

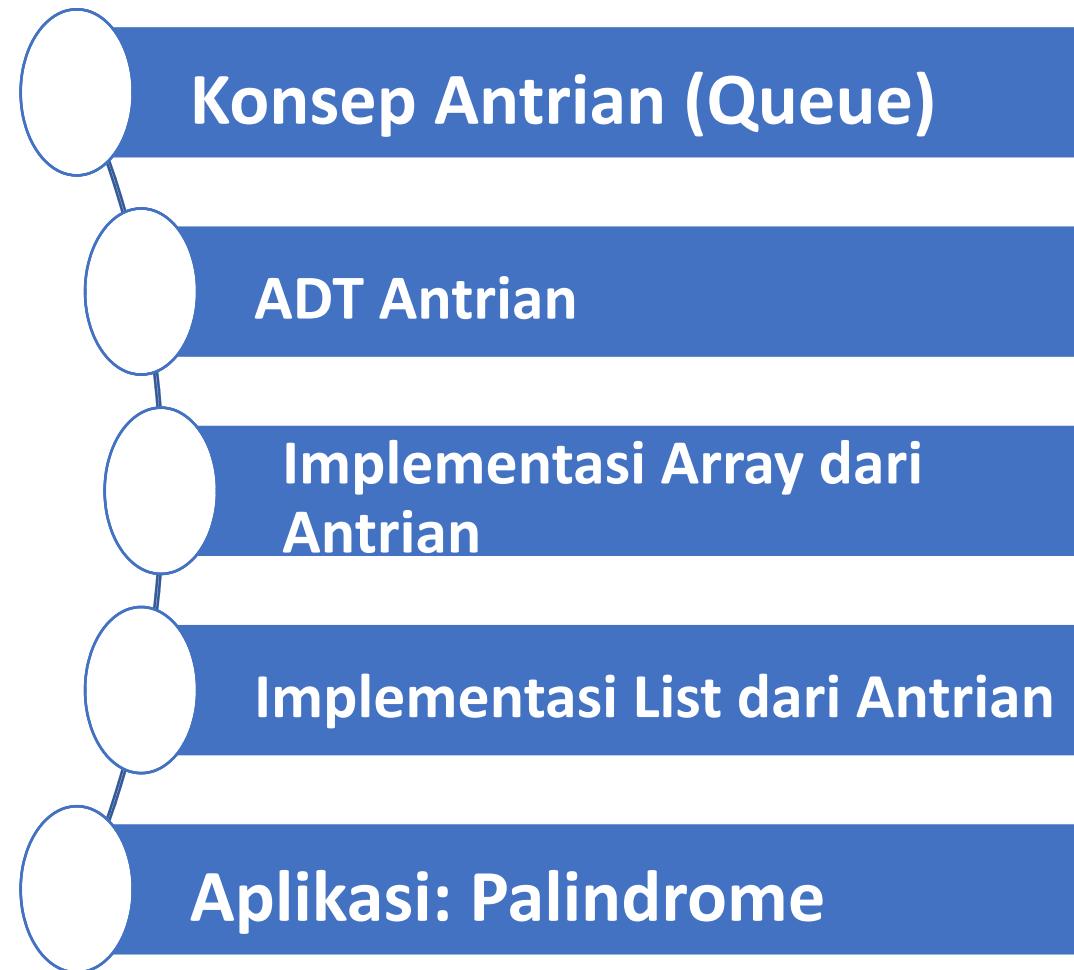


# STRUKTUR DATA (PYTHON)

## “Struktur Antrian (Queue)”

[@SUARGA | [Pertemuan 07]

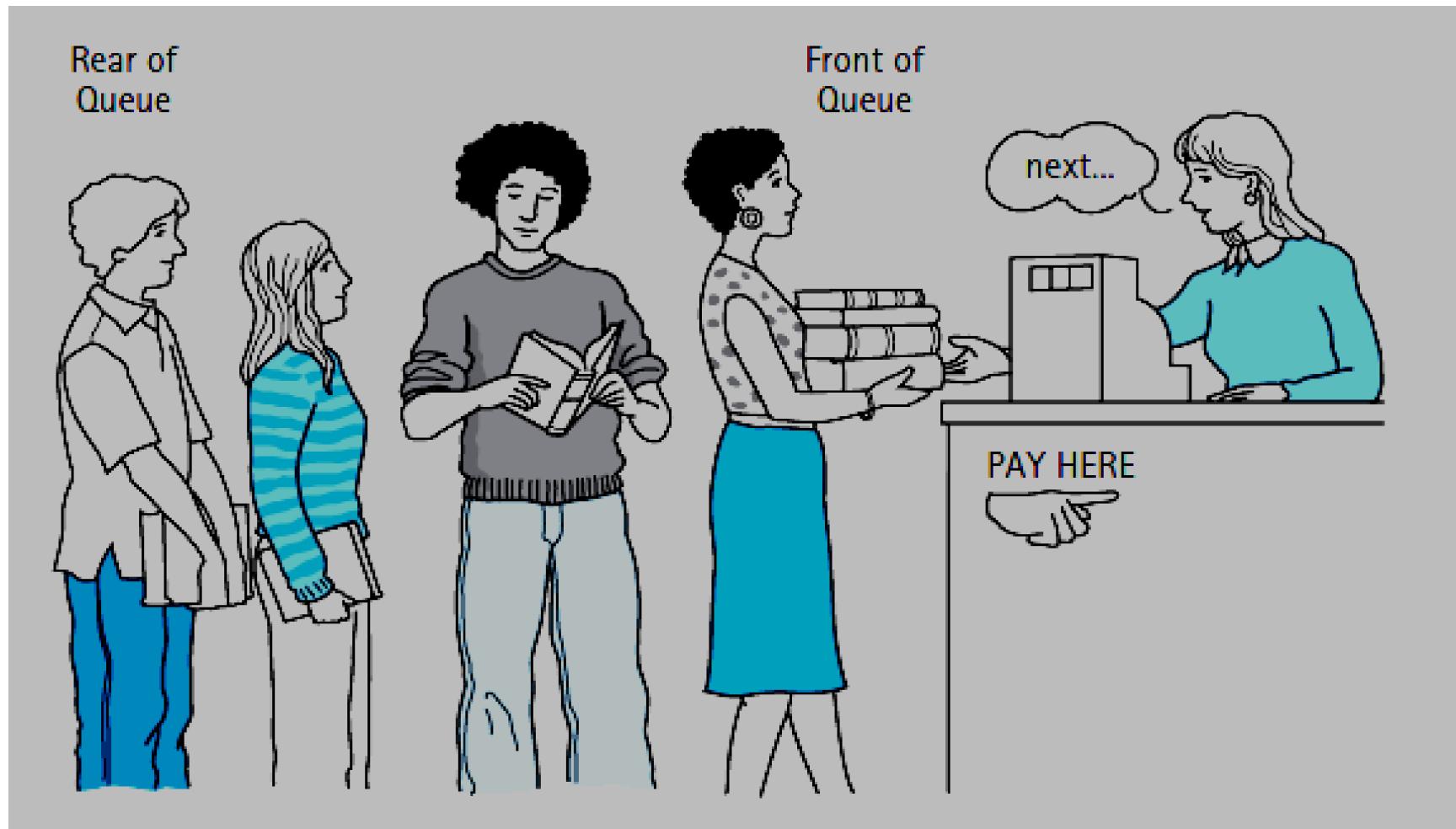
# OutLine



# Konsep Antrian (Queue)



- Struktur Antrian adalah struktur data yang meniru antrian orang yang sedang menunggu pelayanan misalnya didepan seorang teller bank, atau antrian orang yang sedang beli karcis pertunjukkan.
- Apabila diperhatikan dengan saksama maka penambahan orang pada suatu antrian selalu dilakukan pada urutan paling belakang (rear of queue), dan pelayanan selalu dilakukan pada urutan depan (front of queue), sehingga urutan proses antrian sering disebut sebagai FIFO (First In First Out), yang pertama masuk antrian itulah yang pertama dilayani.



# Abstraksi Antrian

- Implementasi Antrian dapat dilakukan dengan membuat tipe data buatan bernama Queue, dengan dua variabel pointer, misalnya sebagai berikut:

Type Queue : record

<

**size** : integer  
    **front**, **rear** : pointer;  
    **isi** : array[1..maks] of item;

>

- Dimana **size** adalah variabel untuk mencacah jumlah elemen dalam antrian, **front** adalah variabel yang menunjuk pada awal (bagian depan) antrian, **rear** adalah variabel yang menunjuk pada bagian akhir antrian, dan **isi** adalah array yang menyimpan isi antrian.

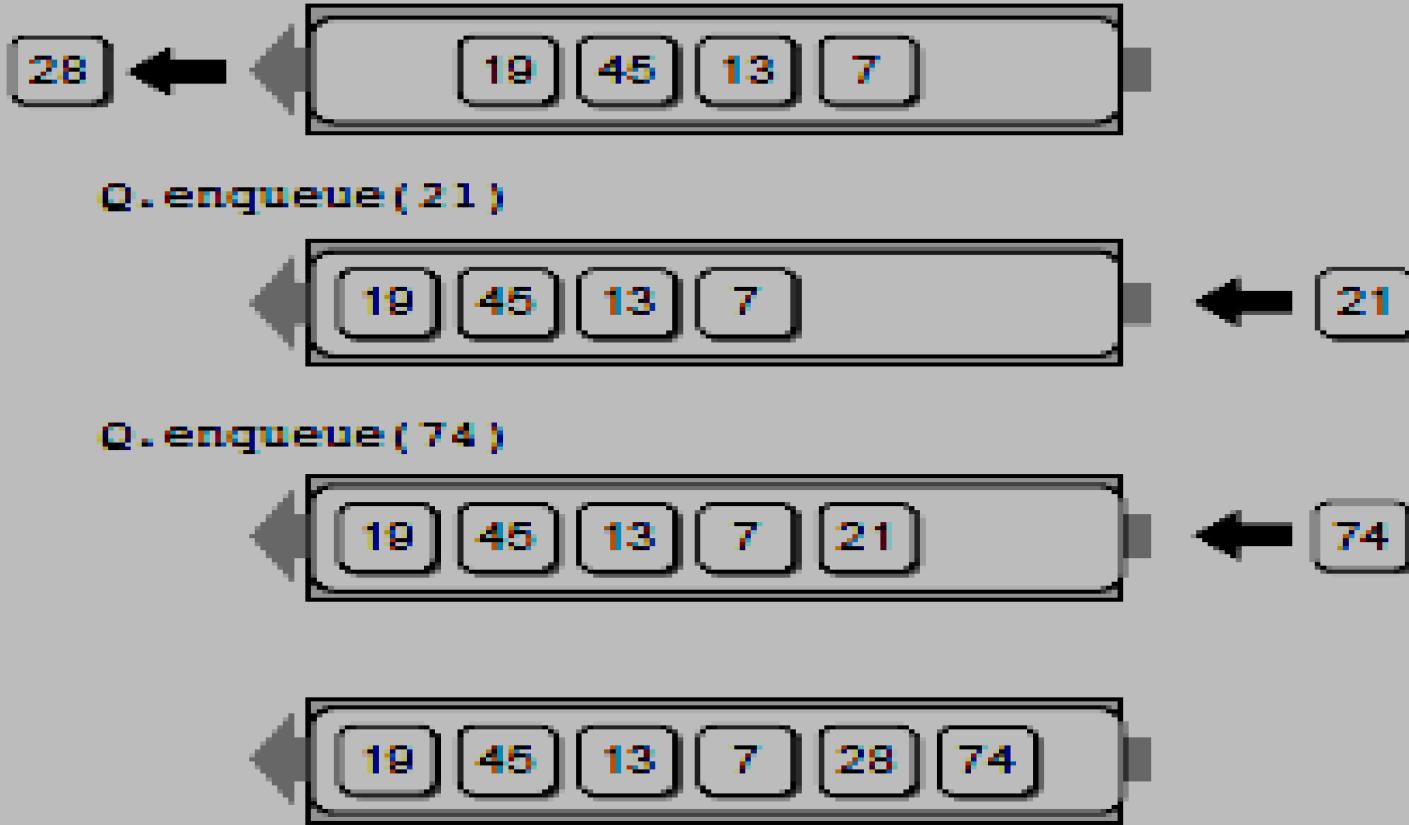


Antrian dengan sisi depan (**front**) dan  
sisi belakang (**back** atau **rear**)

# ADT antrian

- Fungsi ADT yang diperlukan dalam struktur data antrian antara lain adalah:
- **Fungsi untuk memulai antrian (Queue):** menciptakan array untuk antrian, kemudian me-mulai semua indeks, size=0, rear=0, front=1.
- **Fungsi untuk menambahkan elemen (enQueue):** elemen ditambahkan dari belakang, sehingga size bertambah 1, juga rear bertambah 1.
- **Fungsi untuk mengambil elemen (deQueue):** elemen diambil dari antrian selalu dari depan, sehingga size berkurang 1, dan front bertambah 1
- **Fungsi untuk menghitung jumlah elemen (len):** berapa nilai count
- **Fungsi untuk melihat elemen pertama (first)**
- **Fungsi untuk mengubah kapasitas (resize)**
- **Fungsi untuk memeriksa apakah antrian kosong (is\_Empty)**

`x = Q.dequeue()`



## dequeue() dan enqueue() pada antrian

# Algoritma Implementasi Antrian

- Ketika mulai diciptakan (Queue), semua pointer harus di-beri nilai awal, prosedur-nya:

Prosedur Queue(in-out Q:Queue)  
{ memulai suatu antrian }

Definisi variabel:

{ tidak ada }

Rincian Langkah:

```
Q.size <- 0;  
Q.front <- 0;  
Q.rear <- 1;  
return.
```

# Prosedur Enqueue(x)

- Memasukkan objek x ke dalam antrian (enqueue) pada posisi paling belakang (rear) sebagai berikut:

Prosedur enqueue(**input** x:item; **in-out** Q:Queue)

{ menambah elemen kedalam antrian }

Definisi variabel:

{ tidak ada }

Rincian Langkah:

```
if (Q.size = maks) // atau i(Q.isFull())
then write ("tak ada tempat kosong");
else
    Q.size <- Q.size + 1;
    Q.rear <-(Q.rear % maks) + 1;
    Q.isi[Q.rear] <- x;
endif
return.
```

# Prosedur deQueue()

- Proses melayani isi antrian, atau mengambil elemen dari antrian, yaitu pada posisi terdepan (front)

Prosedur deQueue(output x:item; in-out Q:Queue)

{ mengambil satu elemen dari antrian }

Definisi variabel:

{ tidak ada }

Rincian Langkah:

```
if (Q.size = 0) // atau: if (Q.isEmpty())
then write (“antrian kosong ”);
else
    Q.size <- Q.size - 1;
    x <- Q.isi[front];
    front <- (front % maks) + 1;
endif
return;
```

# Fungsi len()

- Menghitung jumlah objek / elemen dalam antrian melalui fungsi len()

Fungsi len(in-out Q:Queue) à integer  
{ menghitung jumlah elemen dalam antrian }

Definisi variabel:

```
int n;
```

Rincian Langkah:

```
n <- Q.size;  
return n;
```

# Implementasi Python Queue, memakai Array

```
#ArrayQueue.py == @Suarga
#=> implementasi Antrian memakai Array
class ArrayQueue:
    #FIFO queue implementation using an array as underlying storage.
    DEFAULT_CAPACITY = 10 # moderate capacity for all new queues

    def __init__(self):
        #Create an empty queue.""""
        self._data = [None]*ArrayQueue.DEFAULT_CAPACITY
        self._size = 0
        self._front = 0
        self._rear = 1

    def __len__(self):
        #Return the number of elements in the queue.""""
        return self._size

    def is_empty(self):
        #Return True if the queue is empty.""""
        return self._size == 0
```

```
def first(self):
    #Return (but do not remove) the element at the front of the queue.
    #Raise Empty exception if the queue is empty.
    if self.is_empty():
        print("Antrian MASIH KOSONG")
        raise Exception('Queue is empty')

    return self._data[self._front]

def dequeue(self):
    #Remove and return the first element of the queue (i.e., FIFO).
    #Raise Empty exception if the queue is empty.
    if self.is_empty():
        print("AWAS dequeue tidak boleh dilakukan!")
        print("Error Antrian sudah KOSONG")
        #raise Exception('Queue is empty')

    x = self._data[self._front]
    self._data[self._front] = None # help garbage collection
    self._front = (self._front + 1) % len(self._data)
    self._size -= 1
    return x
```

```

def enqueue(self, x):
    #Add an element to the back of queue.""""
    if self._size == len(self._data):
        self._resize(2*len(self._data)) # double the array size
    self._rear = (self._front + self._size) % len(self._data)
    self._data[self._rear] = x
    self._size += 1

def _resize(self, cap): # we assume cap >= len(self)
    #Resize to a new list of capacity >= len(self).""""
    old = self._data # keep track of existing list
    self._data = [None]*cap # allocate list with new capacity
    walk = self._front
    for k in range(self._size): # only consider existing elements
        self._data[k] = old[walk] # intentionally shift indices
        walk = (1 + walk) % len(old) # use old size as modulus
    self._front = 0

```

```
#testArrayQueue.py
from ArrayQueue import *
def main():
    Q = ArrayQueue()
    Q.enqueue(5)
    Q.enqueue(3)
    Q.enqueue(4)
    Q.enqueue(2)

    n = len(Q)
    print("Ada : ", n , " data dalam antrian")

    print ("Data pertama : ", Q.first())
    print("melakukan pelayanan:")
    for i in range(n):
        x = Q.dequeue()
        print("Mengambil data : ", x)

    #memeriksa antrian
    if (Q.is_empty()):
        print("Antrian sudah kosong: ")
    else:
        print("Masih ada data dalam antrian")

    print("mencoba mengambil data!")
    Q.dequeue()

main()
```

# Hasil RUN

```
===== RESTART: D:/USER/Python/testArrayQueue.py ====
Ada : 4 data dalam antrian
Data pertama : 5
melakukan pelayanan:
Mengambil data : 5
Mengambil data : 3
Mengambil data : 4
Mengambil data : 2
Antrian sudah kosong:
mencoba mengambil data!
AWAS dequeue tidak boleh dilakukan!
Error Antrian sudah KOSONG
```

# Implementasi Queue memakai List

```
1 #antrian.py => antrian memakai Python List @SUARGA
2 class antrian:
3     #implementasi antrian memakai Python list.
4
5     def __init__(self):
6         #inisialisasi antrian.
7         self._data = [] # memulai dengan list kosong
8
9     def __len__():
10        #memberikan jumlah elemen dalam antrian.
11        return len(self._data)
12
13    def size():
14        return len(self._data)
15
16    def is_empty():
17        #memeriksa apakah antrian kosong.
18        return len(self._data) == 0
19
20    def enqueue(x):
```

```
20     def enqueue(self, x):
21         #memasukkan x ke dalam antrian.
22         self._data.append(x)
23
24     def dequeue(self):
25         #mengambil elemen terdepan dari tumpukan
26         #Raise error Exception bila kosong.
27
28     if self.is_empty():
29         print("Tumpukan KOSONG !")
30         print("dequeue TIDAK BOLEH !")
31         print("ERROR !!")
32         raise Exception('Tumpukan kosong')
33     else:
34         x = self._data[0]
35         self._data.pop(0)
36     return x
37
38     def front(self):
```

```
38     def front(self):
39         #melihat elemen terdepan
40         if self.is_empty():
41             print("ERROR Tumpukan MASIH KOSONG !")
42             print("Belum ada elemen !!!")
43         else:
44             return self._data[0]
45
46     def rear(self):
47         #melihat elemen paling belakang
48         if self.is_empty():
49             print("ERROR Tumpukan MASIH KOSONG !")
50             print("Belum ada elemen !!!")
51             #raise Exception('Tumpukan kosong')
52         else:
53             return self._data[-1]
54
55     def lihat_isi(self):
56         n = self.size()
57         for i in range(n):
58             print(self._data[i])
59
```

```
#Contoh Pemakaian: antrian_test.py
from antrian import *
def main():
    A = antrian()      #membuat antrian
    print("memasukkan 4 elemen:4  6  8  4")
    A.enqueue(4)
    A.enqueue(6)
    A.enqueue(8)
    A.enqueue(4)
    print("Ada ", len(A), " data dalam antrian")
    print("Melihat isi Antrian:")
    A.lihat_isi()
    print("mengambil elemen terdepan")
    x = A.dequeue()
    print(x)
    print("ambil satu lagi ..")
    print(A.dequeue())
    print("periksa elemen terdepan")
    print(A.front())
    print("periksa elemen belakang")
    print(A.rear())
    print("Apakah A sudah kosong?")
    print(A.is_empty())
main()
```

# Hasil RUN

```
===== RESTART: D:\USER\Python\antrian_test.py ===
memasukkan 4 elemen:4 6 8 4
Ada 4 data dalam antrian
Melihat isi Antrian:
4
6
8
4
mengambil elemen terdepan
4
ambil satu lagi ..
6
periksa elemen terdepan
8
periksa elemen belakang
4
Apakah A sudah kosong?
False
```

# Aplikasi Palindrome

- Kalimat palindrome adalah kalimat yang apabila dibaca dari kiri ke kanan akan sama apabila dibaca terbalik dari kanan ke kiri. Contoh dari kalimat kalimat palindrome adalah sebagai berikut:
- Kalimat yang diucapkan Napoleon Bonaparte ketika dibuang ke pulau Elba,  
**“Able was I ere, I saw Elba.”**
- Pujian ke Teddy Roosevelt untuk pembangunan terusan Panama,  
**“A man, a plan, a canal—Panama!”**
- Kalimat yang mungkin terdengar di sebuah restoran Cina,  
**“Won ton, not now”**
- Salah satu nomer plat mobil di Sulawesi Selatan, **“DD 151 DD”**

- Salah satu teknik untuk menguji apakah suatu kalimat, dalam hal ini adalah suatu string, merupakan palindrome atau bukan adalah dengan mengambil setiap karakter dari string ini mulai dari sisi kiri dan memasukkannya ke dalam Stack dan juga ke dalam Queue.
- Kemudian satu persatu diambil dari Stack dan juga dari Queue, lalu dibandingkan, apabila ada karakter yang tidak sama dari pengambilan ini maka kalimat ini bukan palindrome, apabila semua-nya sama maka kalimat ini palindrome.
- Satu hal yang perlu diperhatikan adalah spasi dan tanda baca diabaikan, huruf besar dan huruf kecil dianggap sama. Sebagai contoh perhatikan isi Stack dan isi Queue, ketika diisi dengan kalimat ketiga “Won ton, not now”.

## **Stack :**

w	o	n	t	o	n	n	o	t	n	o	W
---	---	---	---	---	---	---	---	---	---	---	---

**top**

## **Queue :**

w	o	n	t	o	n	n	o	t	n	o	W
---	---	---	---	---	---	---	---	---	---	---	---

**rear**

**front**

Ketika dilakukan perbandingan maka lakukan pop() terhadap Stack dan deQueue pada Queue, sehingga satu persatu huruf dibandingkan, hasilnya sama, maka kalimat ini Palindrome.

# algoritma palindrome

1. Baca kalimat yang akan diperiksa, `readln(kalimat);`
2. Hitung panjangnya: `m = length(kalimat)`
3. Ulangi untuk `idx = 1 s/d m`
  - 3.1 `x = lower(subs(kalimat, idx, 1));`
  - 3.2 bila `x <> spasi & x <> tanda-baca`  
maka: `push(x, S);`  
`addQueue(x, Q);`
4. `palindrome = true;`
5. Ulangi untuk `idx = 1 s/d m`
  - 5.1 `pop(x1, S);`
  - 5.2 `deleteQueue(x2, Q);`
  - 5.3 bila `(x1 <> x2)`  
maka: `palindrome = false;`
6. Bila (`palindrome`)  
maka Cetak(`kalimat, " adalah kalimat palindrome"`);  
bilatidak maka Cetak(`kalimat, " bukan kalimat  
palindrome"`);

# coding: palin.py

```
#palin.py => aplikasi palindrome
from ArrayQueue import ArrayQueue
from ArrayStack import ArrayStack
from Empty import Empty
import string

S=ArrayStack()          #membuat stack
Q=ArrayQueue()          #membuat queue

10 kalimat = input('Masukkan sebuah kalimat : ')
kalimat=kalimat.lower() #jadikan kalimat ini huruf kecil semua
m = len(kalimat)

for c in kalimat:      #ambil setiap huruf dari kalimat
    if (c.isalpha() or c.isdigit()):
        S.push(c)        #masukkan ke stack
        Q.enqueue(c)      #masukkan ke queue

palin = True            #anggaplah palindrome
```

```
palin = True                                #anggaplah palindrome
for i in range(m):
    if (not S.is_empty()):
        x1 = S.pop()                         # x1 ambil dari stack
        x2 = Q.dequeue()                     # x2 ambil dari queue
        if (x1 != x2):
            palin = False                   # bila x1 tidak = x2
            break                          # kalimat bukan palindrome

if (palin):
    print(kalimat, ': palindrome')
else:
    print(kalimat, ': bukan palindrome')

}
```

# Contoh RUN

```
Python Interpreter
*** Remote Interpreter Reinitialized ***
Masukkan sebuah kalimat : palindrome opo ora
palindrome opo ora : bukan palindrome
>>>
*** Remote Interpreter Reinitialized ***
Masukkan sebuah kalimat : Able was I ere, I saw Elba
able was i ere, i saw elba : palindrome
>>>
*** Remote Interpreter Reinitialized ***
Masukkan sebuah kalimat : Won ton, not now
won ton, not now : palindrome
>>> |
```

# Aplikasi: Hot Potato Game

- Sebuah permainan yang sering dimainkan apabila beberapa teman berkumpul, game ini disebut “Hot Potato”.
- Semua orang kumpul membentuk sebuah lingkaran, dan sebuah tongkat atau bendera dipegang oleh orang pertama, kemudian sebuah lagu dimainkan, tongkat atau bendera dioper ke orang yang ada disebelahnya, demikian seterusnya. Kemudian lagu tersebut tiba-tiba di-stop, siapa yang memegang tongkat pada saat itu, akan dikeluarkan dari lingkaran permainan (dialah hot potato).
- Kemudian lagu diputar lagi, dan permainan dilanjutkan. Demikian game ini berlangsung sampai sisa satu orang dalam permainan itu.
- Permainan ini dapat di simulasi menggunakan antrian, implementasinya sbb:

```
11 #HotPotato.py
from antrian import *

def hot_potato(name_list, num):
    sim_queue = antrian()
    for nama in name_list:
        print("%8s masuk lingkaran" % nama)
        sim_queue.enqueue(nama)

    print("\nPermainan Hot Potato di mulai")

    while sim_queue.size() > 1:
        out = sim_queue.dequeue()
        print("%8s keluar dari lingkaran" % out)

    print("Pemain terakhir adalah : %5s " % sim_queue.dequeue())

30 def main():
    print(hot_potato(["Bill","David","Susan","Jane","Kent", "Brad", "Edo"], 7))

if __name__ == '__main__':
    main()
```

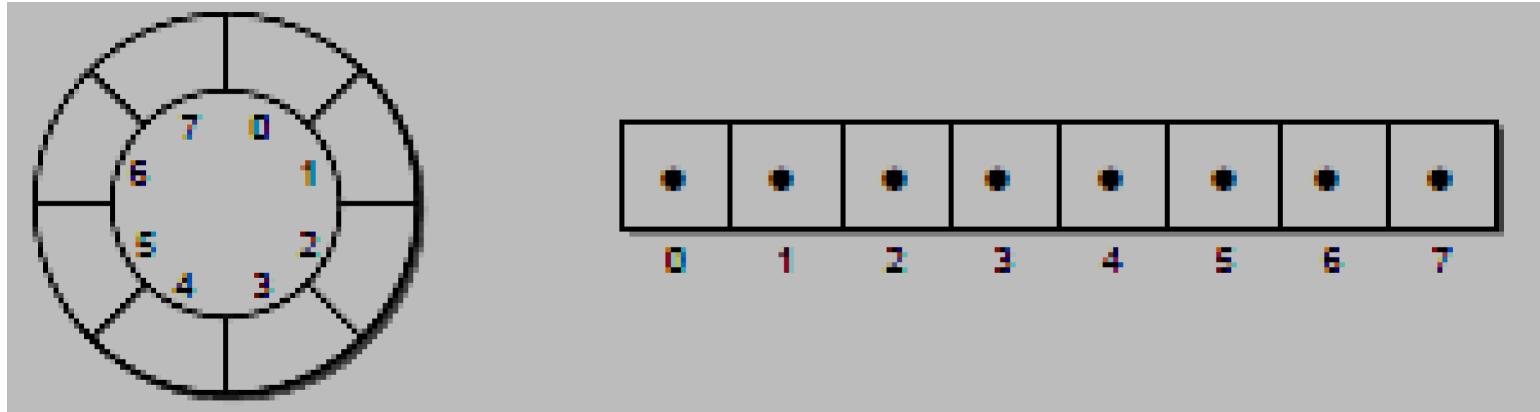
# Hasil RUN

```
>>>
*** Remote Interpreter Reinitialized ***
Bill masuk lingkaran
David masuk lingkaran
Susan masuk lingkaran
Jane masuk lingkaran
Kent masuk lingkaran
Brad masuk lingkaran
Edo masuk lingkaran

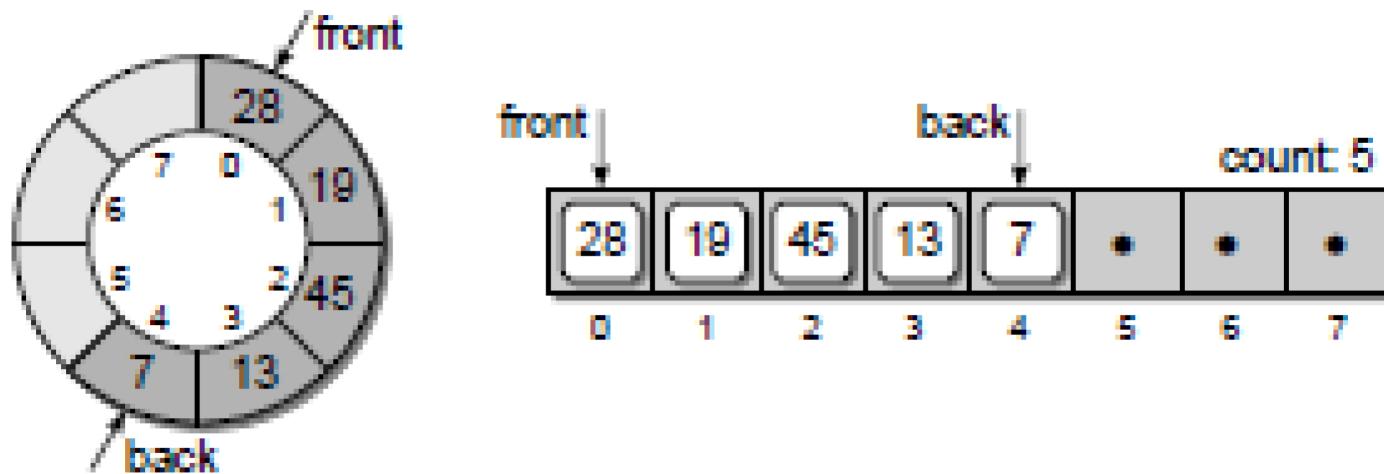
Permainan Hot Potato di mulai
Bill keluar dari lingkaran
David keluar dari lingkaran
Susan keluar dari lingkaran
Jane keluar dari lingkaran
Kent keluar dari lingkaran
Brad keluar dari lingkaran
Pemain terakhir adalah : Edo
None
>>> |
```

# Circular Array untuk Antrian

- Salah satu implementasi yang unik dari antrian adalah implementasi yang memanfaatkan larik lingkaran (circular array).
- Pada implementasi ini suatu larik  $Q[0:n-1]$ , diberi indeks 0 hingga  $(n-1)$ , atau terdiri atas  $n$  buah lokasi. Ketika pointer *rear* adalah  $(n-1)$  maka berarti lokasi berikutnya untuk diisi adalah  $Q[0]$ , bila lokasi ini kosong.

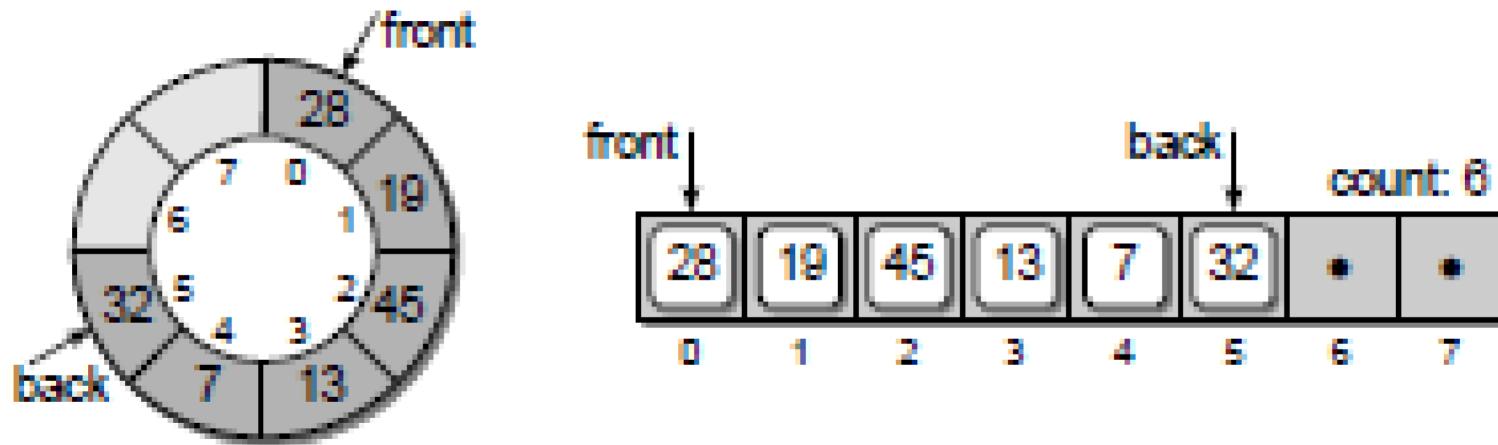


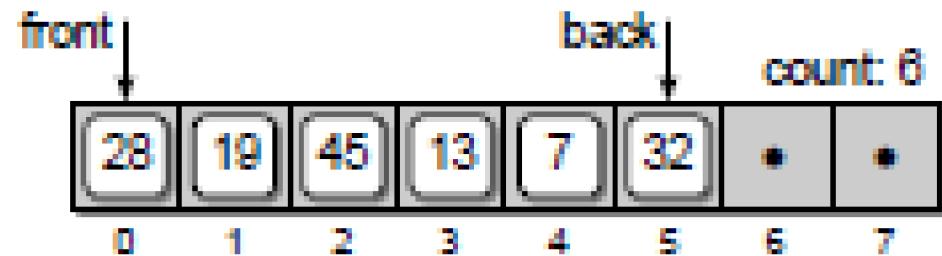
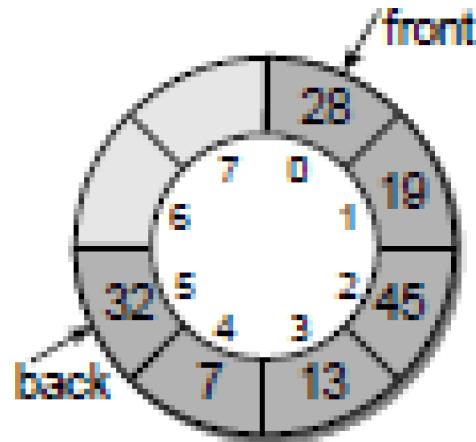
## Model representasi Circular Array untuk Queue



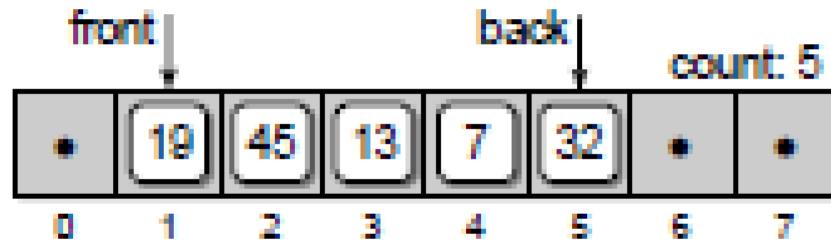
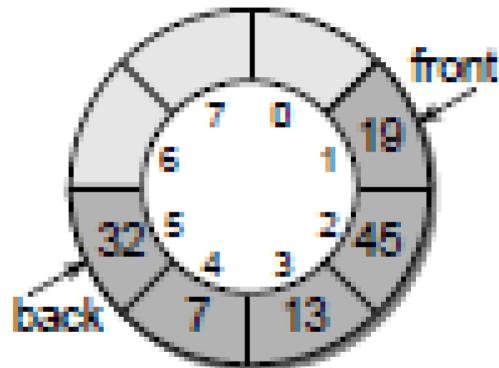
**Contoh Circular Queue, n=5**

Apabila kedalam circular queue ini dimasukkan nilai 32, (addCQ) maka nilai ini ditempatkan dibelakang memakai pointer back (rear). Pertama pointer back menunjuk lokasi no 5, lokasi dibelakang nilai 7, kemudian nilai 32 dimasukkan.





## Nilai 28 diambil (dequeue/deleteCQ) dari antrian



# Prosedur addCQ()

**Prosedur** addCQ(**input** x : item, **in-out** Q:queue)  
{ prosedur untuk menambahkan satu elemen ke dalam  
antrian melingkar }  
**Deklarasi**  
{ tidak ada }

## Deskripsi

```
    Q.rear <- (Q.rear + 1) mod n;  
    if (front = rear)  
    then writeln("Antrian penuh !");  
        Exit;  
    endif;  
    Q(Q.rear) <- x;
```

# Procedur deleteCQ()

Prosedur deleteCQ(out x:item, in-out Q:queue)  
{ prosedur untuk mengambil satu elemen dari antrian }

Deklarasi

{ tidak ada }

Deskripsi

```
if (front = rear)
then writeln("Antrian kosong !");
      Exit;
endif;
Q.front ß (Q.front + 1) mod n;
x ßQ(Q.front);
```

# Implementasi Python: CQueue.py

```
#CQueue.py => circular queue @SUARGA
from Array import Array
class CQueue :
    # Creates an empty queue.
    def __init__( self, maxSize ) :
        self._count = 0
        self._front = 0
        self._rear = maxSize - 1
        self._qArray = Array( maxSize )
    # Returns True if the queue is empty.
    def isEmpty( self ) :
        return self._count == 0
    # Returns True if the queue is full.
    def isFull( self ) :
        return self._count == len(self._qArray)
    # Returns the number of items in the queue.
    def __len__( self ) :
        return self._count
    # Adds the given item to the queue.
```

```
# Adds the given item to the queue.
def addCQ( self, item ):
    assert not self.isFull(), "Cannot enqueue to a full queue."
    maxSize = len(self._qArray)
    self._rear = (self._rear + 1) % maxSize
    self._qArray[self._rear] = item
    self._count += 1

# Removes and returns the first item in the queue.
def deleteCQ( self ):
    assert not self.isEmpty(),"Cannot dequeue from an empty queue."
    item = self._qArray[ self._front ]
    maxSize = len(self._qArray)
    self._front = (self._front + 1) % maxSize
    self._count -= 1
    return item

#mencoba cqueue
```

```
#mencoba cqueue
q=CQueue(6)
print("Memasukkan 6 elemen:")
q.addCQ(1)
q.addCQ(2)
q.addCQ(3)
q.addCQ(4)
q.addCQ(5)
q.addCQ(6)
if q.isFull():
    print('Sudah penuh!')
else:
    print('Masih ada tempat')
print("Menghapus 3 elemen")
print(q.deleteCQ())
print(q.deleteCQ())
print(q.deleteCQ())
print("Apakah CQ kosong?")
if q.isEmpty():
    print('Sudah Kosong ')
else:
    print('Masih ada isi-nya')

print("Masih ada : ")
print(q.__len__())
```

# Hasil RUN

```
Python Interpreter
4
>>>
*** Remote Interpreter Reinitialized ***
Masukkan 6 elemen:
Sudah penuh!
Menghapus 3 elemen
1
2
3
Apakah CQ kosong?
Masih ada isi-nya
Masih ada :
3
>>>
```