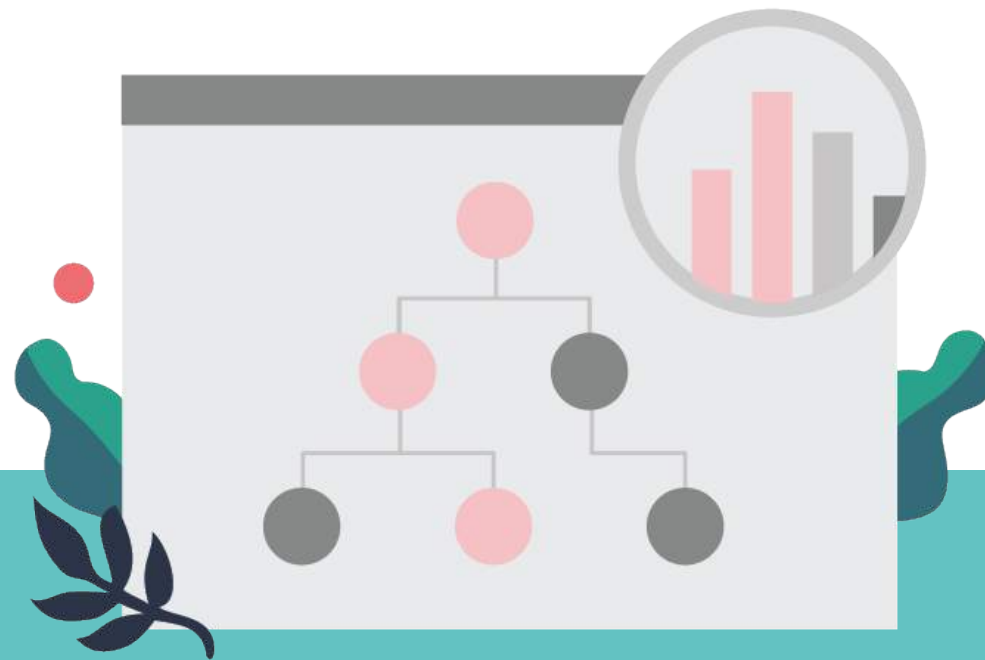


/* 데이터 구조 및 알고리즘 */

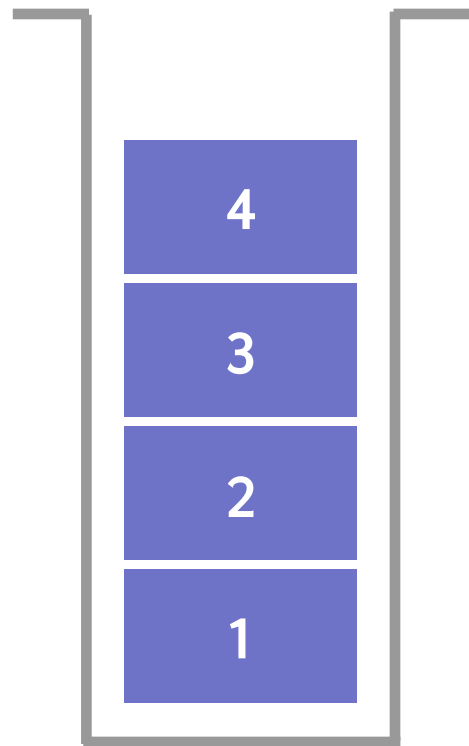
신현규 강사, 화/목 20:00

해싱



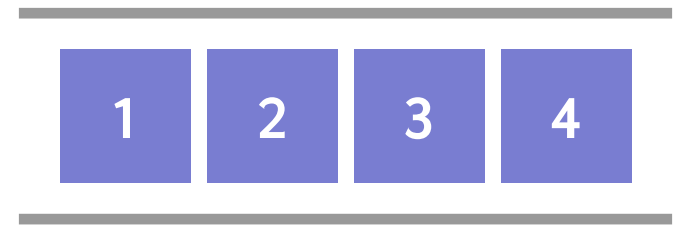
/* elice */

지난시간 요약



Stack

Last In First Out



Queue

First In First Out

퀴즈

Q1.

Q2.

그래서 스택은 언제 쓰나요 ?

스택은 자료구조

무슨 자료를 저장하는가 ?

상태 (Status)

실생활의 예 : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄비



1. 열쇠 찾기



1. 문 열기
2. 서랍 열기
3. 카드 꺼내기
4. 집 나오기

실생활의 예 : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄비



1. 열쇠 찾기



1. 문 열기
2. 서랍 열기
3. 카드 꺼내기
4. 집 나오기

미역국 끓이기

실생활의 예 : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄비



1. 열쇠 찾기



1. 문 열기
2. 서랍 열기
3. 카드 꺼내기
4. 집 나오기

마트. (4)

미역국 끓이기

실생활의 예 : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄비



1. 열쇠 찾기



1. 문 열기
2. 서랍 열기
3. 카드 꺼내기
4. 집 나오기

집. (2)

마트. (4)

미역국 끓이기

실생활의 예 : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄비



1. 열쇠 찾기



1. 문 열기
2. 서랍 열기
3. 카드 꺼내기
4. 집 나오기

집. (2)

마트. (4)

미역국 끓이기

실생활의 예 : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄비



1. 열쇠 찾기



1. 문 열기
2. 셔랍 열기
3. 카드 꺼내기
4. 집 나오기

마트. (4)

미역국 끓이기

실생활의 예 : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 열쇠 찾기



1. 문 열기
2. 셔랍 열기
3. 카드 꺼내기
4. 집 나오기

미역국 끓이기

실생활의 예 : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 열쇠 찾기



1. 문 열기
2. 셔랍 열기
3. 카드 꺼내기
4. 집 나오기

그래서 스택은 언제 쓰나요 ?

“상태”의 의존관계가 생길 때

A라는 일을 마치기 위해서 B라는 일을 먼저 끝내야 할 때

재귀호출

Computational Thinking

실생활의 예 (2)



실생활의 예 (2)



실생활의 예 (2)



실생활의 예 (2)



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄비

실생활의 예 (2)



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄비

실생활의 예 (2)



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄비

1. 옷 찾기

실생활의 예 (2)



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄비



1. 옷 찾기



실생활의 예 (2)



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄비



1. 옷 찾기



1. 방세 입금하기

실생활의 예 (2)



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 옷 찾기



1. 방세 입금하기

실생활의 예 (2)



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 옷 찾기



1. 방세 입금하기

실생활의 예 (2)



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 옷 찾기



1. 방세 입금하기

실생활의 예 (2)



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 옷 찾기



1. 방세 입금하기

실생활의 예 (2)



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 옷 찾기



1. 방세 입금하기

실생활의 예 (2) : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄비

실생활의 예 (2) : 자료구조 관점



미역국
끓이기

1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄비

실생활의 예 (2) : 자료구조 관점



미역국
끓이기

1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄비

실생활의 예 (2) : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄비



1. 옷 찾기



미역국
끓이기

옷
찾기

실생활의 예 (2) : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄비



1. 옷 찾기



미역국
끓이기

옷
찾기

실생활의 예 (2) : 자료구조 관점



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄비



1. 옷 찾기



1. 월세 입금하기

미역국
끓이기

옷
찾기

월세
입금

실생활의 예 (2) : 자료구조 관점



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 옷 찾기



1. 월세 입금하기

미역국
끓이기

옷
찾기

월세
입금

실생활의 예 (2) : 자료구조 관점



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 옷 찾기



1. 월세 입금하기

옷
찾기

월세
입금

실생활의 예 (2) : 자료구조 관점



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 옷 찾기



1. 월세 입금하기

옷
찾기

월세
입금

실생활의 예 (2) : 자료구조 관점



1. 마역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 옷 찾기



1. 월세 입금하기

월세
입금

실생활의 예 (2) : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 옷 찾기



1. 월세 입금하기

월세
입금

실생활의 예 (2) : 자료구조 관점



1. 미역
2. 국간장
3. 후추
4. 고기
5. 냄바



1. 옷 찾기



1. 월세 입금하기

그래서 큐는 언제 쓰나요 ?

“상태”의 의존상태가 없을 때

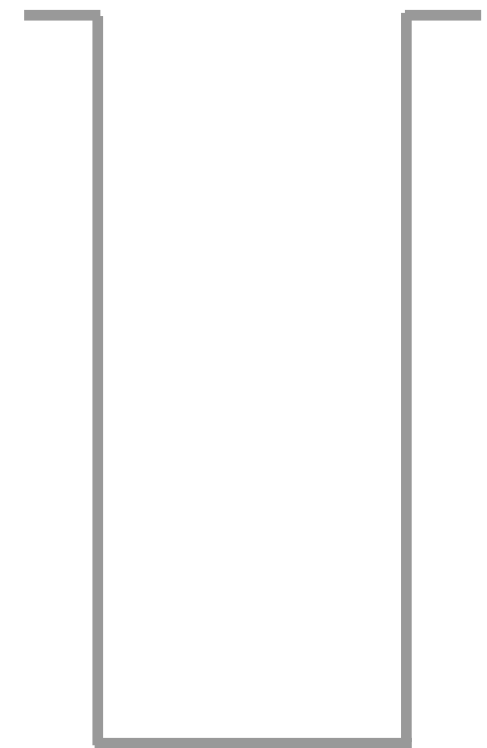
A와 B가 서로 관련이 없지만 모두 하긴 해야할 때

스케줄링, 병렬화

[문제 4] 계산 순서 정하기

스택의 대표적인 활용

((()) ())



Stack

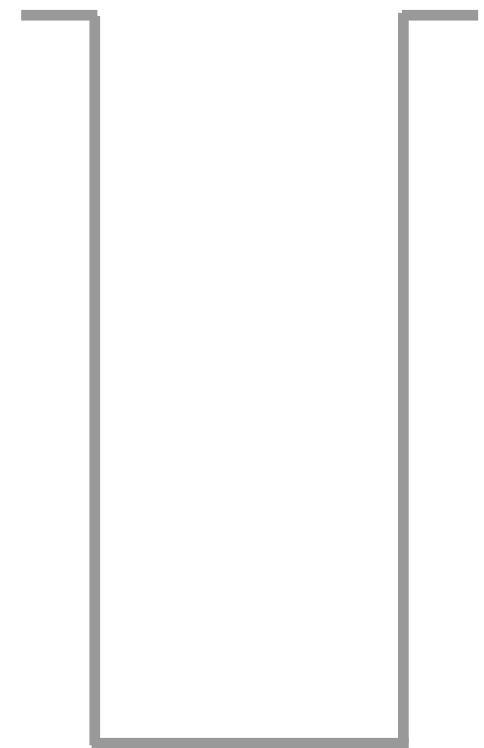
Last In First Out

[문제 4] 계산 순서 정하기

스택의 대표적인 활용



((()) ())



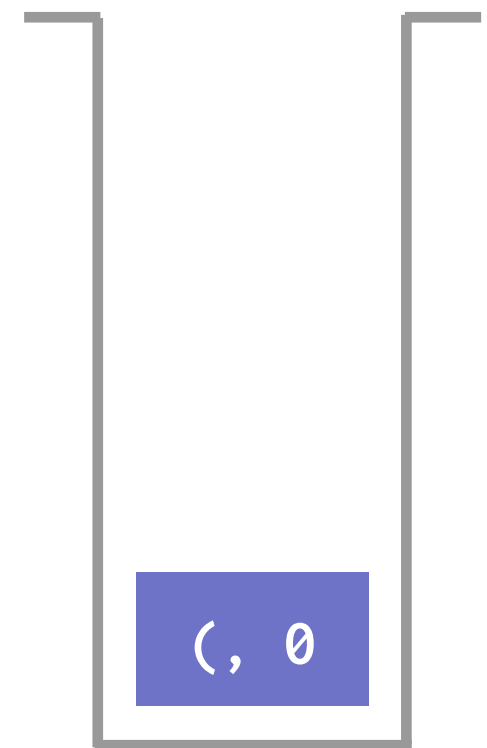
Stack

Last In First Out

[문제 4] 계산 순서 정하기

스택의 대표적인 활용

((()) ())



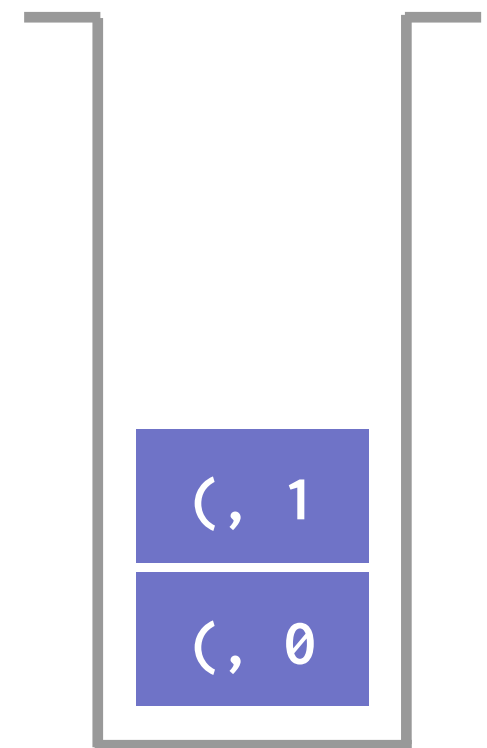

Stack

Last In First Out

[문제 4] 계산 순서 정하기

스택의 대표적인 활용

((()) ())



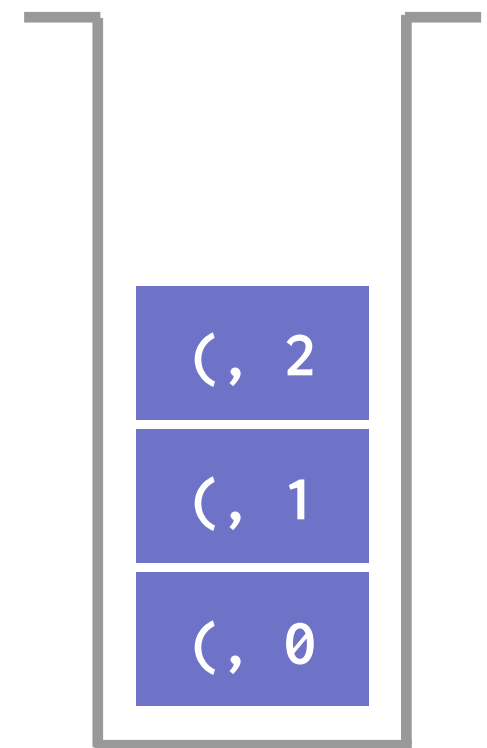

Stack

Last In First Out

[문제 4] 계산 순서 정하기

스택의 대표적인 활용

((()) ())



Stack

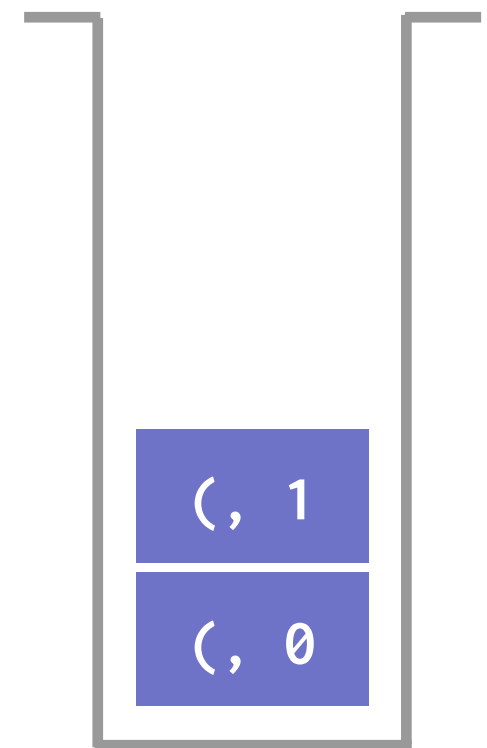

Last In First Out

[문제 4] 계산 순서 정하기

스택의 대표적인 활용

((()) ())

1 1



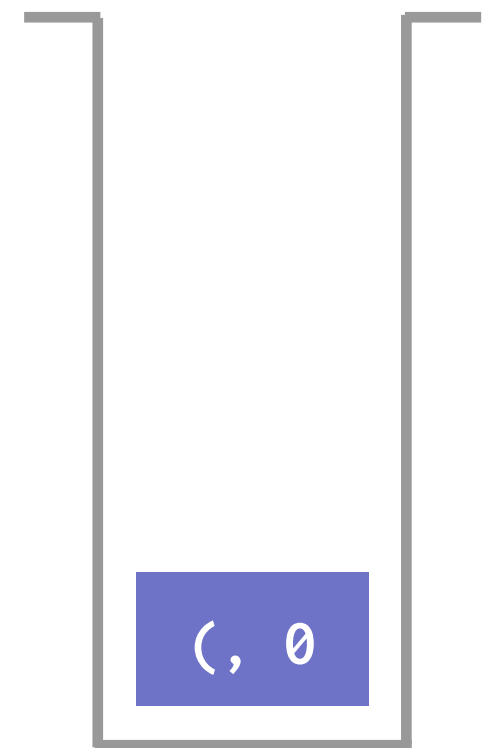

Stack

Last In First Out

[문제 4] 계산 순서 정하기

스택의 대표적인 활용

((()) ())
2 1 1 2



Stack

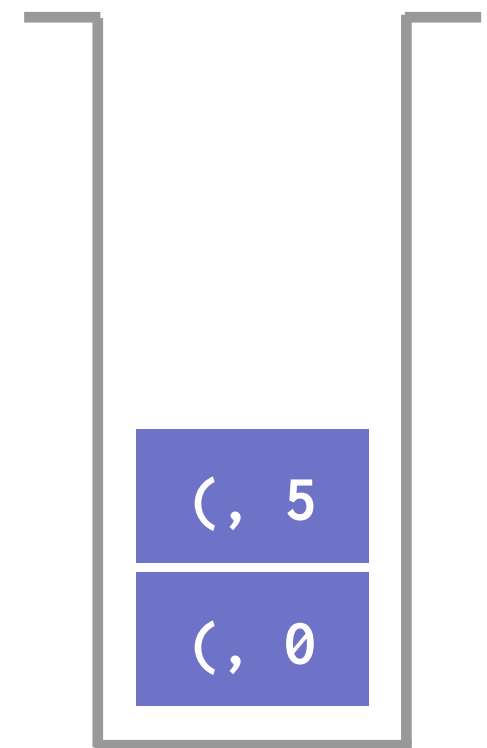
Last In First Out

[문제 4] 계산 순서 정하기

스택의 대표적인 활용

((()) ())
2 1 1 2

↓



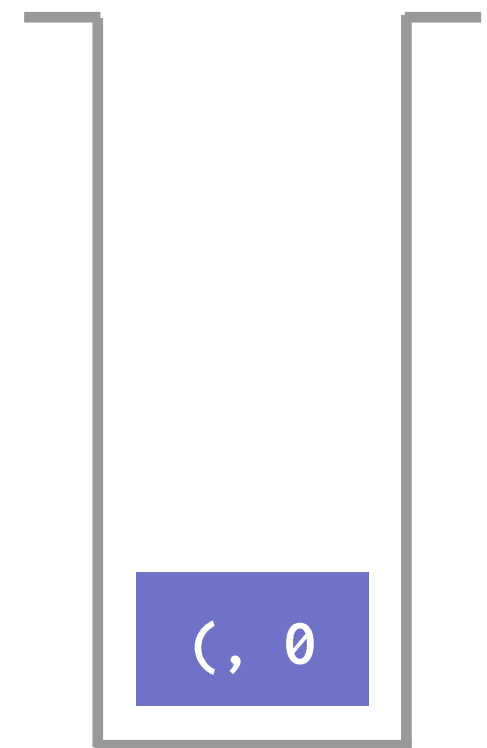

Stack

Last In First Out

[문제 4] 계산 순서 정하기

스택의 대표적인 활용

((()) ())
2 1 1 2 3 3



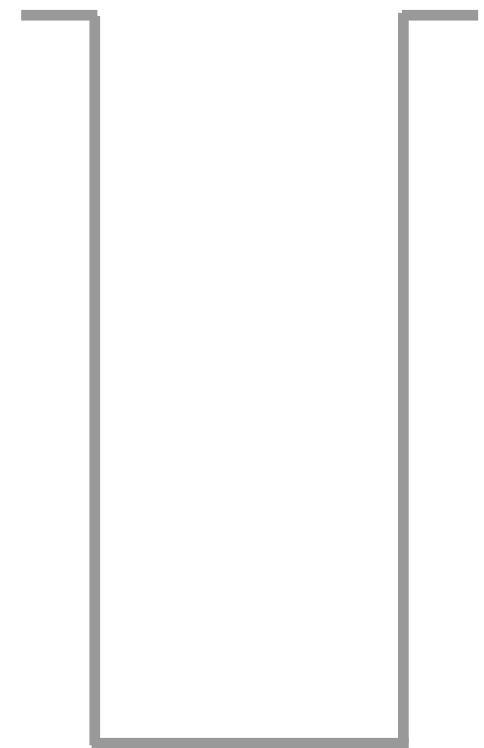
Stack

Last In First Out

[문제 4] 계산 순서 정하기

스택의 대표적인 활용

((()) ())
4 2 1 1 2 3 3 4



Stack

Last In First Out

스택, 큐 요약

선형 자료구조가 제일 어렵다

→ 목적이 모호하기 때문

스택과 큐에는 많은 경우 “상태”를 저장한다

스택과 큐는 그 용도가 다르다

→ 스택은 “상태”의 의존상태가 있을 때 (재귀호출)

→ 큐는 “상태”의 의존상태가 없을 때 (스케줄링, 병렬화)

주차별 커리큘럼

1주차 과정 소개, 배열, 연결리스트, 클래스

2주차 스택, 큐, 해싱

3주차 시간복잡도

4주차 트리, 트리순회, 재귀호출

5주차 힙

6주차 그래프 소개, DFS

7주차 그래프 심화, BFS

8주차 강의 요약, 알고리즘 과정 소개

Key-Value Store

Key를 이용하여 값을 읽고 쓰는 방법 (개념)

Database



Key-Value Store

Key를 이용하여 값을 읽고 쓰는 방법 (개념)



A diagram illustrating a Key-Value Store interaction. On the right, a large gray-outlined rectangle is labeled "Database". On the left, a stylized illustration of a person with a beard, wearing a purple shirt and a black cap, is sitting and using a laptop. A red curved arrow originates from the person's laptop area and points towards the "Database" box. Below the arrow, the text "key: hyungyu.sh@gmail.com" is displayed, indicating the key being used for the database operation.

Database

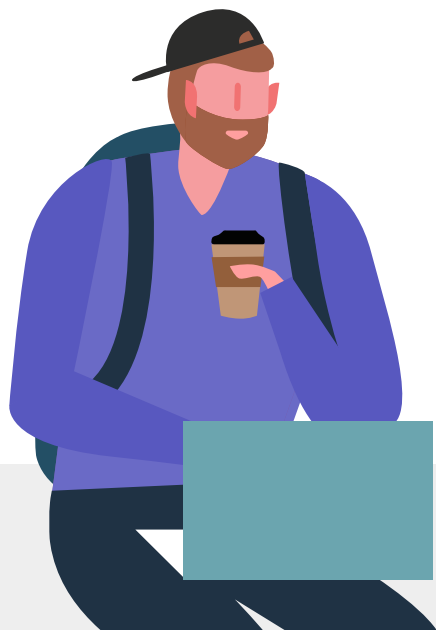
key: hyungyu.sh@gmail.com

Key-Value Store

Key를 이용하여 값을 읽고 쓰는 방법 (개념)

```
value {  
  이름: “신현규”  
  성별: 남  
  취미: 자료구조 디자인  
  특기: 자료구조 강의  
  ...  
}
```

Database



[연습문제] Key-value store

Key가 정수이고, **Value** 역시 정수인
key-value store 구현

시스템 입력

```
db = kvStore()  
db.put(2, 100)  
db.put(4, 20)  
print(db.get(4))  
print(db.get(2))  
print(db.get(7))
```

시스템 출력

```
20  
100  
Not exists
```

Key-Value Store의 구현

풀이 1 : 리스트의 **index**를 **key**로 이용하기

```
db = kvStore()
```

	0	1	2	3	4	5	6
myValue	3				6		



Key-Value Store의 구현

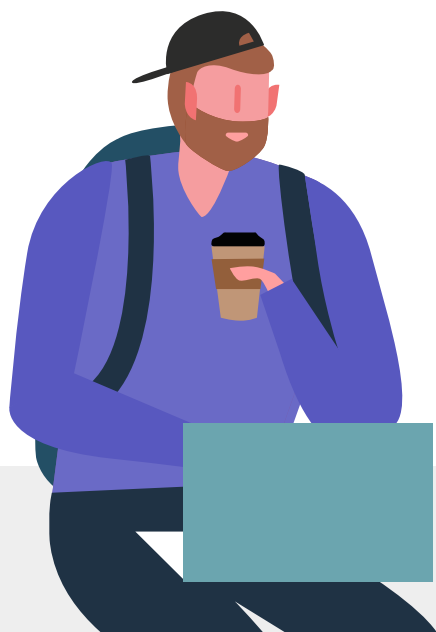
풀이 1 : 리스트의 **index**를 **key**로 이용하기

`db.put(2, 7)`



`db = kvStore()`

	0	1	2	3	4	5	6
myValue	3				6		

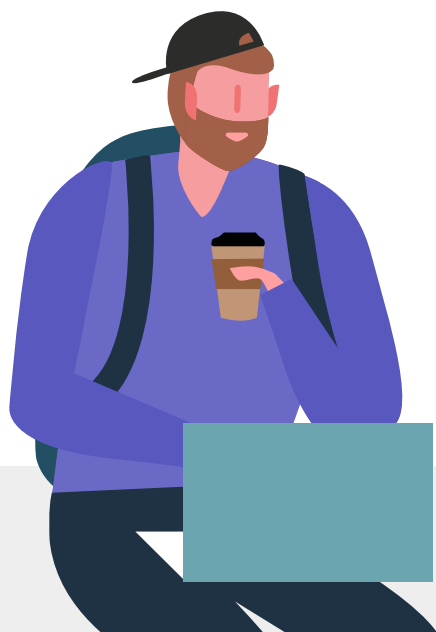


Key-Value Store의 구현

풀이 1 : 리스트의 **index**를 **key**로 이용하기

```
db = kvStore()
```

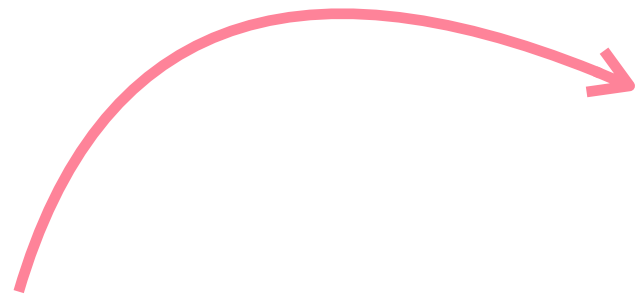
	0	1	2	3	4	5	6
myValue	3		7		6		



Key-Value Store의 구현

풀이 1 : 리스트의 **index**를 **key**로 이용하기

db.get(4)



```
db = kvStore()
```

	0	1	2	3	4	5	6
myValue	3		7		6		



Key-Value Store의 구현

풀이 1 : 리스트의 **index**를 **key**로 이용하기

```
db = kvStore()
```

	0	1	2	3	4	5	6
myValue	3		7		6		

6



Key-Value Store의 구현

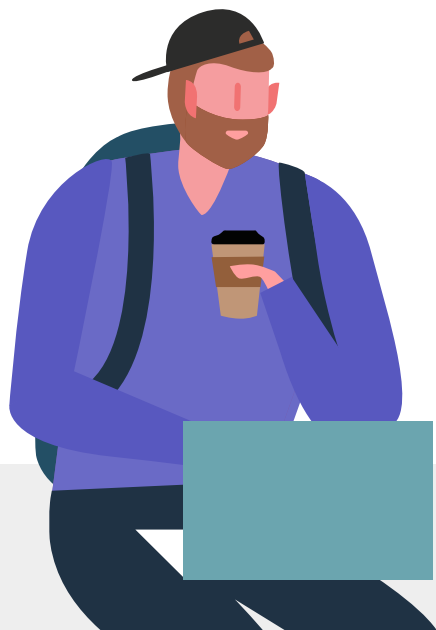
풀이 1 : 리스트의 **index**를 **key**로 이용하기 ; ;

`db.put(100, 2)`



`db = kvStore()`

	0	1	2	3	4	5	6
myValue	3		7		6		



Key-Value Store의 구현

풀이 1 : 리스트의 **index**를 **key**로 이용하기

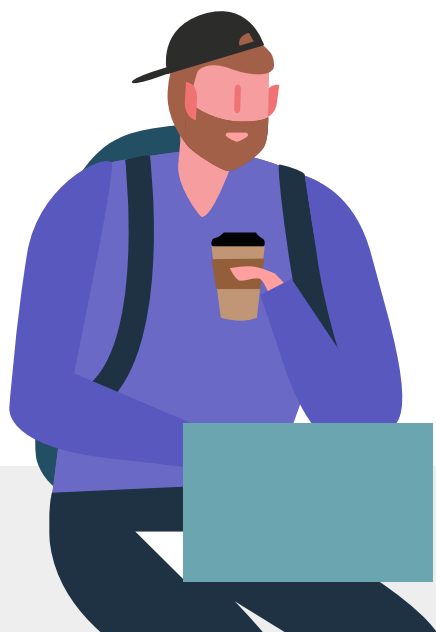
;;;;

db.get(3)



```
db = kvStore()
```

	0	1	2	3	4	5	6
myValue	3		7		6		



풀이 1의 장점과 단점

장점

(운 좋으면) 자료의 **쓰기** 연산이 빠르다
자료의 **읽기** 연산이 빠르다

단점

“**자료가 없다**”를 표현하는 것이 쉽지않다
공간이 지나치게 낭비될 수 있다

Key-Value Store의 구현

풀이 2 : key와 value를 모두 **리스트**에 저장하기

```
db = kvStore()
```

	0	1
myKey	2	3

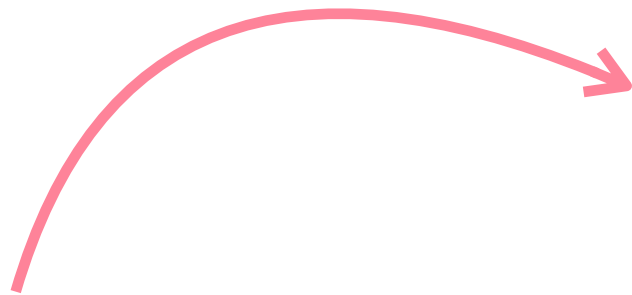
	0	1
myValue	1	7



Key-Value Store의 구현

풀이 2 : key와 value를 모두 **리스트**에 저장하기

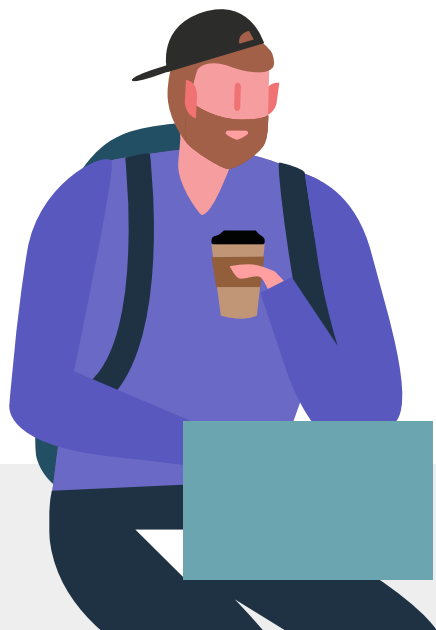
`db.put(4, 8)`



`db = kvStore()`

	0	1
myKey	2	3

	0	1
myValue	1	7



Key-Value Store의 구현

풀이 2 : key와 value를 모두 **리스트**에 저장하기

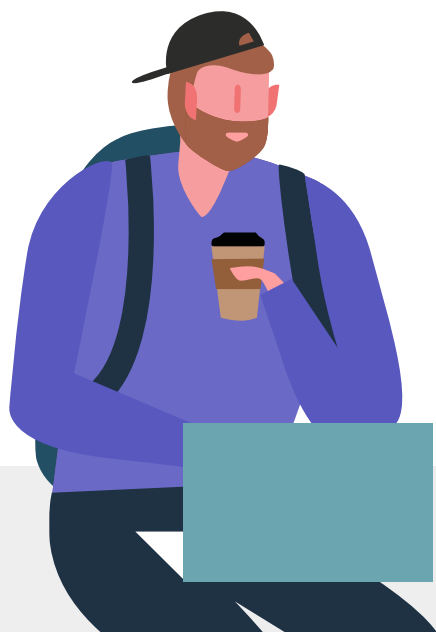
`db.put(4, 8)`



`db = kvStore()`

	0	1	2
myKey	2	3	4

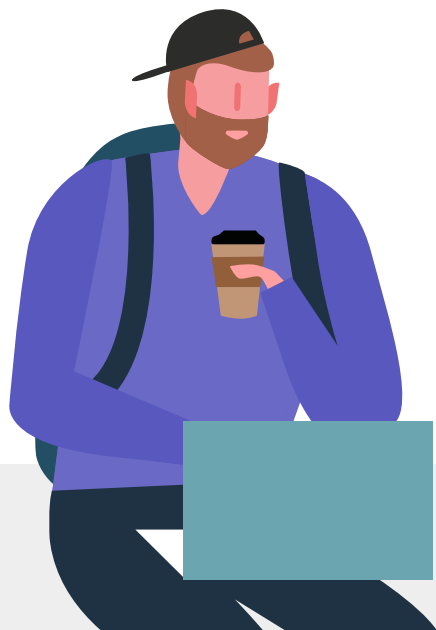
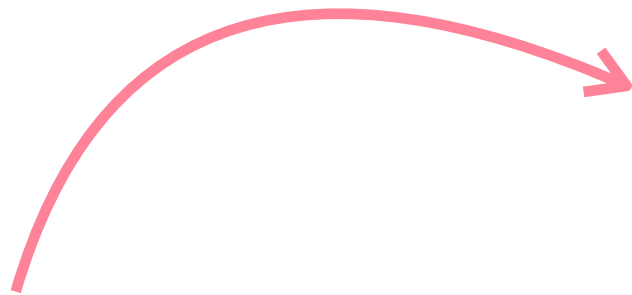
	0	1	2
myValue	1	7	8



Key-Value Store의 구현

풀이 2 : key와 value를 모두 **리스트**에 저장하기

`db.put(100, 2)`



`db = kvStore()`

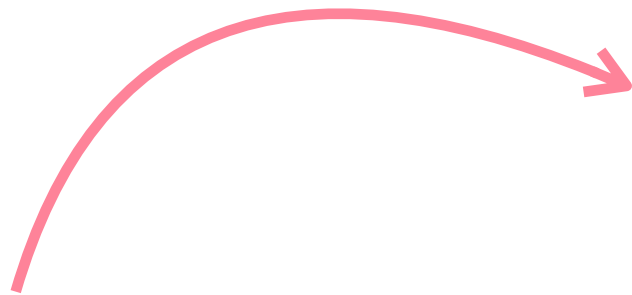
	0	1	2
myKey	2	3	4

	0	1	2
myValue	1	7	8

Key-Value Store의 구현

풀이 2 : key와 value를 모두 **리스트**에 저장하기

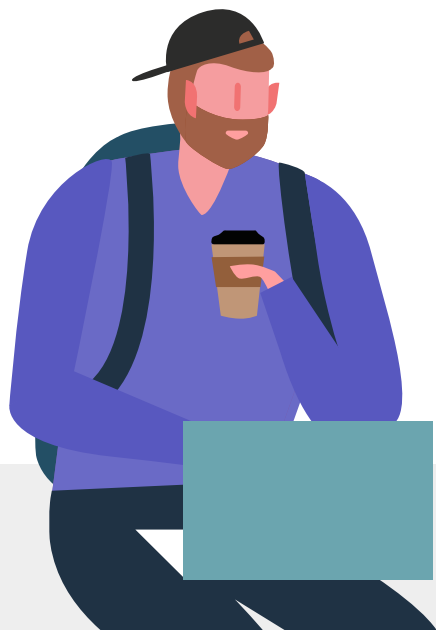
`db.put(100, 2)`



`db = kvStore()`

	0	1	2	3
myKey	2	3	4	100

	0	1	2	3
myValue	1	7	8	2



Key-Value Store의 구현

풀이 2 : key와 value를 모두 **리스트**에 저장하기

db.get(4)



db = kvStore()

	0	1	2	3
myKey	2	3	4	100

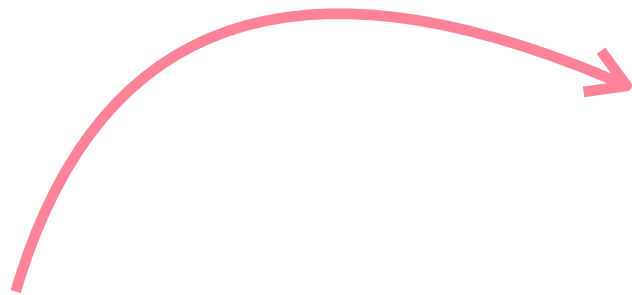
	0	1	2	3
myValue	1	7	8	2



Key-Value Store의 구현

풀이 2 : key와 value를 모두 **리스트**에 저장하기

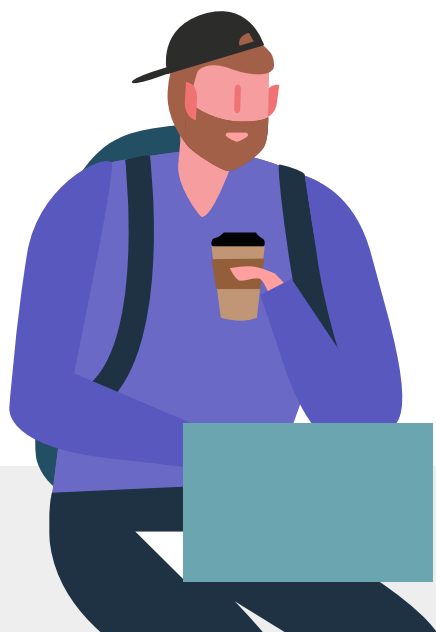
db.get(4)



```
db = kvStore()
```

	0	1	2	3
myKey	2	3	4	100

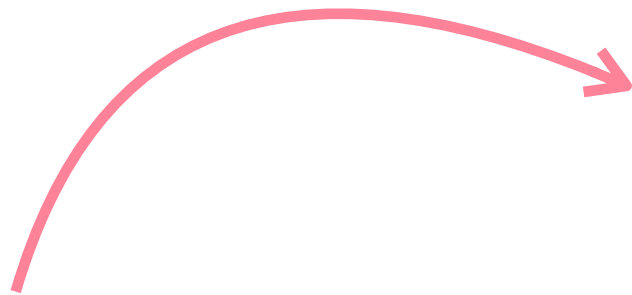
	0	1	2	3
myValue	1	7	8	2



Key-Value Store의 구현

풀이 2 : key와 value를 모두 **리스트**에 저장하기

db.get(4)



db = kvStore()

	0	1	2	3
myKey	2	3	4	100

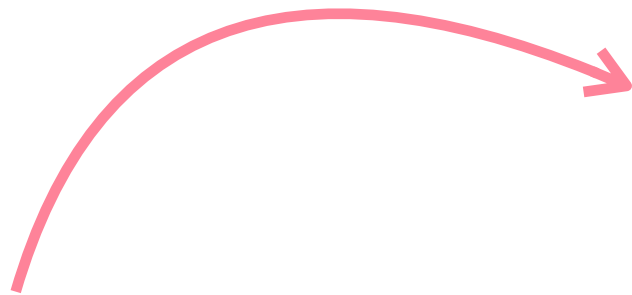
	0	1	2	3
myValue	1	7	8	2



Key-Value Store의 구현

풀이 2 : key와 value를 모두 **리스트**에 저장하기

db.get(4)



db = kvStore()

	0	1	2	3
myKey	2	3	4	100

	0	1	2	3
myValue	1	7	8	2



Key-Value Store의 구현

풀이 2 : key와 value를 모두 **리스트**에 저장하기

```
db = kvStore()
```

	0	1	2	3
myKey	2	3	4	100

	0	1	2	3
myValue	1	7	8	2

8



풀이 2의 장점과 단점

장점

공간의 낭비가 없다

“자료가 없다”를 표현할 수 있다

단점

자료의 읽기 연산이 풀이 1에 비해 느리다

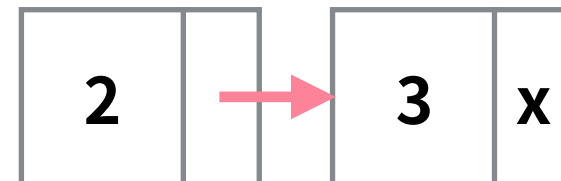
사실 자료의 쓰기 연산도 풀이 1에 비해 느리다

Key-Value Store의 구현

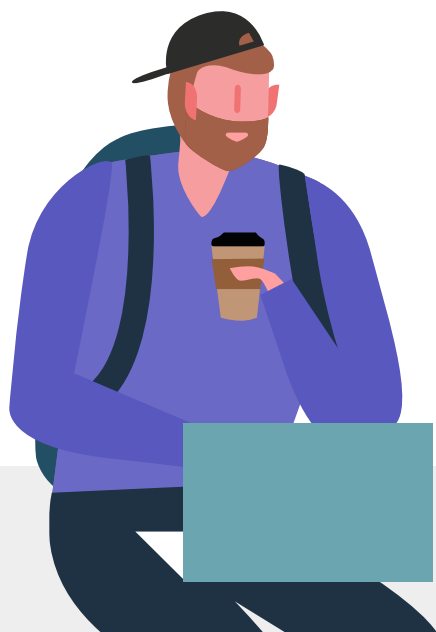
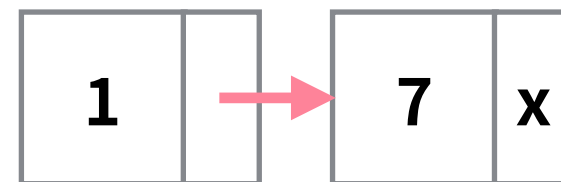
풀이 3 : key와 value를 링크드 리스트에 저장 ?

```
db = kvStore()
```

myKey



myValue



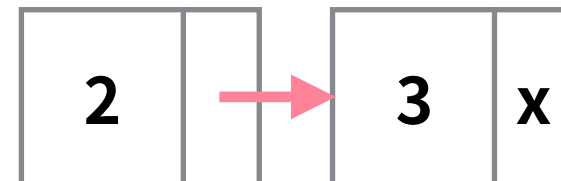
Key-Value Store의 구현

~~풀이 3 : key와 value를 링크드 리스트에 저장?~~

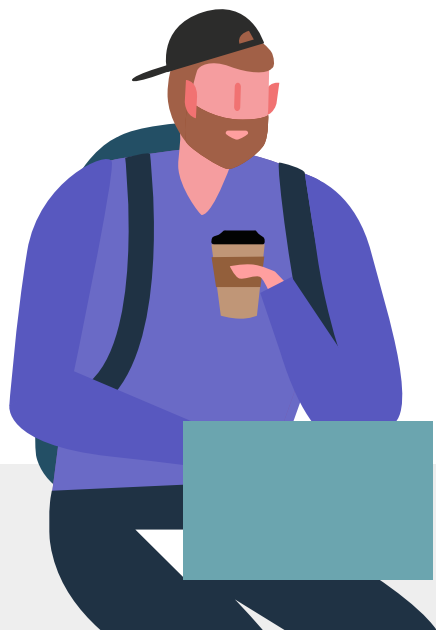
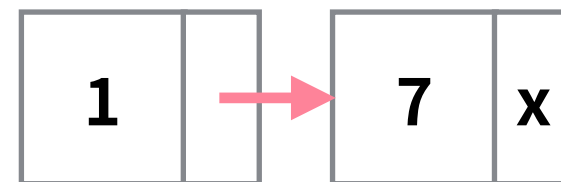
링크드 리스트의 장점을 살리지 못함

```
db = kvStore()
```

myKey



myValue



해싱 (Hashing)



제한된 공간을 이용하여

자료를 단 한번의 연산으로 찾을 수 있는 방법

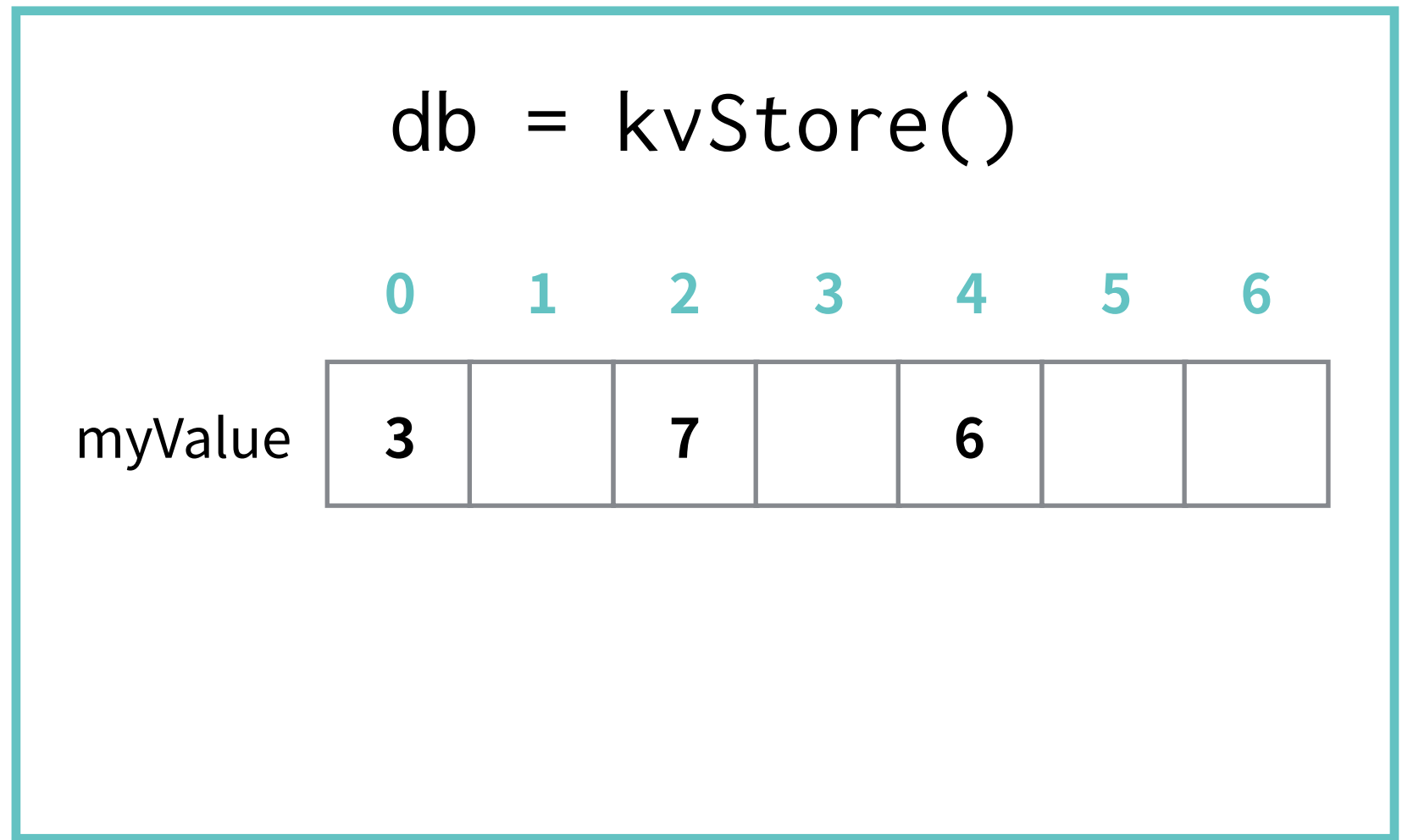
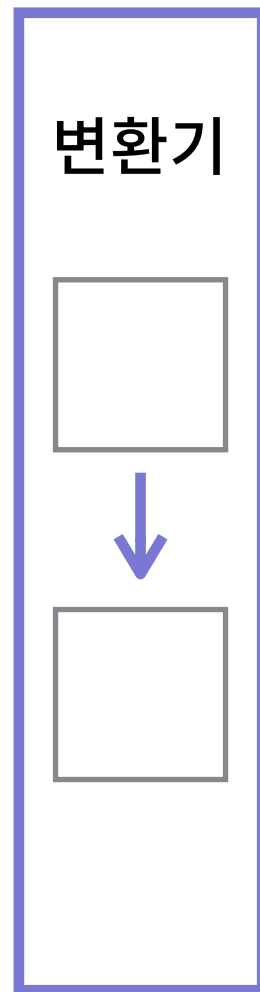
해싱 (Hashing)

(운 좋으면)

제한된 공간을 이용하여
자료를 단 한번의 연산으로 찾을 수 있는 방법

해싱의 구현

Key를 리스트의 **index**로 “잘” 변환한다



해싱의 구현

Key를 리스트의 **index**로 “잘” 변환한다

db.put(143, 2)

변환기



db = kvStore()

0 1 2 3 4 5 6

myValue

3

7

6

해싱의 구현

Key를 리스트의 **index**로 “잘” 변환한다

db.put(143, 2)

변환기

143



3

db = kvStore()

0

1

2

3

4

5

6

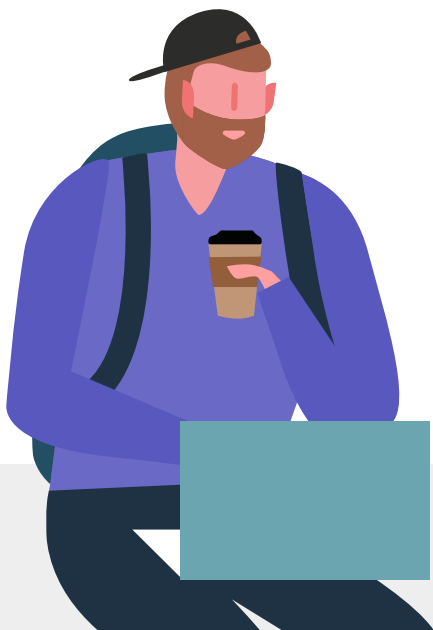
myValue

3

7

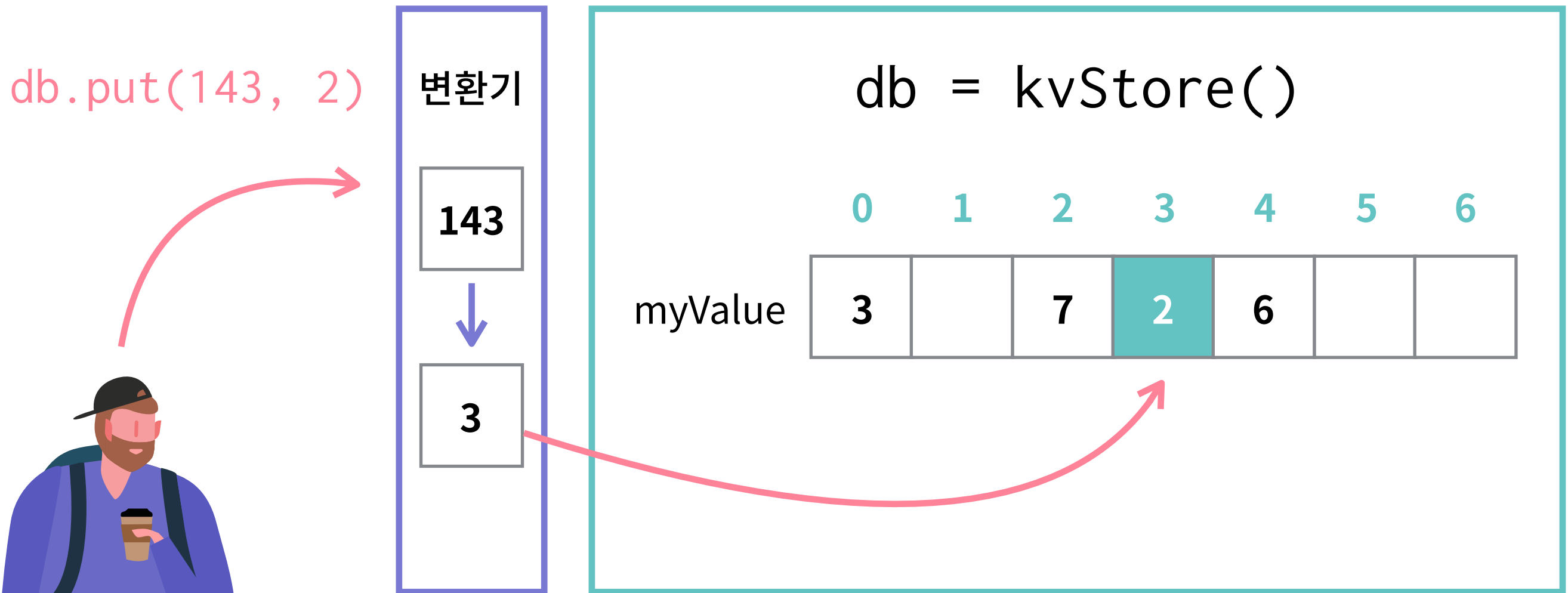
6

/* elice */



해싱의 구현

Key를 리스트의 **index**로 “잘” 변환한다



해싱의 구현

Key를 리스트의 **index**로 “잘” 변환한다

`db.put(174, 2)`

변환기



`db = kvStore()`

0 1 2 3 4 5 6

myValue

3

7

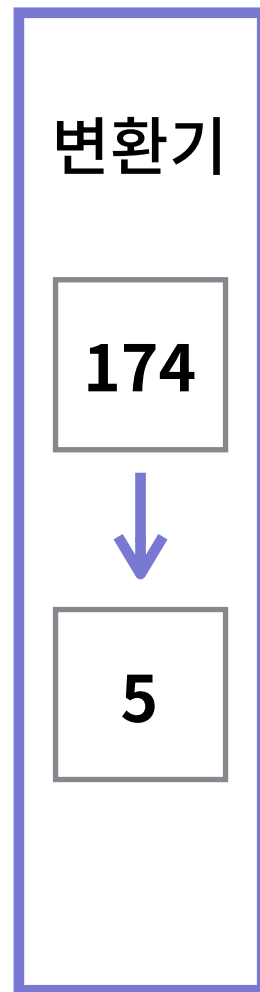
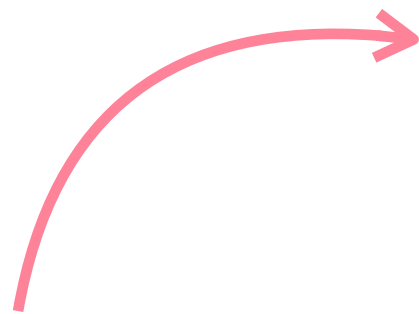
2

6

해싱의 구현

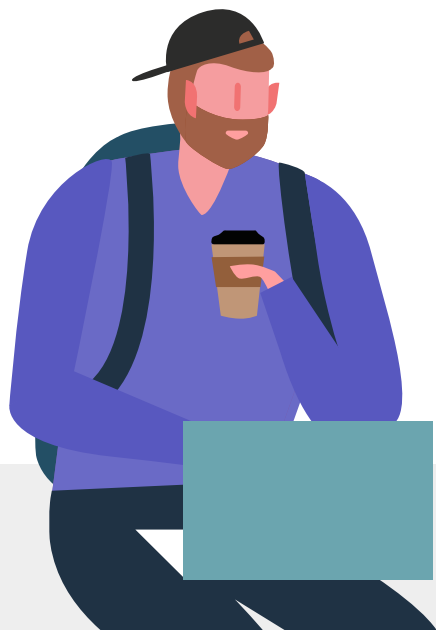
Key를 리스트의 **index**로 “잘” 변환한다

`db.put(174, 2)`



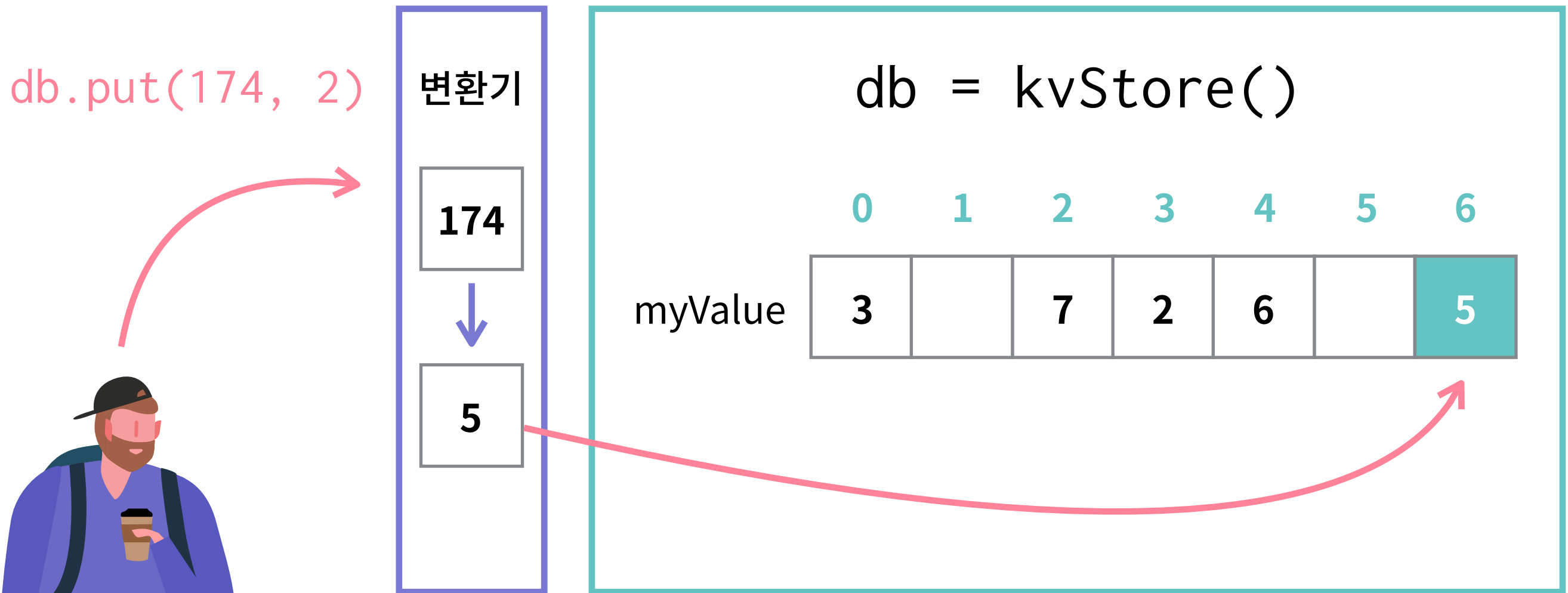
`db = kvStore()`

	0	1	2	3	4	5	6
myValue	3		7	2	6		



해싱의 구현

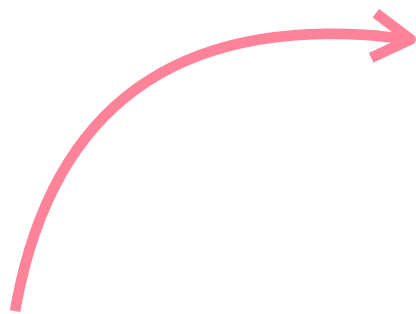
Key를 리스트의 **index**로 “잘” 변환한다



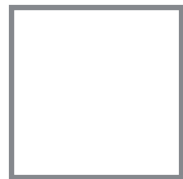
해싱의 구현

Key를 리스트의 **index**로 “잘” 변환한다

`db.get(143)`



변환기



`db = kvStore()`

0 1 2 3 4 5 6

myValue

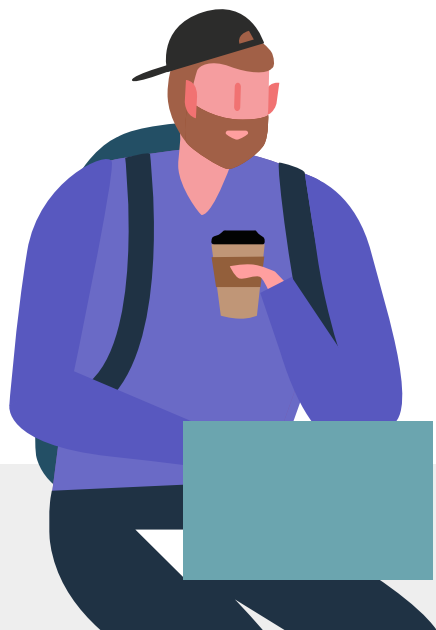
3

7

2

6

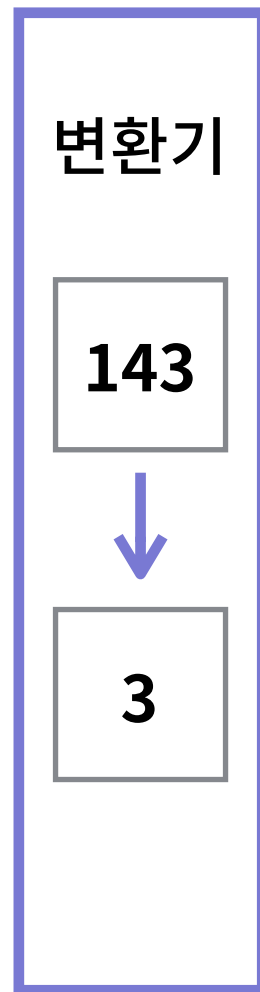
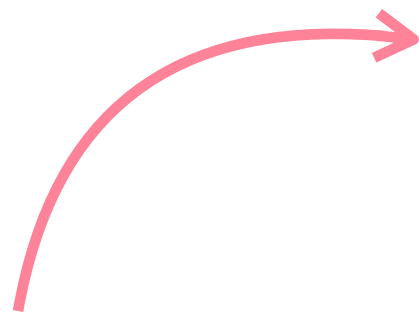
5



해싱의 구현

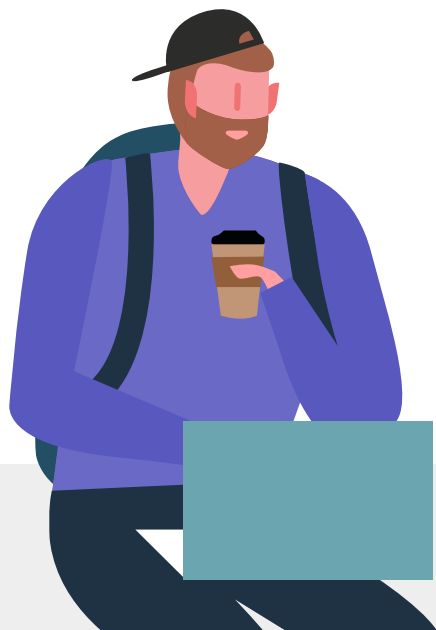
Key를 리스트의 **index**로 “잘” 변환한다

`db.get(143)`



`db = kvStore()`

	0	1	2	3	4	5	6
myValue	3		7	2	6		5



해싱의 구현

Key를 리스트의 **index**로 “잘” 변환한다

`db.get(143)`

변환기

143



3

`db = kvStore()`

0 1 2 3 4 5 6

myValue

3

7

2

6

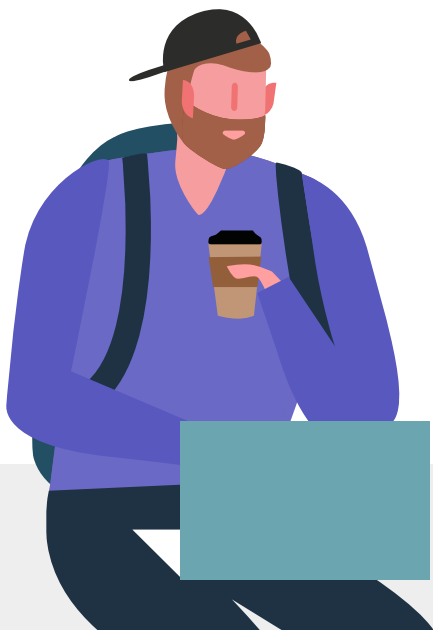
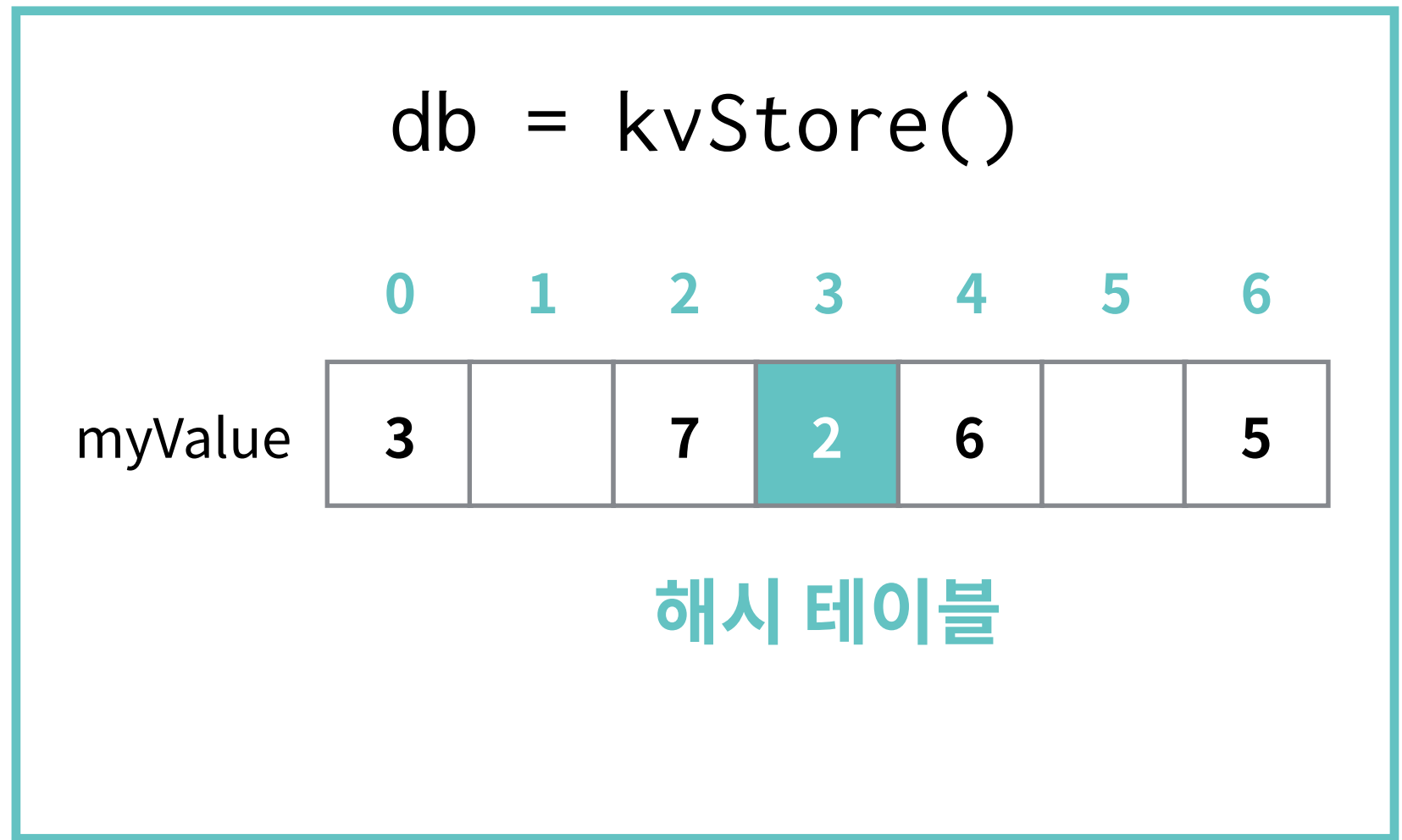
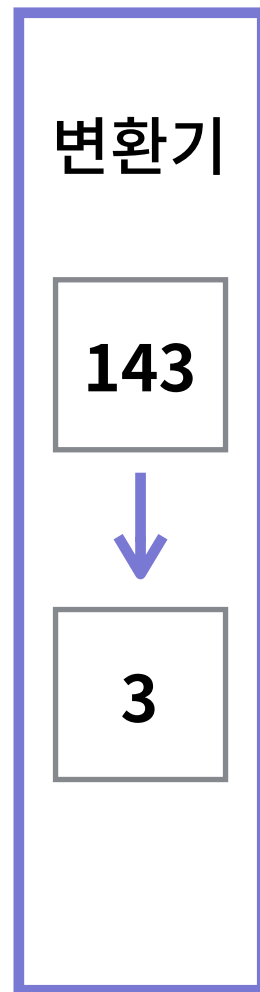
5



용어

Key를 리스트의 **index**로 “잘” 변환한다

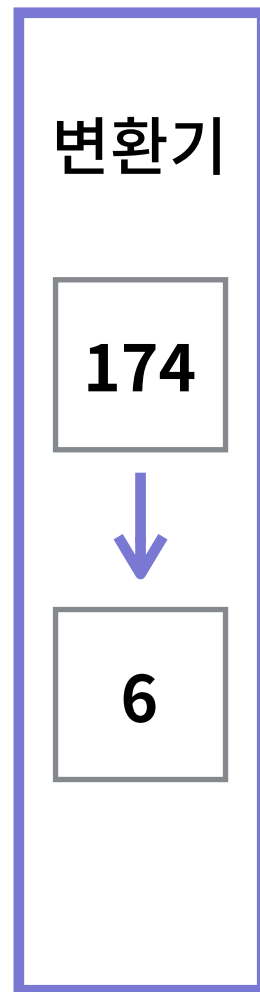
해싱 함수



해싱의 문제점

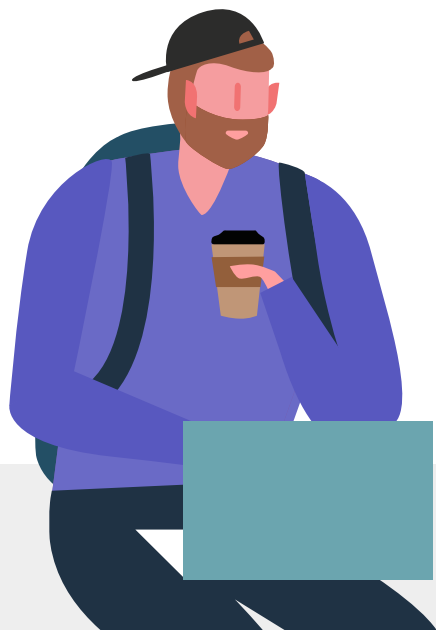
여러 key가 **하나의** index에 대응된다 (충돌)

`db.get(174)`



`db = kvStore()`

	0	1	2	3	4	5	6
myValue	3		7	2	6		5



해싱의 문제점

여러 key가 **하나의** index에 대응된다 (충돌)

`db.get(6)`

변환기

6

↓

6

`db = kvStore()`

0

1

2

3

4

5

6

myValue

3

7

2

6

5

??



해싱의 문제점

값을 읽을 때, 그리고 저장할 때 모두 문제가 발생



충돌의 해결 (읽기)

key와 value를 **함께** 저장한다

그래야 진짜 ***나의 값***인지 알 수 있음

`db.get(174)`

변환기

174



6

`db = kvStore()`

0

1

2

3

4

5

6

myValue

(70, 3)

(16, 7)

(24, 3)

(88, 6)

(174, 5)



충돌의 해결 (읽기)

key와 value를 **함께** 저장한다

그래야 진짜 ***나의 값***인지 알 수 있음

`db.get(174)`

변환기

174



6

`db = kvStore()`

0

1

2

3

4

5

6

myValue

(70, 3)

(16, 7)

(24, 3)

(88, 6)

(174, 5)



충돌의 해결 (읽기)

key와 value를 **함께** 저장한다

그래야 진짜 ***나의 값***인지 알 수 있음

`db.get(174)`

변환기

174



6

`db = kvStore()`

0

1

2

3

4

5

6

myValue

(70, 3)

(16, 7)

(24, 3)

(88, 6)

(174, 5)



충돌의 해결 (읽기)

key와 value를 **함께** 저장한다

그래야 진짜 ***나의 값***인지 알 수 있음

`db.get(6)`

변환기

6



6

`db = kvStore()`

0

1

2

3

4

5

6

myValue

(70, 3)

(16, 7)

(24, 3)

(88, 6)

(174, 5)



충돌의 해결 (읽기)

key와 value를 **함께** 저장한다

그래야 진짜 *나의 값*인지 알 수 있음

`db.get(6)`

변환기

6

6

`db = kvStore()`

0

1

2

3

4

5

6

myValue

(70, 3)

(16, 7)

(24, 3)

(88, 6)

(174, 5)

`/* elice */`

충돌의 해결 (쓰기)

key와 value를 **함께** 저장한다

값을 저장할때는 내 자리가 밀릴 수도 있다

`db.put(23, 2)`

변환기

23



2

`db = kvStore()`

0

1

2

3

4

5

6

myValue

(70, 3)

(16, 7)

(24, 3)

(88, 6)

(174, 5)



충돌의 해결 (쓰기)

key와 value를 **함께** 저장한다

값을 저장할때는 내 자리가 밀릴 수도 있다

`db.put(23, 2)`

변환기

23



2

`db = kvStore()`

0

1

2

3

4

5

6

myValue

(70, 3)

(16, 7)

(24, 3)

(88, 6)

(174, 5)



충돌의 해결 (쓰기)

key와 value를 **함께** 저장한다

값을 저장할때는 내 자리가 밀릴 수도 있다

`db.put(23, 2)`

변환기

23



2

`db = kvStore()`

0

1

2

3

4

5

6

myValue

(70, 3)

(16, 7)

(24, 3)

(88, 6)

(174, 5)



충돌의 해결 (쓰기)

key와 value를 **함께** 저장한다

값을 저장할때는 내 자리가 밀릴 수도 있다

`db.put(23, 2)`

변환기

23



2

`db = kvStore()`

0

1

2

3

4

5

6

myValue

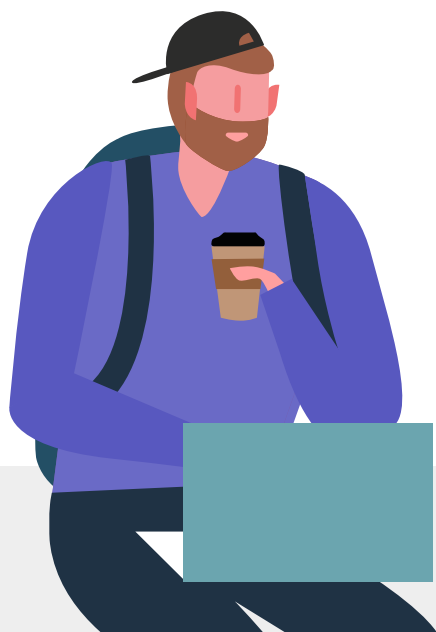
(70, 3)

(16, 7)

(24, 3)

(88, 6)

(174, 5)



충돌의 해결 (쓰기)

key와 value를 **함께** 저장한다

값을 저장할때는 내 자리가 밀릴 수도 있다

`db.put(23, 2)`

변환기

23



2

`db = kvStore()`

0

1

2

3

4

5

6

myValue

(70, 3)

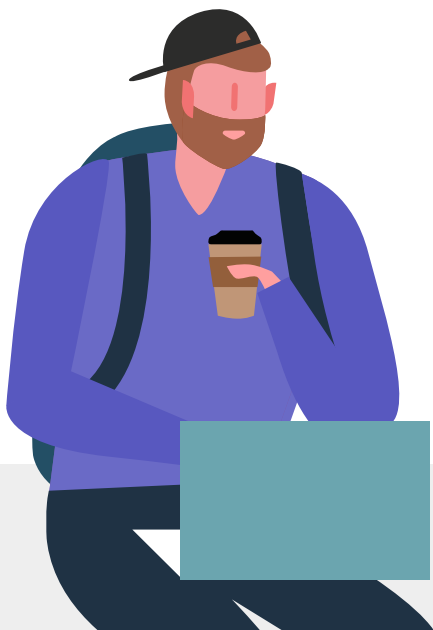
(16, 7)

(24, 3)

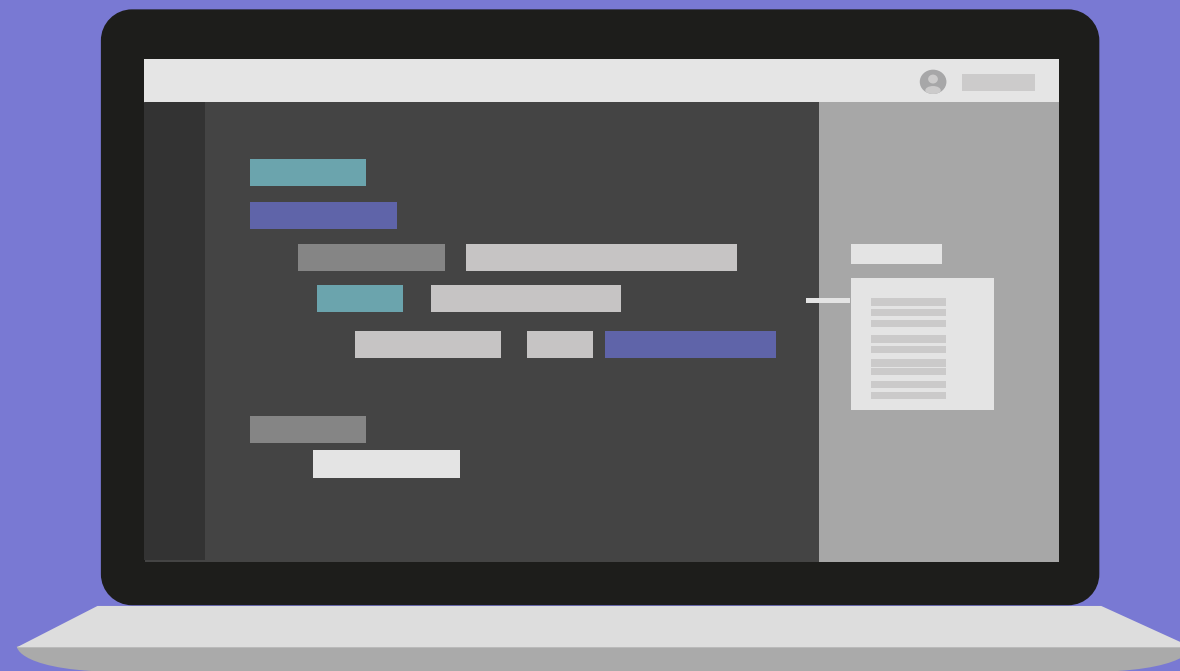
(88, 6)

(23, 2)

(174, 5)



[문제 5] 해싱 구현하기



```
/* elice */
```

얼마나 많은 연산을 수행하나 ?

운 좋을 경우

한 번에 원하는 값을 읽고 쓸 수 있음

운 나쁠 경우

내 값을 찾으려 계속해서 따라가야 함

얼마나 많은 연산을 수행하나 ?

운 좋을 경우

한 번에 원하는 값을 읽고 쓸 수 있음 $O(1)$

운 나쁠 경우

내 값을 찾으려 계속해서 따라가야 함 $O(n)$

운이 나쁘지 않으려면 ?

변환기를 굉장히 잘 만들어야 함

최대한 중복된 원소를 피해야 하기 때문

운이 나쁘지 않으려면 ?

변환기를 굉장히 잘 만들어야 함

최대한 중복된 원소를 피해야 하기 때문

하지만 완벽한 변환기는 없음

변환기의 원리를 알면, 항상 최악의 경우를 만들어 낼 수 있음

운이 나쁘지 않으려면 ?

변환기를 굉장히 잘 만들어야 함

최대한 중복된 원소를 피해야 하기 때문

하지만 완벽한 변환기는 없음

변환기의 원리를 알면, 항상 최악의 경우를 만들어 낼 수 있음

운이 엄청 나쁜 경우는 잘 없어서, 매우 많이 쓰임

일부러 같은 칸에 계속 집어넣기도 힘들다

Dictionary

```
myDictionary = {'key1': 1, 'key2': 2, 'key3': 3}  
print(myDictionary['key1'])  
myDictionary['key1'] = 4  
print(myDictionary['key1'])
```

Dictionary

```
myDictionary = {'key1': 1, 'key2': 2, 'key3': 3}
print(myDictionary['key1'])
myDictionary['key1'] = 4
print(myDictionary['key1'])
```

Dictionary의 내부는 **해싱**으로 동작함

즉, 운 나쁘면 느릴 수도 있다는 뜻
하지만 그렇게 느리게 만들기가 더 어렵다

요약

해싱으로 **Key-value store**를 구현할 수 있다

제한된 공간에서 자료를 한번에 읽고 쓰는것이 가능함

운 좋으면 빠르고, 운 나쁘면 느리다

변환기 (해시 함수)를 잘 만들어야 한다

하지만 일부러 느리게 만드는게 더 어렵다

Dictionary가 해싱으로 구현되어 있다

해싱을 일부러 구현할 필요가 없다

감사합니다!

신현규

E-mail : hyungyu.sh@kaist.ac.kr

Kakao : yougatup

/* elice */

문의 및 연락처

academy.elice.io

contact@elice.io

facebook.com/elice.io

blog.naver.com/elicer