**STRUKTUR DATA**

**STACK AND QUEUE**



Oleh:

**DANUARY BIMA HAMMAM MAYFALAH**

**24091397007**

**Program Studi D4 Manajemen Informatika**

**Fakultas Vokasi**

**Universitas Negeri Surabaya**

**2025**

1. **LAPORAN PRAKTIKUM**

**TUGAS NOMOR 1**

1. Buatlah program Stack dengan Double Linked List

a) Gunakan input angka integer berbeda untuk menguji push(), pop(), dan peek().

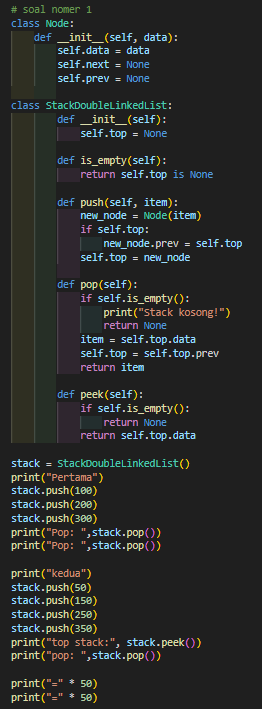
b) Skenario:

• Input pertama: Tambah elemen [100, 200, 300] lalu lakukan pop() dua kali.

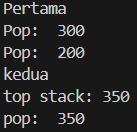
• Input kedua: Tambah elemen [50, 150, 250, 350] lalu lakukan peek() dan pop().

• Bandingkan hasilnya setelah dilakukan operasi pop() beberapa kali.

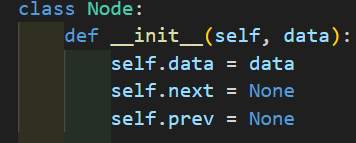
Source code



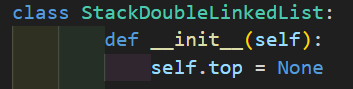
Output



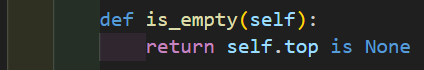
Penjelasan tiap code



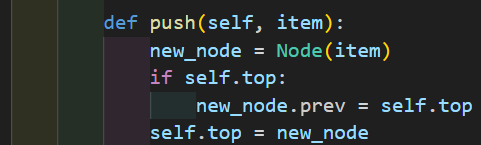
* Node adalah unit dasar dalam Double Linked List.
* Setiap Node memiliki tiga atribut:
* data: Menyimpan nilai data.
* next: Pointer ke Node berikutnya dalam linked list.
* prev: Pointer ke Node sebelumnya dalam linked list



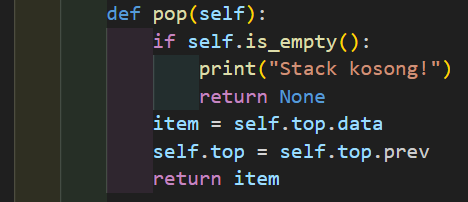
* **\_\_init\_\_**: Konstruktor untuk membuat stack kosong dengan atribut top yang merepresentasikan elemen teratas.



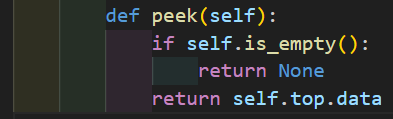
* Mengecek apakah stack kosong atau tidak.
* Jika self.top == None, berarti stack kosong, maka return True.



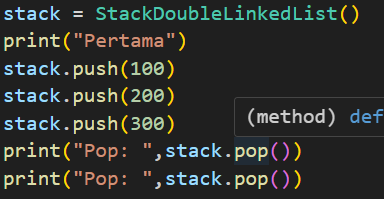
* Menambahkan item ke dalam stack.
* Membuat Node baru dengan data item.
* Jika stack tidak kosong, maka:
  + new\_node.prev = self.top → Menghubungkan node baru ke node teratas sebelumnya.
* Node baru menjadi top.

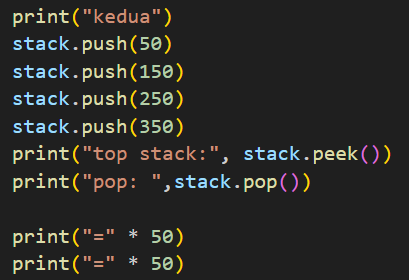


* Menghapus elemen teratas dari stack.
* Jika stack kosong, tampilkan pesan "Stack kosong!" dan return None.
* Simpan nilai self.top.data dalam variabel item.
* Geser top ke self.top.prev (node sebelumnya).
* Kembalikan nilai item.



* Mengembalikan nilai elemen teratas tanpa menghapusnya.
* Jika stack kosong, kembalikan None.





**TUGAS 2**

1. **Queue Biasa dengan Array**

**Gunakan input angka integer yang berbeda untuk menguji enqueue() dan dequeue().**

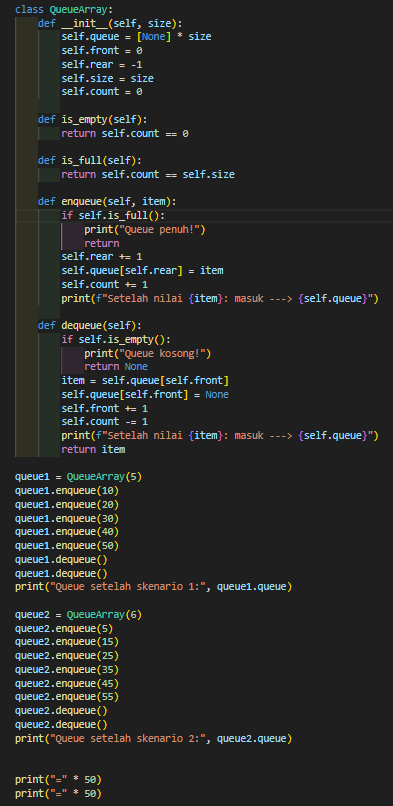
**Skenario:**

**➢ Input pertama: [10, 20, 30, 40, 50]**

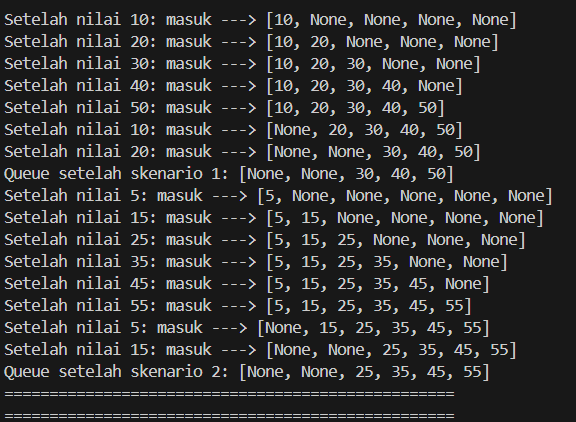
**➢ Input kedua: [5, 15, 25, 35, 45, 55]**

**➢ Bandingkan hasilnya.**

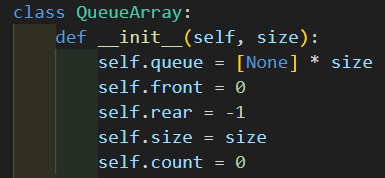
Source Code



Output



Penjelasan tiap code



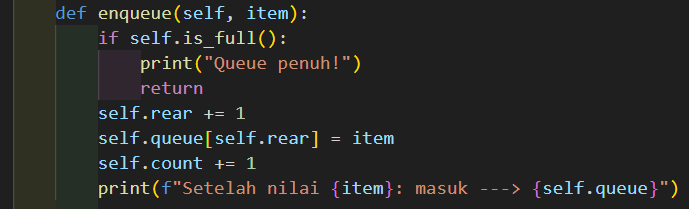
* self.queue = [None] \* size → Membuat array dengan kapasitas size, yang diisi dengan None sebagai nilai awal.
* self.front = 0 → Pointer yang menunjuk ke elemen depan antrian.
* self.rear = -1 → Pointer yang menunjuk ke elemen belakang antrian (karena antrian kosong, maka dimulai dari -1).
* self.size = size → Menyimpan ukuran maksimum dari queue.
* self.count = 0 → Menyimpan jumlah elemen saat ini dalam queue.



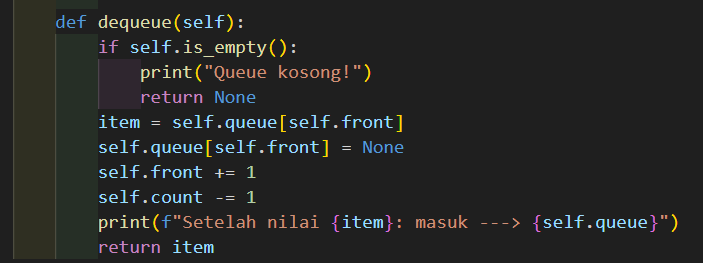
* Mengecek apakah queue kosong.
* Mengembalikan True jika **count** (jumlah elemen) adalah 0.



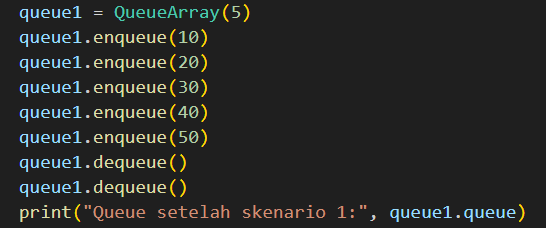
* Mengecek apakah queue penuh.
* Mengembalikan True jika count sama dengan size.



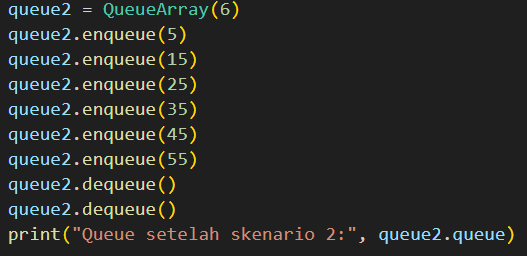
* Cek apakah queue penuh: Jika penuh, cetak "Queue penuh!" dan hentikan proses.
* Geser pointer rear (self.rear += 1) untuk menunjukkan tempat baru di queue.
* Tambahkan item ke dalam self.queue[self.rear].
* Tingkatkan jumlah elemen (self.count += 1).
* Cetak queue setelah item masuk.



* Cek apakah queue kosong: Jika kosong, cetak "Queue kosong!" dan return None.
* Ambil elemen dari depan (self.queue[self.front]).
* Hapus elemen di depan dengan mengubahnya menjadi None.
* Geser front (self.front += 1) agar menunjuk ke elemen berikutnya.
* Kurangi jumlah elemen (self.count -= 1).
* Cetak queue setelah elemen dikeluarkan.



* Membuat queue dengan kapasitas 5.
* Memasukkan elemen: 10, 20, 30, 40, 50 → Queue penuh setelah itu.
* Menghapus dua elemen (dequeue() dua kali) → 10 dan 20 dihapus.
* Cetak queue setelah proses ini.



* Membuat queue dengan kapasitas 6.
* Memasukkan elemen: 5, 15, 25, 35, 45, 55 → Queue penuh setelah itu.
* Menghapus dua elemen (dequeue() dua kali) → 5 dan 15 dihapus.
* Cetak queue setelah proses ini.

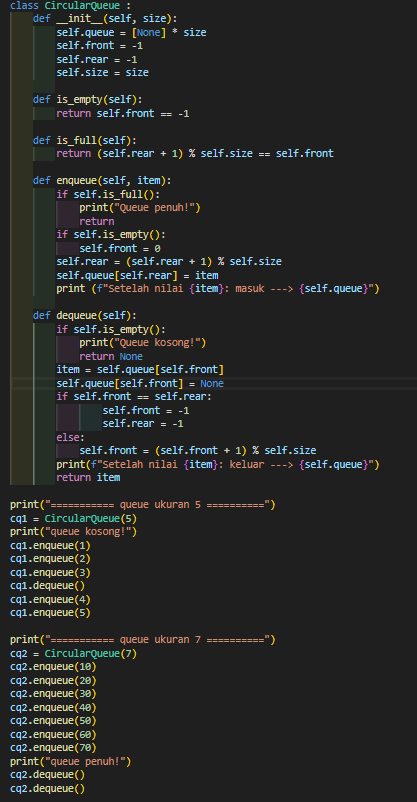
1. **Circularqueue**

**Gunakan ukuran queue yang berbeda dan input angka integer yang berbeda.**

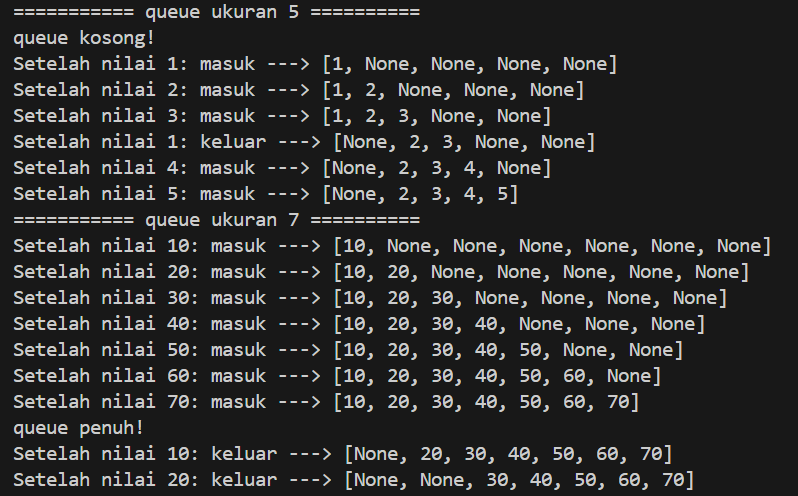
**Skenario**

* **Queue ukuran 5 dengan input: [1, 2, 3, 4, 5]**
* **Queue ukuran 7 dengan input: [10, 20, 30, 40, 50, 60, 70]**
* **Uji kondisi saat queue penuh dan kosong.**

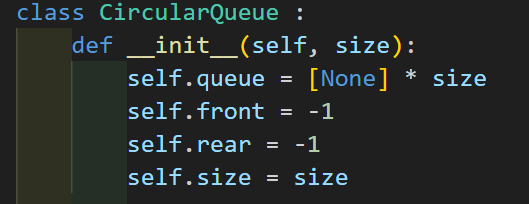
**Source code**



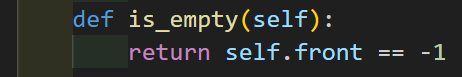
Output



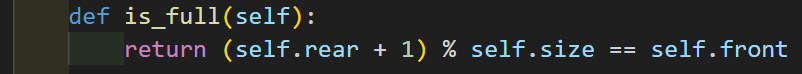
Penjelasan tiap code



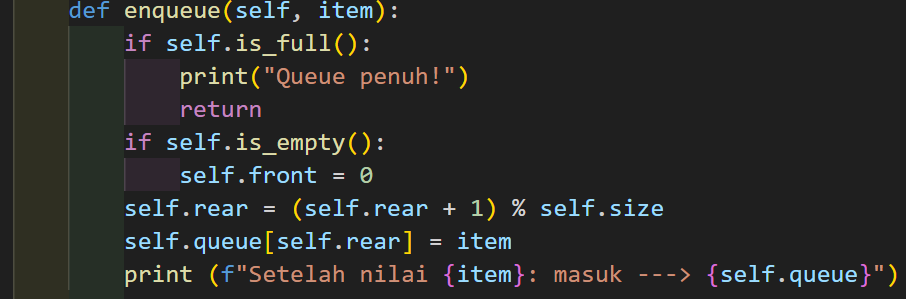
* self.queue = [None] \* size → Membuat array dengan kapasitas size, semua elemen awalnya None.
* self.front = -1 → Menunjuk ke elemen depan queue, awalnya -1 (artinya queue kosong).
* self.rear = -1 → Menunjuk ke elemen belakang queue, awalnya -1.
* self.size = size → Menyimpan ukuran maksimum queue.



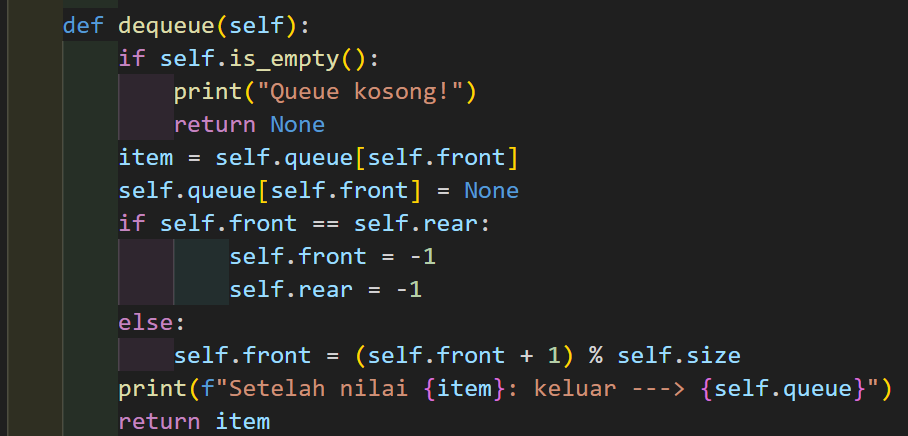
* Mengembalikan True jika front == -1, artinya queue kosong.



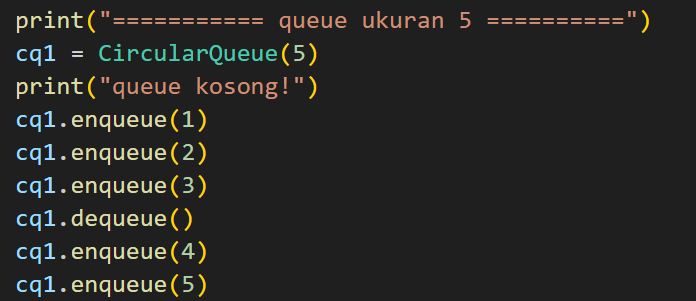
* Cek apakah queue penuh menggunakan rumus modulus:
* Jika rear + 1 sama dengan front dalam format circular, maka queue penuh.
* (self.rear + 1) % self.size == self.front memastikan jika rear sudah berada di posisi terakhir, maka ia akan kembali ke index awal.



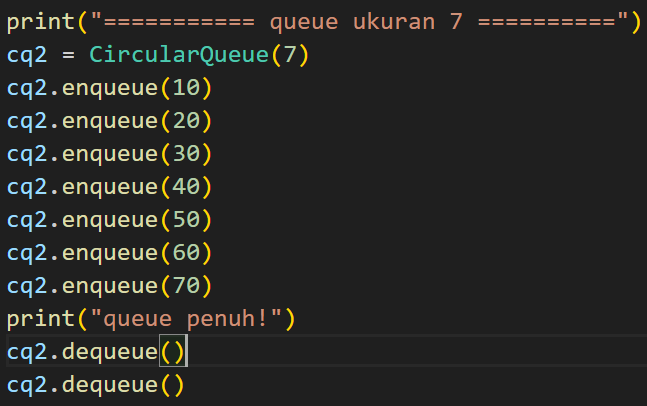
* Cek apakah queue penuh (is\_full()).
* Jika kosong (is\_empty()), set front = 0.
* Menentukan posisi rear:
* (self.rear + 1) % self.size membuat queue bersifat melingkar.
* Masukkan item ke dalam queue di indeks rear.
* Cetak kondisi queue setelah enqueue().



* Cek apakah queue kosong (is\_empty()).
* Ambil elemen di front.
* Set posisi front ke None (menghapus data).
* Jika hanya ada satu elemen (front == rear), reset queue ke keadaan kosong (front = -1, rear = -1).
* Jika tidak, front berpindah ke posisi berikutnya secara circular:
* (self.front + 1) % self.size.



* Membuat Circular Queue dengan ukuran 5.
* Cetak "queue kosong!", karena awalnya queue kosong.
* Memasukkan angka 1, 2, 3.
* Menghapus angka 1 (dequeue()).
* Memasukkan angka 4 dan 5.



* Membuat Circular Queue dengan ukuran 7.
* Memasukkan angka 10, 20, 30, 40, 50, 60, 70 (Queue penuh).
* Cetak "queue penuh!" karena queue sudah terisi maksimal.
* Menghapus angka 10 dan 20 (dequeue() dua kali).

1. **Deque (Double-ended Queue)**

Gunakan input angka integer berbeda untuk add\_front(), add\_rear(), remove\_front(), dan remove\_rear().

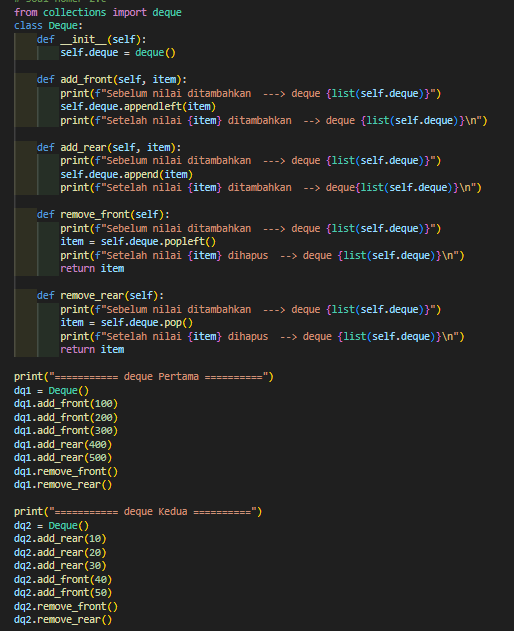
• Skenario

Input pertama: Tambah dari depan [100, 200, 300] → Tambah dari belakang[400, 500]

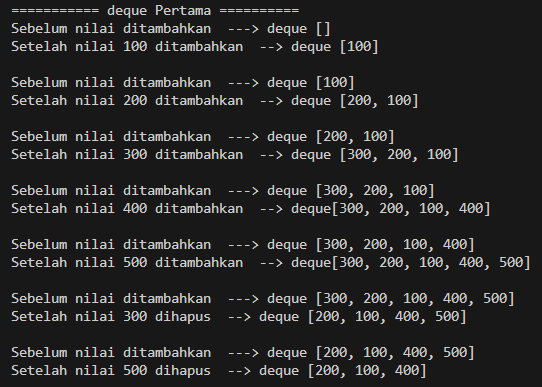
Input kedua: Tambah dari belakang [10, 20, 30] → Tambah dari depan [40, 50]

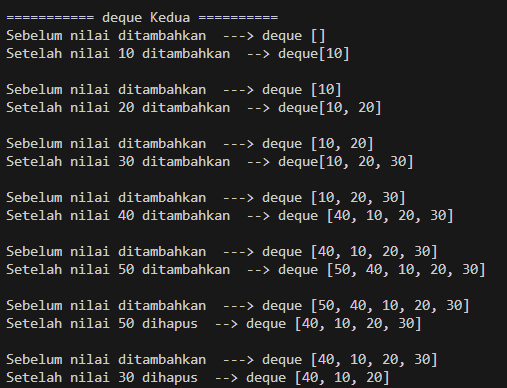
Bandingkan hasilnya setelah dilakukan operasi remove.

**Source code**

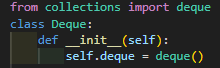


Output

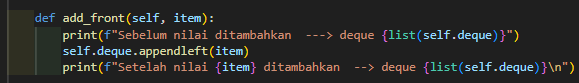




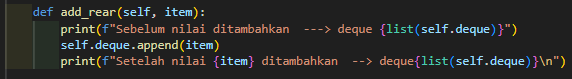
Penjelasan tiap code



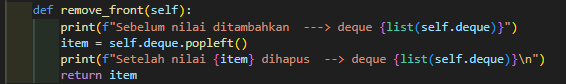
* from collections import deque → Mengimpor deque, yang merupakan implementasi deque yang lebih efisien dibandingkan list biasa.
* self.deque = deque() → Membuat objek deque kosong untuk menyimpan elemen.



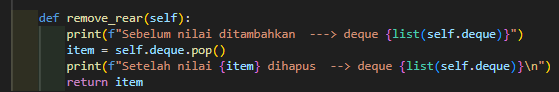
* self.deque.appendleft(item) → Menambahkan item ke depan (kiri) deque.
* Cetak kondisi deque sebelum dan sesudah penambahan.



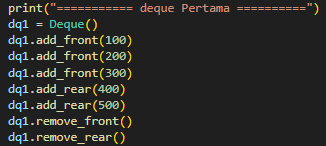
* self.deque.append(item) → Menambahkan item ke belakang (kanan) deque.
* Cetak kondisi deque sebelum dan sesudah penambahan.



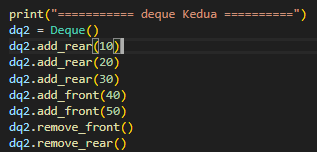
* self.deque.popleft() → Menghapus elemen dari depan (kiri) deque.
* Cetak kondisi deque sebelum dan sesudah penghapusan.
* Return elemen yang dihapus.



* self.deque.pop() → Menghapus elemen dari belakang (kanan) deque.
* Cetak kondisi deque sebelum dan sesudah penghapusan.
* Return elemen yang dihapus.



* Membuat deque kosong (dq1 = Deque()).
* Menambahkan elemen dari depan:
* dq1.add\_front(100) → [100]
* dq1.add\_front(200) → [200, 100]
* dq1.add\_front(300) → [300, 200, 100]
* Menambahkan elemen dari belakang:
* dq1.add\_rear(400) → [300, 200, 100, 400]
* dq1.add\_rear(500) → [300, 200, 100, 400, 500]
* Menghapus elemen dari depan (remove\_front()) → 300 dihapus → [200, 100, 400, 500]
* Menghapus elemen dari belakang (remove\_rear()) → 500 dihapus → [200, 100, 400]



* Membuat deque kosong (dq2 = Deque()).
* Menambahkan elemen dari belakang:
* dq2.add\_rear(10) → [10]
* dq2.add\_rear(20) → [10, 20]
* dq2.add\_rear(30) → [10, 20, 30]
* Menambahkan elemen dari depan:
* dq2.add\_front(40) → [40, 10, 20, 30]
* dq2.add\_front(50) → [50, 40, 10, 20, 30]
* Menghapus elemen dari depan (remove\_front()) → 50 dihapus → [40, 10, 20, 30]
* Menghapus elemen dari belakang (remove\_rear()) → 30 dihapus → [40, 10, 20]

1. **Priority Queue**

**Uji dengan angka integer yang berbeda dan lihat bagaimana elemen diproses berdasarkan prioritasnya.**

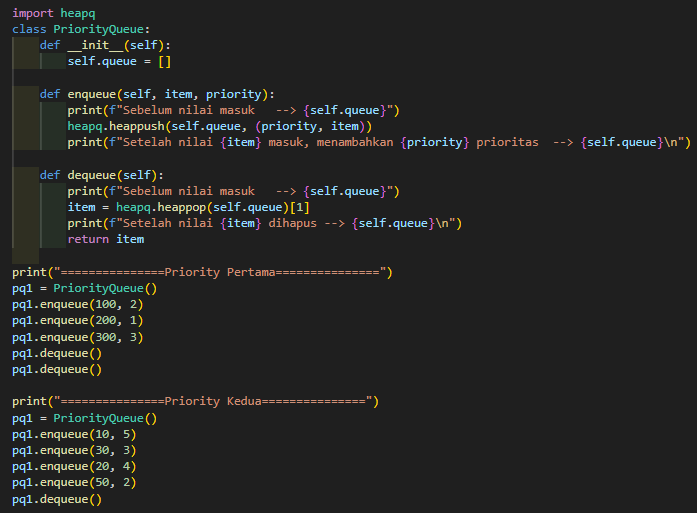
**Contoh skenario:**

**Input pertama: [(2, 100), (1, 200), (3, 50)] → Prioritas lebih kecil lebih dulu diproses.**

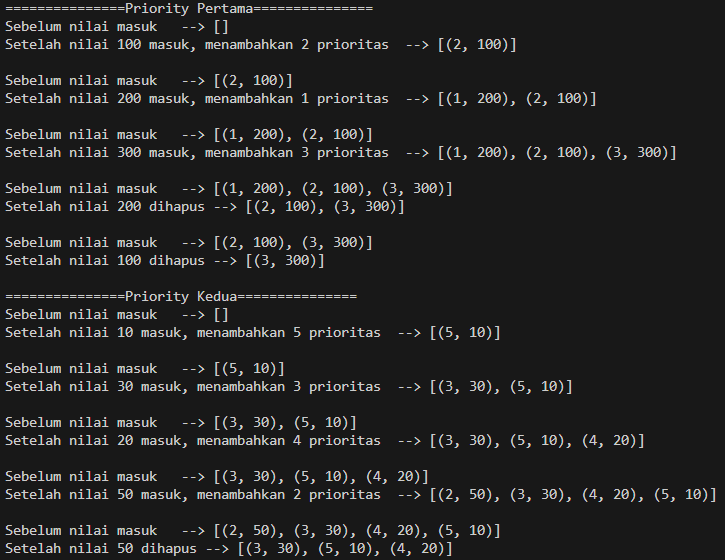
**Input kedua: [(5, 10), (3, 30), (4, 20)]**

**Bandingkan urutan elemen yang diproses.**

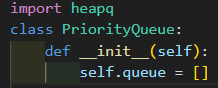
Source Code



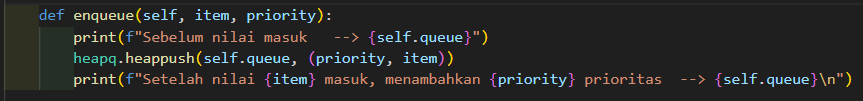
Output



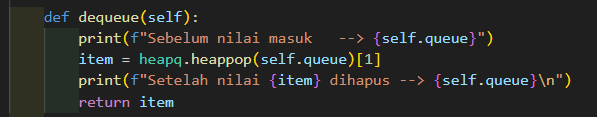
Penjelasan tiap code



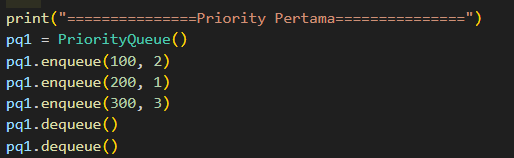
* import heapq → Mengimpor heapq, modul yang mengimplementasikan min-heap di Python.
* self.queue = [] → Membuat list kosong untuk menyimpan elemen dalam bentuk heap.



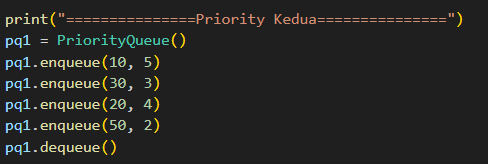
* heapq.heappush(self.queue, (priority, item)) → Menambahkan elemen ke dalam heap, di mana priority menentukan urutan.
* Elemen dengan priority terkecil akan berada di puncak heap.
* Cetak kondisi queue sebelum dan sesudah penambahan.



* heapq.heappop(self.queue) → Menghapus elemen dengan prioritas tertinggi (priority terkecil).
* Cetak kondisi queue sebelum dan sesudah penghapusan.
* Return item yang dihapus (bukan prioritasnya).



* Membuat priority queue kosong (pq1 = PriorityQueue()).
* Menambahkan elemen dengan prioritas:
* pq1.enqueue(100, 2) → [(2, 100)]
* pq1.enqueue(200, 1) → [(1, 200), (2, 100)]
* pq1.enqueue(300, 3) → [(1, 200), (2, 100), (3, 300)]
* Menghapus elemen dengan prioritas tertinggi:
* pq1.dequeue() → 200 (priority 1) keluar → [(2, 100), (3, 300)]
* pq1.dequeue() → 100 (priority 2) keluar → [(3, 300)]



* Membuat priority queue kosong (pq1 = PriorityQueue()).
* Menambahkan elemen dengan prioritas:
* pq1.enqueue(10, 5) → [(5, 10)]
* pq1.enqueue(30, 3) → [(3, 30), (5, 10)]
* pq1.enqueue(20, 4) → [(3, 30), (5, 10), (4, 20)]
* pq1.enqueue(50, 2) → [(2, 50), (3, 30), (4, 20), (5, 10)]
* Menghapus elemen dengan prioritas tertinggi:
* pq1.dequeue() → 50 (priority 2) keluar → [(3, 30), (5, 10), (4, 20)]

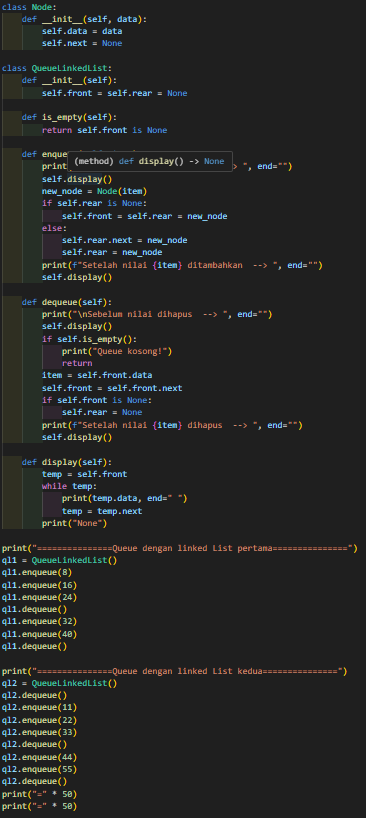
1. **Queue dengan Linked List**

**Gunakan input angka integer yang berbeda untuk menguji enqueue() dan dequeue().**

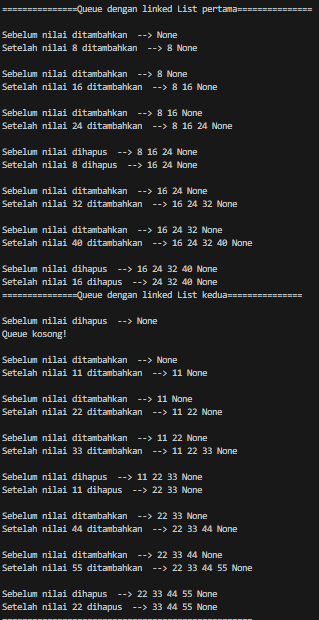
**Contoh skenario:**

* **Input pertama: [8, 16, 24, 32, 40]**
* **Input kedua: [11, 22, 33, 44, 55] Bandingkan hasil setelah operasi dequeue dilakukan.**

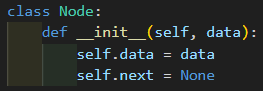
Source code



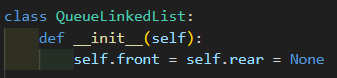
Output



Penjelasan tiap code



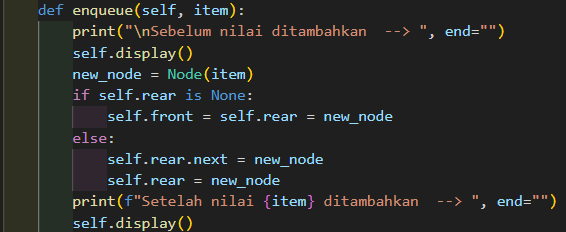
* Mendefinisikan node untuk menyimpan data dan pointer ke node berikutnya.
* self.data menyimpan nilai dari node.
* self.next menunjuk ke node berikutnya dalam antrian.



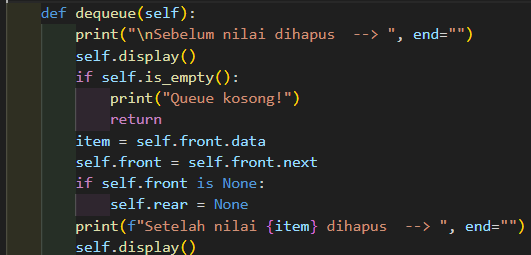
* self.front menunjuk ke elemen pertama dalam antrian.
* self.rear menunjuk ke elemen terakhir dalam antrian.



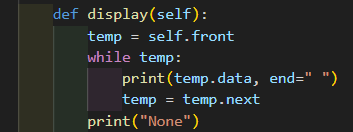
* Mengembalikan True jika antrian kosong (tidak ada elemen).
* self.front akan None jika tidak ada elemen dalam antrian.



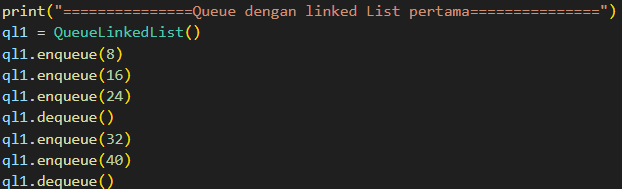
* Menampilkan kondisi antrian sebelum elemen ditambahkan.
* Membuat node baru (new\_node) dengan data item.
* Jika antrian kosong (self.rear is None):
* Node baru menjadi front dan rear.
* Jika antrian tidak kosong:
* Node baru ditambahkan setelah rear, dan rear diperbarui.
* Menampilkan kondisi antrian setelah penambahan.



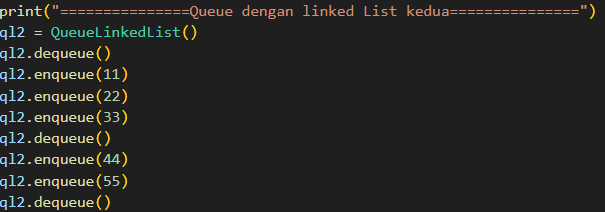
* Menampilkan kondisi antrian sebelum elemen dihapus.
* Jika antrian kosong, cetak "Queue kosong!".
* Ambil nilai dari front (elemen pertama).
* Geser front ke elemen berikutnya (self.front.next).
* Jika setelah penghapusan front menjadi None, maka rear juga harus None.
* Menampilkan kondisi antrian setelah penghapusan.



* Iterasi mulai dari front hingga None, menampilkan data dari setiap node.



* ql2 = QueueLinkedList() → Membuat antrian kosong.
* ql2.dequeue() → Antrian kosong! Cetak: "Queue kosong!"
* ql2.enqueue(11) → Tambah 11, antrian: 11 → None
* ql2.enqueue(22) → Tambah 22, antrian: 11 → 22 → None
* ql2.enqueue(33) → Tambah 33, antrian: 11 → 22 → 33 → None
* ql2.dequeue() → Hapus 11, antrian: 22 → 33 → None
* ql2.enqueue(44) → Tambah 44, antrian: 22 → 33 → 44 → None
* ql2.enqueue(55) → Tambah 55, antrian: 22 → 33 → 44 → 55 → None
* ql2.dequeue() → Hapus 22, antrian: 33 → 44 → 55 → None



* ql2 = QueueLinkedList() → Membuat antrian kosong.
* ql2.dequeue() → Antrian kosong! Cetak "Queue kosong!"
* ql2.enqueue(11) → Tambah 11, antrian: 11 → None
* ql2.enqueue(22) → Tambah 22, antrian: 11 → 22 → None
* ql2.enqueue(33) → Tambah 33, antrian: 11 → 22 → 33 → None
* ql2.dequeue() → Hapus 11, antrian: 22 → 33 → None
* ql2.enqueue(44) → Tambah 44, antrian: 22 → 33 → 44 → None
* ql2.enqueue(55) → Tambah 55, antrian: 22 → 33 → 44 → 55 → None
* ql2.dequeue() → Hapus 22, antrian: 33 → 44 → 55 → None