PENJELASAN DARI CODINGAN TERSEBUT :

```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None
```

- Node adalah kelas yang merepresentasikan elemen dari double linked list.
- self.data menyimpan nilai data dari node.
- self.prev menunjuk ke node sebelumnya (awal diset None).
- self.next menunjuk ke node berikutnya (juga awalnya None).

```
class DoubleLinkedList:
    def __init__(self):
        self.head = None
```

- DoubleLinkedList adalah kelas utama yang mewakili double linked list.
- self.head menyimpan node pertama dari list. Jika belum ada elemen, bernilai None.

```
def append(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
        return
    cur = self.head
    while cur.next:
        cur = cur.next
    cur.next = new_node
    new_node.prev = cur
```

- Fungsi ini menambahkan node baru (new_node) ke akhir list.
- Jika list masih kosong, node baru menjadi head.
- Jika sudah ada data, iterasi sampai node terakhir (cur).
- cur.next = new_node: node lama diarahkan ke node baru.
- new_node.prev = cur: node baru diarahkan balik ke node sebelumnya (cur).

```
def display(self):
    cur = self.head
    while cur:
        print(cur.data, end=" <-> " if cur.next else "\n")
        cur = cur.next
```

- Menampilkan isi list dari awal ke akhir.
- Jika node selanjutnya ada, tampilkan tanda <->, jika tidak cukup cetak data.

```
def delete_first(self):
    if self.head is None:
        print("List kosong")
        return
    print(f"Menghapus node awal: {self.head.data}")
    self.head = self.head.next
    if self.head:
        self.head.prev = None
```

- Menghapus node pertama.
- Jika list kosong, beri pesan.
- Arahkan head ke node berikutnya.
- Jika masih ada node setelahnya, putus hubungan ke node lama dengan prev = None.

```
def delete_last(self):
    if self.head is None:
        print("List kosong")
        return
    cur = self.head
    if cur.next is None:
        print(f"Menghapus node terakhir: {cur.data}")
        self.head = None
        return
    while cur.next:
        cur = cur.next
    print(f"Menghapus node terakhir: {cur.data}")
    cur.prev.next = None
```

- Menghapus node terakhir.
- Jika hanya satu node, langsung kosongkan list (head = None).
- Jika lebih, iterasi ke akhir dan putus hubungan node terakhir dengan sebelumnya.

```
def delete_by_value(self, value):
    if self.head is None:
        print("List kosong")
        return
    cur = self.head
    while cur:
```

```
        if cur.data == value:
            print(f"Menghapus node dengan nilai: {value}")
            if cur.prev:
                cur.prev.next = cur.next
            else:
                self.head = cur.next
            if cur.next:
                cur.next.prev = cur.prev
            return
        cur = cur.next
    print(f"Data {value} tidak ditemukan dalam list.")
```

- Mencari dan menghapus node dengan nilai tertentu.
- Jika node yang ditemukan adalah head, maka head digeser.
- Jika di tengah, sambungkan node sebelumnya dan sesudahnya.
- Jika tidak ditemukan, beri pesan bahwa data tidak ada.

```
dll = DoubleLinkedList()
dll.append(10)
dll.append(20)
dll.append(30)
dll.append(40)
```

- Membuat objek double linked list dan mengisi dengan empat data: 10, 20, 30, 40.

```
print("Isi awal:")
dll.display()
```

- Menampilkan isi awal list.

```
dll.delete_first()
dll.display()
```

- Menghapus node pertama (10), lalu tampilkan list.

```
dll.delete_last()
dll.display()
```

- Menghapus node terakhir (40), lalu tampilkan list.

```
dll.delete_by_value(20)
```

```
dll.display()
```

- Menghapus node dengan data 20, lalu tampilkan hasilnya.

```
dll.delete_by_value(99)
```

- Mencoba menghapus nilai 99 yang tidak ada dalam list.
- Akan menampilkan pesan bahwa data tidak ditemukan.