Kelas Node

```python
class Node:
    """Class representing a node in a doubly linked list."""
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None
```

- **`class Node:`** Mendefinisikan kelas `Node` yang merepresentasikan sebuah node dalam doubly linked list.

- **`def __init__(self, data):`** Merupakan konstruktor yang diinisialisasi dengan parameter `data`, yang akan disimpan dalam node.

- **`self.data = data:`** Menyimpan nilai data yang diberikan ke dalam atribut `data` dari node.

- **`self.prev = None:`** Menginisialisasi atribut `prev` yang akan menunjuk ke node sebelumnya, diatur ke `None` pada awalnya.

- **`self.next = None:`** Menginisialisasi atribut `next` yang akan menunjuk ke node berikutnya, diatur ke `None` pada awalnya.

Kelas DoublyLinkedList

```python
class DoublyLinkedList:
    """Doubly linked list with insert and deletion operations."""
    def __init__(self):
        self.head = None
```

- **`class DoublyLinkedList:`** Mendefinisikan kelas `DoublyLinkedList` yang merepresentasikan struktur data linked list ganda.

- **`def __init__(self):`** Merupakan konstruktor yang diinisialisasi tanpa parameter.

- **`self.head = None:`** Menginisialisasi atribut `head` yang akan menunjuk ke node pertama dalam list, diatur ke `None` pada awalnya.

Metode Append

```python
def append(self, data):
    """Append a node with specified data to the end of the list."""
    new_node = Node(data)
```

- **`def append(self, data):`** Mendefinisikan metode `append` untuk menambahkan node baru ke akhir list.
- **`new_node = Node(data):`** Membuat instance baru dari `Node` dengan data yang diberikan.

```python
    if self.head is None:
        self.head = new_node
        return
```

- **`if self.head is None:`** Memeriksa apakah list kosong (tidak ada node).
- **`self.head = new_node:`** Jika kosong, node baru menjadi head dari list.
- **`return:`** Menghentikan eksekusi metode jika node baru ditambahkan sebagai head.

```python
    last = self.head
    while last.next:
        last = last.next
```

- **`last = self.head:`** Menginisialisasi variabel `last` untuk menunjuk ke node terakhir dalam list.
- **`while last.next:`** Mengulangi hingga mencapai node terakhir (yang tidak memiliki node berikutnya).

```python
    last.next = new_node
    new_node.prev = last
```

- **`last.next = new_node:`** Menghubungkan node terakhir dengan node baru.

- **`new_node.prev = last:`** Mengatur atribut `prev` dari node baru untuk menunjuk ke node terakhir.

Metode Delete dari Awal

```python

def delete_from_start(self):

"""Delete the first node of the list."""

if not self.head:

print("List is empty. No node to delete from start.")

return

```

- **`def delete_from_start(self):`** Mendefinisikan metode untuk menghapus node pertama dari list.

- **`if not self.head:`** Memeriksa apakah list kosong.

- **`print("List is empty. No node to delete from start.")`** Mencetak pesan jika list kosong.

- **`return:`** Menghentikan eksekusi metode jika list kosong.

```python

print(f"Deleting node from start with data: {self.head.data}")

```

- **`print(f"Deleting node from start with data: {self.head.data}")`** Mencetak data dari node yang akan dihapus.

```python

if self.head.next is None:

self.head = None

```

- **`if self.head.next is None:`** Memeriksa apakah hanya ada satu node dalam list.

- **`self.head = None:`** Mengatur head ke `None` jika hanya ada satu node.

```python

else:

```
        self.head = self.head.next
        self.head.prev = None
```

- **`else:`** Jika ada lebih dari satu node, lakukan langkah berikut.
- **`self.head = self.head.next:`** Mengatur head ke node berikutnya.
- **`self.head.prev = None:`** Mengatur atribut `prev` dari node baru menjadi `None`.

Metode Delete dari Akhir

```python

```
def delete_from_end(self):
    """Delete the last node of the list."""
    if not self.head:
        print("List is empty. No node to delete from end.")
        return
```

- **`def delete_from_end(self):`** Mendefinisikan metode untuk menghapus node terakhir dari list.
- **`if not self.head:`** Memeriksa apakah list kosong.
- **`print("List is empty. No node to delete from end.")`** Mencetak pesan jika list kosong.
- **`return:`** Menghentikan eksekusi metode jika list kosong.

```python
    last = self.head
    if last.next is None:
        print(f"Deleting the only node in list with data: {last.data}")
        self.head = None
        return
```

- **`last = self.head:`** Menginisialisasi variabel `last` untuk menunjuk ke node terakhir.
- **`if last.next is None:`** Memeriksa apakah hanya ada satu node dalam list.
- **`print(f"Deleting the only node in list with data: {last.data}")`** Mencetak data dari node yang akan dihapus.

- **`self.head = None:`** Mengatur head ke `None` jika hanya ada satu node.

- **`return:`** Menghentikan eksekusi metode.

```python
    while last.next:
        last = last.next
```

- **`while last.next:`** Mengulangi hingga mencapai node terakhir.

```python
    print(f"Deleting node from end with data: {last.data}")
    last.prev.next = None
```

- **`print(f"Deleting node from end with data: {last.data}")`** Mencetak data dari node yang akan dihapus.

- **`last.prev.next = None:`** Menghubungkan node sebelumnya dengan `None`, menghapus node terakhir dari list.

Metode Delete Berdasarkan Nilai

```python
def delete_by_value(self, data):
    """Delete the first node found with the specified data value."""
    if not self.head:
        print("List is empty. No node to delete by value.")
        return
```

- **`def delete_by_value(self, data):`** Mendefinisikan metode untuk menghapus node berdasarkan nilai data.

- **`if not self.head:`** Memeriksa apakah list kosong.

- **`print("List is empty. No node to delete by value.")`** Mencetak pesan jika list kosong.

- **`return:`** Menghentikan eksekusi metode jika list kosong.

```python
    current = self.head
    while current:
```

- **`current = self.head:`** Menginisialisasi variabel `current` untuk menunjuk ke node pertama.
- **`while current:`** Mengulangi hingga mencapai akhir list.

```python
        if current.data == data:
            print(f"Deleting node with data: {data}")
```

- **`if current.data == data:`** Memeriksa apakah data dari node saat ini sama dengan data yang ingin dihapus.
- **`print(f"Deleting node with data: {data}")`** Mencetak data dari node yang akan dihapus.

```python
            if current.prev is None:
                self.delete_from_start()
```

- **`if current.prev is None:`** Memeriksa apakah node yang akan dihapus adalah head.
- **`self.delete_from_start():`** Memanggil metode untuk menghapus node dari awal.

```python
            elif current.next is None:
                self.delete_from_end()
```

- **`elif current.next is None:`** Memeriksa apakah node yang akan dihapus adalah node terakhir.
- **`self.delete_from_end():`** Memanggil metode untuk menghapus node dari akhir.

```python
            else:
```

```
            current.prev.next = current.next
            current.next.prev = current.prev
```

- **`else:`** Jika node yang akan dihapus berada di tengah.
- **`current.prev.next = current.next:`** Menghubungkan node sebelumnya dengan node berikutnya.
- **`current.next.prev = current.prev:`** Menghubungkan node berikutnya dengan node sebelumnya.

```python
        return  # Delete only the first occurrence
```

- **`return:`** Menghentikan eksekusi metode setelah menghapus node pertama yang ditemukan.

```python
      current = current.next
```

- **`current = current.next:`** Melanjutkan ke node berikutnya.

```python
   print(f"Node with data {data} not found in the list.")
```

- **`print(f"Node with data {data} not found in the list.")`** Mencetak pesan jika node dengan data yang dicari tidak ditemukan.

Metode Print List

```python

def print_list(self):

**   """Print the elements of the list."""**

**   if not self.head:**

**   print("List is empty.")**

**   return**

```

- **`def print_list(self):`** Mendefinisikan metode untuk mencetak elemen dalam list.
- **`if not self.head:`** Memeriksa apakah list kosong.
- **`print("List is empty.")`** Mencetak pesan jika list kosong.
- **`return:`** Menghentikan eksekusi metode jika list kosong.

```python
    current = self.head
    values = []
```

- **`current = self.head:`** Menginisialisasi variabel `current` untuk menunjuk ke node pertama.
- **`values = []:`** Membuat list kosong untuk menyimpan nilai-nilai node.

```python
    while current:
        values.append(str(current.data))
        current = current.next
```

- **`while current:`** Mengulangi hingga mencapai akhir list.
- **`values.append(str(current.data)):`** Menambahkan data dari node saat ini ke dalam list `values`.
- **`current = current.next:`** Melanjutkan ke node berikutnya.

```python
    print(" <-> ".join(values))
```

- **`print(" <-> ".join(values)):`** Mencetak semua nilai dalam list yang dipisahkan oleh " <-> ".

Blok Utama

```python

if __name__ == "__main__":
```

```
# Demonstration of operations
dll = DoublyLinkedList()
print("Adding nodes 10, 20, 30, 40, 50 to the list.")
for val in [10, 20, 30, 40, 50]:
    dll.append(val)
print("Current list:")
dll.print_list()
```

- **`if __name__ == "__main__":`** Memastikan bahwa kode di bawahnya hanya dijalankan jika file ini dieksekusi sebagai program utama.
- **`dll = DoublyLinkedList():`** Membuat instance baru dari `DoublyLinkedList`.
- **`print("Adding nodes 10, 20, 30, 40, 50 to the list.")`** Mencetak pesan untuk menunjukkan bahwa node akan ditambahkan.
- **`for val in [10, 20, 30, 40, 50]:`** Mengulangi untuk setiap nilai dalam list.
- **`dll.append(val):`** Menambahkan nilai ke dalam list.
- **`print("Current list:")`** Mencetak pesan untuk menunjukkan list saat ini.
- **`dll.print_list():`** Memanggil metode untuk mencetak elemen dalam list.

```python
print("\nDelete node from start:")
dll.delete_from_start()
dll.print_list()
```

- **`print("\nDelete node from start:")`** Mencetak pesan untuk menunjukkan bahwa node dari awal akan dihapus.
- **`dll.delete_from_start():`** Memanggil metode untuk menghapus node dari awal.
- **`dll.print_list():`** Memanggil metode untuk mencetak elemen dalam list setelah penghapusan.

```python
print("\nDelete node from end:")
dll.delete_from_end()
dll.print_list()
```

```

- **`print("\nDelete node from end:")`** Mencetak pesan untuk menunjukkan bahwa node dari akhir akan dihapus.

- **`dll.delete_from_end():`** Memanggil metode untuk menghapus node dari akhir.

- **`dll.print_list():`** Memanggil metode untuk mencetak elemen dalam list setelah penghapusan.

```python
    print("\nDelete node by value 30:")
    dll.delete_by_value(30)
    dll.print_list()
```

- **`print("\nDelete node by value 30:")`** Mencetak pesan untuk menunjukkan bahwa node dengan nilai 30 akan dihapus.

- **`dll.delete_by_value(30):`** Memanggil metode untuk menghapus node dengan nilai 30.

- **`dll.print_list():`** Memanggil metode untuk mencetak elemen dalam list setelah penghapusan.

```python
    print("\nDelete node by value 100 (not in list):")
    dll.delete_by_value(100)
    dll.print_list()
```

- **`print("\nDelete node by value 100 (not in list):")`** Mencetak pesan untuk menunjukkan bahwa node dengan nilai 100 (yang tidak ada dalam list) akan dicoba untuk dihapus.

- **`dll.delete_by_value(100):`** Memanggil metode untuk menghapus node dengan nilai 100.

- **`dll.print_list():`** Memanggil metode untuk mencetak elemen dalam list setelah percobaan penghapusan.

Dengan penjelasan ini, Anda dapat memahami bagaimana kode ini berfungsi untuk mengelola dan memanipulasi doubly linked list, termasuk menambah dan menghapus node.
```