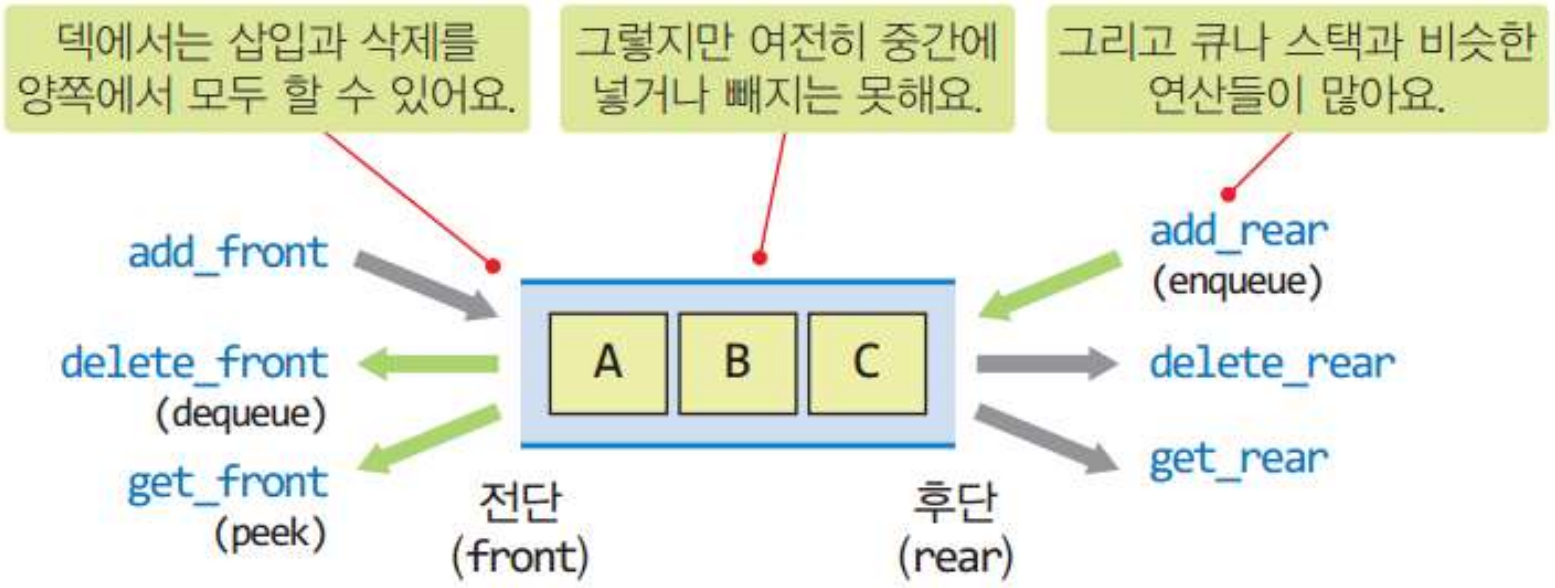
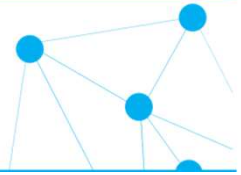
9주차 데크(DEQUE)

덱이란?

- 스택이나 큐보다 입출력이 자유로운 자료구조
- 덱(deque)은 double-ended queue의 줄임말
  - 전단(front)와 후단(rear)에서 모두 삽입과 삭제가 가능한 큐



덱 ADT



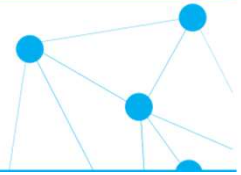
정의 5.2 Deque ADT

데이터: 전단과 후단을 통한 접근을 허용하는 항목들의 모음

연산

- Deque(): 비어 있는 새로운 덱을 만든다.
- isEmpty(): 덱이 비어있으면 True를 아니면 False를 반환한다.
- addFront(x): 항목 x를 덱의 맨 앞에 추가한다.
- deleteFront(): 맨 앞의 항목을 꺼내서 반환한다.
- getFront(): 맨 앞의 항목을 꺼내지 않고 반환한다.
- addRear(x): 항목 x를 덱의 맨 뒤에 추가한다.
- deleteRear(): 맨 뒤의 항목을 꺼내서 반환한다.
- getRear(): 맨 뒤의 항목을 꺼내지 않고 반환한다.
- isFull(): 덱이 가득 차 있으면 True를 아니면 False를 반환한다.
- size(): 덱의 모든 항목들의 개수를 반환한다.
- clear(): 덱을 공백상태로 만든다.

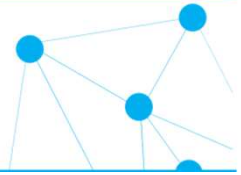
원형 덱의 연산



- 큐와 데이터는 동일함
- 연산은 유사함.

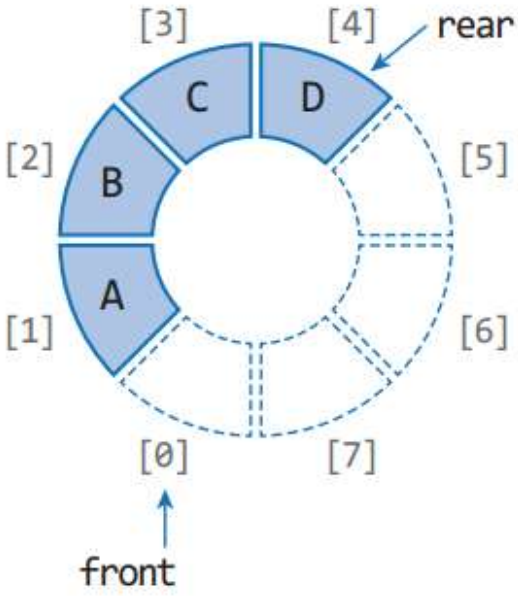
- 큐와 알고리즘이 동일한 연산
  - addRear(), deleteFront(), getFront()
    - 큐의 enqueue, dequeue, peek 연산과 동일
  - 덱의 후단(rear)을 스택의 상단(top)으로 사용하면 addRear(), deleteRear(), getRear() 연산은
    - 스택의 push, pop, peek 연산과 정확히 동일하다.

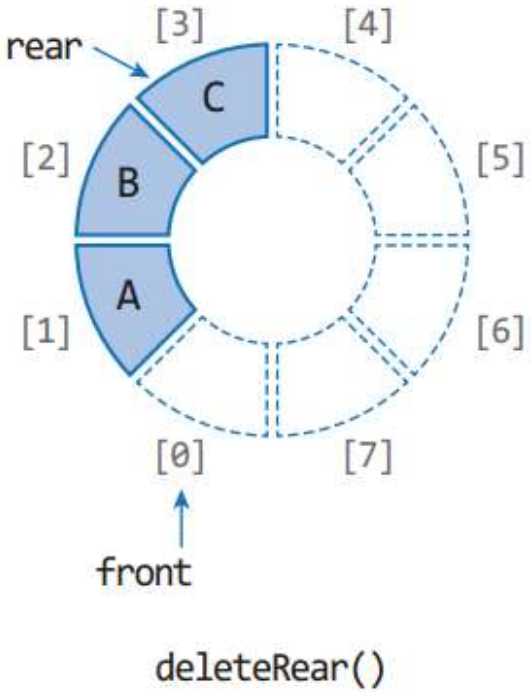
원형 큐에서 추가된 연산

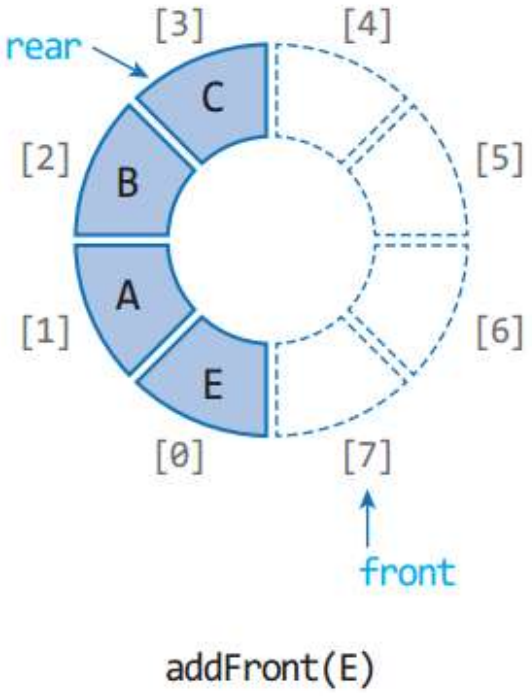


- delete_rear(), add_front(), get_rear()
  - 반 시계방향 회전 필요

```
front ← (front-1 + MAX_QSIZE) % MAX_QSIZE
rear ← (rear –1 + MAX_QSIZE) % MAX_QSIZE
```







- 파이썬에는 데크가 Collections 패키지에 정의되어 있음
- 삽입, 삭제 등의 연산은 파이썬의 리스트의 연산들과 매우 유사

```
01 from collections import deque
02 dq = deque('data')
03 for elem in dq:
04     print(elem.upper(), end='')
05 print()
06 dq.append('r')
07 dq.appendleft('k')
08 print(dq)
09 dq.pop()
10 dq.popleft()
11 print(dq[-1])
12 print('x' in dq)
13 dq.extend('structure')
14 dq.extendleft(reversed('python'))
15 print(dq)
```

새 데크 객체를 생성

맨 뒤와 맨 앞에 항목 삽입

맨 뒤와 맨 앞의 항목 삭제

맨 뒤의 항목 출력

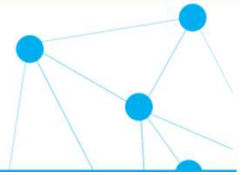
맨 뒤와 맨 앞에 여러 항목 삽입

Console PyUnit

<terminated> deque.py [C:₩Users₩sbyang₩AppData₩Local₩Programs₩Python₩Python36-32₩python.exe]

```
DATA
deque(['k', 'd', 'a', 't', 'a', 'r'])
a
False
deque(['p', 'y', 't', 'h', 'o', 'n', 'd', 'a', 't', 'a', 's', 't', 'r', 'u', 'c', 't', 'u', 'r', 'e'])
```

덱의 구현



- 원형 큐를 상속하여 원형 덱 클래스를 구현

```
class CircularDeque(CircularQueue) :      # CircularQueue에서 상속
```

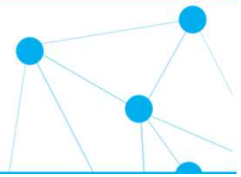
- 덱의 생성자 (상속되지 않음)

```
def __init__( self ) :              # CircularDeque의 생성자
    super().__init__()              # 부모 클래스의 생성자를 호출함
```

  - front, rear, items와 같은 멤버 변수는 추가로 선언하지 않음
  - 자식클래스에서 부모를 부르는 함수가 super()
- 재 사용 멤버들: isEmpty, isFull, size, clear
- 인터페이스 변경 멤버들

```
def addRear( self, item ): self.enqueue(item )     # enqueue 호출
def deleteFront( self ): return self.dequeue()     # 반환에 주의
def getFront( self ): return self.peek()           # 반환에 주의
```

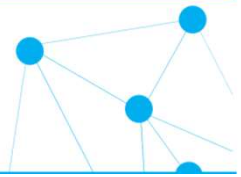
원형 덱의 구현



- 추가로 구현할 메소드

```
def addFront( self, item ):                    # 새로운 기능: 전단 삽입
    if not self.isFull():
        self.items[self.front] = item          # 항목 저장
        self.front = self.front - 1            # 반시계 방향으로 회전
        if self.front < 0 : self.front = MAX_QSIZE - 1

def deleteRear( self ):                        # 새로운 기능: 후단 삭제
    if not self.isEmpty():
        item = self.items[self.rear];          # 항목 복사
        self.rear = self.rear - 1              # 반시계 방향으로 회전
        if self.rear < 0 : self.rear = MAX_QSIZE - 1
        return item                            # 항목 반환

def getRear( self ):                           # 새로운 기능: 후단 peek
    return self.items[self.rear]
```

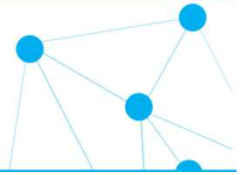
테스트 프로그램



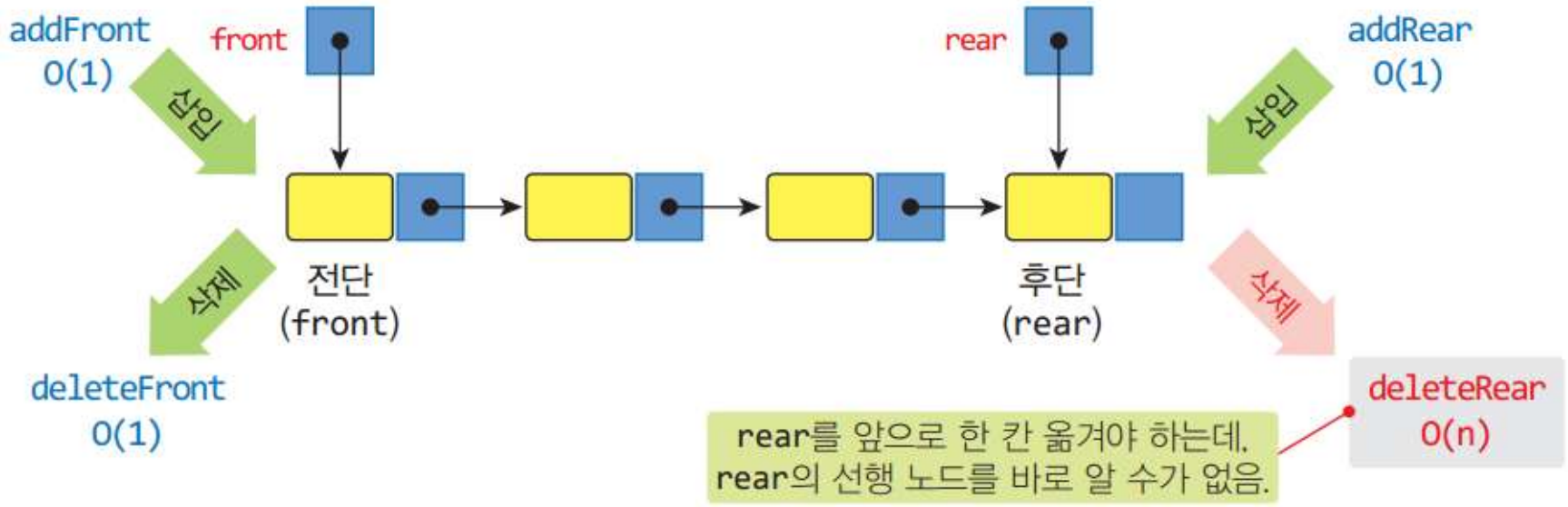
```
dq = CircularDeque()                    # 덱 객체 생성. f=r=0
for i in range(9):                      # i : 0, 1, 2, ... 8
      if i%2==0 : dq.addRear(i)         # 짝수는 후단에 삽입:
      else : dq.addFront(i)             # 홀수는 전단에 삽입
dq.display()                            # front=6,rear=5
for i in range(2): dq.deleteFront()     # 전단에서 두 번의 삭제: f=8
for i in range(3): dq.deleteRear()      # 후단에서 세 번의 삭제: r=2
dq.display()
for i in range(9,14): dq.addFront(i)    # i : 9, 10, ... 13 : f=3
dq.display()
```

```
C:₩WINDOWS₩system32₩cmd.exe
[f=6,r=5] ==>  [7, 5, 3, 1, 0, 2, 4, 6, 8]
[f=8,r=2] ==>  [3, 1, 0, 2]
[f=3,r=2] ==>  [13, 12, 11, 10, 9, 3, 1, 0, 2]
```

이중연결리스트의 응용: 연결된 덱

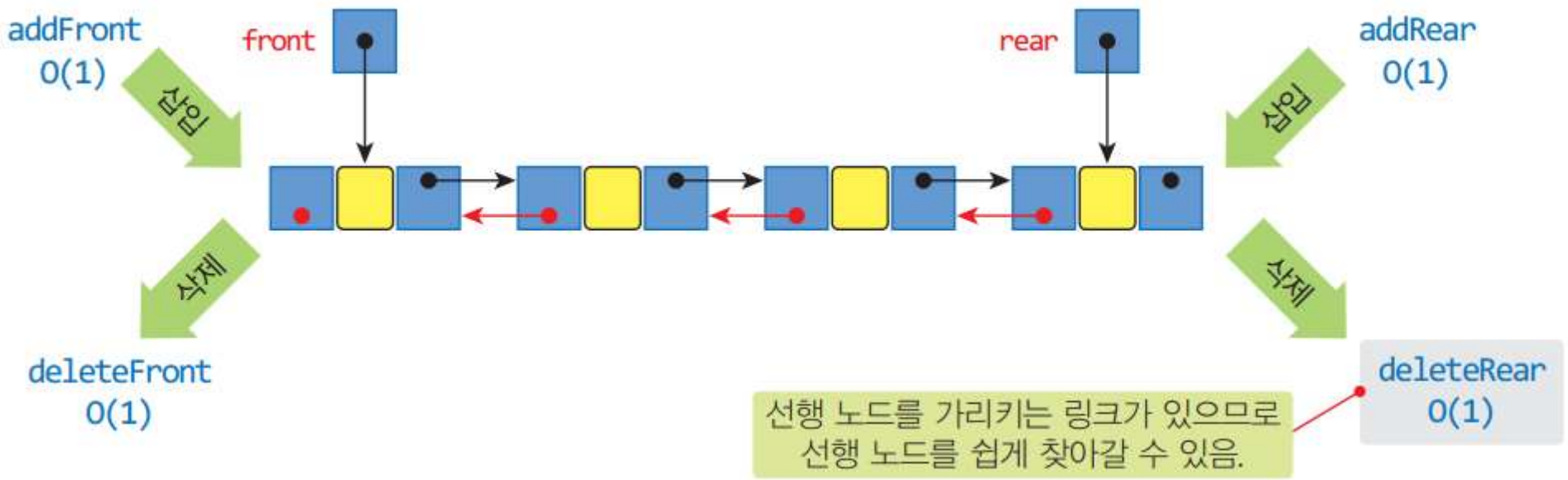


- 단순연결리스트로 구현한 덱



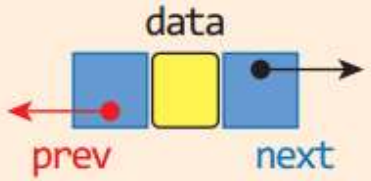
- 해결 방안은?
  - 이중연결리스트 사용

이중연결리스트로 구현한 덱



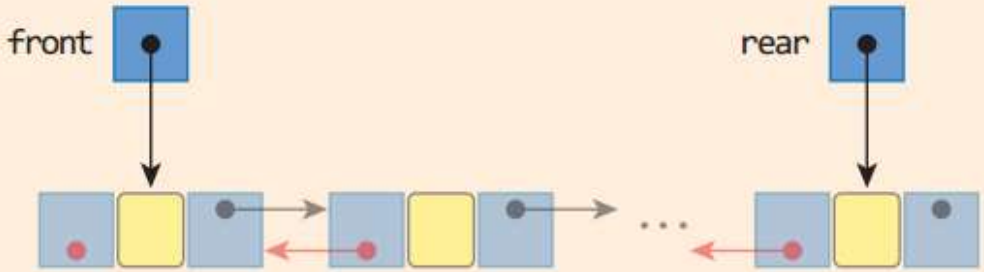
- 이중연결리스트를 위한 노드

```
class DNode:                                    # 이중연결리스트를 위한 노드
    def __init__ (self, elem, prev = None, next = None):
        self.data = elem
        self.prev = prev
        self.next = next
```



연결된 덱 클래스

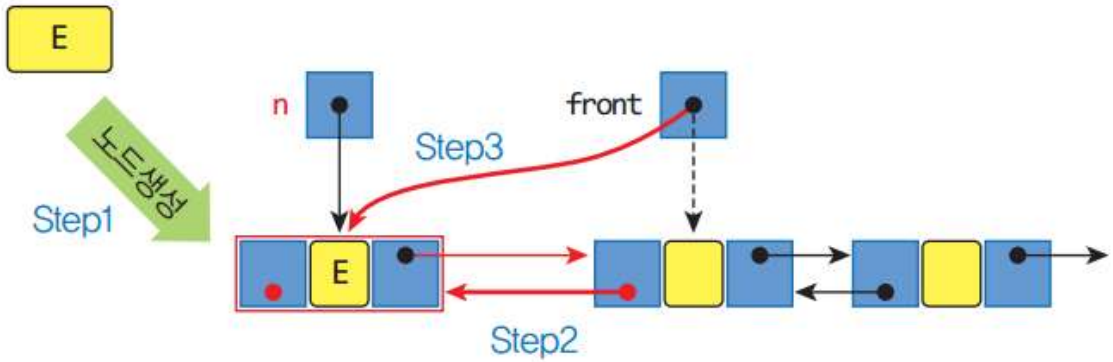
```
class DoublyLinkedDeque:
    def __init__( self ):
        self.front = None
        self.rear = None

    def isEmpty( self ): return self.front == None          # 공백상태 검사
    def clear( self ): self.front = self.front = None       # 초기화
```




```
def size( self ):
        node = self.front
        count = 0
        while not node == None :
            node = node.next
            count += 1
        return count
```

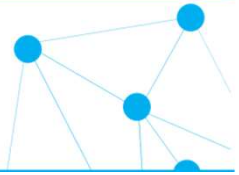
```
def display( self, msg='LinkedStack:'):
        print(msg, end='')
        node = self.front
        while not node == None :
            print(node.data, end=' ')
            node = node.next
        print()
```

addFront(), addRear()

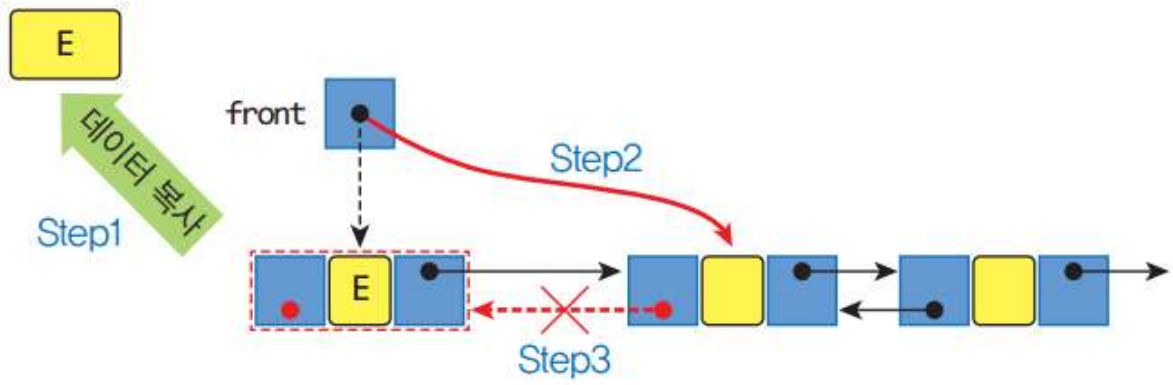



```
def addFront( self, item ):
    node = DNode(item, None, self.front)          # Step1
    if( self.isEmpty()):                          # 공백이면
        self.front = self.rear = node             # fron와 rear가 모두 node
    else :                                        # 공백이 아니면
        self.front.prev = node                    # Step2
        self.front = node                         # Step3
def addRear( self, item ):
    node = DNode(item, self.rear, None)           # Step1
    if( self.isEmpty()):                          # 공백이면
        self.front = self.rear = node             # fron와 rear가 모두 node
    else :                                        # 공백이 아니면
        self.rear.next = node                     # Step2
        self.rear = node                          # Step3
```

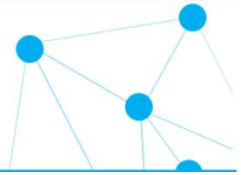
deleteFront(), deleteRear()



```
def deleteFront( self ):
    if not self.isEmpty():
        data = self.front.data
        self.front = self.front.next
        if self.front==None :
            self.rear = None
        else:
            self.front.prev = None          # Step3
        return data                         # Step4

def deleteRear( self ):
    if not self.isEmpty():
        data = self.rear.data               # Step1
        self.rear = self.rear.prev          # Step2
        if self.rear==None :                # 노드가 하나 뿐이면
            self.front = None               # front도 None으로 설정
        else:
            self.rear.next = None           # Step3
        return data                         # Step4
```

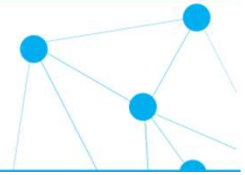


테스트 프로그램



```
dq = DoublyLinkedDeque()              # 연결된 덱 만들기
```

```
for i in range(9):                    # i : 0, 1, 2, ... 8
      if i%2==0 : dq.addRear(i)       # 짝수는 후단에 삽입:
      else : dq.addFront(i)           # 홀수는 전단에 삽입
dq.display()                          # front=6,rear=5
for i in range(2): dq.deleteFront()   # 전단에서 두 번의 삭제: f=8
for i in range(3): dq.deleteRear()    # 후단에서 세 번의 삭제: r=2
dq.display()
for i in range(9,14): dq.addFront(i)  # i : 9, 10, ... 13 : f=3
dq.display()
```



감사합니다