Nama : Muhamad Rio Abdul Talib

Nim : 24241056

Kelas : PTI-B

Penjelasan Baris per Baris: Linked List dalam Python

1. Fungsi buat_node(data)

```
def buat_node(data):
    return {'data': data, 'next': None}
```

Membuat satu node berbentuk dictionary dengan dua kunci: 'data' menyimpan nilai, 'next' menunjuk ke node berikutnya (awalnya None).

2. Fungsi tambah_node(head, data)

```
def tambah_node(head, data):
    new_node = buat_node(data)
    if head is None:
        return new_node
```

Menambahkan node baru di akhir linked list.Jika head kosong, node baru menjadi head.

3. Fungsi cetak_linked_list(head)

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

Menampilkan isi linked list dari head ke tail dengan format visual: data → data → NULL.

4. Fungsi traversal_to_count_nodes(head)

```
    current['next'] = new_node
    return head
```

Menghitung jumlah node dalam linked list dengan traversing satu per satu.

5. 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")
```

Menampilkan isi linked list dari head sampai tail dalam format visual → data → data → NULL.

6. Traversal Tambahan

```
def traversal_to_count_nodes(head):
    count = 0
    current = head
    while current is not None:
        count += 1
        current = current['next']
    return count
```

Menghitung jumlah node dalam linked list.

7.

```
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
```

Mendapatkan node terakhir (tail) dari linked list

8. Penyisipan di Depan

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

```
    return new_node
```

Menyisipkan node di depan (awal list) → node baru akan menunjuk ke head sebelumnya.

9. Penyisipan di Posisi Tertentu

```
def sisip_dimana_aja(head, data, position):
    new_node = {'data': data, 'next': None}
    if position == 0:
        return sisip_depan(head, data)
```

Fungsi menyisipkan node di posisi mana pun. Jika posisi 0, gunakan sisip_depan.

10.
```
    current = head
    index = 0
    while current is not None and index < position - 1:
        current = current['next']
        index += 1
```

Traversal menuju node sebelum posisi target.

11.
```
if current is None:
        print("Posisi melebihi panjang linked list!")
        return head
```

Cek apakah posisi valid.

12.
```
new_node['next'] = current['next']
    current['next'] = new_node
    return head
```

Menyisipkan node di posisi yang ditentukan.

13. Menghapus Node Pertama (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']
```

Menghapus node pertama dan mengembalikan head yang baru.

14. Menghapus Node Terakhir (Tail)

```
def hapus_tail(head):
    if head is None:
        print('Linked-List Kosong, tidak ada yang bisa dihapus!')
        return None
```

Cek apakah list kosong.

15.
```
if head['next'] is None:
        print(f"Node dengan data '{head['data']}' dihapus. Linked list sekarang kosong.")
        return None
```

Jika hanya ada satu node, hapus node itu dan kembalikan None.

16.
```
current = head
    while current['next']['next'] is not None:
        current = current['next']
```

Traversal hingga mencapai node sebelum tail.

17.
```
print(f"\nNode dengan data '{current['next']['data']}' dihapus dari akhir.")
    current['next'] = None
    return head
```

Hapus tail dengan memutus sambungan next dari node sebelumnya.

18. Menghapus Node di Tengah (Posisi Tertentu)

```
def hapus_tengah(head, position):
    if head is None:
        print('\nLinked-List Kosong, tidak ada yang bisa dihapus!')
```

return None

Cek jika list kosong.

19. if position < 0:

print('\nPosisi Tidak Valid')

return head

Validasi posisi.

20. if position == 0:

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

hapus_head(head)

return head['next']

Kalau posisi 0, sama dengan hapus head.

21. current = head

index = 0

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

current = current['next']

index += 1

Mencari node sebelum posisi target.

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

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

return head

Validasi panjang list.

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

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

return head

Node dihapus dengan melewati node target (current['next'] = current['next']['next']).