PRAKTEK 22

```
# praktek 22
# function membuat node
def buat_node (data) :
     return {'data' : data,
             'next' : None}
```

Penjelasan:

- **Tujuan:** Membuat sebuah node baru dalam bentuk dictionary.
- **Isi node:**
  - `'data'`: menyimpan nilai/data.
  - `'next'`: mengarah ke node selanjutnya (default `None`, karena belum terhubung)

```
# function untuk menambah node di akhir list
def tambah_node (head, data):
      #buat node baru
      new_node = buat_node(data)
```

Penjelasan:

- **Langkah 1:** Buat node baru dengan data yang diberikan.
- `new_node`: node baru yang siap ditambahkan ke list

```
# cek apakah list kosong
      if head is None:
            return new_node
```

Penjelasan:

- **Langkah 2:** Jika `head` masih `None` (linked list masih kosong), maka node baru menjadi head-nya.

```
current = head
      # traversal ke semua elemen list (node)
      while current['next'] is not None :
            current = current ['next']
```

Penjelasan:

- **Langkah 3:** Jika list tidak kosong, cari node terakhir (`current['next'] is None` berarti akhir list).
- Gunakan loop untuk berpindah dari satu node ke node selanjutnya.

```
#jika sudah sampai di akhir list
      current ['next'] = new_node
      return head
```

Penjelasan:

- **Langkah 4:** Sambungkan node baru ke node terakhir dengan mengatur `next` dari node terakhir.
- **Return head** untuk memastikan struktur list tetap bisa diakses dari awal.

```
# function menampilokan linked list
def cetak_linked_list(head):
    # menyimpan isi head ke dalam current
    current = head
    # cetak penanda head
    print("head", end='->')
```

Penjelasan:

Mulai dari `head`, dan tampilkan "Head → ".

```
# pindah current ke node berikutnya
    while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
```

Penjelasan:

Selama `current` tidak kosong:

- Cetak `data` dari node.
- Pindah ke node berikutnya.

```
print('Null')
```

Penjelasan:

Setelah sampai node terakhir, tampilkan "NULL" sebagai akhir dari list.

```
head = None
```

Penjelasan:

- Awalnya linked list kosong (`head` = None).

```
head = tambah_node(head, 10)
head = tambah_node(head, 11)
head = tambah_node(head, 12)
```

- Tambahkan tiga node berturut-turut dengan nilai 10, 11, dan 12.

```
print('linked list')
cetak_linked_list(head)
```

Penjelasan:

- Cetak isi linked list yang telah dibuat.

PRAKTEK 23

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

Penjelasan:

- Membuat sebuah node baru dengan `data` yang diberikan.
- Node berupa dictionary dengan 2 kunci: `'data'` dan `'next'`.
- `'data'` menyimpan nilai, `'next'` menyimpan referensi ke node berikutnya (atau `None` jika ini node terakhir).

```
# menambahkan node di akhir list
def tambah_node(head, data):
    new_node = buat_node(data)
    if head is None:
        return new_node
```

Penjelasan:

- Menambahkan node baru ke **akhir linked list**.
- Jika `head` adalah `None`, berarti list masih kosong → node baru menjadi `head`.

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

Penjelasan:

Jika list tidak kosong, cari node terakhir (`current['next']` adalah `None`).

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

Penjelasan:

- Setelah ketemu node terakhir, tautkan node barunya ke situ.
- Return head agar list tetap bisa digunakan.

```
# traversal untuk cetak isi linked-list
def traversal_to_display(head):
    current = head
    print('Head', end=' → ')
```

Penjelasan:

Mulai dari node paling awal (`head`), cetak `"Head → "` sebagai awalan visual.

```
while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
```

Penjelasan:

Selama masih ada node (`current` tidak `None`), cetak data lalu maju ke node berikutnya.

```
print("NULL")
```

Penejlasan:

- Cetak `"NULL"` sebagai penanda akhir list.

```
# traversal untuk menghitung jumlah elemen dalam linked-list
def traversal_to_count_nodes(head):
    count = 0
    current = head
```

Penejlasan:

Inisialisasi penghitung count dan mulai dari head.

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

Penejlasan:

Tambah `count` untuk setiap node, lanjut ke node berikutnya.

```
 return count
```

Penejlasan:

Kembalikan jumlah node yang ditemukan.

```
# traversal untuk mencari dimana tail (node terakhir)
def traversal_to_get_tail(head):
    if head is None:
        return None
```

Penejlasan:

Jika list kosong, tidak ada tail → return None.

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

Penejlasan:

Maju terus hingga ketemu node yang next-nya None, berarti itu node terakhir.

```
return current
```

Penejlasan:

Kembalikan node terakhir (tail).

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

Penejlasan:

Membuat linked list kosong (`head = None`) dan tambahkan 4 node: 10 → 15 → 117 → 19.

```
print("Isi Linked-List")
traversal_to_display(head)
```

Penejlasan:

Cetak seluruh isi linked list.

```
print("Jumlah Nodes = ", traversal_to_count_nodes(head))
```

Penejlasan:

Hitung dan tampilkan jumlah node (dalam hal ini 4).

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

Penejlasan:

Tampilkan nilai dari node pertama (`head`), yaitu `10`.

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

Penjelasan:

Ambil node terakhir (tail) dan tampilkan datanya, yaitu `19`.

PRAKTEK 24

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

Penejelasan:

- Fungsi ini menyisipkan **node baru di awal linked list**.
- `new_node` berisi dictionary dengan:
  - `'data'`: nilai data yang ingin dimasukkan.
  - `'next'`: referensi ke node yang sebelumnya menjadi head (yaitu `head` lama).
- Node baru akan langsung menjadi `head` baru.
- `return new_node` akan mengubah `head` ke node baru tersebut.

```
def cetak_linked_list(head):
    current = head
    print('Head', end=' → ')
```

Penejlasan:

- Memulai pencetakan isi linked list dari node pertama (`head`).
- Tampilkan teks `'Head → '` sebagai awalan tampilan visual.

```
while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
```

Penjelasan:

Selama node masih ada (`current` tidak `None`):

- Cetak nilai `data` dari node saat ini.
- Pindah ke node berikutnya.

```
print("NULL")
```

Penjelasan:

Setelah traversal selesai, tampilkan `"NULL"` sebagai penanda akhir linked list.

```
head = None
```

Penjelasan:

Awalnya linked list kosong.

```
head = sisip_depan(head, 30)
head = sisip_depan(head, 20)
head = sisip_depan(head, 10)
```

Penjelasan:

Menyisipkan tiga node **di depan** secara bertahap:

- Node `10` disisipkan terakhir, sehingga menjadi head.
- Hasil akhir list: `10 → 20 → 30 → NULL`

```
print("Isi Linked-List Sebelum Penyisipan di Depan")
cetak = cetak_linked_list(head)
```

Penjelasan:

- Menampilkan isi linked list sebelum penambahan data `99`.
- Variabel `cetak` sebenarnya tidak dibutuhkan karena fungsi `cetak_linked_list` hanya mencetak, tidak mengembalikan nilai.

```
data = 99
head = sisip_depan(head, data)
```

Penjelasan:

- Menyisipkan node `99` di awal.
- Node ini akan menjadi head baru.
- Sekarang list menjadi: `99 → 10 → 20 → 30 → NULL`

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

Penjelasan:

Menampilkan data yang baru disisipkan.

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

Penjelasan:

Cetak seluruh isi linked list setelah penambahan node 99 di awal.

PRAKTEK 25

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

Penjelasan:

- Membuat node baru (`new_node`) berupa dictionary dengan kunci `'data'` dan `'next'`.
- `data` adalah nilai yang ingin disisipkan.
- `next` diisi dengan `head`, artinya node baru akan menunjuk ke node yang sebelumnya menjadi head.
- Node baru ini menjadi **head** baru setelah disisipkan.

```
ef sisip_dimana_aja(head, data, position):
    new_node = {'data': data, 'next': None}
```

Penjelasan:

- Membuat node baru dengan data tertentu dan `next` awalnya `None`.

```
if position == 0:
        return sisip_depan(head, data)
```

Penjelasan:

- Jika posisi penyisipan adalah 0 (di depan), maka gunakan fungsi `sisip_depan`.

```
current = head
    index = 0
```

Penjelasan:

- Persiapan traversal: `current` menunjuk ke head, `index` mulai dari 0.

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

Penjelasan:

- Menelusuri node sampai ke posisi sebelum tempat penyisipan (`position - 1`).

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

Penjelasan:

- Jika mencapai akhir list sebelum sampai ke posisi yang diinginkan, posisi tidak valid → tidak ada penyisipan.

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

Penjelasan:

- Hubungkan node baru ke node setelahnya (`current['next']`).
- Hubungkan node sebelumnya (`current`) ke node baru.

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

Penjelasan:

- Mencetak isi dari linked list mulai dari head sampai akhir.
- Setiap node dicetak dalam format `data →`, diakhiri dengan `"NULL"`.

```
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)
```

Penjelasan:

- Membuat linked list awal:
- `70 → 50 → 10 → 20 → 30 → NULL`

```
print("Isi Linked-List Sebelum Penyisipan")
cetak = cetak_linked_list(head)
```

Penjelasan:

- Menampilkan isi linked list sebelum disisipkan node baru.

```
data = 99
pos = 3
head = sisip_dimana_aja(head, data, pos)
```

Penjelasan:

- Menyisipkan data `99` di posisi ke-3.
- Traversal:
  - Posisi 0: 70
  - Posisi 1: 50
  - Posisi 2: 10 → sisipkan setelah ini.
  - Posisi 3: **99**
  - Sisanya bergeser: 20 → 30

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

Penjelasan:

- Menampilkan informasi penyisipan.

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

Penjelasan:

- Menampilkan linked list setelah penyisipan:
- `70 → 50 → 10 → 99 → 20 → 30 → NULL`

PRAKTEK 26

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

Penjelasan:

- Fungsi ini menyisipkan node baru **di depan** (head) linked list.
- Membuat node baru (bertipe dictionary) dengan:
- `'data'`: nilai data dari node.
- `'next'`: menunjuk ke node head sebelumnya.
- Mengembalikan node baru sebagai `head` baru dari linked list.
- Mengembalikan node baru sebagai `head` baru dari linked list.

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

Penjelasan:

- Fungsi untuk menyisipkan node baru di **posisi tertentu**.
- Membuat node baru dengan data tertentu.

```
if position == 0:
        return sisip_depan(head, data)
```

Penjelasan:

- Jika posisi penyisipan adalah 0, langsung gunakan fungsi `sisip_depan`.

```
current = head
    index = 0
```

Penjelasan:

- Persiapan traversal: mulai dari head dan posisi 0.

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

Penjelasan:

- Traversal menuju node sebelum posisi target.

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

Penjelasan:

- Jika traversal gagal menemukan posisi yang valid, batalkan penyisipan.

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

Penjelasan:

Lakukan penyisipan:

- Sambungkan `new_node` ke node berikutnya.
- Ubah `next` dari node sebelumnya agar menunjuk ke node baru.

```
def hapus_head(head):
```

Penjelasan:

- Fungsi untuk **menghapus node pertama (head)**.

```
 if head is None:
        print("Linked-List kosong, tidak ada yang bisa")
        return None
```

Penjelasan:

- Jika linked list kosong, beri pesan dan kembalikan `None`.

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

Penjelasan:

- Tampilkan informasi data node yang akan dihapus.
- Kembalikan node setelah head sebagai head yang baru.

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

Penjelasan:

- Fungsi untuk mencetak semua isi linked list dari head hingga akhir.
- Mulai dari head, cetak label awal "Head →".
- Selama node masih ada, cetak datanya dan pindah ke node berikutnya.
- Setelah mencapai akhir list, cetak "NULL" sebagai penutup.

```
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
```

Penjelasan:

- Linked list awal kosong (`head` bernilai `None`).
- Menyisipkan node ke depan satu per satu:

- Setelah semua disisipkan, urutan menjadi:
- `Head → 70 → 50 → 10 → 20 → 30 → NULL`

```
print("Isi Linked-List Sebelum Penghapusan")
cetak_linked_list(head)
```

Penjelasan:

- Menampilkan isi linked list sebelum penghapusan head.

```
head = hapus_head(head)
```

Penjelasan:

- Menghapus node pertama (`70`), dan mengatur `head` ke node berikutnya (`50`).

```
print("Isi Linked-List Setelah Penghapusan Head ")
cetak_linked_list(head)
```

Penjelasan:

- Menampilkan isi linked list setelah penghapusan:
- `Head → 50 → 10 → 20 → 30 → NULL`