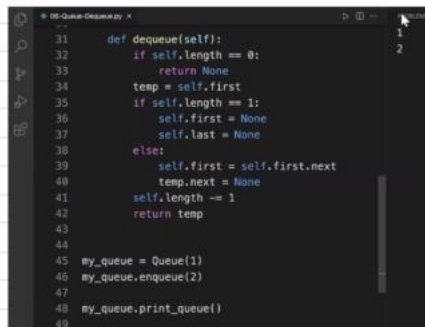


Soal ada 3:

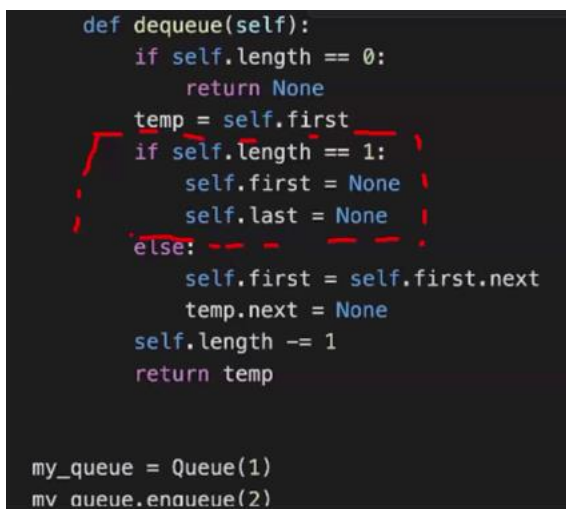
1. Diketahui kode program

Tuliskan hasil output yang muncul jika kode program diatas dijalankan



```
31 def dequeue(self):
32     if self.length == 0:
33         return None
34     temp = self.first
35     if self.length == 1:
36         self.first = None
37         self.last = None
38     else:
39         self.first = self.first.next
40         temp.next = None
41     self.length -= 1
42     return temp
43
44
45 my_queue = Queue(1)
46 my_queue.enqueue(2)
47
48 my_queue.print_queue()
49
```

2. Jelaskan maksud potongan kode program dalam garis putus-putus yang ada
Contoh :



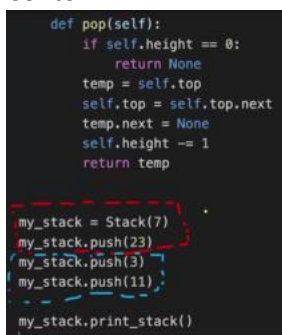
```
def dequeue(self):
    if self.length == 0:
        return None
    temp = self.first
    if self.length == 1:
        self.first = None
        self.last = None
    else:
        self.first = self.first.next
        temp.next = None
    self.length -= 1
    return temp

my_queue = Queue(1)
my_queue.enqueue(2)
```

Kode program yang di kelilingi oleh garis putus-putus digunakan untuk meng handle if condition atau untuk menangani kasus khusus pada konteks dequeue. Kasus khusus yang dimaksud adalah jika queue hanya berisi 1 node. Kasus khusus ini akan membuat kondisi tidak valid jika tidak ditangani dengan baik. Untuk itu penanganannya jika queuenya hanya berisi 1 node dan itu akan di dequeue (dihapus) maka pointer first dan last langsung diarahkan ke kondisi none, sehingga queue tetap dalam kondisi yang valid.

3. Diketahui kode program untuk skenario percobaan adalah sebagai berikut.

Contoh :

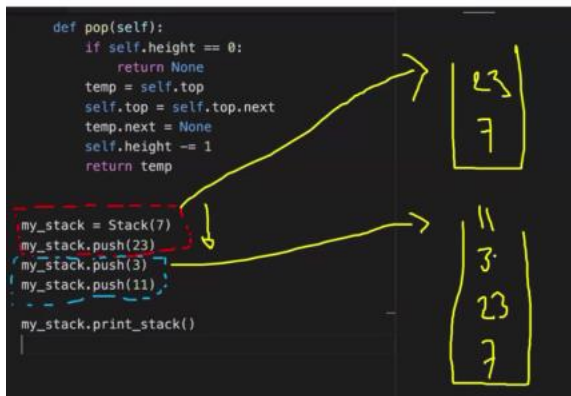


```
def pop(self):
    if self.height == 0:
        return None
    temp = self.top
    self.top = self.top.next
    temp.next = None
    self.height -= 1
    return temp

my_stack = Stack(7)
my_stack.push(23)
my_stack.push(3)
my_stack.push(11)

my_stack.print_stack()
```

Gambarkan perubahan yang terjadi setiap satu blok kode program (blok kode program ditandai dengan garis putus-putus) diatas selesai dieksekusi!



Contoh penjelasan potongan program :

```
class LinkedList:
```

```
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1
```

jelaskan potongan kode program diatas

Potongan kode program di atas adalah sebuah definisi kelas Python bernama LinkedList yang merepresentasikan struktur data linked list. Kelas ini memiliki satu metode konstruktor `__init__` yang dijalankan saat objek baru dari kelas LinkedList dibuat.

Dalam metode konstruktor ini, dilakukan tiga tindakan utama:

- Membuat sebuah objek baru dari kelas Node dengan nilai value yang diberikan.
- Mengatur nilai head dan tail menjadi objek `new_node`. Karena `new_node` adalah satu-satunya node dalam linked list yang baru dibuat, maka head dan tail menunjuk ke node ini.
- Mengatur nilai length menjadi 1, karena saat ini hanya ada satu node dalam linked list.

Dengan demikian, potongan kode program ini membuat sebuah linked list dengan satu node saja yang memiliki nilai value sesuai dengan parameter yang diberikan saat objek LinkedList dibuat.

```
new_node = Node(value)
    self.head = new_node
    self.tail = new_node
    self.length = 1
```

jelaskan potongan kode program diatas

Pertama, baris pertama `new_node = Node(value)` membuat sebuah objek baru dari kelas Node dengan nilai value yang diteruskan sebagai parameter saat membuat objek LinkedList. Dengan demikian, `new_node` merupakan representasi dari node pertama dalam linked list.

Baris kedua dan ketiga `self.head = new_node` dan `self.tail = new_node` mengatur node pertama `new_node` sebagai head dan tail dari linked list. Dalam linked list, head merujuk pada node pertama, sementara tail merujuk pada node terakhir. Karena saat ini linked list hanya memiliki satu node, maka head dan tail mengacu pada `new_node`.

Baris keempat `self.length = 1` mengatur panjang linked list menjadi 1, karena saat ini linked list hanya memiliki satu node.

Secara keseluruhan, potongan kode ini membuat linked list dengan satu node, yaitu `new_node`, sebagai node pertama, dan mengatur `head`, `tail`, dan `length` untuk merepresentasikan linked list ini.

```
new_node = Node(value)
    if self.length == 0:
        self.head = new_node
        self.tail = new_node
    else:
        self.tail.next = new_node
        self.tail = new_node
    self.length += 1
    return True
```

jelaskan potongan kode program diatas

Potongan kode program di atas adalah sebuah metode pada kelas `LinkedList` yang bertanggung jawab untuk menambahkan sebuah node baru ke akhir linked list.

Pertama, pada baris pertama `new_node = Node(value)` dibuat sebuah objek `new_node` dari kelas `Node` dengan nilai `value` yang diteruskan sebagai parameter.

Selanjutnya, pada baris ke-2 dan ke-3, kita memeriksa apakah linked list masih kosong. Jika panjang linked list (`self.length`) adalah 0, maka `new_node` dijadikan sebagai `head` dan `tail` dari linked list. Jika tidak, `new_node` dijadikan sebagai node terakhir (`tail`) dengan menghubungkannya pada node sebelumnya menggunakan `self.tail.next = new_node`, dan kemudian memperbarui nilai `tail` dengan `self.tail = new_node`.

Pada baris ke-6, nilai `length` ditambahkan 1 karena telah ditambahkan sebuah node baru ke linked list.

Pada baris terakhir, metode mengembalikan nilai `True` untuk menunjukkan bahwa penambahan node telah berhasil dilakukan.

Secara keseluruhan, potongan kode program ini menambahkan sebuah node baru ke akhir linked list, mengupdate nilai `head`, `tail`, dan `length` sesuai dengan node baru yang ditambahkan, dan mengembalikan nilai `True` sebagai tanda bahwa penambahan node berhasil dilakukan.

```
if index < 0 or index > self.length:
    return False
if index == 0:
    return self.prepend(value)
if index == self.length:
    return self.append(value)
new_node = Node(value)
temp = self.get(index - 1)
new_node.next = temp.next
temp.next = new_node
self.length += 1
return True
```

jelaskan potongan kode program diatas

Potongan kode program di atas adalah sebuah metode pada kelas `LinkedList` yang bertanggung jawab untuk menyisipkan sebuah node baru pada posisi tertentu dalam linked list.

Pertama, pada baris ke-1, dilakukan pengecekan apakah nilai index kurang dari 0 atau lebih besar dari `self.length`. Jika ya, maka metode akan mengembalikan `False` untuk menunjukkan bahwa sisipan node tidak dapat dilakukan pada indeks yang diberikan.

Selanjutnya, pada baris ke-2 dan ke-3, dilakukan pengecekan apakah nilai index sama dengan 0 atau `self.length`. Jika index sama dengan 0, maka metode memanggil metode `prepend` untuk menyisipkan node pada posisi awal linked list. Jika index sama dengan `self.length`, maka metode memanggil metode `append` untuk menyisipkan node pada posisi akhir linked list.

Jika index tidak sama dengan 0 atau `self.length`, maka pada baris ke-6, sebuah objek `new_node` dibuat dengan nilai `value` sebagai parameter. Kemudian, pada baris ke-7, kita mencari node sebelum posisi yang diinginkan dengan memanggil metode `get` dengan argumen `index-1`.

Pada baris ke-8 dan ke-9, kita menghubungkan node baru `new_node` dengan linked list dengan mengatur `new_node.next` ke `temp.next` dan `temp.next` ke `new_node`. Pada baris ke-10, `length` diupdate dengan menambahkan 1.

Pada baris ke-11, metode mengembalikan nilai `True` sebagai tanda bahwa operasi sisipan telah berhasil dilakukan.

Secara keseluruhan, potongan kode program ini menyisipkan sebuah node baru pada posisi tertentu dalam linked list. Jika index kurang dari 0 atau lebih besar dari `self.length`, atau jika index sama dengan 0 atau `self.length`, maka metode memanggil metode `prepend` atau `append` untuk menyisipkan node pada posisi awal atau akhir linked list. Jika tidak, metode mencari node sebelum posisi yang diinginkan, kemudian menyisipkan node baru di antara node sebelum dan setelah posisi yang diinginkan.

```
def reverse_list(self):
    prev_node = None
    temp_node = self.head
    while temp_node:
        next_node = temp_node.next
        temp_node.next = prev_node
        prev_node = temp_node
        temp_node = next_node
    self.head = prev_node
```

jelaskan potongan kode program diatas

Potongan kode program di atas adalah sebuah metode pada kelas `LinkedList` yang bertanggung jawab untuk membalik urutan node pada linked list.

Pertama, pada baris ke-2, kita membuat variabel `prev_node` dan menginisialisasinya dengan nilai `None`. Variabel ini akan digunakan untuk menyimpan node sebelum node saat ini.

Pada baris ke-3, kita membuat variabel `temp_node` dan menginisialisasinya dengan `self.head`. Variabel ini akan digunakan untuk menyimpan node saat ini.

Pada baris ke-4, kita memulai loop `while` untuk memproses setiap node pada linked list. Loop akan berjalan selama `temp_node` bukan nilai `None`.

Pada baris ke-5, kita membuat variabel `next_node` dan menginisialisasinya dengan node berikutnya setelah `temp_node`.

Pada baris ke-6, kita menghubungkan `temp_node.next` dengan `prev_node` untuk membalikkan urutan node pada linked list.

Pada baris ke-7, kita mengupdate variabel `prev_node` dengan `temp_node`.

Pada baris ke-8, kita mengupdate variabel `temp_node` dengan `next_node` untuk memproses node berikutnya pada loop.

Setelah loop selesai, pada baris ke-9, kita mengupdate `self.head` dengan `prev_node` untuk menyelesaikan pembalikan urutan node pada linked list.

Secara keseluruhan, potongan kode program ini akan membalik urutan node pada linked list dengan menggunakan teknik iterasi dengan bantuan tiga variabel: `prev_node`, `temp_node`, dan `next_node`. Selama loop, setiap node akan dihubungkan ke node sebelumnya untuk membalikkan urutan node. Setelah loop selesai, `self.head` akan diupdate dengan node terakhir pada linked list, yang sebelumnya merupakan `self.tail`.

List adalah struktur data yang memungkinkan penyimpanan dan pengaksesan elemen secara acak. Dalam Python, list didefinisikan dengan tanda kurung siku (`[]`) dan elemen-elemennya dipisahkan oleh tanda koma. List dapat diubah (mutable) dan ukuran list dapat diubah sesuai kebutuhan.

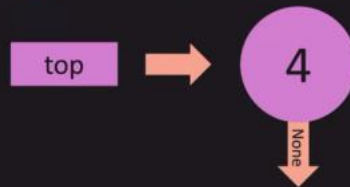
Linked list adalah struktur data yang terdiri dari sejumlah simpul (node) yang saling terhubung. Setiap simpul terdiri dari data dan sebuah pointer yang menunjuk ke simpul berikutnya dalam daftar. Linked list memungkinkan penambahan atau penghapusan elemen di tengah-tengah daftar dengan cepat, namun pengaksesan elemen secara acak relatif lambat. Linked list dapat dibagi menjadi dua jenis: singly linked list dan doubly linked list.

Doubly linked list adalah bentuk khusus dari linked list di mana setiap simpul memiliki dua pointer: satu yang menunjuk ke simpul sebelumnya dan satu lagi yang menunjuk ke simpul berikutnya. Hal ini memungkinkan pengaksesan elemen secara maju atau mundur dengan cepat, namun membutuhkan lebih banyak memori untuk menyimpan pointer tambahan.

- Append (menambahkan node baru di akhir list)
- Prepend (menambahkan node di awal list)
- Insert (menambahkan node baru pada suatu list menggunakan index <index diawal operasi>)
- Pop (menghapus suatu node)
- Pop first (menghapus node pertama atau index ke-0 pada suatu list)
- Set (mengubah suatu node pada list dengan menggunakan index)
- Remove (menghapus suatu node berdasarkan posisi <index>)
- Reserve (membalik urutan elemen dalam linked list)
- Stack/tumpukan <apabila ingin mengambil/menghapus node di urutan paling bawah/akhir indexnya maka yang atas/yang awal harus dikeluarkan terlebih dahulu>

Stack : Constructor

```
class Stack:
    def __init__(self, value):
        new_node = Node(value)
        self.top = new_node
        self.height = 1
```



1. Push (menambah node di stack/memasukkan data)

Stack : Push Code

```
def push(self, value):
    new_node = Node(value)
    if self.height == 0:
        self.top = new_node
    else:
        new_node.next = self.top
        self.top = new_node
    self.height += 1
```

2. Pop (mengambil/menghapus node di stack <pasti node yg paling atas/paling akhir indexnya>)

```
def pop(self):
    if self.height == 0:
        return None
    temp = self.top
    self.top = self.top.next
    temp.next = None
    self.height -= 1
    return temp.value

my_stack = Stack(7)
my_stack.push(23)
my_stack.push(3)
my_stack.push(11)

print(my_stack.pop(), '\n')
my_stack.print_stack()
```

- Queue/antrian (node pertama yang masuk juga akan menjadi node pertama yang keluar)

Queue : Constructor

```
class Queue:
    def __init__(self, value):
        new_node = Node(value)
        self.first = new_node
        self.last = new_node
        self.length = 1
```

```
def print_queue(self):
    temp = self.first
    while temp is not None:
        print(temp.value)
        temp = temp.next
```

```
my_queue = Queue(4)
my_queue.print_queue()
```

1. Enqueue (masuk kedalam antrian)

Queue : Enqueue Skenario Percobaan

```
def enqueue(self, value):  
    new_node = Node(value)  
    if self.first is None:  
        self.first = new_node  
        self.last = new_node  
    else:  
        self.last.next = new_node  
        self.last = new_node  
    self.length += 1  
  
my_queue = Queue(1)  
my_queue.enqueue(2)  
  
my_queue.print_queue()
```

2. Dequeue (keluar dari antrian)

Queue : Dequeue Code

```
def dequeue(self):  
    if self.length == 0:  
        return None  
    temp = self.first  
    if self.length == 1:  
        self.first = None  
        self.last = None  
    else:  
        self.first = self.first.next  
        temp.next = None  
    self.length -= 1  
    return temp
```

```
def reverse_list(self):  
    prev_node = None  
    temp_node = self.head  
    while temp_node:  
        next_node = temp_node.next  
        temp_node.next = prev_node  
        prev_node = temp_node  
        temp_node = next_node  
    self.head = prev_node
```

- Variabel `prev_node` diinisialisasi dengan nilai `None`, yang akan digunakan untuk menyimpan simpul sebelumnya saat melakukan pengulangan.
- Variabel `temp_node` diinisialisasi dengan `self.head`, yaitu simpul pertama dalam linked list.
- Dalam pengulangan `while`, proses ini akan berjalan selama `temp_node` tidak bernilai `None`.
- Variabel `next_node` diisi dengan simpul berikutnya yang akan diakses dalam iterasi berikutnya.
- Pointer `next` dalam `temp_node` diubah sehingga menunjuk ke `prev_node`. Hal ini akan membalikkan arah pointer di setiap simpul dalam linked list.
- Variabel `prev_node` diatur menjadi `temp_node`, yang berarti simpul saat ini menjadi simpul sebelumnya dalam iterasi berikutnya.
- Variabel `temp_node` diatur menjadi `next_node`, yaitu simpul berikutnya yang akan diakses dalam iterasi berikutnya.

- Setelah iterasi selesai, variabel `self.head` diatur menjadi `prev_node`, yaitu simpul terakhir dalam linked list yang merupakan simpul pertama setelah dilakukan pembalikan

`self.value = value`

Penulisan "self" pada program ini merujuk pada objek yang sedang diakses oleh metode tersebut. Dalam bahasa Python, "self" digunakan sebagai referensi atau penunjuk pada objek itu sendiri. Sehingga, potongan program "`self.value = value`" mengatur nilai atribut "value" pada objek yang sedang diakses oleh metode tersebut, dengan nilai yang diberikan oleh parameter "value" pada saat pemanggilan metode.

`self.length += 1`

menambahkan nilai dari atribut "length" pada objek yang sedang diakses oleh metode tersebut, dengan nilai sebesar 1.

`if index < 0 or index > self.length:`

Dengan demikian, jika nilai dari variabel "index" kurang dari 0 atau lebih besar dari nilai atribut "length" pada objek, kondisi if tersebut akan bernilai True. Hal ini menunjukkan bahwa nilai dari variabel "index" berada di luar rentang yang valid untuk akses elemen pada objek. Sebagai hasilnya, metode tersebut tidak akan melakukan operasi yang diinginkan, dan mungkin akan menghasilkan pesan kesalahan atau penanganan kesalahan yang sesuai.

`temp = self.get(index - 1)`

Penulisan "self" pada program ini juga merujuk pada objek yang sedang diakses oleh metode tersebut. Dalam bahasa Python, "self" digunakan sebagai referensi atau penunjuk pada objek itu sendiri. Sehingga, potongan program "`temp = self.get(index - 1)`" mengambil nilai dari elemen pada posisi sebelumnya dari posisi yang diinginkan pada objek yang sedang diakses oleh metode tersebut, dan nilai tersebut disimpan pada variabel "temp".

`Return True`

Pada potongan program "`return True`", metode tersebut mengembalikan nilai True, yang menunjukkan bahwa operasi yang diinginkan pada metode tersebut telah berhasil dilakukan. Nilai True biasanya digunakan untuk menandakan keberhasilan atau kebenaran dari suatu operasi atau kondisi.

Penulisan "return" pada program ini juga digunakan untuk menghentikan eksekusi dari metode tersebut, sehingga kode setelah pernyataan "return" tidak akan dijalankan. Sebagai contoh, jika metode tersebut dipanggil sebagai bagian dari suatu kondisi if, nilai True yang dikembalikan dapat menyebabkan blok if tersebut dijalankan.