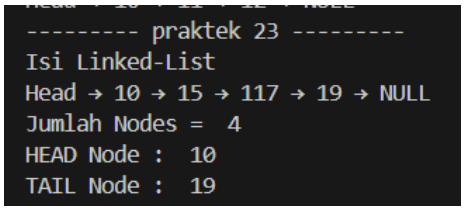
```
‘def traversal_to_get_tail(head):
    if head is None:
        return None’
```

Mencari node terakhir (tail). Jika head adalah None, kembalikan None karena tidak ada node dalam list.

```
‘current = head
while current['next'] is not None:
    current = current['next']
return current’
```

Jika list tidak kosong, fungsi ini mengiterasi sampai menemukan node terakhir dan mengembalikannya.

Output



Praktek 24

```python
# membuat node baru
def sisip_depan(head, data):
    new_node = {'data': data, 'next': head}
    return new_node

# menampilkan linked-list
def cetak_linked_list(head):
    current = head
    print('Head', end=' → ')
    while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
    print("NULL")

# Penerapan membuat linked-list awal
head = None
head = sisip_depan(head, 30)
head = sisip_depan(head, 20)
head = sisip_depan(head, 10)

# cetak isi linked-list awal
print("Isi Linked-List Sebelum Penyisipan di Depan")
cetak = cetak_linked_list(head)

# Penyisipan node
data = 99
head = sisip_depan(head, data)

print("\nData Yang Disispkan : ", data)

# cetak isi setelah penyisipan node baru di awal
print("\nIsi Linked-List Setelah Penyisipan di Depan")
cetak_linked_list(head)
```

Membuat node baru yang berisi data ke node yang menjadi head.

```
‘def sisip_depan(head, data):
    new_node = {'data': data, 'next': head}
    return new_node’
```

new_node adalah **node baru** dengan isi:

'data': data → misalnya 'data': 99

'next': head → node baru akan menunjuk ke **head lama** (misalnya node 10)

new_node langsung **dikembalikan sebagai head yang baru**, jadi dia otomatis berada di depan linked list.

Output

```
--------- praktek 24 ---------
Isi Linked-List Sebelum Penyisipan di Depan
Head → 10 → 20 → 30 → NULL

Data Yang Disispkan :  99

Isi Linked-List Setelah Penyisipan di Depan
Head → 99 → 10 → 20 → 30 → NULL
```

Praktek 25

```
# membuat node baru
def sisip_depan(head, data):
    new_node = {'data': data, 'next': head}
    return new_node

# sisip node diposisi mana saja
def sisip_dimana_aja(head, data, position):
    new_node = {'data': data, 'next': None}

    # cek jika posisi di awal pakai fungsi sisip_depan()
    if position == 0:
        return sisip_depan(head, data)

    current = head
    index = 0


# traversal menuju posisi yang diinginkan dan bukan posisi 0
    while current is not None and index < position - 1:
        current = current['next']
        index += 1

    if current is None:
        print("Posisi melebihi panjang linked list!")
        return head

    # ubah next dari node sebelumnya menjadi node baru
    new_node['next'] = current['next']
    current['next'] = new_node
    return head

## menampilkan linked-list
def cetak_linked_list(head):
    current = head
    print('Head', end=' → ')
    while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
    print("NULL")

# Penerapan
# membuat linked-list awal
head = None
head = sisip_depan(head, 30)
head = sisip_depan(head, 20)
head = sisip_depan(head, 10)
head = sisip_depan(head, 50)
head = sisip_depan(head, 70)

# cetak isi linked-list awal
print("Isi Linked-List Sebelum Penyisipan")
cetak = cetak_linked_list(head)

# Penyisipan node
data = 99
pos = 3
head = sisip_dimana_aja(head, data, pos)

print("\nData Yang Disispkan : ", data)
print("Pada posisi : ", pos, "")

# cetak isi setelah penyisipan node baru di awal
print("\nIsi Linked-List Setelah Penyisipan di tengah")
cetak_linked_list(head)
```

`‘if position == 0:`

`return sisip_depan(head, data)’`

Berarti disisipkan di awal. Pakai fungsi ‘sisip_depan’.

`‘current = head`

`index = 0’`

Mulai dari node awal dan index 0.

`‘while current is not None and index < position - 1:`

```
        current = current['next']
        index += 1’
```

Traversal untuk mencapai **node sebelum posisi target**.

‘data = 99

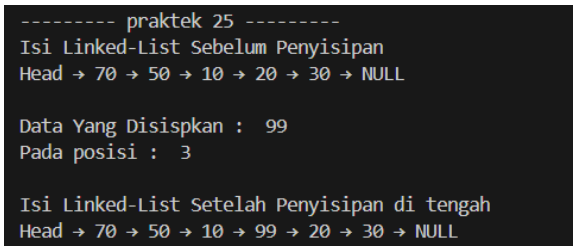
pos = 3

head = sisip_dimana_aja(head, data, pos)’

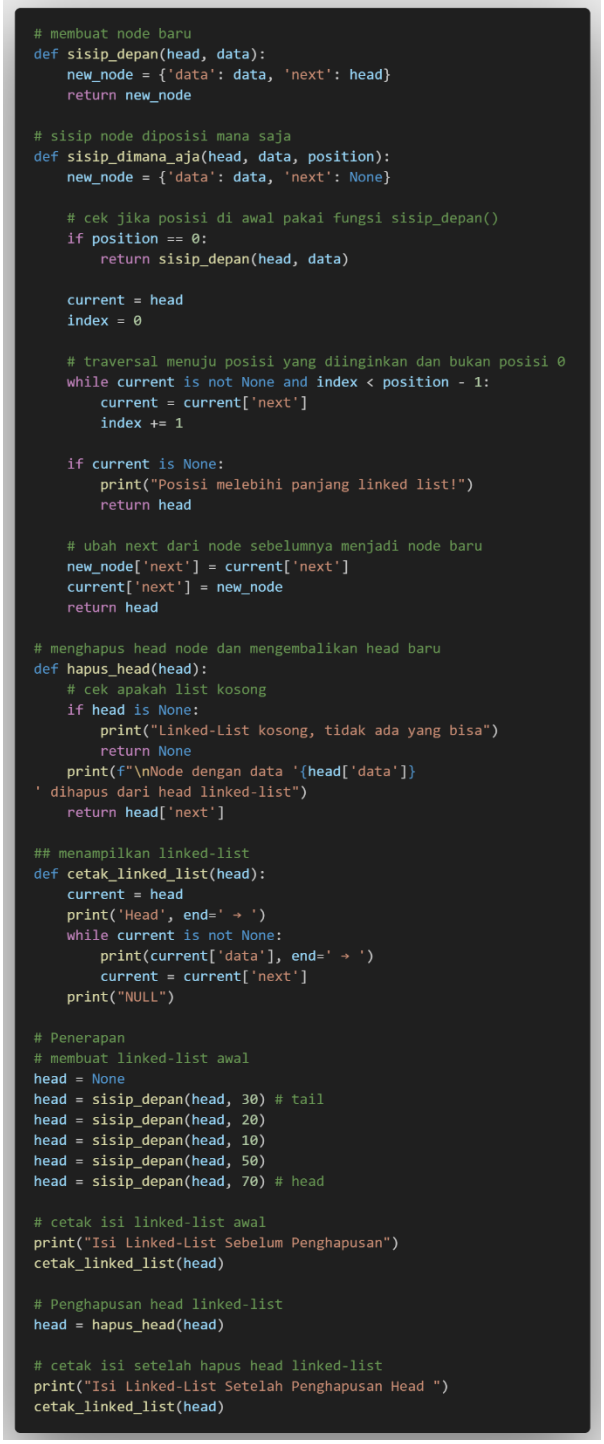
Menyisipkan node dengan data 99 di posisi ke-3.

Artinya: setelah node ke-2 (0-based index), yaitu setelah 10.

Output



Praktek 26



```
# membuat node baru
def sisip_depan(head, data):
    new_node = {'data': data, 'next': head}
    return new_node

# sisip node diposisi mana saja
def sisip_dimana_aja(head, data, position):
    new_node = {'data': data, 'next': None}

    # cek jika posisi di awal pakai fungsi sisip_depan()
    if position == 0:
        return sisip_depan(head, data)

    current = head
    index = 0

    # traversal menuju posisi yang diinginkan dan bukan posisi 0
    while current is not None and index < position - 1:
        current = current['next']
        index += 1

    if current is None:
        print("Posisi melebihi panjang linked list!")
        return head

    # ubah next dari node sebelumnya menjadi node baru
    new_node['next'] = current['next']
    current['next'] = new_node
    return head

# menghapus head node dan mengembalikan head baru
def hapus_head(head):
    # cek apakah list kosong
    if head is None:
        print("Linked-List kosong, tidak ada yang bisa")
        return None
    print(f"\nNode dengan data '{head['data']}
' dihapus dari head linked-list")
    return head['next']

## menampilkan linked-list
def cetak_linked_list(head):
    current = head
    print('Head', end=' → ')
    while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
    print("NULL")

# Penerapan
# membuat linked-list awal
head = None
head = sisip_depan(head, 30) # tail
head = sisip_depan(head, 20)
head = sisip_depan(head, 10)
head = sisip_depan(head, 50)
head = sisip_depan(head, 70) # head

# cetak isi linked-list awal
print("Isi Linked-List Sebelum Penghapusan")
cetak_linked_list(head)

# Penghapusan head linked-list
head = hapus_head(head)

# cetak isi setelah hapus head linked-list
print("Isi Linked-List Setelah Penghapusan Head ")
cetak_linked_list(head)
```

```
‘if position == 0:
    return sisip_depan(head, data)
current = head
index = 0’
```

Menyisipkan di depan, gunakan fungsi ‘sisip_depan’, Mulai dari awal linked list.

Setelah ‘return head['next']’, node lama (yang sebelumnya head) tidak punya referensi lagi (jika tidak disimpan di variabel lain) atau bisa dibilang diskip langsung ke data selanjutnya.

Output

```
--------- praktek 26 ---------
Isi Linked-List Sebelum Penghapusan
Head → 70 → 50 → 10 → 20 → 30 → NULL

Node dengan data '70' dihapus dari head linked-list
Isi Linked-List Setelah Penghapusan Head
Head → 50 → 10 → 20 → 30 → NULL
```

Praktek 27

```
# membuat node baru
def sisip_depan(head, data):
    new_node = {'data': data, 'next': head}
    return new_node

# menghapus head node dan mengembalikan head baru
def hapus_tail(head):
    # cek apakah head node == None
    if head is None:
        print('Linked-List Kosong, tidak ada yang bisa dihapus!')
        return None

    # cek node hanya 1
    if head['next'] is None:
        print(f"Node dengan data '{head['data']}
' dihapus. Linked list sekarang kosong.")
        return None

    current = head
    while current['next']['next'] is not None:
        current = current['next']

    print(f"\nNode dengan data '{current['next']['data']}
' dihapus dari akhir.")
    current['next'] = None
    return head

## menampilkan linked-list
def cetak_linked_list(head):
    current = head
    print('Head', end=' → ')
    while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
    print("NULL")

# Penerapan
# membuat linked-list awal
head = None
head = sisip_depan(head, 30) # tail
head = sisip_depan(head, 20)
head = sisip_depan(head, 10)
head = sisip_depan(head, 50)
head = sisip_depan(head, 70) # head

# cetak isi linked-list awal
print("Isi Linked-List Sebelum Penghapusan")
cetak_linked_list(head)

# Penghapusan tail linked-list
head = hapus_tail(head)

# cetak isi setelah hapus Tail linked-list
print("Isi Linked-List Setelah Penghapusan Tail ")
cetak_linked_list(head)
```

```
def sisip_depan(head, data):
    new_node = {'data': data, 'next': head}
    return new_node
```

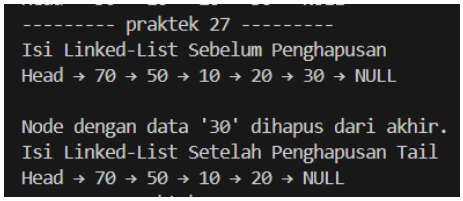
Setiap kali memanggil sisip_depan, data baru selalu menjadi head (paling depan), dan node lama menjadi next dari node baru.

```
    while current['next']['next'] is not None:
        current = current['next']
    print(f"\nNode dengan data '{current['next']['data']}' dihapus dari akhir.")
    current['next'] = None
    return head
```

Loop sampai current adalah node **sebelum tail** (node ke-2 terakhir) ini dicek dengan: current['next']['next'] is not None.

Kemudian di cetak untuk menampilkan data tail yang dihapus sekaligus menghapus node terakhir (jadi None).

Output



Praktek 28

```
# Praktek 28 : Menghapus node di posisi manapun (tengah)
# membuat node baru
def sisip_depan(head, data):
    new_node = {'data': data, 'next': head}
    return new_node

# menghapus head node dan mengembalikan head baru
def hapus_head(head):
    # cek apakah list kosong
    if head is None:
        print("Linked-List kosong, tidak ada yang bisa")
        return None
    print(f"\nNode dengan data '{head['data']}
' dihapus dari head linked-list")
    return head['next']

# menghapus node pada posisi manapun (tengah)
def hapus_tengah(head, position):
    # cek apakah head node == None
    if head is None:
        print('\nLinked-List Kosong, tidak ada yang bisa dihapus!')
        return None

    # cek apakah posisi < 0
    if position < 0:
        print('\nPosisi Tidak Valid')
        return head

    # Cek apakah posisi = 0
    if position == 0:
        print(f"Node dengan data '{head['data']}
' dihapus dari posisi 0.")
        hapus_head(head)
        return head['next']

    current = head
    index = 0

    # cari node sebelum posisi target
    while current is not None and index < position -1:
        current = current['next']
        index += 1

    # Jika posisi yang diinputkan lebih besar dari panjang list
    if current is None or current['next'] is None:
        print("\nPosisi melebih panjang dari linked-list")
        return head

    print(f"\nNode dengan data '{current['next']['data']}
' dihapus dari posisi {position}.")
    current['next'] = current['next']['next']
    return head

## menampilkan linked-list
def cetak_linked_list(head):
    current = head
    print('Head', end=' → ')
    while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
    print("NULL")

# Penerapan
# membuat linked-list awal
head = None
head = sisip_depan(head, 30) # tail
head = sisip_depan(head, 20)
head = sisip_depan(head, 10)
head = sisip_depan(head, 50)
head = sisip_depan(head, 70) # head

# cetak isi linked-list awal
print("Isi Linked-List Sebelum Penghapusan")
cetak_linked_list(head)

# Penghapusan ditengah linked-list
head = hapus_tengah(head, 2)

# cetak isi setelah hapus tengah linked-list
print("\nIsi Linked-List Setelah Penghapusan Tengah ")
cetak_linked_list(head)
```

```
if position == 0:
    print(f"Node dengan data '{head['data']}' dihapus dari posisi 0.")
    hapus_head(head)
    return head['next']
```

Jika posisi 0, berarti kita ingin menghapus head.

Fungsi hapus_head(head) dipanggil untuk cetak info, tapi tidak dipakai hasilnya.

```
current = head
index = 0
while current is not None and index < position -1:
    current = current['next']
    index += 1
```

Menelusuri node dari head sampai ke node di posisi-1.

`current['next']` yaitu node yang akan dihapus.

```
if current is None or current['next'] is None:
    print("\nPosisi melebihi panjang dari linked-list")
    return head
```

Jika tidak menemukan node pada position - 1, maka posisi terlalu besar yang artinya tidak ada penghapusan

```
print(f"\nNode dengan data '{current['next']['data']}' dihapus dari posisi {position}.")
current['next'] = current['next']['next']
```

Cetak data yang dihapus.

Potong referensi ke node target, sehingga node tersebut tidak dirujuk lagi, atau bisa juga Kita lewatin aja node itu, jadi dia nggak kepake lagi dan bakal dibuang otomatis sama Python.

Output