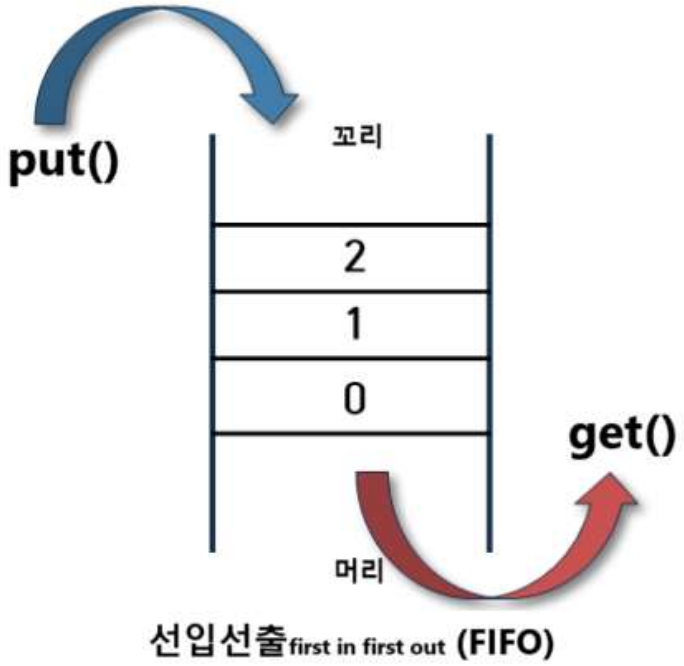
Queue

주요 내용

- 큐 추상 자료형
- 큐 자료구조
- 큐 활용

큐의 정의

- 큐queue: 항목 추가는 꼬리에서, 항목 삭제는 머리에서 이루어지는 선형 자료형
- 먼저 들어온 항목이 먼저 나간다는 **선입선출**first in, first out(FIFO) 원리를 따름
- 항목의 삭제가 추가된 순서대로 진행



큐 활용 예제

- 은행: 대기 순서에 따른 번호표 배분
- 프린터: 순서에 따른 서류 인쇄
- 키보드 입력: 키보드 입력값을 버퍼buffer에 잠시 저장한 다음에 처리 입력 순서대로 처리
- 컴퓨터 CPU: 사용자 명령을 순차적으로 처리

Queue 추상 자료형

- `Queue(maxsize=0)`: 비어 있는 큐 생성. `maxsize`를 최대로 많이 담을 수 있는 항목의 수. 0인 경우엔 항목 수 제한 없음.
- `put(item)`: `maxsize`가 초과되지 않을 때 꼬리에 항목 추가
- `get()`: 머리 항목 삭제. 삭제된 항목 반환.
- `empty()`: 큐가 비었는지 여부 판단. 부울값 반환.
- `full()`: 큐가 꽉 차 있는지 여부 판단. 부울값 반환.
- `qsize()`: 큐에 포함된 항목 개수 반환.

Queue 추상 클래스

In [1]:
```
class Queue:
    def __init__(self, maxsize=0):
        raise NotImplementedError

    def qsize(self):
        raise NotImplementedError

    def empty(self):
        raise NotImplementedError

    def full(self):
        raise NotImplementedError

    def put(self, item):
        raise NotImplementedError

    def get(self):
        raise NotImplementedError
```

큐 자료구조 구현

`collections.deque` 활용

- 꼬리에서의 항목 추가와 삭제를 빠르게 실행

In [2]:
```
from collections import deque
```

In [3]:

```
class myQueue(Queue):
    def __init__(self, maxsize=0):
        self._maxsize = maxsize
        self._container = deque([])

    def __repr__(self):
        return repr(self._container).replace("deque", 'myQueue')

    def qsize(self):
        return len(self._container)

    def empty(self):
        return not self._container

    def full(self):
        if self._maxsize <= 0:
            return False
        elif self.qsize() < self._maxsize:
            return False
        else:
            return True

    def put(self, item):
        if not self.full():
            self._container.appendleft(item)
        else:
            print("추가되지 않아요!")

    def get(self):
        return self._container.pop()
```

In [12]:

```
q = myQueue(maxsize=4)

q.put(4)
q.put("dog")
q.put(True)
print(q)
q.put(8.4)
print(q.full())
print(q)
q.put("하나 더?")
print(q)
print(q.get())
print(q.get())
print(q.qsize())
print(q)
print(q.empty())
```

```
myQueue([True, 'dog', 4])
True
myQueue([8.4, True, 'dog', 4])
추가되지 않아요!
myQueue([8.4, True, 'dog', 4])
4
dog
2
myQueue([8.4, True])
False
```

`queue.Queue` 클래스

- 파이썬 `queue` 모듈이 큐 추상 자료형을 자료구조로 구현한 `Queue` 클래스 지원
- `collections.deque`를 이용하여 정의한 `myQueue`가 항목의 추가와 삭제를 4, 5배 정도 빠르게 실행

put() 메서드 비교

In [5]:
```
%%time

n = 100_000
q1 = myQueue(maxsize=0)

for k in range(n):
    q1.put(k)
```

```
CPU times: user 10.5 ms, sys: 0 ns, total: 10.5 ms
Wall time: 10.4 ms
```

In [6]:
```
import queue
```

In [7]:
```
%%time

n = 100_000
q2 = queue.Queue(maxsize=0)

for k in range(n):
    q2.put(k)
```

```
CPU times: user 52.5 ms, sys: 0 ns, total: 52.5 ms
Wall time: 52.4 ms
```

`get()` 메서드 비교

In [8]:
```
%%time

for k in range(n):
    q1.get()
```

```
CPU times: user 7.23 ms, sys: 0 ns, total: 7.23 ms
Wall time: 7.33 ms
```

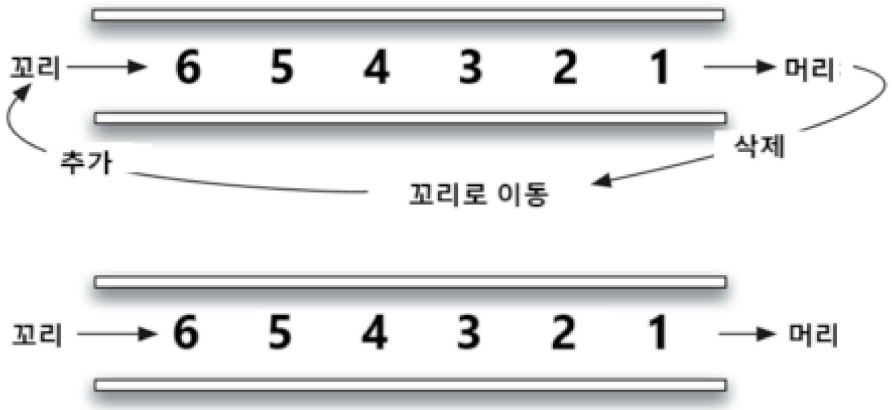
In [9]:
```
%%time

for k in range(n):
    q2.get()
```

```
CPU times: user 53.5 ms, sys: 0 ns, total: 53.5 ms
Wall time: 53.3 ms
```

큐 활용: 폭탄 돌리기 게임

게임 구현

- 게임 시작: 참여자들의 수 `player`와 폭탄 돌리기 횟수 `count`
- 폭탄 시작 위치: 큐의 머리
- 폭탄 전달: 머리 항목 삭제 후 바로 꼬리에 추가
- 탈락: `cout`번의 폭탄 돌리기 이후 머리에 위치한 사람 탈락
- 게임 정지: 한 명이 남을 때까지 반복



In [10]:
```
def bombing(player, count):

    player_queue = myQueue()

    for p in range(1, player+1):
        player_queue.put(p)

    while player_queue.qsize() > 1:
        for i in range(count):
            player_queue.put(player_queue.get())

        player_queue.get()

    return player_queue.get()
```

In [11]:
```
print(bombing(6, 5))
```

```
4
```