**Queue**

* Antrian, pengoperasian berdasarkan aturan FIFO (First in first out). sama modelnya dengan antrian, yang pertama mengantri dialah yang diambil. Kebalikan dari Stack.
* Terdapat satu ujung yang bernama “ekor” (rear) dan satu ujung yang bernama “depan” (front). penambahan data terjadi di ekor dan penghapusan dari depan.

Operasi dalam Queue :

* Queue() = inisialisasi variabel yang mendandakan bahwa Queue
* Enqueue(item) = menambahkan item kedalam sebua Queue
* Dequeue(item) = menghapus item dalam Queue
* isEmpty() = mengecek data
* Size() = mengetahui ukuran suatu Queue

Syntax konsep Queue :

|  |
| --- |
| def createQueue():  q = []  return (q)  def enqueue(q,data):  q.insert(0,data)  return (q)  def dequeue(q):  data = q.pop()  return (data)  def isEmpty(q):  return (q==[])  def size(q):  return (len(q)) |

**Deque**

* Antrian berujung dua, sama denga Queue. Tapi untuk penambahan dan pengapusan data terjadi dimana saja, bisa di rear atau di front.

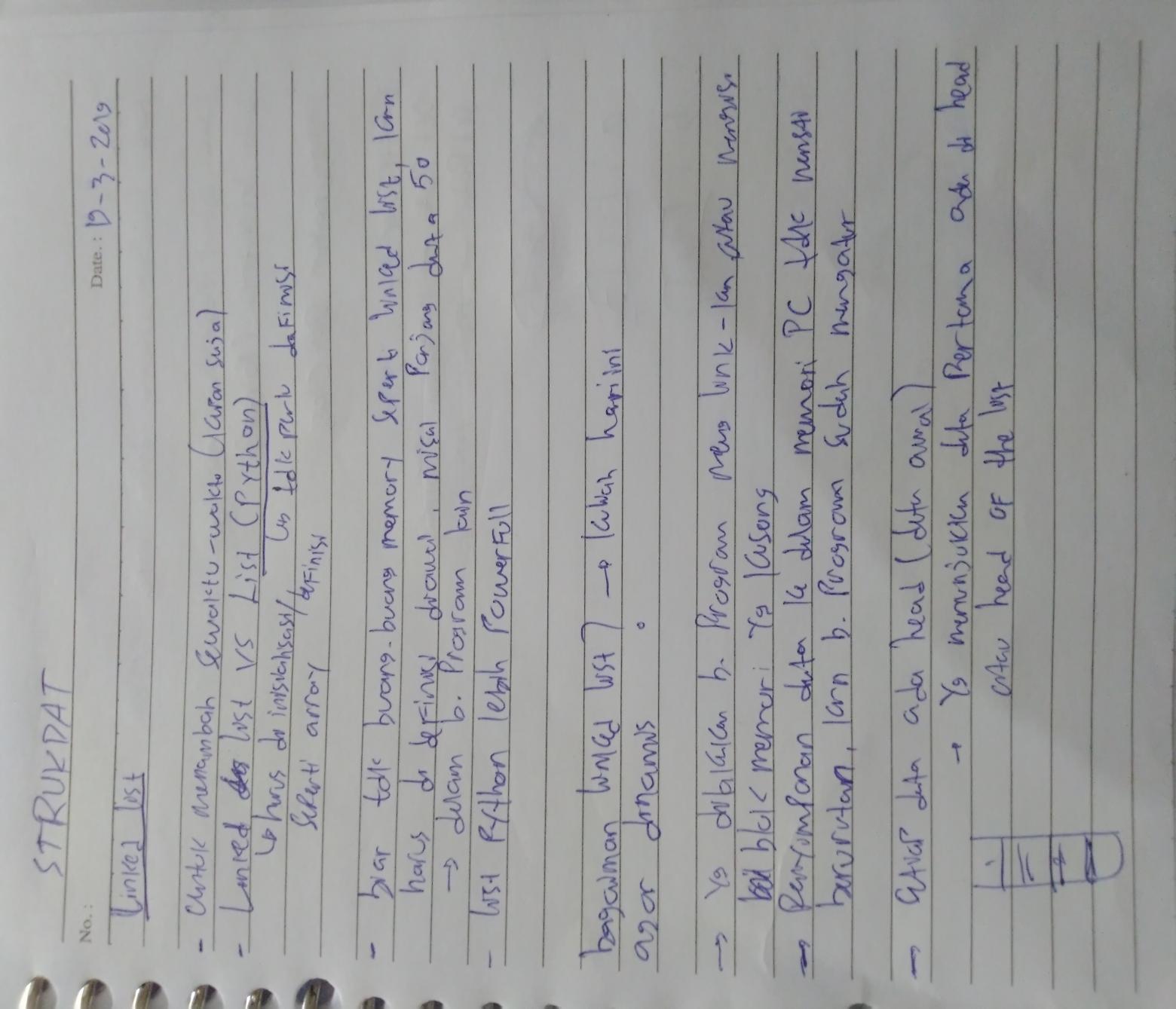
Operasi dalam Queue :

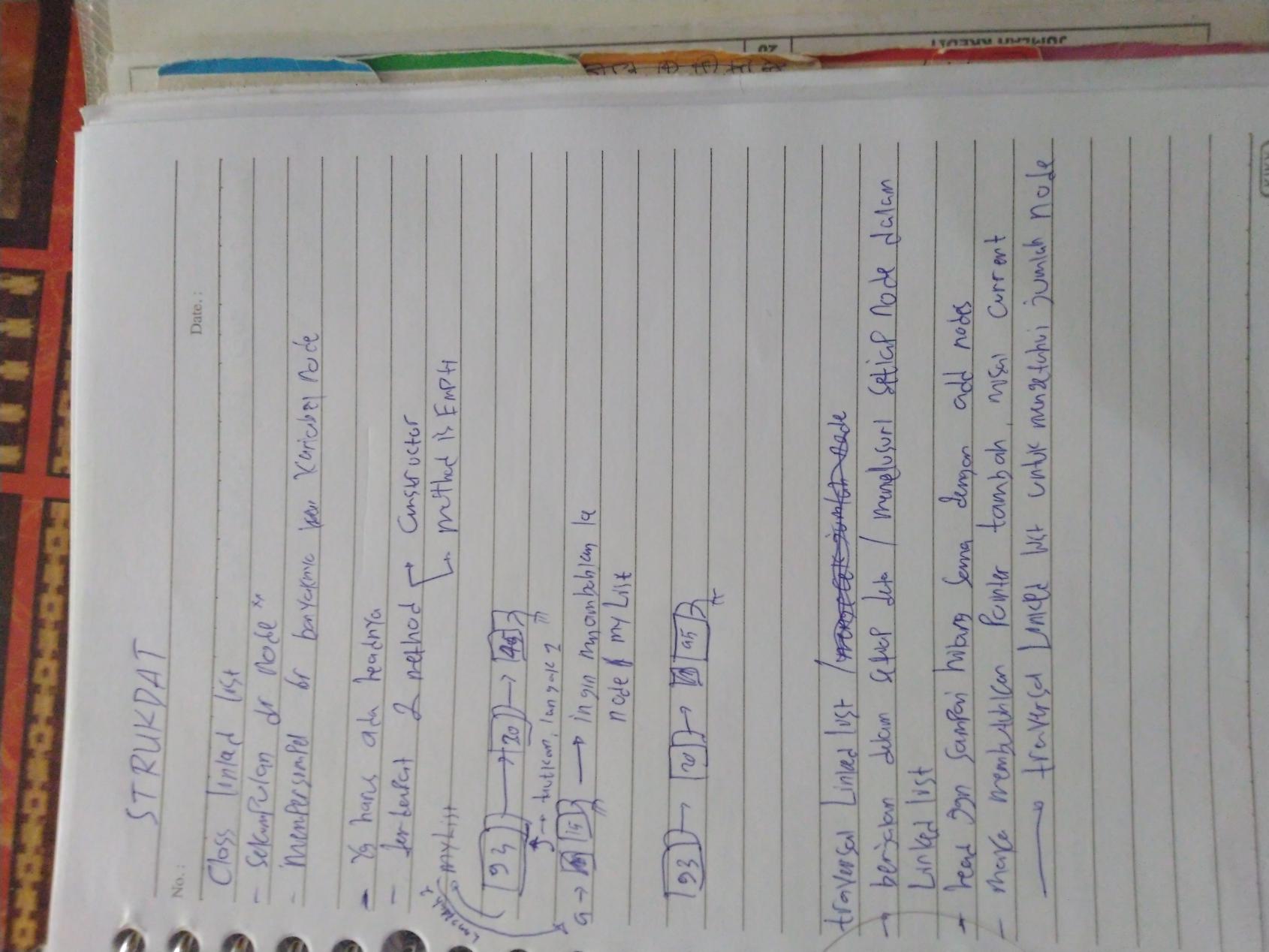
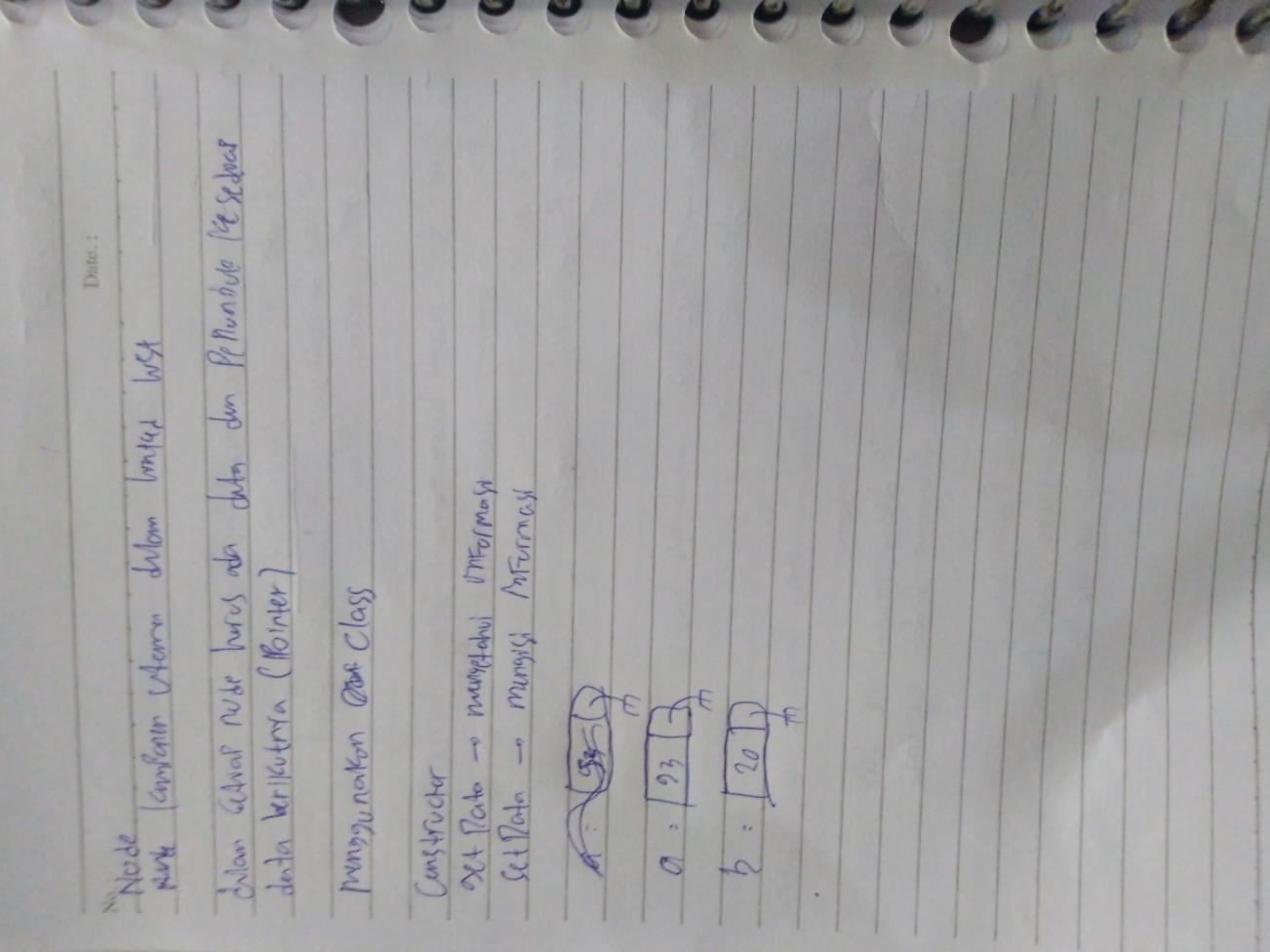
* deque() = inisialisasi variabel yang mendandakan bahwa Queue
* addFront(item) = menambahkan item ke depan
* addRear(item) = menambahkan item ke rear
* removeFront() = menghapus data di front
* removeRear() = menghapus data di rear
* isEmpty() = mengecek data dalam deque
* Size() = mengetahui ukuran suatu deque

Syntax konsep Deque :

|  |
| --- |
| def deque():  d = []  return (d)  def addFront(d,data):  d.append(data)  return (d)  def addRear(d,data):  d.insert(0,data)  return (d)  def removeRear(d):  data = d.pop(0)  return (data)  def removeFront(d):  data = d.pop()  return (data)  def isEmpty(d):  return (d==[])  def size(d):  return (len(d)) |

**Linked List**

****

****