**TUGAS STRUKTUR DATA**

*Tugas Ini Dibuat Guna Memenuhi Tugas Struktur Data*

**Dosen pengampu:**

**Adam bachtiar, s.kom, M.MT**



**Disusun Oleh :**

**M. Damar Alkaromi**

**NIM : 24241045**

**PROGRAM STUDI PENDIDIKAN TEKNOLOGI INFORMASI**

**FAKULTAS SAINS, TEHNIK DAN TERAPAN**

**UNIVERSITAS PENDIDIKAN MANDALIKA MATARAM**

**2025**

1. **PRAKTEK 22**



**BAGIAN 1: MEMBUAT NODE**

# function untuk membuat node

def buat\_node(data):

return {'data': data, 'next': None}

1. def buat\_node(data):  
   ➜ Membuat sebuah **fungsi** bernama buat\_node yang menerima **data** sebagai input.
2. return {'data': data, 'next': None}  
   ➜ Fungsi ini mengembalikan sebuah dictionary yang mewakili satu **node** dalam linked list:
   * 'data': data → menyimpan nilai dari node.
   * 'next': None → node ini belum menunjuk ke node selanjutnya (masih akhir/ujung).

**BAGIAN 2: MENAMBAHKAN NODE DI AKHIR LIST**

# menambahkan node di akhir list

def tambah\_node(head, data):

1. Fungsi tambah\_node menerima:
   * head: node pertama dari linked list.
   * data: nilai baru yang ingin dimasukkan.

new\_node = buat\_node(data)

1. Membuat node baru berisi data yang ingin ditambahkan.

if head is None:

return new\_node

1. Jika list masih kosong (head masih None), maka node baru langsung jadi kepala (head).

current = head

1. Kalau head sudah ada, kita mulai dari awal (current jadi node pertama).

while current['next'] is not None:

current = current['next']

1. Lakukan **perulangan** untuk berjalan ke node berikutnya sampai menemukan node terakhir (yang next-nya None).

current['next'] = new\_node

1. Sambungkan node terakhir ke node baru dengan mengatur 'next'-nya.

return head

1. Kembalikan node awal (head) agar tetap bisa diakses.

**BAGIAN 3: MENAMPILKAN LINKED LIST**

# menampilkan linked-list

def cetak\_linked\_list(head):

1. Fungsi untuk mencetak isi dari linked list, mulai dari head.

current = head

print('Head', end=' → ')

1. Mulai dari head, dan cetak "Head → " sebagai penanda awal.

while current is not None:

print(current['data'], end=' → ')

current = current['next']

1. Selama node belum habis (current tidak None):

* Cetak data di dalam node.
* Lanjut ke node berikutnya (current = current['next']).

print("NULL")

1. Setelah sampai akhir list, cetak NULL sebagai penanda ujung list.

**BAGIAN 4: CONTOH PENGGUNAAN**

# Contoh Penerapan

# Head awal dari linked-list

head = None

1. Awalnya, linked list masih kosong (head belum ada isinya).

# Tambah node

head = tambah\_node(head, 10)

head = tambah\_node(head, 11)

head = tambah\_node(head, 12)

15–17. Tambahkan tiga node ke dalam linked list:

* Pertama berisi 10, jadi kepala.
* Kedua berisi 11, ditambahkan di belakang.
* Ketiga berisi 12, ditambahkan di belakang juga.

# cetak linked-list

print('Linked-List : ')

cetak\_linked\_list(head)

18–19. Cetak seluruh isi dari linked list dari head hingga NULL.

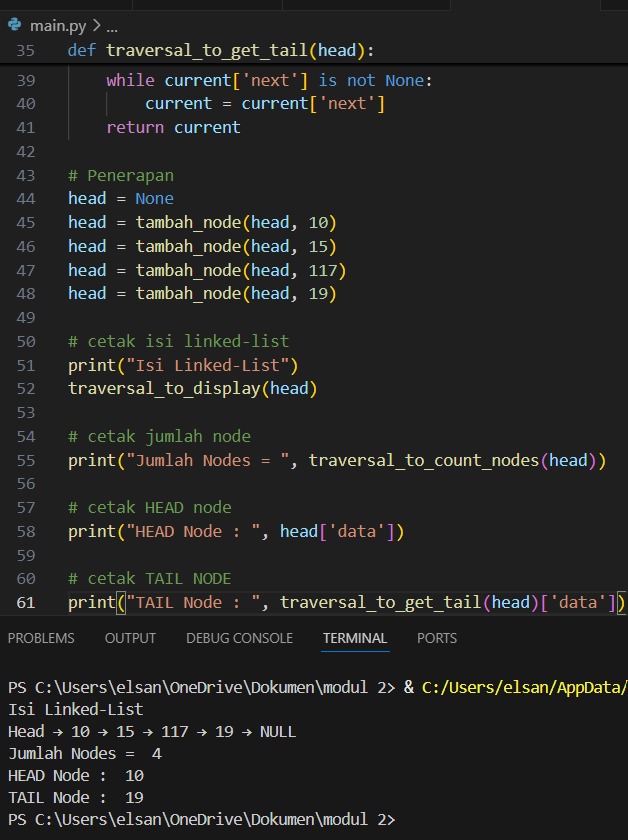
**HASIL OUTPUT YANG AKAN MUNCUL**

Linked-List :

Head → 10 → 11 → 12 → NULL

Kalau kamu ingin menambahkan fitur seperti **hapus node**, **sisip di tengah**, atau ubah ke versi **berbasis class (OOP)**, tinggal bilang saja!

**2. PRAKTEK 23**



**Bagian 1: Membuat Node Baru**

# function untuk membuat node

def buat\_node(data):

return {'data': data, 'next': None}

1. Fungsi buat\_node(data) membuat sebuah **node** (kotak) yang berisi:
   * data: nilainya,
   * next: sambungan ke node berikutnya, awalnya None karena belum terhubung.

**Bagian 2: Menambahkan Node di Akhir**

# menambahkan node di akhir list

def tambah\_node(head, data):

1. Fungsi tambah\_node menerima:
   * head: node pertama dari linked list,
   * data: nilai baru yang ingin dimasukkan ke dalam linked list.

new\_node = buat\_node(data)

1. Buat node baru dengan nilai data.

if head is None:

return new\_node

1. Kalau list masih kosong (head kosong), node baru langsung jadi head.

current = head

while current['next'] is not None:

current = current['next']

1. Kalau head sudah ada, cari node terakhir (yang next-nya None).

current['next'] = new\_node

1. Sambungkan node terakhir dengan node baru.

return head

1. Kembalikan head agar tetap bisa digunakan.

**Bagian 3: Menampilkan Isi Linked List**

# traversal untuk cetak isi linked-list

def traversal\_to\_display(head):

1. Fungsi ini akan **menelusuri dan menampilkan isi** dari linked list.

current = head

print('Head', end=' → ')

1. Mulai dari head, tampilkan tulisan "Head →".

while current is not None:

print(current['data'], end=' → ')

current = current['next']

1. Cetak isi setiap node sampai habis (sampai None).

print("NULL")

1. Tampilkan NULL sebagai akhir list.

**Bagian 4: Menghitung Jumlah Node**

# traversal untuk menghitung jumlah elemen dalam linked-list

def traversal\_to\_count\_nodes(head):

count = 0

1. Buat variabel count untuk menghitung jumlah node.

current = head

while current is not None:

count += 1

current = current['next']

1. Mulai dari head, tambahkan 1 untuk setiap node yang ditemukan.

return count

1. Kembalikan hasil hitungan jumlah node.

**Bagian 5: Mencari Node Terakhir (Tail)**

# traversal untuk mencari dimana tail (node terakhir)

def traversal\_to\_get\_tail(head):

1. Fungsi ini mencari node terakhir (tail).

if head is None:

return None

1. Kalau list kosong, langsung kembalikan None.

current = head

while current['next'] is not None:

current = current['next']

1. Telusuri dari head sampai menemukan node yang next-nya kosong.

return current

1. Kembalikan node terakhir.

**Bagian 6: Penerapan dan Output**

# Penerapan

head = None

1. Awalnya list kosong (head = None).

head = tambah\_node(head, 10)

head = tambah\_node(head, 15)

head = tambah\_node(head, 117)

head = tambah\_node(head, 19)

20–23. Tambahkan 4 node satu per satu:

* 10
* 15
* 117
* 19

Semua disambung jadi satu linked list.

**Bagian 7: Cetak dan Tampilkan Informasi**

# cetak isi linked-list

print("Isi Linked-List")

traversal\_to\_display(head)

24–25. Cetak isi semua node dari awal sampai akhir.

Head → 10 → 15 → 117 → 19 → NULL

# cetak jumlah node

print("Jumlah Nodes = ", traversal\_to\_count\_nodes(head))

1. Tampilkan jumlah total node:

Jumlah Nodes = 4

# cetak HEAD node

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

1. Tampilkan data dari node pertama (head):

HEAD Node : 10

# cetak TAIL NODE

print("TAIL Node : ", traversal\_to\_get\_tail(head)['data'])

1. Tampilkan data dari node terakhir (tail):

TAIL Node : 19

**Kesimpulan:**

Kamu telah membuat:

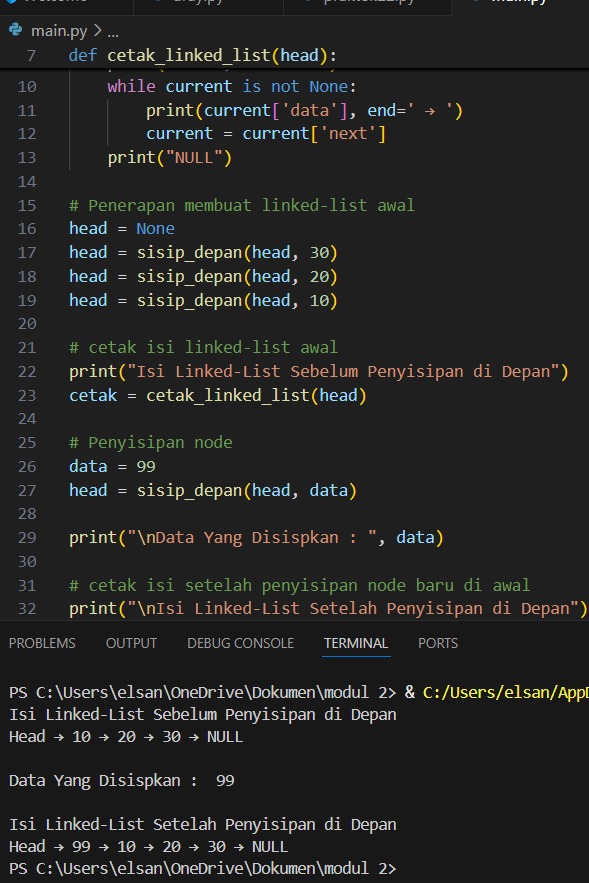
* Fungsi untuk **buat node**,
* Tambah node di akhir,
* **Traversal** untuk:
  + Menampilkan isi,
  + Menghitung jumlah,
  + Menemukan node terakhir.

Semua ini sudah membentuk struktur **single linked list** manual menggunakan dictionary.

Kalau mau lanjut, kamu bisa coba buat fitur:

* Hapus node,
* Sisip di tengah,
* Atau buat versi class (OOP). Siap bantu juga!

1. **PRAKTEK 24**



**Bagian 1: Fungsi Penyisipan di Depan**

# membuat node baru

def sisip\_depan(head, data):

new\_node = {'data': data, 'next': head}

return new\_node

1. def sisip\_depan(head, data):  
   ➜ Mendefinisikan fungsi untuk menyisipkan node baru **di depan** linked list.
2. new\_node = {'data': data, 'next': head}  
   ➜ Membuat node baru:
   * data: berisi nilai yang dimasukkan,
   * next: menunjuk ke head saat ini, agar node baru jadi node pertama (head).
3. return new\_node  
   ➜ Node baru sekarang menjadi kepala (head) dari linked list.

**Bagian 2: Menampilkan Linked List**

# 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")

1. Fungsi cetak\_linked\_list bertugas **menampilkan isi linked list** dari awal (head) hingga akhir (NULL):
   * Mulai dari head,
   * Cetak setiap isi node (data),
   * Berjalan ke node berikutnya hingga current menjadi None.

**Bagian 3: Penerapan – Membuat Linked List Awal**

# Penerapan membuat linked-list awal

head = None

1. Awalnya list kosong (head = None).

head = sisip\_depan(head, 30)

head = sisip\_depan(head, 20)

head = sisip\_depan(head, 10)

1. Tambahkan node satu per satu **di depan**:
   * Tambah 30: head jadi node 30 → NULL,
   * Tambah 20: head jadi node 20 → 30 → NULL,
   * Tambah 10: head jadi node 10 → 20 → 30 → NULL.

**Bagian 4: Cetak Linked List Sebelum Penyisipan**

# cetak isi linked-list awal

print("Isi Linked-List Sebelum Penyisipan di Depan")

cetak = cetak\_linked\_list(head)

1. Cetak isi linked list sebelum ada penyisipan baru:

Head → 10 → 20 → 30 → NULL

**Bagian 5: Penyisipan Node Baru di Depan**

# Penyisipan node

data = 99

head = sisip\_depan(head, data)

1. Menyisipkan nilai baru 99 ke paling depan:
   * head sekarang menjadi 99 → 10 → 20 → 30 → NULL.

**Bagian 6: Tampilkan Data yang Disisipkan**

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

1. Cetak nilai 99 yang baru saja disisipkan.

**Bagian 7: Cetak Linked List Setelah Penyisipan**

# cetak isi setelah penyisipan node baru di awal

print("\nIsi Linked-List Setelah Penyisipan di Depan")

cetak\_linked\_list(head)

1. Cetak ulang isi linked list **setelah** disisipkan:

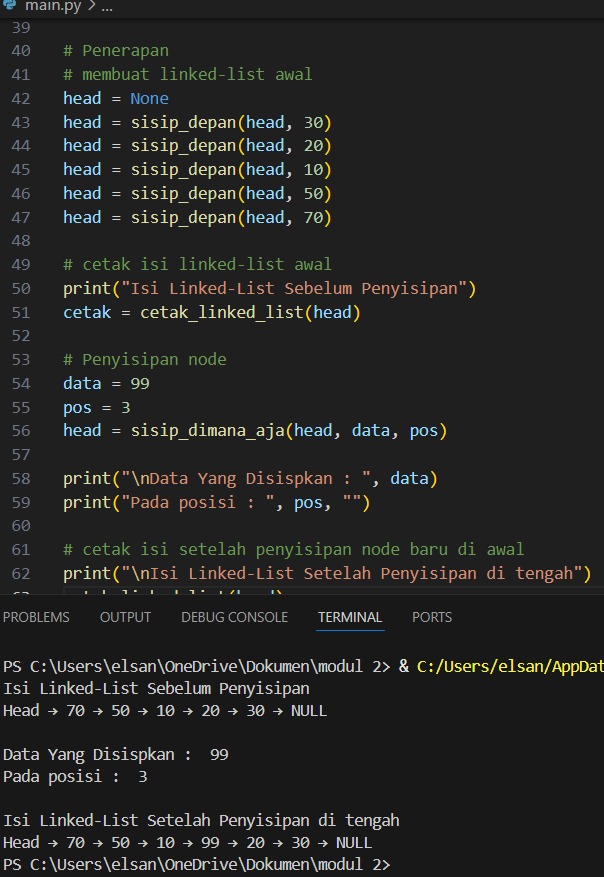
Head → 99 → 10 → 20 → 30 → NULL

**Kesimpulan**

Kode ini memperlihatkan:

* Cara **menyisipkan node di awal** linked list,
* Cara menampilkan seluruh isi list dari head ke tail,
* Hasil penyisipan terlihat langsung dari perbandingan **sebelum dan sesudah**.

1. **PRAKTEK 25**



**BAGIAN 1: Fungsi Menyisipkan Node di Depan**

def sisip\_depan(head, data):

new\_node = {'data': data, 'next': head}

return new\_node

1. **sisip\_depan(head, data)**: fungsi untuk menambahkan node di **paling depan**.
2. Membuat node baru (new\_node) berisi:
   * data: nilai yang diberikan,
   * next: menunjuk ke node pertama saat ini (head).
3. Node baru dikembalikan dan menjadi head yang baru.

**BAGIAN 2: Fungsi Menyisipkan Node di Posisi Tertentu**

def sisip\_dimana\_aja(head, data, position):

new\_node = {'data': data, 'next': None}

1. Fungsi sisip\_dimana\_aja() akan menyisipkan node di **posisi yang ditentukan** (bukan hanya di awal).
2. Buat new\_node berisi data, dan next awalnya kosong (None).

if position == 0:

return sisip\_depan(head, data)

1. Kalau posisi yang diinginkan adalah 0 (di depan), pakai fungsi sisip\_depan() saja.

current = head

index = 0

1. Siapkan current untuk menelusuri list, mulai dari head.
2. Gunakan index untuk mencatat posisi saat ini.

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

current = current['next']

index += 1

1. Loop berjalan untuk menemukan node **sebelum posisi yang dituju**.
   * Misal posisi yang dituju = 3, maka loop berhenti di node ke-2 (index = 2).

if current is None:

print("Posisi melebihi panjang linked list!")

return head

1. Jika posisi terlalu besar (melebihi panjang list), cetak pesan peringatan dan **jangan sisipkan** apa pun.

new\_node['next'] = current['next']

current['next'] = new\_node

1. Sambungkan new\_node ke node setelahnya,
2. Lalu, sambungkan node sebelumnya (current) ke new\_node.  
   ➜ Proses sisip selesai.

return head

1. Kembalikan head agar list tetap utuh.

**BAGIAN 3: 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")

1. Fungsi ini akan **menampilkan isi linked list dari depan hingga akhir**.
2. Loop mencetak setiap node sampai current menjadi None.

**BAGIAN 4: Penerapan – Membuat Linked List Awal**

# 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)

1. Awalnya linked list kosong (head = None).  
   17–21. Tambahkan 5 node satu per satu di depan:

* 70 → 50 → 10 → 20 → 30 → NULL

**BAGIAN 5: Cetak Linked List Sebelum Penyisipan**

print("Isi Linked-List Sebelum Penyisipan")

cetak = cetak\_linked\_list(head)

1. Cetak isi linked list **sebelum penyisipan node baru**.

**BAGIAN 6: Proses Penyisipan**

data = 99

pos = 3

head = sisip\_dimana\_aja(head, data, pos)

1. Siapkan data 99 untuk disisipkan.
2. Tentukan posisi (pos = 3), artinya data 99 akan disisipkan setelah node ke-2 (pada index ke-3).
3. Panggil sisip\_dimana\_aja() untuk menyisipkan node tersebut.

**BAGIAN 7: Tampilkan Info Penyisipan**

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

print("Pada posisi : ", pos, "")

26–27. Tampilkan nilai yang disisipkan dan posisinya.

**BAGIAN 8: Cetak Linked List Setelah Penyisipan**

print("\nIsi Linked-List Setelah Penyisipan di tengah")

cetak\_linked\_list(head)

28–29. Cetak isi list setelah penyisipan:  
Jika sebelumnya:

Head → 70 → 50 → 10 → 20 → 30 → NULL

Maka sesudah penyisipan 99 di posisi ke-3:

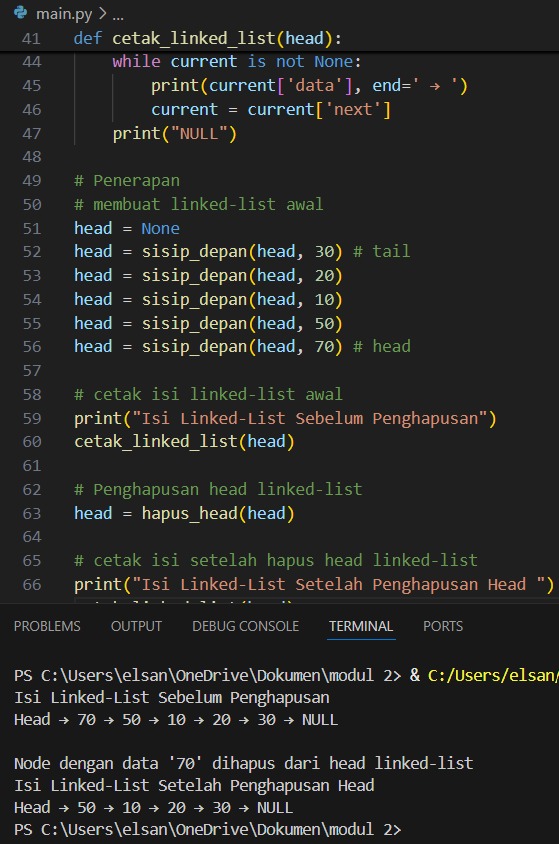
Head → 70 → 50 → 10 → 99 → 20 → 30 → NULL

**KESIMPULAN**

Fungsi sisip\_dimana\_aja() bisa menyisipkan node:

* Di awal (posisi 0),
* Di tengah mana saja,
* Dan menolak jika posisi terlalu besar.

**5. PRAKTEK 26**



**BAGIAN 1: Fungsi sisip\_depan**

def sisip\_depan(head, data):

new\_node = {'data': data, 'next': head}

return new\_node

1. Fungsi sisip\_depan() digunakan untuk **menyisipkan node di depan**.
2. new\_node adalah dictionary (objek node) yang menyimpan:
   * 'data': nilai yang diberikan,
   * 'next': menunjuk ke head lama (node sebelumnya).
3. Fungsi mengembalikan node baru yang menjadi head sekarang.

**BAGIAN 2: Fungsi sisip\_dimana\_aja**

def sisip\_dimana\_aja(head, data, position):

new\_node = {'data': data, 'next': None}

1. Membuat node baru (new\_node) untuk disisipkan di posisi tertentu.

if position == 0:

return sisip\_depan(head, data)

1. Jika posisi yang diminta adalah 0, langsung gunakan fungsi sisip\_depan().

current = head

index = 0

1. Gunakan variabel current untuk menyusuri node, dan index untuk menghitung posisi.

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

current = current['next']

index += 1

1. Loop ini akan berjalan hingga current berada **sebelum** posisi yang dituju.
   * Misalnya position = 3, maka current akan berada di posisi ke-2 (karena index < 2).

if current is None:

print("Posisi melebihi panjang linked list!")

return head

1. Jika posisi melebihi jumlah node dalam list, tampilkan pesan dan **jangan lakukan penyisipan**.

new\_node['next'] = current['next']

current['next'] = new\_node

return head

1. Hubungkan new\_node ke node setelah current.
2. Lalu hubungkan current ke new\_node.
3. Return head agar linked list tetap utuh.

**BAGIAN 3: Fungsi hapus\_head**

def hapus\_head(head):

if head is None:

print("Linked-List kosong, tidak ada yang bisa")

return None

1. Fungsi hapus\_head() akan menghapus node paling depan.
2. Cek dulu: jika head kosong (linked list kosong), cetak pesan dan kembalikan None.

print(f"\nNode dengan data '{head['data']}' dihapus dari head linked-list")

return head['next']

1. Cetak data yang dihapus.
2. Kembalikan head['next'], artinya **node kedua jadi head baru**.

**BAGIAN 4: Fungsi cetak\_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")

1. Fungsi untuk **menampilkan isi linked list** dari depan sampai akhir.
2. Gunakan loop untuk cetak satu per satu data node hingga habis (None).

**BAGIAN 5: Penerapan (Main Program)**

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

1. Awalnya, head = None (linked list kosong).
2. Tambahkan node dari belakang ke depan (karena pakai sisip\_depan()):

* Hasil akhir:
* Head → 70 → 50 → 10 → 20 → 30 → NULL

print("Isi Linked-List Sebelum Penghapusan")

cetak\_linked\_list(head)

1. Cetak isi linked list **sebelum node pertama dihapus**.

head = hapus\_head(head)

1. Hapus node paling depan (70), dan head sekarang menunjuk ke 50.

print("Isi Linked-List Setelah Penghapusan Head ")

cetak\_linked\_list(head)

1. Cetak isi linked list **setelah node head dihapus**.

**OUTPUT YANG DITAMPILKAN**

Isi Linked-List Sebelum Penghapusan

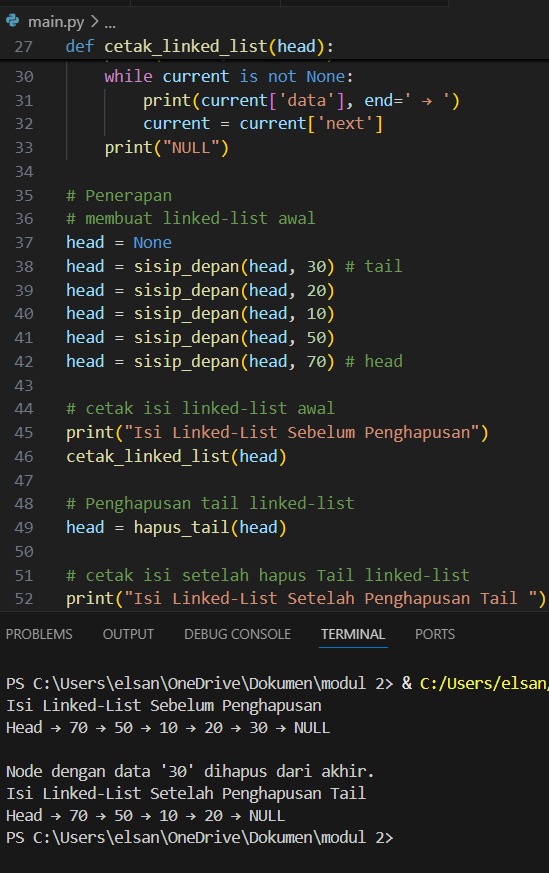
Head → 70 → 50 → 10 → 20 → 30 → NULL

Node dengan data '70' dihapus dari head linked-list

Isi Linked-List Setelah Penghapusan Head

Head → 50 → 10 → 20 → 30 → NULL

1. **PRAKTEK 27**



FUNGSI sisip\_depan()

def sisip\_depan(head, data):

new\_node = {'data': data, 'next': head}

return new\_node

1. Fungsi ini menyisipkan node di depan dari linked list.
2. data adalah nilai yang ingin disimpan.
3. Node baru (new\_node) akan menunjuk ke head lama.
4. Fungsi mengembalikan node baru sebagai head yang baru.
5. FUNGSI hapus\_tail()

def hapus\_tail(head):

1. Fungsi ini untuk menghapus node paling akhir (tail).

if head is None:

print('Linked-List Kosong, tidak ada yang bisa dihapus!')

return None

1. Jika linked list kosong (head = None), tampilkan pesan dan kembalikan None.

if head['next'] is None:

print(f"Node dengan data '{head['data']}' dihapus. Linked list sekarang kosong.")

return None

1. Jika hanya ada satu node saja, maka node itu dihapus dan linked list jadi kosong.

current = head

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

current = current['next']

1. current digunakan untuk menelusuri node.
2. Loop ini berjalan hingga current berada di node sebelum tail (dua langkah sebelum None).

print(f"\nNode dengan data '{current['next']['data']}' dihapus dari akhir.")

current['next'] = None

return head

1. Cetak node mana yang dihapus.
2. Putuskan koneksi ke node terakhir (current['next'] = None) — sekarang dia menjadi tail.
3. Kembalikan head supaya linked list tetap bisa diakses.

FUNGSI cetak\_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")

1. Menampilkan seluruh isi linked list dari awal hingga akhir (NULL).
2. Gunakan loop untuk cetak data dari setiap node satu per satu.

PENERAPAN (MAIN PROGRAM)

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

1. Membuat linked list dengan data 70 → 50 → 10 → 20 → 30.
   * Urutannya dari belakang karena disisipkan di depan.
   * Jadi 70 adalah head, 30 adalah tail.

print("Isi Linked-List Sebelum Penghapusan")

cetak\_linked\_list(head)

1. Menampilkan isi linked list sebelum dilakukan penghapusan tail.

head = hapus\_tail(head)

1. Menghapus node terakhir (30) dari linked list.

print("Isi Linked-List Setelah Penghapusan Tail ")

cetak\_linked\_list(head)

1. Menampilkan linked list setelah node tail dihapus.

HASIL YANG DITAMPILKAN

Misalnya hasilnya seperti ini:

Isi Linked-List Sebelum Penghapusan

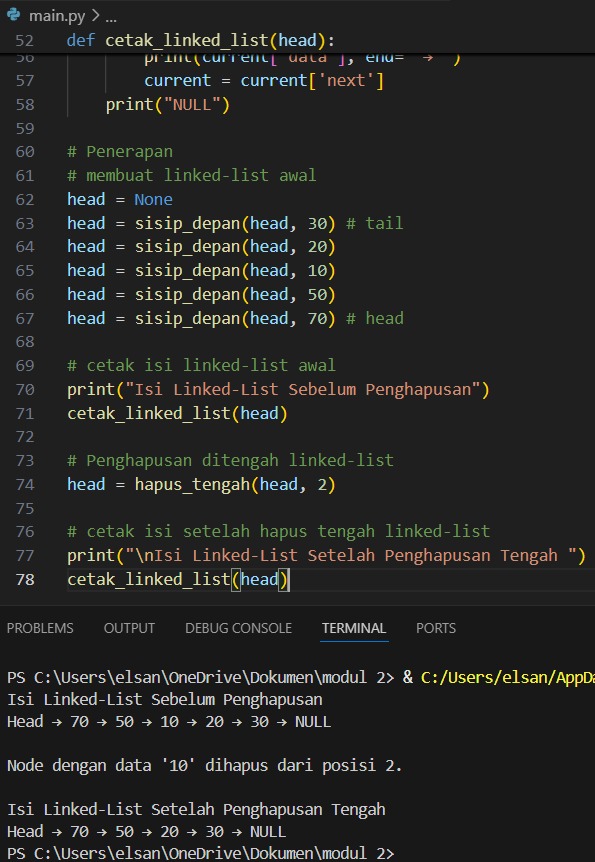
Head → 70 → 50 → 10 → 20 → 30 → NULL

Node dengan data '30' dihapus dari akhir.

Isi Linked-List Setelah Penghapusan Tail

Head → 70 → 50 → 10 → 20 → NULL

1. **PRAKTEK 28**



**FUNGSI sisip\_depan(head, data)**

def sisip\_depan(head, data):

new\_node = {'data': data, 'next': head}

return new\_node

1. Membuat **node baru** dengan data.
2. next menunjuk ke head yang lama.
3. Node baru dikembalikan sebagai head baru.

**FUNGSI hapus\_head(head)**

def hapus\_head(head):

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']

1. Menghapus **node pertama (head)**.
2. Jika kosong, tampilkan pesan dan kembalikan None.
3. Jika tidak kosong, tampilkan data yang dihapus, lalu kembalikan node setelah head.

**FUNGSI hapus\_tengah(head, position)**

def hapus\_tengah(head, position):

1. Fungsi ini digunakan untuk **menghapus node di posisi tertentu** (tengah).

if head is None:

print('\nLinked-List Kosong, tidak ada yang bisa dihapus!')

return None

1. Jika linked list kosong, tampilkan pesan dan keluar dari fungsi.

if position < 0:

print('\nPosisi Tidak Valid')

return head

1. Posisi tidak boleh negatif.

if position == 0:

print(f"Node dengan data '{head['data']}' dihapus dari posisi 0.")

hapus\_head(head)

return head['next']

1. Jika posisi adalah 0, berarti kita ingin hapus head. Panggil hapus\_head().

Catatan penting: hapus\_head(head) sudah mengembalikan head['next'], jadi baris return head['next'] ini **tidak tepat**, seharusnya cukup:

return hapus\_head(head)

current = head

index = 0

1. Siapkan variabel untuk traversing ke node sebelum node yang mau dihapus.

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

current = current['next']

index += 1

1. Loop untuk mencari node **sebelum posisi target**.

if current is None or current['next'] is None:

print("\nPosisi melebih panjang dari linked-list")

return head

1. Cek apakah posisi melebihi panjang list.

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

current['next'] = current['next']['next']

return head

1. Hapus node di posisi tersebut dengan **melewatkan** node itu.
2. Kembalikan head.

**FUNGSI cetak\_linked\_list(head)**

def cetak\_linked\_list(head):

current = head

print('Head', end=' → ')

while current is not None:

print(current['data'], end=' → ')

current = current['next']

print("NULL")

1. Menampilkan isi linked list dari awal sampai akhir.

**PENERAPAN**

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

1. Membuat linked list seperti ini:  
   70 → 50 → 10 → 20 → 30

print("Isi Linked-List Sebelum Penghapusan")

cetak\_linked\_list(head)

1. Tampilkan isi sebelum penghapusan.

head = hapus\_tengah(head, 2)

1. Hapus node di posisi ke-2 (yaitu node dengan nilai 10).

print("\nIsi Linked-List Setelah Penghapusan Tengah ")

cetak\_linked\_list(head)

1. Cetak isi linked list setelah penghapusan.

**OUTPUT YANG DITAMPILKAN**

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

**PENINGKATAN YANG DISARANKAN**

Di bagian ini:

if position == 0:

print(f"Node dengan data '{head['data']}' dihapus dari posisi 0.")

hapus\_head(head)

return head['next']

Harusnya cukup:

if position == 0:

return hapus\_head(head)