

큐(Queue)

# 주요 내용

- 선형 자료구조
- 큐 자료구조
- 큐 활용

# 선형 자료구조

- 선형 자료구조
  - 여러 개의 항목을 일렬로 나열하여 저장
  - 나열된 순서, 즉 저장된 위치가 중요
- 항목의 추가/삭제 방식에 따라 작동
  - 큐 queue
  - 스택 stack
  - 덱 deque
  - 연결 리스트 linked list
  - 정렬 리스트 sorted list

- 선형 자료구조의 양 끝을 부르는 이름
  - 머리: 헤드(head), 탑(top), 프론트(front)
  - 꼬리: 테일(tail), 바텀(bottom), 리어(rear)
- 파이썬 제공
  - 큐와 스택: `queue` 모듈의 `Queue` 클래스와 `LifoQueue` 클래스
  - 덱: `collections` 모듈의 `deque` 클래스

# 큐의 정의

- 먼저 들어온 항목이 먼저 나간다는 **선입선출** first in, first out(FIFO) 원리를 따름
- 항목들의 추가와 삭제는 입력 순서에 따라 한쪽 방향으로 이동



## 큐의 활용

- 은행 창구에서 번호표
- 프린터는 인쇄 과제가 생성된 순서대로 출력
- 키보드 타이핑 내용이 버퍼(buffer)와 같은 큐에 임시 저장되어 있다가 순서대로 편집기에 표시됨.
- 컴퓨터 CPU가 사용자가 내린 명령문을 처리하는 것 또한 특정 형식의 큐를 이용

## 예제: 버퍼 활용

```
>>> help(print)
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    ... Prints the values to a stream, or to sys.stdout by default.

    ... sep
        ... string inserted between values, default a space.
    ... end
        ... string appended after the last value, default a newline.
    ... file
        ... a file-like object (stream); defaults to the current sys.stdout.
    ... flush
        ... whether to forcibly flush the stream.
```

- `end` 키워드와 `flush` 키워드 조합

```
import time

for i in range(10):
    print(i, end=' ')
    time.sleep(1)
```

```
C:\Users\jisun\PycharmProjects\algorithms>
```

- `flush=True`로 지정하면 다르게 작동함.

```
import time

for i in range(10):
    print(i, end=' ', flush=True)
    time.sleep(1)
```

# Queue 추상 자료형

- `Queue(maxsize=0)`: 비어 있는 큐 생성. `maxsize` 를 양의 정수로 지정하면 항목을 최대 그만큼 담을 수 있음. 아니면 제한 없음.
- `put(item)`: `maxsize`가 양의 정수인 경우 최대 항목수를 초과 한 경우 먼저 머리(`head, front`)를 제거한 다음 새로운 항목을 꼬리(`rear, tail`)에 추가. 반환값 없음.
- `get()`: 머리(`head, front`) 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.

# 큐 자료구조 구현

```
class Queue:
    def __init__(self, maxsize=0):
        self.maxsize = maxsize
        self._items = []
    def __repr__(self):
        return f"<<{self._items}>>"
    def empty(self):
        return not bool(self._items)
    def full(self):
        if self.maxsize <= 0:
            return False
        elif self.qsize() == self.maxsize:
            return True
        else:
            return False
    def put(self, item):
        if self.full() == True:
            self.get()
        self._items.insert(0, item)
    def get(self):
        return self._items.pop()
    def qsize(self):
        return len(self._items)
```

```
>>> q = Queue(maxsize=4)

>>> print(q.empty())
True

>>> q.put(4)

>>> q.put("dog")

>>> q.put(True)

>>> print(q)
<<[True, 'dog', 4]>>

>>> print(q.full())
False

>>> print(q.qsize())
3

>>> print(q.empty())
False
```

```
>>> q.put(8.4)

>>> print(q.full())
True

>>> print(q)
<<[8.4, True, 'dog', 4]>>

>>> q.put("하나 더?")

>>> print(q)
<<['하나 더?', 8.4, True, 'dog']>>

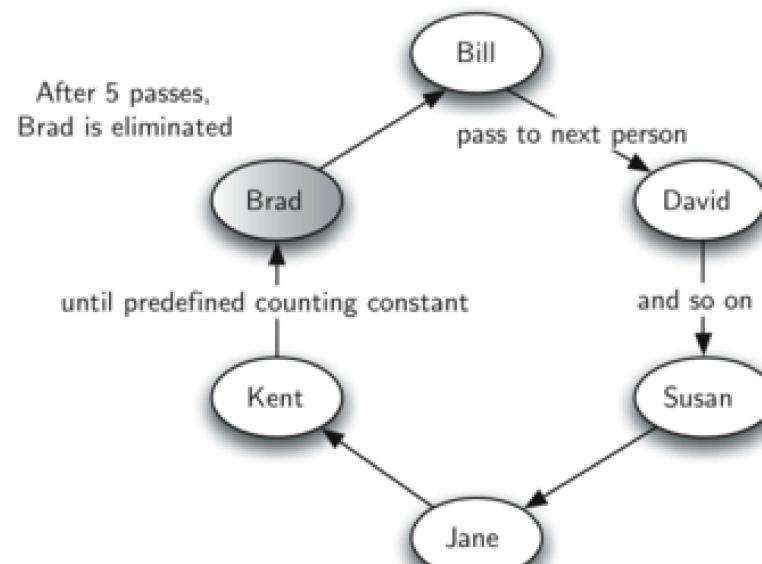
>>> print(q.get())
dog

>>> print(q.get())
True

>>> print(q.qsize())
2

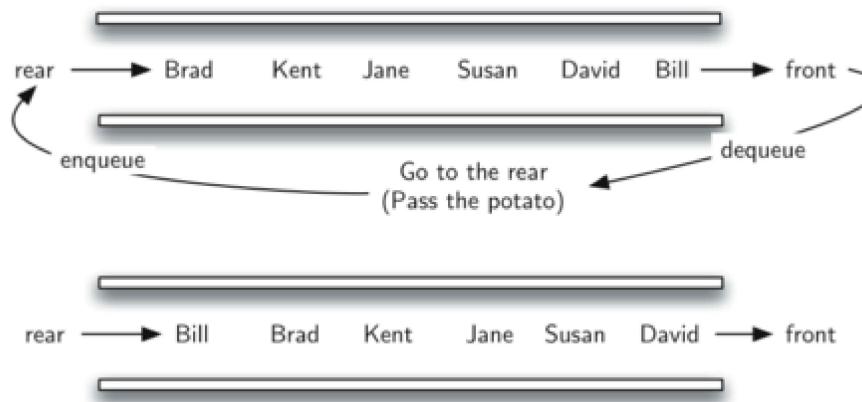
>>> print(q)
<<['하나 더?', 8.4]>>
```

# 실전 예제 1: 폭탄 돌리기



# 게임 구현

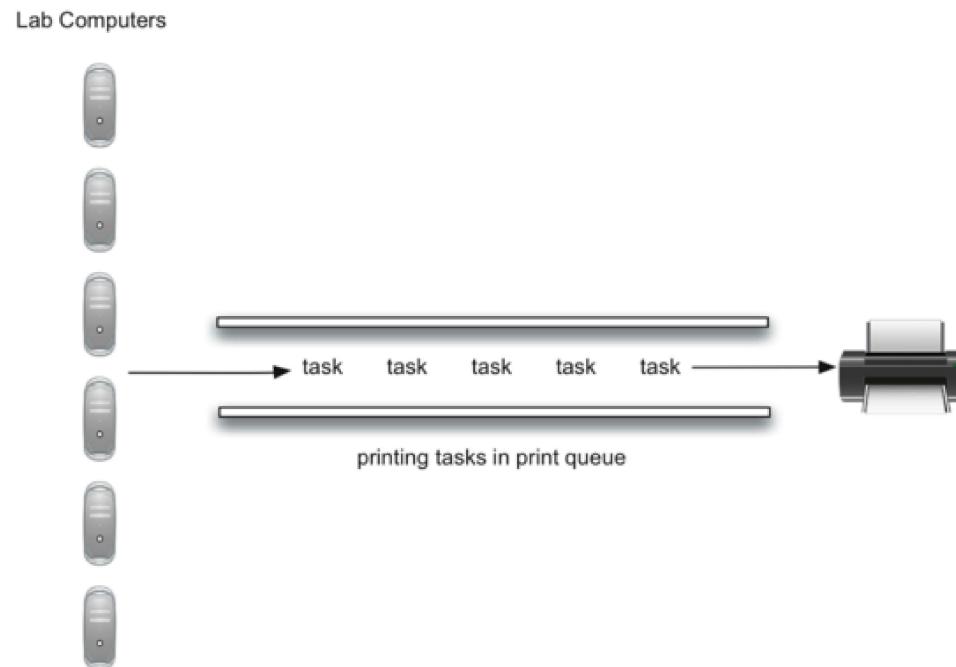
- 게임 시작: 사람들의 리스트(`name_list`)와 정수(`num`)
- 폭탄 위치: 큐의 머리(리스트의 가장 오른편)에 있는 사람
- 폭탄 전달: 머리 항목 삭제 후 바로 꼬리(리스트의 0번 인덱스 위치)에 추가
- 탈락: `num` 번의 폭탄 돌리기 이후 머리에 위치한 사람 탈락
- 게임 정지: 한 명이 남을 때까지 반복



```
def hot_potato(name_list, num):
    sim_queue = Queue()
    for name in name_list:
        sim_queue.put(name)
    while sim_queue.qsize() > 1:
        for i in range(num):
            sim_queue.put(sim_queue.get())
        sim_queue.get()
    return sim_queue.get()

>>> print(hot_potato(["형택", "진서", "은혜", "민규", "정은", "청용"], 7))
은혜
```

# 실전 예제 2: 프린터 인쇄 모의실험



## 필요한 객체 확인

- 프린터 객체: `Printer` 클래스의 인스턴스. 프린터의 기본 기능 제공
  - 인쇄 속도: 분당 출력 쪽 수(ppm)
  - 프린터 상태: `busy` 또는 대기
  - 인쇄 과제 실행
  - 인쇄 과제별 수행 시간 측정
- 인쇄 과제 객체: `Task` 클래스의 인스턴스. 실행할 인쇄 과제 정보 저장
  - 출력 쪽 수
  - 인쇄 과제가 생성된 시간
  - 인쇄까지 대기 시간: 생성부터 인쇄 시작까지의 대기 시간
- 인쇄 큐 객체: `Queue` 클래스의 인스턴스
  - 인쇄 과제 대기 목록

## 180분의 1의 확률

모의실험에 대한 추가 전제 사항이 있다.

- 사람들이 무작위적으로 시간당 20건의 인쇄 명령 실행
- 인쇄는 과제당 최대 20쪽까지 허용.

$$\frac{20 \text{ 건}}{1 \text{ 시간}} = \frac{20 \text{ 건}}{3600 \text{ 초}} = \frac{1 \text{ 건}}{180 \text{ 초}}$$

```
def new_print_task():
    ... return 180 == random.randrange(1, 181)
```

## Printer 클래스

- 분당 출력 쪽 수
- 다음 인쇄 과제 지정하기
- 현재 실행 중인 인쇄 과제 확인
- 수행중인 인쇄 과제의 남은 실행 시간동안 busy 상태 유지. 인쇄 과정을 간접접으로 묘사함.

```
class Printer:
    def __init__(self, ppm):
        self.page_rate = ppm
        self.current_task = None
        self.time_remaining = 0

    def tick(self):
        if self.current_task is not None:
            self.time_remaining = self.time_remaining - 1
            if self.time_remaining <= 0:
                self.current_task = None

    def busy(self):
        return self.current_task is not None

    def start_next(self, new_task):
        self.current_task = new_task
        self.time_remaining = new_task.get_pages() * 60 / self.page_rate
```

## Task 클래스

- 인쇄 과제 생성 시간
- 인쇄 대상 페이지 수: 1~20 사이의 무작위 수
- 과제 생성 후 인쇄 시작까지 대기 시간

```
import random

class Task:  
    def __init__(self, time):  
        self.timestamp = time  
        self.pages = random.randrange(1, 21)  
  
    def get_stamp(self):  
        return self.timestamp  
  
    def get_pages(self):  
        return self.pages  
  
    def waiting_time(self, current_time):  
        return current_time - self.timestamp
```

## 모의실험 내용

### 1. 초당 180분의 1의 확률로 인쇄 과제 생성

- 생성된 시간 저장. 나중에 프린터가 실행될 때의 시간을 확인하여 대기시간을 측정할 수 있도록 함.
- 인쇄 과제 생성 후 바로 인쇄 과제 큐에 추가

### 1. 프린터가 대기 상태이고 인쇄 과제 큐에 과제가 남아 있으면 아래 과제 수행

- 인쇄 과제 큐의 헤드를 삭제하고 수행할 과제로 지정
- 해당 과제의 대기 시간을 계산(현재 시간과 과제 생성시간의 차이)한 후에 모든 과제의 대기 기간 리스트에 추가
- 인쇄 과제의 인쇄 쪽 수를 확인한 후에 인쇄에 필요한 시간 계산. 해당 시간 동안 프린터 상태가 `busy`로 표시되어 다음 인쇄 과제들은 큐에서 기다리게 됨.
- 해당 인쇄 과제가 완수되면 대기 상태로 전환됨.

## 모의실험 구현

- `simulation()` 함수: 지정된 시간동안 인쇄 작업을 수행할 때 인쇄 과제당 평균 대기 시간 계산
  - `num_seconds`: 프린터 작동 시간
  - `pages_per_minutes`: 분당 출력 페이지 수
- 가정: `for` 반복문이 초당 1회 실행
  - `new_print_task()`
  - `tick()`

```
def simulation(num_seconds, pages_per_minute):
    lab_printer = Printer(pages_per_minute)
    print_queue = Queue()
    waiting_times = []

    for current_second in range(num_seconds):
        if new_print_task():
            task = Task(current_second)
            print_queue.put(task)
        if (not lab_printer.busy()) and (not print_queue.empty()):
            nexttask = print_queue.get()
            waiting_times.append(nexttask.waiting_time(current_second))
            lab_printer.start_next(nexttask)
            lab_printer.tick()
        average_wait = sum(waiting_times) / len(waiting_times)

    return average_wait
```

# 모의 실험 1

- 프린터 성능과 실행시간
  - 분당 출력 페이지 수: 5장
  - 프린터를 실행 시간: 1시간
- 모의실험 수행 횟수: 100번

```
>>> import numpy as np
>>> average_wait_list = []
>>> for i in range(100):
...     average_wait_list.append(simulation(3600, 5))
>>> print(f"평균 대기시간: {np.mean(average_wait_list):6.2f} 초")
평균 대기시간: 123.28 초
```

## 모의 실험 2

- 분당 출력 페이지수: 10장

```
>>> import numpy as np
>>> average_wait_list = []
>>> for i in range(100):
...     average_wait_list.append(simulation(3600, 10))
>>> print(f"평균 대기시간: {np.mean(average_wait_list):6.2f} 초")
평균 대기시간: 19.40 초
```