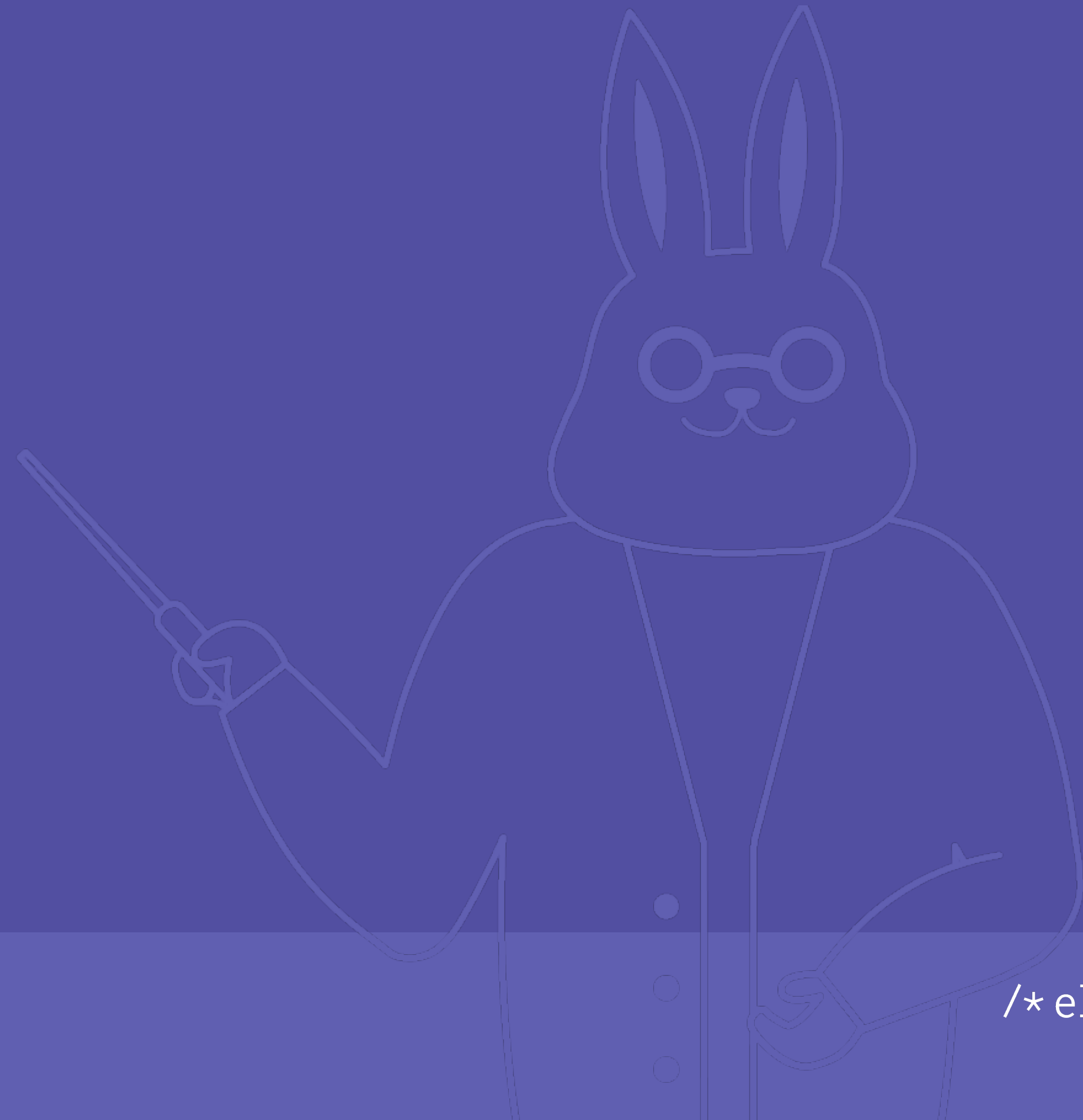


자료구조

2장 스택과 큐

김경민 선생님



Contents

- 01. 스택, 큐의 개념
- 02. 스택, 큐의 의미
- 03. 스택으로 풀 수 있는 문제
- 04. 큐로 풀 수 있는 문제

01

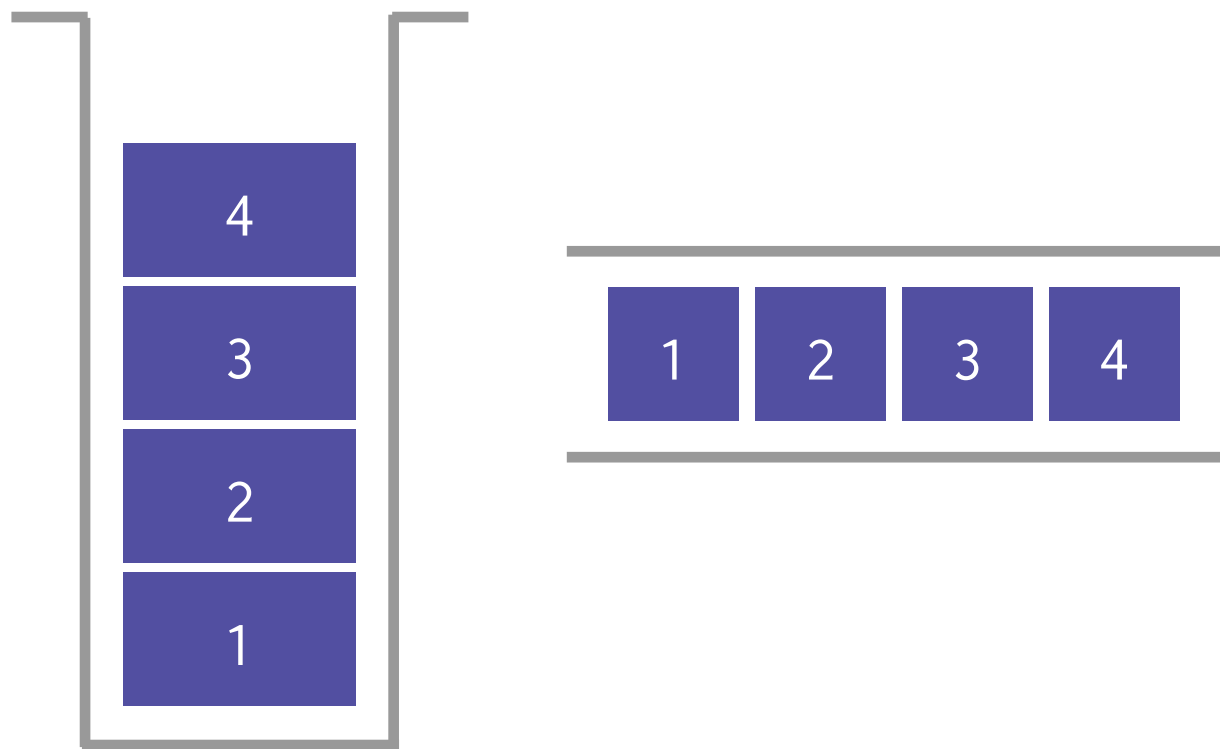
스택, 큐의 개념



01 스택, 큐의 개념

✔ 대표적인 자료구조의 예시

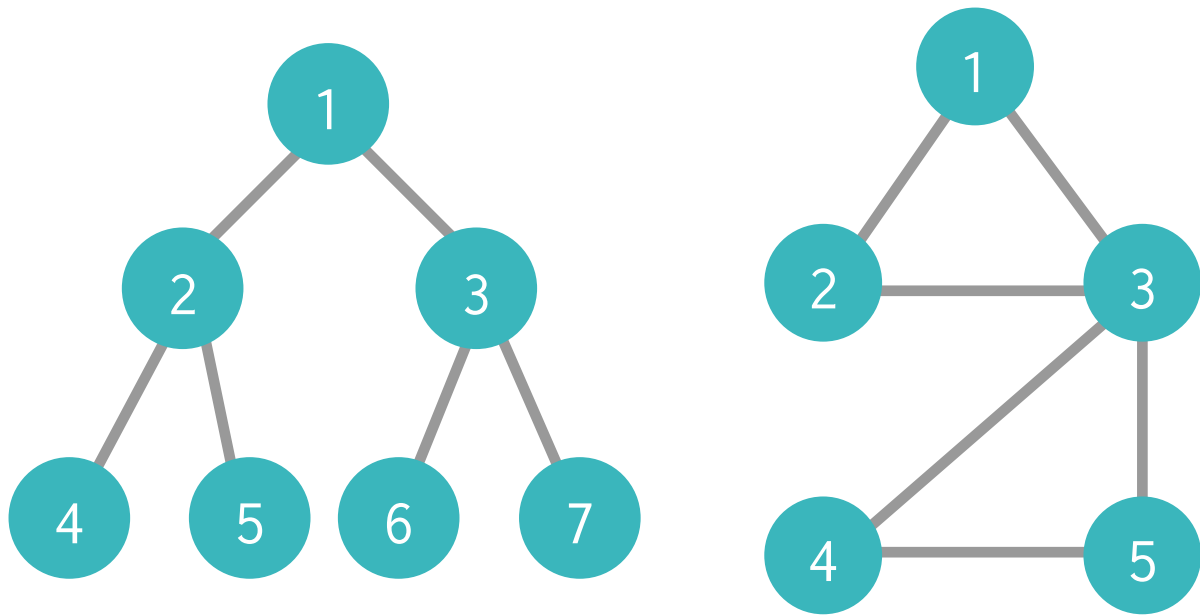
선형 구조



스택 (Stack)

큐 (Queue)

비선형 구조



트리 (Tree)

그래프 (Graph)

01 스택, 큐의 개념

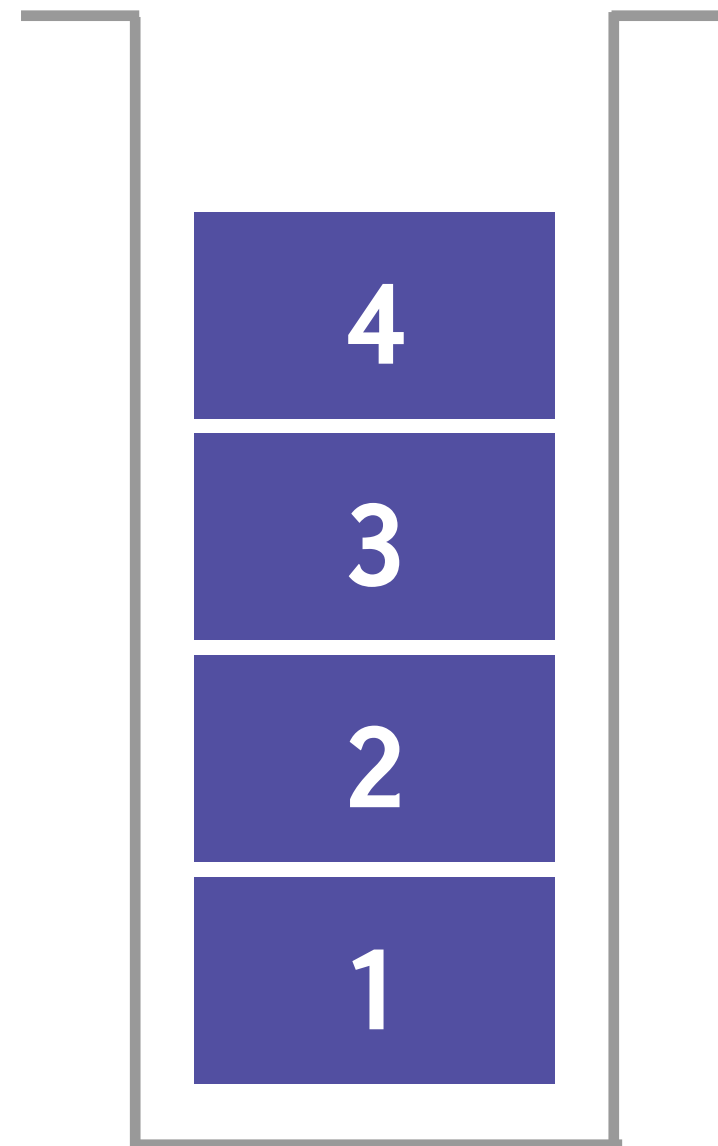
✓ 대표적인 자료구조의 예시

선형 구조 : 자료가 **순서**를 가지고 **연속**되어 있음

비선형 구조 : 선형 구조에 **해당하지 않는** 자료구조

01 스택, 큐의 개념

✓ 스택



Stack

Last In First Out

한쪽 끝에서만 자료를 넣고 뺄 수 있는 자료구조

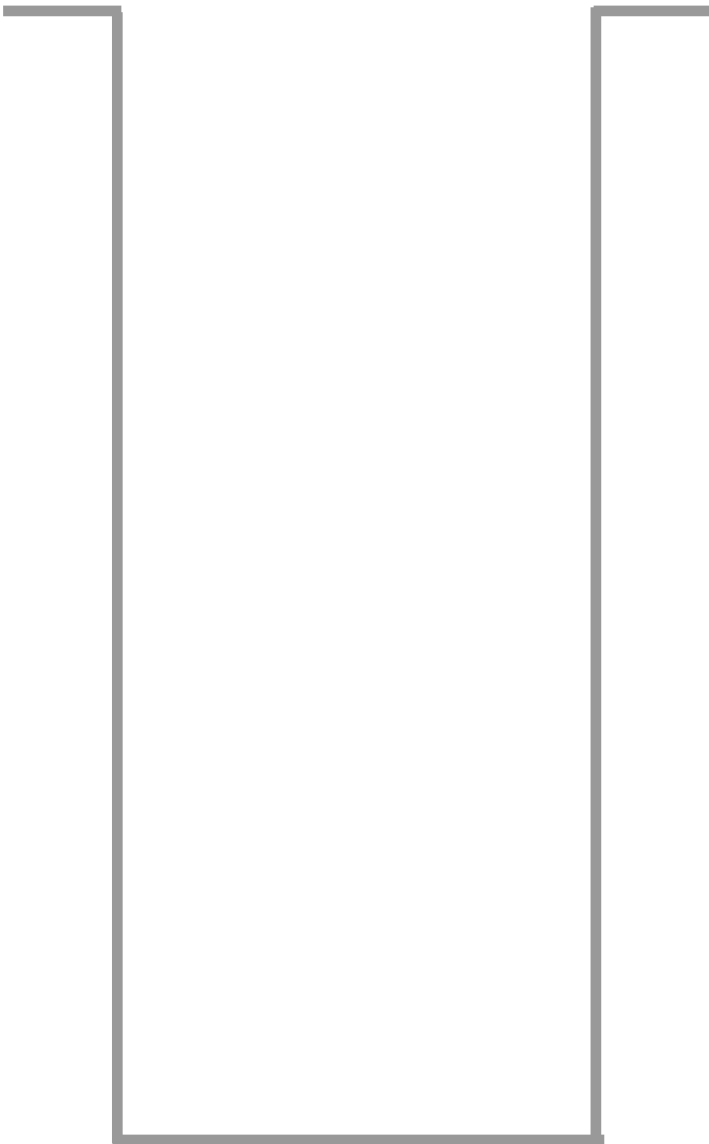
스택이 지원하는 연산 목록

- push : 스택에 자료를 넣는 연산
- pop : 스택에서 자료를 빼는 연산
- top : 스택의 가장 위에 있는 자료를 반환하는 연산
- empty : 스택이 비어있는지 여부를 반환하는 연산

`/* elice */`

01 스택, 큐의 개념

✓ 스택



push 1

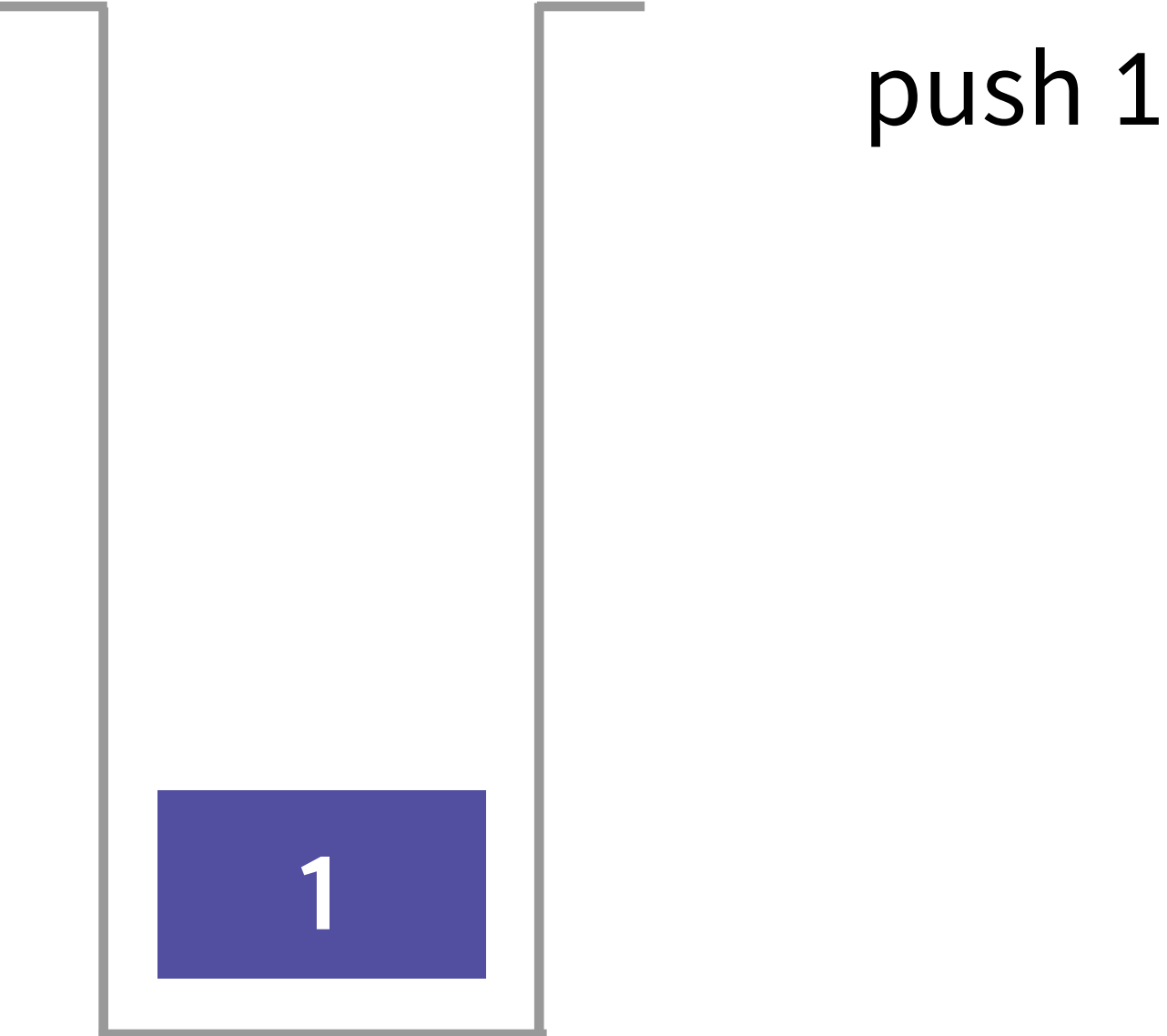
Stack

Last In First Out

01 스택, 큐의 개념

✓

스택

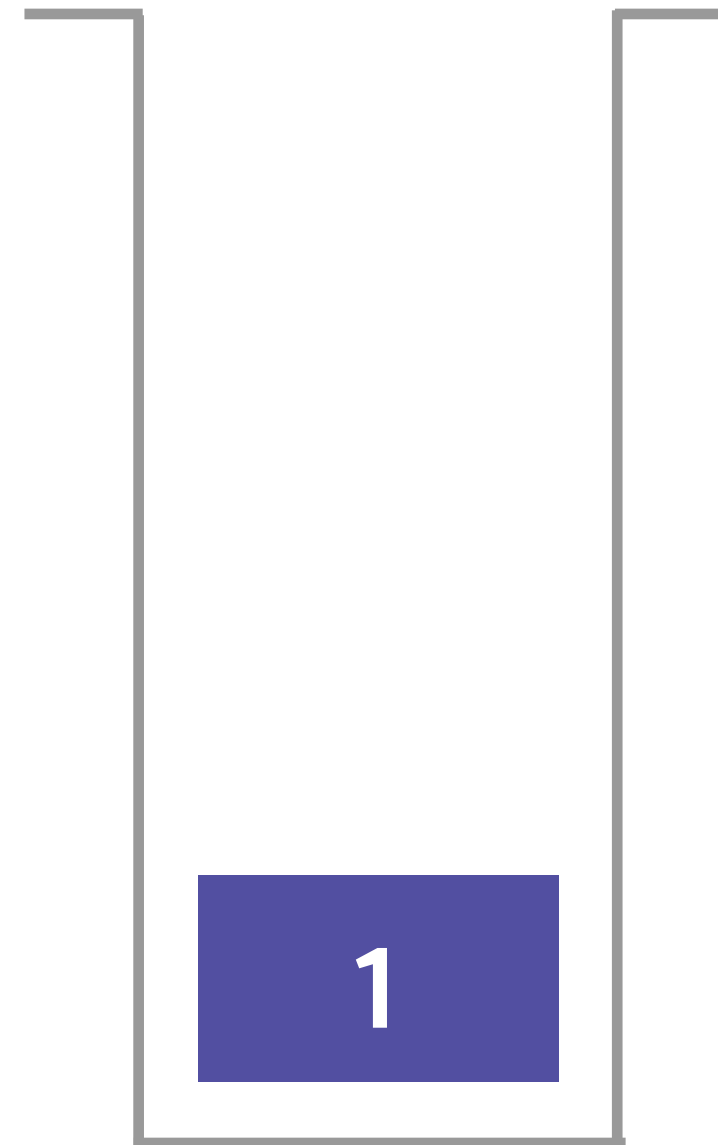


Stack

Last In First Out

01 스택, 큐의 개념

✓ 스택



push 1
push 2

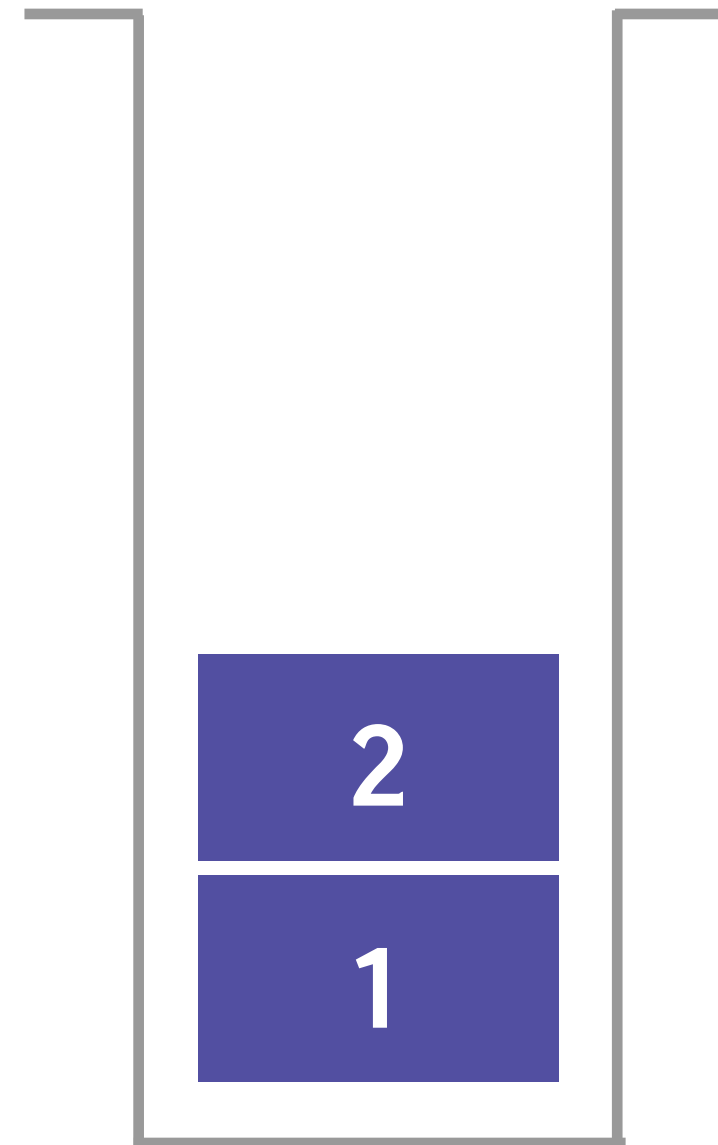
Stack

Last In First Out

/ elice */*

01 스택, 큐의 개념

✓ 스택



push 1
push 2

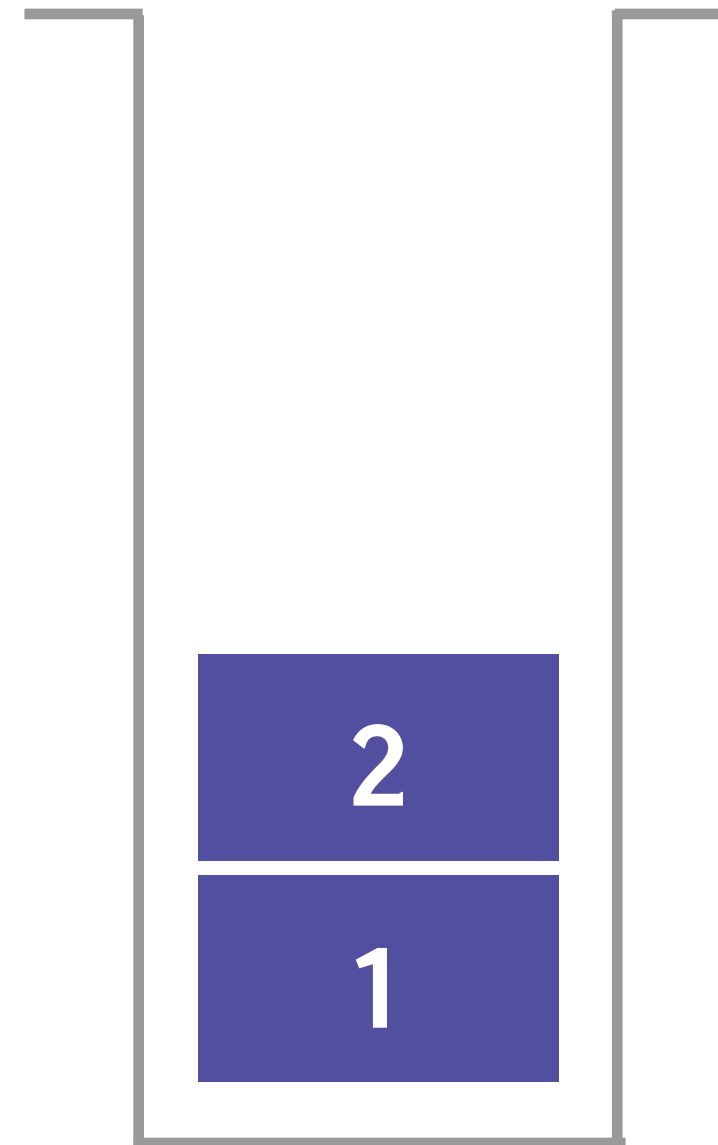
Stack

Last In First Out

`/* elice */`

01 스택, 큐의 개념

✓ 스택



push 1
push 2
pop

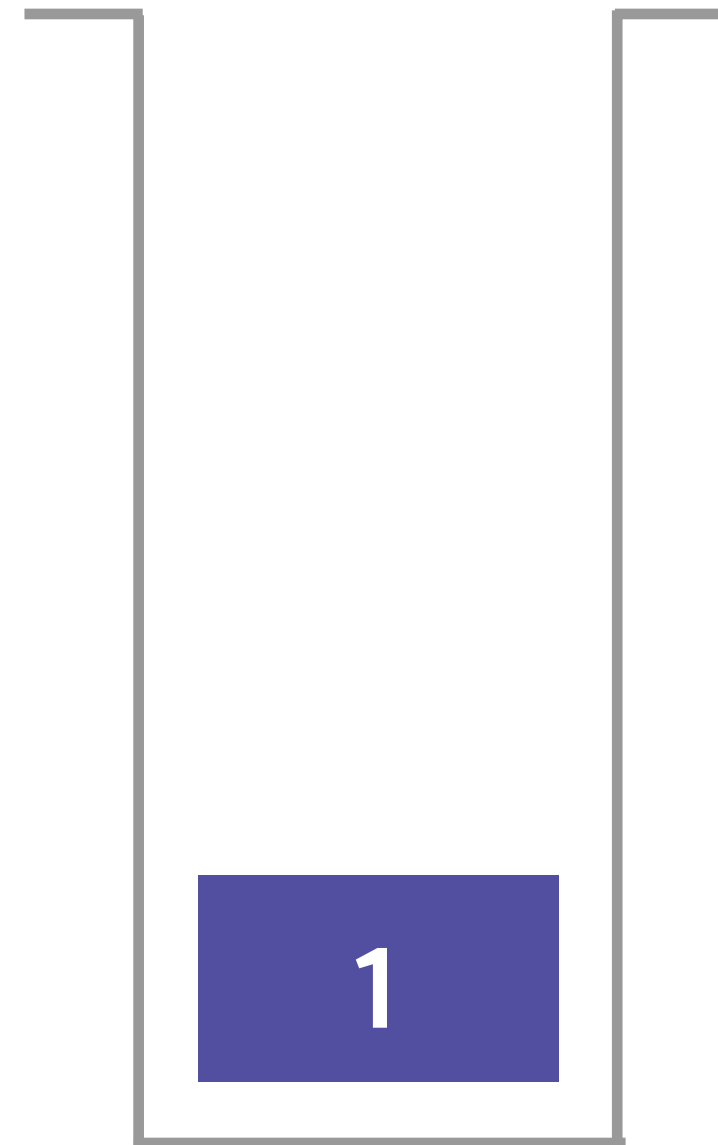
Stack

Last In First Out

`/* elice */`

01 스택, 큐의 개념

✓ 스택



push 1
push 2
pop

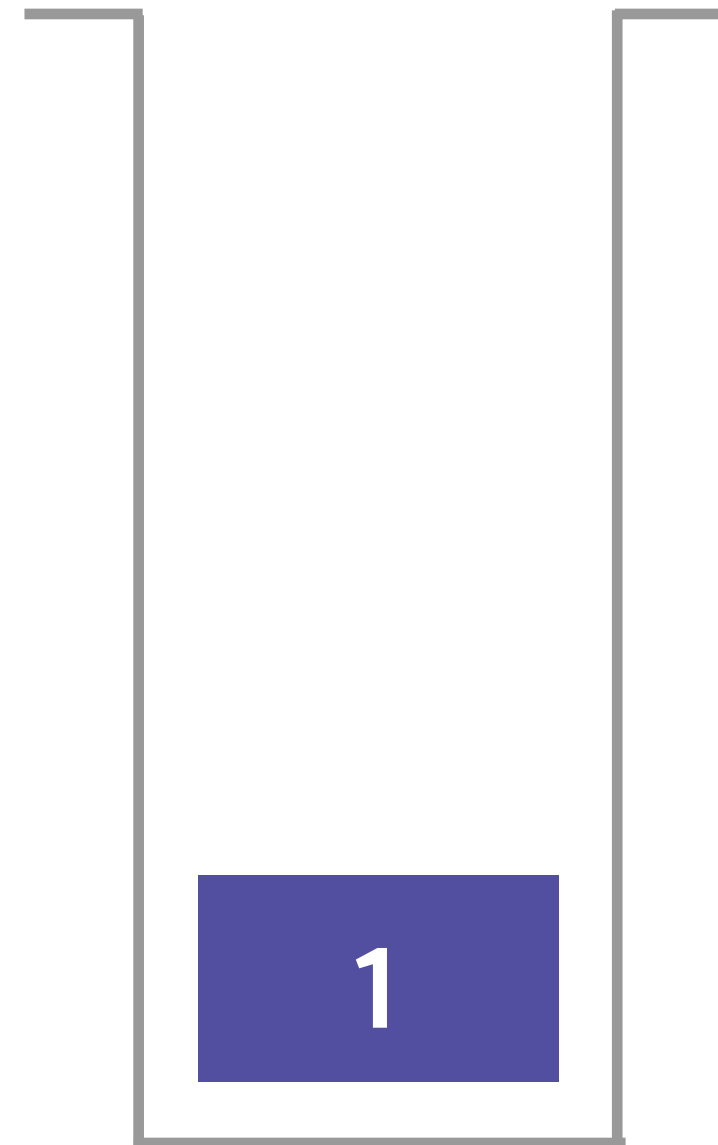
Stack

Last In First Out

`/* elice */`

01 스택, 큐의 개념

✓ 스택



push 1
push 2
pop
push 3

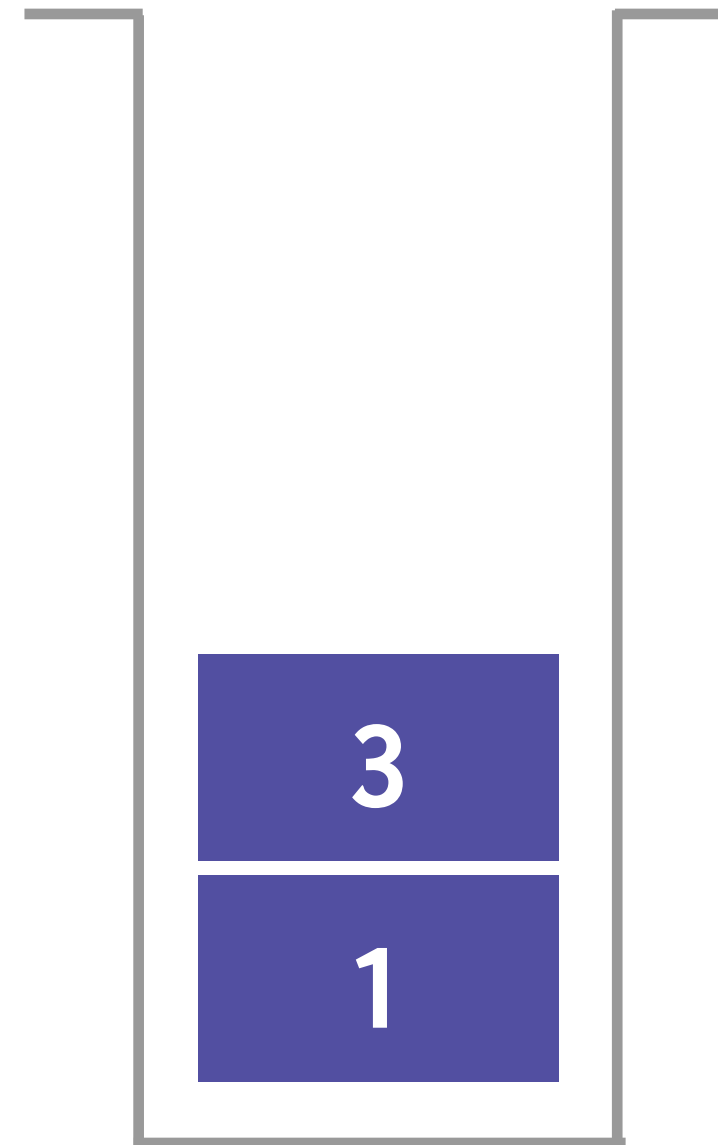
Stack

Last In First Out

/ elice */*

01 스택, 큐의 개념

✓ 스택



push 1
push 2
pop
push 3

Stack

Last In First Out

/ elice */*

01 스택, 큐의 개념

✓ 스택

나중에 들어간 자료가 **먼저** 출력되기 때문에

Last In First Out(LIFO) 자료구조라고도 한다.

`/* elice */`

01 스택, 큐의 개념

✓ 스택

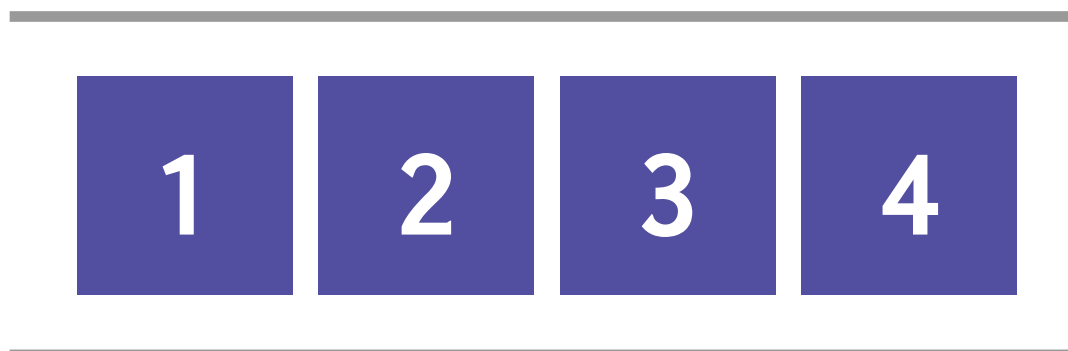
파이썬의 리스트는 append, pop 등의 연산을 지원하므로
리스트를 이용하여 스택을 구현할 수 있다.

01 스택, 큐의 개념

✓ 큐

입구와 출구가 **각각 한 쪽 끝에** 존재하는 자료구조

큐가 지원하는 **연산 목록**



- push : 큐에 자료를 **넣는** 연산
- pop : 큐에서 자료를 **빼는** 연산
- front : 큐의 **가장 앞에 있는 자료**를 반환하는 연산
- back : 큐의 **가장 뒤에 있는 자료**를 반환하는 연산
- empty : 큐가 **비어있는지** 여부를 반환하는 연산

Queue

First In First Out

`/* elice */`

01 스택, 큐의 개념

✓ 큐

push 1



Queue
First In First Out

01 스택, 큐의 개념

✓ 큐

push 1



Queue

First In First Out

`/* elice */`

01 스택, 큐의 개념

✓ 큐

push 1
push 2



Queue

First In First Out

/ elice */*

01 스택, 큐의 개념

✓ 큐

push 1
push 2



Queue

First In First Out

`/* elice */`

01 스택, 큐의 개념

✓ 큐

push 1
push 2
pop



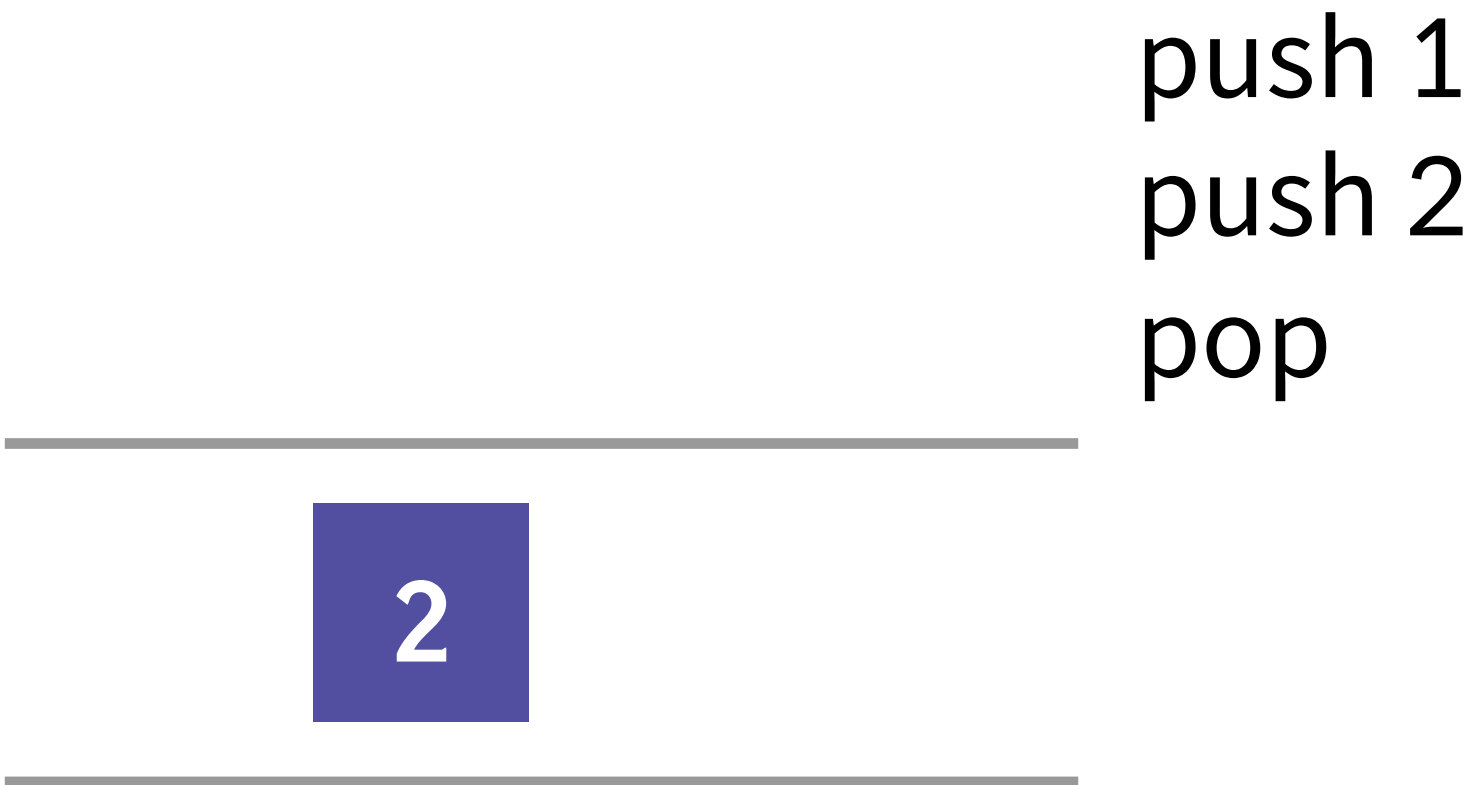
Queue

First In First Out

`/* elice */`

01 스택, 큐의 개념

✓ 큐



Queue
First In First Out

`/* elice */`

01 스택, 큐의 개념

✓ 큐



push 1
push 2
pop
push 3

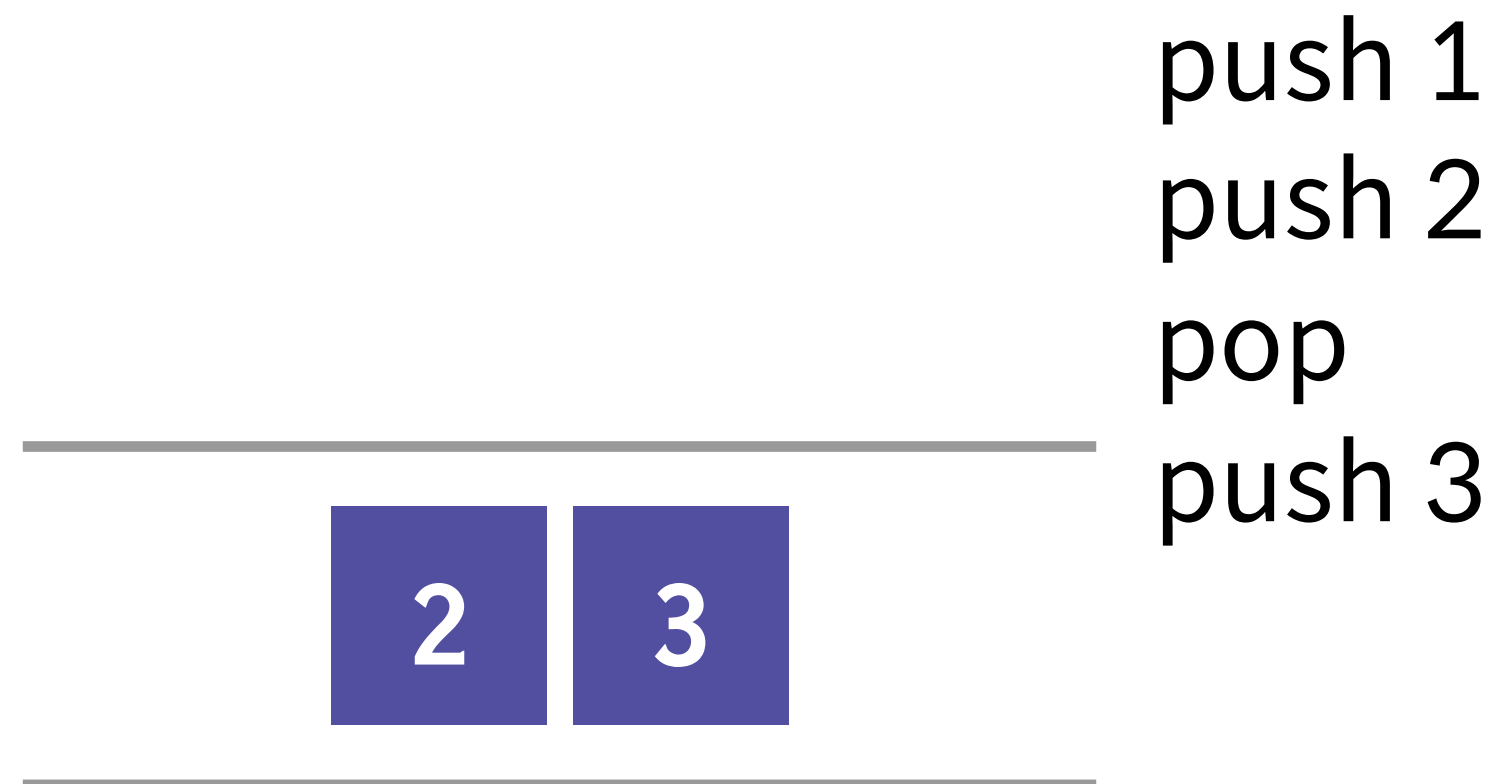
Queue

First In First Out

`/* elice */`

01 스택, 큐의 개념

✓ 큐



Queue

First In First Out

/ elice */*

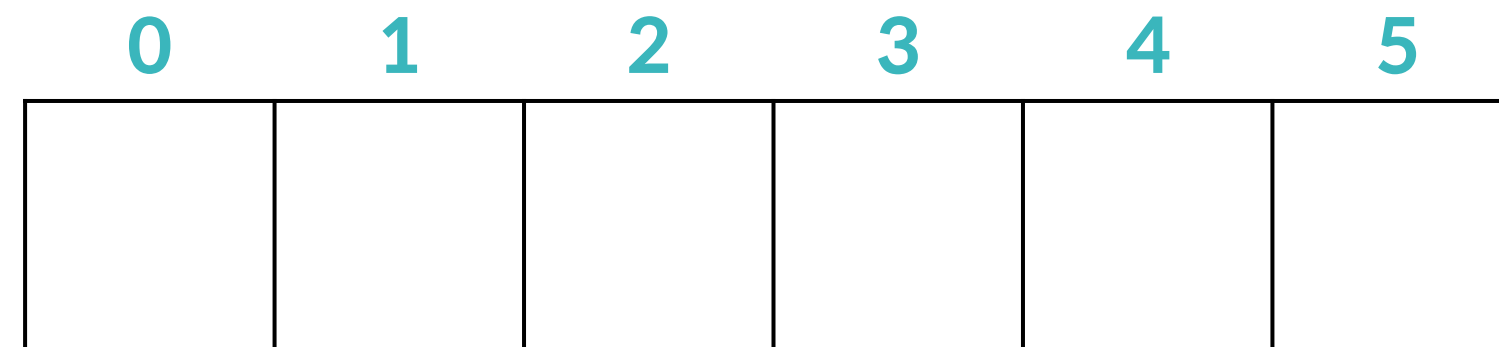
01 스택, 큐의 개념

✓ 큐

선형 자료구조이기 때문에 **배열**을 이용하여 구현이 가능하기는 하지만
여러 문제가 존재한다.

01 스택, 큐의 개념

✓ 큐



배열로 구현된 크기가 6인 큐가 있다.

head와 **rear**는 인덱스를 나타내며, 큐에 포함되어있는 내용은

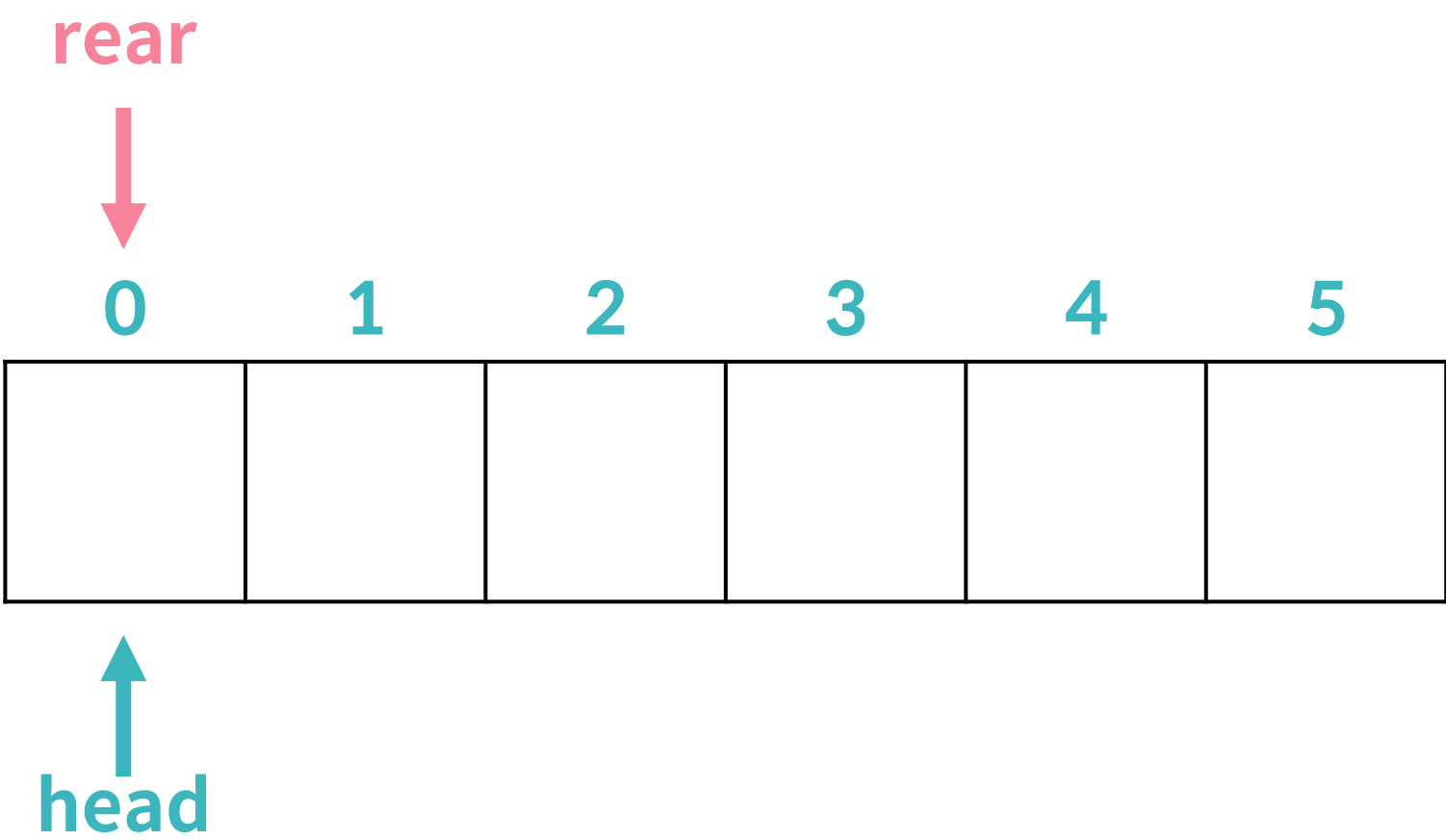
head 이상 rear 미만의 인덱스들이다.

`/* elice */`

01 스택, 큐의 개념

✓ 큐

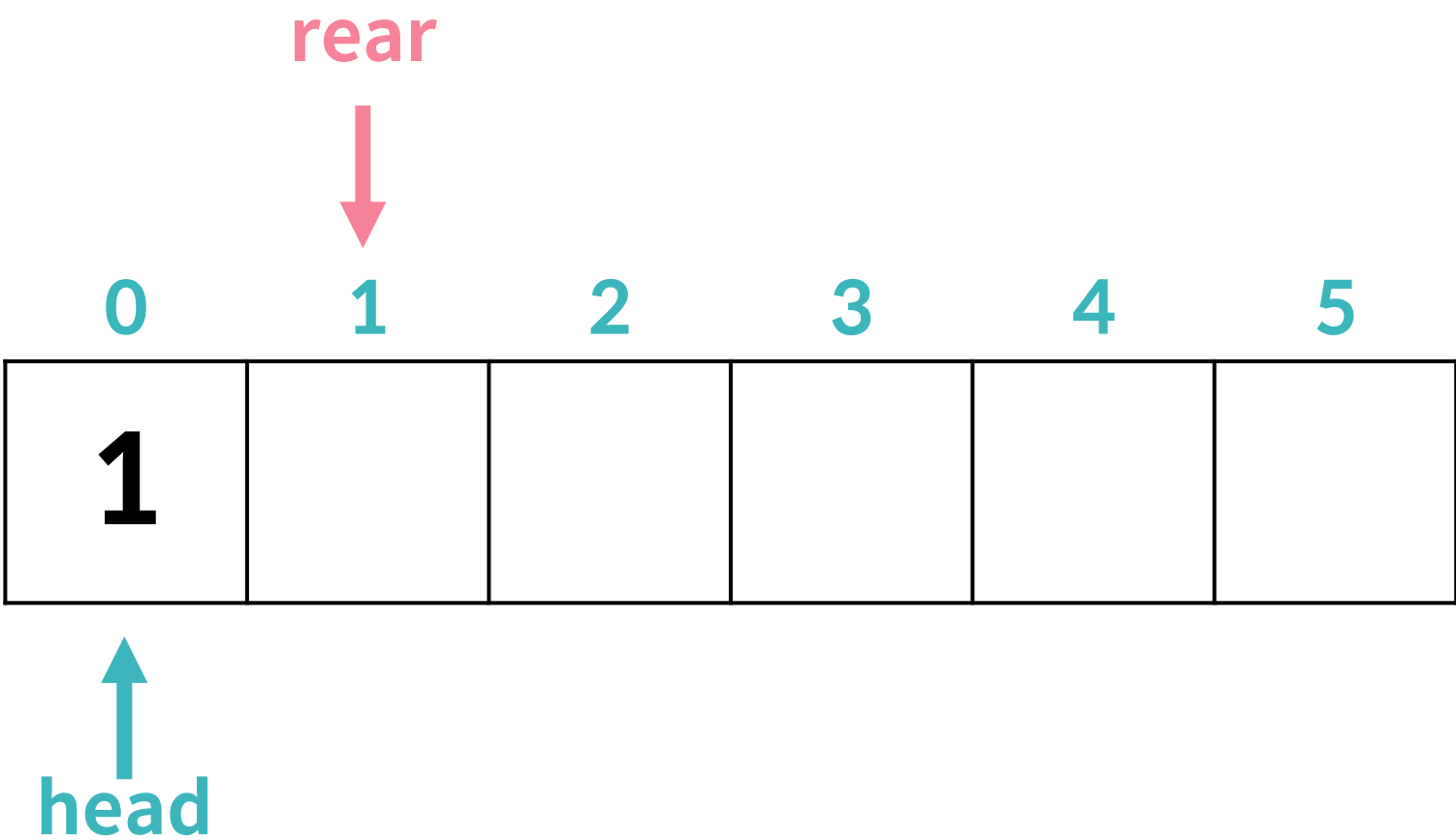
push 1



01 스택, 큐의 개념

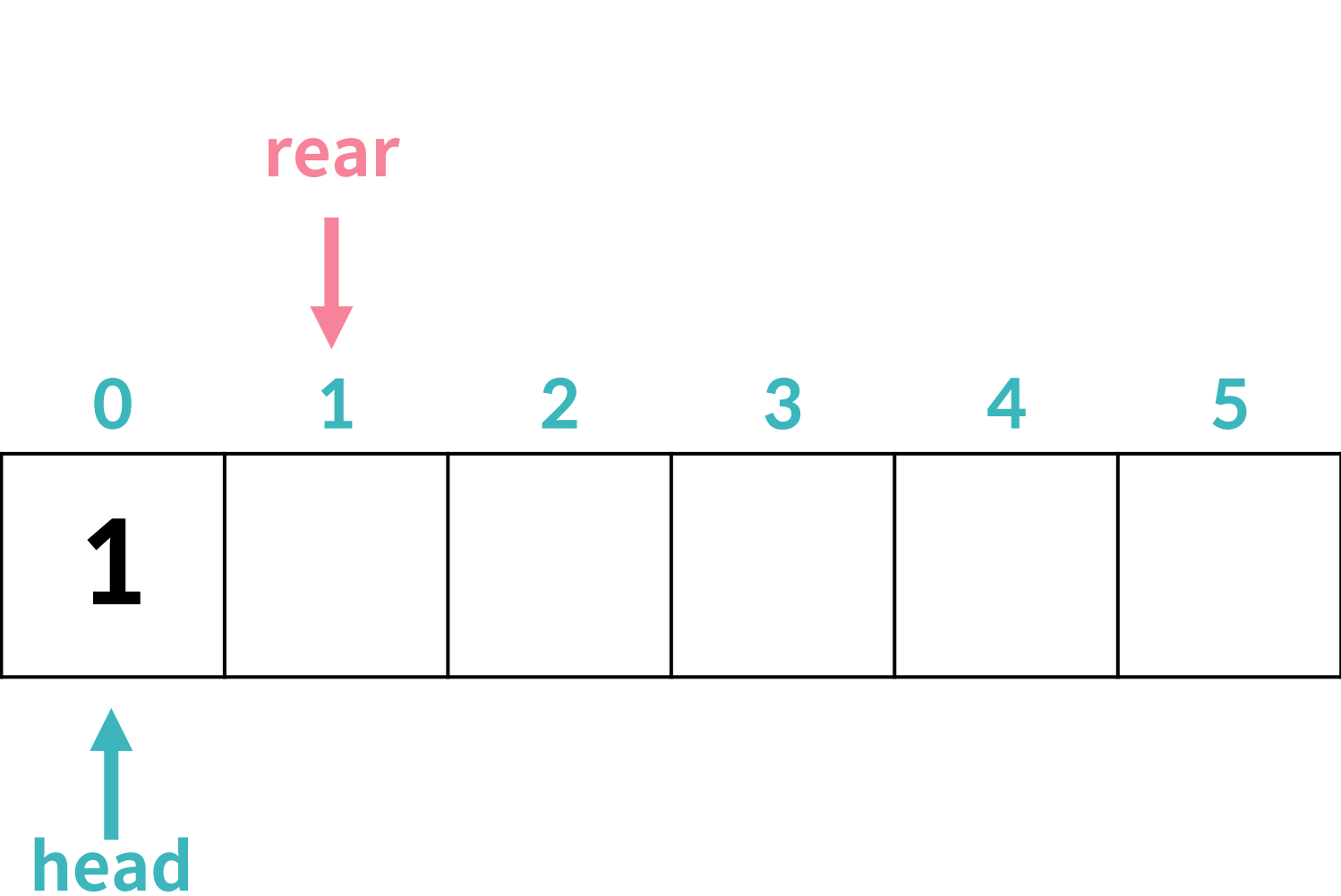
✓ 큐

push 1



01 스택, 큐의 개념

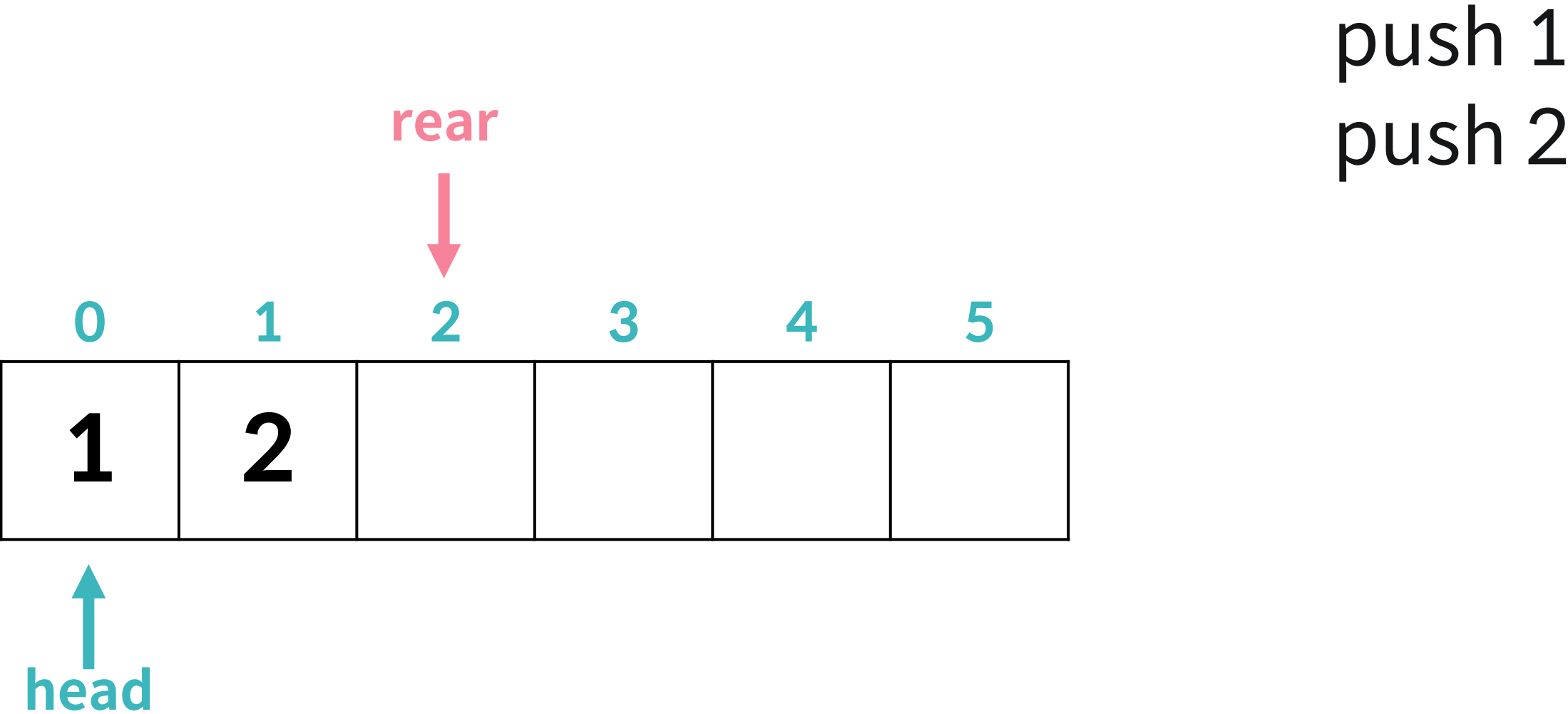
✓ 큐



push 1
push 2

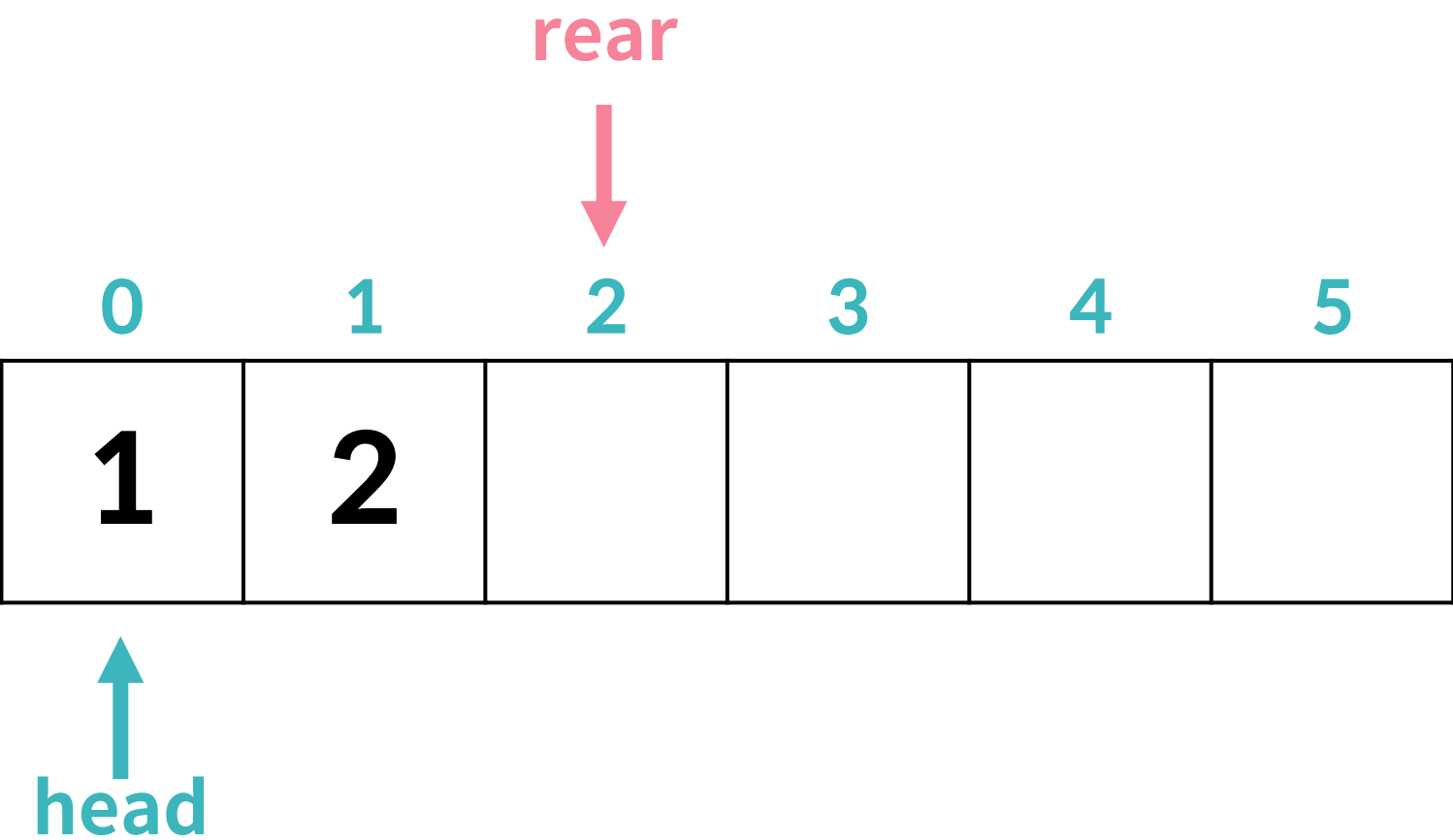
01 스택, 큐의 개념

✓ 큐



01 스택, 큐의 개념

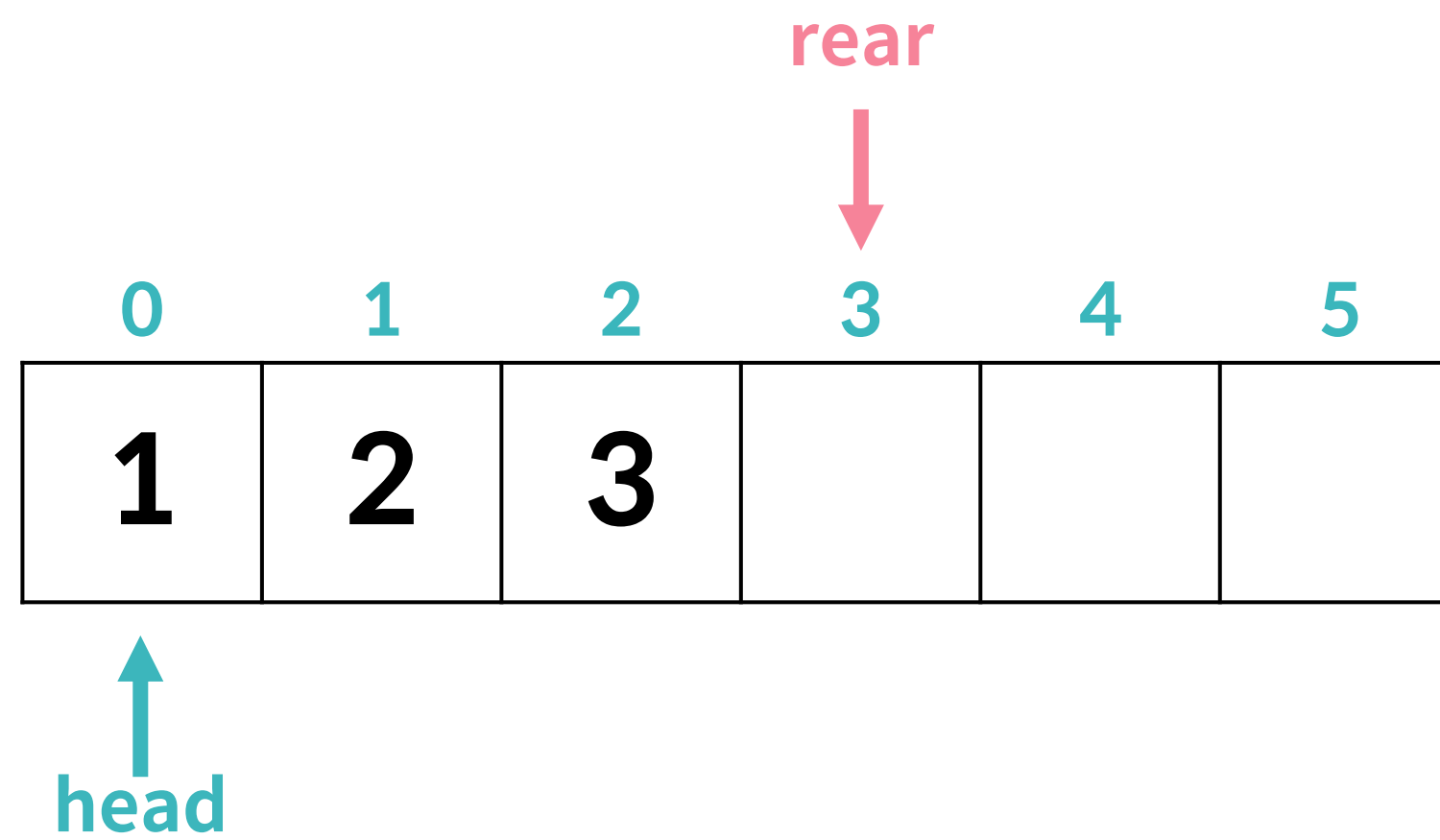
✓ 큐



push 1
push 2
push 3

01 스택, 큐의 개념

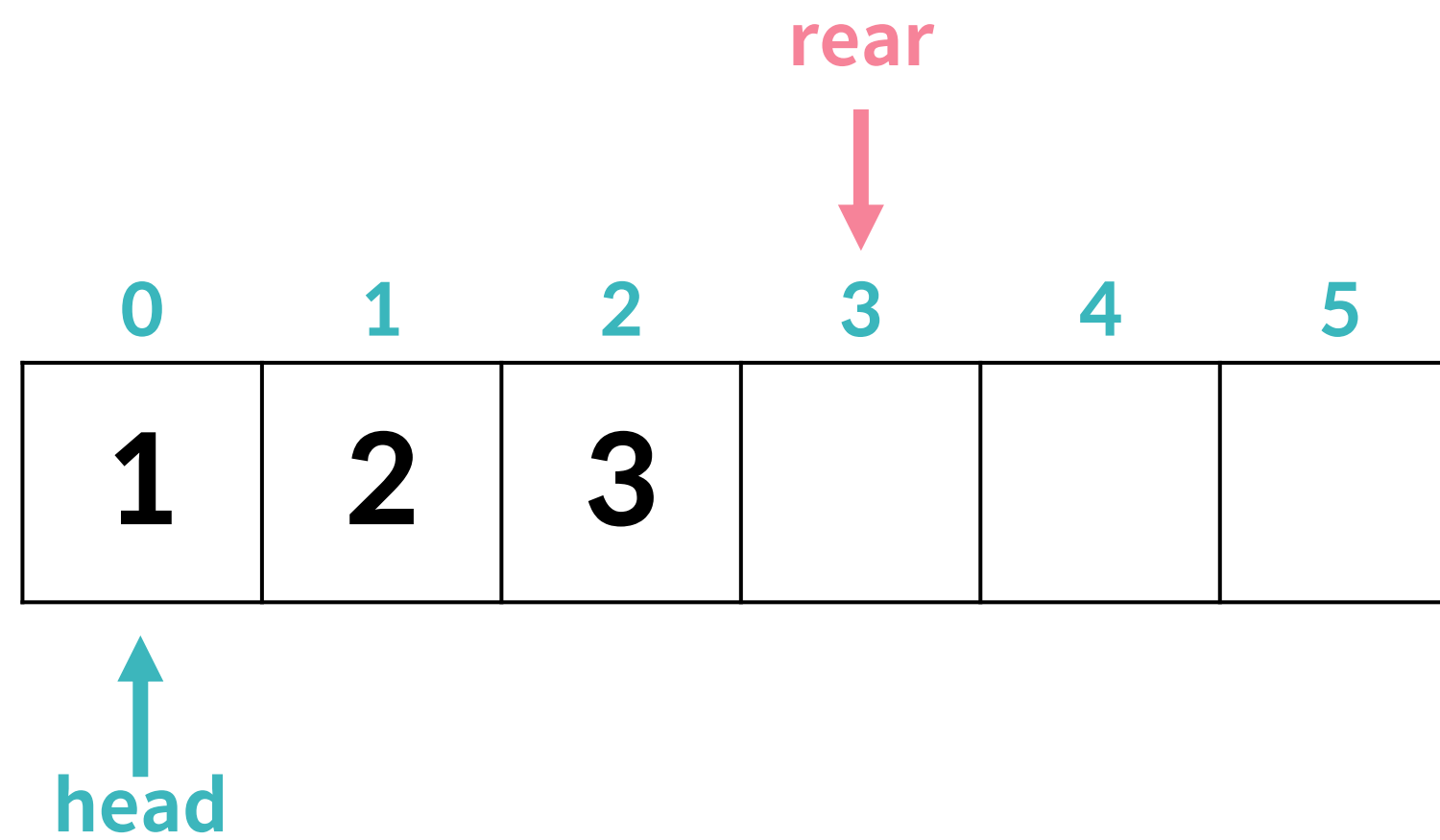
✓ 큐



push 1
push 2
push 3

01 스택, 큐의 개념

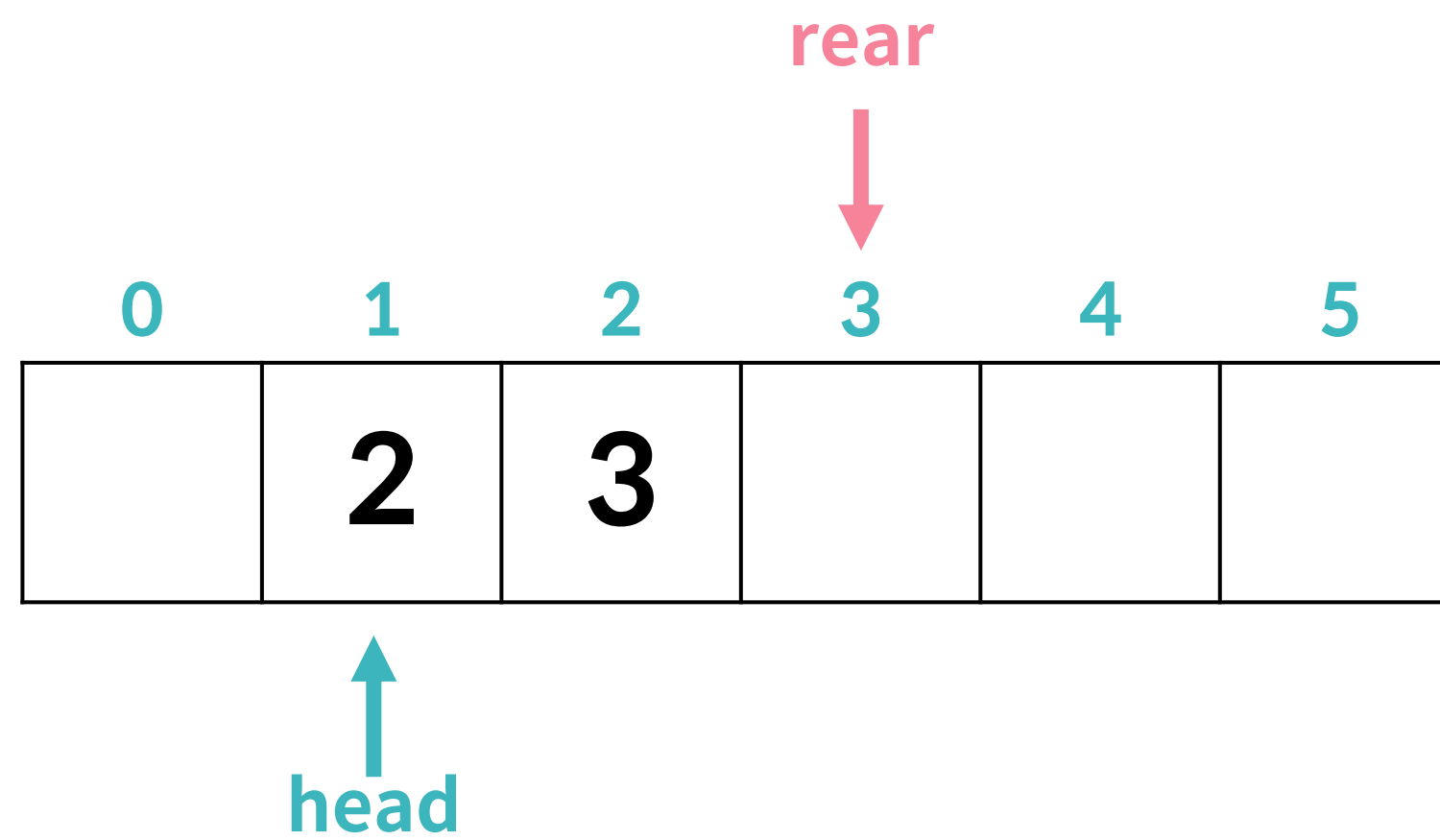
✓ 큐



push 1
push 2
push 3
pop

01 스택, 큐의 개념

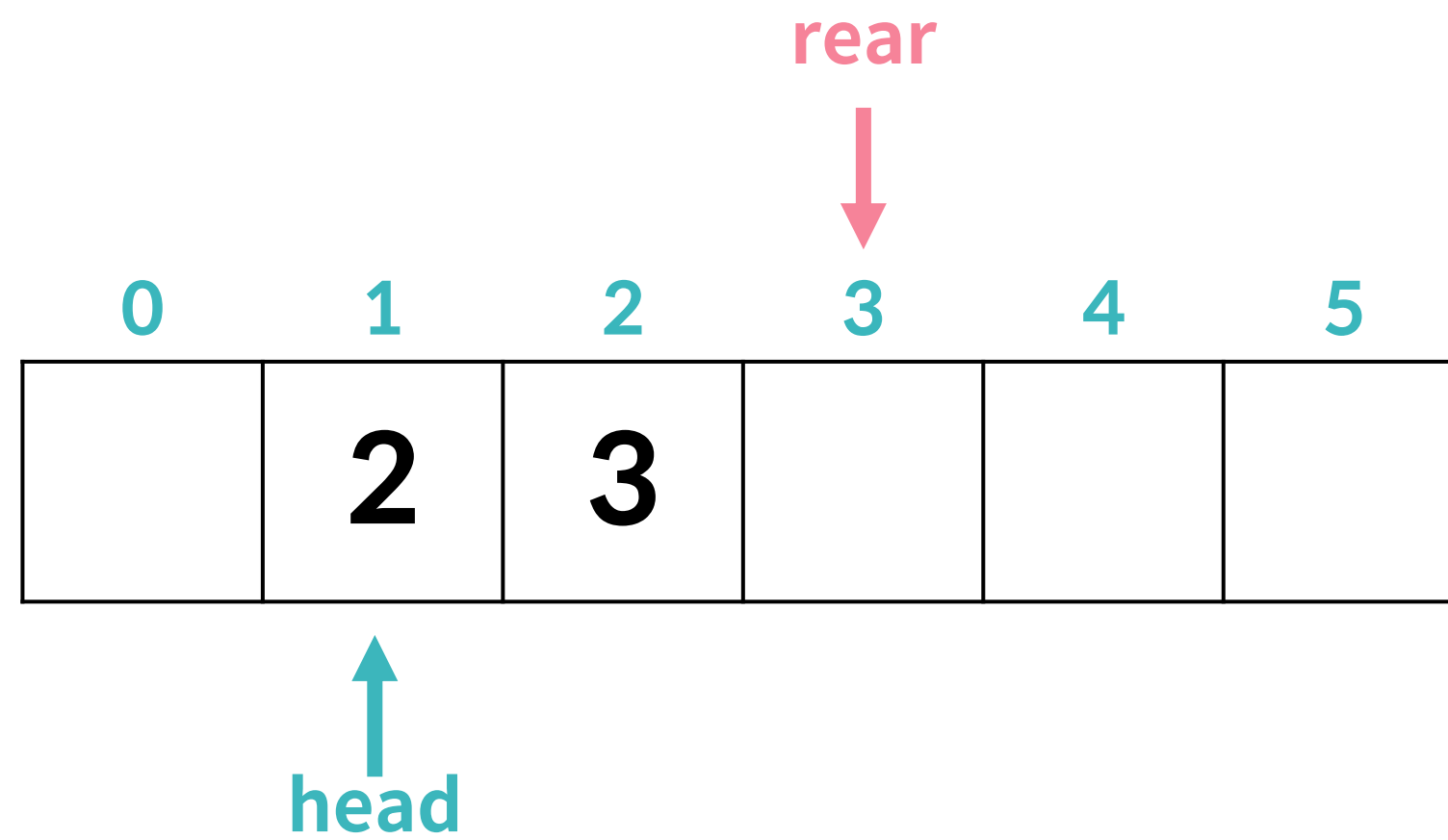
✓ 큐



push 1
push 2
push 3
pop

01 스택, 큐의 개념

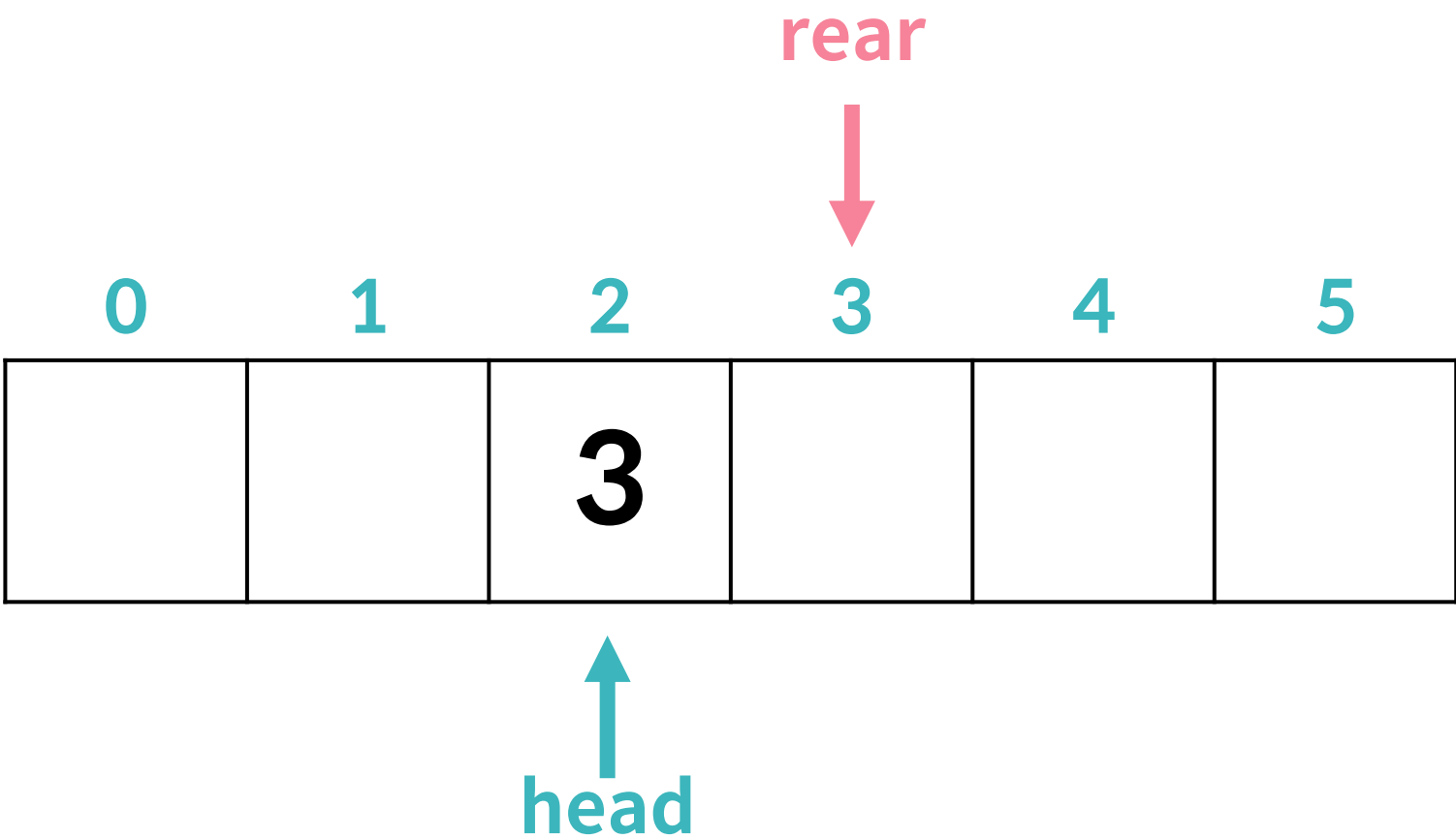
✓ 큐



push 1
push 2
push 3
pop
pop

01 스택, 큐의 개념

✓ 큐

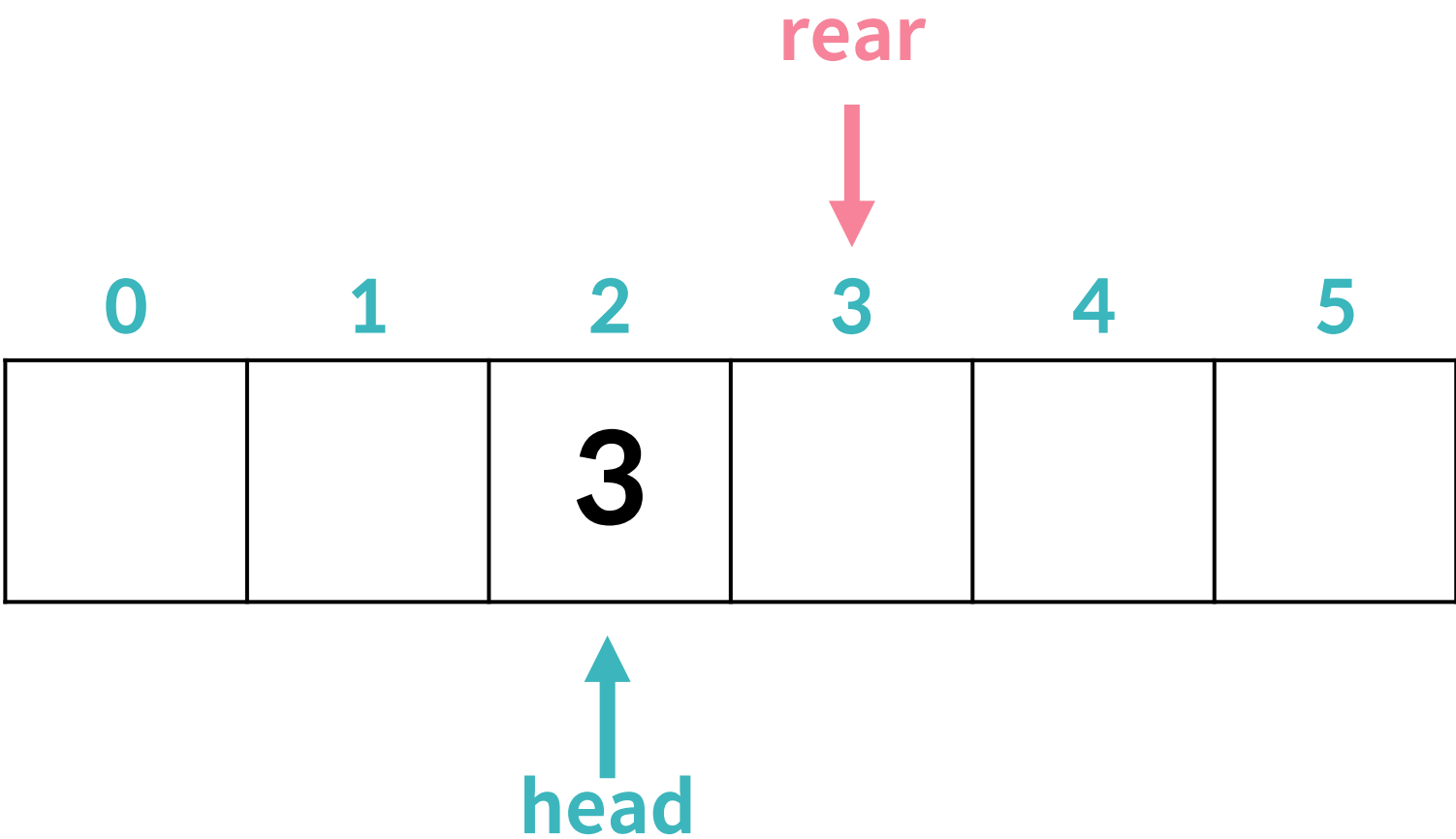


push 1
push 2
push 3
pop
pop

01 스택, 큐의 개념

✓ 큐

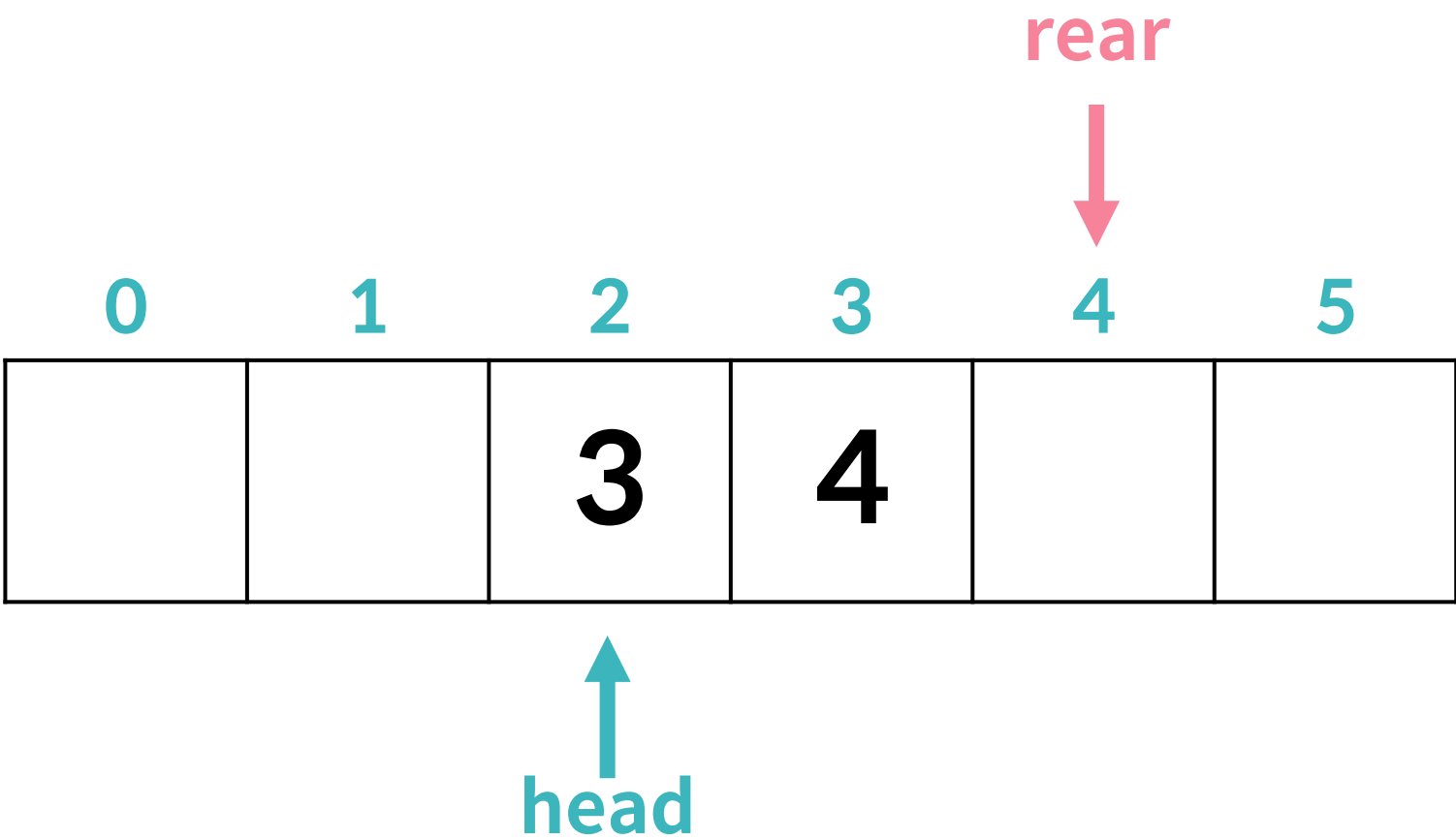
push 4



01 스택, 큐의 개념

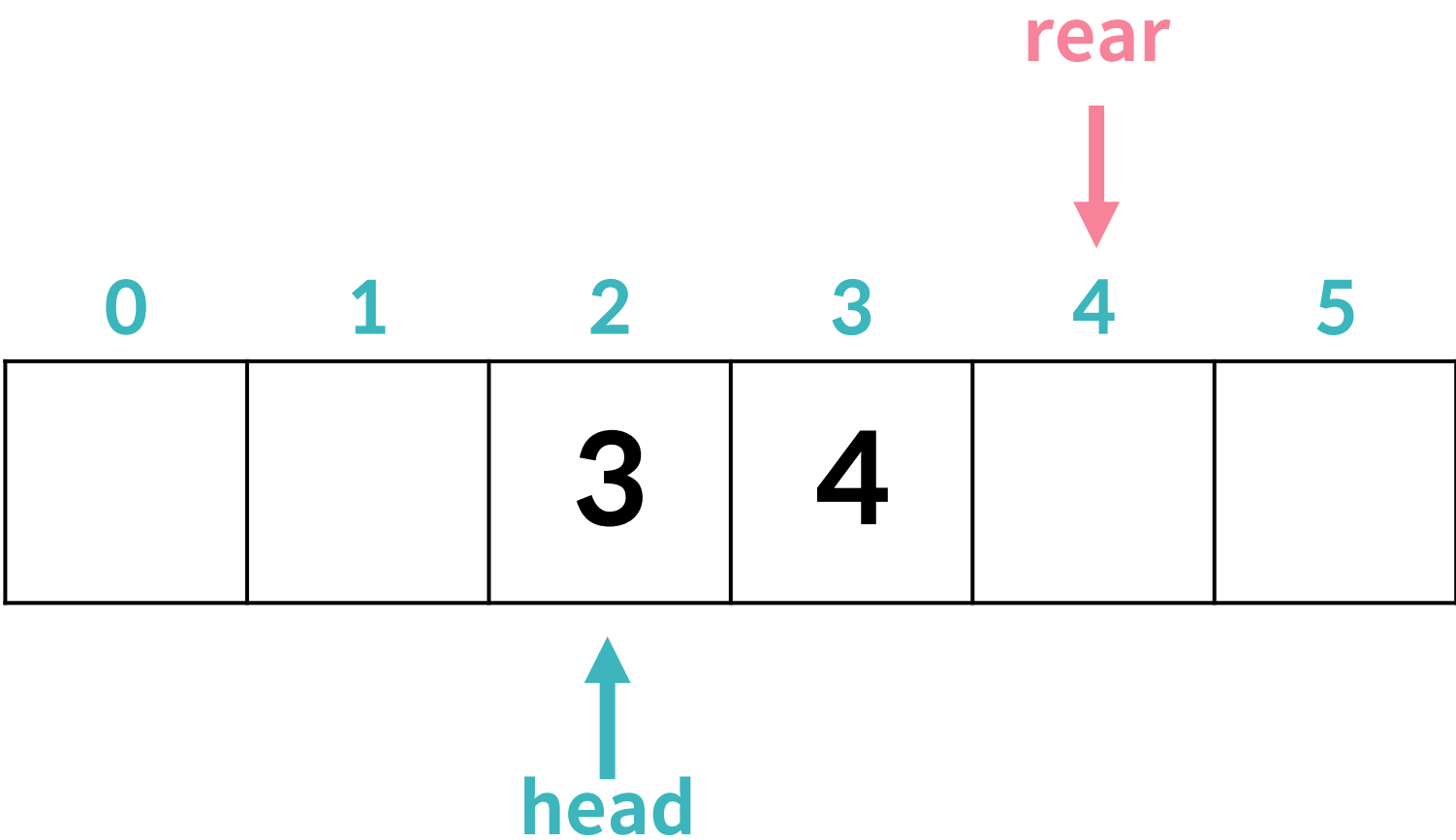
✓ 큐

push 4



01 스택, 큐의 개념

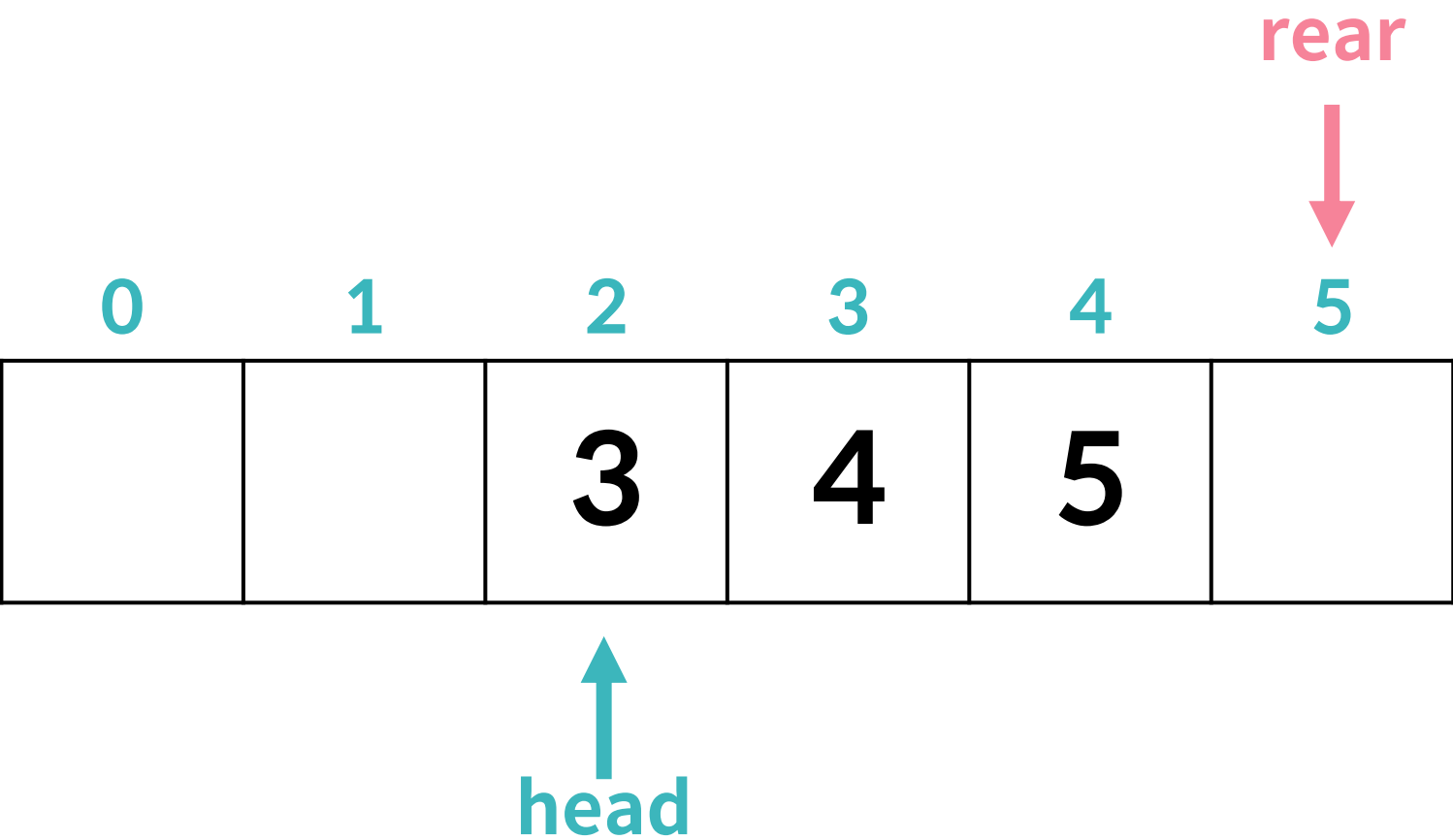
✓ 큐



push 4
push 5

01 스택, 큐의 개념

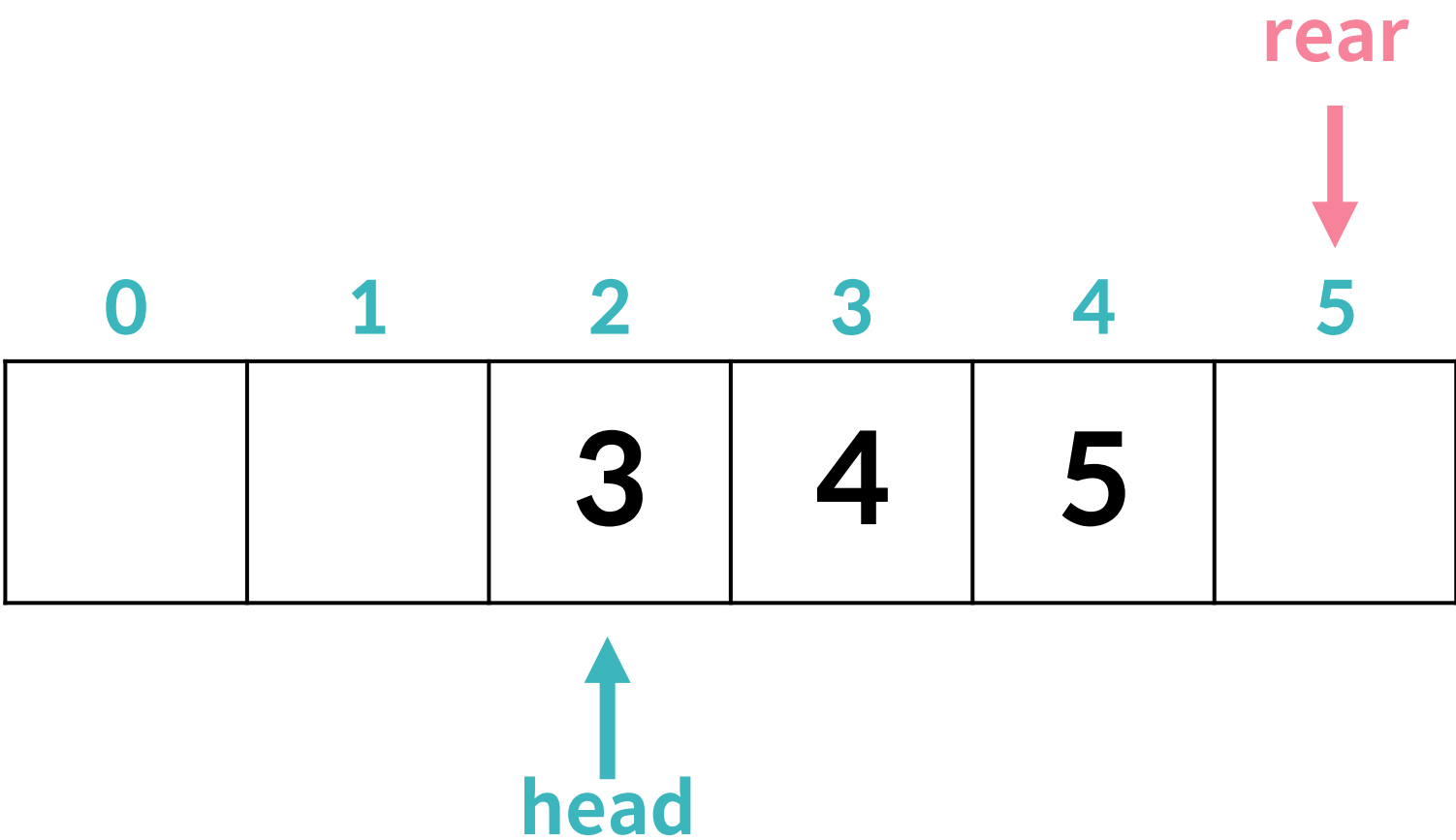
✓ 큐



push 4
push 5

01 스택, 큐의 개념

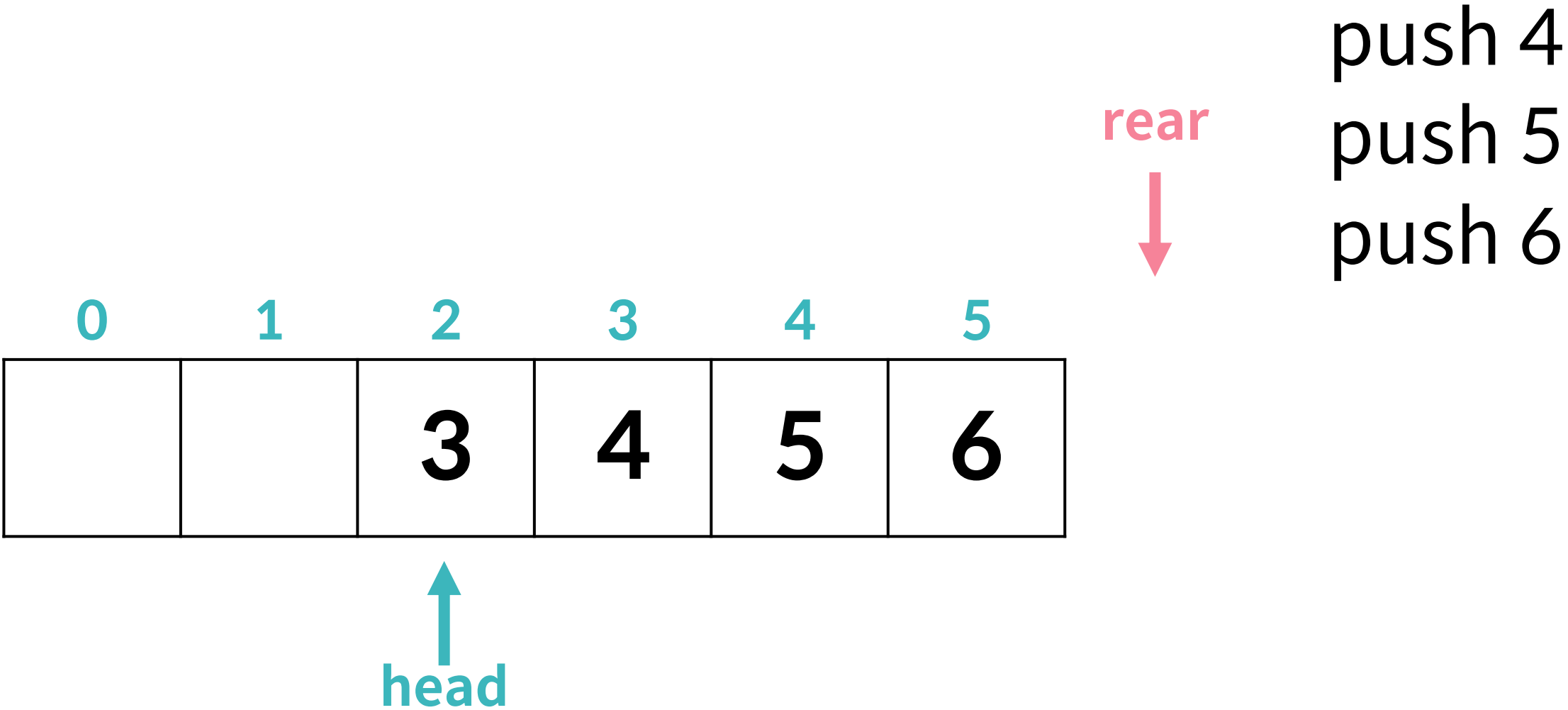
✓ 큐



push 4
push 5
push 6

01 스택, 큐의 개념

✓ 큐



01 스택, 큐의 개념

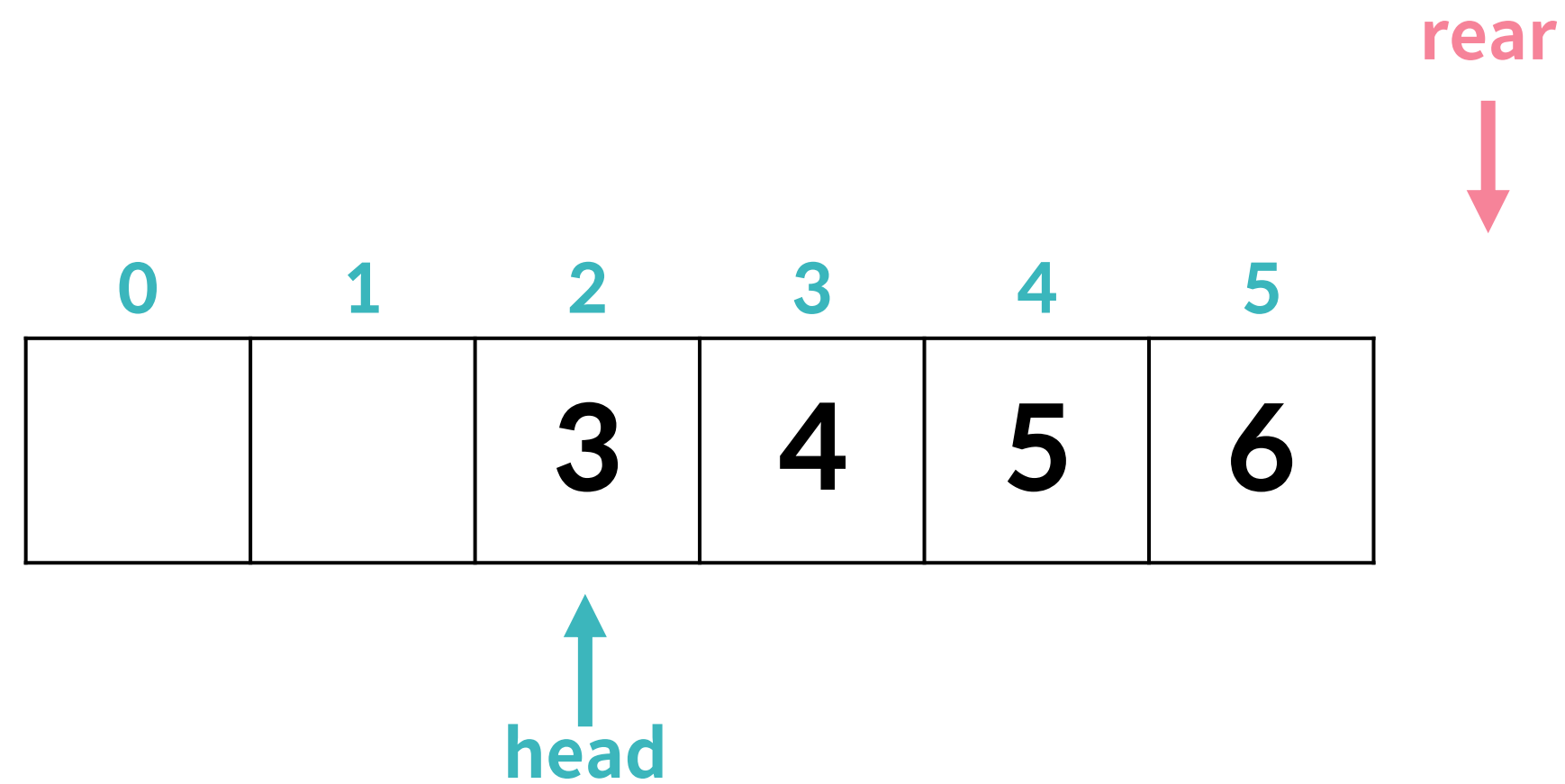
✓ 큐

push - **rear** 위치에 **자료를 추가**하고 rear를 **1 증가**시킴

pop - **head** 위치의 **자료를 제거**하고 head를 **1 증가**시킴

01 스택, 큐의 개념

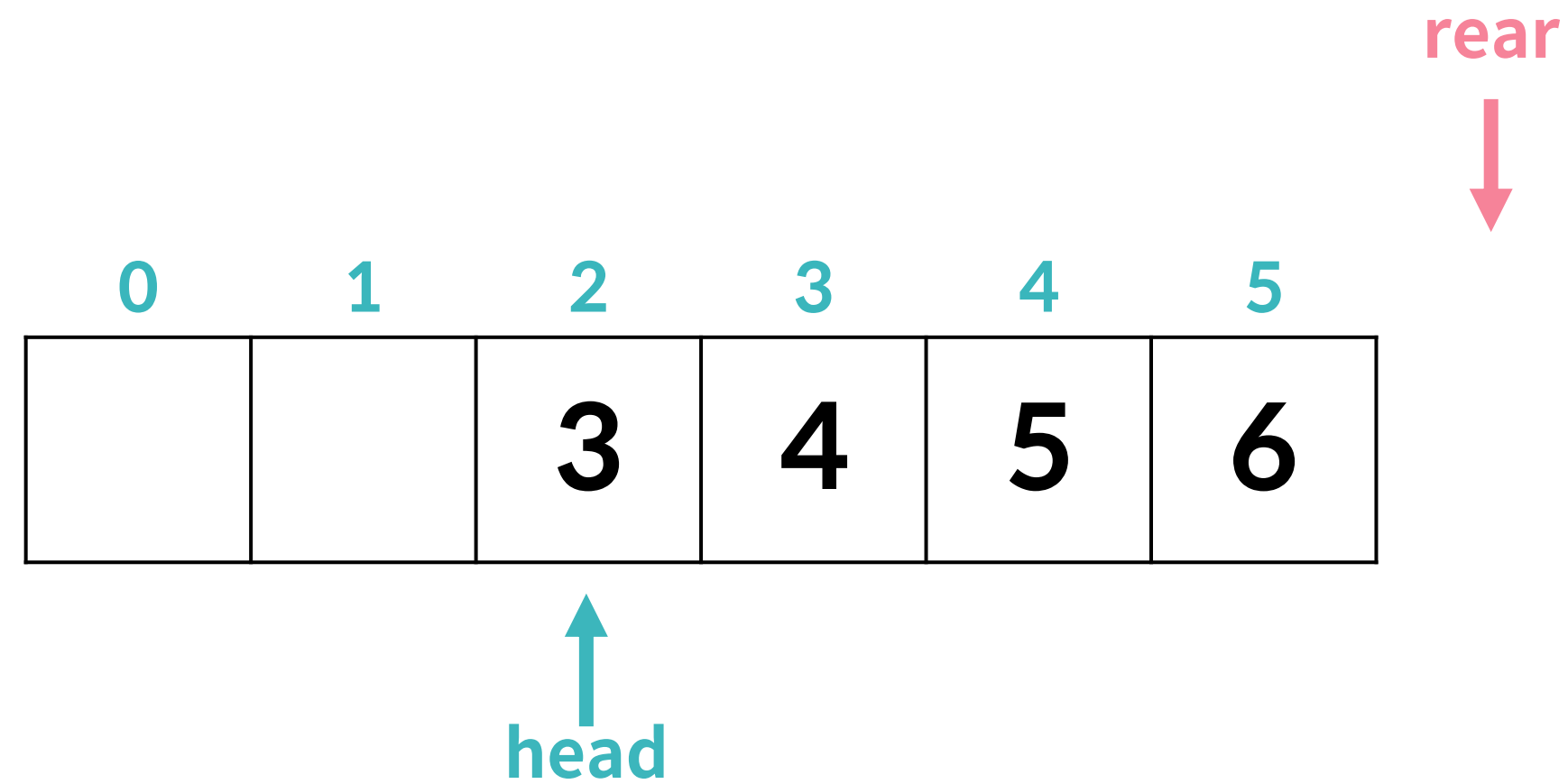
✓ 큐



위 경우는 크기가 6인 큐에 **자료가 4개** 들어있고,
빈 공간이 2개 있는 상태이다.

01 스택, 큐의 개념

✓ 큐



그러나, 위의 경우에는 **rear**가 가리키는 인덱스가 **6**이기 때문에 자료를 추가할 수 없다.
인덱스가 배열의 크기를 **초과**했기 때문이다.

01 스택, 큐의 개념

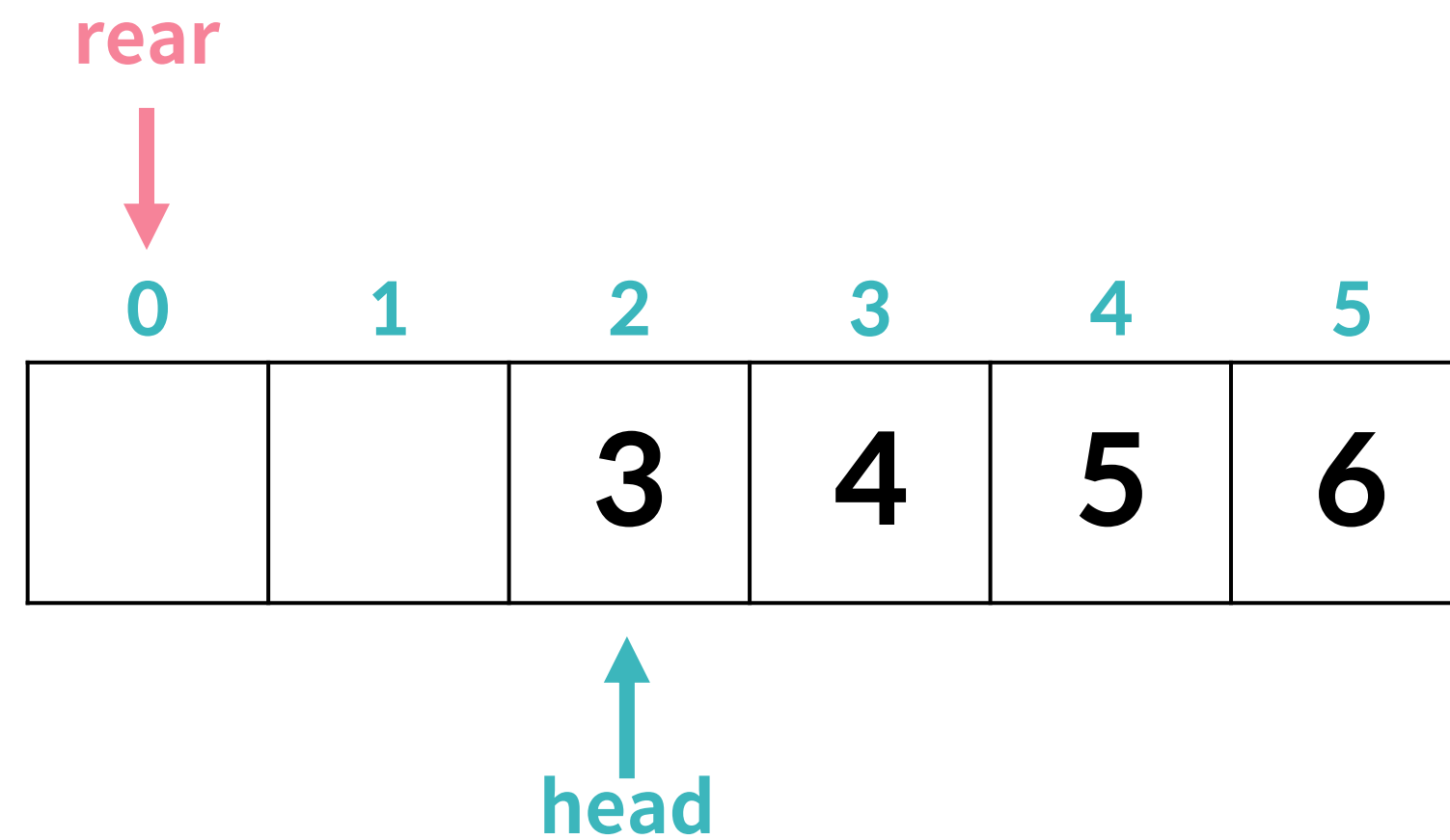
✓ 원형 큐

배열로 큐를 구현하였을 때의 문제점을 보완하기 위한 자료구조가 있다.

바로 '원형 큐'이다.

01 스택, 큐의 개념

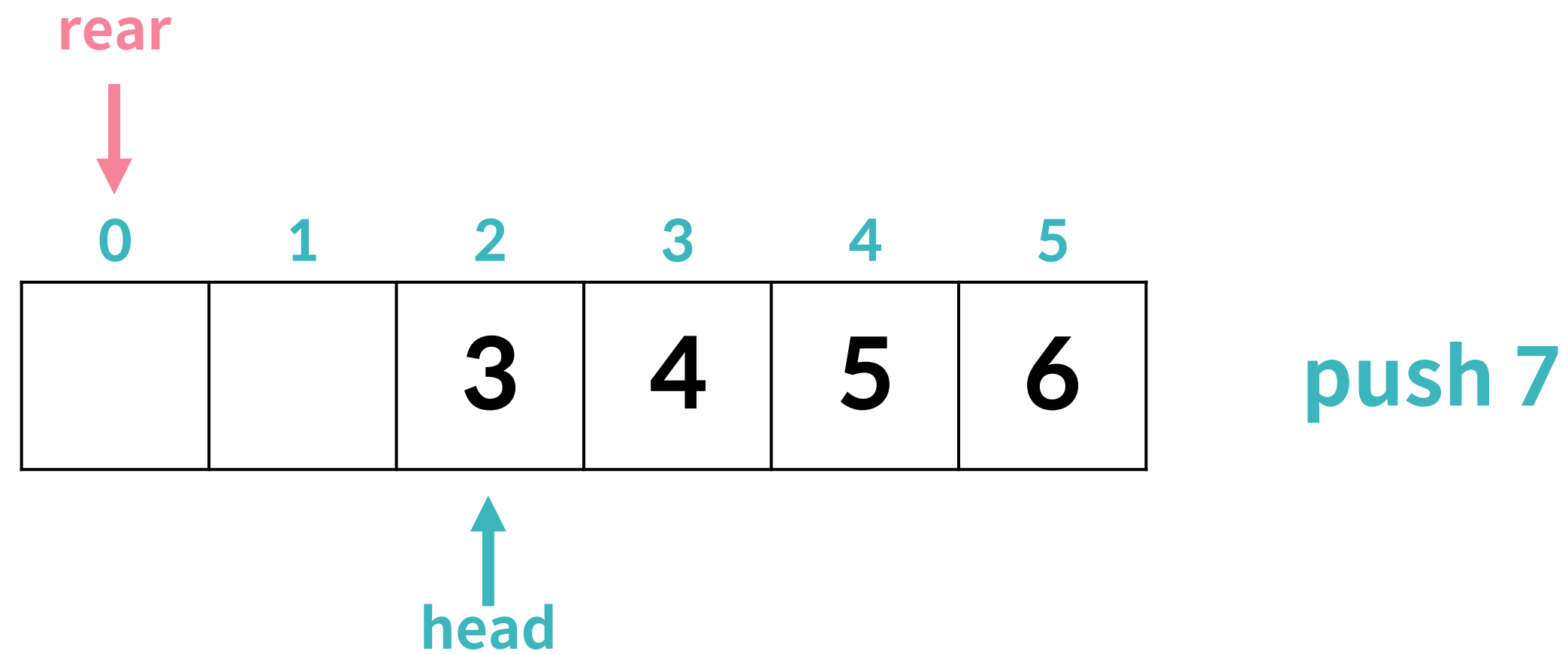
✓ 원형 큐



원형 큐의 경우, rear나 head가 큐의 끝에 도달하면
큐의 맨 앞으로 보내어 문제를 해결한다.

01 스택, 큐의 개념

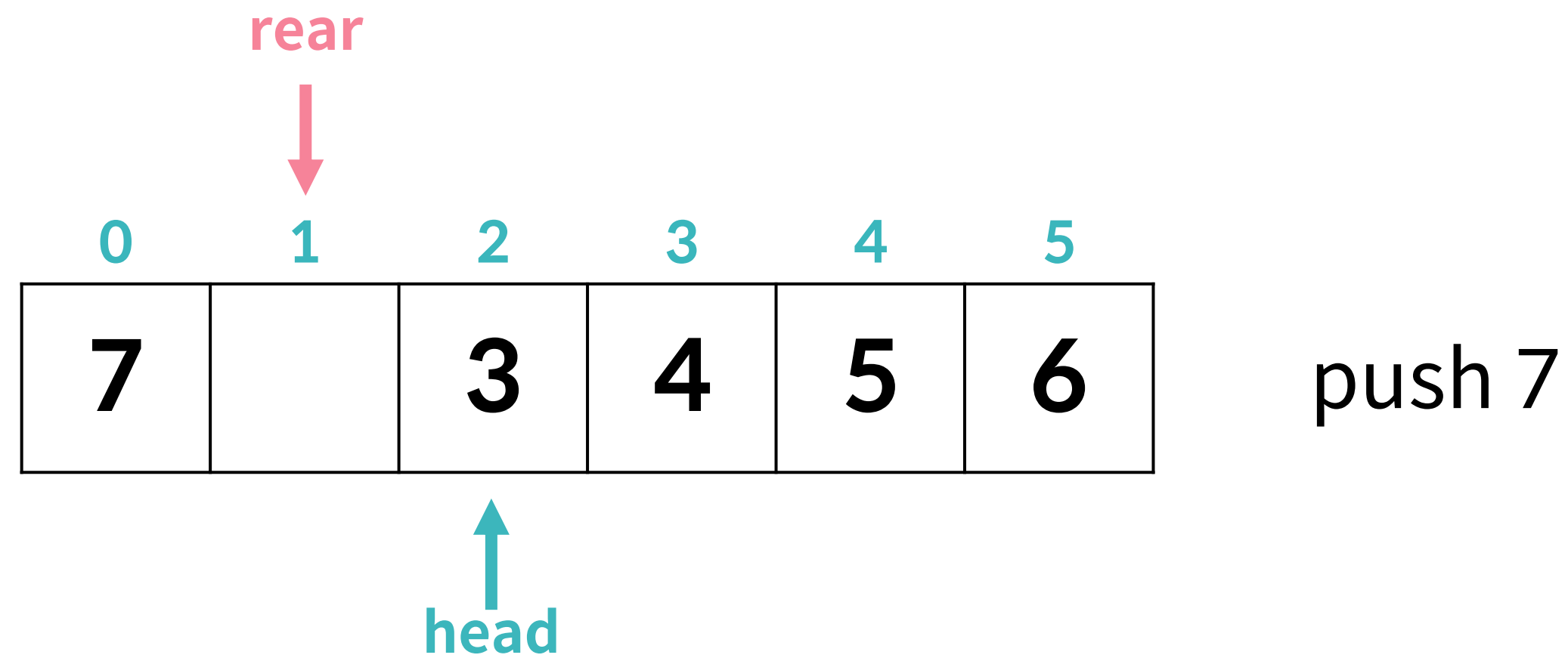
✓ 원형 큐



원형 큐의 경우, rear나 head가 큐의 끝에 도달하면
큐의 맨 앞으로 보내어 문제를 해결한다.

01 스택, 큐의 개념

✓ 원형 큐



원형 큐의 경우, rear나 head가 큐의 끝에 도달하면
큐의 맨 앞으로 보내어 문제를 해결한다.

01 스택, 큐의 개념

✓ 링크드 큐

연결 리스트로 구현한 큐를 '링크드 큐' 라고 한다.

삽입과 삭제가 제한되지 않는 점과,
크기의 제한이 존재하지 않는 점이 편리하다.

01 스택, 큐의 개념

✓ 큐

Example

```
import queue  
q = queue.Queue()
```

큐를 실제로 사용하고자 할 때는 파이썬 기본 라이브러리의 **queue 모듈**의 **Queue 클래스**를 사용할 수 있다.

`/* elice */`

01 스택, 큐의 개념

✓ 마무리

스택과 큐는 선형 구조이다.

선형 구조는 자료들이 순서를 가지고 연속된 자료구조이다.

01 스택, 큐의 개념

✓ 마무리

스택은 나중에 들어온 자료가 먼저 출력되고,
큐는 먼저 들어온 자료가 먼저 출력된다.

01 스택, 큐의 개념

✓ 마무리

파이썬의 **리스트(배열)**를 사용하여 구현할 때
스택은 간편하게 구현이 가능하지만
큐의 경우는 **여러 문제점**이 발생한다.

01 스택, 큐의 개념

✓ 마무리

큐를 배열로 구현하였을 때의 문제점을 해결하는 방법은
원형 큐, 링크드 큐 등이 있다.

01 스택, 큐의 개념

✓ 마무리

스택과 큐를 **실제로 사용**하고자 할 때
스택은 그냥 **파이썬의 리스트**를,
큐는 **queue 모듈의 Queue 클래스**를 사용하면 된다.

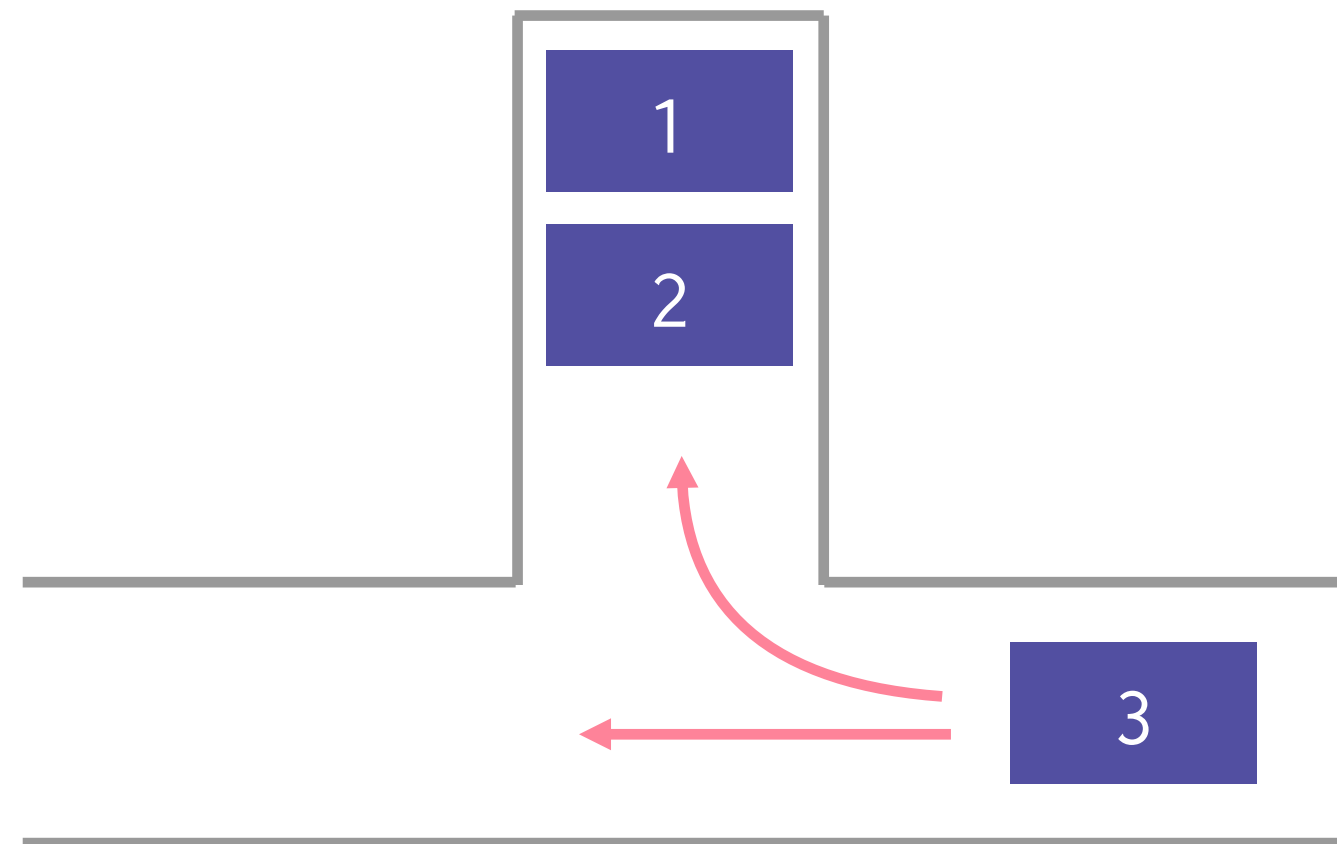
02

스택, 큐의 의미



02 스택, 큐의 의미

✓ 스택과 큐의 의미



자료구조는 정의하기 나름이기 때문에 위와 같은 자료구조도 만들 수는 있다.
하지만 왜 **스택**과 **큐**를 중요하게 다루고 따로 공부할까?

02 스택, 큐의 의미

✓ 스택과 큐의 의미

스택과 큐는 컴퓨터 내에서 **중요한 역할**을 갖고 있기 때문이다.
그리고 스택과 큐는 **가장 기본적인** 형태의 자료구조 중 하나이며
스택과 큐로 해결할 수 있는 문제인지 알기 위해서는
스택과 큐의 **의미**를 알고 있어야 한다.

02 스택, 큐의 의미

✓ 스택과 큐의 의미

아래 질문에 대답할 수 있어야 한다.

스택, 큐를 왜 배우는가?

스택, 큐를 어디에 쓰는가?

스택, 큐가 중요한 이유가 무엇인가?

02 스택, 큐의 의미

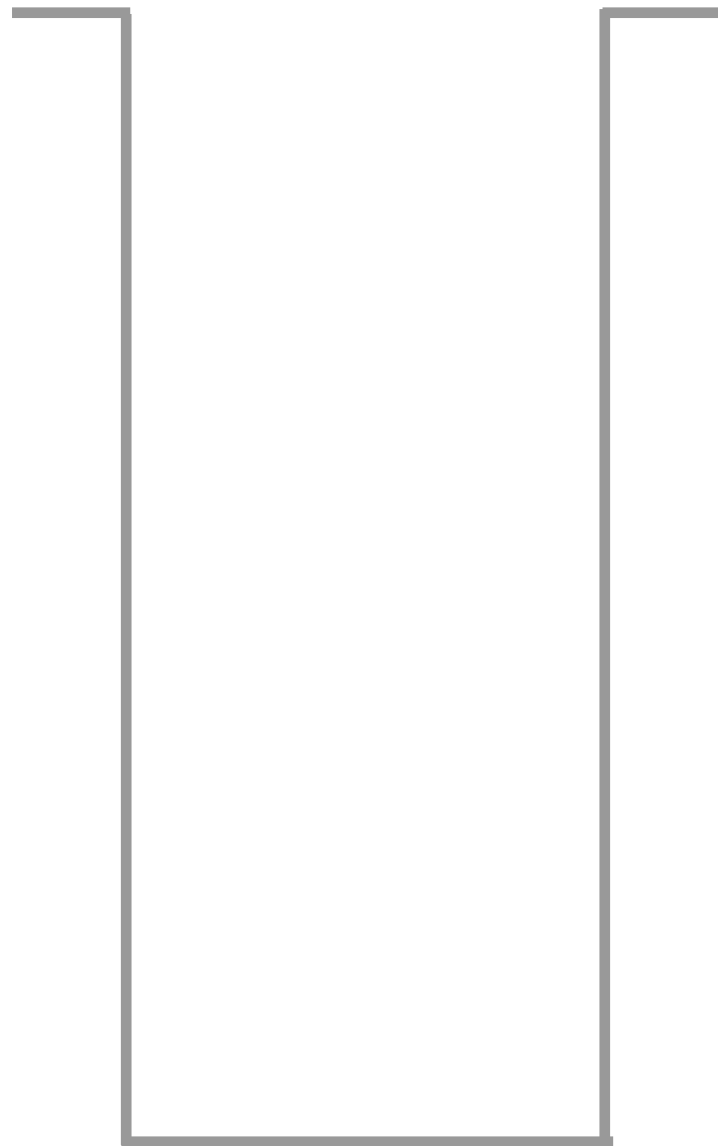
✓ 스택이 활용되는 대표적 예시

콜 스택(Call Stack)

컴퓨터 프로그램에서 **현재 실행 중인 함수(서브루틴)**을 저장하는 역할을 한다.

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def a(v) :  
    n = b(v)  
    return n
```

```
def b(v) :  
    m = c(v) * 2  
    return m
```

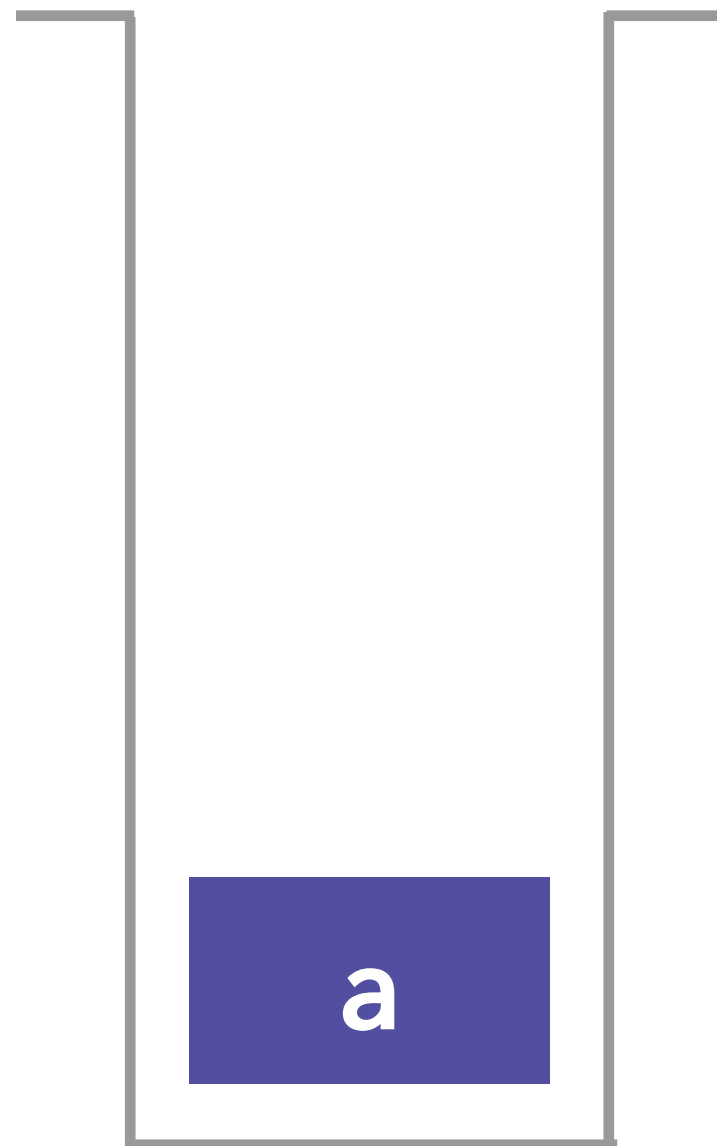
```
def c(v) :  
    return v ** 2
```

→ `print(a(5))`

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

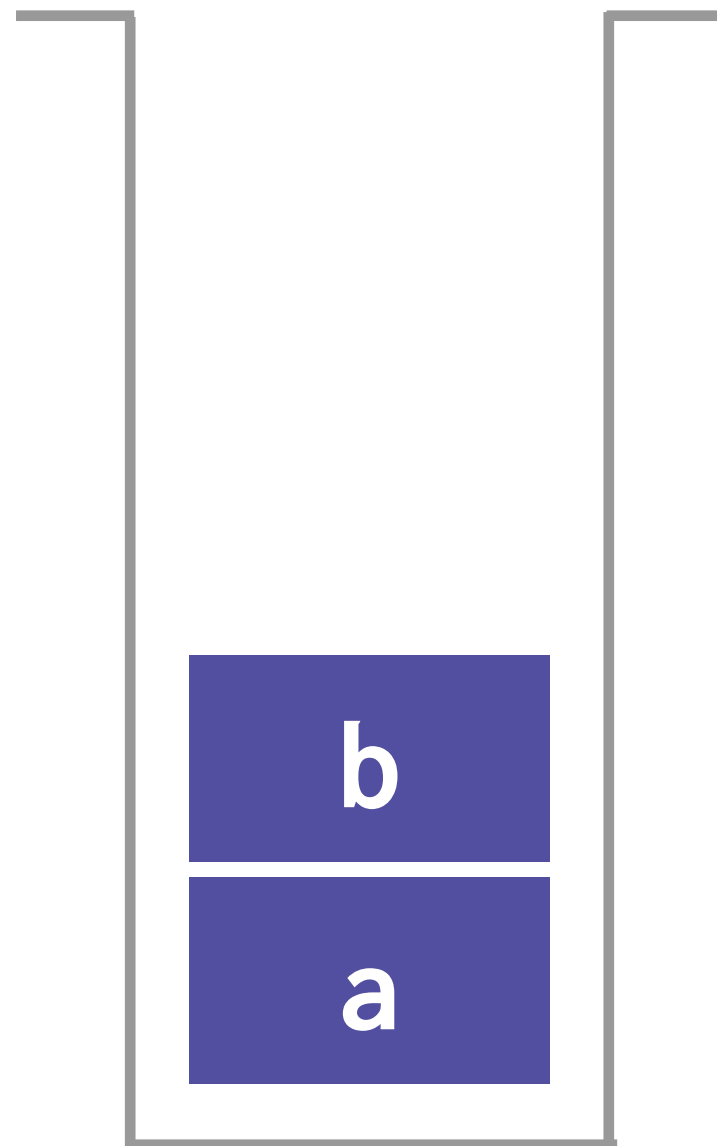
Example

```
def a(v) :  
    n = b(v)  
    return n  
  
def b(v) :  
    m = c(v) * 2  
    return m  
  
def c(v) :  
    return v ** 2  
  
print(a(5))
```

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

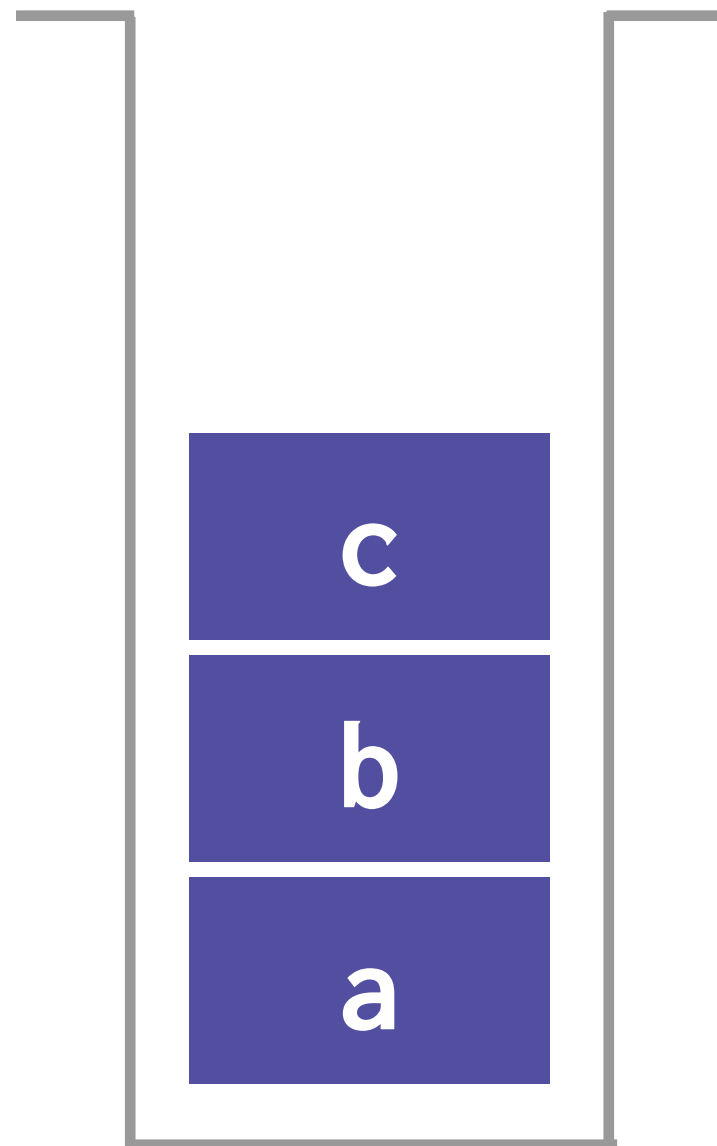
Example

```
def a(v) :  
    n = b(v)  
    return n  
  
def b(v) :  
    m = c(v) * 2  
    return m  
  
def c(v) :  
    return v ** 2  
  
print(a(5))
```

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

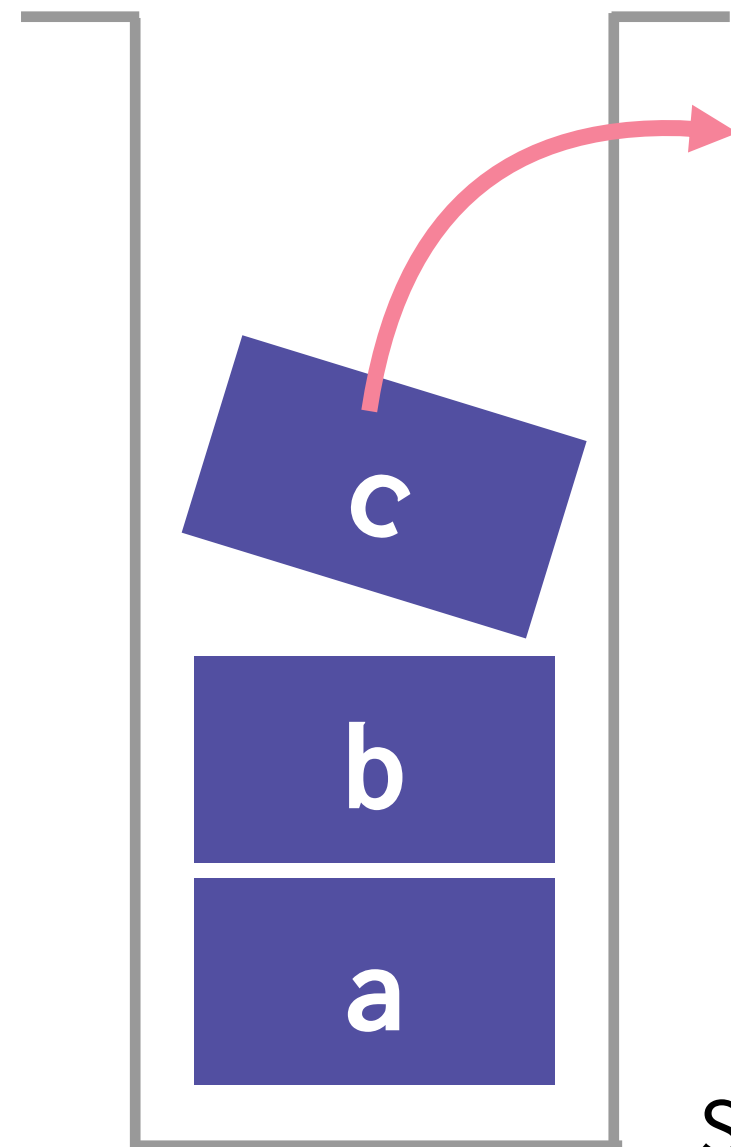
Example

```
def a(v) :  
    n = b(v)  
    return n  
  
def b(v) :  
    m = c(v) * 2  
    return m  
  
def c(v) :  
    return v ** 2  
  
print(a(5))
```

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

return이 실행되면
Stack에서 현재 함수를
pop하게 된다.

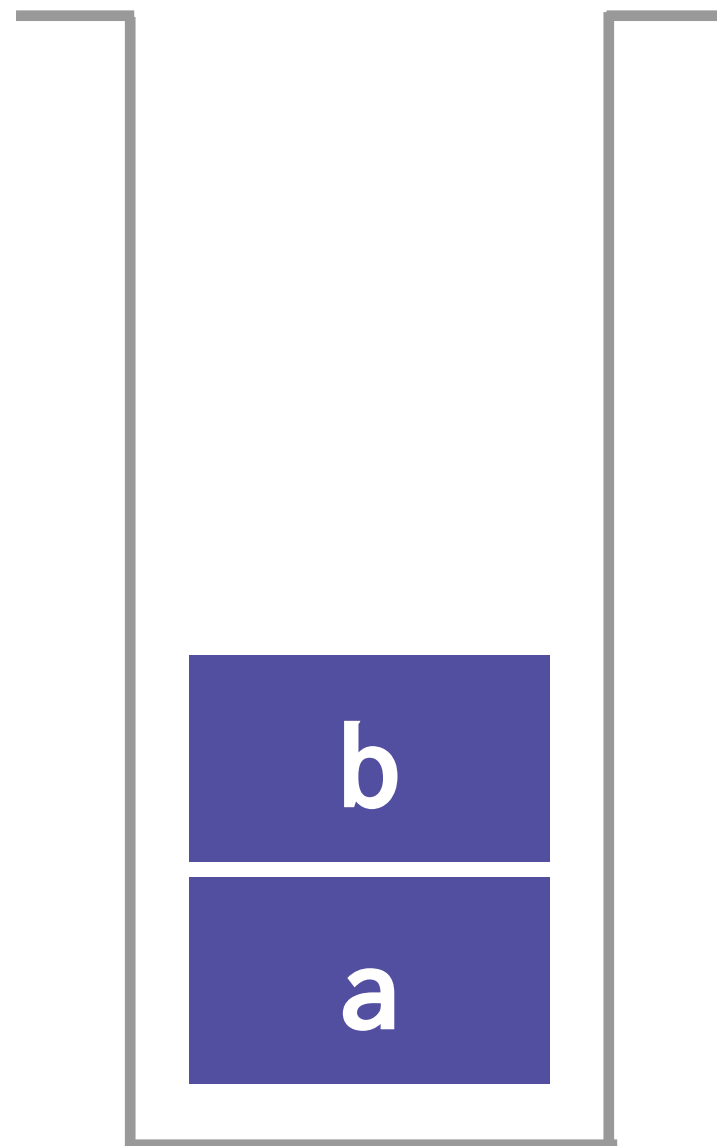
Example

```
def a(v) :  
    n = b(v)  
    return n  
  
def b(v) :  
    m = c(v) * 2  
    return m  
  
def c(v) :  
    return v ** 2  
  
print(a(5))
```

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Call Stack에 이전에
코드가 실행된 상황을
기록해두었기 때문에
반환할 지점을 찾아
돌아갈 수 있다.

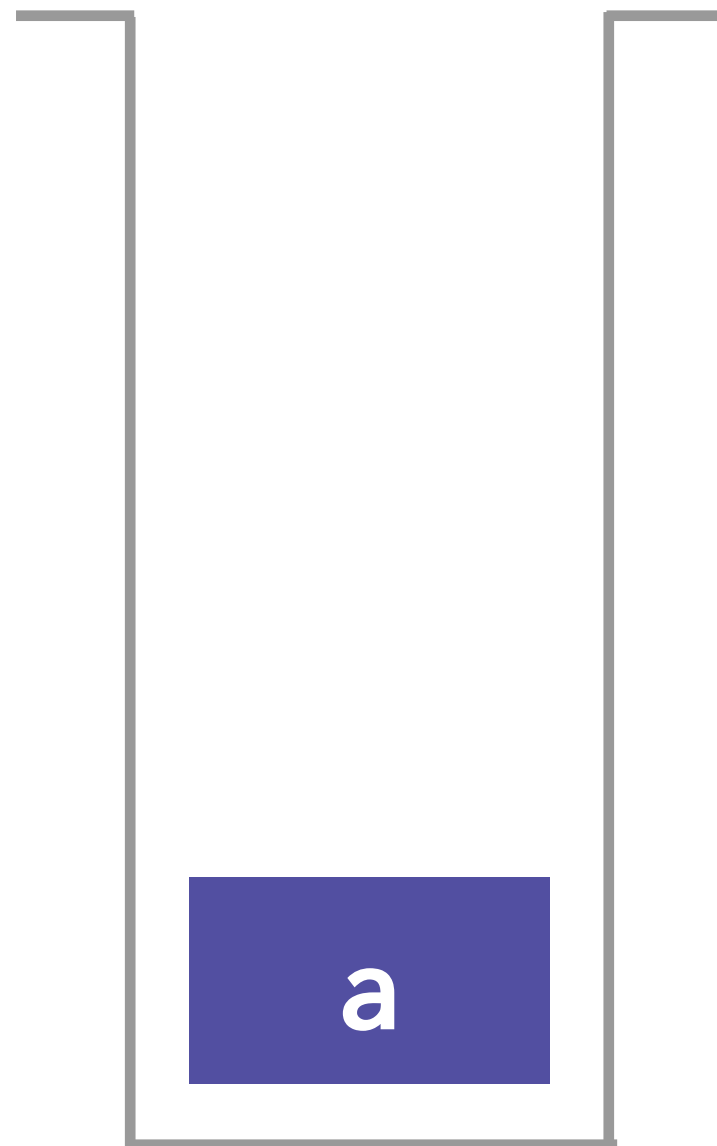
Example

```
def a(v) :  
    n = b(v)  
    return n  
  
def b(v) :  
    m = c(v) * 2  
    return m  
  
def c(v) :  
    return v ** 2  
  
print(a(5))
```

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def a(v) :  
    n = b(v)  
    return n
```

```
def b(v) :  
    m = c(v) * 2  
    return m
```

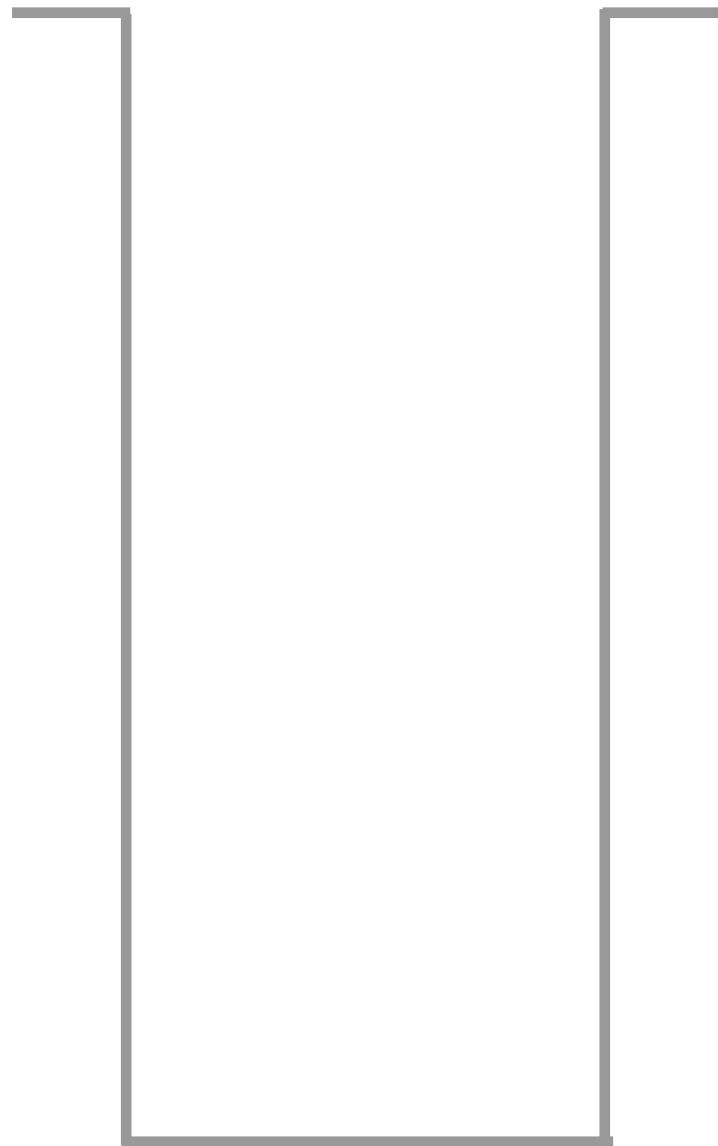
```
def c(v) :  
    return v ** 2
```

```
print(a(5))
```

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def a(v) :  
    n = b(v)  
    return n
```

```
def b(v) :  
    m = c(v) * 2  
    return m
```

```
def c(v) :  
    return v ** 2
```

→ `print(a(5))` 50 출력

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시

스택은 **의존관계가 있는 상태**를 저장한다.

어떤 일보다 **더 먼저** 처리되어야 하는 일이 있다면,
스택에 저장할 수 있다.

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시

factorial
 $0! = 1$
 $n! = n * (n - 1)!$



Example

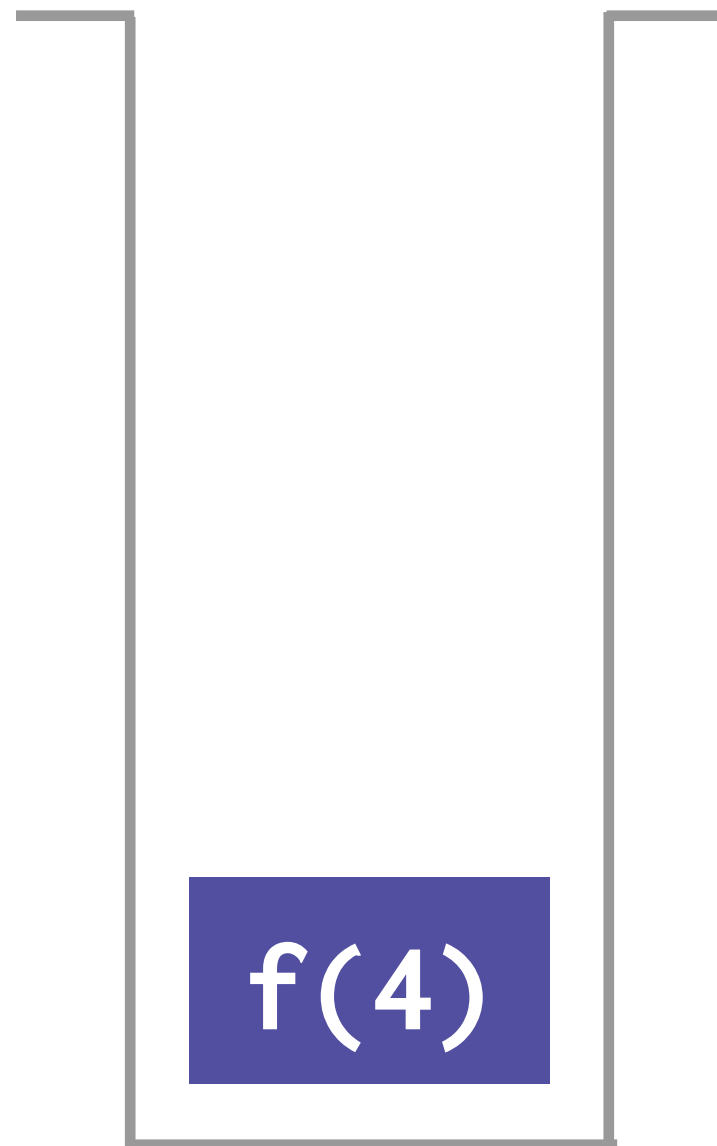
```
def factorial(n) :  
    if n == 0 :  
        return 1  
  
    return n * factorial(n-1)  
  
print(factorial(4))
```

팩토리얼 연산의 정의를 코드로 옮긴 factorial 함수이다.
이 함수는 **재귀함수**, 즉 자기 자신을 호출하는 함수이다.

/ elice */*

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def factorial(n) :    현재 n의 값 : 4

    if n == 0 :
        return 1

    return n * factorial(n-1)

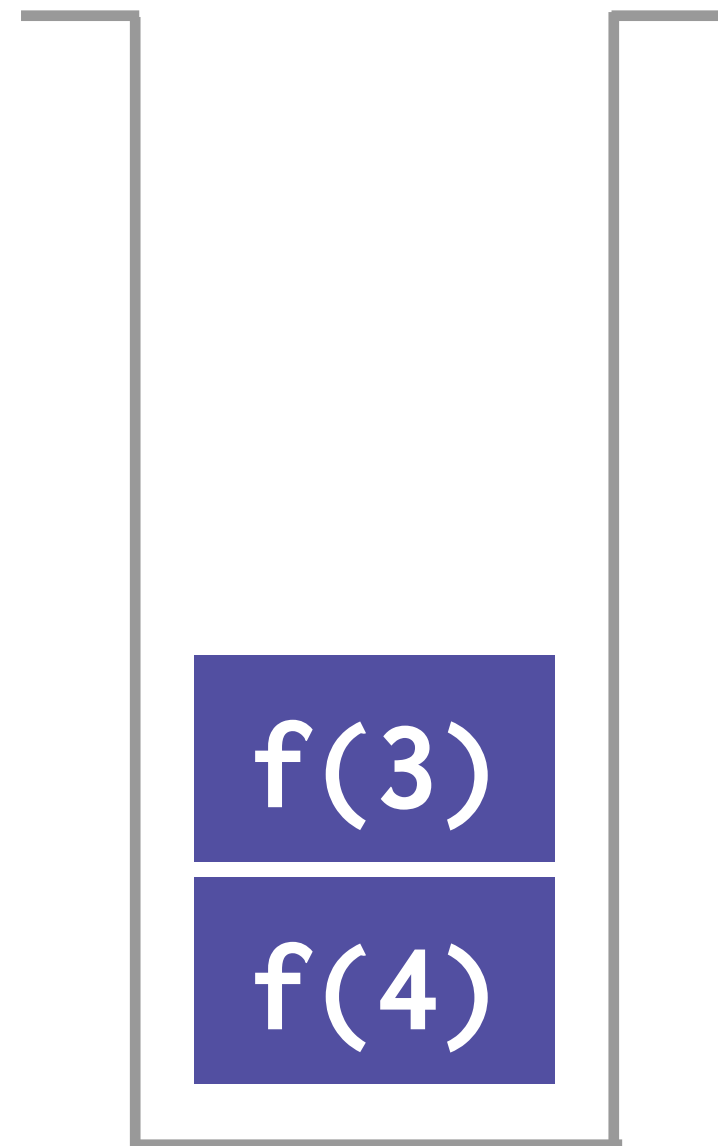
→ print(factorial(4))
```

처음에 4!의 값을 구하기 위해
`factorial(4)` 를 호출한다.

`/* elice */`

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def factorial(n) :    현재 n의 값 : 3

    if n == 0 :
        return 1

    return n * factorial(n-1)

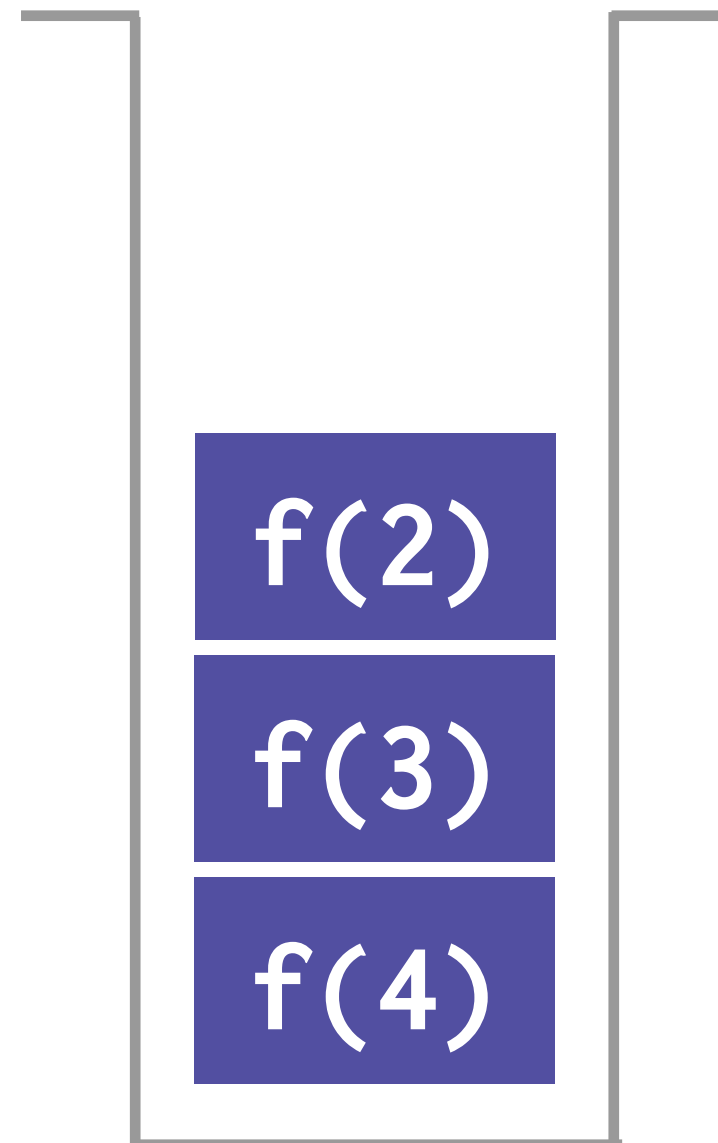
print(factorial(4))
```

4!를 구하기 위해서는 **3!을 먼저** 구해야 하므로
`factorial(3)` 을 호출한다.

/ elice */*

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def factorial(n) :    현재 n의 값 : 2

    if n == 0 :
        return 1

    return n * factorial(n-1)

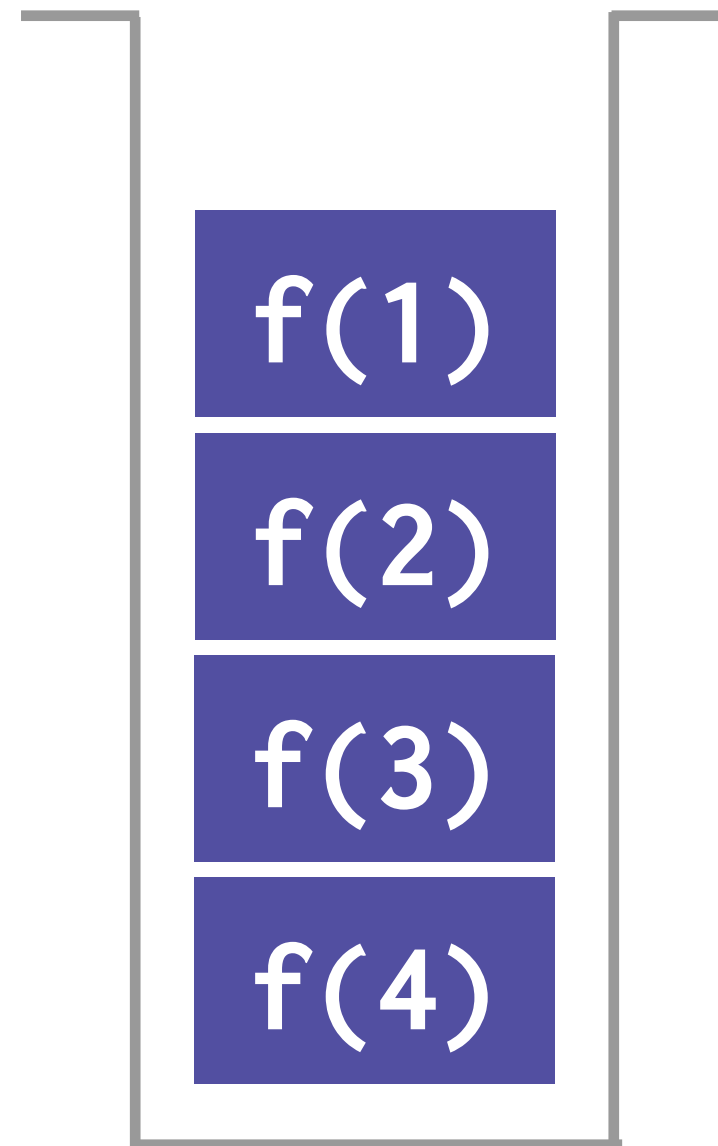
print(factorial(4))
```

3!를 구하기 위해서는 **2!을 먼저** 구해야 하므로
`factorial(2)`를 호출한다.

/ elice */*

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def factorial(n) :    현재 n의 값 : 1

    if n == 0 :
        return 1

    return n * factorial(n-1)

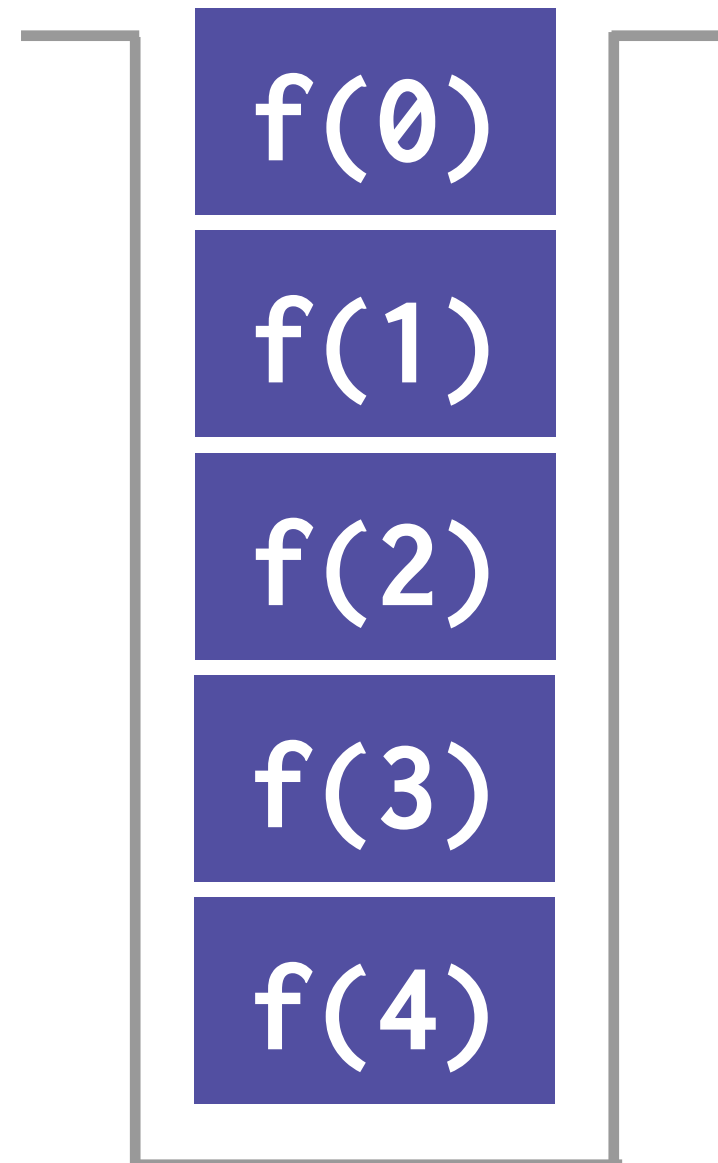
print(factorial(4))
```

2!를 구하기 위해서는 **1!을 먼저** 구해야 하므로
factorial(1)을 호출한다.

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def factorial(n) :    현재 n의 값 : 0

    if n == 0 :
        return 1

    return n * factorial(n-1)

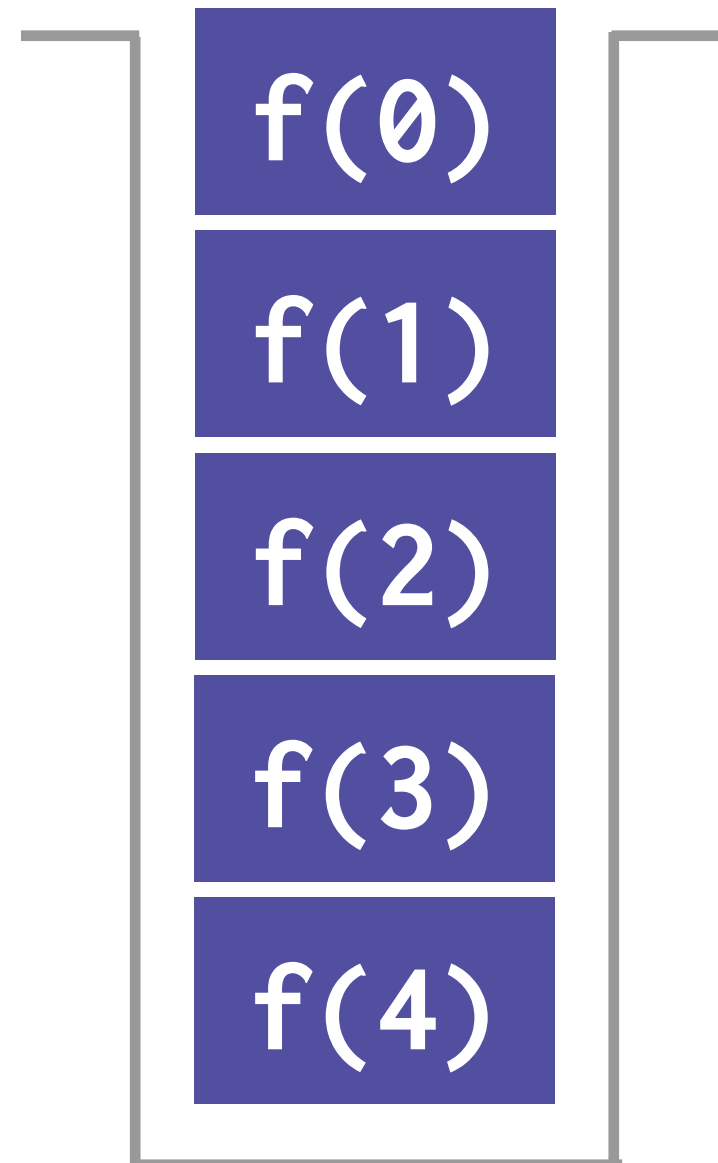
print(factorial(4))
```

1!를 구하기 위해서는 0!을 먼저 구해야 하므로
factorial(0)을 호출한다.

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def factorial(n) :    현재 n의 값 : 0

    if n == 0 : #True
        return 1

    return n * factorial(n-1)

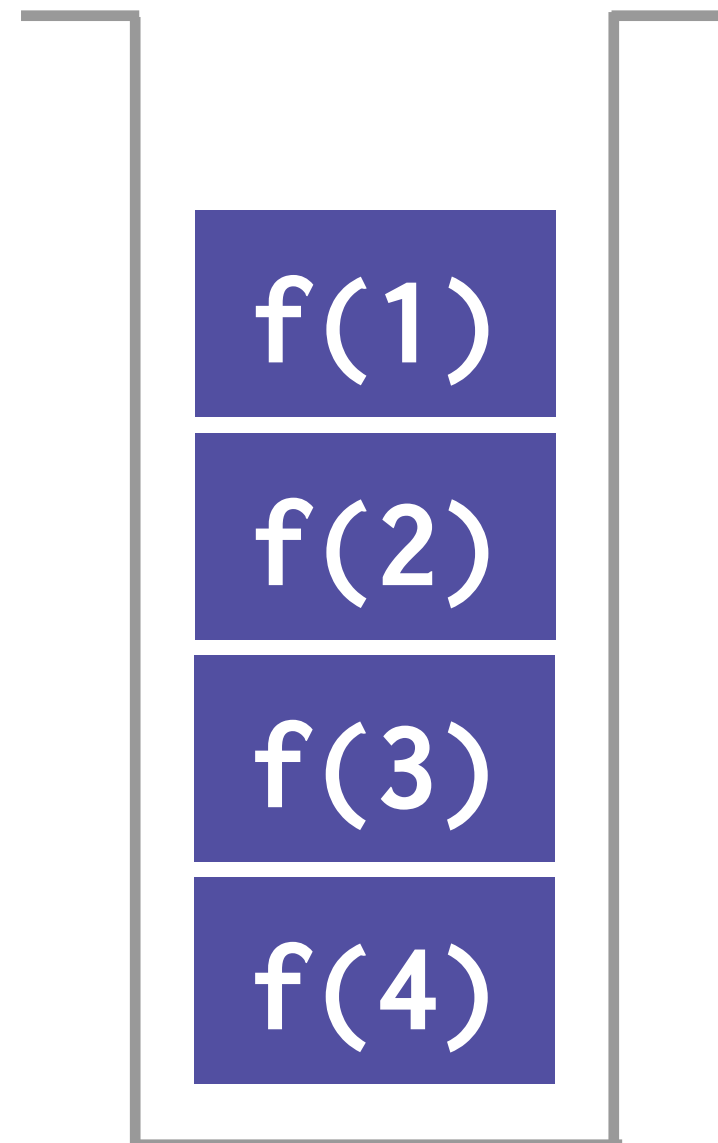
print(factorial(4))
```

0!의 값은 1로 정의되어 있다.
따라서 조건문을 통과하고, 1을 반환한다.

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def factorial(n) :    현재 n의 값 : 1  
  
    if n == 0 : #True  
        return 1  
  
    return n * factorial(n-1)  
print(factorial(4))
```

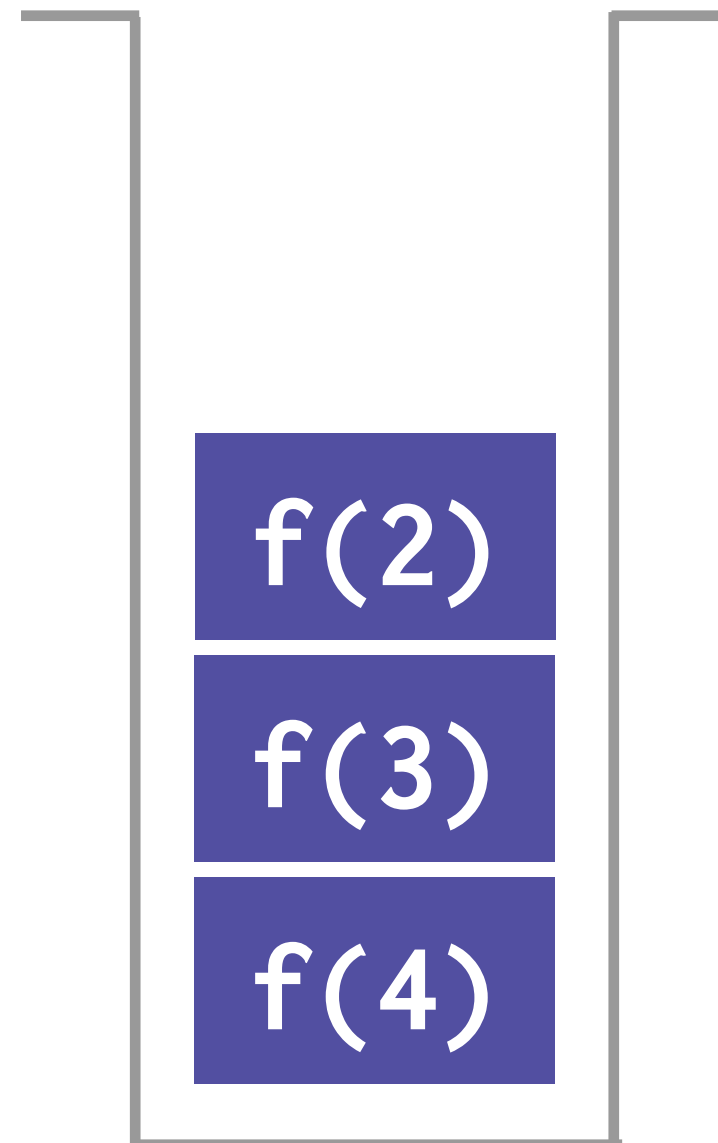
A red arrow points from the `print(factorial(4))` line to the `factorial(n-1)` line. Another red arrow points from the `n-1` to the value `1` below it.

`factorial(0)`의 반환값은
Call Stack에 의해 `factorial(1)`에 전달된다.

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def factorial(n) :    현재 n의 값 : 2

    if n == 0 : #True
        return 1

    return n * factorial(n-1)

print(factorial(4))
```

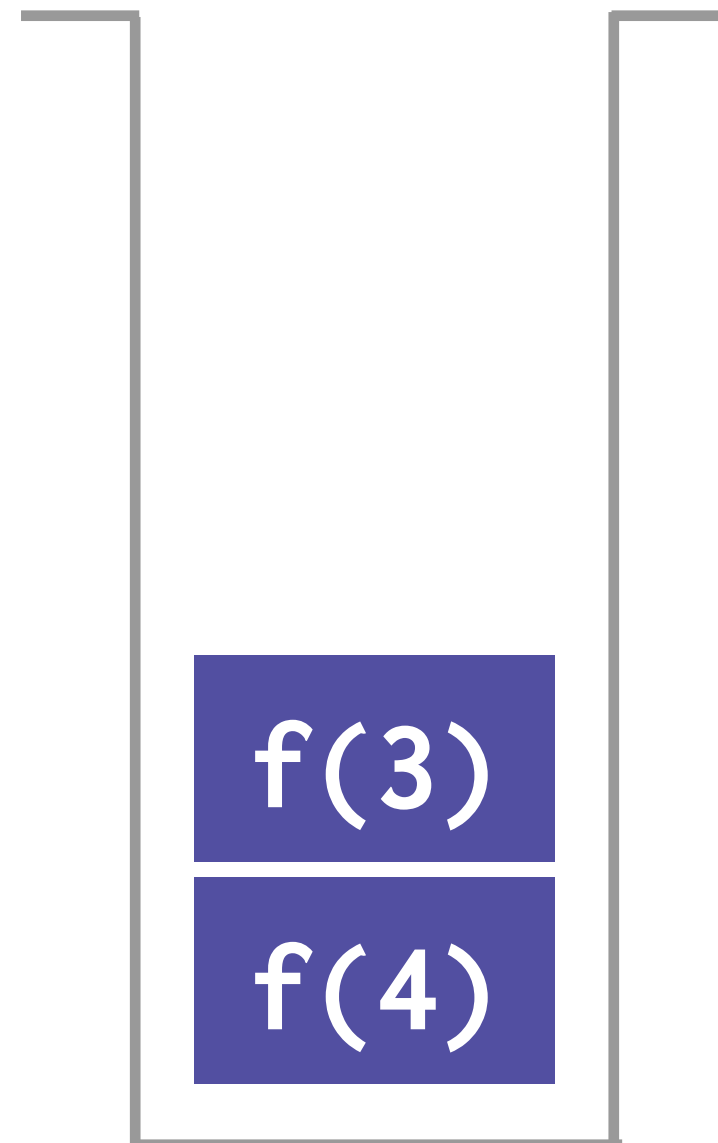
A red arrow points from the left to the line `return n * factorial(n-1)`. Another red arrow points from the `n-1` to the value `1` in the `print(factorial(4))` line.

factorial(1)의 반환값은
Call Stack에 의해 factorial(2)에 전달된다.

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def factorial(n) :    현재 n의 값 : 3

    if n == 0 : #True
        return 1

    return n * factorial(n-1)

print(factorial(4))
```

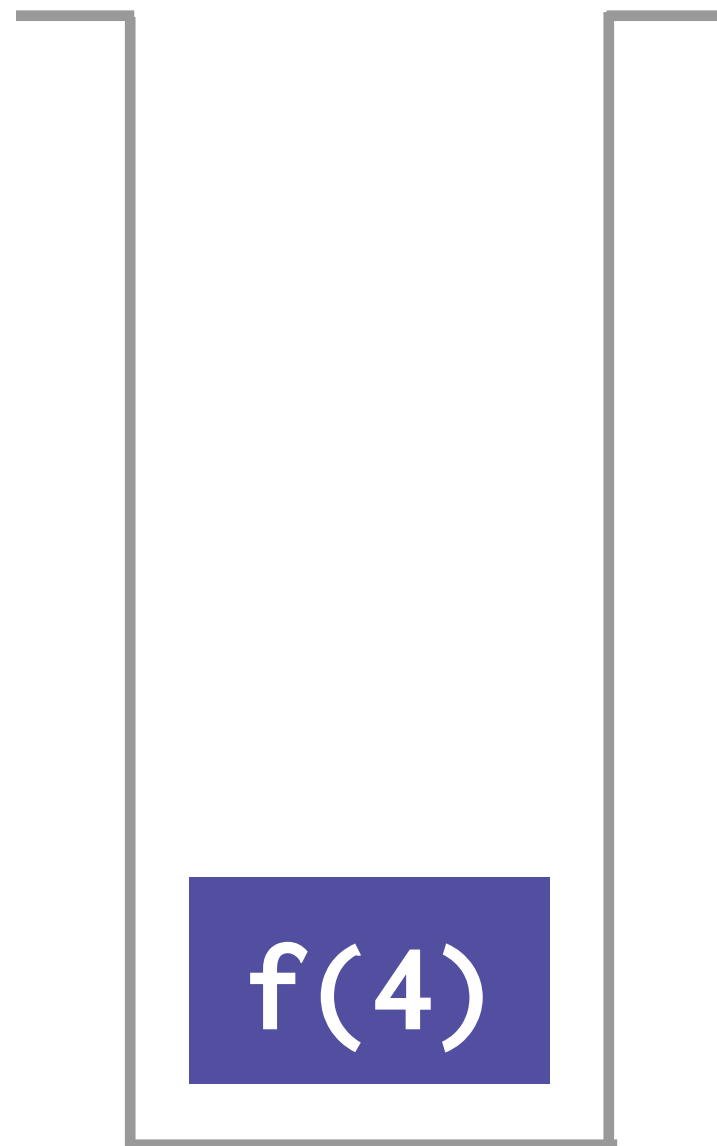
The diagram shows the execution of a factorial function. A red arrow points from the 'return' statement in the code to the 'f(3)' box in the call stack diagram. Another red arrow points from the 'n-1' in the recursive call to the number '2' below it, indicating the next step in the calculation.

factorial(2)의 반환값은
Call Stack에 의해 factorial(3)에 전달된다.

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def factorial(n) :    현재 n의 값 : 4

    if n == 0 : #True
        return 1

    return n * factorial(n-1)

print(factorial(4))
```

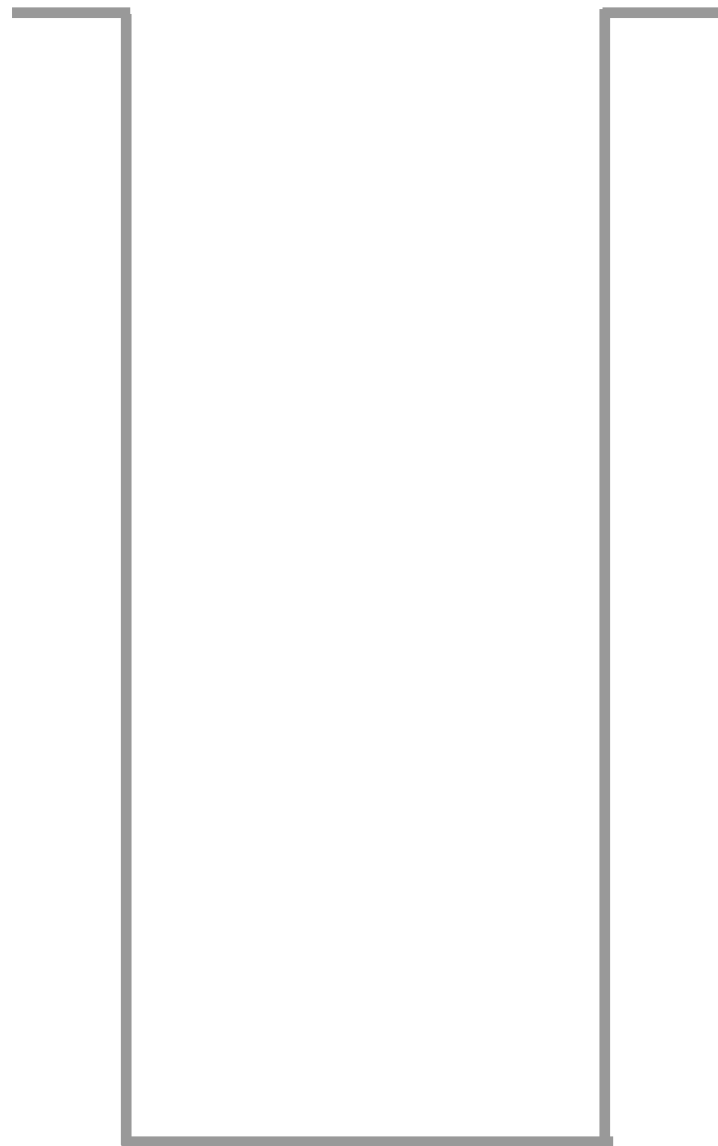
A red arrow points from the `print(factorial(4))` line to the `return n * factorial(n-1)` line. Another red arrow points from the `factorial(n-1)` to the value `6` below it.

`factorial(3)`의 반환값은
Call Stack에 의해 `factorial(4)`에 전달된다.

/ elice */*

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시



Call Stack

Stack의 가장 위에 있는 작업이
현재 수행중인 작업이다.

Example

```
def factorial(n) :  
    if n == 0 : #True  
        return 1  
  
    return n * factorial(n-1)  
  
→ print(factorial(4))    24 출력
```

factorial(4)의 반환값은
함수 바깥의 print로 전달되어 출력된다.

/* elice */

02 스택, 큐의 의미

✓ 스택이 활용되는 대표적 예시

$n!$ 를 구하기 위해서는 $(n - 1)!$ 를 먼저 구해야 했던 것처럼
여러 작업들 사이에 **의존관계**가 있을 때
스택을 이용하여 표현할 수 있다.

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

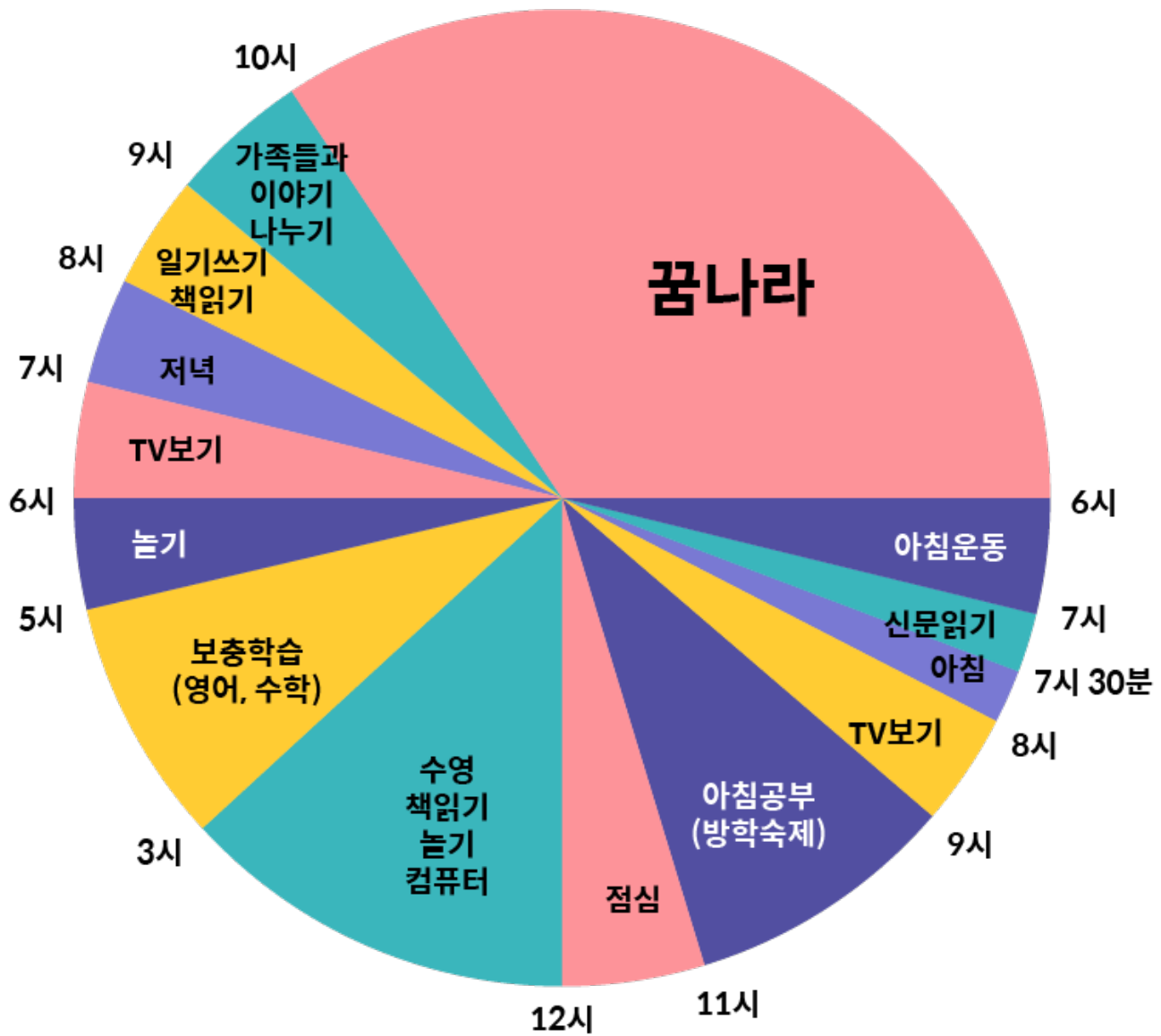
스케줄링(Scheduling)

운영 체제가 **프로세스**를 관리하는 기법으로
동시에 실행되는 여러 프로세스에 CPU 등의 **자원 배정**을 적절히 함으로써
성능을 개선할 수 있다.

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

간단한 스케줄링 예시



02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

오전 계획

`/* elice */`

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

아침
운동

오전 계획

1. 아침운동

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

아침
운동

신문
읽기

오전 계획

1. 아침운동
2. 신문 읽기

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

아침
운동

신문
읽기

TV
보기

오전 계획

1. 아침운동
2. 신문 읽기
3. TV 보기

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시



해야 할 일을 모두 큐에 정리한다.

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

아침
운동

신문
읽기

TV
보기

아침
공부

오전 계획

1. 아침운동 (수행 완료)

2. 신문 읽기

3. TV 보기

4. 아침 공부

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

신문
읽기

TV
보기

아침
공부

오전 계획

1. 아침운동 (수행 완료)
2. 신문 읽기 (수행 완료)
3. TV 보기
4. 아침 공부

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

TV
보기

아침
공부

오전 계획

1. 아침운동 (수행 완료)
2. 신문 읽기 (수행 완료)
3. TV 보기 (수행 완료)
4. 아침 공부

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

아침
공부

오전 계획

1. 아침운동 (수행 완료)
2. 신문 읽기 (수행 완료)
3. TV 보기 (수행 완료)
4. 아침 공부(수행 완료)

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

오전 계획

1. 아침운동 (수행 완료)
2. 신문 읽기 (수행 완료)
3. TV 보기 (수행 완료)
4. 아침 공부(수행 완료)

수행을 완료한 작업들은 큐에서 pop한다.

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

컴퓨터가 작업하는 프로세스의 속도는
사람의 생활 계획표의 작업보다 **수행 시간**이 훨씬 짧다.

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

여러 프로세스를 동시에 수행할 수 있게 하기 위한 기법인
시분할 시스템을 비롯하여 운영체제의 스케줄링 알고리즘은 매우 다양하지만
대체로 '큐'를 기반으로 스케줄링을 관리하고 있다.

02 스택, 큐의 의미

✓ 큐가 활용되는 대표적 예시

이와 같이 어떤 작업이 **병렬적**으로 이루어져도 관찬을 때,
작업들 사이에 **의존관계가 없다면**
큐에 저장하여 관리할 수 있다.

02 스택, 큐의 의미

✓ 정리

스택은 어떤 작업이 다른 작업보다 **먼저** 이루어져야만 하는 경우
큐는 여러 작업들이 **동시에(병렬적으로)** 이루어져도 상관없는 경우

02 스택, 큐의 의미

✓ 정리

스택은 의존관계가 있는 경우

큐는 의존관계가 없는 경우

`/* elice */`

02 스택, 큐의 의미

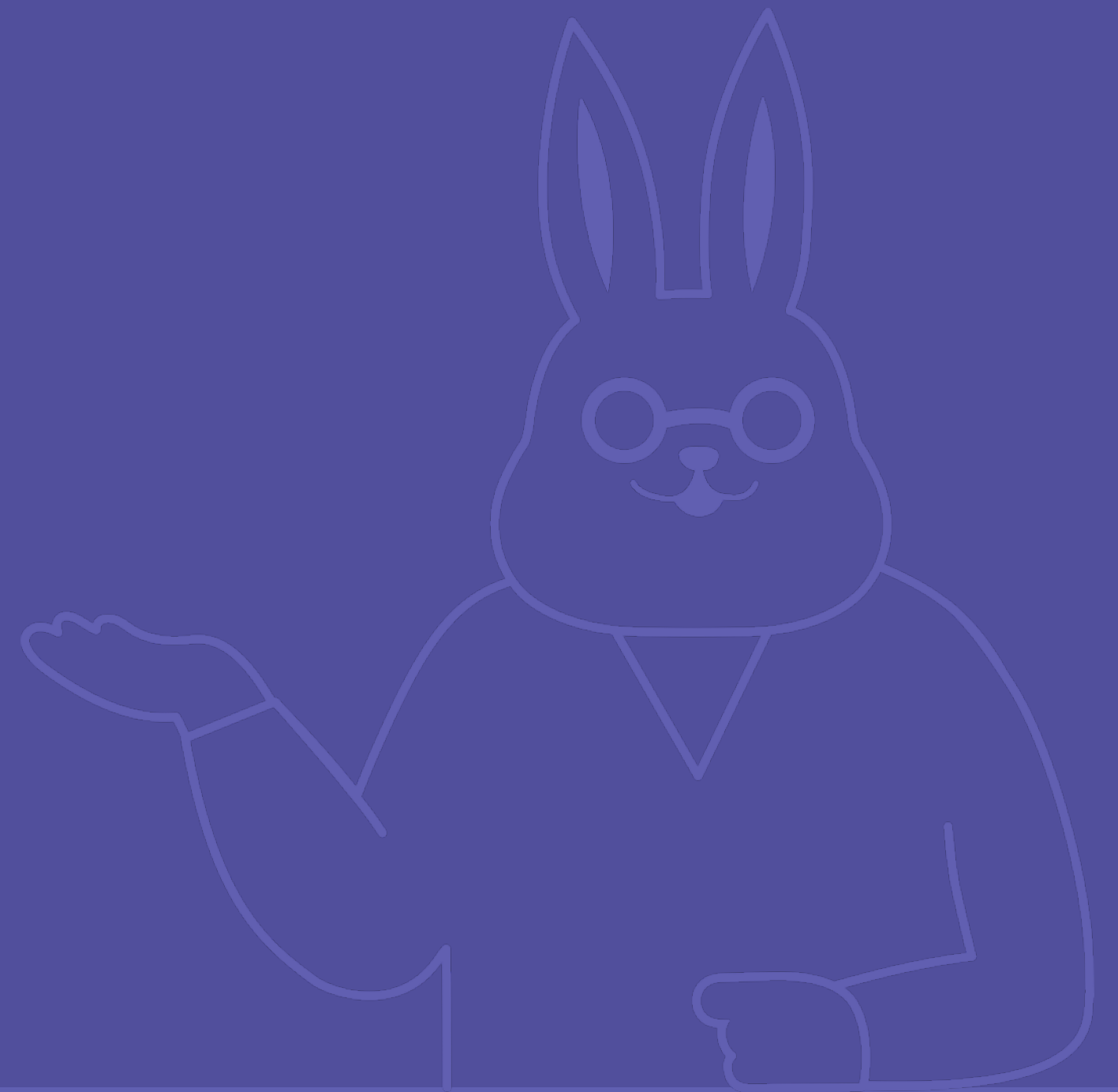
✓ 정리

앞서 **콜 스택**과 **재귀함수**로 스택의 활용 예시를 다루었는데,
실제로 문제 풀이에서 **스택**을 사용해야 하는 경우
재귀함수를 이용하여 구현할 수 있다.

(예 : 그래프, 트리의 DFS 탐색 등)

03

스택으로 풀 수 있는 문제



03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

입력된 괄호 문자열이 **올바른 괄호 문자열인지** 확인해봐야 한다.

입력 예시

```
((()))  
()  
(()())  
)()
```

출력 예시

```
Yes  
No  
Yes  
No
```


03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

괄호 문자열을 구성하는 각 괄호를
하나씩 처리해보자.

우선 **올바른** 괄호 문자열의 경우부터 알아보자.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

입력된 괄호 문자열 : ((()))
괄호를 **하나씩** 처리해보자.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

1번째 괄호 : 정상 입력

(

입력된 괄호 문자열 : ((()))

괄호를 **하나씩** 처리해보자.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

2번째 괄호 : 정상 입력

((

입력된 괄호 문자열 : ((()))

괄호를 **하나씩** 처리해보자.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

3번째 괄호 : 정상 입력

(((

입력된 괄호 문자열 : ((()))

괄호를 **하나씩** 처리해보자.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

4번째 괄호 : 정상 입력

(((^{완성})

입력된 괄호 문자열 : ((()))

괄호를 **하나씩** 처리해보자.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

5번째 괄호 : 정상 입력

((()))
완성

입력된 괄호 문자열 : ((()))
괄호를 **하나씩** 처리해보자.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

6번째 괄호 : 정상 입력

((())) 완성

입력된 괄호 문자열 : ((()))

괄호를 **하나씩** 처리해보자.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

괄호가 중첩되었을 때,
늦게 등장한 여는 괄호는
일찍 등장한 여는 괄호보다 더 먼저 완성된다.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

이번에는 **올바르지 않은 경우**를 알아보자.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

입력된 괄호 문자열 : (()
괄호를 **하나씩** 처리해보자.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

1번째 괄호 : 정상 입력

(

입력된 괄호 문자열 : (()
괄호를 **하나씩** 처리해보자.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

2번째 괄호 : 정상 입력

((

입력된 괄호 문자열 : ((
괄호를 **하나씩** 처리해보자.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

3번째 괄호 : 정상 입력

(()^{완성}


입력된 괄호 문자열 : (()
괄호를 **하나씩** 처리해보자.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

닫히지 않은 괄호가 존재하므로 올바르지 않은 괄호이다.



(()

입력된 괄호 문자열 : (()
괄호를 하나씩 처리해보자.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

입력된 괄호 문자열 : ()
괄호를 **하나씩** 처리해보자.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

1번째 괄호 : 정상 입력

(

입력된 괄호 문자열 : ()
괄호를 **하나씩** 처리해보자.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

2번째 괄호 : 정상 입력

완성 ()

입력된 괄호 문자열 : ()
괄호를 **하나씩** 처리해보자.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기

닫는 괄호가 닫을 수 있는 여는 괄호가 없기 때문에 올바르지 않은 괄호이다.

()

입력된 괄호 문자열 : ()
괄호를 하나씩 처리해보자.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 요약

1. 아직 완성되지 않은 여는 괄호들이 있을 때, 나중에 등장한 여는 괄호는 항상 먼저 등장한 여는 괄호보다 일찍 완성된다.

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 요약

1. 아직 완성되지 않은 여는 괄호들이 있을 때, 나중에 등장한 여는 괄호는 항상 먼저 등장한 여는 괄호보다 일찍 완성된다.
2. 닫는 괄호를 처리할 때, 닫을 수 있는 여는 괄호가 존재하지 않은 경우는 올바르지 않은 경우이다.

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 요약

1. 아직 완성되지 않은 여는 괄호들이 있을 때, 나중에 등장한 여는 괄호는 항상 먼저 등장한 여는 괄호보다 일찍 완성된다.
2. 닫는 괄호를 처리할 때, 닫을 수 있는 여는 괄호가 존재하지 않은 경우는 올바르지 않은 경우이다.
3. 모든 괄호들을 처리하고 나서, 여는 괄호가 남김없이 모두 완성된 경우가 올바른 경우이다.

03 스택으로 풀 수 있는 문제

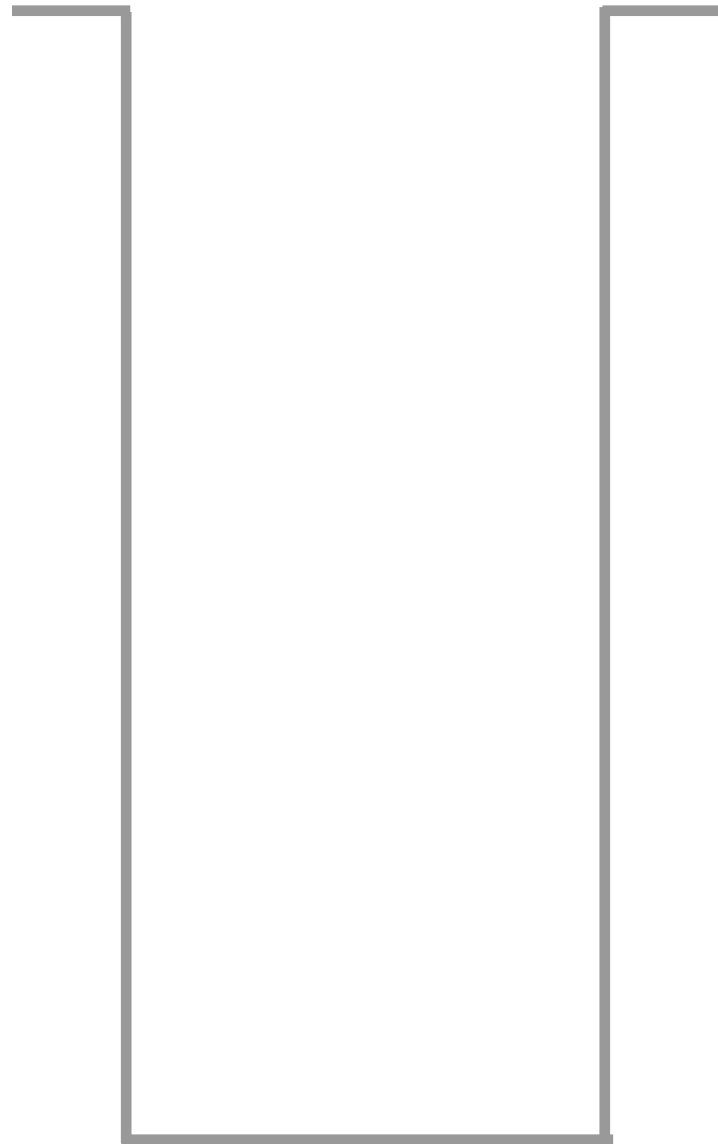
✓ [실습3] 올바른 괄호인지 판단하기

괄호 문자열을 구성하는 각 괄호를
하나씩 처리해보자.

이번엔 **스택**을 이용해보자.

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시1



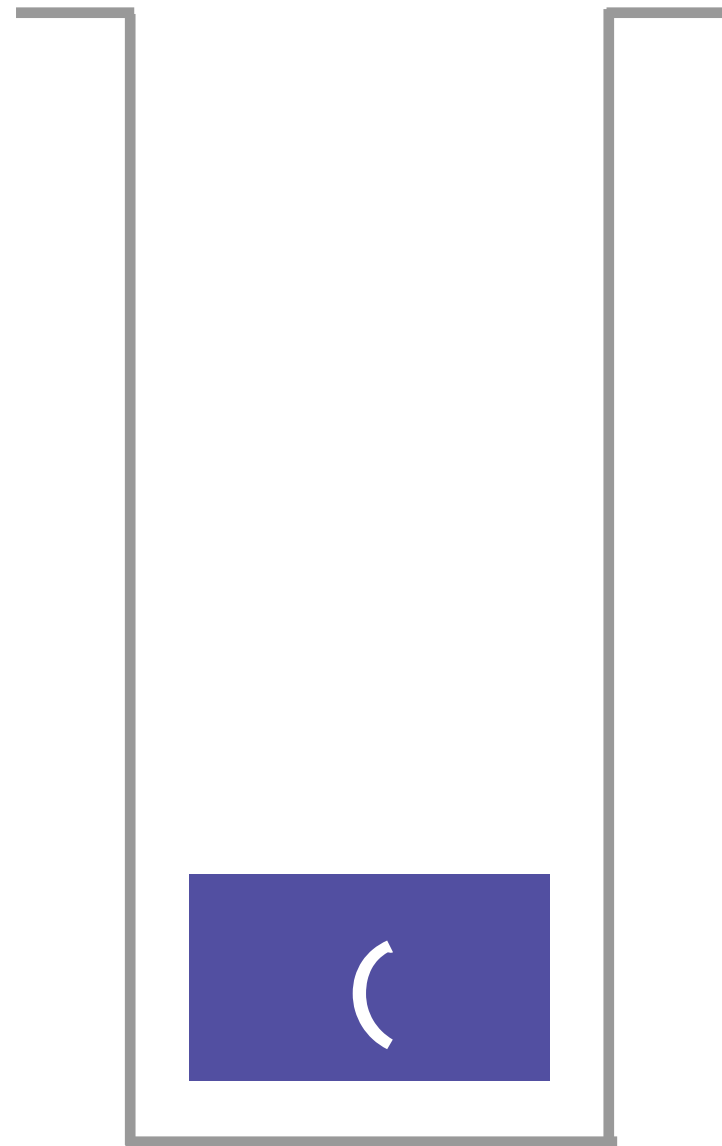
Stack

아직 닫히지 않은
'(' 를 저장한다.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시1



Stack

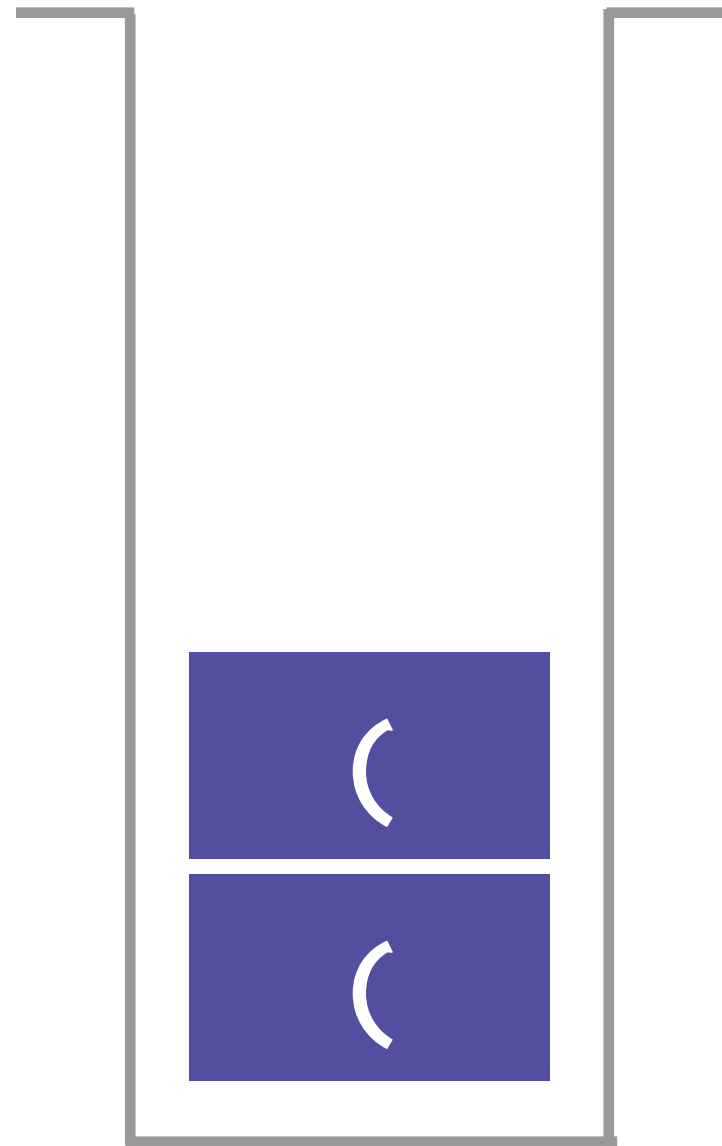
아직 닫히지 않은
'(' 를 저장한다.

(

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시1



Stack

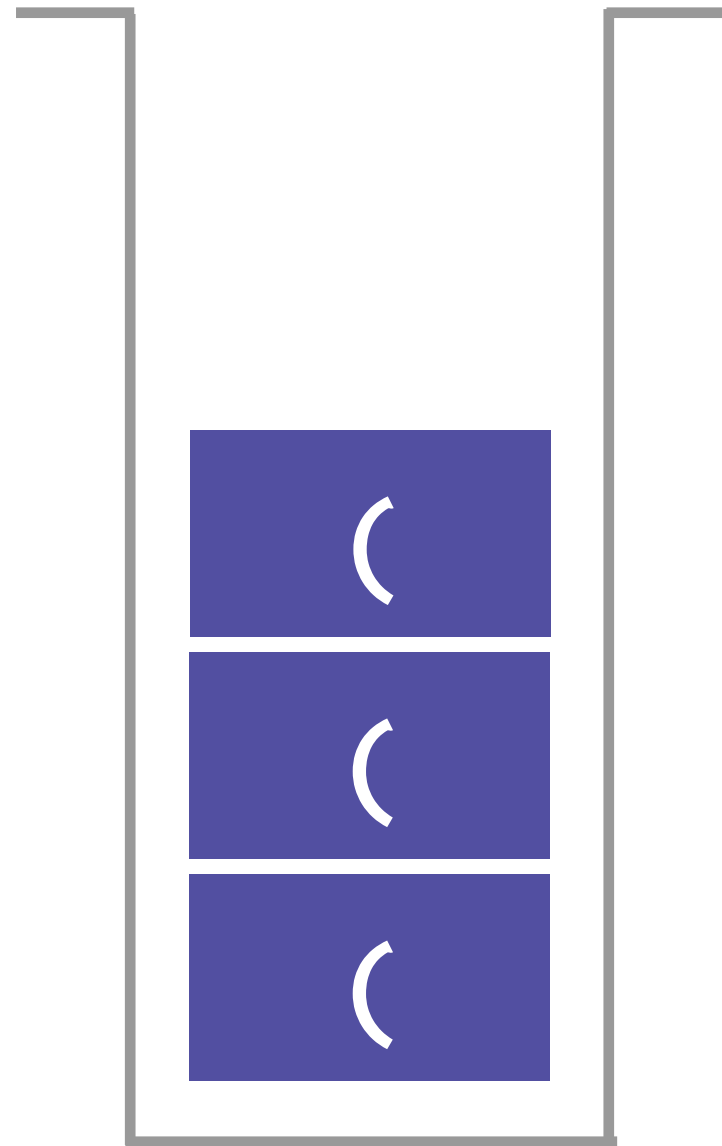
아직 닫히지 않은
'(' 를 저장한다.

((

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시1



Stack

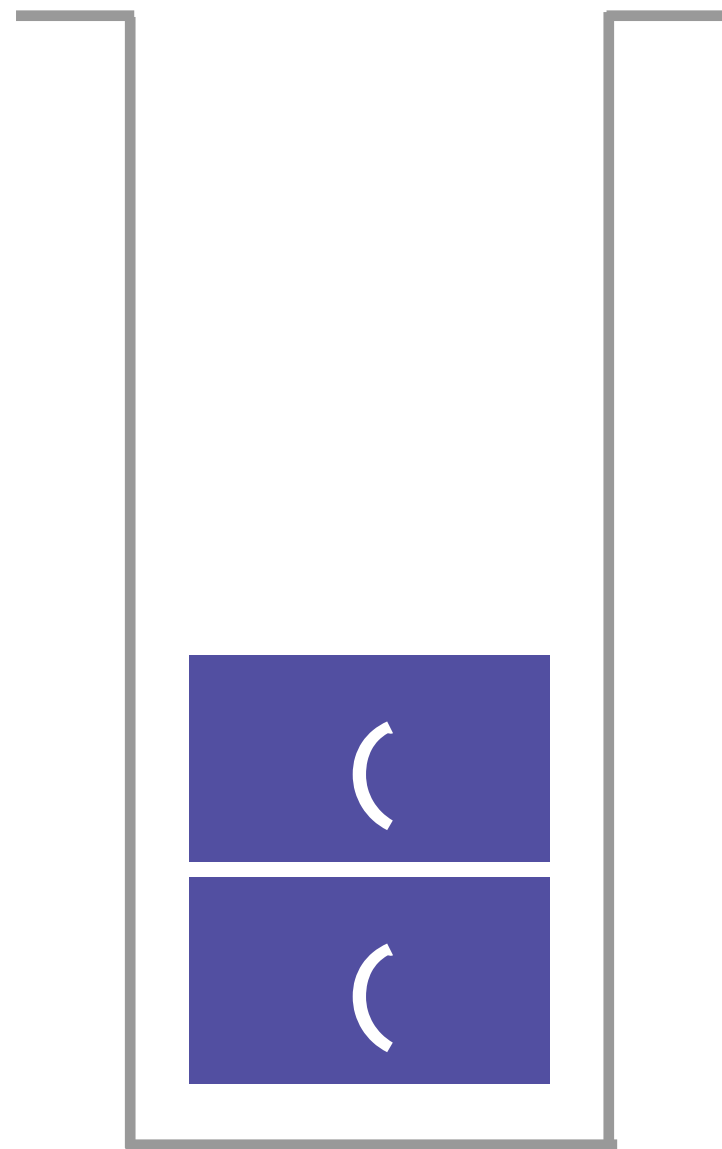
아직 닫히지 않은
'(' 를 저장한다.

(((

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시1



Stack

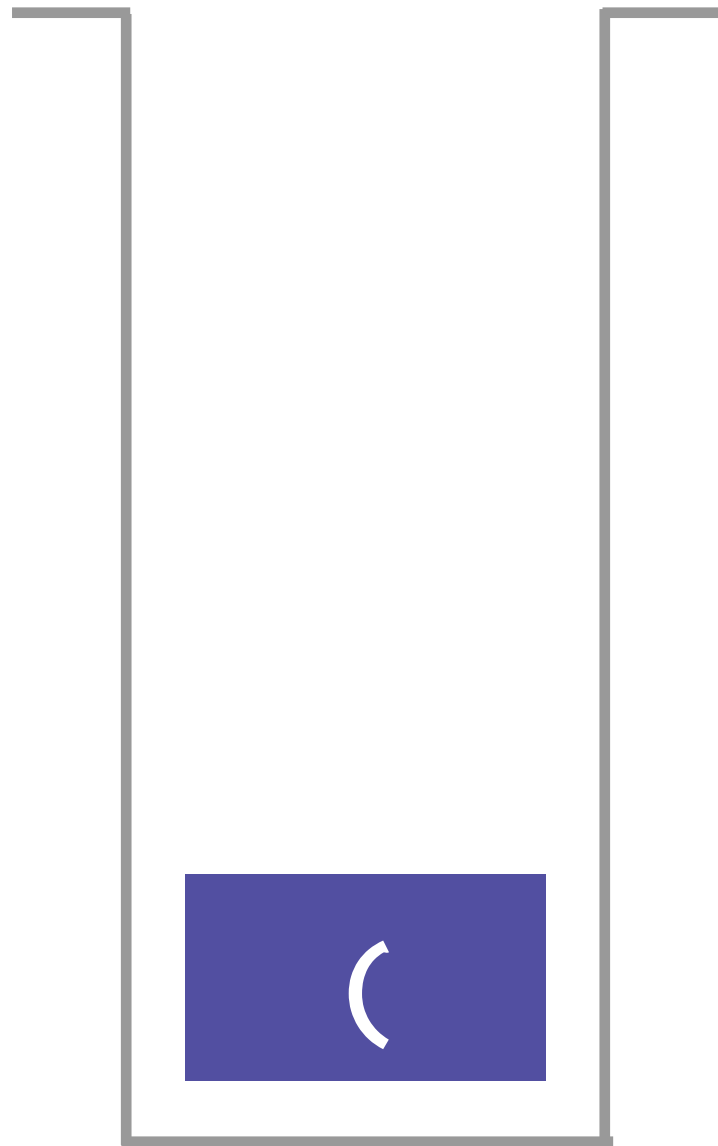
아직 닫히지 않은
'(' 를 저장한다.

((()
pop

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시1



Stack

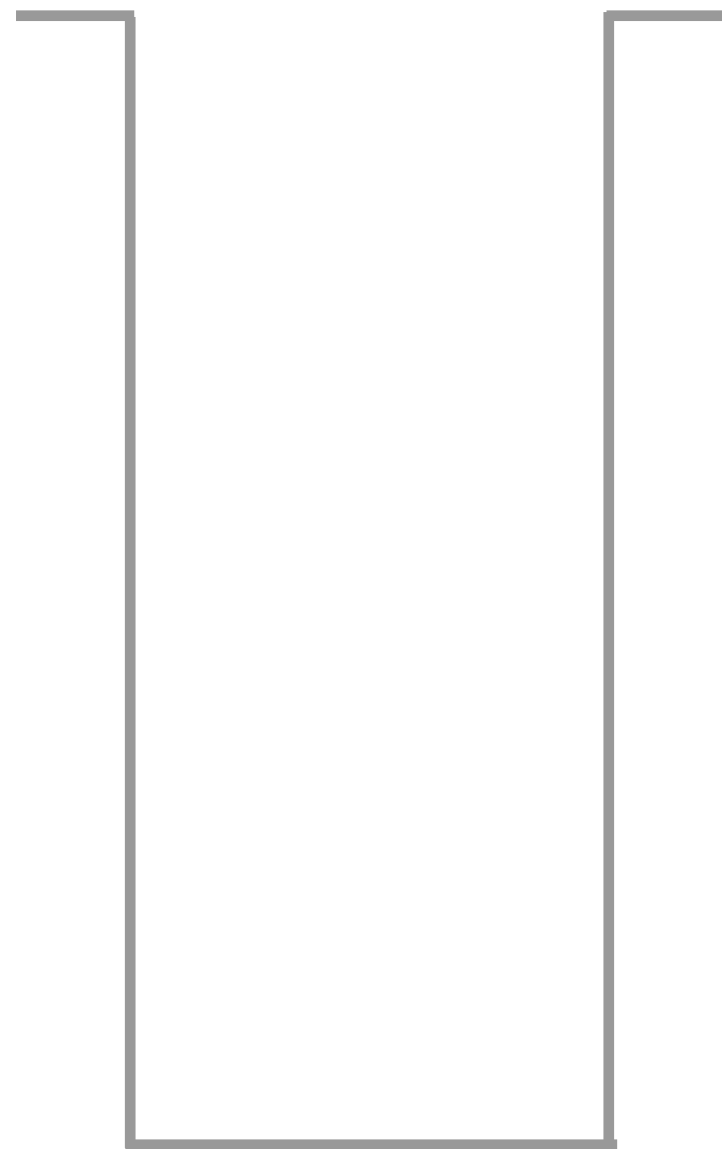
아직 닫히지 않은
'(' 를 저장한다.

((())
pop

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시1



Stack

아직 닫히지 않은
'(' 를 저장한다.

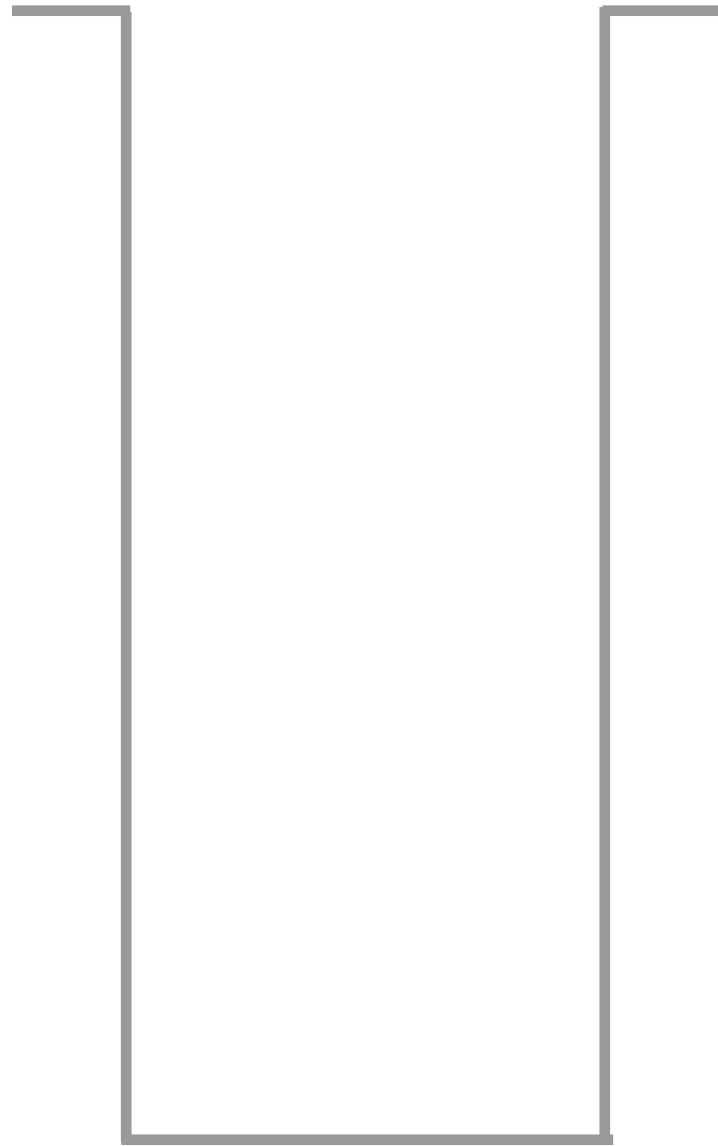
((()))
pop

괄호 입력이 모두 끝났을 때
스택이 비어있으면
올바른 괄호 문자열이다.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시2



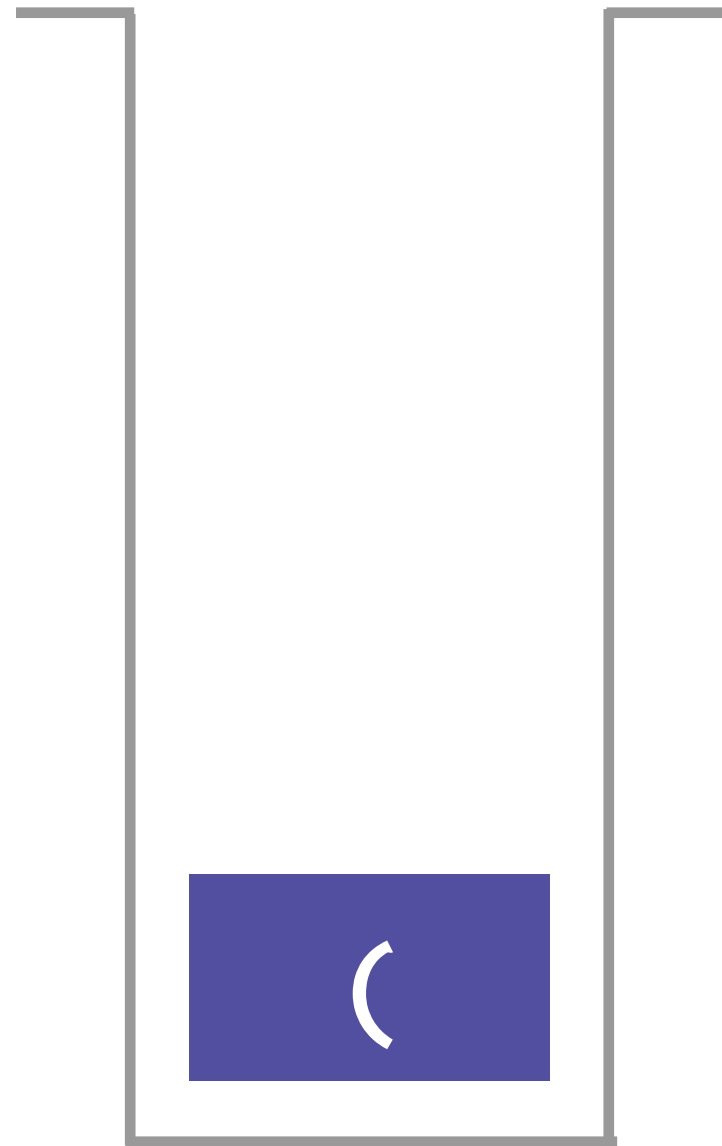
Stack

아직 닫히지 않은
'(' 를 저장한다.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시2



Stack

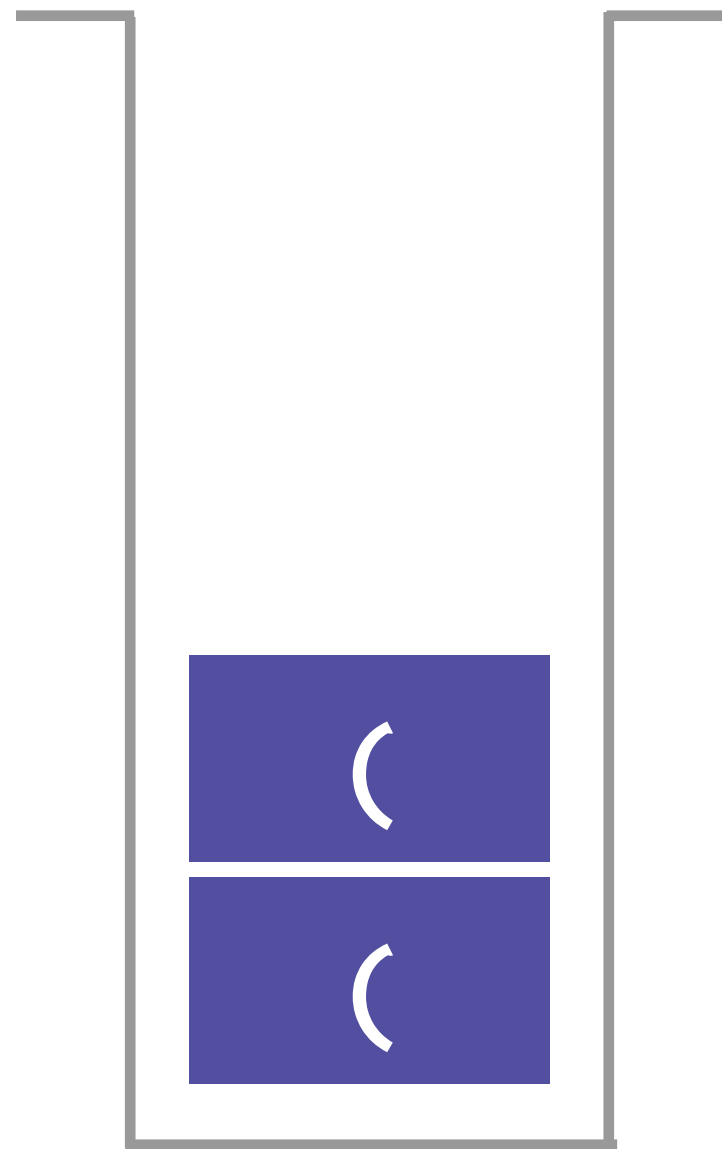
아직 닫히지 않은
'(' 를 저장한다.

(

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시2



Stack

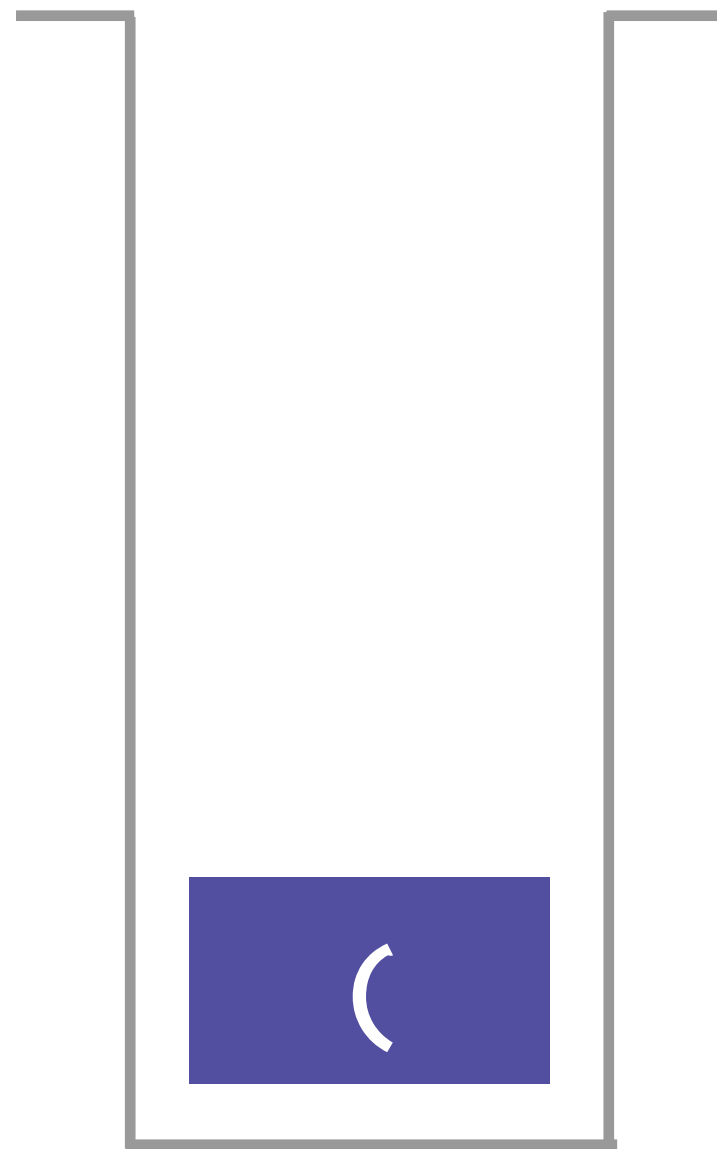
아직 닫히지 않은
'(' 를 저장한다.

((

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시2



Stack

아직 닫히지 않은
'(' 를 저장한다.

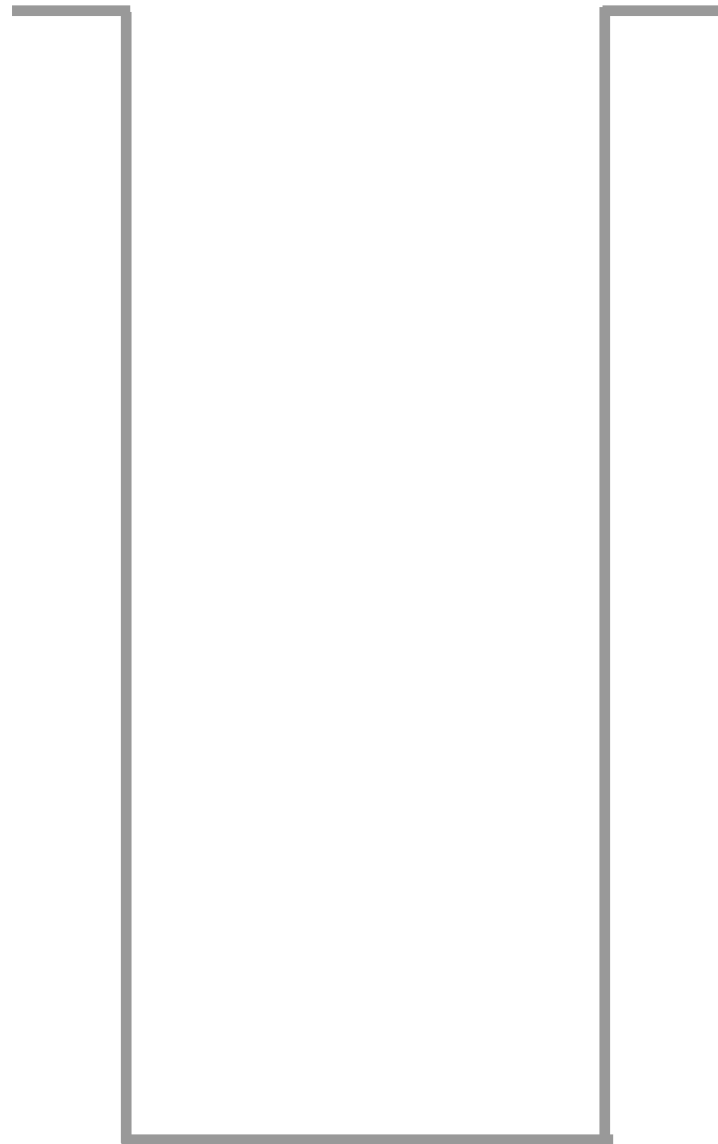
(()
pop

괄호 입력이 모두 끝났을 때
스택에 괄호가 남아있으면
올바르지 않은 괄호 문자열이다.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시3



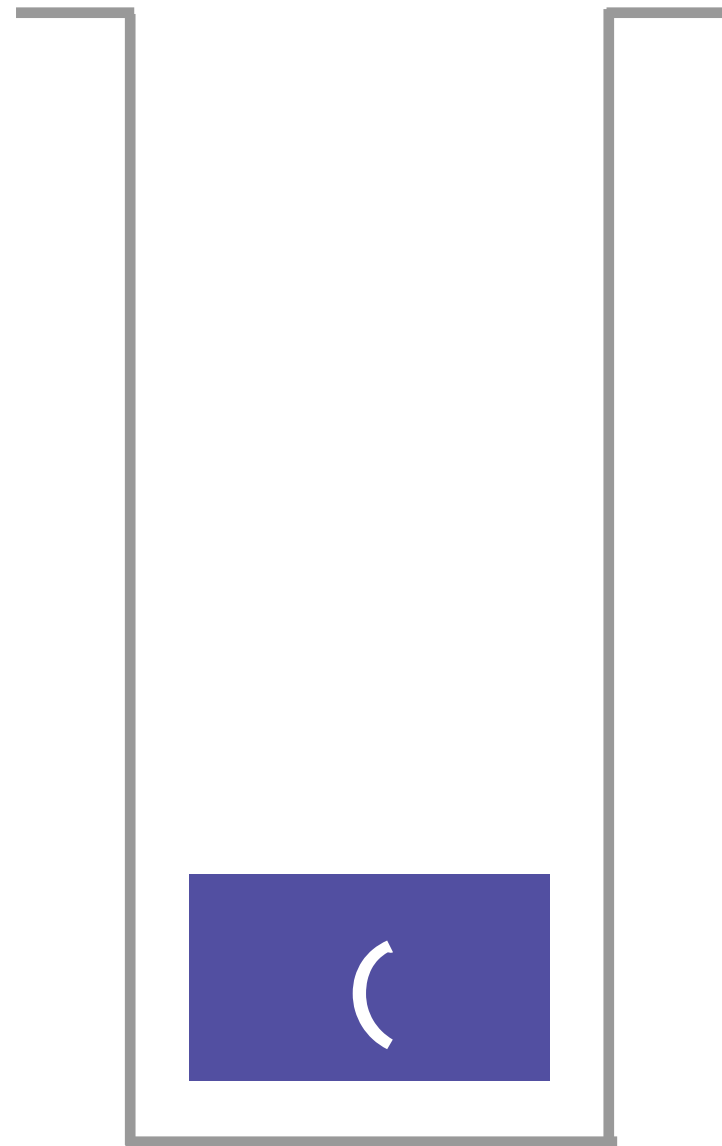
Stack

아직 닫히지 않은
'(' 를 저장한다.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시3



Stack

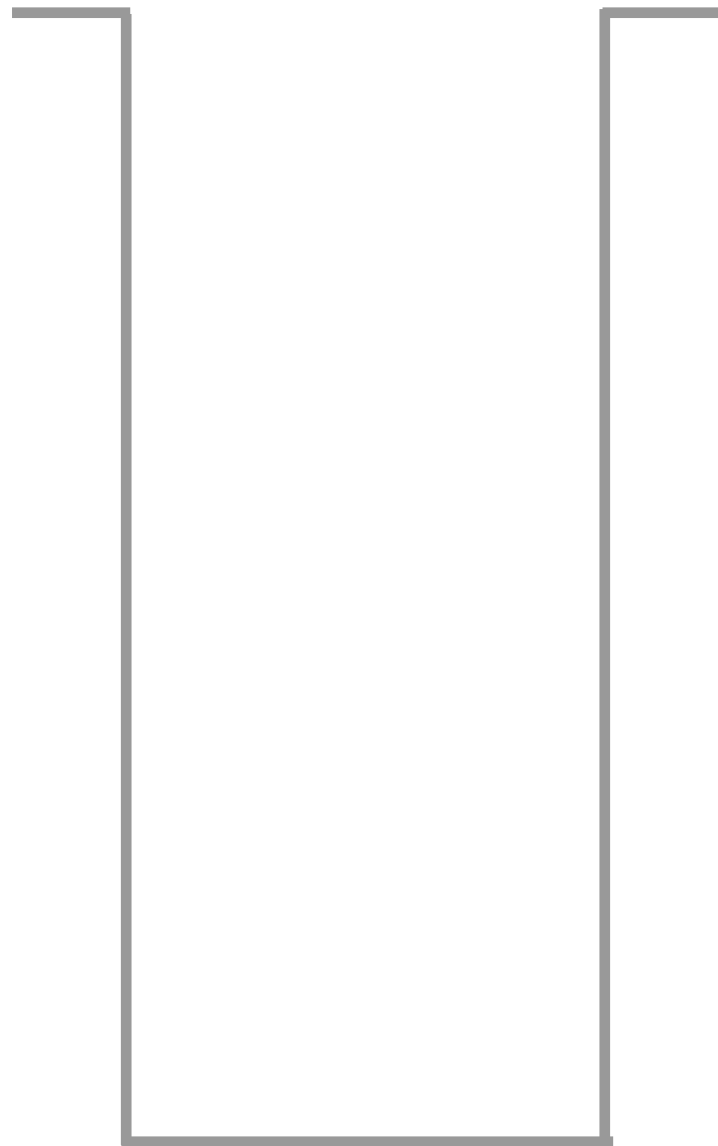
아직 닫히지 않은
'(' 를 저장한다.

(

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시3



Stack

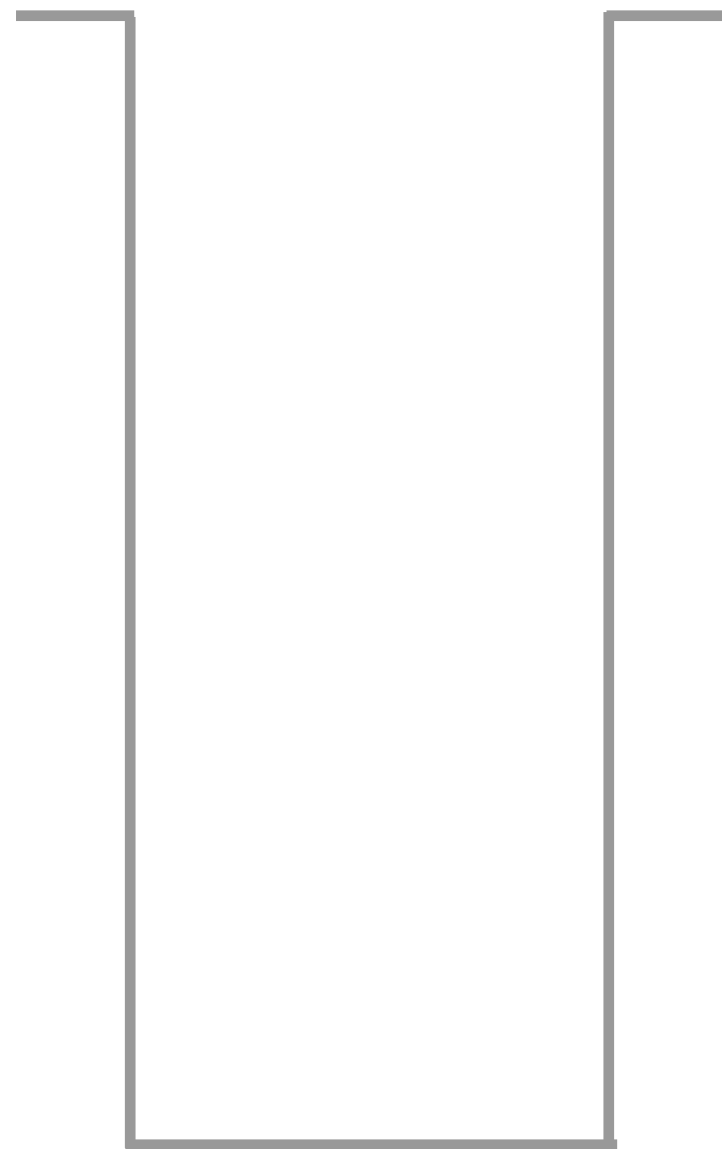
아직 닫히지 않은
'(' 를 저장한다.

()
pop

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 예시3



Stack

아직 닫히지 않은
'(' 를 저장한다.

pop 시도
()

스택이 **비어있는 상태**에서
닫는 괄호가 입력된다면
올바르지 않은 괄호 문자열이다.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습3] 올바른 괄호인지 판단하기 - 풀이

1. 괄호 문자열의 각 괄호를 **처음부터 하나씩 검토**한다.
2. **여는** 괄호라면 스택에 **넣고**, **닫는** 괄호라면 스택에서 괄호를 하나 **제거**한다.
3. 괄호 문자열을 **처리한 결과**에 따라 다음의 값을 반환한다.
 - (a) 괄호 문자열을 모두 처리하고 나서 **스택이 비어있다면** 올바른 괄호이다.
 - (b) (a)가 아니거나, **스택이 비어있는 상태**에서 **닫는 괄호**가 입력되었다면 올바른 괄호가 아니다.

03 스택으로 풀 수 있는 문제

✓ [실습4] 계산 순서 정하기

괄호가 포함된 수식이 주어질 때
괄호를 계산하는 순서를 구해보자.

입력 예시

```
((()))  
(()())  
(())( )  
( ) ( ) ( )
```

출력 예시

```
3 2 1 1 2 3  
3 1 1 2 2 3  
2 1 1 2 3 3  
1 1 2 2 3 3
```


03 스택으로 풀 수 있는 문제

✓ [실습4] 계산 순서 정하기

[실습3] 의 문제에서,
스택에 저장된 괄호가 pop되는 순서를 따로 저장해두면
[실습4] 도 풀 수 있다.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습4] 계산 순서 정하기

사실 [실습3]에서는 스택에 저장되는 자료가 어떤 값을 갖든 문제를 푸는 데 아무 영향을 끼치지 않는다.

스택에 저장되는 값이 여는 괄호가 아니라 다른 값이어도 스택이 정상적으로 push, pop이 되는 것을 확인할 수 있으면 된다.

03 스택으로 풀 수 있는 문제

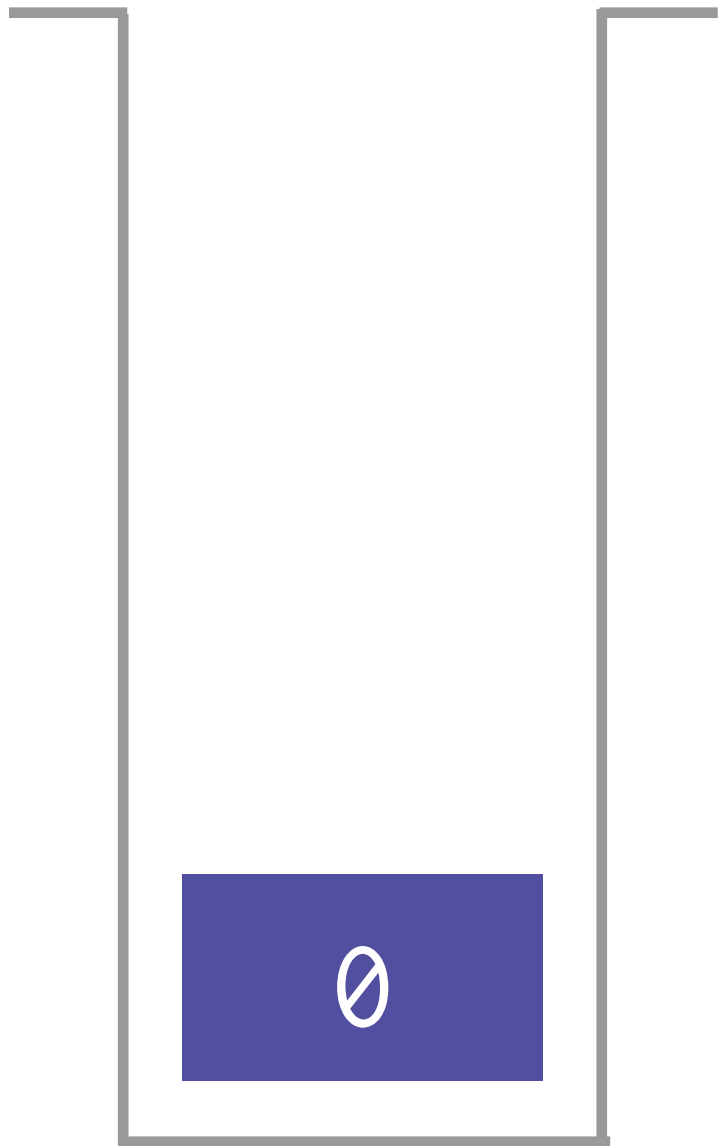
✓ [실습4] 계산 순서 정하기

이번에는 **닫는 괄호**가 입력될 때,
이 닫는 괄호에 **대응하는 여는 괄호가 어떤 괄호인지 특정**해야 할 필요가 있다.

따라서 스택에 저장하는 값을 '**여는 괄호의 인덱스**'로 하면
임의의 닫는 괄호에 대응하는 **여는 괄호가 어느 위치에 있는지** 알 수 있다.

03 스택으로 풀 수 있는 문제

✓ [실습4] 계산 순서 정하기



Stack

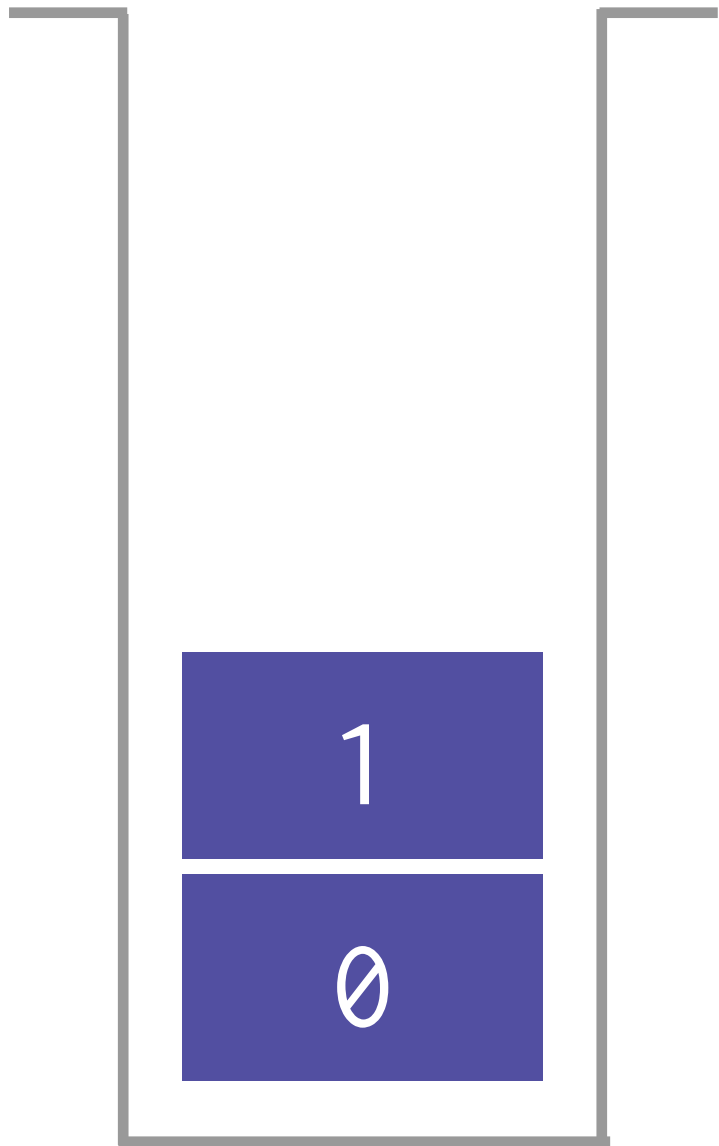
여는 괄호의
인덱스를 저장한다.

(

0	1	2	3	4	5

03 스택으로 풀 수 있는 문제

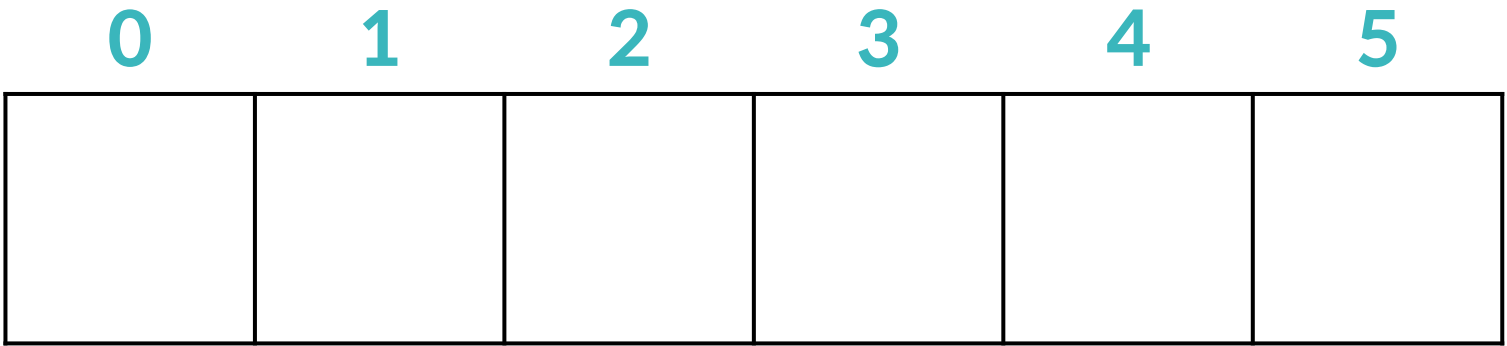
✓ [실습4] 계산 순서 정하기



Stack

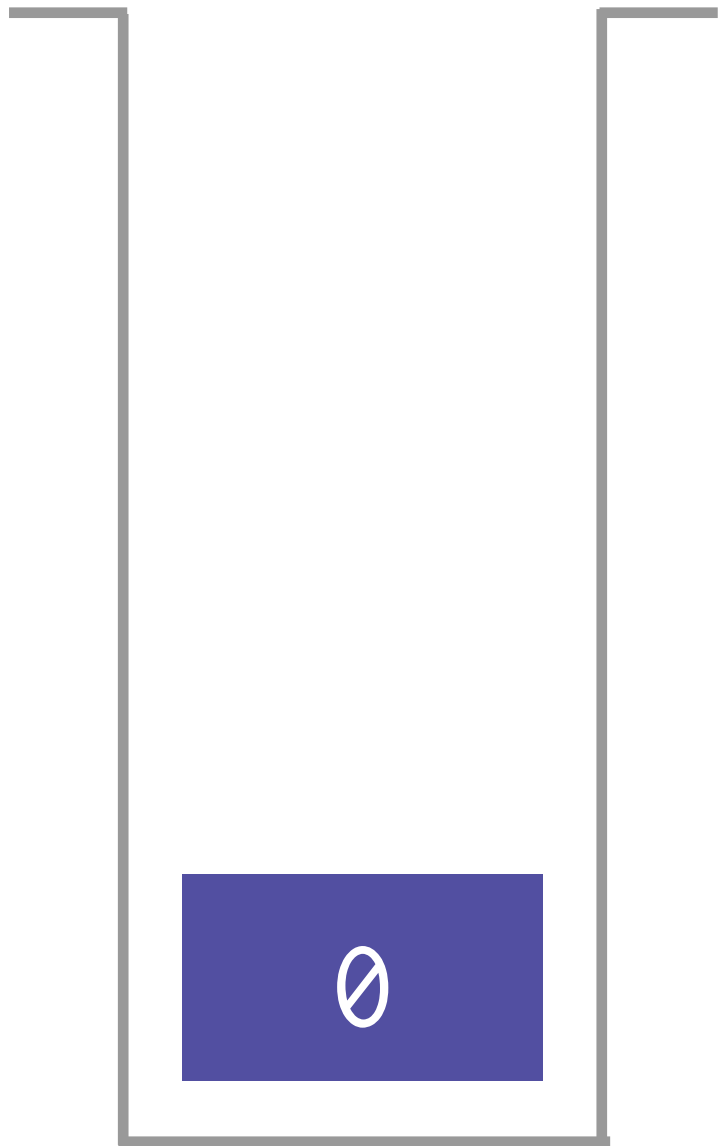
여는 괄호의
인덱스를 저장한다.

((



03 스택으로 풀 수 있는 문제

✓ [실습4] 계산 순서 정하기



Stack

여는 괄호의
인덱스를 저장한다.

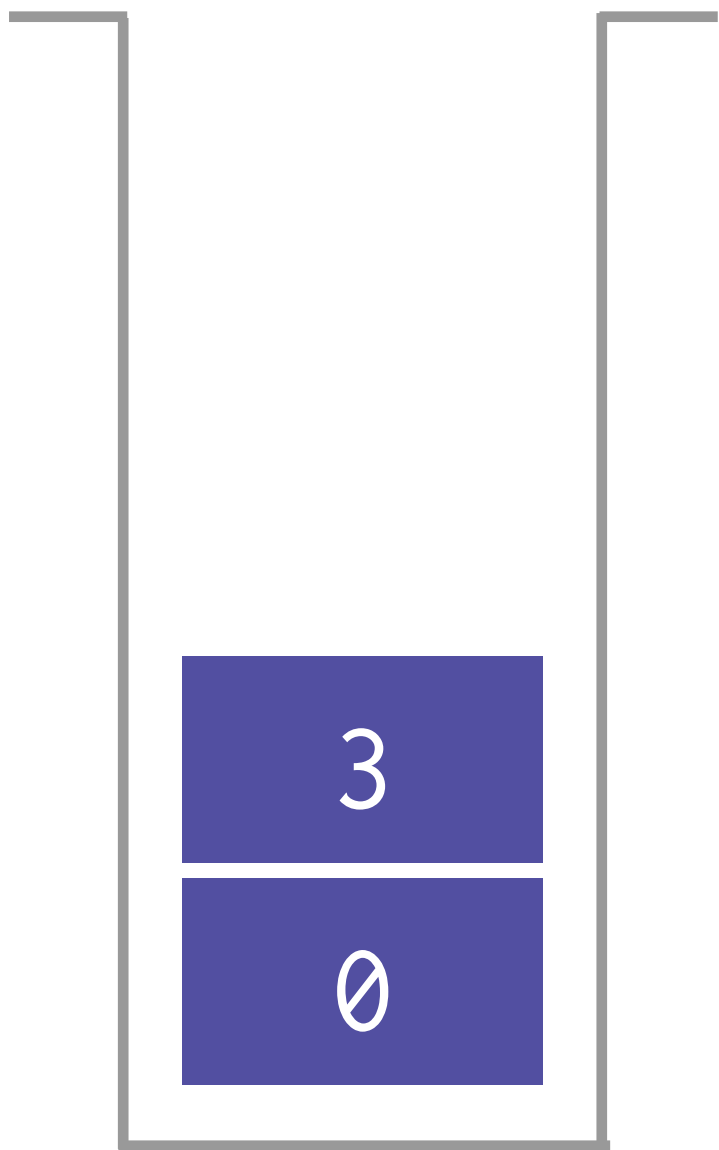
1이 pop되면서, 여는 괄호의 인덱스인 1과
현재 닫는 괄호의 인덱스인 2에 값을 기록한다.

(()

0	1	2	3	4	5
	1	1			

03 스택으로 풀 수 있는 문제

✓ [실습4] 계산 순서 정하기



Stack

여는 괄호의
인덱스를 저장한다.

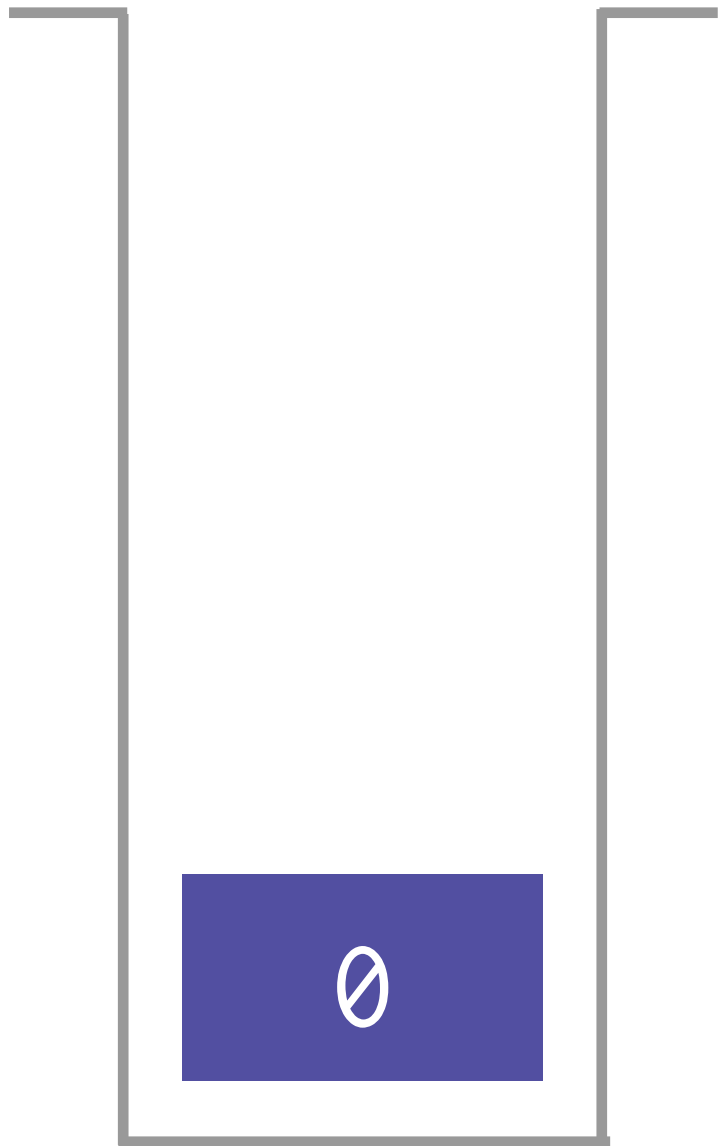
(() (

0	1	2	3	4	5
	1	1			

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습4] 계산 순서 정하기



Stack

여는 괄호의
인덱스를 저장한다.

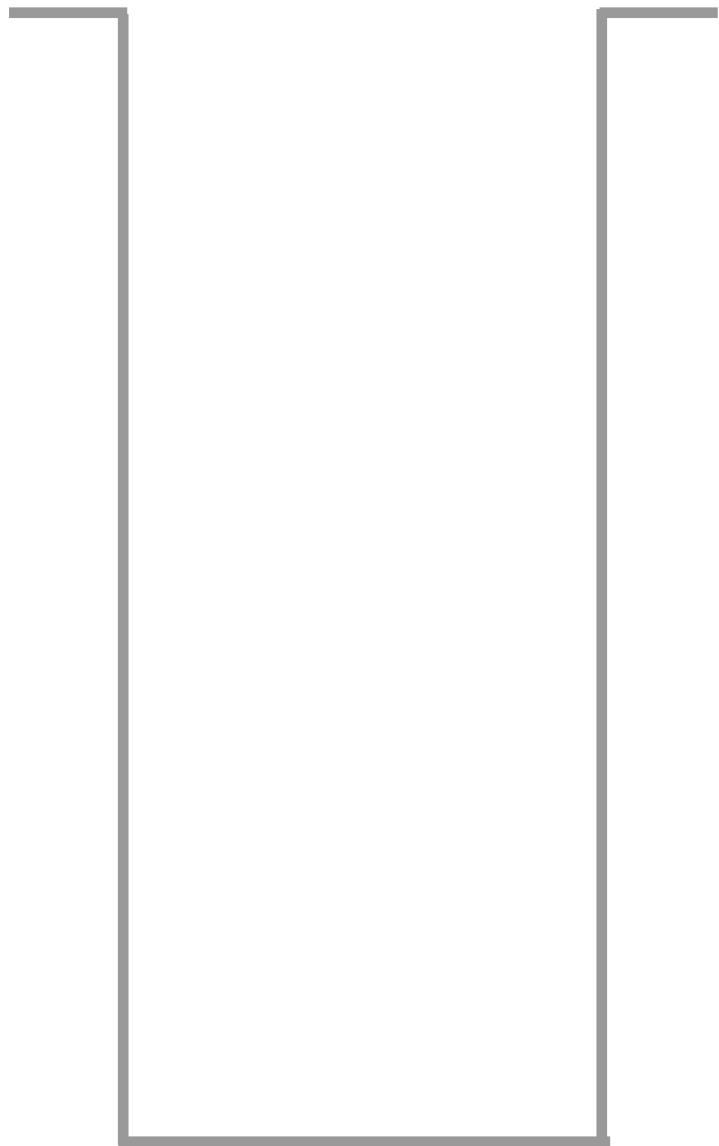
3이 pop되면서, **여는 괄호**의 인덱스인 **3**과
현재 **닫는 괄호**의 인덱스인 **4**에 값을 기록한다.

(())

0	1	2	3	4	5
	1	1	2	2	

03 스택으로 풀 수 있는 문제

✓ [실습4] 계산 순서 정하기



Stack

여는 괄호의
인덱스를 저장한다.

0이 pop되면서, **여는 괄호**의 인덱스인 **0**과
현재 **닫는 괄호**의 인덱스인 **5**에 값을 기록한다.

(() ())

0	1	2	3	4	5
3	1	1	2	2	3

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

스택을 이용하여 주어진 수열을 만들 수 있는지 없는지 검사해보자.

입력 예시

2 1 4 3

3 1 2 4

4 3 5 6 2 1

4 3 5 6 1 2

출력 예시

True

False

True

False

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

규칙

1부터 N 까지의 모든 정수를 **한 번씩** 스택에 입력하고, 출력한다.

이 때, 무조건 **작은 수는 큰 수보다 먼저** 입력되어야 한다.

예를 들어 1보다 3이 먼저 입력되는 경우는 없다.

위와 같은 규칙대로 수를 입출력했을 때 주어진 수열을 만들 수 있는지 없는지 알아보자.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[2, 1, 4, 3] : push, push, pop, pop, push, push, pop, pop

[3, 1, 4, 3] : 불가능

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

문제 풀이 방법

실제로 스택에 숫자들을 넣어보면서
가능한 경우인지 아닌지 확인해보면 된다.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[2, 1, 4, 3]을 만들 수 있는가?

입력해야 할 숫자 : 1, 2, 3, 4

출력된 숫자 :

Numbers

`/* elice */`

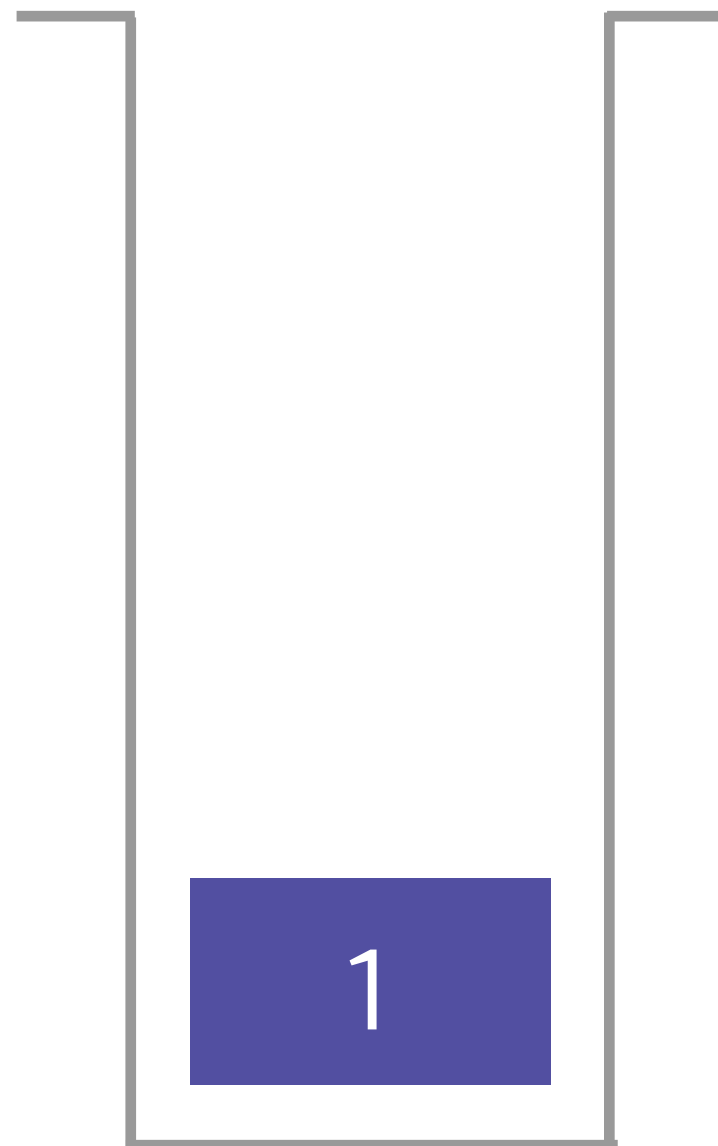
03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[2, 1, 4, 3]을 만들 수 있는가?

입력해야 할 숫자 : 2, 3, 4

출력된 숫자 :



Numbers

`/* elice */`

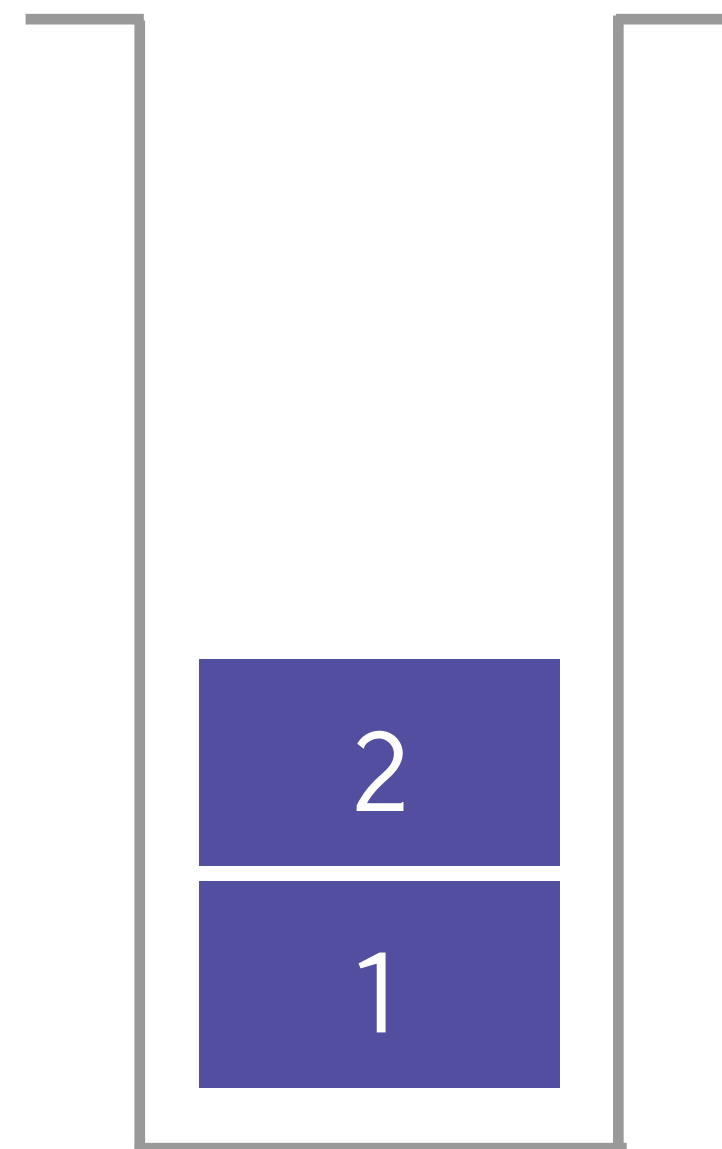
03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[2, 1, 4, 3]을 만들 수 있는가?

입력해야 할 숫자 : 3, 4

출력된 숫자 :



Numbers

`/* elice */`

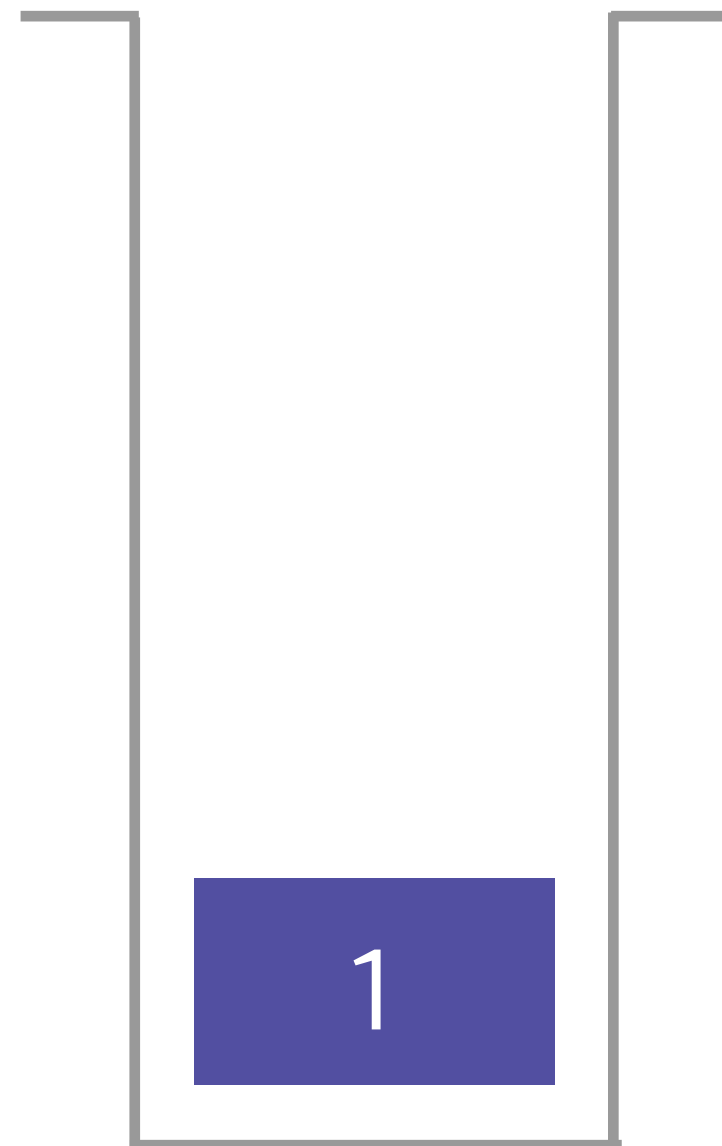
03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[2, 1, 4, 3]을 만들 수 있는가?

입력해야 할 숫자 : 3, 4

출력된 숫자 : 2



Numbers

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[2, 1, 4, 3]을 만들 수 있는가?

입력해야 할 숫자 : 3, 4

출력된 숫자 : 2, 1

Numbers

`/* elice */`

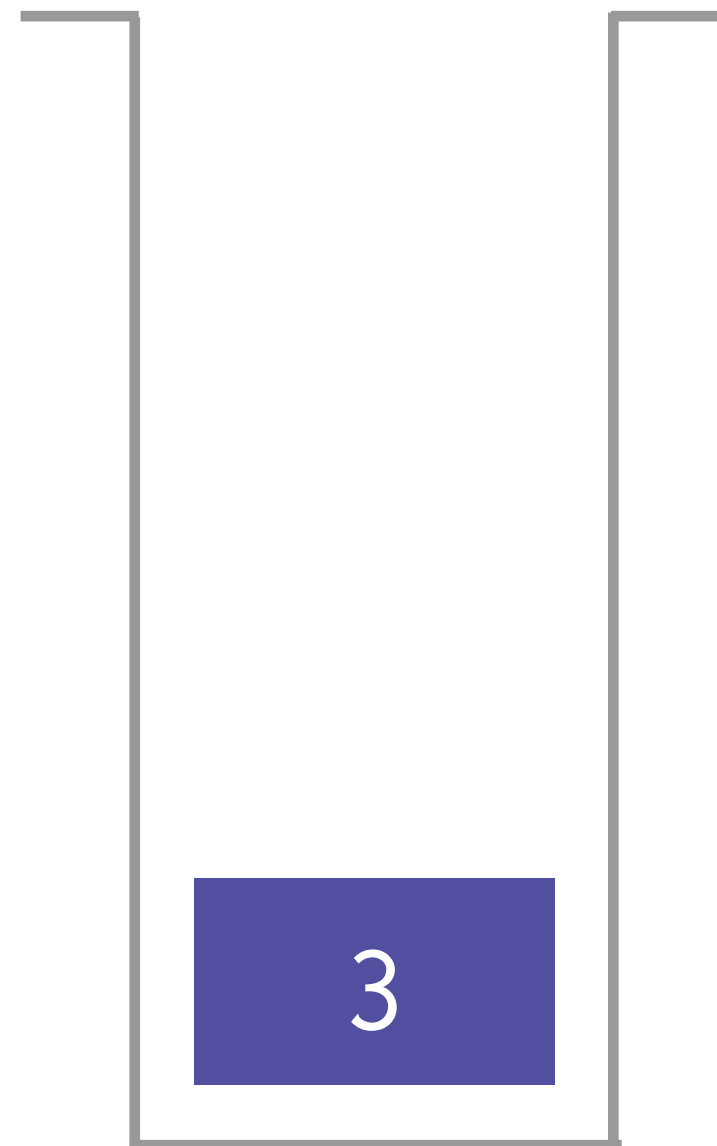
03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[2, 1, 4, 3]을 만들 수 있는가?

입력해야 할 숫자 : 4

출력된 숫자 : 2, 1



Numbers

`/* elice */`

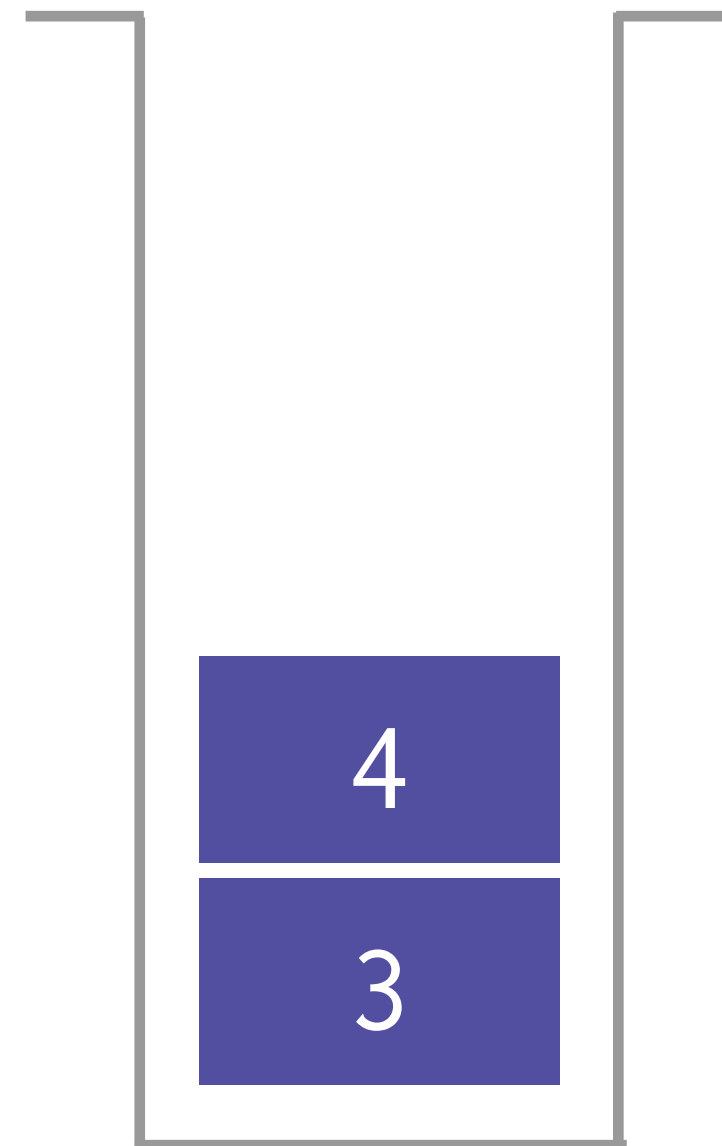
03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[2, 1, 4, 3]을 만들 수 있는가?

입력해야 할 숫자 :

출력된 숫자 : 2, 1



Numbers

`/* elice */`

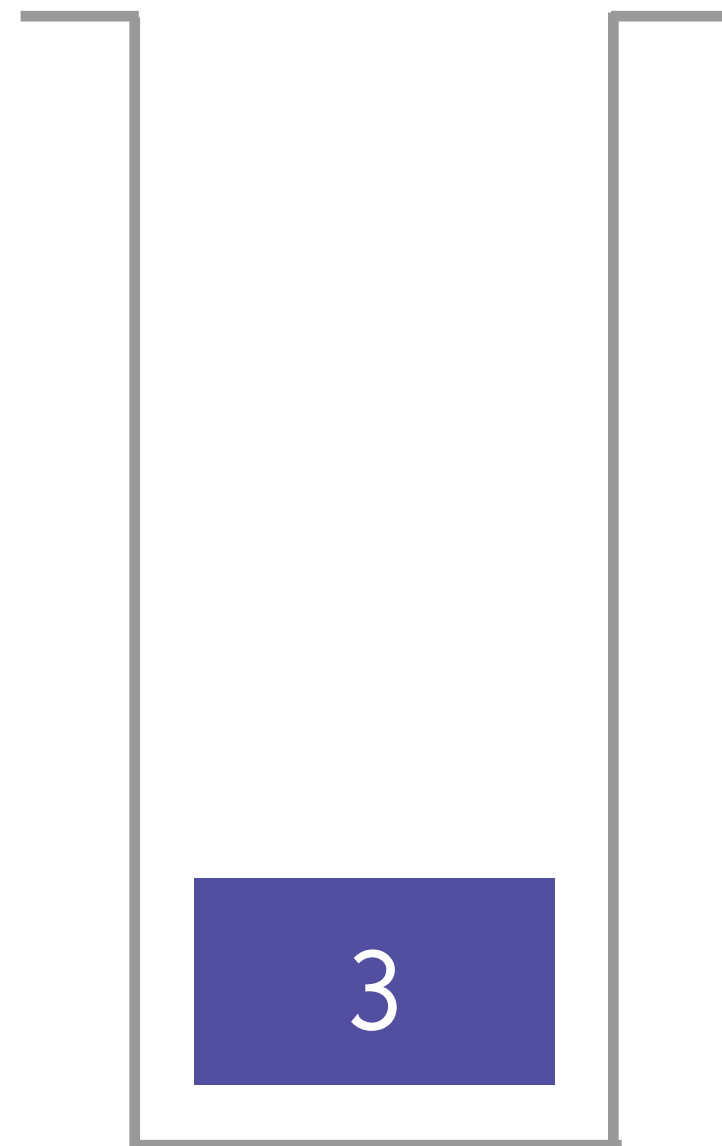
03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[2, 1, 4, 3]을 만들 수 있는가?

입력해야 할 숫자 :

출력된 숫자 : 2, 1, 4



Numbers

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[2, 1, 4, 3]을 만들 수 있는가?

입력해야 할 숫자 :

출력된 숫자 : 2, 1, 4, 3 (성공)

Numbers

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[3, 1, 2, 4]를 만들 수 있는가?

입력해야 할 숫자 : 1, 2, 3, 4

출력된 숫자 :

Numbers

`/* elice */`

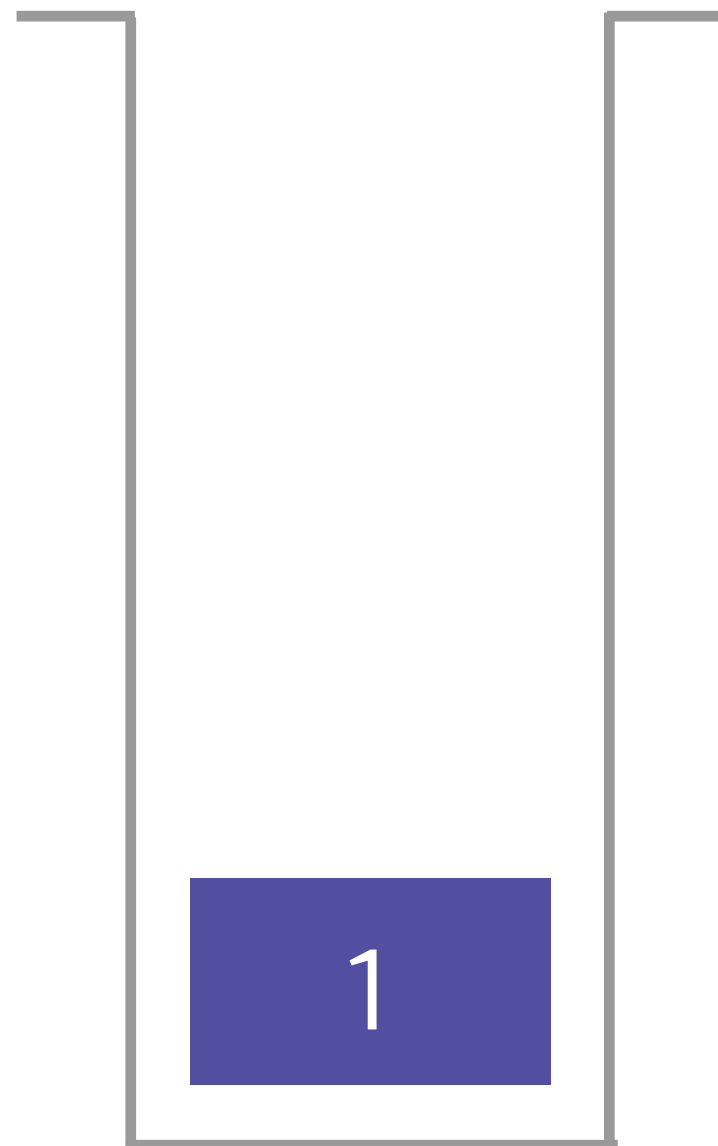
03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[3, 1, 2, 4]를 만들 수 있는가?

입력해야 할 숫자 : 2, 3, 4

출력된 숫자 :



Numbers

`/* elice */`

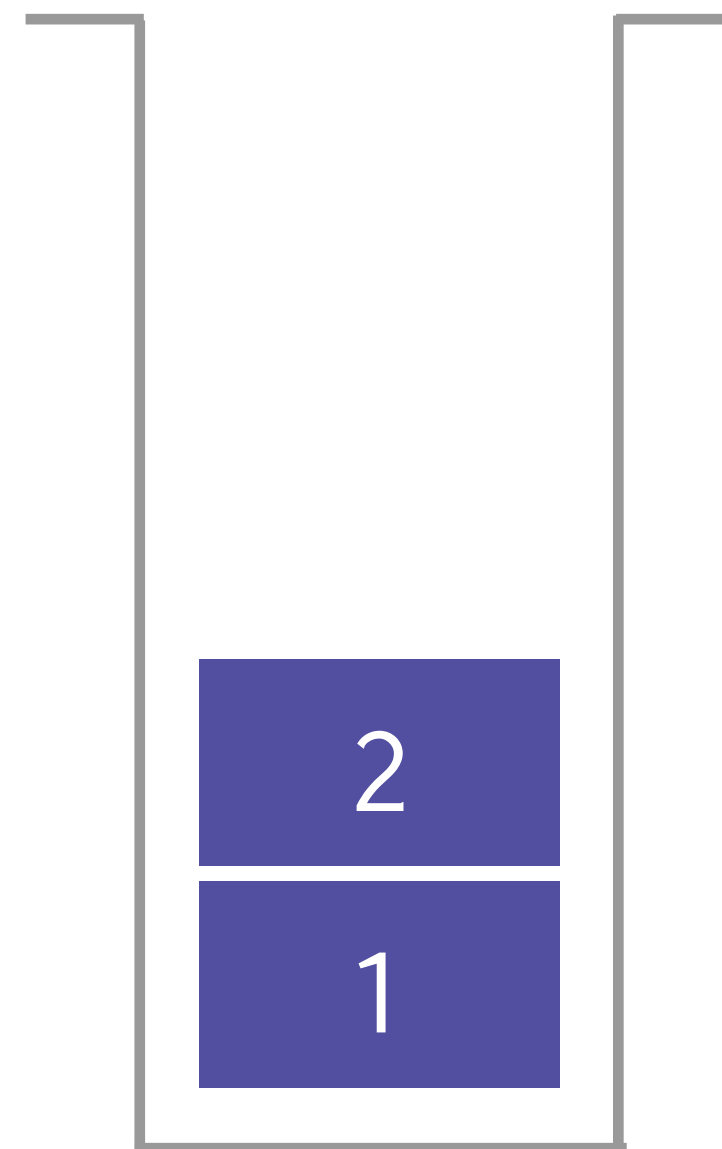
03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[3, 1, 2, 4]를 만들 수 있는가?

입력해야 할 숫자 : 3, 4

출력된 숫자 :



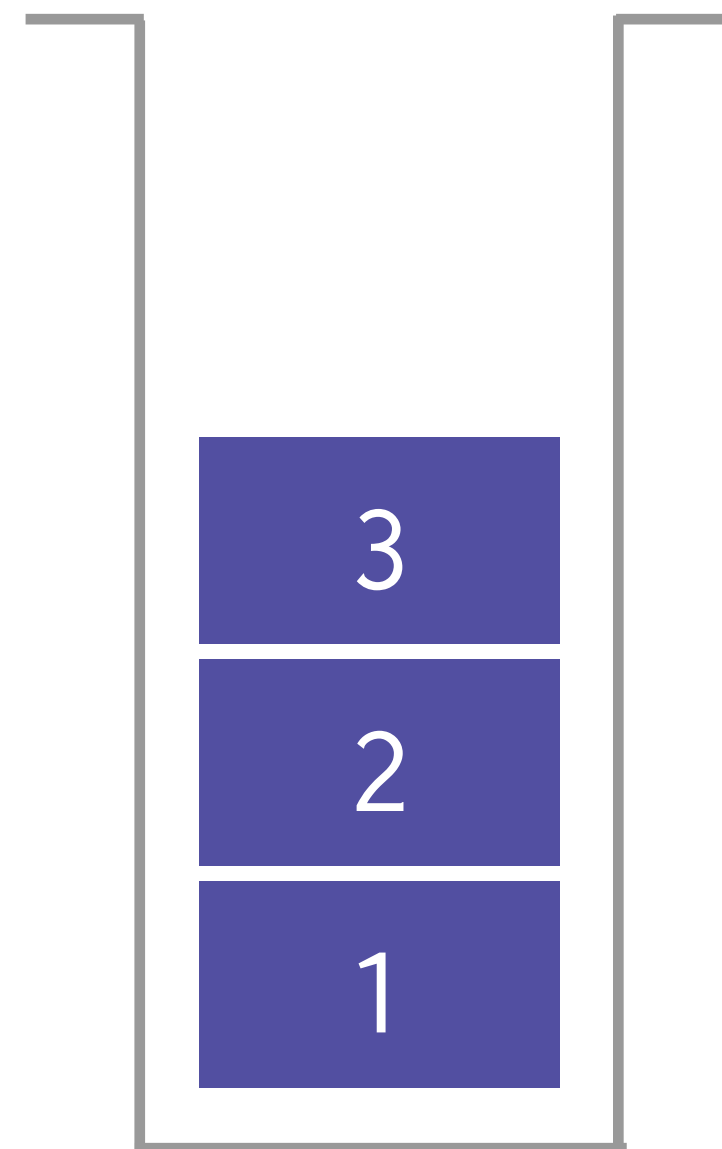
Numbers

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[3, 1, 2, 4]를 만들 수 있는가?



Numbers

입력해야 할 숫자 : 4

출력된 숫자 :

`/* elice */`

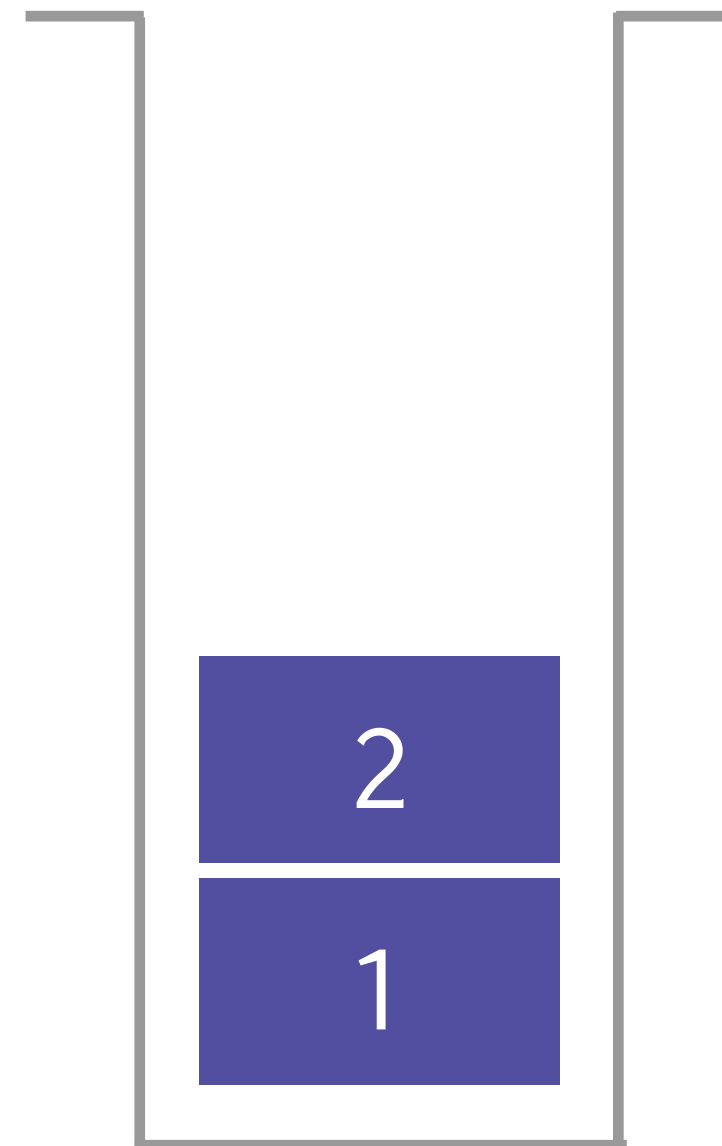
03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[3, 1, 2, 4]를 만들 수 있는가?

입력해야 할 숫자 : 4

출력된 숫자 : 3



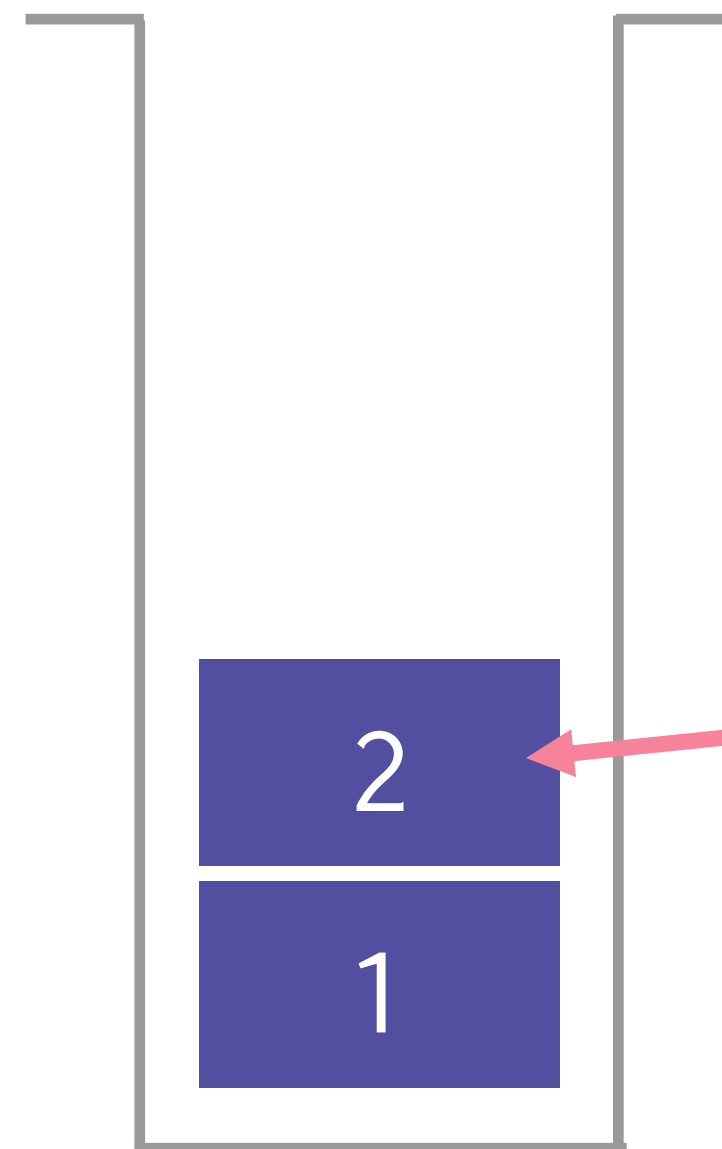
Numbers

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[3, 1, 2, 4]를 만들 수 있는가?



입력해야 할 숫자 : 4

1을 출력해야 하는데
스택의 맨 위에 2가 있는 상태이다.

출력된 숫자 : 3

Numbers

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

숫자를 출력할 땐 아래와 같은 2가지 경우가 있다.

1. 출력해야 하는 숫자가 추가해야 하는 숫자보다 **큰** 경우
2. 출력해야 하는 숫자가 추가해야 하는 숫자보다 **작은** 경우

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

1의 경우에는 간단하다.

출력을 목표로 하는 숫자가 스택에 들어갈 때까지 **스택에 계속 추가**해주면 된다.

[2, 1, 4, 3]의 **2**를 출력하기 위해 **1, 2**를 스택에 넣어주었던 것과 같다.

`/* elice */`

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

2의 경우에는는 또 두 가지 경우로 나뉜다.

스택의 맨 위의 값이 현재 출력해야 하는 숫자인 경우 → 그냥 출력

스택의 맨 위의 값이 현재 출력해야 하는 숫자가 아닌 경우 → 실패

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습5] 스택으로 수열 만들기

[3, 1, 2, 4]에서 1을 출력해야 할 때
추가할 다음 숫자는 4였고, 스택의 맨 위의 값은 2였다.
따라서 [3, 1, 2, 4]는 규칙대로 출력할 수 있는 수열이 아니다.

03 스택으로 풀 수 있는 문제

✓ [실습6] 메모장

알파벳으로 된 한 문장을 저장할 수 있는 메모장을 만들어보자.

입력 예시

```
abcdef # origin text
3      # numbers of command
P g    # put alphabet 'g'
L      # move cursor to left
P z    # put alphabet 'z'
```

출력 예시

```
abcdefzg
```

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습6] 메모장

메모장이 처리할 수 있는 명령어는 아래 4가지이다.

L 커서를 왼쪽으로 한 칸 이동

R 커서를 오른쪽으로 한 칸 이동

D 커서 왼쪽에 있는 문자 하나 삭제

P s 임의의 알파벳 s를 커서 왼쪽에 추가

03 스택으로 풀 수 있는 문제

✓ [실습6] 메모장

a b c | d e f

커서를 기준으로 왼쪽, 오른쪽으로 분리해서 생각해보자.
왼쪽 문자들의 개수를 *left*, 오른쪽 문자들의 개수를 *right*라고 한다.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습6] 메모장

a b | c d e f

커서를 **왼쪽**으로 움직이면
*left*가 1 **감소**하고, *right*가 1 **증가**한다.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습6] 메모장

a b c d | e f

반대로, 커서를 **오른쪽**으로 움직이면
*left*가 1 **증가**하고, *right*가 1 **감소**한다.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습6] 메모장

a b c | d e f

커서가 정중앙에 있었을 때 각각 **L**, **R** 명령어를 수행한 결과
왼쪽에서는 **c**가, 오른쪽에서는 **d**가 각각 반대편으로 이동했다.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습6] 메모장

a b c | d e f

커서가 **왼쪽**으로 계속 이동한다고 할 때,
항상 **a, b보다 c가** 오른쪽으로 먼저 이동한다.
c가 커서에 더 가깝기 때문이다.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습6] 메모장

a b c | d e f

마찬가지로 **d**가 **e, f**보다
항상 더 먼저 왼쪽으로 이동한다.

/* elice */

03 스택으로 풀 수 있는 문제

✓ [실습6] 메모장

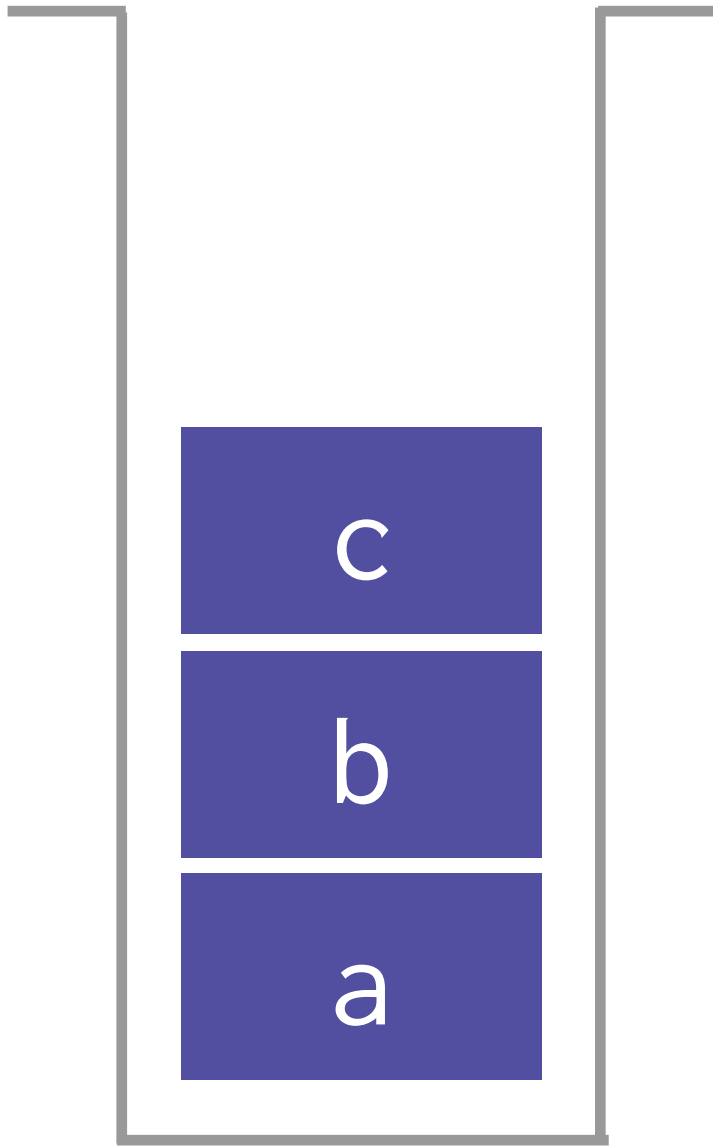
a b c | d e f

따라서 커서를 기준으로 **왼쪽 문자열**과 **오른쪽 문자열**을
스택으로 생각할 수 있다.

/* elice */

03 스택으로 풀 수 있는 문제

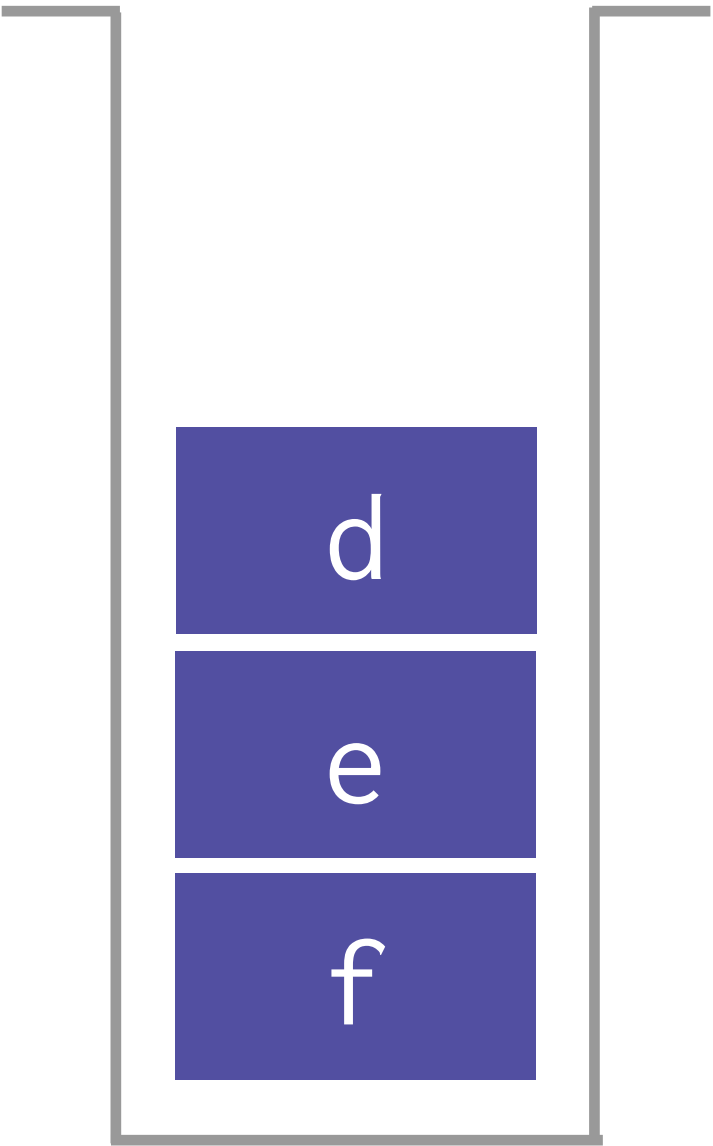
✓ [실습6] 메모장



Left

a b c | d e f

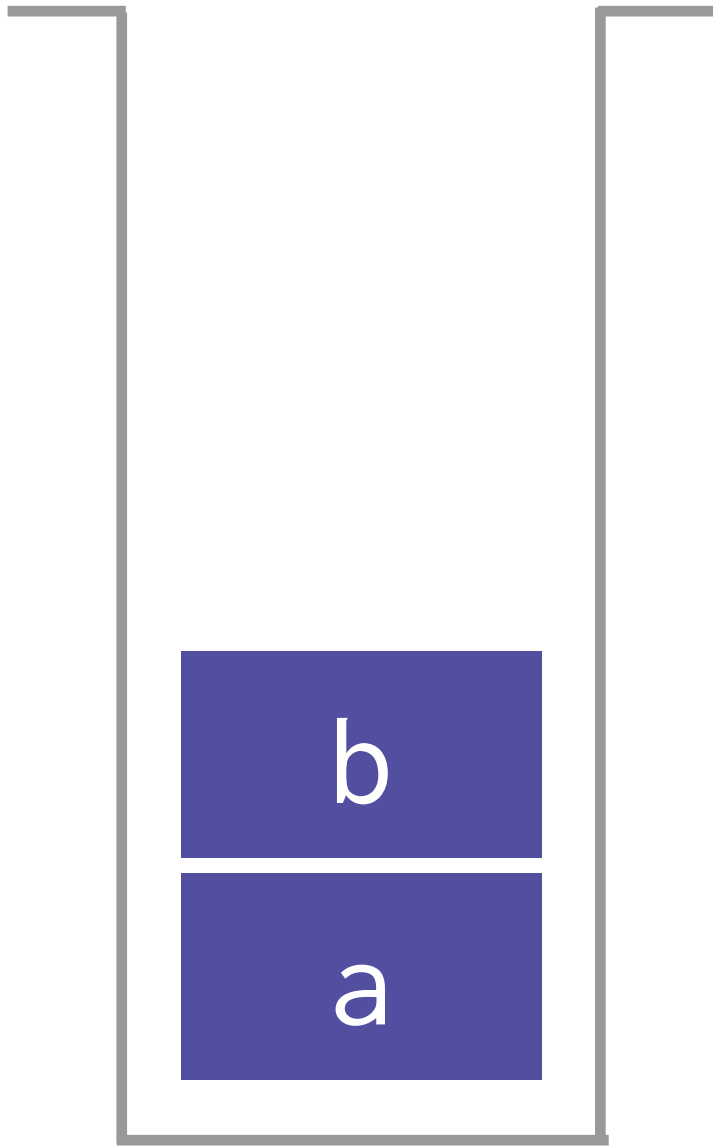
명령어 목록



Right

03 스택으로 풀 수 있는 문제

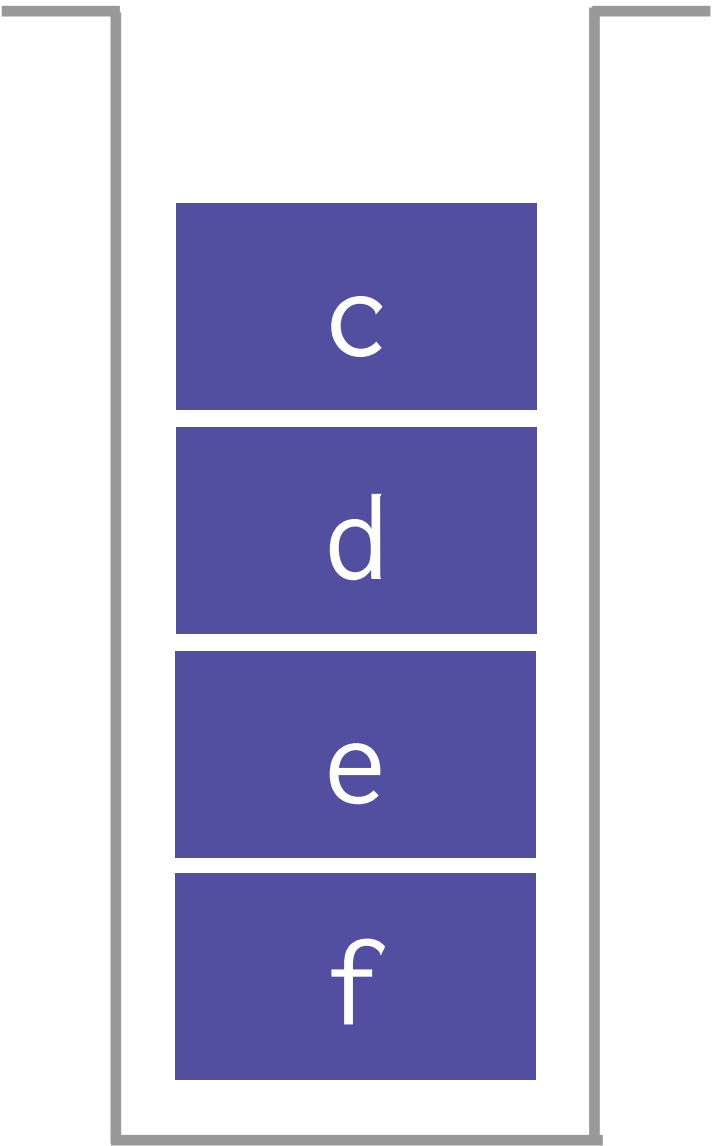
✓ [실습6] 메모장



Left

a b | c d e f

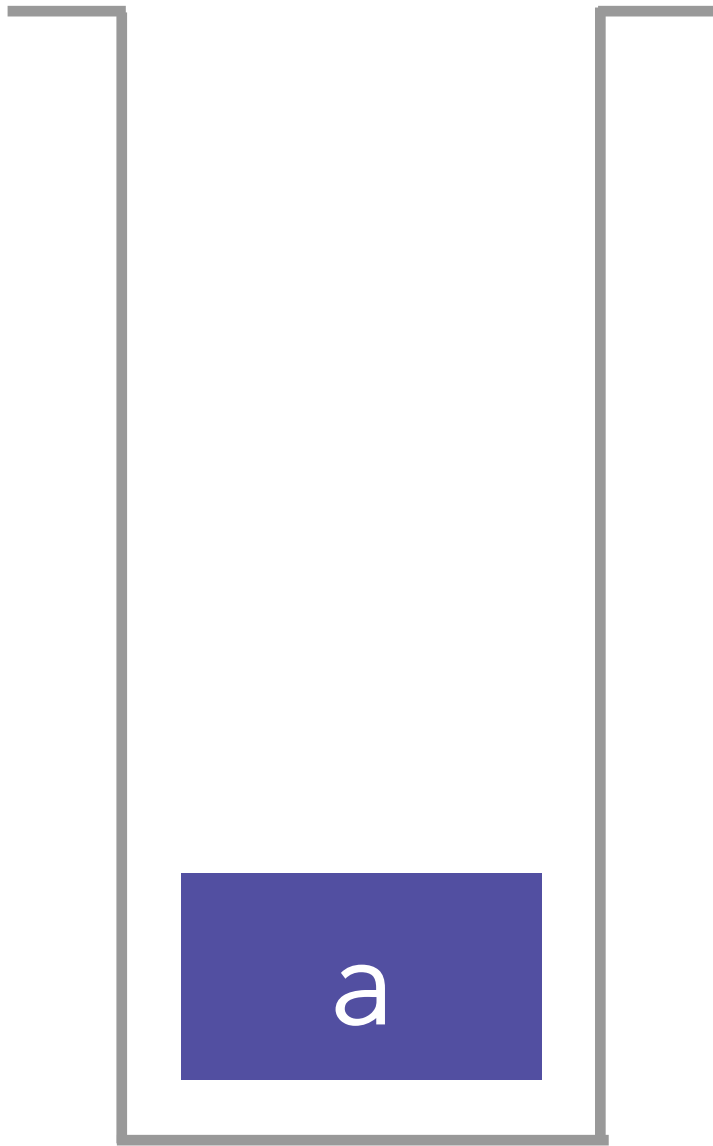
명령어 목록
L



Right

03 스택으로 풀 수 있는 문제

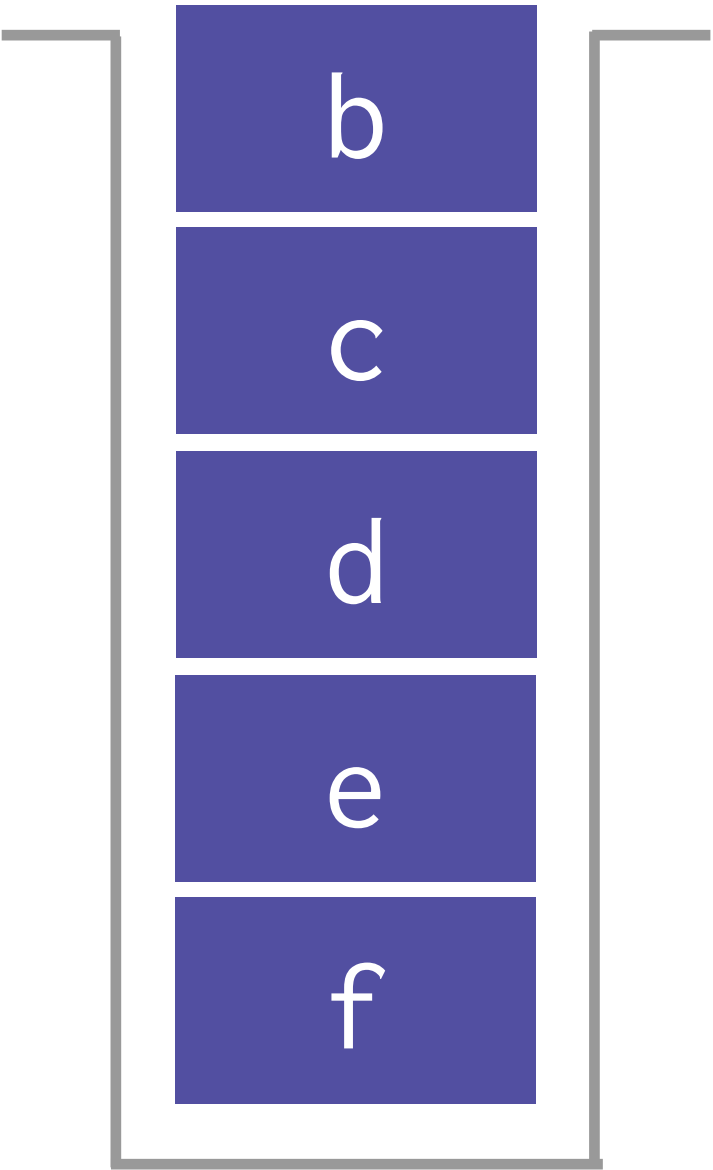
✓ [실습6] 메모장



Left

a | b c d e f

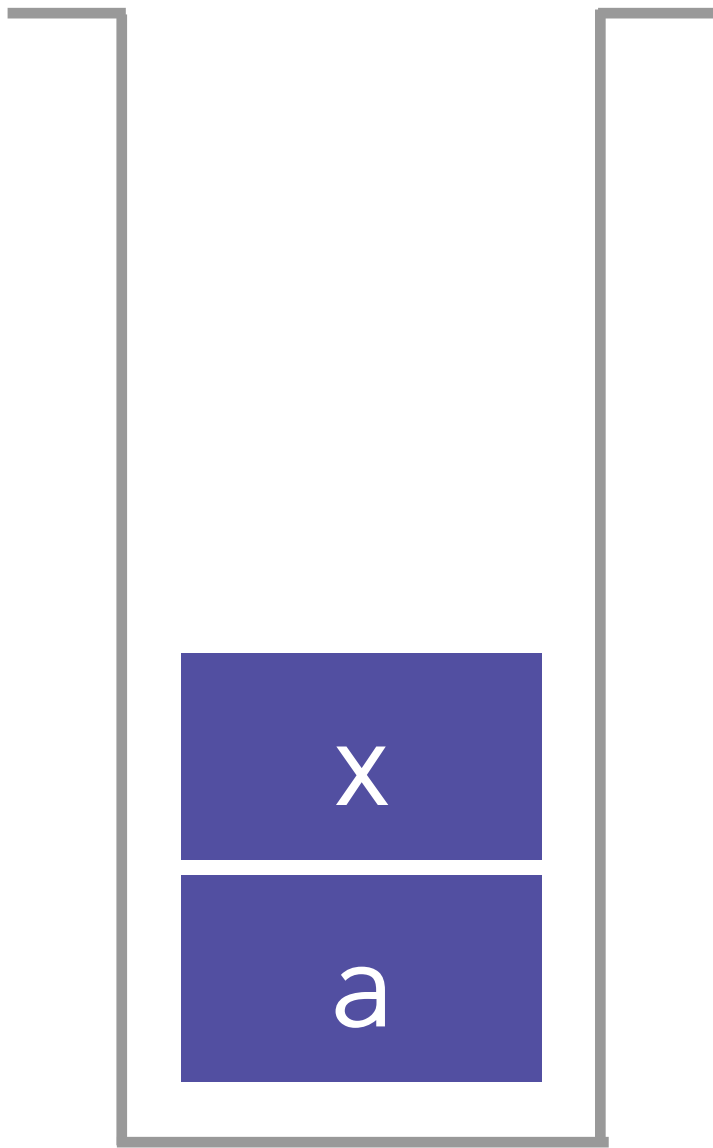
명령어 목록
L
L



Right

03 스택으로 풀 수 있는 문제

✓ [실습6] 메모장



Left

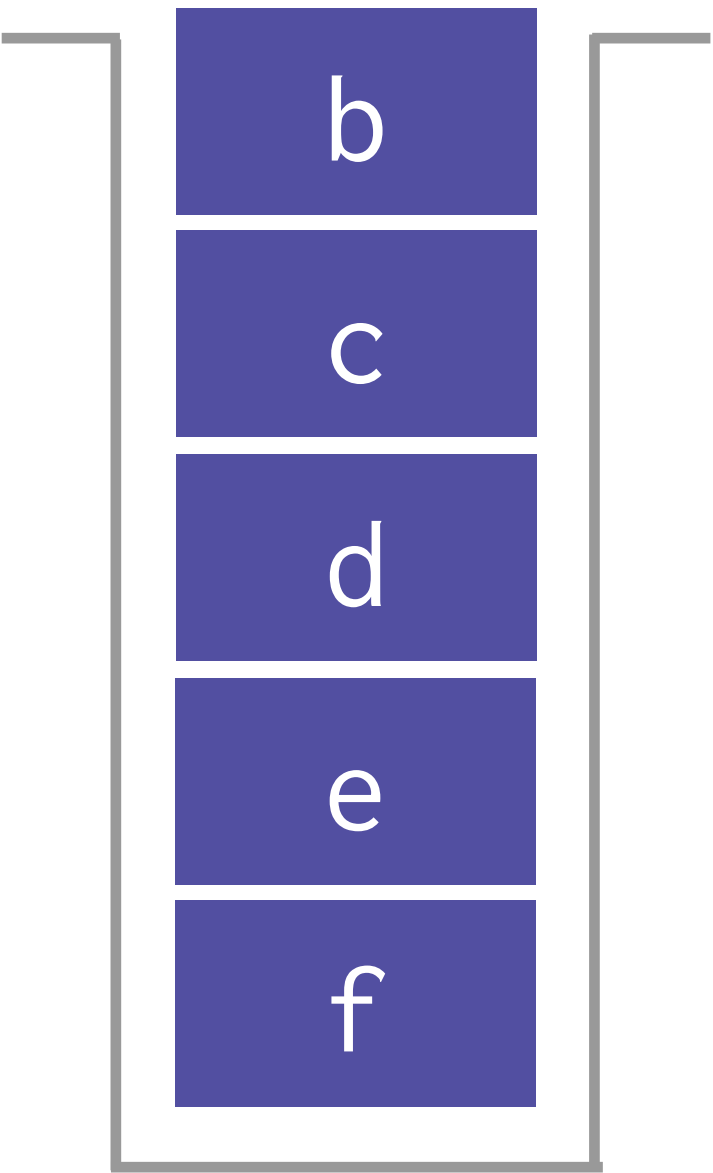
a x | b c d e f

명령어 목록

L

L

P x

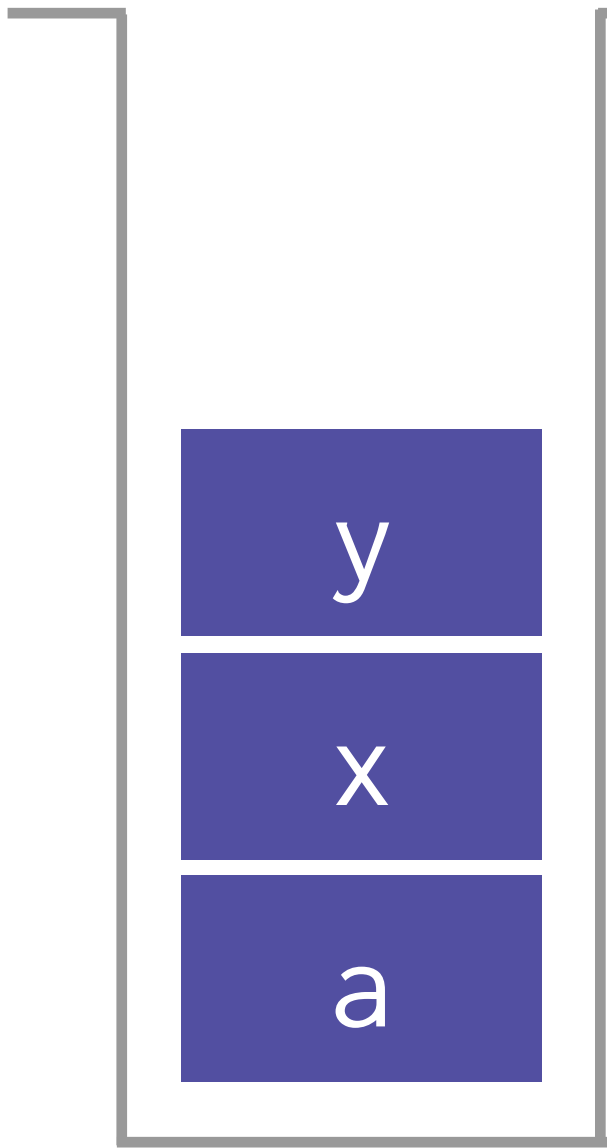


Right

/* elice */

03 스택으로 풀 수 있는 문제

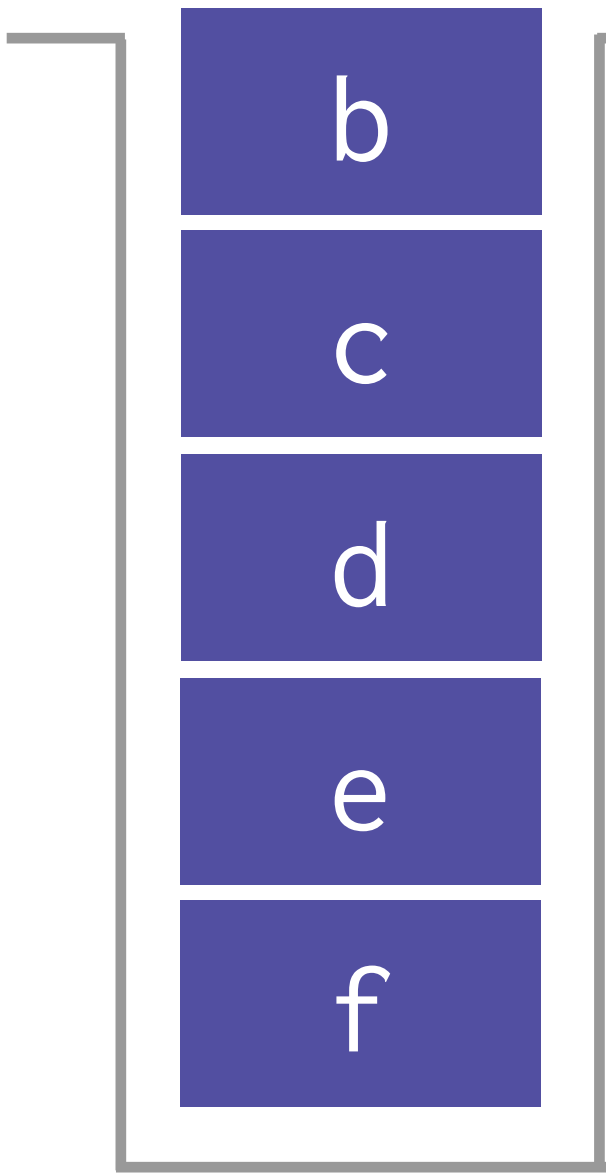
✓ [실습6] 메모장



Left

a x y | b c d e f

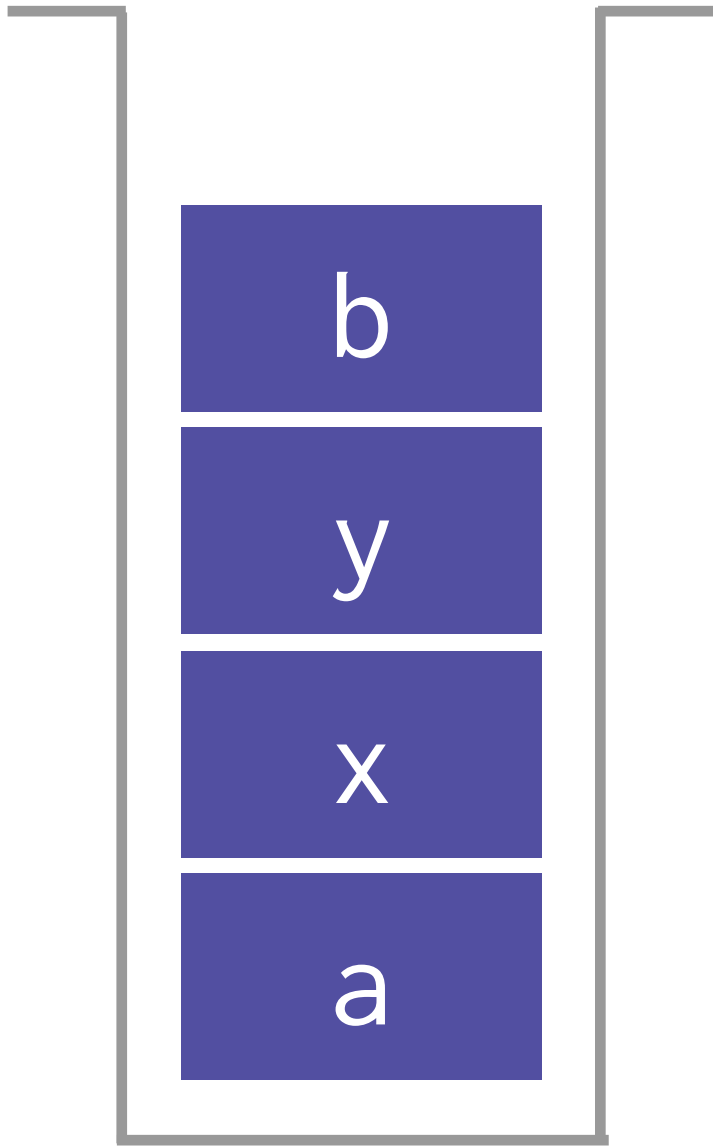
명령어 목록
L
L
P x
P y



Right

03 스택으로 풀 수 있는 문제

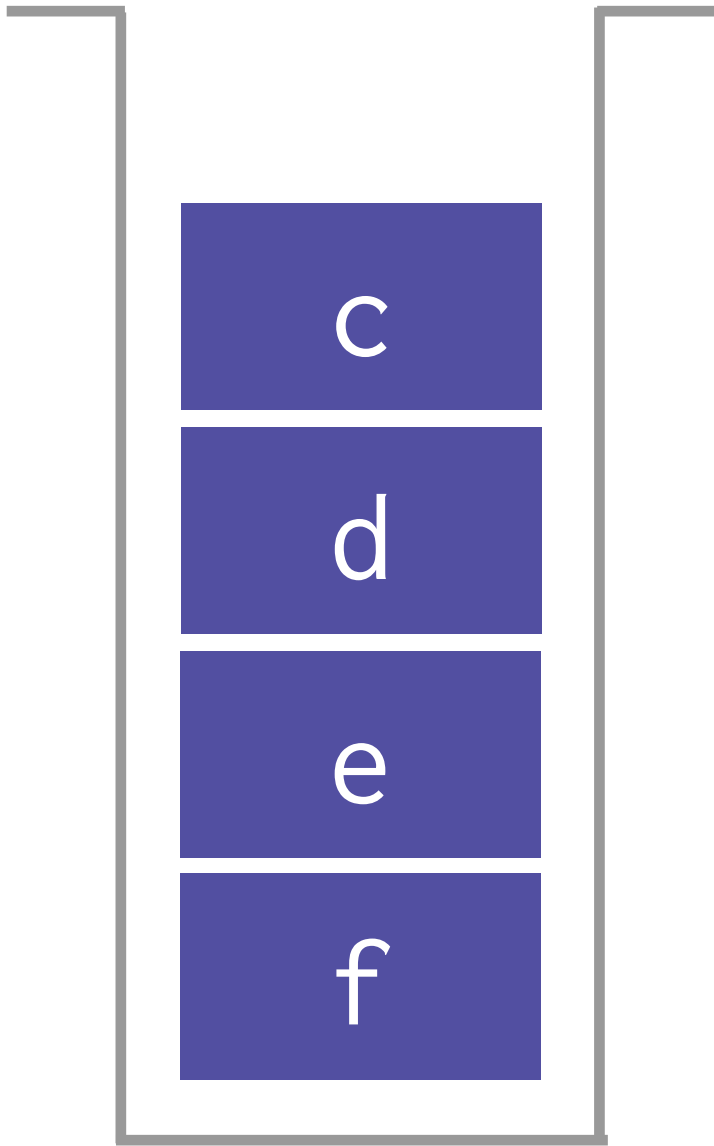
✓ [실습6] 메모장



Left

a x y b | c d e f

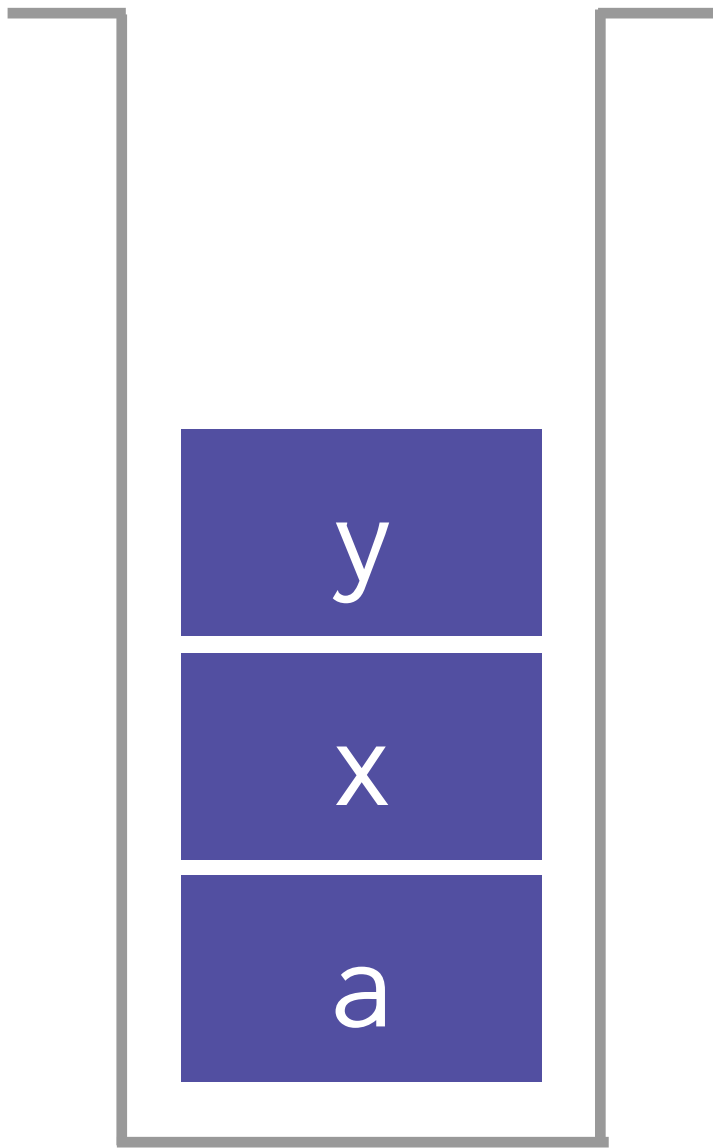
명령어 목록
L
L
P x
P y
R



Right

03 스택으로 풀 수 있는 문제

✓ [실습6] 메모장

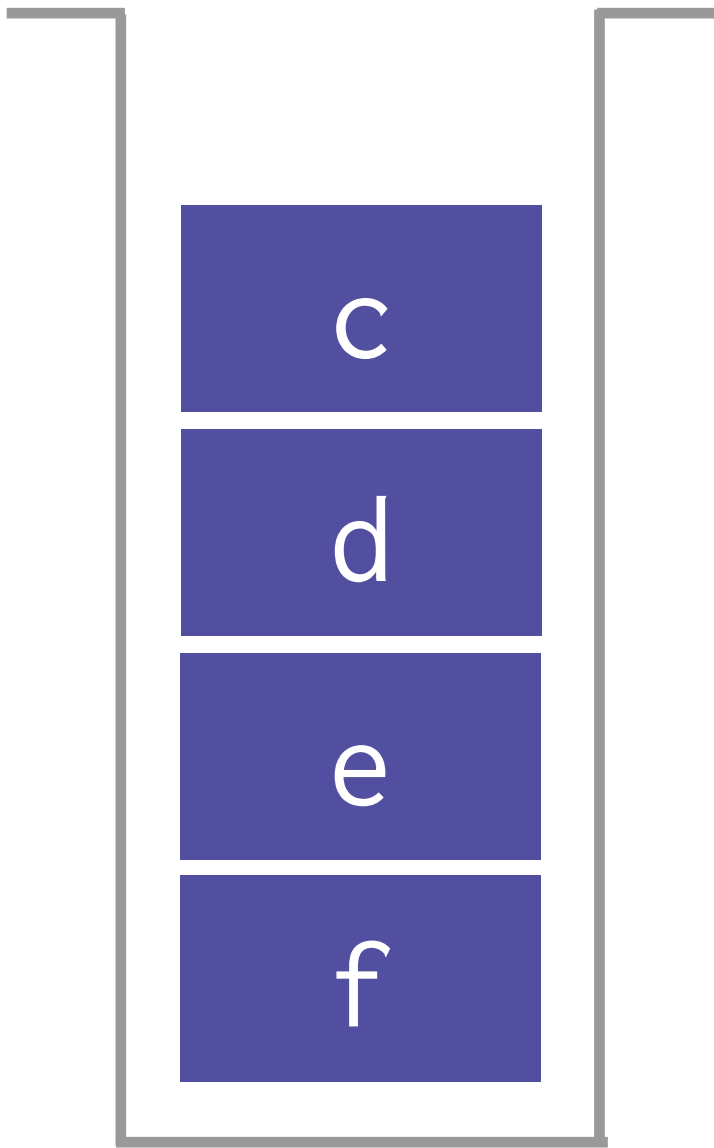


Left

a x y | c d e f

명령어 목록

L
L
P x
P y
R
D



Right

/* elice */

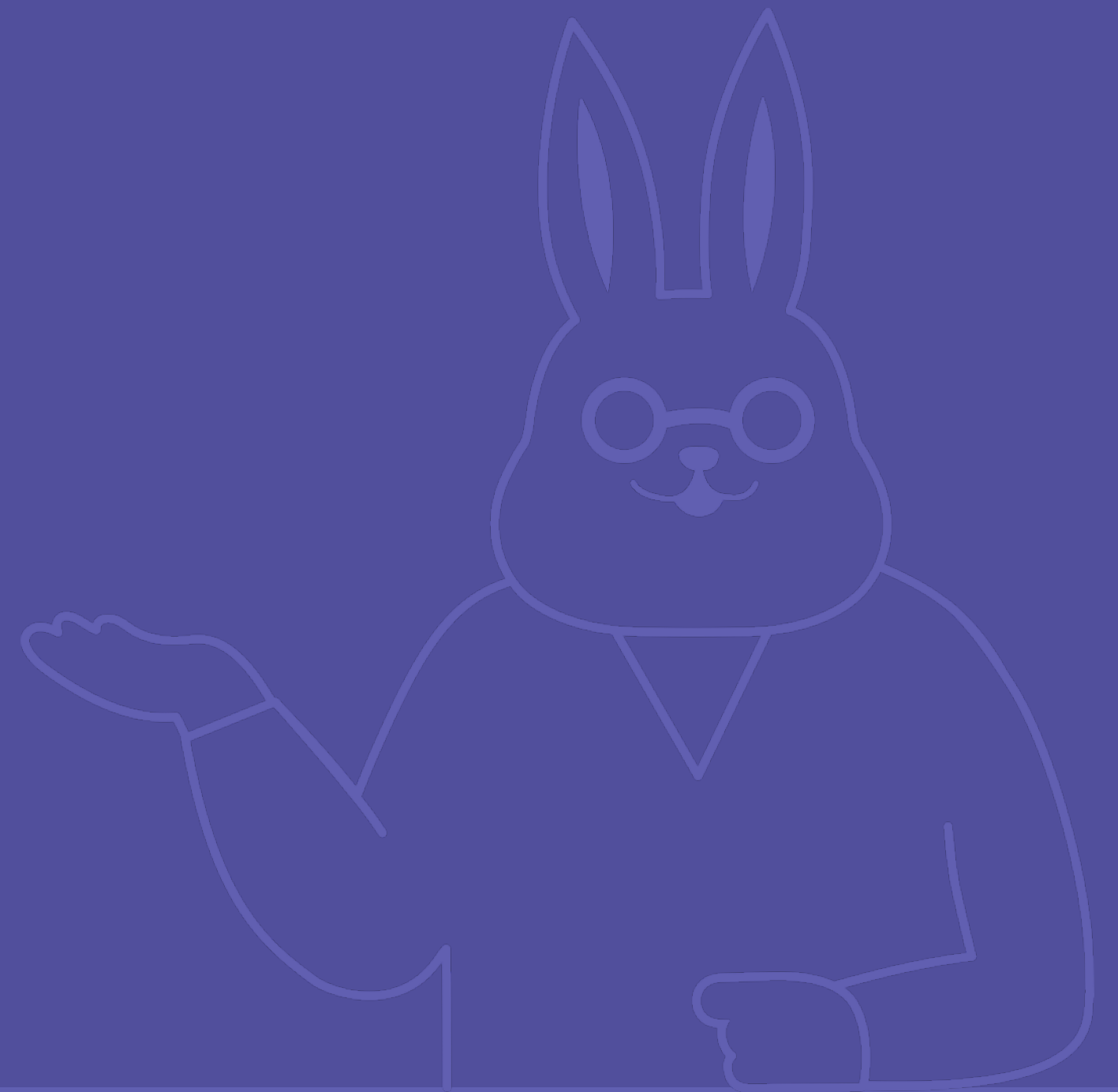
03 스택으로 풀 수 있는 문제

✓ [실습6] 메모장 - 정리

1. 커서를 기준으로 왼쪽 문자열과 오른쪽 문자열을 각각 스택으로 저장한다.
2. 각 명령어에 대해 아래와 같이 **스택**을 사용한다.
 - L : **왼쪽** 스택의 알파벳을 **제거**하고, 그 알파벳을 **오른쪽** 스택에 **추가**
 - R : **오른쪽** 스택의 알파벳을 **제거**하고, 그 알파벳을 **왼쪽** 스택에 **추가**
 - D : **왼쪽** 스택의 알파벳 **제거**
 - P : **왼쪽** 스택에 알파벳 **추가**

04

큐로 풀 수 있는 문제



04 큐로 풀 수 있는 문제

✓ [실습7] 주문 처리하기

고객의 주문을 들어온 순서대로 처리해보자.
단, VIP의 주문은 일반 주문보다 우선순위가 높다.

주문에 대한 정보는 (t, d) 의 형태로 주어지며
 t 는 주문이 접수되는 시각을,
 d 는 주문을 처리하는 데 드는 시간을 의미한다.

04 큐로 풀 수 있는 문제

✓ [실습7] 주문 처리하기

일반 주문

1, 3

3, 3

5, 3

6, 3

VIP 주문

7, 3

12, 3

현재 시각 : 0

04 큐로 풀 수 있는 문제

✓ [실습7] 주문 처리하기

일반 주문

3, 3

5, 3

6, 3

VIP 주문

7, 3

12, 3

현재 시각 : 4

04 큐로 풀 수 있는 문제

✓ [실습7] 주문 처리하기

일반 주문

5, 3

6, 3

VIP 주문

7, 3

12, 3

현재 시각 : 7

04 큐로 풀 수 있는 문제

✓ [실습7] 주문 처리하기

일반 주문

5, 3

6, 3

VIP 주문

12, 3

현재 시각 :

10

VIP의 주문을 먼저 처리한다

04 큐로 풀 수 있는 문제

✓ [실습7] 주문 처리하기

일반 주문

6, 3

VIP 주문

12, 3

현재 시각 : 13

04 큐로 풀 수 있는 문제

✓ [실습7] 주문 처리하기

일반 주문

6, 3

VIP 주문

현재 시각 : 16

04 큐로 풀 수 있는 문제

✓ [실습7] 주문 처리하기

일반 주문

VIP 주문

현재 시각 : 19

04 큐로 풀 수 있는 문제

✓ [실습7] 주문 처리하기

1, 3 3, 3 7, 3 5, 3 12, 3 6, 3

최종적으로 위와 같은 순서대로 주문을 처리하게 된다.

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

입력된 수에 대한 **요세푸스 순열**을 출력해보자.

입력 예시

```
7 3
6 2
```

출력 예시

```
3 6 2 7 5 1 4
2 4 6 3 1 5
```

`/* elice */`

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

요세푸스 순열이란?

고대 예루살렘에서 살았던 요세푸스는 로마와의 전쟁에 참전했는데,
자신을 포함한 동료들이 로마군에 포위되었다.

동료들은 로마군에 붙잡히느니 자결을하기로 하였으나,
종교적 이유로 직접 자결할 수는 없어 서로가 서로를 죽이고 마지막 남은 병사가 항복하기로 하였다.

요세푸스는 동료들의 뜻과는 달리 살고 싶었고,
끝까지 살아남기 위한 방법을 생각했다.

/* elice */

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

요세푸스 순열이란?

1번부터 N 번까지 번호를 가진 N 명의 사람들이 있다.

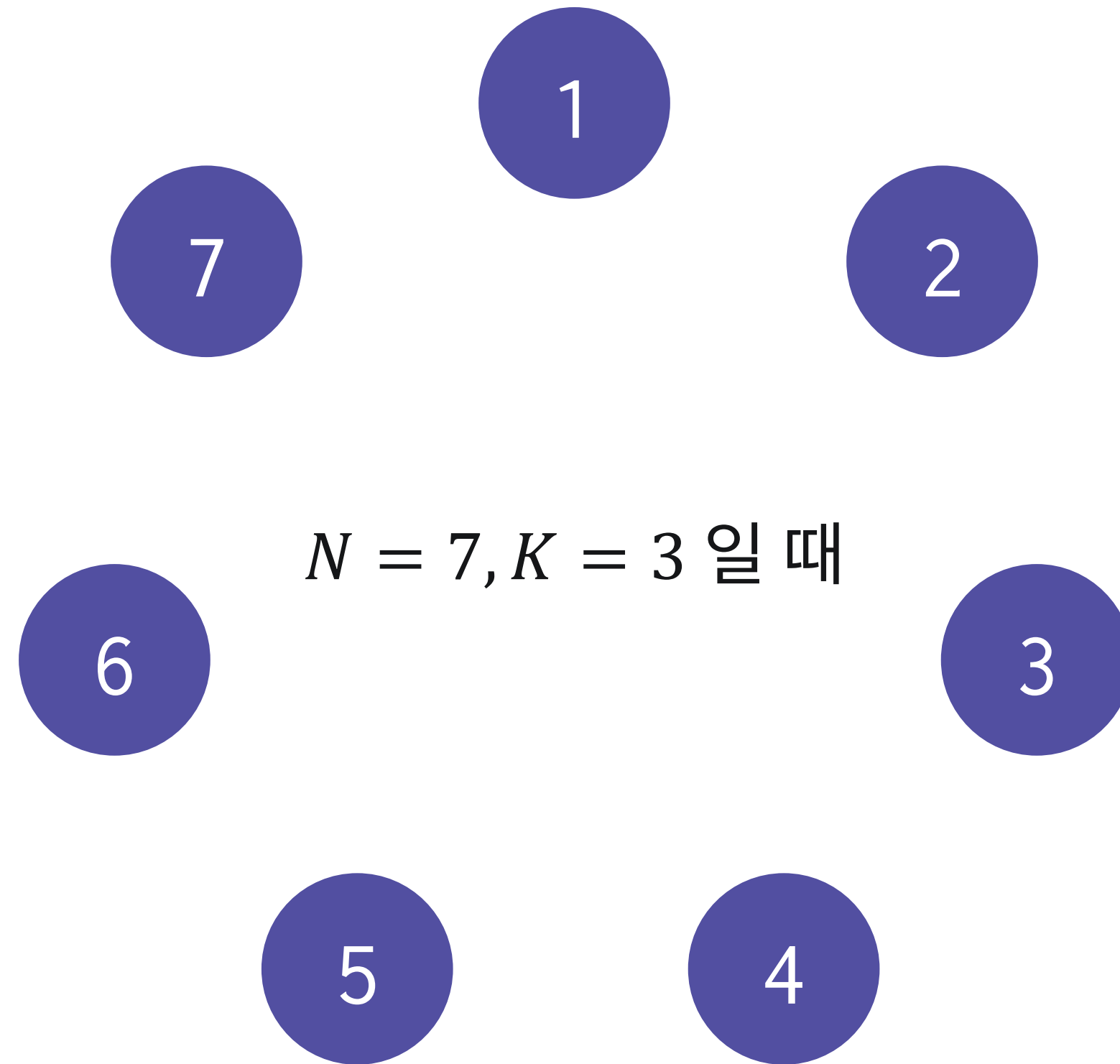
1번부터 순서대로 세었을 때 K 번째 사람을 원에서 제거한다.

그리고 나서, 제거된 사람의 다음 사람부터 K 번째 사람을 원에서 또 제거한다.

위 과정을 반복하면서 원을 이루고 있는 사람들이 모두 제거되었을 때,
제거된 사람들의 번호를 순서대로 나열한 것을 요세푸스 순열이라고 한다.

04 큐로 풀 수 있는 문제

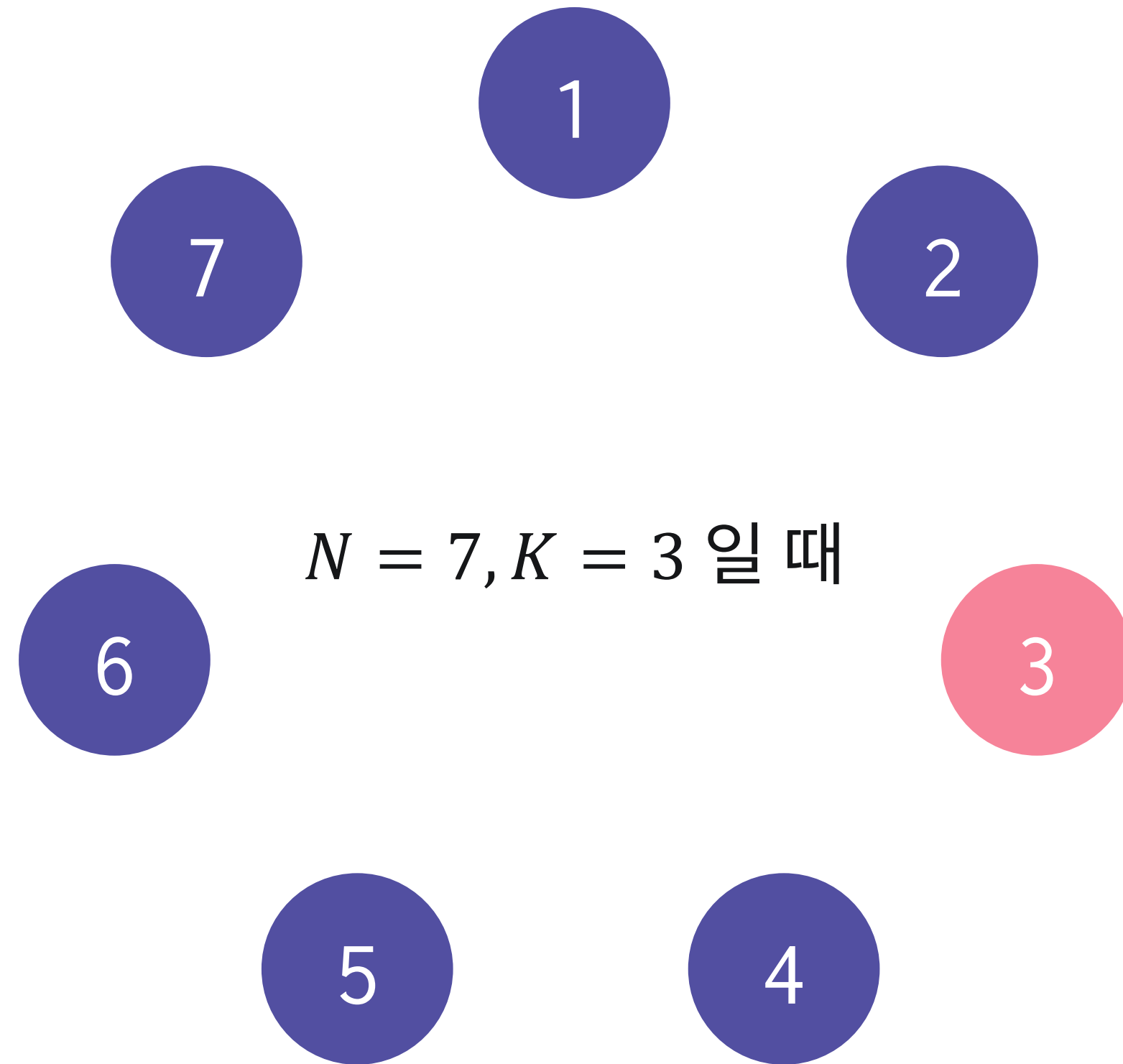
✓ [실습8] 요세푸스 순열



/* elice */

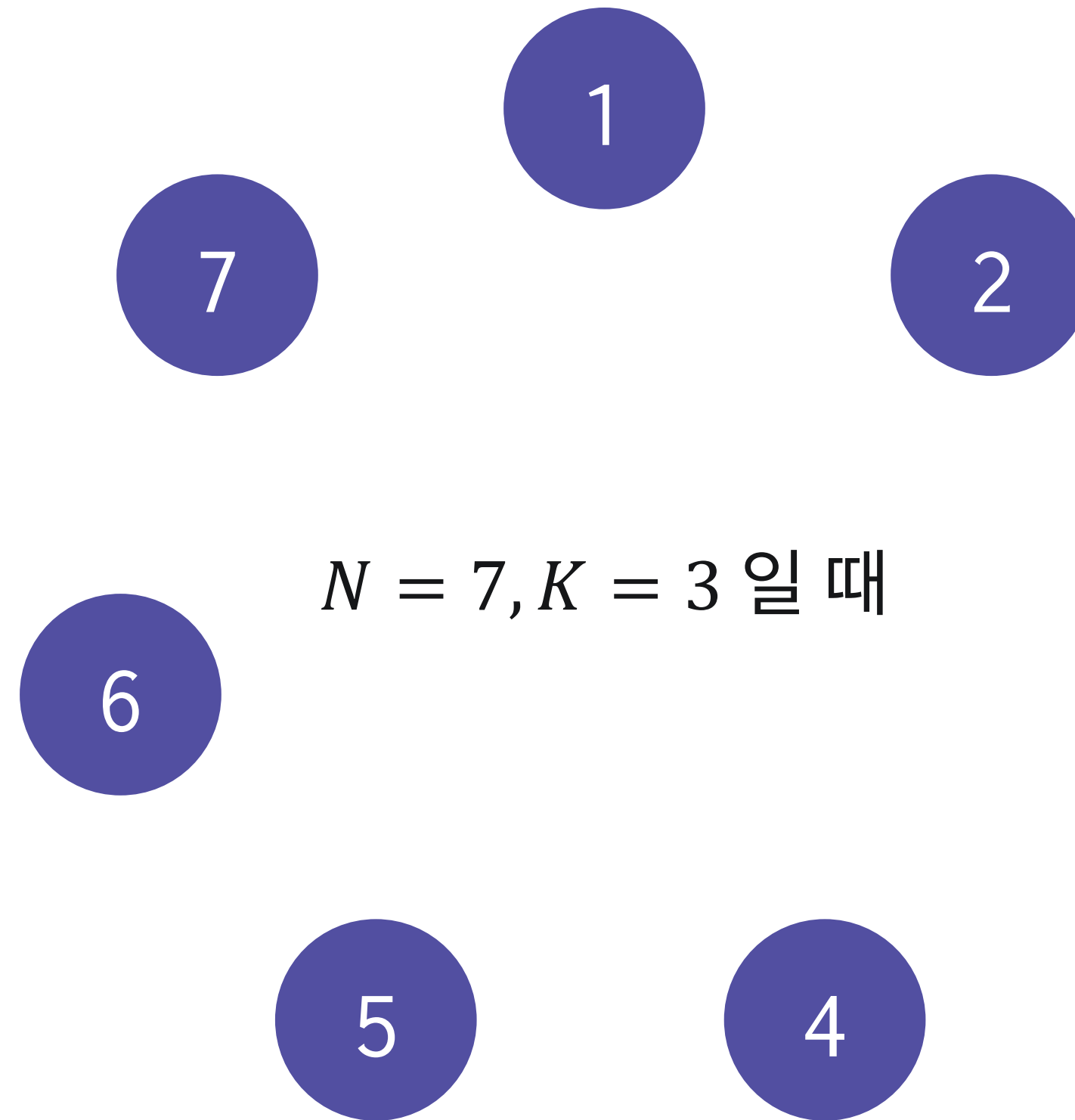
04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열



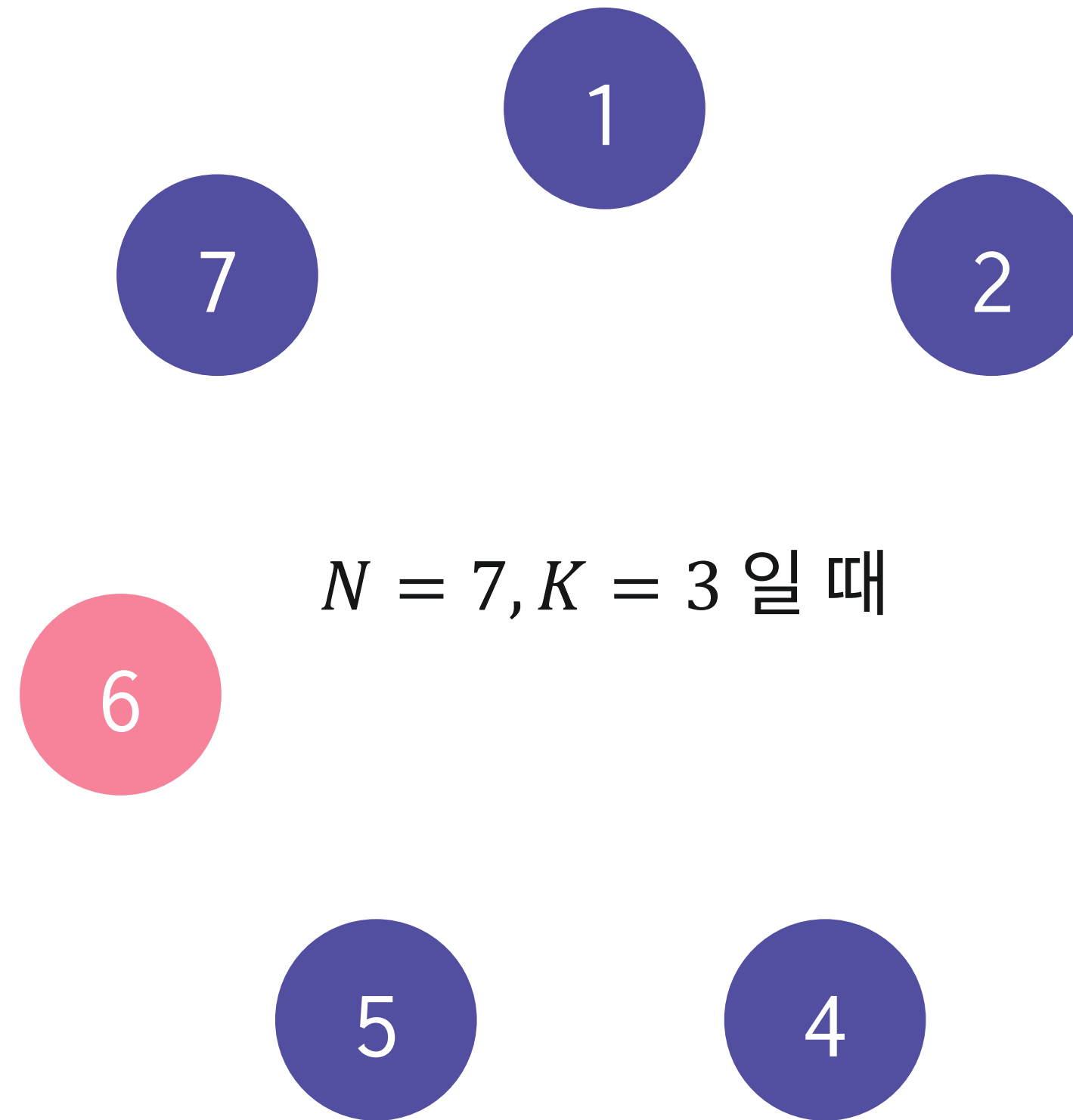
04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열



04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열



04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열



$N = 7, K = 3$ 일 때



`/* elice */`

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열



$N = 7, K = 3$ 일 때



`/* elice */`

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열



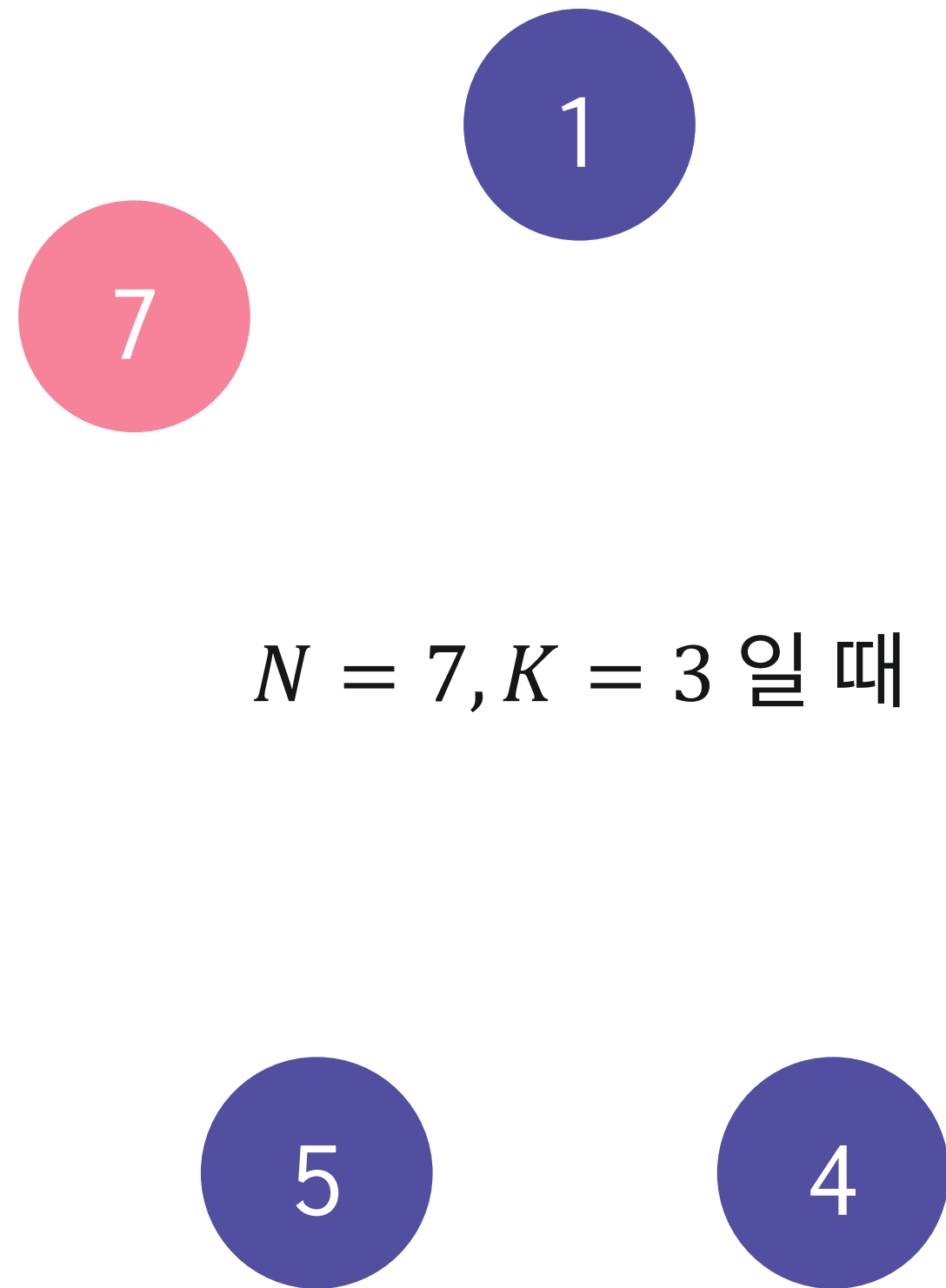
$N = 7, K = 3$ 일 때



`/* elice */`

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열



04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

1

$N = 7, K = 3$ 일 때

5

4

`/* elice */`

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

1

$N = 7, K = 3$ 일 때

5

4

`/* elice */`

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

1

$N = 7, K = 3$ 일 때

4

/* elice */

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

1

$N = 7, K = 3$ 일 때

4

/* elice */

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

$N = 7, K = 3$ 일 때

4

`/* elice */`

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

$N = 7, K = 3$ 일 때

4

/* elice */

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

$N = 7, K = 3$ 일 때

`/* elice */`

04 큐로 풀 수 있는 문제

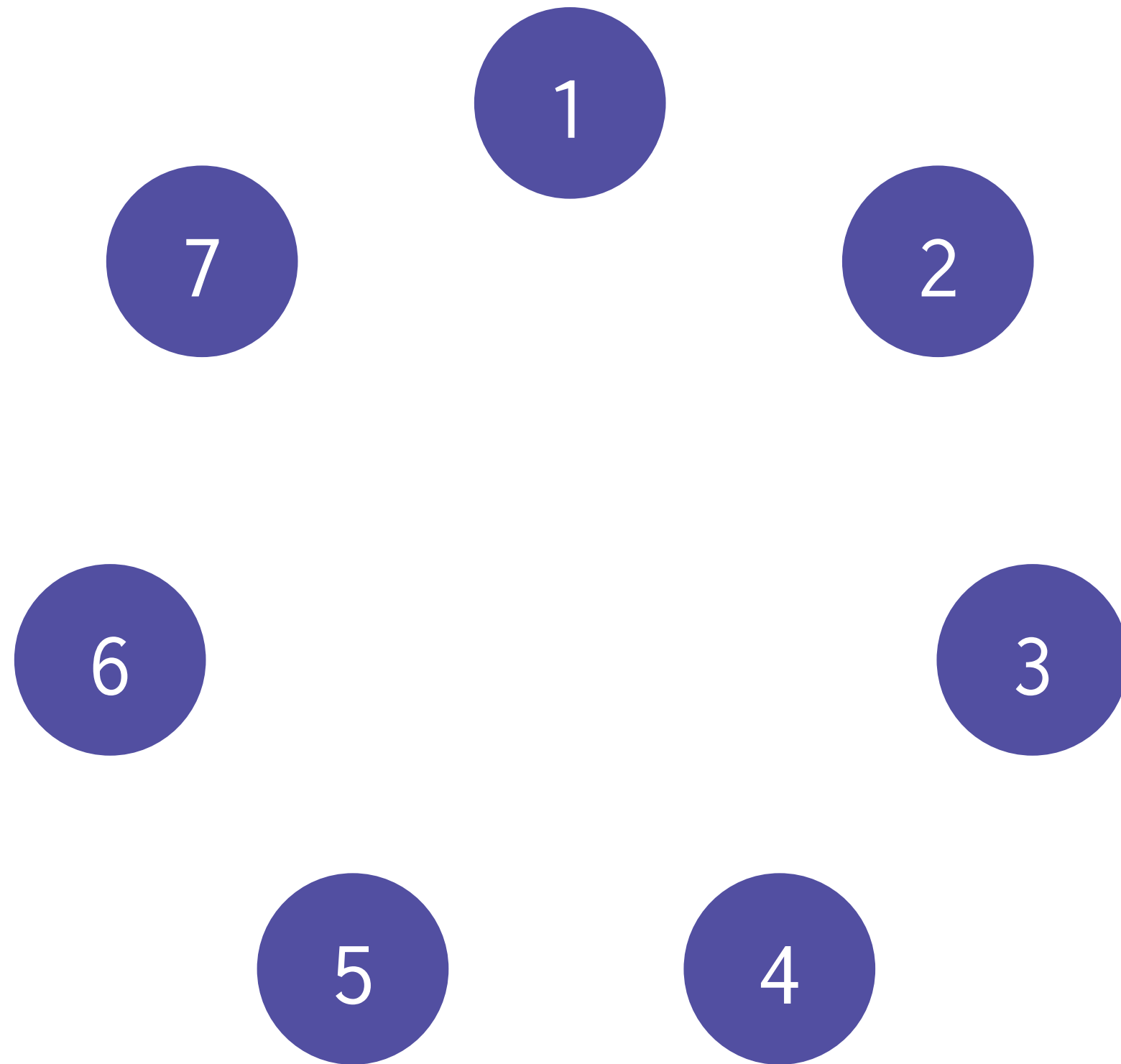
✓ [실습8] 요세푸스 순열



$N = 7, K = 3$ 일 때 제거되는 순서는 위와 같다.

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열



문제를 다르게 생각해보자.

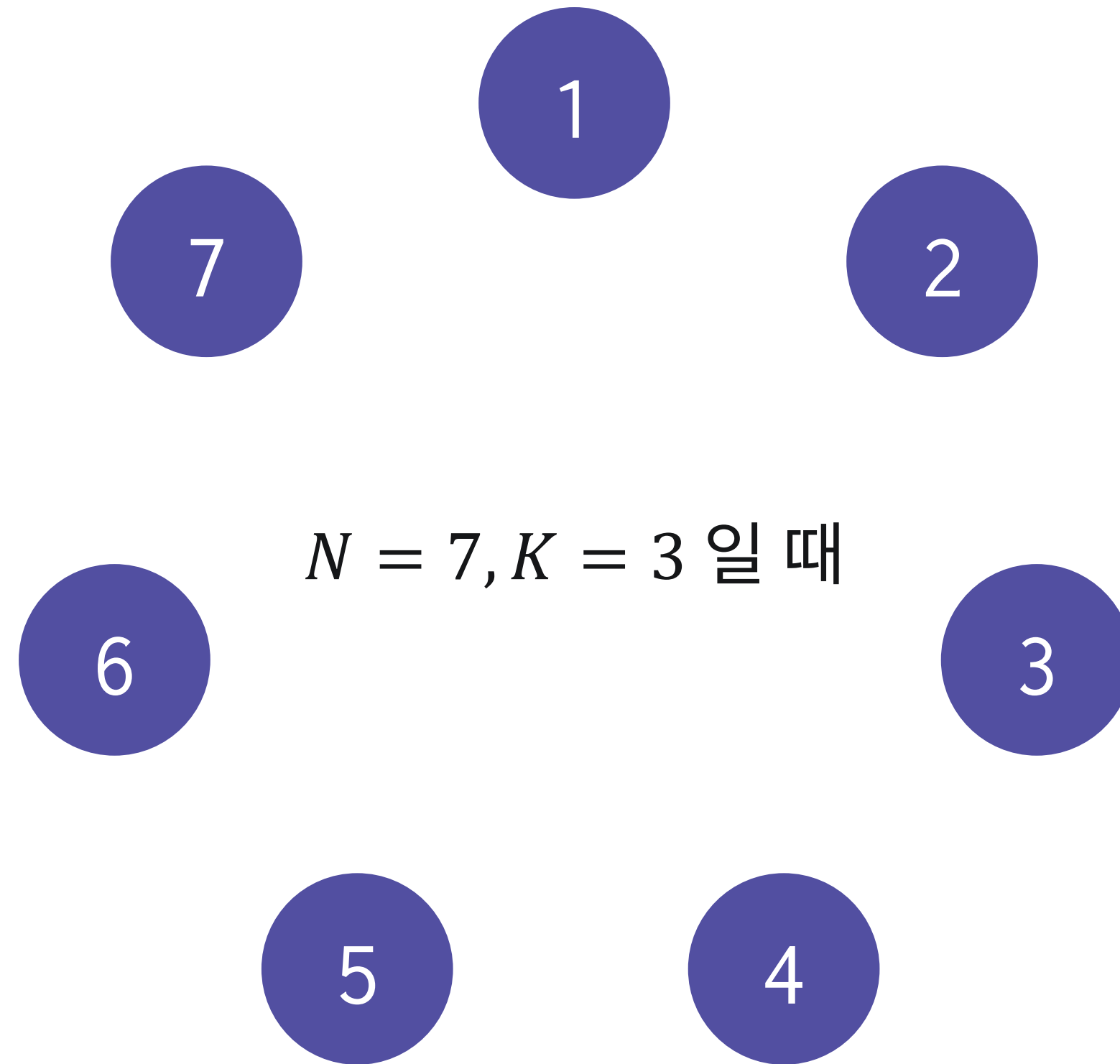
1번부터 K 번째 사람이 제거되는 것이 아니라,

원을 $K - 1$ 번 반시계 방향으로 회전시켰을 때

12시 위치에 있는 사람이
제거되는 것으로 생각해보자.

04 큐로 풀 수 있는 문제

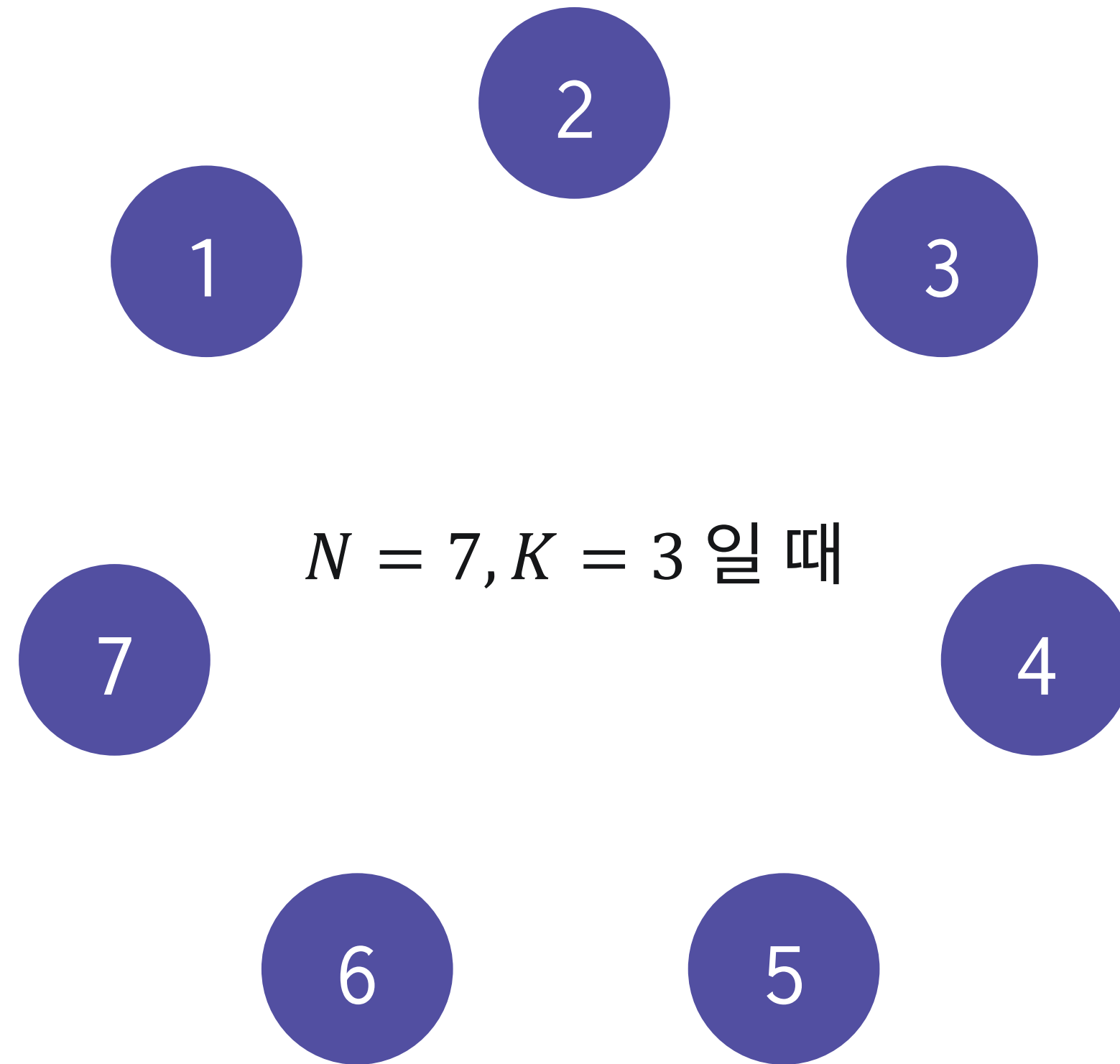
✓ [실습8] 요세푸스 순열



/* elice */

04 큐로 풀 수 있는 문제

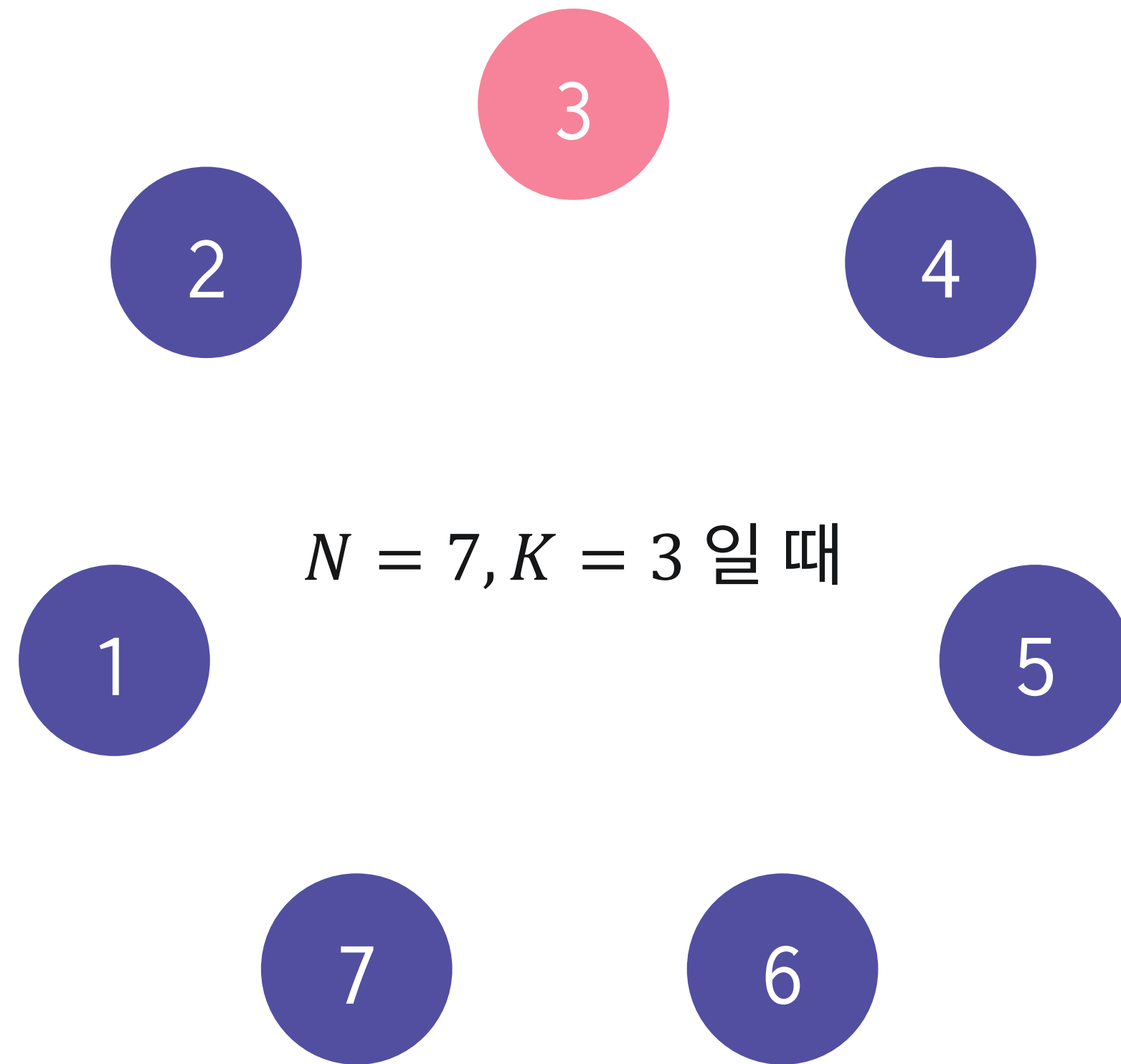
✓ [실습8] 요세푸스 순열



/* elice */

04 큐로 풀 수 있는 문제

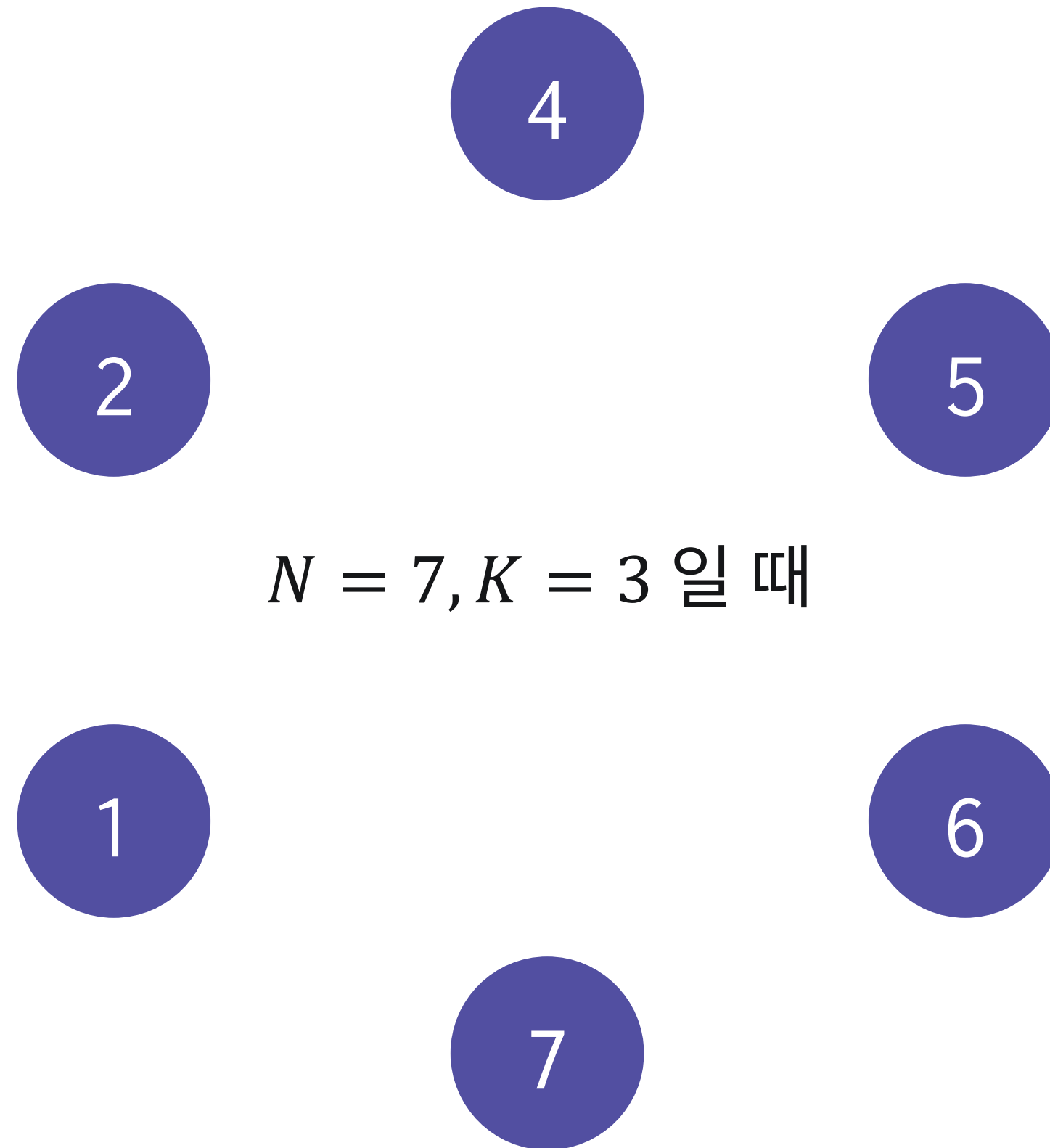
✓ [실습8] 요세푸스 순열



/* elice */

04 큐로 풀 수 있는 문제

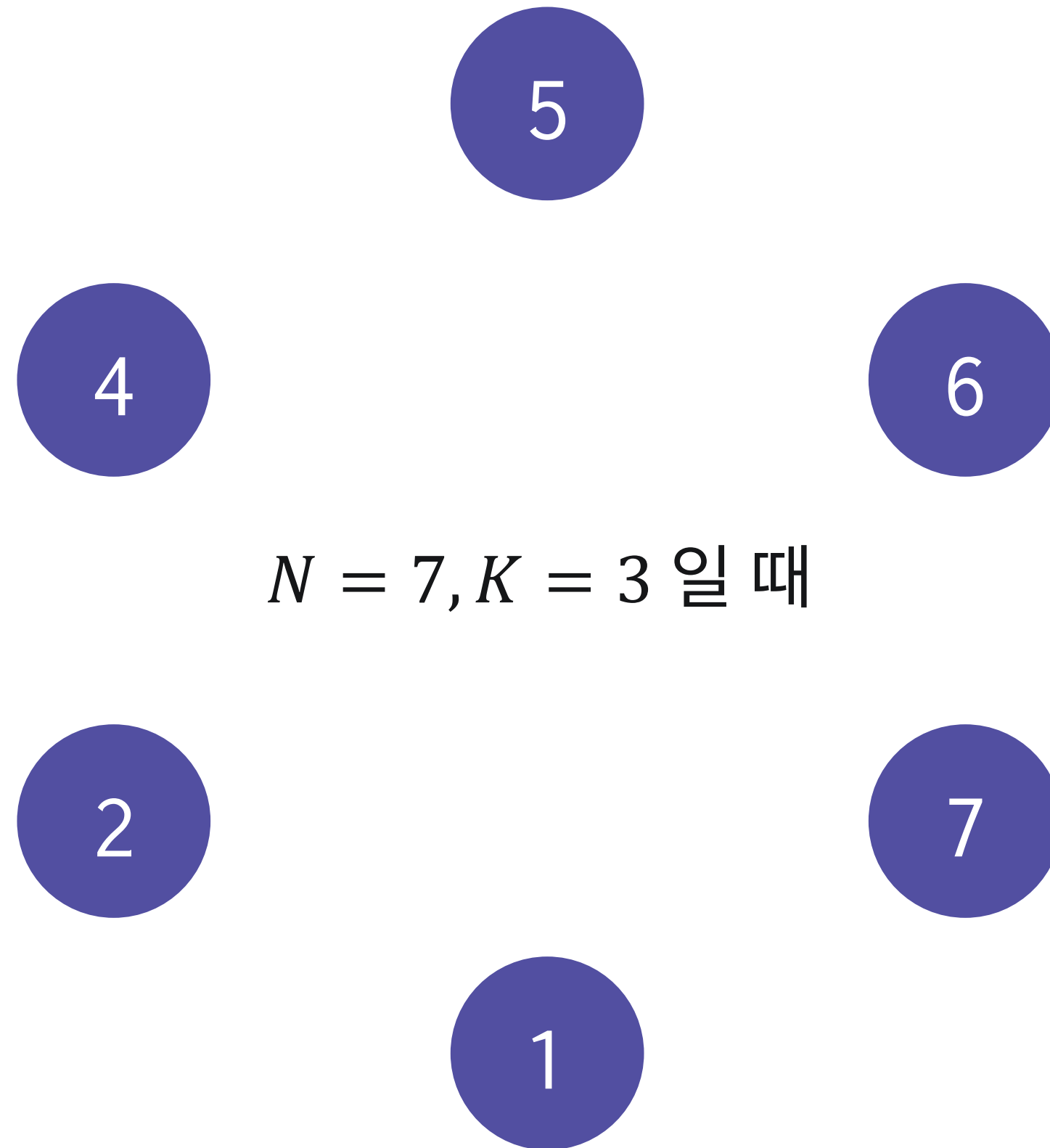
✓ [실습8] 요세푸스 순열



/* elice */

04 큐로 풀 수 있는 문제

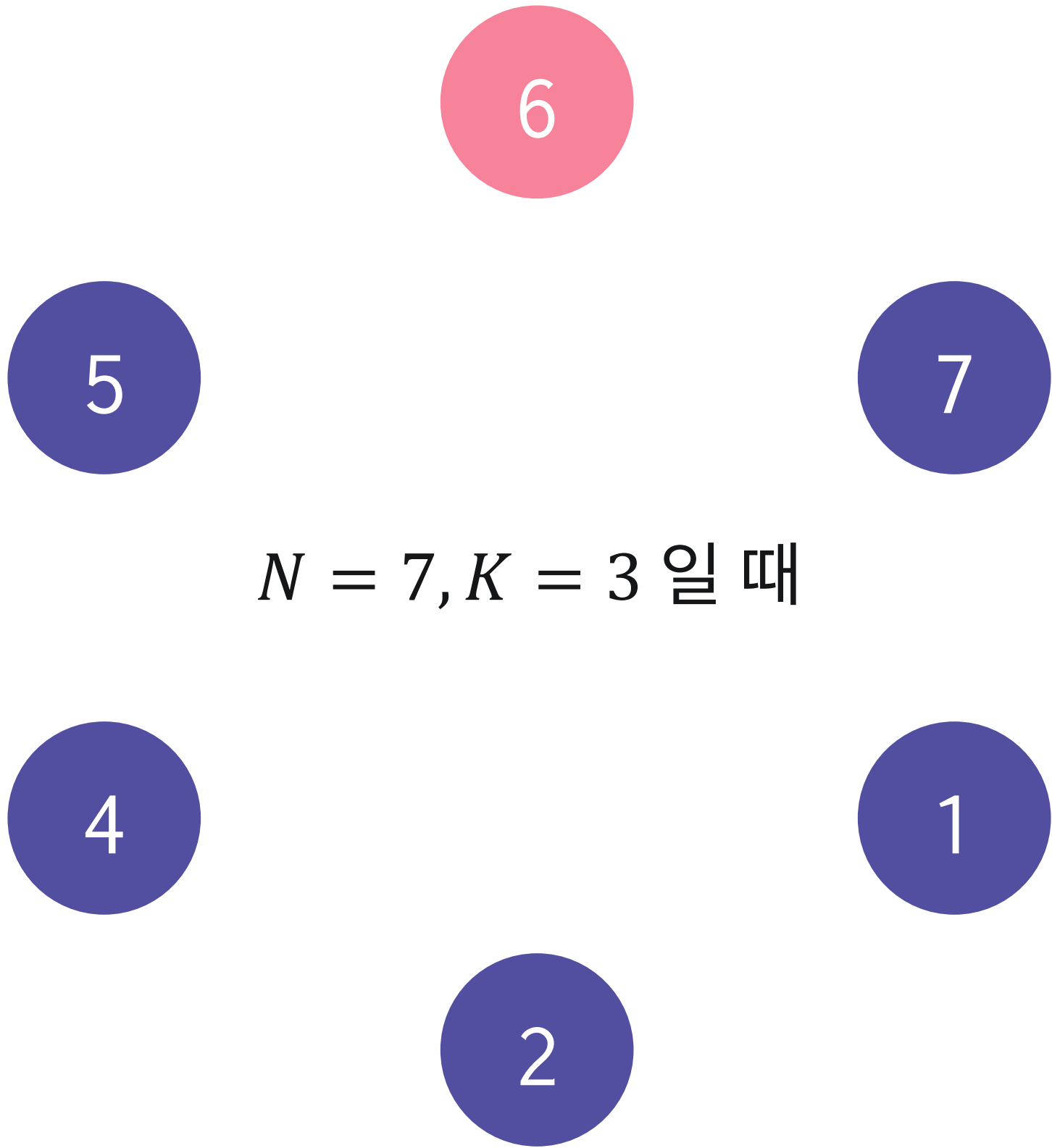
✓ [실습8] 요세푸스 순열



/* elice */

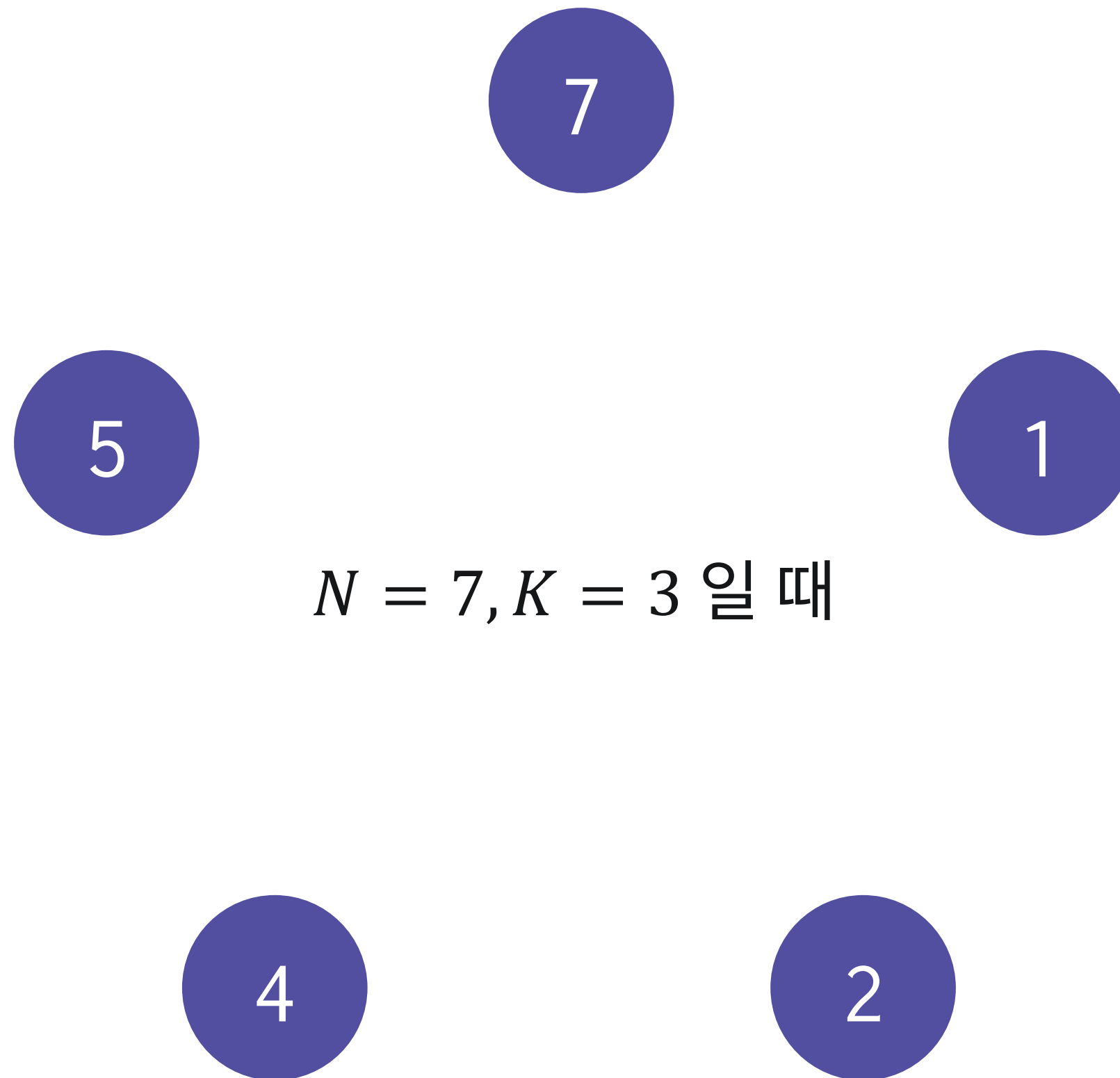
04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열



04 큐로 풀 수 있는 문제

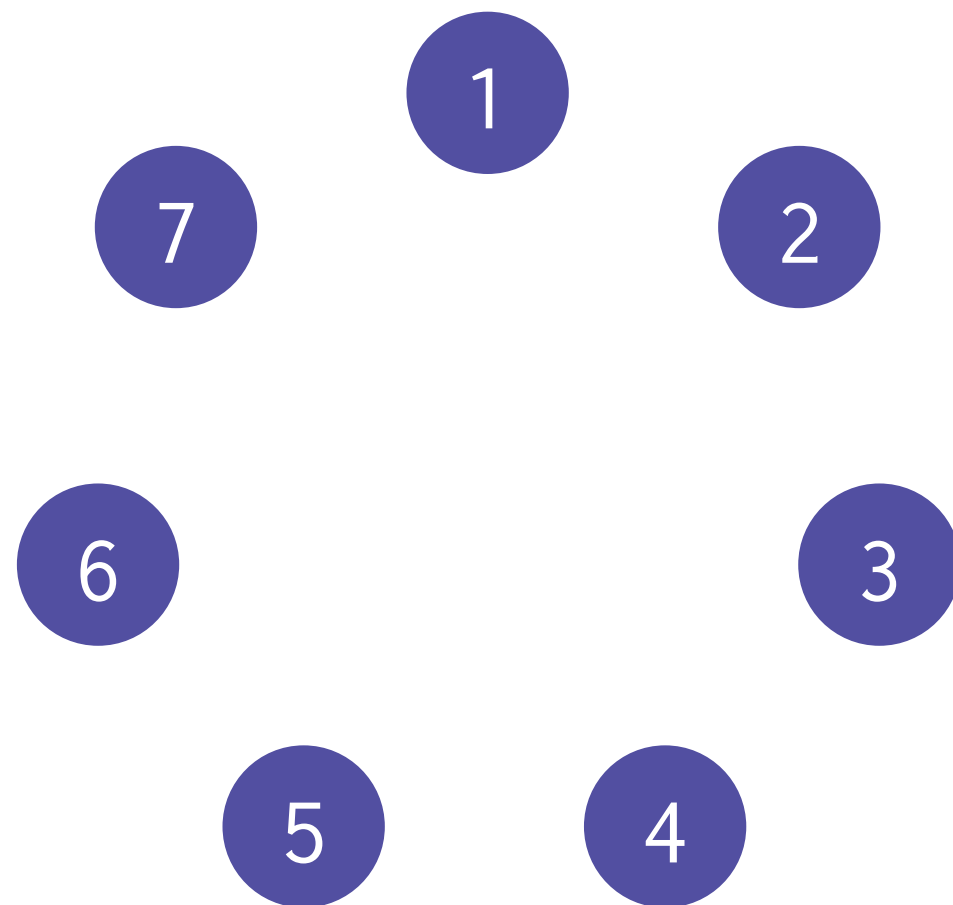
✓ [실습8] 요세푸스 순열



/* elice */

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열



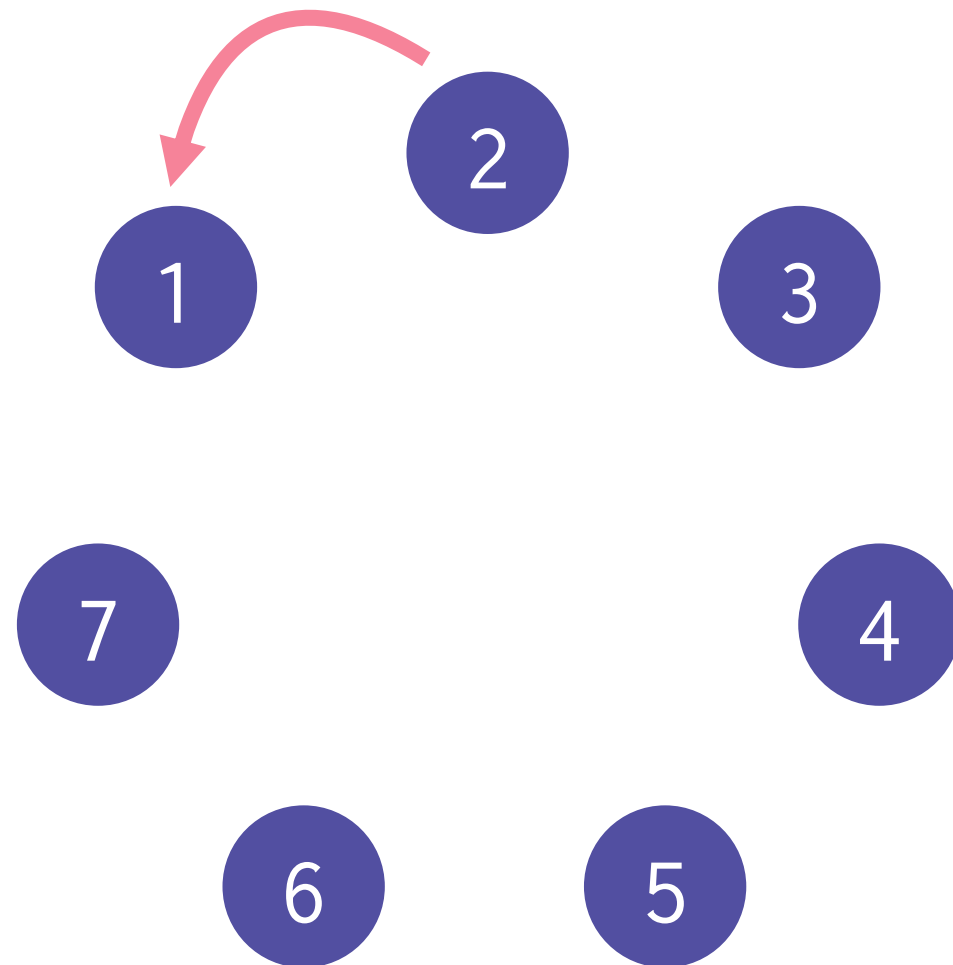
원을 이루고 있는 사람들은 다음과 같이 표현할 수 있다.

맨 처음을 12시 방향의 사람이라고 하면,

왼쪽 그림은 **[1, 2, 3, 4, 5, 6, 7]** 이 된다.

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열



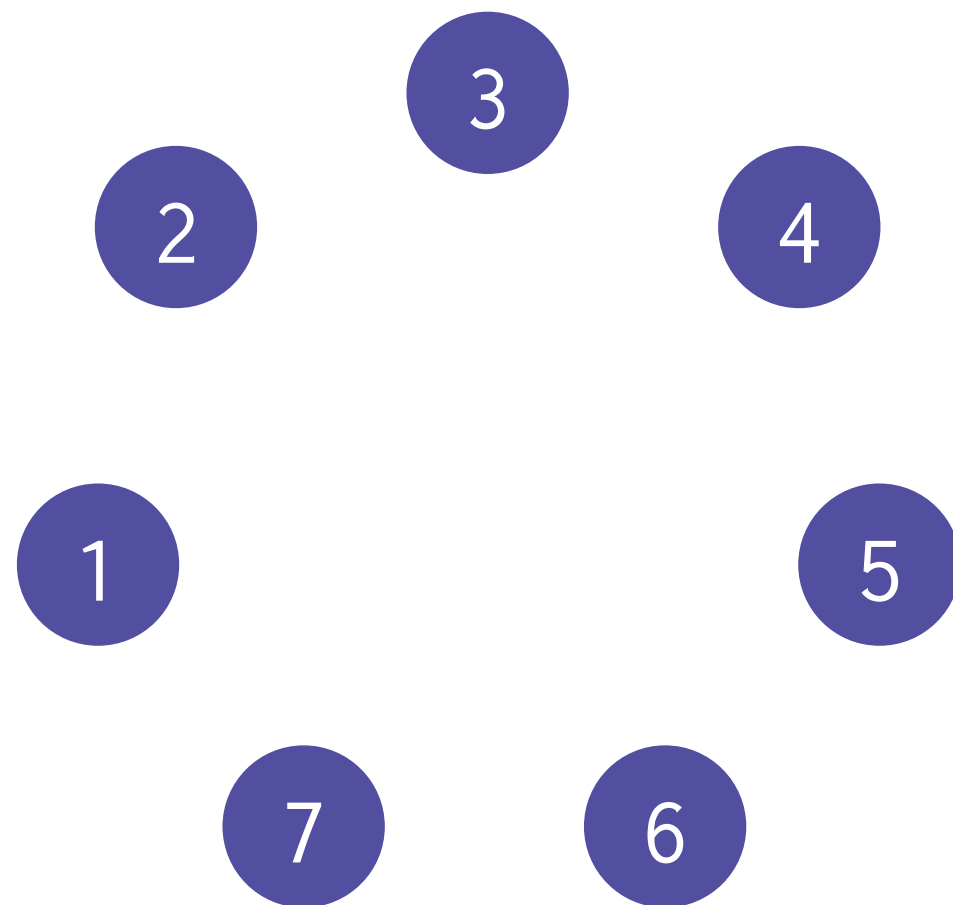
한 번 회전하면 12시 방향의 사람이 바뀌고,

자기 자신은 맨 마지막 순서로 밀려난다.

따라서 **[2, 3, 4, 5, 6, 7, 1]** 이 된다.

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

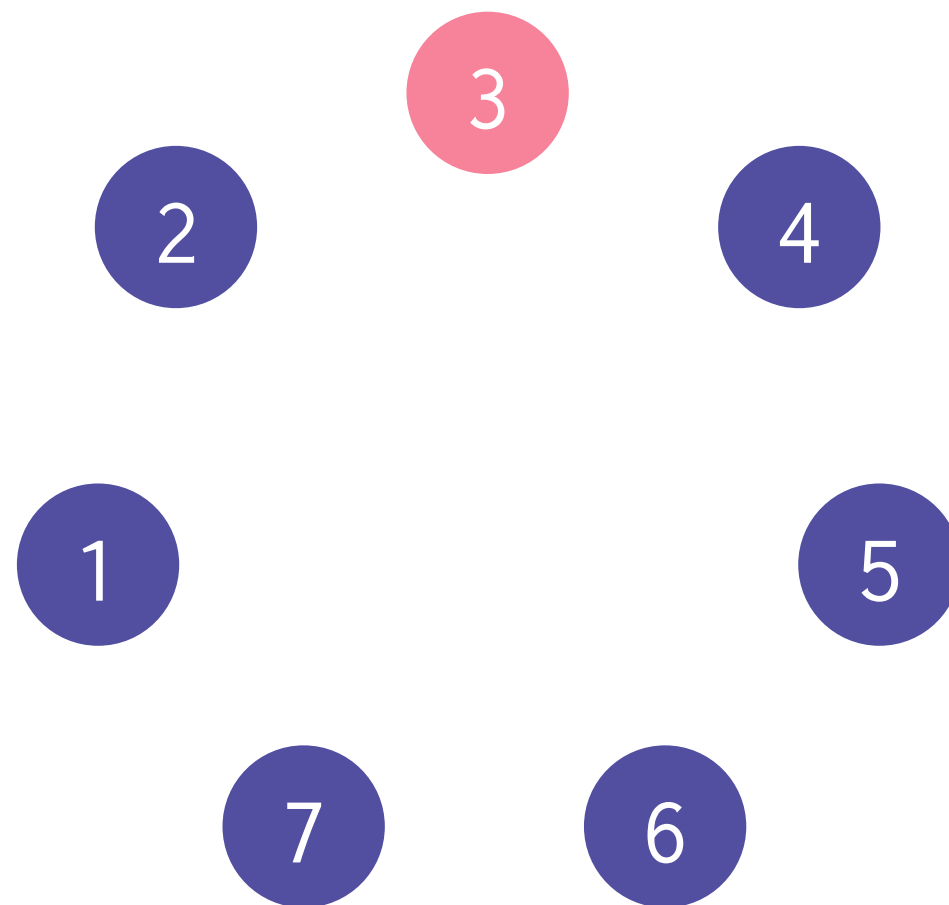


다시 한번 회전하면 12시 방향의 사람은 3이 되고,

[3, 4, 5, 6, 7, 1, 2] 이 된다.

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열



$K - 1$ 번 회전하였으므로, 현재 12시에 있는 3번을 제거한다.

[3, 4, 5, 6, 7, 1, 2]에서

[4, 5, 6, 7, 1, 2]가 된다.

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

이와 같이 계속 자료구조의 연산이 이루어진다고 할 때,
가장 적절한 자료구조는 **큐**이다.

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

원이 회전하면 **큐에서 자료를 출력**한 후
그 **출력한 자료를 다시 큐에 입력**해주면 된다.

큐는 FIFO 자료구조이기 때문에 맨 앞에 있는 요소가
제거되기 때문이다.

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

배열을 사용해도 되지만, **시간상 불리함**이 있다.

배열에서 어떤 요소를 제거하면 **뒤에 있는 모든 요소**들이
인덱스를 맞추기 위해 **한 칸씩 이동**하는 작업을 하기 때문이다.

04 큐로 풀 수 있는 문제

✓ [실습8] 요세푸스 순열

1. 12시에 있는 사람을 시작으로 보고, 1번부터 N 번까지 큐에 저장한다.
2. 큐에 저장된 자료가 **모두 제거될 때까지** 아래의 작업을 반복한다.
 - (a) 큐를 출력(pop)하고, 출력된 수를 다시 큐에 입력(push)한다.
이 작업을 **$K-1$ 번** 반복한다.
 - (b) 큐를 출력(pop)한다.
 - (c) (b)에 의해 출력된 수를 순서대로 저장한다.