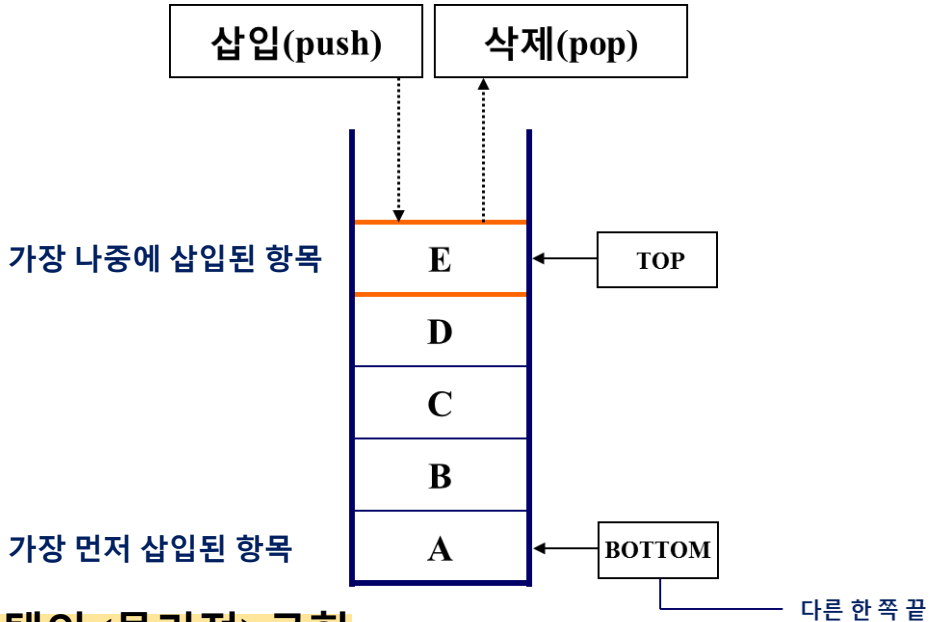
스택과 큐

HaRim Jung, Ph.D.
Research Professor
School of Information and
Communication Engineering,
Sungkyunkwan University, Korea

스택 (1/6)

❑ 스택(Stack)

- 한 쪽 끝 위치(**Top**)에서만 새로운 항목을 삽입(Push)하거나 기존 항목을 삭제 **및 반환**(Pop)하는 논리적 선형 구조
  - **시간 순으로** 먼저 삽입된 항목이 **나중에 삭제**되는, i.e., 후입선출(LIFO: **Last-In, First-Out**) 구조
  - Top에서만 삽입·삭제가 이루어지는(적용 가능한 연산이 제한된) **특수한 리스트(List)**

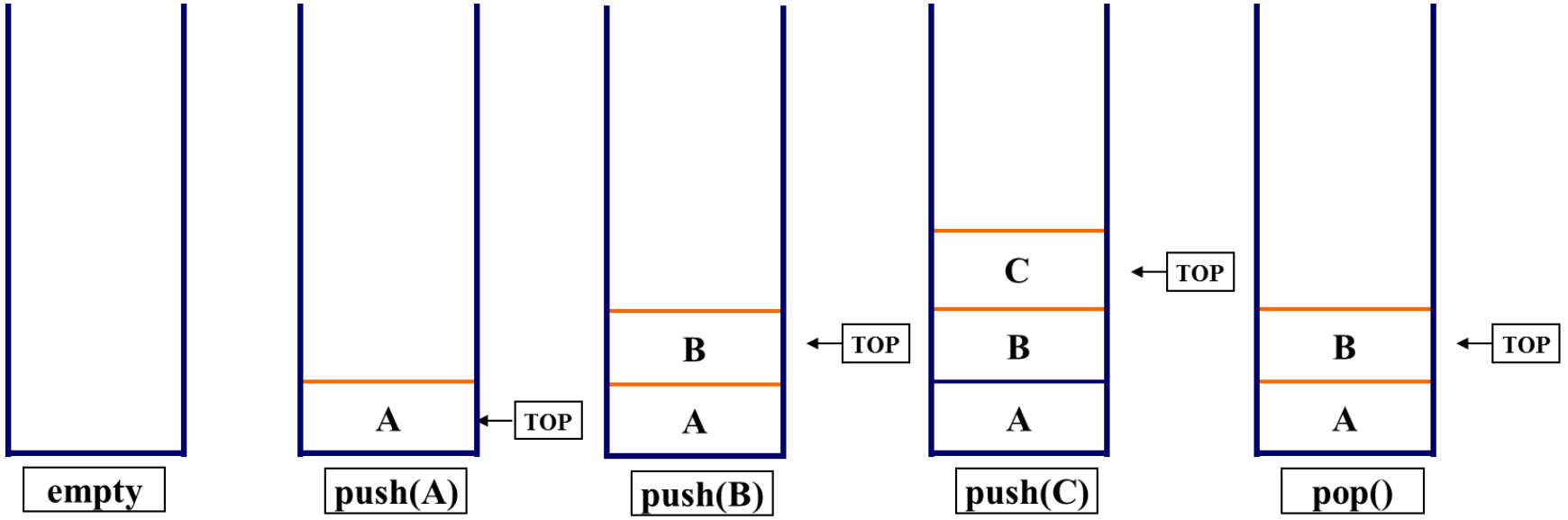


- **스택의 (물리적) 구현**
  - Python 리스트(동적 배열)에 의한 구현 –push·pop 연산의 시간복잡도 **O(1)**
  - 단순 연결 리스트(Singly Linked List)에 의한 구현 – push·pop 연산의 시간복잡도 **O(1)**

스택 (2/6)

❑ 스택에 적용 가능한 주요 연산

- **Stack():** 빈 스택 생성
- **push(item):** 기존 Top 다음 위치에 item 삽입
- **pop():** Top 위치에 존재하는 item **삭제 및 반환**
- **peek():** Top 위치에 존재하는 item **반환**
- **is_empty():** 스택이 empty이면 True 반환
- **size():** 스택의 사이즈 반환

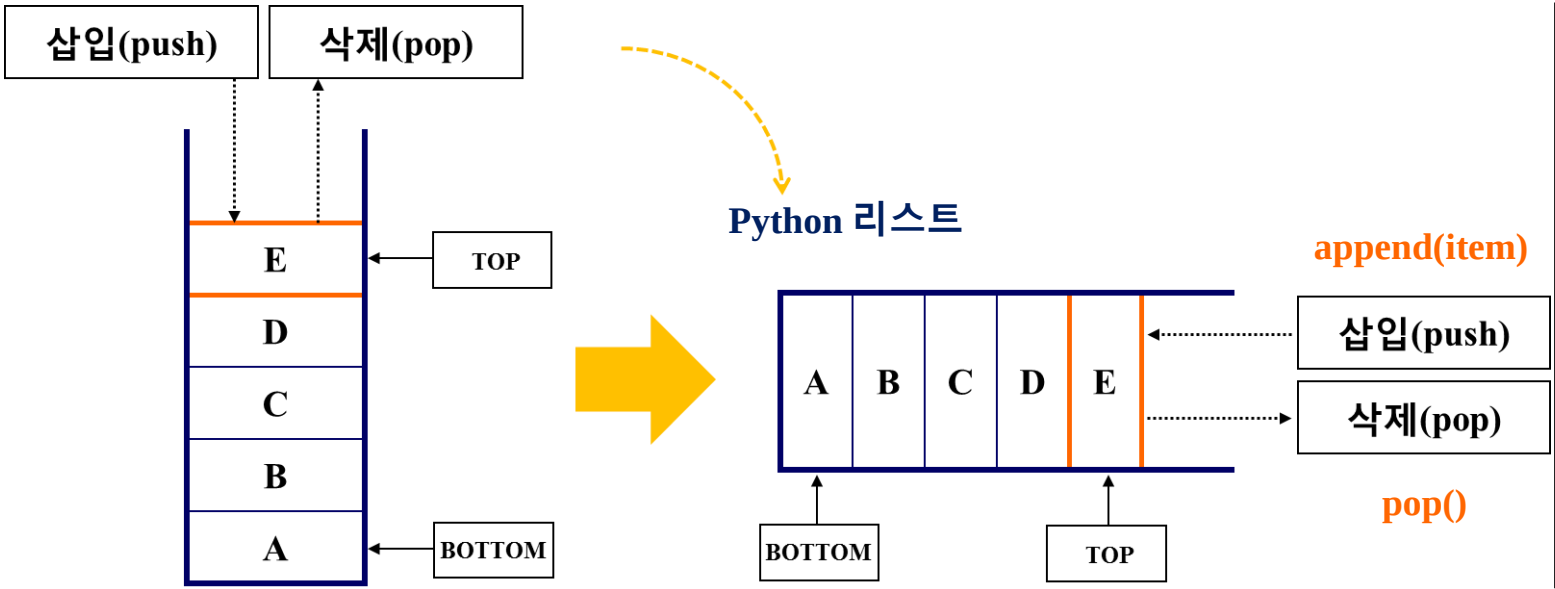


스택의 push와 pop 연산

스택 (3/6)

❑ Python 리스트로 구현한 스택

- append(item)와 pop() 메소드로 push(item)와 pop() 구현 가능
  - 하지만 Python 리스트를 그대로 사용하되 한쪽 끝에서만 item을 삽입·삭제 하기로 약속하는 것보다 item 삽입·삭제 위치를 스택 Top으로 제한하는 것이 더 바람직하므로 Stack 클래스를 정의




```
In [23]: stack = []
    ...: stack.append(1)
    ...: print(stack)
    ...: stack.append(2)
    ...: print(stack)
    ...: stack.append(3)
    ...: print(stack)
    ...: stack.pop()
    ...: print(stack)
    ...: stack.pop()
    ...: print(stack)
[1]
[1, 2]
[1, 2, 3]
[1, 2]
[1]
```

스택 (4/6)

❑ Python 리스트로 구현한 스택 contd.

- 스택 객체를 위한 클래스 정의

```
1  class Stack:
2      def __init__(self):
3          self.items = []
4
5      def is_empty(self):
6          return self.items == []
7
8      def push(self, item):
9          self.items.append(item)
10
11     def pop(self):
12         if self.is_empty():
13             return None
14         else:
15             return self.items.pop()
16
17     def peek(self):
18         if self.is_empty():
19             return None
20         else:
21             return self.items[len(self.items)-1]
22
23     def size(self):
24         return len(self.items)
```

Stack 클래스

```
20 if __name__ == "__main__":
21
22     s = Stack()
23     print(s.is_empty())
24     s.push(4)
25     s.push('dog')
26     print(s.peek())
27     s.push(True)
28     print(s.size())
29     print(s.is_empty())
30     s.push(8.4)
31     print(s.pop())
32     print(s.pop())
33     print(s.size())
```

일련의 스택 연산과 출력

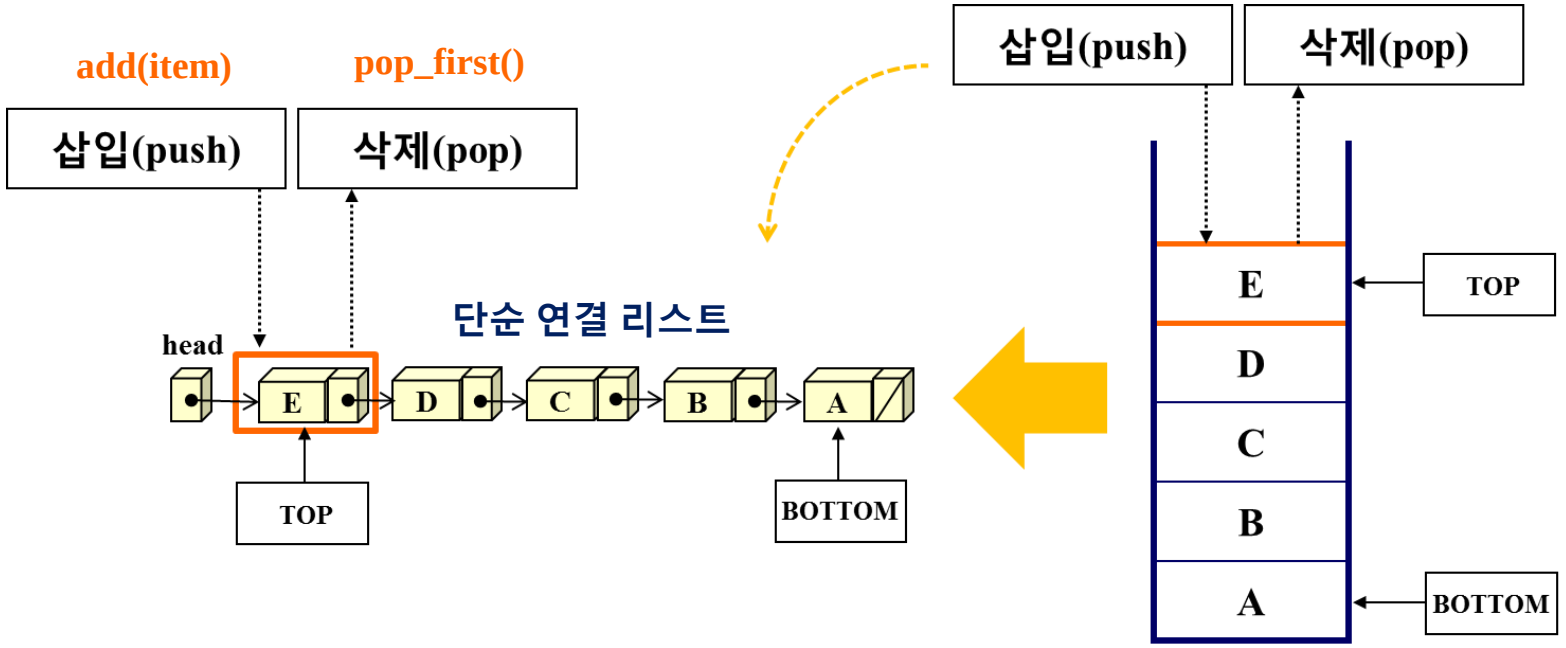
```
True
dog
3
False
8.4
True
2
```

결과

스택 (5/6)

❑ 단순 연결 리스트로 구현한 스택

- 단순 연결 리스트의 add(item)와 (항목 반환을 위해) 수정된 pop_first() 메소드로 push(item)와 pop() 구현 가능
  - Slide 4에서 언급한 대로, 단순 연결 리스트를 수정하여 사용하되 한쪽 끝(head)에서만 item을 삽입·삭제 하기로 약속하는 것보다 item 삽입·삭제 위치를 스택 Top으로 제한하는 것이 더 바람직하므로 Stack 클래스를 정의



스택 (6/6)

❑ 단순 연결 리스트로 구현한 스택 contd.

- 스택 객체를 위한 클래스 정의

```
1  class Node:
2      def __init__(self, item):
3          self.item = item
4          self.next = None
5
6      def get_item(self): return self.item
7
8      def get_next(self): return self.next
9
10     def set_item(self, new_item):
11         self.item = new_item
12
13     def set_next(self, new_next):
14         self.next = new_next
15
16 class Stack:
17     def __init__(self):
18         self.head = None
19
20     def is_empty(self):
21         return self.head == None
```

스택 Top

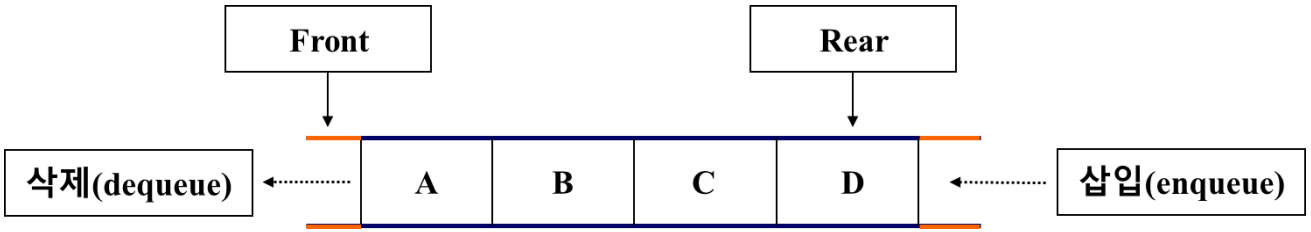
계속

```
23     def push(self, item):
24         temp = Node(item)
25         temp.set_next(self.head)
26         self.head = temp
27
28     def pop(self):
29         if self.is_empty():
30             return None
31         else:
32             popped_item = self.head.get_item()
33             self.head = self.head.get_next()
34             return popped_item
35
36     def peek(self):
37         if self.is_empty():
38             return None
39         else:
40             return self.head.get_item()
41
42     def size(self):
43         current = self.head
44         count = 0
45         while current != None:
46             count = count + 1
47             current = current.get_next()
48         return count
```

큐 (1/5)

❑ 큐(Queue)

- 한 쪽 끝 위치(**Rear**)에서 새로운 항목을 삽입(Enqueue)하고, 다른 한 쪽 끝 위치(**Front**)에서 기존 항목을 삭제 및 반환(Dequeue)하는 논리적 선형 구조
  - **시간 순으로** 먼저 삽입된 항목이 **먼저 삭제**되는, i.e., 선입선출(FIFO: **First-In, First-Out**) 구조
  - Rear(**뒤**)에서만 삽입이 이루어지고 Front(**앞**)에서만 삭제가 이루어지는(적용 가능한 연산이 제한된) **특수한 리스트(List)**



- **큐의 (물리적) 구현**
  - **Python 리스트(동적 배열)**에 의한 구현 – enqueue 연산의 시간복잡도 **O(1)**, dequeue 연산의 시간복잡도 **O(N)**
  - 단순 연결 리스트(Singly Linked List without Tail Variable)에 의한 구현 – enqueue 연산의 시간복잡도 **O(N)**, dequeue 연산의 시간복잡도 **O(1)**
  - **환형 연결 리스트(Circular Linked List)**에 의한 구현 – enqueue· dequeue 연산의 시간복잡도 **O(1)**

큐 (2/5)

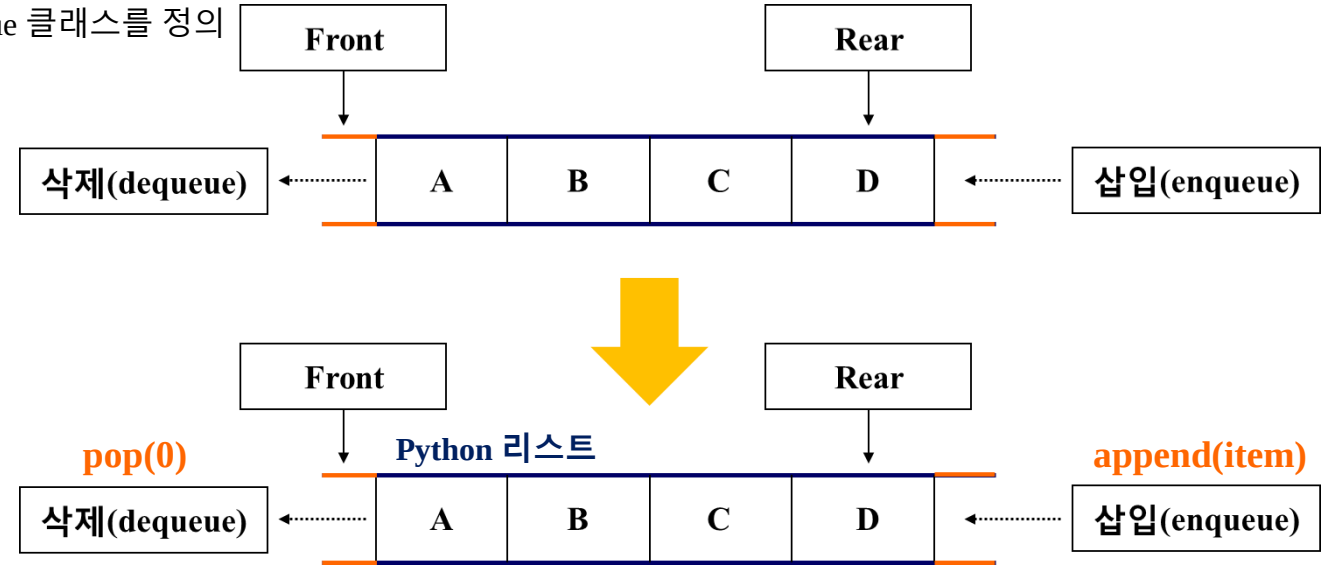
❑ 큐에 적용 가능한 주요 연산

- **Queue():** 빈 큐 생성
- **enqueue(item):** 기존 Rear 위치에 item 삽입
- **dequeue():** Front 위치에 존재하는 item **삭제 및 반환**
- **is_empty():** 큐가 empty이면 True 반환
- **size():** 큐의 사이즈 반환

⋮

❑ Python 리스트로 구현한 큐

- append(item)와 pop(0) 메소드로 enqueue(item)와 dequeue() 구현 가능하지만, Stack 클래스를 정의한 이유와 동일한 이유(Slide 4 참조)로 Queue 클래스를 정의



큐 (3/5)

Python 리스트로 구현한 큐 contd.

- 큐 객체를 위한 클래스 정의

```
1  class Queue:
2      def __init__(self):
3          self.items = []
4
5      def is_empty(self):
6          return self.items == []
7
8      def enqueue(self, item):
9          self.items.append(item)
10
11     def dequeue(self):
12         if self.is_empty():
13             return None
14         else:
15             return self.items.pop(0)
16
17     def size(self):
18         return len(self.items)
```

Queue 클래스

```
20 if __name__ == "__main__":
21
22     q = Queue()
23     print(q.is_empty())
24     q.enqueue(4)
25     q.enqueue('dog')
26     q.enqueue(True)
27     print(q.size())
28     print(q.is_empty())
29     q.enqueue(8.4)
30     print(q.dequeue())
31     print(q.dequeue())
32     print(q.size())
```

일련의 큐 연산과 출력

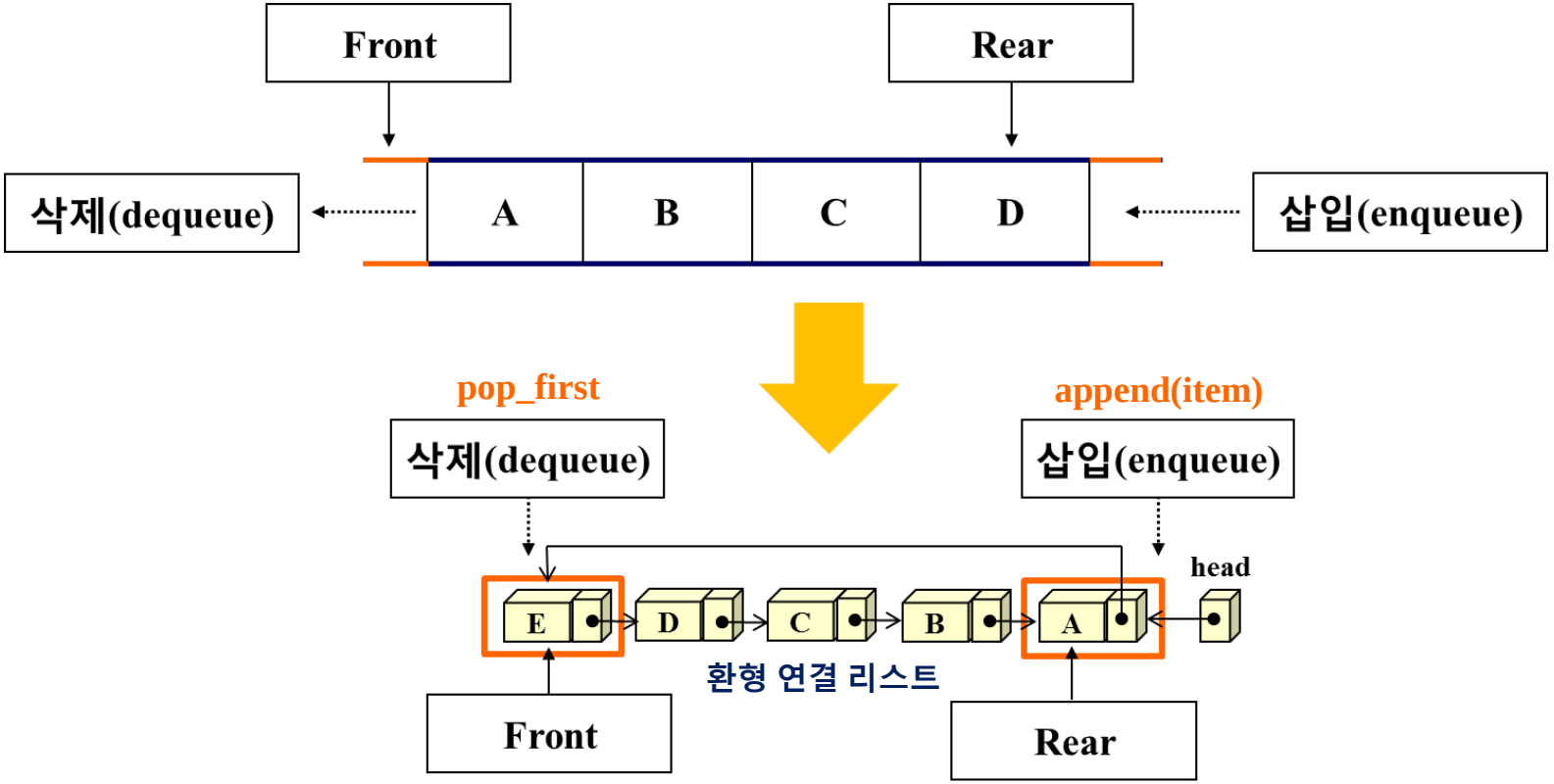
```
True
3
False
4
dog
2
```

결과

큐 (4/5)

❑ 환형 연결 리스트로 구현한 큐

- 환형 연결 리스트의 append(item)와 **(항목 반환을 위해) 수정된** pop_first() 메소드로 enqueue(item)와 dequeue() 구현 가능
  - Stack 클래스를 정의한 이유와 동일한 이유(Slide 4 참조)로 Queue 클래스를 정의



큐 (5/5)

❑ 환형 연결 리스트로 구현한 큐 contd.

- 큐 객체를 위한 클래스 정의

```
1  from node import Node
2  class Queue:
3      def __init__(self):
4          self.head = None
5
6      def is_empty(self):
7          return self.head == None
8
9      def enqueue(self, item):
10         temp = Node(item)
11         if self.is_empty():
12             temp.set_next(temp)
13             self.head = temp
14         else:
15             temp.set_next(self.head.get_next())
16             self.head.set_next(temp)
17             self.head = temp
```

큐 Rear

계속

```
19     def dequeue(self):
20         if self.is_empty():
21             return None
22         else:
23             temp = self.head.get_next()
24             dequeued_item = temp.get_item()
25             if temp == temp.get_next():
26                 self.head = None
27             else:
28                 self.head.set_next(temp.get_next())
29             return dequeued_item
30
31     def size(self):
32         count = 0
33         if self.is_empty():
34             return count
35         temp = self.head.get_next()
36         current = temp
37         while True:
38             count = count + 1
39             current = current.get_next()
40             if current != temp:
41                 continue
42             else:
43                 break
44         return count
```

삭제 항목 반환을 위한 지역 변수