

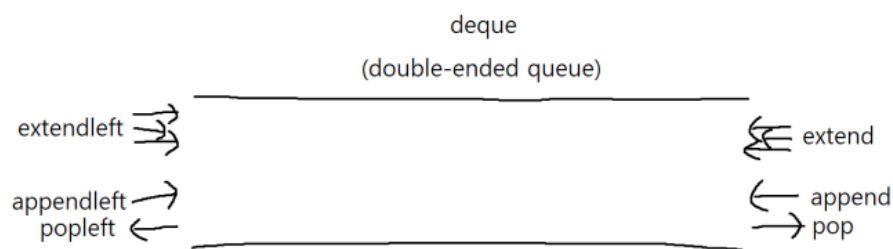
파이썬에서 큐(queue) 자료구조를 구현하기 위해 collections 모듈의 deque(덱)을 사용하고 있다.
제대로 알고 쓰기 위해 정리를 해보았다.

덱 (자료 구조)

덱(*deque*, "*deck*"과 발음이 같음 ← *double-ended queue*)은 양쪽 끝에서 삽입과 삭제가 모두 가능한 자료 구조의 한 형태이다.
두 개의 포인터를 사용하여, 양쪽에서 삭제와 삽입을 발생시킬 수 있다. 큐와 스택을 합친 형태로 생각할 수 있다.

[https://ko.wikipedia.org/wiki/%EB%8D%B1_\(%EC%9E%90%EB%A3%8C_%EA%B5%AC%EC%A1%B0\)](https://ko.wikipedia.org/wiki/%EB%8D%B1_(%EC%9E%90%EB%A3%8C_%EA%B5%AC%EC%A1%B0))

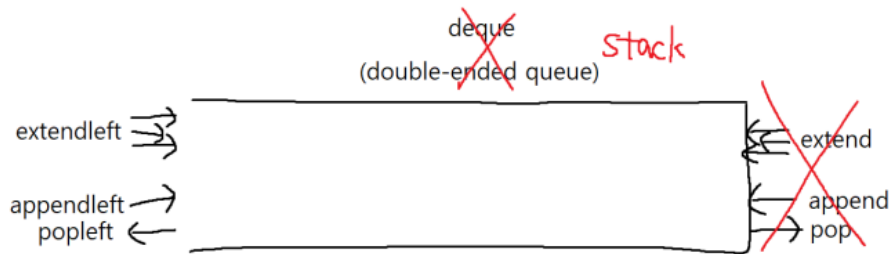
양 끝에서의 삽입, 삭제와 관련된 method 일부를 그림으로 표현해 보았다.



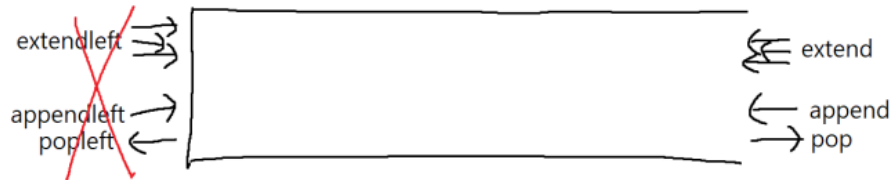
extend, append, pop 뒤에 right가 생략되어 있다고 생각하면 이해하기 편하다.

변형

한쪽 방향의 함수만 사용한다면 스택(stack)으로 이용할 수 있다.



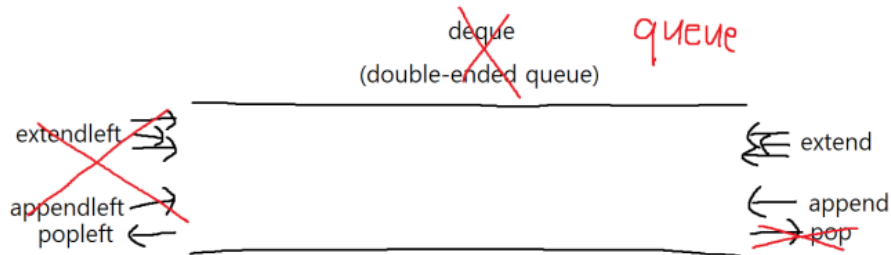
or



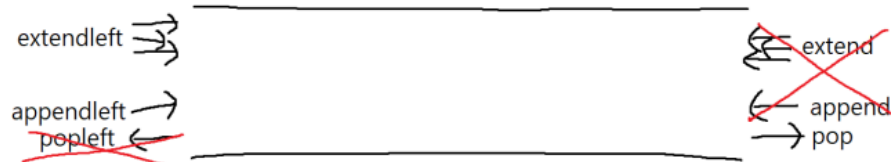
한쪽 방향에서는 삽입 함수, 반대쪽 방향에서는 삭제 함수만 사용한다면 큐로 이용할 수 있다.

지금까지는 큐로 이용해왔기에, 보통 append, extend와 popleft 함수만 이용해왔다.

당연히 반대로 appendleft, extendleft와 pop 함수만 이용해도 큐로 이용할 수 있을 것이다.



or



데크의 종류

스크롤 - 입력이 한쪽 끝으로만 가능하도록 설정한 데크(입력 제한 데크)

셀프 - 출력이 한쪽 끝으로만 가능하도록 설정한 데크(출력 제한 데크)

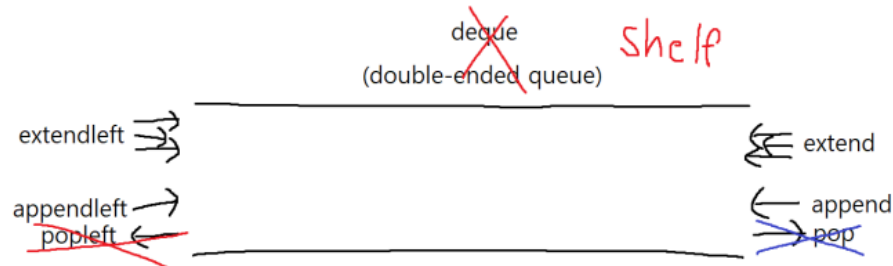
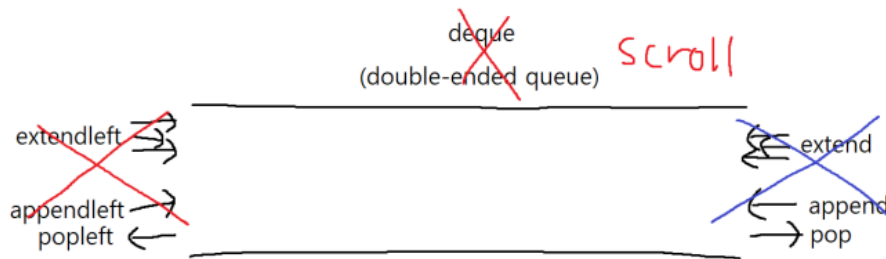
[https://ko.wikipedia.org/wiki/%EB%8D%B1_\(%EC%9E%90%EB%A3%8C_%EA%B5%AC%EC%A1%B0\)](https://ko.wikipedia.org/wiki/%EB%8D%B1_(%EC%9E%90%EB%A3%8C_%EA%B5%AC%EC%A1%B0))

한쪽에서의 삽입에 제한을 둔 덱을 입력 제한 덱(Input restricted deque) 혹은 스크롤(Scroll)이라고 하며,

한쪽에서의 삭제에 제한을 둔 덱을 출력 제한 덱(Output restricted deque) 혹은 셀프(Shelf)라고 한다.

앞서 표현한 것과 같이 그림으로 표현하자면 아래와 같은 느낌이겠다.

파랑 X, 빨강 X 표시한 부분의 함수 둘 중 하나를 선택하여 사용을 제한하면 된다.



나머지 method까지 정리한 표는 다음과 같다.

n 은 deque 요소의 개수, k 는 extend의 경우 extend 하는 요소들의 개수를 말한다.

리턴 값을 따로 표 항목으로 정리하면 좋을 텐데 귀찮아서...

아래 블로그 글을 참고했다.

<https://hbj0209.tistory.com/30>

함수(Method)	시간 복잡도(Time Complexity)	설명(Description)
append(x)	$O(1)$	deque의 맨 오른쪽에 x 원소를 추가한다.
appendleft(x)	$O(1)$	deque의 맨 왼쪽에 x 원소를 추가한다.
clear()	$O(n)$	deque의 모든 원소를 삭제한다.
copy()	$O(n)$	deque의 shallow copy를 리턴한다.
count(x)	$O(n)$	deque의 원소 중 x인 원소의 개수를 리턴한다. 없을 시 0을 리턴한다.
extend(iterable)	$O(k)$	deque의 오른쪽에 iterable 한 원소들을 추가한다. (파이썬 'str' 객체(object)는 iterable 하고, 'int' 객체는 iterable 하지 않다.) iterable 하지 않은 원소를 추가하면 TypeError가 발생한다.
extendleft(iterable)	$O(k)$	deque의 왼쪽에 iterable 한 인자들을 추가한다. (파이썬 'str' 객체(object)는 iterable 하고, 'int' 객체는 iterable 하지 않다.) iterable 하지 않은 원소를 추가하면 TypeError가 발생한다.
index(x[, start[, stop]])		deque 안에 있는 x의 인덱스를 리턴한다. 일치하는 원소가 여러 개일 경우 가장 작은 인덱스를 리턴한다. []는 없어도 되고, 있으면 start에서 stop 사이의 범위 내로 한정한다. x가 deque 안에 없으면 ValueError가 발생한다.
insert(i, x)	$O(n)$	x를 i 번째 인덱스에 삽입한다.

		deque의 범위를 넘어서 i가 접근할 경우 IndexError가 발생한다.
pop()	O(1)	deque의 맨 오른쪽에 있는 원소를 삭제하고 삭제한 원소를 리턴한다. 원소가 없을 시 IndexError가 발생한다.
popleft()	O(1)	deque의 맨 왼쪽에 있는 원소를 삭제하고 삭제한 원소를 리턴한다. 원소가 없을 시 IndexError가 발생한다.
remove(value)	O(n)	value와 일치하는 원소를 제거한다. 일치하는 원소가 여러 개일 경우 가장 작은 인덱스의 원소를 지운다. value에 해당하는 원소가 없을 시 ValueError가 발생한다.
reverse()	O(n)	deque의 원소들을 뒤집는다. 다시 말하면, deque의 모든 원소들의 현재 인덱스 값을 (deque의 맨 끝 인덱스 값 - 현재 인덱스 값)으로 바꾼다.
rotate(k)	O(k)	deque의 원소들을 k 번씩 이동시킨다. k에 양수를 넣으면 오른쪽으로, 음수를 넣으면 왼쪽으로 이동한다. 맨 오른쪽에 있는 원소는 오른쪽으로 이동하면 맨 왼쪽으로, 맨 왼쪽에 있는 원소는 왼쪽으로 이동하면 맨 오른쪽으로 온다. 다시 말하면, deque의 모든 원소들의 현재 인덱스 값을 {(현재 인덱스 값 + k) % (deque의 맨 끝 인덱스 값)}으로 바꾼다.

copy 함수가 리턴하는 값이 왜 shallow copy(얕은 복사)인지 조금 헷갈렸다.

이참에 shallow copy, deep copy 개념을 바로 잡기로 했다.

내가 이해한 대로라면 d1과 d1을 shallow copy 한 d2는 서로 값이 변경되면 영향을 받아야 한다.

하지만 d1[0], d2[0] 둘 다 같은 객체이다가, d1[0]에 값을 할당하자 d1[0]이 다른 객체가 되어 버렸다.

즉 d2[0]은 그대로인데, d1[0]만 바뀐 것이다.

```

from collections import deque

l1 = [1, 2, 3]
d1 = deque(l1)
d2 = d1.copy()

print("d1 =", id(d1))
print("d2 =", id(d2))
print("d1 is d2 =", d1 is d2)
print("d1[0] =", id(d1[0]))
print("d2[0] =", id(d2[0]))
print("d1[0] is d2[0] =", d1[0] is d2[0])

d1[0] = 0

print("d1[0] =", id(d1[0]))
print("d2[0] =", id(d2[0]))
print("d1[0] is d2[0] =", d1[0] is d2[0])

''' 출력(매번 실행 시 숫자 값 자체는 달라질 수 있음)
d1 = 140154514109632
d2 = 140154514111264
d1 is d2 = False
d1[0] = 140154583291616
d2[0] = 140154583291616
d1[0] is d2[0] = True
d1[0] = 140154583291584
d2[0] = 140154583291616
d1[0] is d2[0] = False
'''

```

참고로 id 함수는 객체를 인자로 받아서 객체의 고유값을 반환하는 함수, is 또는 is not은 동일한 객체 여부를 판별하는 연산자이다.

"==", "!=" 같은 관계 연산자와는 조금 다르다. "=="은 값(데이터) 그 자체를 비교하는 것이지만 is는 객체의 고유값을 비교한다.

```

a = [1, 2, 3]
b = [1, 2, 3]
print(a is b)
print(a == b)
''' 출력
False
True
'''

```

파이썬에서 mutable(변경 가능한) 객체는 할당된 값이 바뀌더라도 같은 객체를 가리키고,

immutable(변경 불가능한) 객체는 할당된 값이 바뀌면 다른 객체를 가리킨다.

d1[0]은 'int', immutable 객체이다. 값이 1에서 0으로 바뀌었으니 다른 객체를 가리키는 것이 당연하다.

하지만 리스트 안에 리스트, 즉 mutable 객체 안에 mutable 한 객체가 들어간 텍을 copy 하는 경우 이야기가 달라진다.

이번엔 d1을 이러한 형태의 텍으로 초기화하고 d1[0][0]을 0으로 바꾸어보았다.

이제 d1[0]은 'int'가 아닌 'list', mutable 객체이다. 따라서 리스트 안의 원소인 d1[0][0]이 바뀌더라도 같은 객체를 가리킨다.

d1과 d1을 shallow copy 한 d2는 서로 영향을 받기에, 완전히 별도의 객체라고 생각하면 안 된다.
내부의 객체들까지 완전히 copy 하기 위해서는 deep copy(깊은 복사)를 해야 한다.

```
from collections import deque

l1 = [[1, 2, 3], [4, 5, 6]]
d1 = deque(l1)
d2 = d1.copy()

print("d1 =", id(d1))
print("d2 =", id(d2))
print("d1 is d2 =", d1 is d2)
print("d1[0] =", id(d1[0]))
print("d2[0] =", id(d2[0]))
print("d1[0] is d2[0] =", d1[0] is d2[0])

d1[0][0] = 0

print("d1[0] =", id(d1[0]))
print("d2[0] =", id(d2[0]))
print("d1[0] is d2[0] =", d1[0] is d2[0])

''' 출력 (매번 실행 시 숫자 값 자체는 달라질 수 있음)
d1 = 140138281751744
d2 = 140138281753376
d1 is d2 = False
d1[0] = 140138281694208
d2[0] = 140138281694208
d1[0] is d2[0] = True
d1[0] = 140138281694208
d2[0] = 140138281694208
d1[0] is d2[0] = True
'''
```

shallow copy, deep copy에 대한 보다 자세한 설명은 아래 블로그 글을 참고하면 된다.

<https://blog.naver.com/ekldhrtlsda/221764190608>

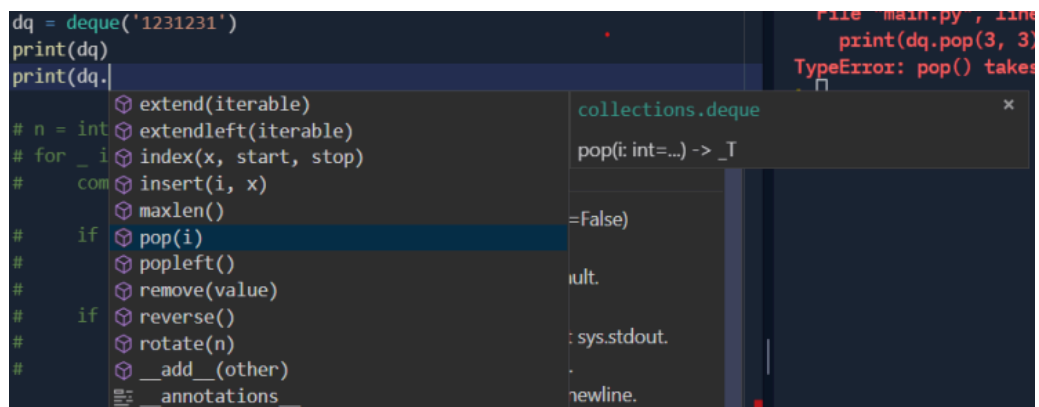
<https://wikidocs.net/16038>

파이썬의 mutable, immutable 객체들을 비교하여 설명한 글은 아래에!

<https://zzonglove.tistory.com/21>

레퍼런스(참조), 포인터(주소)가 유사한 개념이라는 걸 이제야 알았다. 정확한 차이는 나중에 제대로 공부해야겠다.

<https://namocom.tistory.com/869>



```
dq = deque('1231231')
print(dq)
print(dq)

# n = int
# for _ i index(x, start, stop)
# com insert(i, x)
# maxlen()
# if pop(i)
# popleft()
# remove(value)
# if reverse()
# rotate(n)
# __add__(other)
# __annotations__
```

collections.deque

pop(i: int=...) -> _T

=False)

ult.

: sys.stdout.

newline.

replit에서 pop 함수에 인자가 1개 필요한 것처럼 나오는데, 이건 원인은 모르겠지만 오류인 듯.

우상단에 살짝 나와있는데, pop 함수에 인자를 넣으면 pop 함수는 인자를 받지 않으므로 TypeError가 발생한다.
애초에 pop()은 인자를 받고 popleft()는 인자를 받지 않는 것부터 이상하다.

참고로 pop 위에 있는 maxlen 함수는 호출하면 TypeError: 'NoneType' object is not callable 에러가 발생한다.
덱을 좀 이해한 것 같다면, 연습 삼아 아래 문제를 풀어볼 것을 추천한다.

<https://www.acmicpc.net/problem/10866>



10866번: 덱

문제 정수를 저장하는 덱(Deque)를 구현한 다음, 입력으로 주어지는 명...

www.acmicpc.net

내 풀이는 여기에

```
from collections import deque
import sys

dq = deque()

n = int(sys.stdin.readline())
for _ in range(n):
    command = list(map(str, sys.stdin.readline().split()))

    if command[0] == "push_front":
        num = int(command[1])
        dq.appendleft(num)
    if command[0] == "push_back":
        num = int(command[1])
        dq.append(num)

    if command[0] == "pop_front":
        if dq:
            num = dq.popleft()
            print(num)
        else:
            print(-1)
    if command[0] == "pop_back":
        if dq:
            num = dq.pop()
            print(num)
        else:
            print(-1)
    if command[0] == "size":
        print(len(dq))
    if command[0] == "empty":
        print(0 if dq else 1)
    if command[0] == "front":
        print(dq[0] if dq else -1)
    if command[0] == "back":
        print(dq[len(dq) - 1] if dq else -1)
```