TUGAS STRUKTUR DATA SMESTER II

Dosen pengampu:

ADAM BACHTIAR, S.kom.,M.MT

OLEH:

MUHAMMAD INDRA WAHYU ARDIKA (24241006)

PENDIDIKAN TEKNOLOGI INFORMASI

FAKULTAS TEHNIK

UNIVERSITAS PENDIDIKAN MANDALIKA

2025

PERAKTEK 1

```
# impor library numpy
import numpy as np

# membuat array dengan numpy
nilai_siswa = np.array([85, 55, 40, 90])

# akses data pada array
print(nilai_siswa[3])
```

HASIL

90

PENJELASAN

Baris 1# impor library numpy

Komentar ini menjelaskan bahwa baris berikutnya akan digunakan untuk mengimpor library numpy, yang merupakan library Python untuk operasi numerik dan array.

Baris 2 = mengimpor library numpy dan memberi alias np. Dengan cara ini, kamu bisa menggunakan np sebagai singkatan untuk memanggil fungsi-fungsi dari numpy.

Baris 3 =Komentar ini menjelaskan bahwa baris selanjutnya akan digunakan untuk membuat array menggunakan numpy.

Baris 4 = membuat array numpy berisi empat nilai: 85, 55, 40, dan 90, lalu menyimpannya dalam variabel nilai_siswa.

Baris 5 = Komentar ini memberi tahu bahwa baris berikut akan mengakses (mengambil) salah satu data dari array.

Baris 6 = mencetak nilai pada indeks ke-3 dari array nilai_siswa. Karena indeks dimulai dari 0, maka nilai_siswa[3] adalah 90.

PERAKTEK 2

```
# impor libaray numpy
import numpy as np

# membuat array dengan numpy
nilai_siswa_1 = np.array([75, 65, 45, 80])
nilai_siswa_2 = np.array([[85, 55, 40], [50, 40, 99]])

# cara akses elemen array
print(nilai_siswa_1[0])
print(nilai_siswa_2[1][1])

# mengubah nilai elemen array
nilai_siswa_1[0] = 88
nilai_siswa_2[1][1] = 70

# cek perubahannya dengan akses elemen array
print(nilai_siswa_1[0])
print(nilai_siswa_2[1][1])

# Cek ukuran dan dimensi array
print("Ukuran Array : ", nilai_siswa_1.shape)
print("Ukuran Array : ", nilai_siswa_2.shape)
print("Dimensi Array : ", nilai_siswa_2.ndim)
```

HAISL

```
75
40
88
70
Ukuran Array :  (4,)
Ukuran Array :  (2, 3)
Dimensi Array :  2
```

PENJELASAN

Baris 1 = Menampilkan jumlah dimensi dari nilai_siswa_2. Output: 2 karena array tersebut adalah 2 dimensi (baris dan kolom).

Mengimpor library NumPy dan memberi alias np. Ini membuat kita bisa menulis np.array daripada numpy.array. NumPy adalah library Python untuk operasi matematika dan manipulasi array.

Baris 2 = Membuat array 1 dimensi (vektor) dengan 4 nilai: 75, 65, 45, 80. Disimpan dalam variabel nilai_siswa_1

Baris 3 = Membuat array 2 dimensi (matriks) berisi 2 baris dan 3 kolom. Disimpan di variabel nilai_siswa_2.

Baris 4 = Mencetak elemen pertama dari nilai_siswa_1. Indeks 0 menunjukkan posisi pertama dalam array. Output: 75.

Baris 5 = Mengakses dan mencetak nilai pada baris ke-2 dan kolom ke-2 dari nilai_siswa_2. (ingat: indeks mulai dari 0). Output: 40.

Baris 6 = Mengubah nilai elemen pertama (indeks 0) pada nilai_siswa_1 dari 75 menjadi 88.

Baris 7 = Mengubah nilai baris ke-2, kolom ke-2 (indeks [1][1]) pada nilai_siswa_2 dari 40 menjadi 70.

Baris 8 = Mencetak ulang elemen pertama nilai_siswa_1 setelah diubah. Output: 88.

Baris 9 = Mencetak ulang nilai pada indeks [1][1] dari nilai_siswa_2 setelah diubah. Output: 70.

Baris 10 = Menampilkan ukuran atau bentuk array nilai_siswa_1. Output: (4,) yang artinya array 1 dimensi dengan 4 elemen.

Baris 11 = Menampilkan ukuran array nilai_siswa_2. Output: (2, 3) artinya 2 baris dan 3 kolom.

Baris 12 = Menampilkan jumlah dimensi dari nilai_siswa_2. Output: 2 karena array tersebut adalah 2 dimensi (baris dan kolom).

PRAKTEK 3

```python
# impor library numpy
import numpy as np

# membuat array
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

# menggunakan operasi penjumlahan pada 2 array
print(a + b)       # array([5, 7, 9])

# Indexing dan Slicing pada Array
arr = np.array([10, 20, 30, 40])
print(arr[1:3])    # array([20, 30])


# iterasi pada array
for x in arr:
    print(x)
```

HASIL

```
[5 7 9]
[20 30]
10
20
30
40
```

PENJELASAN

Baris 1 = Mengimpor library NumPy dan memberi alias np, agar lebih singkat saat digunakan.

Baris 2 = Membuat array 1 dimensi dan menyimpannya ke variabel a.

Baris 3 = Membuat array 1 dimensi dan menyimpannya ke variabel b.

Baris 4 = Menambahkan dua array secara elemen per elemen (penjumlahan vektor).

Baris 5 = Membuat array 1 dimensi dan menyimpannya ke variabel arr.

Baris 6 = Mengambil bagian dari array dari indeks 1 hingga sebelum indeks 3 (slicing).

Baris 7 = Memulai perulangan untuk setiap elemen dalam array arr.

Baris 8 = Mencetak setiap elemen dalam array satu per satu selama iterasi berlangsung.

PRAKTEK 4

```
# membuat array
arr = [1, 2, 3, 4, 5]

# Linear Traversal ke tiap elemen arr
print("Linear Traversal: ", end=" ")
for i in arr:
    print(i, end=" ")
print()
```

HASIL

```
Linear Traversal:  1 2 3 4 5
```

PENJELASAN

Baris 1 = Membuat list (array dalam konteks Python biasa) yang berisi lima elemen: 1, 2, 3, 4, 5. Disimpan dalam variabel arr.

Baris 2 = Mencetak teks "Linear Traversal: " ke layar **tanpa pindah baris**, karena end=" " memberi spasi di akhir, bukan baris baru.

Baris 3 = Memulai perulangan (looping) **linear traversal**, yaitu mengakses setiap elemen dalam arr satu per satu.

Baris 4 = Mencetak elemen saat ini (i) **di baris yang sama**, dengan spasi setelahnya. Ini membuat semua angka tercetak berurutan dalam satu baris.

Baris 5 = Mencetak baris kosong untuk **pindah ke baris baru** setelah loop selesai, agar hasil lebih rapi.

PRAKTEK 5

```
# membuat array
arr = [1, 2, 3, 4, 5]

# Reverse Traversal dari elemen akhir
print("Reverse Traversal: ", end="")
for i in range(len(arr) - 1, -1, -1):
    print(arr[i], end=" ")
print()
```

HASIL

```
Reverse Traversal: 5 4 3 2 1
```

PENJELASAN

Baris 1 = Membuat list (array) bernama arr yang berisi lima elemen: 1, 2, 3, 4, 5.

Baris 2 = Mencetak teks "Reverse Traversal: " tanpa pindah baris karena end="" mencegah baris baru setelah teks.

Baris 3 = Melakukan perulangan **secara terbalik** dari indeks terakhir ke indeks pertama.

- len(arr) - 1 → indeks terakhir (4)
- -1 → batas akhir (tidak termasuk, jadi sampai indeks 0)
- -1 → langkah mundur satu per satu

Baris 4 = Mencetak elemen pada indeks i dari arr, diikuti spasi, **tanpa pindah baris**.

Baris 5 = Pindah ke baris baru setelah selesai mencetak semua elemen secara terbalik.

PRAKTEK 7

```
# membuat array
arr = [1, 2, 3, 4, 5]

# mendeklarasikan nilai awal
n = len(arr)
i = 0

print("Linear Traversal using while loop: ", end=" ")
# Linear Traversal dengan while
while i < n:
    print(arr[i], end=" ")
    i += 1
print()
```

HASIL

```
Traversal using while loop:  1 2 3 4 5
```

PENJELASAN

Baris 1 = Membuat list arr yang berisi elemen-elemen: 1, 2, 3, 4, 5.

Baris 2 = Menghitung panjang (jumlah elemen) dari arr, hasilnya disimpan di variabel n (nilai n akan menjadi 5).

Baris 3 = Inisialisasi variabel i sebagai penghitung indeks, dimulai dari 0.

Baris 4 = Mencetak teks pembuka tanpa pindah ke baris baru karena end=" ".

Baris 5 = Memulai loop while yang akan berjalan selama i kurang dari n (selama masih dalam rentang indeks array).

Baris 6 = Mencetak elemen arr pada indeks i di baris yang sama, diikuti spasi.

Baris 7 = Menambahkan 1 ke nilai i agar loop bergerak ke indeks berikutnya.

Baris 8 = Mencetak baris baru setelah loop selesai untuk merapikan output.

PRAKTEK 8

```
# membuat array
arr = [1, 2, 3, 4, 5]

# mendeklarasikan nilai awal
start = 0
end = len(arr) - 1

print("Reverse Traversal using while loop: ", end=" ")
# Reverse Traversal dengan while
while start < end:

    arr[start], arr[end] = arr[end], arr[start]
    start += 1
    end -= 1
print(arr)
```

HAISL

```
Reverse Traversal using while loop:  [5, 4, 3, 2, 1]
```

PENJELASAN

Baris 1 = Membuat list arr yang berisi lima elemen: [1, 2, 3, 4, 5].

Baris 2 = Inisialisasi variabel start sebagai indeks awal (posisi kiri dari array).

Baris 3 = Inisialisasi variabel end sebagai indeks terakhir array (posisi kanan). len(arr) - 1 = 5 - 1 = 4.

Baris 4 = Mencetak teks penjelasan **tanpa pindah baris**, karena end=" ".

Baris 5 = Loop while akan berjalan selama indeks start masih lebih kecil dari end.
Tujuannya adalah menukar elemen dari kedua ujung array menuju tengah.

Baris 6 = Menukar elemen di posisi start dengan elemen di posisi end.
Contoh pertama: arr[0] ditukar dengan arr[4].

Baris 7 = Memajukan indeks start ke kanan satu langkah.

Baris 8 = Memundurkan indeks end ke kiri satu langkah.

Baris 9 = Memundurkan indeks end ke kiri satu langkah.

PRAKTEK 9

```
# membuat array
arr = [12, 16, 20, 40, 50, 70]

# cetak arr sebelum penyisipan
print("Array Sebelum Insertion : ", arr)

# cetak panjang array sebelum penyisipan
print("Panjang Array : ", len(arr))

# menyisipkan array di akhir elemen menggunakan .append()
arr.append(26)

# cetak arr setelah penyisipan
print("Array Setelah Insertion : ", arr)

# cetak panjang array setelah penyisipan
print("Panjang Array : ", len(arr))
```

HASIL

```
Array Sebelum Insertion :  [12, 16, 20, 40, 50, 70]
Panjang Array :  6
Array Setelah Insertion :  [12, 16, 20, 40, 50, 70, 26]
Panjang Array :  7
```

PENJELASAN

Baris 1 = Membuat list arr berisi 6 elemen angka.

Baris 2 = Mencetak isi array sebelum elemen baru ditambahkan.

Baris 3 = Mencetak jumlah elemen dalam array sebelum penambahan. Fungsi len() menghitung panjang list.

Baris 4 = Menambahkan elemen 26 ke **akhir array** menggunakan metode .append().

Baris 5 = Mencetak isi array setelah elemen 26 disisipkan di bagian akhir.

Baris 6 = Mencetak panjang array setelah penambahan. Harusnya bertambah satu elemen menjadi 7.

PRAKTEK 10

```
# membuat array
arr = [12, 16, 20, 40, 50, 70]

# cetak arr sebelum penyisipan
print("Array Sebelum Insertion : ", arr)

# cetak panjang array sebelum penyisipan
print("Panjang Array : ", len(arr))

# menyisipkan array pada tengah elemen menggunakan .insert(pos, x)
arr.insert(4, 5)

# cetak arr setelah penyisipan
print("Array Setelah Insertion : ", arr)

# cetak panjang array setelah penyisipan
print("Panjang Array : ", len(arr))
```

HASIL

```
Array Sebelum Insertion :  [12, 16, 20, 40, 50, 70]
Panjang Array :  6
Array Setelah Insertion :  [12, 16, 20, 40, 5, 50, 70]
Panjang Array :  7
```

PENJELASAN

Baris 1 = Membuat list arr dengan 6 elemen angka.

Baris 2 = Menampilkan isi array sebelum dilakukan penyisipan elemen baru.

Baris 3 = Menampilkan jumlah elemen dalam array sebelum penambahan. Hasilnya adalah 6.

Baris 4 = Menyisipkan nilai 5 ke dalam array di posisi indeks 4.
Elemen yang sebelumnya berada di posisi tersebut (50) dan setelahnya akan bergeser ke kanan.

Setelah penyisipan, isi array menjadi:

[12, 16, 20, 40, 5, 50, 70]

Baris 5 = Menampilkan isi array setelah elemen 5 disisipkan di indeks ke-4.

Baris 6 = Menampilkan panjang array setelah penambahan. Sekarang panjangnya menjadi 7.

PRAKTEK 11

```
# membuat array
a = [10, 20, 30, 40, 50]
print("Array Sebelum Deletion : ", a)

# menghapus elemen array pertama yang nilainya 30
a.remove(30)
print("Setelah remove(30):", a)

# menghapus elemen array pada index 1 (20)
popped_val = a.pop(1)
print("Popped element:", popped_val)
print("Setelah pop(1):", a)

# Menghapus elemen pertama (10)
del a[0]
print("Setelah del a[0]:", a)
```

HASIL

```
Array Sebelum Deletion :  [10, 20, 30, 40, 50]
Setelah remove(30): [10, 20, 40, 50]
Popped element: 20
Setelah pop(1): [10, 40, 50]
Setelah del a[0]: [40, 50]
```

PENJELASAN

Baris 1 = Membuat sebuah list bernama a yang berisi elemen-elemen: [10, 20, 30, 40, 50].

Baris 2 = Mencetak string "Array Sebelum Deletion : " diikuti dengan isi list a saat ini. Output: Array Sebelum Deletion : [10, 20, 30, 40, 50]

Baris 3 = Metode remove() digunakan untuk menghapus elemen pertama yang memiliki nilai 30 dari list a.

Setelah perintah ini, list a menjadi [10, 20, 40, 50].

Jika nilai 30 tidak ditemukan dalam list, akan muncul ValueError.

Baris 4 = Mencetak string "Setelah remove(30):" diikuti dengan isi list a setelah elemen 30 dihapus. Output: Setelah remove(30): [10, 20, 40, 50].

Baris 5 = Metode pop() digunakan untuk menghapus elemen pada indeks ke-1 (nilai 20) dari list a dan mengembalikannya.

Nilai yang dihapus disimpan dalam variabel popped_val.

Setelah perintah ini, list a menjadi [10, 40, 50].

Jika indeks yang diberikan tidak valid, akan muncul IndexError.

Baris 6 = Mencetak string "Popped element:" diikuti dengan nilai yang disimpan dalam variabel popped_val. Output: Popped element: 20.

Baris 7 = Mencetak string "Setelah pop(1):" diikuti dengan isi list a setelah elemen pada indeks ke-1 dihapus. Output: Setelah pop(1): [10, 40, 50].

Baris 8 = Pernyataan del digunakan untuk menghapus elemen pada indeks ke-0 (nilai 10) dari list a.

 Setelah perintah ini, list a menjadi [40, 50].

 Jika indeks yang diberikan tidak valid, akan muncul IndexError

Baris 9 = Mencetak string "Setelah del a[0]:" diikuti dengan isi list a setelah elemen pada indeks ke-0 dihapus. Output: Setelah del a[0]: [40, 50].

PRAKTEK 12

```
# impor library numpy
import numpy as np

# membuat matiks dengan numpy
matriks_np = np.array([[1,2,3],
                       [4,5,6],
                       [7,8,9]])
```

HASIL

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

PENJELASAN

Baris 1 = Pernyataan ini mengimpor library NumPy dan memberinya alias np. NumPy adalah library fundamental untuk komputasi ilmiah di Python, menyediakan objek array multidimensi dan berbagai fungsi untuk operasi matematika dan aljabar line

Baris 2-4 = Membuat objek array 2D (matriks) menggunakan np.array().

Array yang dihasilkan memiliki bentuk 3x3, dengan elemen-elemen sebagai berikut:

[[1, 2, 3],

[4, 5, 6],

[7, 8, 9]]

Matriks ini dapat digunakan untuk berbagai operasi matematika dan aljabar linear, seperti perkalian matriks, inversi, dan determinan.

PRAKTEK 13

```
X = [[12,7,3],
    [4,5,6],
    [7,8,9]]

Y = [[5,8,1],
    [6,7,3],
    [4,5,9]]

result = [[0,0,0],
          [0,0,0],
          [0,0,0]]

# proses penjumlahan dua matriks menggunakan nested loop
# mengulang sebanyak row (baris)
for i in range(len(X)):
   # mengulang sebanyak column (kolom)
   for j in range(len(X[0])):
       result[i][j] = X[i][j] + Y[i][j]

print("Hasil Penjumlahan Matriks dari LIST")

# cetak hasil penjumlahan secara iteratif
for r in result:
   print(r)
```

HASIL

```
Hasil Penjumlahan Matriks dari LIST
[17, 15, 4]
[10, 12, 9]
[11, 13, 18]
```

PENJELASAN

Baris 1-3 = Matriks X dan Y adalah dua matriks 3x3 yang akan dijumlahkan.

Matriks result diinisialisasi dengan nilai 0, berukuran sama dengan X dan Y, untuk menyimpan hasil penjumlahan.

Baris 4 = len(X) mengembalikan jumlah baris dalam matriks X. Misalnya, jika X adalah matriks 3x3, maka len(X) akan menghasilkan 3.

Fungsi range(len(X)) menghasilkan urutan angka dari 0 hingga len(X)-1, yang digunakan untuk mengakses setiap indeks baris dalam X.

Baris 5 = X[0] mengakses baris pertama dari matriks X. Misalnya, jika X adalah matriks 3x3, maka X[0] akan menghasilkan [12, 7, 3].

len(X[0]) mengembalikan jumlah elemen dalam baris pertama, yang merupakan jumlah kolom dalam matriks. Dalam contoh ini, len(X[0]) akan menghasilkan 3.

Fungsi range(len(X[0])) menghasilkan urutan angka dari 0 hingga len(X[0])-1, yang digunakan untuk mengakses setiap indeks kolom dalam baris ke-i dari X.

Baris 6 = X[i][j] mengakses elemen pada baris ke-i dan kolom ke-j dari matriks X.

Y[i][j] mengakses elemen pada baris ke-i dan kolom ke-j dari matriks Y.

Penjumlahan X[i][j] + Y[i][j] dilakukan untuk setiap pasangan elemen yang bersesuaian dari X dan Y.

Hasil penjumlahan disimpan dalam result[i][j], yaitu elemen pada baris ke-i dan kolom ke-j dari matriks result.

Baris 7 = Perintah ini mencetak string "Hasil Penjumlahan Matriks dari LIST" ke layar.

Tujuannya adalah untuk memberikan konteks atau judul sebelum menampilkan hasil penjumlahan matriks.

Baris 8 = for r in result: adalah loop yang mengiterasi setiap elemen dalam list result.Setiap elemen r adalah sebuah list yang mewakili satu baris dalam matriks hasil penjumlahan.

Baris 9 = print(r) mencetak setiap baris r ke layar.

PRAKTEK 14

```
# impor library numpy
import numpy as np

# Membuat matriks dengan numpy
X = np.array([
    [12,7,3],
    [4,5,6],
    [7,8,9]])

Y = np.array(
    [[5,8,1],
    [6,7,3],
    [4,5,9]])

# Operasi penjumlahan dua matrik numpy
result = X + Y

# cetak hasil
print("Hasil Penjumlahan Matriks dari NumPy")
print(result)
```

HASIL

```
Hasil Penjumlahan Matriks dari NumPy
[[17 15  4]
 [10 12  9]
 [11 13 18]]
```

PENJELASAN

Baris 1

- Mengimpor library NumPy dan memberinya alias np.
- NumPy adalah library Python yang menyediakan dukungan untuk array multidimensi dan berbagai fungsi matematika tingkat tinggi.

Baris 2-8

Membuat dua matriks 3x3 menggunakan np.array().

Penjelasan:

- X dan Y adalah objek array NumPy yang masing-masing berukuran 3x3.
- Setiap elemen dalam array adalah angka integer.

Baris 9

- Menjumlahkan dua matriks X dan Y secara elemen-wise.
- **Penjelasan**:
    - Operator + digunakan untuk menjumlahkan dua array NumPy. Ini adalah cara yang efisien dan umum digunakan dalam NumPy untuk operasi penjumlahan matriks.
    - Setiap elemen dalam matriks X dijumlahkan dengan elemen yang bersesuaian dalam matriks Y.

Baris 10-11

Mencetak hasil penjumlahan matriks ke layar.

Penjelasan:

- print("Hasil Penjumlahan Matriks dari NumPy") mencetak judul atau keterangan.
- print(result) mencetak matriks hasil penjumlahan.

PRAKTEK 15

```
# impor library numpy
import numpy as np

# Membuat matriks dengan numpy
X = np.array([
    [12,7,3],
    [4,5,6],
    [7,8,9]])

Y = np.array(
    [[5,8,1],
    [6,7,3],
    [4,5,9]])

# Operasi pengurangan dua matrik numpy
result = X - Y

# cetak hasil
print("Hasil Pengurangan Matriks dari NumPy")
print(result)
```

HAISL

```
Hasil Pengurangan Matriks dari NumPy
[[ 7 -1  2]
 [-2 -2  3]
 [ 3  3  0]]
```

PENJELASAN

Baris 1

- Mengimpor library NumPy dan memberinya alias np.
- **Penjelasan**: NumPy adalah library Python yang menyediakan dukungan untuk array multidimensi dan berbagai fungsi matematika tingkat tinggi.

Baris 2-9

Membuat dua matriks 3x3 menggunakan np.array().

Penjelasan:

- X dan Y adalah objek array NumPy yang masing-masing berukuran 3x3.
- Setiap elemen dalam array adalah angka integer.

Baris 10

Mengurangi dua matriks X dan Y secara elemen-wise.

Penjelasan:

- Operator - digunakan untuk melakukan pengurangan elemen demi elemen antara dua array NumPy. Ini adalah cara yang efisien dan umum digunakan dalam NumPy untuk operasi pengurangan matriks.
- Setiap elemen dalam matriks X dikurangi dengan elemen yang bersesuaian dalam matriks Y.

Baris 11-12

Mencetak hasil pengurangan matriks ke layar.

Penjelasan:

- print("Hasil Pengurangan Matriks dari NumPy") mencetak judul atau keterangan.
- print(result) mencetak matriks hasil pengurangan.

PRAKTEK 16

```
# impor library numpy
import numpy as np

# Membuat matriks dengan numpy
X = np.array([
    [12,7,3],
    [4,5,6],
    [7,8,9]])

Y = np.array(
    [[5,8,1],
    [6,7,3],
    [4,5,9]])

# Operasi perkalian dua matrik numpy
result = X * Y

# cetak hasil
print("Hasil Perkalian Matriks dari NumPy")
print(result)
```

HASIL

```
Hasil Perkalian Matriks dari NumPy
[[60 56  3]
 [24 35 18]
 [28 40 81]]
```

PENJELASAN

Baris 1

- Mengimpor library NumPy dan memberinya alias np.
- **Penjelasan**: NumPy adalah library Python yang menyediakan dukungan untuk array multidimensi dan berbagai fungsi matematika tingkat tinggi.

Baris 2-9

Membuat dua matriks 3x3 menggunakan np.array().

Penjelasan:

- X dan Y adalah objek array NumPy yang masing-masing berukuran 3x3.
- Setiap elemen dalam array adalah angka integer.

Baris 10

Melakukan perkalian elemen demi elemen antara matriks X dan Y.

Penjelasan:

- Operator * digunakan untuk melakukan perkalian elemen demi elemen antara dua array NumPy. Ini dikenal sebagai Hadamard product.
- Setiap elemen dalam matriks X dikalikan dengan elemen yang bersesuaian dalam matriks Y.
- Sebagai contoh, elemen pertama dari X (12) dikalikan dengan elemen pertama dari Y (5), menghasilkan 60. Proses ini diulang untuk setiap elemen yang bersesuaian.

Baris 11-12

Mencetak hasil perkalian matriks ke layar.

Penjelasan:

- print("Hasil Perkalian Matriks dari NumPy") mencetak judul atau keterangan.
- print(result) mencetak matriks hasil perkalian.

PRAKTEK 17

```
# Praktek 17 : Operasi Pembagian Matriks dengan numpy
# impor library numpy
import numpy as np

# Membuat matriks dengan numpy
X = np.array([
    [12,7,3],
    [4,5,6],
    [7,8,9]])

Y = np.array(
    [[5,8,1],
    [6,7,3],
    [4,5,9]])

# Operasi pembagian dua matrik numpy
result = X / Y

# cetak hasil
print("Hasil Pembagian Matriks dari NumPy")
print(result)
```

HASIL

```
Hasil Pembagian Matriks dari NumPy
[[2.4        0.875      3.        ]
 [0.66666667 0.71428571 2.        ]
 [1.75       1.6        1.        ]]
```

PENJELASAN

Baris 1

Mengimpor library NumPy dan memberinya alias np.

Penjelasan: NumPy adalah library Python yang menyediakan dukungan untuk array multidimensi dan berbagai fungsi matematika tingkat tinggi.

Baris 2-9

Membuat dua matriks 3x3 menggunakan np.array().

Penjelasan:

- X dan Y adalah objek array NumPy yang masing-masing berukuran 3x3.
- Setiap elemen dalam array adalah angka

Baris 10

- Melakukan pembagian elemen demi elemen antara matriks X dan Y.
- **Penjelasan**:
    - Operator / digunakan untuk melakukan pembagian elemen demi elemen antara dua array NumPy. Ini dikenal sebagai Hadamard division.
    - Setiap elemen dalam matriks X dibagi dengan elemen yang bersesuaian dalam matriks Y.
    - Sebagai contoh, elemen pertama dari X (12) dibagi dengan elemen pertama dari Y (5), menghasilkan 2.4. Proses ini diulang untuk setiap elemen yang bersesuaian.

Baris 11

Mencetak hasil pembagian matriks ke layar.

Penjelasan:

- print("Hasil Pembagian Matriks dari NumPy") mencetak judul atau keterangan.
- print(result) mencetak matriks hasil pembagian.

PRAKTEK 18

```python
# impor library numpy
import numpy as np

# membuat matriks
matriks_a = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

# cetak matriks
print("Matriks Sebelum Transpose")
print(matriks_a)

# transpose matriks_a
balik = matriks_a.transpose()

# cetak matriks setelah dibalik
print("Matriks Setelah Transpose")
print(balik)
```

HASIL

```
Matriks Sebelum Transpose
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Matriks Setelah Transpose
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

PENJELASAN

Baris 1

- Mengimpor library NumPy dan memberinya alias np.
- **Penjelasan**: NumPy adalah library Python yang menyediakan dukungan untuk array multidimensi dan berbagai fungsi matematika tingkat tinggi.

Baris 2-5

Membuat matriks 3x3 menggunakan np.array().

Penjelasan:

- matriks_a adalah objek array NumPy yang berukuran 3x3.
- Setiap elemen dalam array adalah angka integer.

Baris 6-7

Mencetak matriks sebelum dilakukan operasi transpose.

Penjelasan:

- print("Matriks Sebelum Transpose") mencetak judul atau keterangan.
- print(matriks_a) mencetak matriks matriks_a ke layar.

Baris 8

Melakukan operasi transpose pada matriks matriks_a.

Penjelasan:

- transpose() adalah metode NumPy yang digunakan untuk membalikkan baris dan kolom dari sebuah matriks. Untuk matriks dua dimensi, operasi ini menukar posisi baris dan kolom.
- Hasil dari operasi ini adalah matriks baru yang disimpan dalam variabel balik.

Baris 9-10

Mencetak matriks setelah dilakukan operasi transpose.

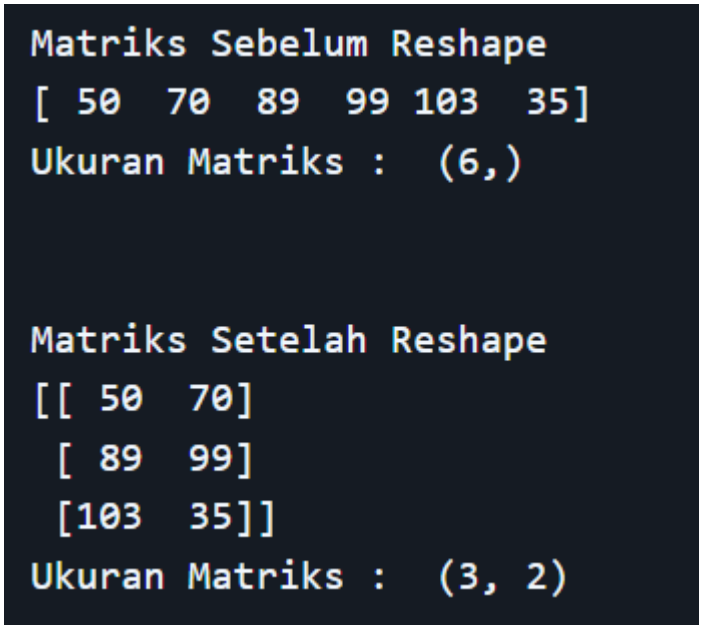
Penjelasan:

- print("Matriks Setelah Transpose") mencetak judul atau keterangan.
- print(balik) mencetak matriks balik ke layar.

PRAKTEK 19

```python
# impor library numpy
import numpy as np

# membuat array 1 dimensi
arr_1d = np.array([50, 70, 89, 99, 103, 35])

# cetak matriks sebelum reshape
print("Matriks Sebelum Reshape")
print(arr_1d)
print("Ukuran Matriks : ", arr_1d.shape)
print("\n")

# mengubah matriks menjadi ordo 3 x 2
ubah = arr_1d.reshape(3, 2)

# cetak matriks setelah reshape ke ordo 3 x 2
print("Matriks Setelah Reshape")
print(ubah)
print("Ukuran Matriks : ", ubah.shape)
```

HASIL



PENJELASAN

Baris 1

Mengimpor library NumPy dan memberinya alias np.

Penjelasan: NumPy adalah library Python yang menyediakan dukungan untuk array multidimensi dan berbagai fungsi matematika tingkat tinggi

Baris 2

- Membuat array 1D menggunakan np.array().
- **Penjelasan**:
  - arr_1d adalah objek array NumPy yang berukuran 1x6.
  - Setiap elemen dalam array adalah angka integer.

Baris 3-6

Mencetak array sebelum dilakukan operasi *reshape*.

Penjelasan:

- print("Matriks Sebelum Reshape") mencetak judul atau keterangan.
- print(arr_1d) mencetak array arr_1d ke layar.

- print("Ukuran Matriks : ", arr_1d.shape) mencetak ukuran (shape) dari array arr_1d. Metode .shape mengembalikan tuple yang menunjukkan dimensi array

baris 7

Melakukan operasi *reshape* pada array arr_1d menjadi matriks 2D dengan 3 baris dan 2 kolom.

Penjelasan:

- reshape(3, 2) mengubah array 1D menjadi matriks 2D dengan 3 baris dan 2 kolom.
- Setiap elemen dalam array arr_1d akan diatur ulang dalam matriks ubah sesuai dengan urutan elemen aslinya.

Baris 8-10

Mencetak matriks setelah dilakukan operasi *reshape*.

Penjelasan:

- print("Matriks Setelah Reshape") mencetak judul atau keterangan.
- print(ubah) mencetak matriks ubah ke layar.
- print("Ukuran Matriks : ", ubah.shape) mencetak ukuran (shape) dari matriks ubah. Metode .shape mengembalikan tuple yang menunjukkan dimensi matriks.

PRAKTEK 20

```
# vektor baris
vek_1 = np.array([1, 2, 3])

# vektor kolom
vek_2 = np.array([1],
                 [2],
                 [3])
# atau menggunakan transpose()
vek_3 = np.array([1, 2, 3]).T

print("Vektor Baris")
print(vek_1)
print("vektor Kolom")
print(vek_2)
print("Vektor Kolom dengan transpose()")
print(vek_3)
```

HASIL

```
Vektor Baris
[1 2 3]
vektor Kolom
[[1]
 [2]
 [3]]
Vektor Kolom dengan transpose()
[[1]
 [2]
 [3]]
```

PENJELASAN

Baris 1

Membuat vektor baris dengan elemen 1, 2, dan 3.

Penjelasan:

- np.array([1, 2, 3]) membuat array NumPy satu dimensi (1D) dengan elemen-elemen yang diberikan.
- Vektor ini memiliki bentuk (shape) (3,), yang berarti 3 elemen dalam satu dimensi

Baris 2-4

Membuat vektor kolom dengan elemen 1, 2, dan 3.

Penjelasan:

- Sintaks ini akan menghasilkan kesalahan (ValueError) karena np.array() mengharapkan satu argumen iterable (seperti list atau tuple) yang berisi elemen-elemen array.
- Untuk membuat vektor kolom yang benar, Anda perlu menyusun elemen-elemen dalam satu iterable, seperti [[1], [2], [3]].

Baris 5

Membuat vektor kolom dengan elemen 1, 2, dan 3 menggunakan operasi transpose.

Penjelasan:

- np.array([1, 2, 3]) membuat array 1D dengan elemen-elemen yang diberikan.
- .T adalah atribut yang digunakan untuk mentranspose array. Namun, karena array ini satu dimensi, mentranspose-nya tidak mengubah bentuknya.
- Untuk memastikan array menjadi vektor kolom, Anda dapat menggunakan np.reshape(3, 1) atau np.newaxis.

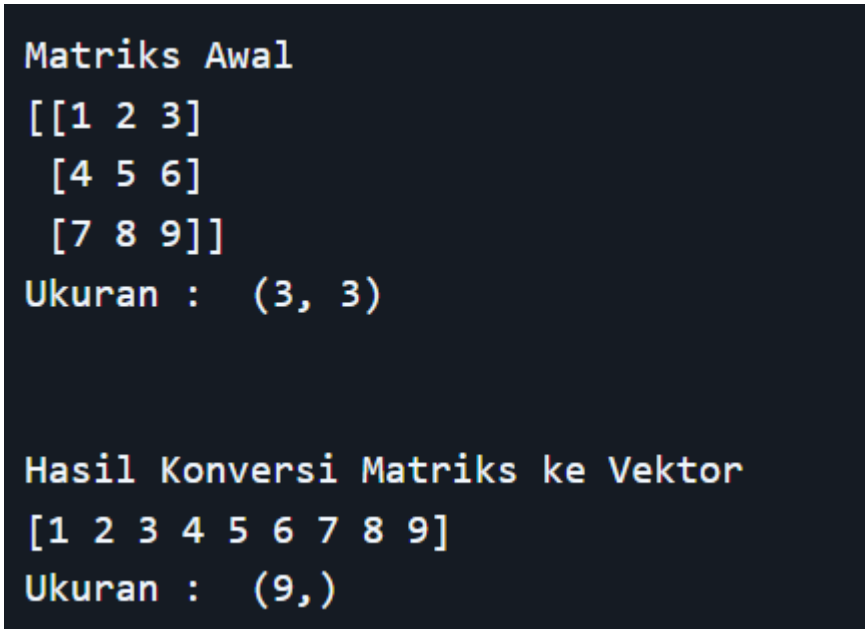
Baris 6-8

- Mencetak vektor-vektor yang telah dibuat ke layar.
- **Penjelasan**:
    - print("Vektor Baris") mencetak judul untuk vektor baris.
    - print(vek_1) mencetak vektor baris vek_1.
    - print("Vektor Kolom") mencetak judul untuk vektor kolom.
    - print(vek_2) mencetak vektor kolom vek_2.
    - print("Vektor Kolom dengan Transpose") mencetak judul untuk vektor kolom yang dibuat dengan transpose.
    - print(vek_3) mencetak vektor kolom vek_3

PRAKTEK 21

```python
# impor library numpy
import numpy as np

# membuat matriks
matriks_a = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

# cetak matriks awal
print("Matriks Awal")
print(matriks_a)
print("Ukuran : ", matriks_a.shape)
print("\n")

# ubah matriks menjadi vektor
jd_vektor = matriks_a.flatten()

# cetak vektor
print("Hasil Konversi Matriks ke Vektor")
print(jd_vektor)
print("Ukuran : ", jd_vektor.shape)
```

HASIL



PENJELASAN

Baris 1 = Baris ini mengimpor pustaka NumPy dengan alias np, yang memungkinkan Anda menggunakan fungsionalitas NumPy dalam kode Anda

Baris 2-5 = Baris ini membuat objek matriks_a yang merupakan array 2D (matriks) berukuran 3x3 menggunakan fungsi np.array(). Matriks ini berisi angka dari 1 hingga 9.

Baris 6-9 = Bagian ini mencetak teks "Matriks Awal", diikuti dengan isi matriks matriks_a dan ukuran matriks menggunakan properti .shape.

Baris 10 = Baris ini menggunakan metode .flatten() untuk mengonversi matriks 2D matriks_a menjadi array 1D (vektor) jd_vektor. Secara default, .flatten() menyusun elemen-elemen matriks dalam urutan baris (row-major order) .

Baris 11-13 = Bagian ini mencetak teks "Hasil Konversi Matriks ke Vektor", diikuti dengan isi dari jd_vektor dan ukuran vektor tersebut menggunakan properti .shape.

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
    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)
```

HASIL

```
Linked-List :
head → 10 → 11 → 12 → NULL
```

PENJELASAN

Baris 1-2 = Fungsi buat_node menerima parameter data dan mengembalikan sebuah dictionary yang merepresentasikan sebuah node. Node tersebut memiliki dua kunci: data untuk menyimpan nilai data dan next yang diinisialisasi dengan None, menunjukkan bahwa node ini belum terhubung ke node lain.

Baris 3-12 = Fungsi tambah_node menambahkan node baru dengan nilai data ke akhir linked list yang dimulai dari head:

- Jika head adalah None (linked list kosong), maka node baru menjadi head.
- Jika tidak, fungsi akan menelusuri linked list hingga menemukan node terakhir (yang next-nya adalah None), kemudian menghubungkannya dengan node baru.

Baris 13-19 = Fungsi cetak_linked_list menampilkan isi linked list mulai dari head:

- Dimulai dengan mencetak 'Head → '.
- Kemudian, fungsi menelusuri setiap node, mencetak nilai data dari node tersebut, dan bergerak ke node berikutnya melalui next.
- Proses ini berlanjut hingga mencapai node terakhir (yang next-nya adalah None), kemudian mencetak 'NULL' sebagai penanda akhir linked list.

Baris 20-25 = Bagian ini mendemonstrasikan penggunaan fungsi-fungsi di atas:

- head diinisialisasi sebagai None, menandakan linked list kosong.
- Tiga node dengan nilai 10, 11, dan 12 ditambahkan ke linked list menggunakan tambah_node.
- Terakhir, cetak_linked_list digunakan untuk menampilkan isi linked list yang telah terbentuk.

PERAKTEK 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")
```

```python
# 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)
```

```python
# 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'])
```

HASIL

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

PENJELASAN

Baris 1-2 = Fungsi buat_node menerima parameter data dan mengembalikan sebuah dictionary yang merepresentasikan sebuah node. Node tersebut memiliki dua kunci:

- data: untuk menyimpan nilai data.
- next: diinisialisasi dengan None, menunjukkan bahwa node ini belum terhubung ke node lain.

Baris 3-11 = Fungsi tambah_node menambahkan node baru dengan nilai data ke akhir linked list yang dimulai dari head:

- Jika head adalah None (linked list kosong), maka node baru menjadi head.
- Jika tidak, fungsi akan menelusuri linked list hingga menemukan node terakhir (yang next-nya adalah None), kemudian menghubungkannya dengan node baru.

Baris 12-18 = Fungsi traversal_to_display menampilkan isi linked list mulai dari head:

- Dimulai dengan mencetak 'Head → '.
- Kemudian, fungsi menelusuri setiap node, mencetak nilai data dari node tersebut, dan bergerak ke node berikutnya melalui next.
- Proses ini berlanjut hingga mencapai node terakhir (yang next-nya adalah None), kemudian mencetak 'NULL' sebagai penanda akhir linked list.

Baris 19-25 = Fungsi traversal_to_count_nodes menghitung jumlah node dalam linked list:

- Dimulai dengan count diinisialisasi dengan 0.
- Fungsi menelusuri setiap node, menambah count setiap kali menemukan node baru, dan bergerak ke node berikutnya melalui next.
- Proses ini berlanjut hingga mencapai node terakhir (yang next-nya adalah None), kemudian mengembalikan nilai count.

Baris 26-32 = Fungsi traversal_to_get_tail menemukan node terakhir (tail) dalam linked list:

- Jika head adalah None (linked list kosong), maka fungsi mengembalikan None.
- Fungsi menelusuri linked list hingga menemukan node terakhir (yang next-nya adalah None), kemudian mengembalikan node tersebut.

Baris 33-42 = Bagian ini mendemonstrasikan penggunaan fungsi-fungsi di atas:

- head diinisialisasi sebagai None, menandakan linked list kosong.
- Empat node dengan nilai 10, 15, 117, dan 19 ditambahkan ke linked list menggunakan tambah_node.
- Fungsi traversal_to_display digunakan untuk menampilkan isi linked list yang telah terbentuk.
- Fungsi traversal_to_count_nodes digunakan untuk menghitung jumlah node dalam linked list.
- Fungsi traversal_to_get_tail digunakan untuk menemukan node terakhir (tail) dalam linked list.

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)
```

HASIL

```
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 1-3 = Fungsi sisip_depan menerima dua parameter: head (node pertama dari linked list) dan data (nilai yang akan disisipkan). Fungsi ini membuat sebuah dictionary baru yang merepresentasikan node dengan dua kunci:

- data: menyimpan nilai yang diberikan.
- next: menunjuk ke node sebelumnya yang menjadi head.

Node baru ini kemudian dikembalikan sebagai head yang baru dari linked list.

Baris 4-10 = Fungsi cetak_linked_list digunakan untuk menampilkan isi linked list mulai dari head:

- Dimulai dengan mencetak 'Head → '.
- Kemudian, fungsi menelusuri setiap node, mencetak nilai data dari node tersebut, dan bergerak ke node berikutnya melalui next.
- Proses ini berlanjut hingga mencapai node terakhir (yang next-nya adalah None), kemudian mencetak 'NULL' sebagai penanda akhir linked list.

Baris 11-21 = Bagian ini mendemonstrasikan penggunaan fungsi sisip_depan untuk menyisipkan node baru di awal linked list:

- head diinisialisasi sebagai None, menandakan linked list kosong.
- Tiga node dengan nilai 30, 20, dan 10 ditambahkan ke linked list menggunakan sisip_depan.
- Fungsi cetak_linked_list digunakan untuk menampilkan isi linked list sebelum dan setelah penyisipan node baru dengan nilai 99.

PRAKTEK 25

```python
# 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)
```

HASIL

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

Data Yang Disispkan :  99
Pada posisi :  3

Isi Linked-List Setelah Penyisipan di tengah
Head → 70 → 50 → 10 → 99 → 20 → 30 → NULL
```

PENJELASAN

Baris 1-3 = **Fungsi sisip_depan**: Menambahkan node baru di awal linked list.

Parameter:

- head: Node pertama dari linked list yang ada.
- data: Nilai yang akan disimpan dalam node baru.

Proses:

- Membuat new_node, yaitu dictionary dengan dua kunci:
    - 'data': Menyimpan nilai data.
    - 'next': Menyimpan referensi ke node pertama (head) dari linked list yang ada.
- Mengembalikan new_node sebagai node pertama yang baru.

Baris 4-5 = **Fungsi sisip_dimana_aja**: Menambahkan node baru di posisi tertentu dalam linked list.

Parameter:

- head: Node pertama dari linked list yang ada.
- data: Nilai yang akan disimpan dalam node baru.
- position: Indeks posisi (dimulai dari 0) di mana node baru akan disisipkan.

Proses:

- Membuat new_node, yaitu dictionary dengan dua kunci:
    - 'data': Menyimpan nilai data.
    - 'next': Menyimpan referensi ke node berikutnya (diinisialisasi dengan None).

Baris 6-7 = **Pengecekan Posisi 0**: Jika posisi yang diminta adalah 0 (awal), maka fungsi akan memanggil sisip_depan untuk menambahkan node baru di awal linked list.

Baris 8-9 = **Inisialisasi Traversal**: Menyiapkan variabel current untuk menelusuri linked list dan index untuk menghitung posisi saat traversal.

Baris 10-12 = **Traversal**: Melakukan perulangan untuk menelusuri linked list hingga mencapai posisi sebelum posisi yang diinginkan (position - 1).

- **Kondisi Berhenti**:
    - current is not None: Masih ada node berikutnya.
    - index < position - 1: Belum mencapai posisi yang diinginkan.

Baris 13-15 = **Pengecekan Posisi Valid**: Jika setelah traversal, current adalah None, berarti posisi yang diminta melebihi panjang linked list. Fungsi akan mencetak pesan kesalahan dan mengembalikan head tanpa perubahan.

Baris 16-19 = **Penyisipan Node Baru**:

- Mengatur 'next' dari new_node untuk menunjuk ke node setelah current.
- Mengubah 'next' dari current untuk menunjuk ke new_node, sehingga new_node disisipkan setelah current.

Mengembalikan head: Mengembalikan node pertama dari linked list yang telah diperbarui.

Baris 20-26 = **Fungsi cetak_linked_list**: Menampilkan isi dari linked list mulai dari head.

Proses:

- Menelusuri setiap node dalam linked list.
- Mencetak nilai 'data' dari setiap node diikuti dengan tanda panah (→).
- Setelah mencapai akhir linked list (current adalah None), mencetak "NULL" sebagai penanda akhir.

Baris 27-32 = **Membuat Linked List**: Menambahkan beberapa node di awal linked list menggunakan fungsi sisip_depan.

Hasil: Linked list menjadi: 70 → 50 → 10 → 20 → 30 → NULL

Baris 33-34 = **Menampilkan Linked List**: Mencetak isi linked list sebelum penyisipan node baru.

Baris 35-37 = **Menyisipkan Node Baru**: Menambahkan node baru dengan nilai 99 di posisi ke-3 dalam linked list menggunakan fungsi sisip_dimana_aja.

- **Hasil**: Linked list menjadi: 70 → 50 → 10 → 99 → 20 → 30 → NULL.

Baris 38-39 = **Menampilkan Informasi Penyisipan**: Mencetak informasi mengenai data yang disisipkan dan posisi 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)
```

HASIL

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

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

PENJELASAN

Baris 1-3 = **Tujuan**: Menambahkan node baru di awal linked list.

Proses:

- Membuat dictionary new_node dengan dua kunci:
    - 'data': Menyimpan nilai data yang diberikan.
    - 'next': Menunjuk ke node pertama yang lama (head).
- Mengembalikan new_node sebagai node pertama yang baru.

Baris 4-18 = **Tujuan**: Menyisipkan node baru pada posisi tertentu dalam linked list.

Proses:

- Membuat dictionary new_node dengan dua kunci:
    - 'data': Menyimpan nilai data yang diberikan.
    - 'next': Diinisialisasi sebagai None.
- Jika position adalah 0, panggil fungsi sisip_depan untuk menambahkan node di awal.
- Melakukan traversal untuk mencapai posisi sebelum posisi yang diinginkan.
- Jika posisi valid, sisipkan new_node setelah node yang ditemukan.

Baris 19-24 = **Tujuan**: Menghapus node pertama (head) dari linked list.

Proses:

- Jika head adalah None, berarti linked list kosong; tampilkan pesan kesalahan dan kembalikan None.
- Jika tidak kosong, tampilkan informasi node yang dihapus.
- Kembalikan node kedua (head['next']) sebagai head yang baru.

Baris 25-31 = **Tujuan**: Menampilkan isi linked list dari head hingga akhir.

Proses:

- Melakukan traversal dari head hingga akhir linked list.
- Mencetak nilai 'data' dari setiap node diikuti dengan tanda panah (→).
- Setelah mencapai akhir, mencetak "NULL" sebagai penanda akhir.

Baris 32-42 = **Tujuan**: Membuat linked list, menampilkan isinya, menghapus node pertama, dan menampilkan isi linked list setelah penghapusan.

Proses:

- Membuat linked list dengan menambahkan beberapa node di awal menggunakan fungsi sisip_depan.
- Menampilkan isi linked list sebelum penghapusan.
- Menghapus node pertama menggunakan fungsi hapus_head.
- Menampilkan isi linked list setelah penghapusan.

PRAKTEK 27

```python
# 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)
```

HASIL

```
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 1: Mendefinisikan fungsi sisip_depan yang menerima dua parameter: head (node pertama dari linked list) dan data (nilai yang akan disisipkan).

Baris 2: Membuat node baru berupa dictionary dengan dua kunci:

- 'data': Menyimpan nilai yang diberikan (data).
- 'next': Menyimpan referensi ke node sebelumnya (head), sehingga node baru ini akan menjadi node pertama dalam linked list.

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

Baris 4: Mendefinisikan fungsi hapus_tail yang menerima satu parameter: head (node pertama dari linked list).

Baris 5-7: Mengecek apakah linked list kosong (head adalah None). Jika ya, mencetak pesan bahwa tidak ada yang bisa dihapus dan mengembalikan None.

Baris 8-10: Mengecek apakah hanya ada satu node dalam linked list (head adalah satu-satunya node). Jika ya, mencetak pesan bahwa node tersebut dihapus dan mengembalikan None, menjadikan linked list kosong.

Baris 11-13: Membuat variabel current yang dimulai dari head. Kemudian, melakukan iterasi untuk menemukan node kedua terakhir (sebelum tail). Iterasi berhenti ketika current['next']['next'] adalah None, yang berarti current adalah node kedua terakhir.

Baris 14-16: Mencetak pesan bahwa node dengan data tertentu (tail) dihapus dari akhir. Kemudian, mengubah referensi next dari node kedua terakhir menjadi None, menghapus node terakhir dari linked list. Fungsi mengembalikan head yang tidak berubah.

Baris 17: Mendefinisikan fungsi cetak_linked_list yang menerima satu parameter: head (node pertama dari linked list).

Baris 18: Membuat variabel current yang dimulai dari head.

Baris 19: Mencetak kata 'Head' sebagai penanda awal linked list.

Baris 20-26: Melakukan iterasi melalui linked list dan mencetak nilai dari setiap node (current['data']), diikuti dengan tanda panah ('→'). Setelah mencetak data, current diperbarui ke node berikutnya (current['next']).

Baris 27: Setelah iterasi selesai, mencetak "NULL" sebagai penanda akhir linked list.

Baris 28: Inisialisasi head sebagai None, menandakan linked list kosong.

Baris 29-33: Menyisipkan node baru di depan linked list dengan nilai 30, 20, 10, 50, dan 70 secara berurutan. Setiap penyisipan menjadikan node baru sebagai head yang baru.

Baris 34: Mencetak pesan "Isi Linked-List Sebelum Penghapusan".

Baris 35: Memanggil fungsi cetak_linked_list untuk menampilkan isi linked list sebelum penghapusan tail.

Baris 36: Memanggil fungsi hapus_tail untuk menghapus node terakhir (tail) dari linked list.

Baris 37: Mencetak pesan "Isi Linked-List Setelah Penghapusan Tail".

Baris 38: Memanggil fungsi cetak_linked_list untuk menampilkan isi linked list setelah penghapusan tail.

PRAKTEK 28

```
# 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)
```

HASIL

```
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

Baris 1-22: Fungsi ini memiliki tujuan untuk menghapus node pada posisi tertentu (misalnya node ke-1, ke-2, dst.) dalam *linked-list*. Posisi dimulai dari 0 (seperti indeks pada list Python).

Oke, mari kita mulai dari bagaian awal, fungsi hapus_tengah() menerima dua parameter yaitu head dan position. Dimana head adalah node pertama dalam *linked-list* sementara position sebagai posisi index dari node yang ingin dihapus.

Baris 23:Selanjutnya dalam fungsi ini ada beberapa pengecekan validitas yang pertama adalah mengecek apakah *linked-list* kosong

Baris 24-27:jika head == None, maka list kosong, dan tidak ada node yang bisa dihapus. Tetapi jika tidak lanjutkan ke pengecekan validitas berikutnya

Baris 28-31:Pada kode diatas pengecekan validitas dilakukan ketika user memasukkan parameter posisi kurang nail 0 atau nilai negatif if position < 0:, maka program akan mengeluarkan pesan bahwa posisi yang dimasukkan tidak valid print('\nPosisi Tidak Valid'), dan tidak terjadi penghapusan sehingga kembali ke *linked-list* semula

Baris 32-36: Pada potongan kode diatas kita akan melakukan pengecekan validitas ketika user memamsukkan posisi node adalah 0 if position == 0, maka program akan memberi pesan bahwa node dengan data tertentu dihapus dari posisi 0 print(f"Node dengan data '{head['data']}' dihapus dari posisi 0.") dan menjalankan fungsi hapus_head(head) yang terpisah dari fungsi ini. Kemudian mengembalikan node seteleh head menjadi head baru.

Baris 37-42: Pada bagian awal kita inisialisasi
pointer current pada node pertama (head), dan current akan digunakan untuk menelusuri *linked-list*, selanjutnya index = 0 merupakan penanda awal posisi saat traversal, dan digunakan untuk membandingkan posisition.

Berikutnya adalah melakukan perulangan untuk menemukan node sebelum posisi target. Kemudian position - 1 dimaksudkan untuk menghentikan perulangan ketika menemukan satu node sebelum node yang ingin dihapus. contohnya jika kita ingin menghapus node ke dua maka perulangan akan dihentikan pada ketika mencapai node ke satu.

Selama kondisi perulangan diatas masih terpenuhi maka jalankan
perintah current = current['next'], artinya
pindahkan current ke node berikutnya, dan perbarui nilai index += 1

Baris 43-46:potongan kode diatas merupakan sebuah pengecekan validitas yang berfungsi untuk memastikan posisi node yang diinputkan tidak melebih panjang dari *linked-list* if current is None or current['next'] is None. Jika kondisi ini terpenuhi maka cetak pesan print("\nPosisi melebih panjang dari linked-list") dan kembalikan nilai *linked-list* yang awal.

Baris 47-50:potongan kode diatas print(f"\nNode dengan data '{current['next']['data']}' dihapus dari posisi {position}.") diawali pemberitahuan kepada user bahwa node yang berada pada posisi yang dimaksud akan dihapus. Kode current['next']['data'] → itulah node target yang ingin kita hapus. dan mengembalikan nilai head *linked-list* setelah node yang terpilih dihapus.

Baris 51-59:Untuk menyisipkan node baru di awal/depan *double linked-list*, kita buat node baru dengan penunjuk/pointer sebelumnya (prev) sebagai NULL dan penunjuk/pointer berikutnya (next) ke head *double linked-list* saat ini. Kemudian, kita periksa apakah *linked-list* tidak kosong, lalu kita perbarui penunjuk/pointer sebelumnya (prev) dari head saat ini ke node baru. Terakhir, kita kembalikan node baru sebagai head *linked-list*.

PRAKTEK 29

```
# Praktek 29 : Membuat Double Linked-List
# membuat node baru
def buat_node_double(data):
    return {'data': data, 'prev': head, 'next': None}

# Menambahkan node baru di awal double linked-list
def tambah_node_depan(head, data):
    new_node = buat_node_double(data)
    new_node['next'] = head
    new_node['prev'] = None

    if head is not None:
        head['prev'] = new_node

    return new_node

# Mencetak double linked-list dengan traversal maju
def cetak_dll(head):
    current = head
    print('HEAD', end=' <-> ')
    while current:
        print(current['data'], end=' <-> ')
        current = current['next']
    print('NULL')
```

```
# Penerapannya
# Head awal dari linked-list
head = None

# Tambah Node
head = tambah_node_depan(head, 16) # 16
head = tambah_node_depan(head, 19) # 16 <-> 19

# Cetak double linked-list sebelum penyisipan di awal node
print("Double Linked-list Awal Sebelum Penyisipan : \n", end='')
cetak_dll(head)

# Tambah Node didepan double linked-list
head = tambah_node_depan(head, 22) # 16 <-> 19 <-> 22
head = tambah_node_depan(head, 99) # 16 <-> 19 <-> 22 <-> 99

# Cetak double linked-list setelah penyisipan di awal node
print("\nDouble Linked-list Awal Setelah Penyisipan: \n", end='')
cetak_dll(head)
```

HASIL

```
Double Linked-list Awal Sebelum Penyisipan :
HEAD <-> 19 <-> 16 <-> NULL

Double Linked-list Awal Setelah Penyisipan:
HEAD <-> 99 <-> 22 <-> 19 <-> 16 <-> NULL
```

PENJELASAN

Baris 1-2:Kode diatas adalah sebuah fungsi dengan
nama buat_node_double(data), yang menerima satu parameter yaitu data.
Dimana node baru ini nantinya akan memiliki komponen seperti ini node = {

'data': data, # bergantung pada value yang diinputkan

'prev': None,

'next': None

}

Baris 3-9: Pada potongan kode diatas terdapat 5 langkah, yang terdiri dari :

1. Langkah 1 : membuat node baru dengan memanggil fungsi buat_node_double(data)
2. Langkah 2 : setelah node baru terbuat, ubah nilai dari bagaian next dari node untuk menyambungkannya dengan head sebelumnya.
3. Langkah 3 : prev dari node baru diberi nilai None
4. Langkah 4 : jika list tidak kosong, head['prev'] node sebelumnya arahkan ke node baru
5. Langkah 5 : node baru menjadi head baru

Penyisipan di akhir melibatkan penelusuran seluruh *linked-list* hingga mencapai node terakhir. Kemudian, kita tetapkan referensi berikutnya (next) dari node terakhir untuk menunjuk ke node baru dan referensi sebelumnya (prev) dari node baru untuk menunjuk ke node terakhir. Dengan demikian, node baru menjadi elemen terakhir dalam *linked-list*.

Berikut ini adalah algoritmanya :

1. Buat node baru (new_node) dengan data yang baru
2. cek apakah head kosong :
    - Jika kosong -> node baru langsung jadi head
3. Jika tidak kosong :
    - Traversal (jelajahi) dari head ke tail (node akhir)
    - Atur :
        - tail['next'] = new_node
        - new_node['prev'] = tail
4. Kembalikan head (karena head tidak berubah)

Berikut adalah implementasi algoritma penyisipan node baru di akhir *double linked-list* :

PRAKTEK 30

```python
# Praktek 30 : Membuat Double Linked-List di Akhir
# membuat node baru
def buat_node_double(data):
    return {'data': data, 'prev': head, 'next': None}

# Menambahkan node baru di akhir double linked-list
def tambah_node_akhir(head, data):
    new_node = buat_node_double(data)

    # Jika list kosong
    if head is None:
        return new_node

    # Jika list tidak kosong, cari node terakhir
    current = head
    while current['next'] is not None:
        current = current['next']

    # Sambungkan node terakhir ke node baru
    current['next'] = new_node
    new_node['prev'] = current

    return head
```

```
# Mencetak double linked-list dengan traversal maju
def cetak_dll(head):
    current = head
    print('HEAD', end=' <-> ')
    while current:
        print(current['data'], end=' <-> ')
        current = current['next']
    print('NULL')

# Penerapannya
# Head awal dari linked-list
head = None

# Tambah Node
head = tambah_node_depan(head, 16) # 16
head = tambah_node_depan(head, 19) # 19 <-> 16

# Cetak double linked-list sebelum penyisipan di akhir node
print("Double Linked-list Sebelum Penyisipan diakhir: \n", end='')
cetak_dll(head)

# Tambah Node diakhir double linked-list
head = tambah_node_akhir(head, 22) # 19 <-> 16 <-> 22
head = tambah_node_akhir(head, 99) # 19 <-> 16 <-> 22  <-> 19
```

```
# Cetak double linked-list setelah penyisipan di akhir node
print("\nDouble Linked-list Setelah Penyisipan diakhir: \n", end='')
cetak_dll(head)
```

HASIL

```
Double Linked-list Sebelum Penyisipan diakhir:
HEAD <-> 19 <-> 16 <-> NULL

Double Linked-list Setelah Penyisipan diakhir:
HEAD <-> 19 <-> 16 <-> 22 <-> 99 <-> NULL
```

PENJELASAN

Baris 1: bertujuan untuk membuat node baru dengan nilai data pada akhir *double linked-list*. Terdapat dua parameter yaitu :

1. head yang merupakan node pertama (awal) dari *linked-list*
2. data yang merupakan nilai (isi) yang akan dimasukkan dalam node baru

baris 2: baris kode ini digunakan untuk membuat node baru dengan memanggil fungsi diluar yaitu buat_node_double(data).
Variabel new_node menjadi *dictionary* yang mewakili node baru, misalnya :

{'data': 50, 'prev': None, 'next': None}

node baru dengan nilai data = 50, prev = None, dan next = None.
Nilai prev dan next masih bernilai None karena belum terhubung dengan node lainnya.

Baris 3-4:Potongan kode ini berfungsi untuk pengecekan validitas, apakah list dalam kondisi kosong atau tidak, jiak iya maka new_node yang baru dibuat, langsung menjadi head dari *double linked-list*, dan kemudian node baru dikembalikan return.

Baris 5-7: dengan potongan kode diatas kita akan menjelajahi (traversal) dari node pertama sama akhir. Sebelum penjelajahan dimulai pertama kita buat variabel bantuan yaitu current yang akan menampung pointer dari setiap node dalam hal ini karena penjelajahan dilakukan maju jadi yang dibutuhkan hanya bagian pointer next saja.

Selanjutnya while current['next'] is not None: atau selama isi dari next pointer tidak None, maka perintah current = current['next'] dijalankan, artinya pindah ke node berikutnya. Perulangan ini akan dilakuakan sampai dengan current['next'] == None atau sudah tidak ada node lagi (node terakhir).

Baris 8-10: kita akan menyambungkan pointer next dari node terakhir yang awalnya None menjadi new_node, current['next'] = new_node, dan pointer prev dari new_node menjadi current, new_node['prev'] = current.

Terakhir kita kembalikan nilai head dari *linked-list*.

Penyisipan node pada Tengah atau Posisi Spesifik Double *Linked-List*.

Idenya adalah untuk menelusuri (traversal) *linked list* guna menemukan node pada posisi - 1, katakanlah node saat ini. Jika posisinya valid, buat node baru dengan data yang diberikan dan perbarui pointer-nya: Tetapkan pointer next dari node baru ke next dari node saat ini dan pointer prev dari node baru ke node saat ini. Demikian pula, perbarui pointer next dari node saat ini ke node baru dan pointer prev dari node baru di next ke node baru

Algoritmanya adalah sebagai berikut ini :

1. Buat node baru dengan data baru
2. Lakukan pengecekan validitas, jika posisi node = 1, maka gunakan fungsi sisip depan
3. Jika posisi < 0, maka munculkan pesan error posisi tidak valid
4. jika posisi bukan 0, atau kurang dari 0, maka lakukan traversal dari head sampai dengan posisi yang diinginkan
5. sisipkan node baru diantara dua node
6. update pointer next dan prev dari node baru agar terhubung
7. kembalikan nilai head

PRAKTEK 31

```
# membuat node baru
def buat_node_double(data):
    return {'data': data, 'prev': head, 'next': None}

# Menambahkan node baru di awal double linked-list
def tambah_node_depan(head, data):
    new_node = buat_node_double(data)
    new_node['next'] = head
    new_node['prev'] = None

    if head is not None:
        head['prev'] = new_node

    return new_node

# sisip node diposisi mana saja
def sisip_double_dimana_aja(head, data, position):
    # membuat node baru
    new_node = buat_node_double(data)

    # cek jika posisi = 0 gunakan fungsi tambah_node_depan()
    if position == 0:
        return tambah_node_depan(head, data)
```

```
# cek jika posisi = 0 gunakan fungsi tambah_node_depan()
if position == 0:
    return tambah_node_depan(head, data)

# cek jika posisi < 0, input tidak valid
if position < 0:
    print('\nPosisi Tidak Valid')
    return head

# deklarasi node pertama
current = head
index = 1

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

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

```
    # sisipkan node diantara current dan current.next
    next_node = current['next']
    current['next'] = new_node
    new_node['prev'] = current
    new_node['next'] = next_node
    if next_node is not None:
        next_node['prev'] = new_node


    return head

# Mencetak double linked-list dengan traversal maju
def cetak_dll(head):
    current = head
    print('HEAD', end=' <-> ')
    while current:
        print(current['data'], end=' <-> ')
        current = current['next']
    print('NULL')

# Penerapannya
# Head awal dari linked-list
head = None
```

```
# Tambah Node
head = tambah_node_depan(head, 16) # 16
head = tambah_node_depan(head, 19) # 16 <-> 19

# Cetak double linked-list sebelum penyisipan di awal node
print("Double Linked-list Awal Sebelum Penyisipan Tengah: \n", end='')
cetak_dll(head)

# Tambah Node pada posisi mana saja, di double linked-list
head = sisip_double_dimana_aja(head, 22, 1) # 19 <-> 22 <-> 16
head = sisip_double_dimana_aja(head, 10, 2) # 19 <-> 10 <-> 22 <-> 16
head = sisip_double_dimana_aja(head, 30, 3) # 9 <-> 10 <-> 30 <-> 22 <-> 16

# Cetak double linked-list setelah penyisipan di awal node
print("\nDouble Linked-list Awal Setelah Penyisipan Tengah: \n", end='')
cetak_dll(head)
```

HASIL

```
Double Linked-list Awal Sebelum Penyisipan Tengah:
HEAD <-> 19 <-> 16 <-> NULL

Double Linked-list Awal Setelah Penyisipan Tengah:
HEAD <-> 19 <-> 10 <-> 30 <-> 22 <-> 16 <-> NULL
```

PENJELASAN

Baris 1: bertujuan untuk menambahkan atau menyisipkan sebuah node ditengah *double linked-list*. Fungsi ini menerima tiga parameter yaitu :

1. head, merupakan node pertama dari *linked-list*
2. data, merupakan nilai pada node yang akan disisipkan
3. position, merupakan posisi (indeks, mulai dari 0) dimana lokasi node baru akan disispkan ke ``*linked-list*

baris 2: perintah ini digunakan untuk membuat node baru dengan memanggil fungsi buat_node_double(data) diluar fungsi ini, dan akan mengembalikan nilai berupa *dictionary* seperti dibawah ini :

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

node baru dengan nilai data = 50, prev = None, dan next = None. Nilai prev dan next masih bernilai None karena belum terhubung dengan node lainnya.

Baris 3-4:Potongan kode ini akan melakukan pengecekan validitas posisi node, dimana jika posisi yang diinputkan adalah sama dengan 0 if position == 0, artinya node baru ingin diletakkan pada awal *linked-list*. Oleh karena itu kita akan memanggil fungsi tambah_node_depan() untuk melakukannya.

Baris 5-7:Potongan kode diatas adalah pengecekan validatas inputan dari user, yang mana mengecek apakah posisi node kurang dari 0 (if position < 0>), jika ya, maka tampilkan pesan print('\nPosisi Tidak Valid') dan kembalikan nilai head semula. Potongan kode ini bertujuan untuk mengantisipasi user memasukkan posisi kurang dari 0.

Baris 8-12: pertama adalah kita deklarasikan lebih dulu variabel current sebagai variabel sementara yang kita gunakan untuk memulai proses traversal, dimana variabel ini diberi nilai awal node pertama (current = head). Selain itu kita buat juga variabel penghitung posisi yaitu index yang diberi nilai awal adalah 1, yang menunjukkan proses traversal dimulai dari node berindex 1.

Berikutnya while current is not None and index < position - 1, selama current tidak sama dengan None dan index kurang dari position - 1 maka perintah current = current['next'] dijalankan artinya pindah ke node berikutnya. Selain itu update juga nilai dari index dengan menambahkan 1, index +=1 agar kita tahu ada diposisi ke berapa. Perulangan ini akan berhenti sampai node sebelum index node yang menjadi tujuan kita. Contoh node baru akan disisipkan pada index ke 3 maka, traversal akan berhenti tepat satu node sebelumnya, yaitu pada index ke 2.

Baris 13-16:potongan kode diatas untuk mengecek validitas posisi yang diinputkan oleh user tidak melebih panjang dari *linked-list*, maka tampilkan pesan print("Posisi melebihi panjang linked list!") dan kembalikan nilai head semula

Baris 17-23: Lalu bagaimana cara menyisipkannya? Sederhananya kita akan menyisipkan sebuah node diantar dua node yaitu :

1. Setelah current, dan
2. Sebelum current['next'] yang kita sebut next_node

kode next_node = current['next'] berfungsi memindahkan node pada posisi sebelum node yang ingin sisipkan ke dalam variabel next_node. Artinya ini adalah node setelah node yang akan disispkan.

Sebagai analogi, Misalkan kita ingin menyisipkan seorang siswa baru (new_node) di barisan antara siswa bernama **Andi** (current) dan **Budi** (next_node). kita simpan dulu informasi tentang siapa Budi next_node = current['next'], agar tidak "terlupakan" setelah kita ubah koneksi antara Andi dan siswa baru nanti

Selanjutnya kode current['next'] = new_node untuk mengubah koneksi current, sehingga current sekarang node berikutnya dalah new_node. Analoginya kita meminta **siswa baru** untuk baris di belakang **Andi**. Artinya setelah Andi ada siswa baru dibelakangya.

Lalu node baru harus mengetahui node apa yang ada didepannya oleh karena itu kode new_node['prev'] = current. Analoginya **siswa baru** mengetahui **Andi** berada didepannya yang artinya Andi adalah node sebelum **siswa baru**.

Nah selanjutnya **siswa baru** juga harus tahu siapa setelah dia, karenanya kode new_node['next'] = next_node digunakan agar node baru terkoneksi dengan node setelahnya ,yaitu next_node. Analoginya ketika **siswa baru** ditanya siapa setelahnya dia tahu **Budi** setelahnya.

Selanjutnya kita perlu memvalidasi apakah node setelah node baru benar-benar ada if next_node is not None, jika next_node == None berarti node_baru ditempatkan di akhir list. Namun jika next_node is not None maka kita perlu next_node perlu mengubah pointernya agar node sebelumnya adalah new_node. Analoginya kita memberi tahu **Budi** bahwa siswa sebelum dia adalah **siswa baru** dan bukan **Andi** lagi.