

## Single Linked List

### Praktek 22:

```
# 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

# 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)
```

Hasilnya:

```
Linked-List :  
Head → 10 → 11 → 12 → NULL  
  
** Process exited - Return Code: 0 **  
Press Enter to exit terminal  
|
```

Penjelasannya:

Baris 1: Komentar: Mendefinisikan fungsi untuk membuat node.

Baris 2: Fungsi `buat_node(data)` mengembalikan sebuah dictionary yang mewakili node dengan kunci 'data' dan 'next', di mana 'next' diatur ke None.

Baris 4: Komentar: Mendefinisikan fungsi untuk menambahkan node ke akhir linked list.

Baris 5: Fungsi `tambah_node(head, data)` menerima node awal (head) dan data baru.

Baris 6: Membuat node baru menggunakan fungsi `buat_node`.

Baris 7–8: Jika head bernilai None (linked list masih kosong), node baru akan menjadi head dan dikembalikan.

Baris 9: Jika tidak kosong, inialisasi current sebagai head.

Baris 10–11: Melakukan iterasi sampai menemukan node terakhir (di mana 'next' bernilai None).

Baris 12: Menghubungkan node terakhir dengan node baru.

Baris 13: Mengembalikan head yang telah diperbarui.

Baris 15: Komentar: Mendefinisikan fungsi untuk mencetak isi linked list.

Baris 16: Inialisasi current sebagai head.

Baris 17: Mencetak teks awal Head → sebagai penanda awal list.

Baris 18–20: Selama current tidak kosong, cetak nilai data dan panah →, lalu pindah ke node berikutnya.

Baris 21: Setelah selesai mencetak semua node, cetak NULL sebagai akhir dari linked list.

Baris 23: Komentar: Awal bagian contoh penggunaan fungsi di atas.

Baris 24: Menginisialisasi head dengan None, menandakan linked list masih kosong.

Baris 26–28: Menambahkan tiga node dengan data 10, 11, dan 12 ke dalam linked list. Setiap penambahan node akan mengembalikan head yang diperbarui.

Baris 30: Mencetak label "Linked-List :" ke layar.

Baris 31: Memanggil fungsi `cetak_linked_list(head)` untuk mencetak isi dari linked list yang telah dibuat.

### Praktek 23:

```

# 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)

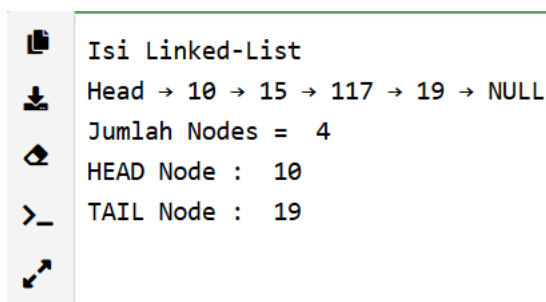
# cetak jumlah node
print("Jumlah Nodes = ", traversal_to_count_nodes(head))

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

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

```

Hasilnya:



```

Isi Linked-List
Head → 10 → 15 → 117 → 19 → NULL
Jumlah Nodes = 4
HEAD Node : 10
TAIL Node : 19

```

Penjelasannya:

Baris 1: Komentar: Mendefinisikan fungsi untuk membuat node.

Baris 2: Fungsi `buat_node(data)` akan mengembalikan dictionary dengan kunci 'data' dan 'next'. Nilai 'next' diset `None` karena node belum dihubungkan ke node lain.

Baris 4: Komentar: Fungsi untuk menambahkan node di akhir linked list.

Baris 5: Membuat node baru dengan data yang diberikan.

Baris 6–7: Jika head masih kosong (linked list belum ada), node baru langsung menjadi head dan dikembalikan.

Baris 8: Jika sudah ada node, variabel `current` diisi dengan head.

Baris 9–10: Melakukan iterasi hingga node terakhir (di mana 'next' bernilai `None`).

Baris 11: Menghubungkan node terakhir ke node baru dengan menyetel 'next'.

Baris 12: Mengembalikan head dari linked list.

Baris 14: Komentar: Fungsi traversal untuk mencetak isi linked list.

Baris 15: Inisialisasi `current` ke node head.

Baris 16: Mencetak teks 'Head → ' sebagai penanda awal linked list.

Baris 17–19: Selama node belum `None`, cetak isi 'data' dan panah, lalu lanjut ke node berikutnya.

Baris 20: Cetak 'NULL' sebagai akhir dari list.

Baris 22: Komentar: Fungsi traversal untuk menghitung jumlah node dalam linked list.

Baris 23: Inisialisasi penghitung `count` dengan nilai 0.

Baris 24: Inisialisasi `current` ke head.

Baris 25–27: Iterasi setiap node, menambah `count`, lalu lanjut ke node berikutnya.

Baris 28: Mengembalikan jumlah total node.

Baris 30: Komentar: Fungsi traversal untuk mendapatkan tail (node terakhir) dari linked list.

Baris 31–32: Jika head kosong (None), kembalikan None.

Baris 33: Inisialisasi current ke head.

Baris 34–35: Iterasi sampai node terakhir ditemukan.

Baris 36: Kembalikan node terakhir.

Baris 38: Komentar: Bagian penerapan kode.

Baris 39–42: Menambahkan empat node dengan nilai 10, 15, 117, dan 19 ke dalam linked list.

Baris 44: Mencetak label "Isi Linked-List" ke layar.

Baris 45: Memanggil fungsi traversal\_to\_display(head) untuk mencetak isi linked list.

Baris 47: Mencetak jumlah node dalam linked list menggunakan traversal\_to\_count\_nodes(head).

Baris 49: Mencetak data dari node head (head['data']).

Baris 51: Mencetak data dari node terakhir (tail) menggunakan fungsi traversal\_to\_get\_tail(head)['data'].

## Praktek 24:

```
# 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 Disisipkan : ", data)

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

```

Hasilnya:

```

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

Data Yang Disisipkan : 99

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

```

Penjelasan:

Baris 1: Komentar: Mendefinisikan fungsi untuk menyisipkan node baru di depan (awal) linked list.

Baris 2: Fungsi `sisip_depan(head, data)` membuat node baru berupa dictionary dengan data yang diisi dan next diarahkan ke head (node lama).

Baris 3: Mengembalikan node baru tersebut sebagai head baru dari linked list.

Baris 5: Komentar: Mendefinisikan fungsi untuk menampilkan isi linked list.

Baris 6: Inisialisasi current ke node head.

Baris 7: Mencetak label "Head" sebagai penanda awal linked list.

Baris 8–10: Selama node belum None, cetak isi datanya dan lanjut ke node berikutnya.

Baris 11: Mencetak teks "NULL" sebagai penanda akhir list.

Baris 13: Komentar: Membuat linked list awal tanpa node (`head = None`).

Baris 14: Menyisipkan node dengan data 30 ke depan linked list.

Baris 15: Menyisipkan node dengan data 20 ke depan linked list.

Baris 16: Menyisipkan node dengan data 10 ke depan linked list.

Baris 18: Mencetak pesan bahwa linked list akan ditampilkan sebelum proses penyisipan.

Baris 19: Menampilkan isi linked list sebelum ada penyisipan data tambahan.

Baris 21: Komentar: Menyisipkan data baru (99) di awal linked list.

Baris 22: Data 99 disisipkan di depan, dan dijadikan head baru.

Baris 24: Mencetak data yang baru saja disisipkan.

Baris 26: Mencetak pesan bahwa isi linked list akan ditampilkan setelah penyisipan.

Baris 27: Menampilkan isi linked list setelah node baru disisipkan di depan.

## 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 Disisipkan : ", 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)


```

Hasilnya:




Isi Linked-List Sebelum Penyisipan

Head → 70 → 50 → 10 → 20 → 30 → NULL



Data Yang Disisipkan : 99

Pada posisi : 3



Isi Linked-List Setelah Penyisipan di tengah

Head → 70 → 50 → 10 → 99 → 20 → 30 → NULL

Penjelasan:

Baris 1: Komentar: Mendefinisikan fungsi untuk menyisipkan node baru di awal linked list.

Baris 2: Fungsi sisip\_depan() menerima head dan data. Membuat node baru dengan data yang diberikan dan next menunjuk ke head sebelumnya.

Baris 3: Mengembalikan node baru sebagai head baru.

Baris 5: Komentar: Mendefinisikan fungsi sisip\_dimana\_aja() untuk menyisipkan node di posisi tertentu.

Baris 6: Membuat node baru dengan data dan next bernilai None.

Baris 8: Mengecek jika posisi penyisipan adalah 0, maka pakai fungsi sisip\_depan().

Baris 9: Mengembalikan hasil dari sisip\_depan() jika posisi adalah awal.

Baris 11: Menginisialisasi current ke head.

Baris 12: Menginisialisasi variabel index sebagai penghitung posisi saat traversal.

Baris 14–16: Perulangan untuk menelusuri linked list hingga posisi sebelum tempat penyisipan.

Baris 17–19: Jika posisi melebihi panjang linked list, cetak peringatan dan kembalikan head.

Baris 21–22: Menyisipkan node baru dengan cara mengubah pointer next dari node sebelumnya.

Baris 23: Mengembalikan head yang telah diperbarui.



Baris 25: Komentar: Fungsi untuk mencetak isi linked list.

Baris 26: Inisialisasi current ke node head.

Baris 27: Mencetak teks 'Head' sebagai tanda awal.

Baris 28–30: Selama belum sampai akhir, cetak data tiap node dan lanjut ke node berikutnya.

Baris 31: Mencetak 'NULL' sebagai akhir dari linked list.

Baris 33: Komentar: Bagian penerapan kode.

Baris 34: Inisialisasi head sebagai None (linked list kosong).

Baris 35–39: Menyisipkan node dengan data 30, 20, 10, 50, dan 70 di awal (jadi urutannya 70 → 50 → 10 → 20 → 30).

Baris 41: Mencetak teks untuk menandakan bahwa isi linked list sebelum penyisipan akan ditampilkan.

Baris 42: Memanggil fungsi cetak untuk menampilkan isi linked list.

Baris 44: Komentar: Bagian penyisipan node baru.

Baris 45: Menentukan data node baru yang akan disisipkan.

Baris 46: Menentukan posisi penyisipan (misalnya posisi ke-3).

Baris 47: Menyisipkan node baru ke posisi ke-3 dalam linked list.

Baris 49–50: Menampilkan data dan posisi penyisipan.

Baris 52: Mencetak teks bahwa isi linked list setelah penyisipan di posisi tengah akan ditampilkan.

Baris 53: Memanggil fungsi cetak untuk menampilkan linked list setelah penyisipan.

## **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)

```

Hasilnya:

```

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

```

Penjelasan:

Baris 1: Komentar: Mendefinisikan fungsi untuk menyisipkan node baru di awal linked list.

Baris 2: Fungsi sisip\_depan() menerima head dan data, lalu membuat node baru yang menunjuk ke head lama.

Baris 3: Mengembalikan node baru sebagai head baru.

Baris 5: Komentar: Fungsi sisip\_dimana\_aja() untuk menyisipkan node di posisi mana pun.

Baris 6: Membuat node baru dengan data dan next bernilai None.

Baris 8–9: Jika posisi yang diinginkan adalah 0 (awal), maka panggil fungsi sisip\_depan() dan kembalikan hasilnya.

Baris 11–12: Inisialisasi pointer current ke head dan index sebagai penghitung posisi saat traversal.

Baris 14–16: Melakukan traversal menuju posisi yang diinginkan (position - 1).

Baris 17–19: Jika pointer current menjadi None, berarti posisi melebihi panjang linked list. Cetak peringatan dan kembalikan head.

Baris 21–22: Menyisipkan node baru dengan menyambungkannya ke node berikutnya dan menyambungkan node sebelumnya ke node baru.

Baris 23: Kembalikan head yang telah diperbarui.

Baris 25: Komentar: Fungsi untuk menghapus node pertama (head).

Baris 26: Mengecek apakah linked list kosong (head = None).

Baris 27–28: Jika kosong, cetak pesan bahwa tidak ada node yang bisa dihapus, dan kembalikan None.

Baris 29: Jika tidak kosong, tampilkan data yang akan dihapus.

Baris 30: Kembalikan head['next'] sebagai head baru.

Baris 32: Komentar: Fungsi untuk mencetak isi linked list.

Baris 33: Inisialisasi pointer current ke head.

Baris 34: Cetak 'Head' sebagai penanda awal.

Baris 35–37: Selama belum mencapai akhir (NULL), cetak isi data tiap node.

Baris 38: Cetak 'NULL' sebagai akhir dari linked list.

Baris 40: Komentar: Penerapan kode dimulai di bawah ini.

Baris 41: Inisialisasi linked list kosong (head = None).

Baris 42–46: Menyisipkan beberapa node di depan sehingga urutan akhir menjadi: 70 → 50 → 10 → 20 → 30.

Baris 48: Menampilkan isi linked list sebelum penghapusan head.

Baris 49: Memanggil fungsi cetak\_linked\_list().

Baris 51: Menghapus node pertama (head) dengan memanggil fungsi hapus\_head().

Baris 53: Menampilkan isi linked list setelah penghapusan head.

Baris 54: Mencetak isi linked list yang sudah diperbarui dengan head baru.

## 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)

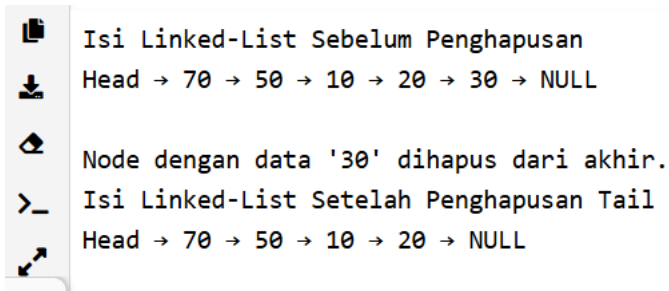
```

```

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

```

Hasilnya:



```

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

Node dengan data '30' dihapus dari akhir.

Isi Linked-List Setelah Penghapusan Tail
Head → 70 → 50 → 10 → 20 → NULL

```

Penjelasan:

Baris 2: Komentar: Fungsi untuk menyisipkan node baru di awal linked list.

Baris 3: Membuat dictionary new\_node yang berisi data dan next yang menunjuk ke node head.

Baris 4: Mengembalikan new\_node sebagai head baru dari linked list.

Baris 7: Komentar: Fungsi untuk menghapus node terakhir (tail) dari linked list.

Baris 8: Mengecek apakah linked list kosong (head == None).

Baris 9: Jika kosong, tampilkan pesan bahwa tidak ada node yang bisa dihapus.

Baris 10: Kembalikan None karena linked list kosong.

Baris 13: Mengecek apakah hanya ada satu node dalam linked list (`head['next'] == None`).

Baris 14: Menampilkan pesan bahwa node tersebut akan dihapus dan linked list menjadi kosong.

Baris 15: Mengembalikan None sebagai linked list kosong setelah penghapusan.

Baris 17: Menyimpan node head ke variabel current untuk traversal.

Baris 18: Melakukan perulangan untuk mencari node kedua terakhir (`current['next']['next'] != None`).

Baris 19: Pindah ke node berikutnya hingga mencapai node kedua terakhir.

Baris 21: Menampilkan pesan bahwa node terakhir akan dihapus.

Baris 22: Memutus sambungan ke node terakhir dengan mengatur `current['next'] = None`.

Baris 23: Mengembalikan head sebagai linked list yang sudah diperbarui.

Baris 26: Komentar: Fungsi untuk mencetak isi linked list.

Baris 27: Menyimpan head ke variabel current untuk traversal.

Baris 28: Mencetak label awal "Head →".

Baris 29: Perulangan untuk mencetak nilai data dari setiap node selama node tidak None.

Baris 30: Pindah ke node berikutnya.

Baris 31: Setelah loop selesai, mencetak "NULL" sebagai akhir dari tampilan linked list.

Baris 34: Komentar: Inisialisasi linked list dengan beberapa node menggunakan fungsi `sisip_depan`.

Baris 35: Menyisipkan node 30 ke depan.

Baris 36: Menyisipkan node 20 ke depan, sekarang urutannya: 20 → 30.

Baris 37: Menyisipkan node 10 ke depan, urutan: 10 → 20 → 30.

Baris 38: Menyisipkan node 50 ke depan, urutan: 50 → 10 → 20 → 30.

Baris 39: Menyisipkan node 70 ke depan, urutan akhir: 70 → 50 → 10 → 20 → 30.

Baris 42: Mencetak label sebelum penghapusan tail.

Baris 43: Memanggil fungsi `cetak_linked_list()` untuk menampilkan isi linked list sebelum dihapus.

Baris 46: Memanggil fungsi `hapus_tail()` untuk menghapus node terakhir dari linked list.

Baris 49: Mencetak label setelah penghapusan tail.

Baris 50: Memanggil `cetak_linked_list()` untuk menampilkan isi linked list setelah penghapusan.

## **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 melebihi 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)

```

Hasilnya:

```

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

Node dengan data '10' dihapus dari posisi 2.

Isi Linked-List Setelah Penghapusan Tengah
Head → 70 → 50 → 20 → 30 → NULL

```

Penjelasan:

Bagian: Membuat Node Baru

Baris 3: Komentar: Fungsi untuk menyisipkan node baru di depan linked list.

Baris 4: Membuat new\_node yang berisi data dan next menunjuk ke head.

Baris 5: Mengembalikan new\_node sebagai head baru.

Bagian: Menghapus Node di Head

Baris 8: Komentar: Fungsi untuk menghapus node paling awal (head).

Baris 9: Mengecek apakah linked list kosong.



Baris 10: Jika kosong, tampilkan pesan dan kembalikan None.

Baris 11: Jika tidak, cetak data node yang dihapus.

Baris 12: Mengembalikan node berikutnya sebagai head baru.

Bagian: Menghapus Node di Tengah

Baris 15: Komentar: Fungsi untuk menghapus node pada posisi tertentu di linked list.

Baris 16: Mengecek apakah linked list kosong.

Baris 17: Jika kosong, cetak pesan dan kembalikan None.

Baris 20: Mengecek apakah posisi yang diminta kurang dari 0.

Baris 21: Jika ya, cetak pesan bahwa posisi tidak valid.

Baris 22: Kembalikan head tanpa perubahan.

Baris 25: Mengecek apakah posisi yang diminta adalah 0 (head).

Baris 26: Jika ya, tampilkan pesan penghapusan.

Baris 27: Memanggil fungsi `hapus_head()` untuk menghapus node pertama.

Baris 28: Kembalikan node berikutnya dari head sebagai head baru.

Baris 30: Menyimpan head ke variabel current.

Baris 31: Inisialisasi index posisi ke 0.

Baris 34–36: Melakukan traversal hingga satu posisi sebelum posisi yang ingin dihapus.

Baris 39: Mengecek apakah current atau `current['next']` kosong (posisi terlalu jauh).

Baris 40: Jika ya, cetak pesan dan kembalikan head.

Baris 43: Cetak data node yang akan dihapus dari posisi tersebut.

Baris 44: Ubah referensi next dari node sebelumnya untuk melewati node yang dihapus.

Baris 45: Kembalikan head yang sudah diperbarui.

Bagian: Menampilkan Isi Linked List

Baris 48: Komentar: Fungsi untuk menampilkan isi linked list.

Baris 49: Menyimpan head ke variabel current.

Baris 50: Mencetak label awal "Head →".

Baris 51–53: Melakukan traversal dan mencetak setiap data dalam node hingga mencapai akhir (NULL).

Bagian: Penerapan

Baris 56–60: Membuat linked list awal dengan menyisipkan node dari belakang ke depan:

- `sisip_depan(head, 30) → jadi tail`
- lalu 20, 10, 50
- terakhir 70 menjadi head
- hasil akhir: `70 → 50 → 10 → 20 → 30`

Baris 63: Mencetak isi linked list sebelum penghapusan.

Baris 66: Menghapus node pada posisi ke-2 (10 akan dihapus karena indeks dimulai dari 0).

Baris 69: Mencetak isi linked list setelah penghapusan node tengah.

- hasil akhir:  $70 \rightarrow 50 \rightarrow 20 \rightarrow 30$