

>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

프로그래밍의 정석

파이썬

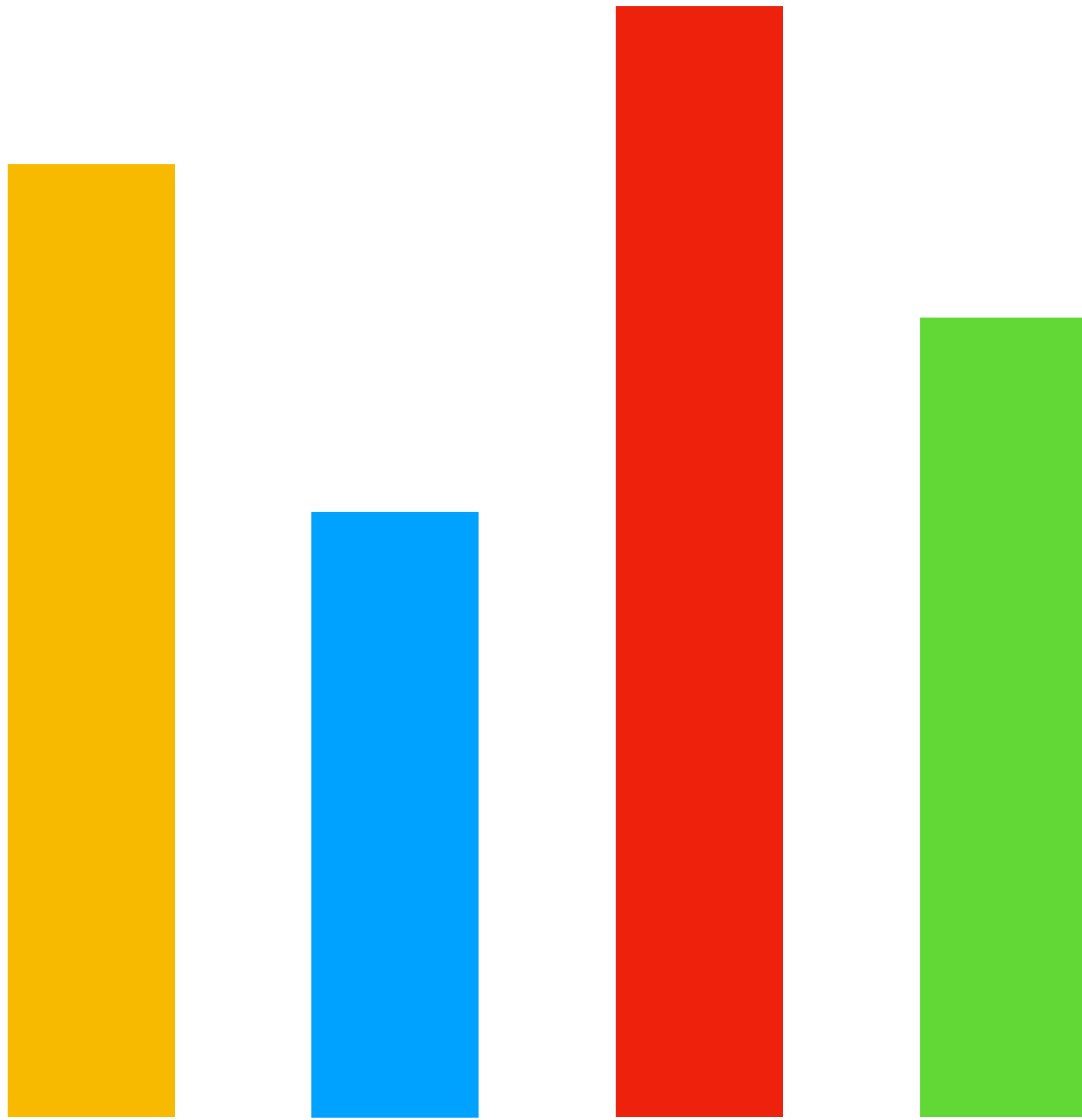
도경구 지음



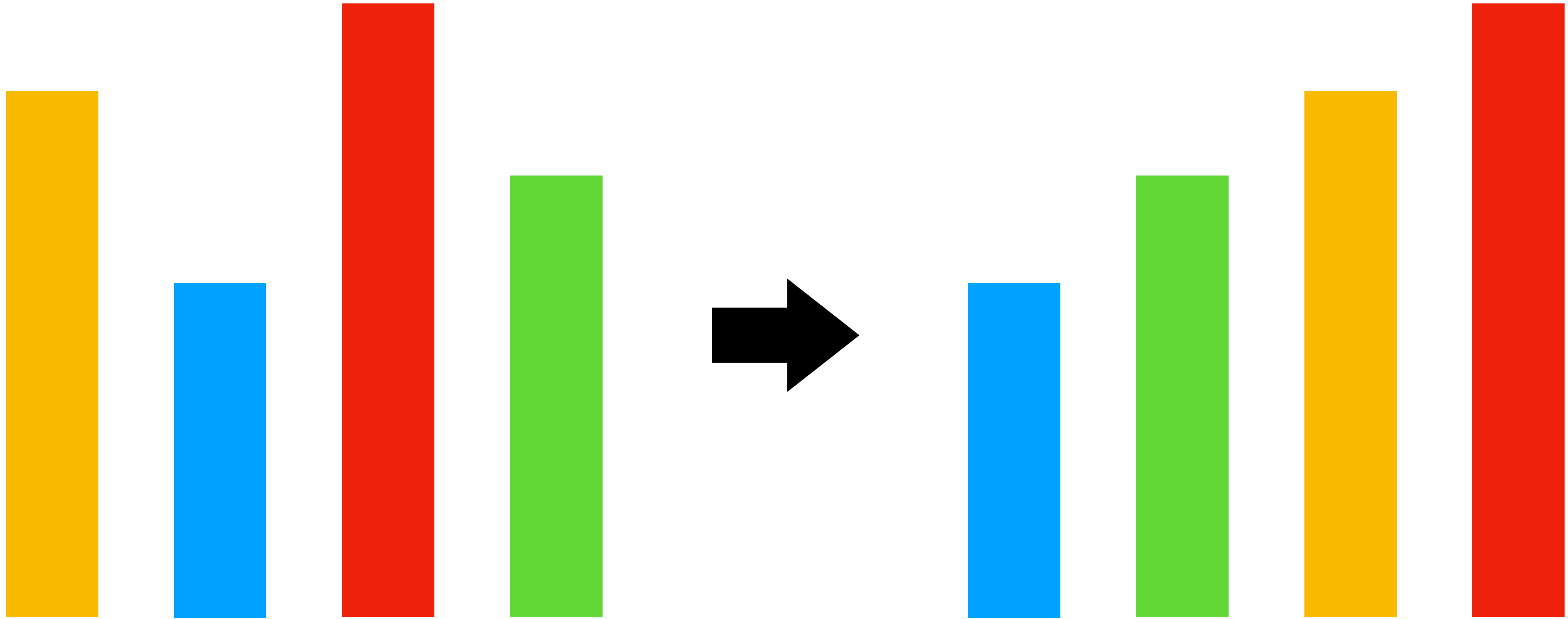
CHAPTER 5

재귀와 반복 : 정렬

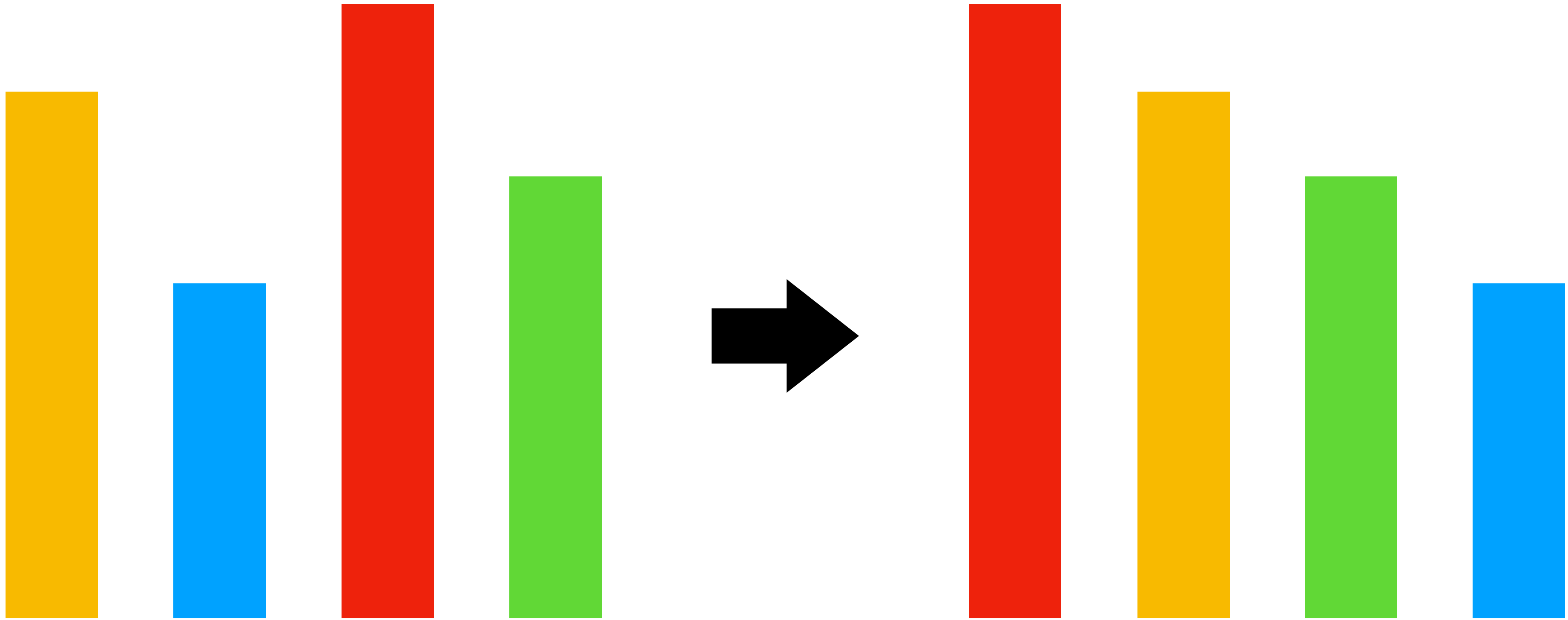
정렬 Sorting



정렬 Sorting



정렬 Sorting



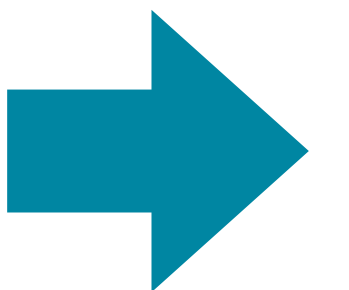
리스트 List

[3, 5, 4, 2]

["컴퓨터과학", "대학교", "소프트웨어", "인공지능"]

ns.sort()

리스트



프로그래밍의 정석
파이썬

5

재귀와 반복 : 정렬

5.1 시퀀스 · 5.2 리스트 정렬

CHAPTER 5

재귀와 반복 : 정렬

✓ 5.1 시퀀스

5.2 리스트 정렬

시퀀스

Sequence

시퀀스 타입 Sequence Type	리스트 List	[4, 6, 9, 11]	
	튜플 Tuple	('컴퓨터과학', 1, '짱')	
	정수범위 Range	range(3,9)	

시퀀스

Sequence

시퀀스 타입 Sequence Type	리스트 List	[4, 6, 9, 11]	
	튜플 Tuple	('컴퓨터과학', 1, '짱')	
	정수범위 Range	range(3,9)	
텍스트 시퀀스 타입 Text Sequence Type	문자열 String	"컴퓨터과학"	

시퀀스

Sequence

시퀀스 타입 Sequence Type	리스트 List	[4, 6, 9, 11]	수정 가능 Mutable
	튜플 Tuple	('컴퓨터과학', 1, '짱')	수정 불가능 Immutable
	정수범위 Range	range(3, 9)	
텍스트 시퀀스 타입 Text Sequence Type	문자열 String	"컴퓨터과학"	

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

odds →

1	3	5	77	9
---	---	---	----	---

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

odds →

1	3	5	77	9
---	---	---	----	---

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

odds



1	33	55	77	9
---	----	----	----	---

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28, 22+23, 23+28]
```

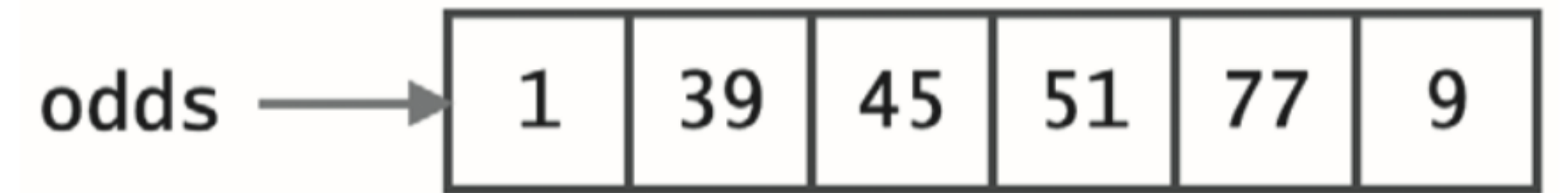



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28, 22+23, 23+28]
```



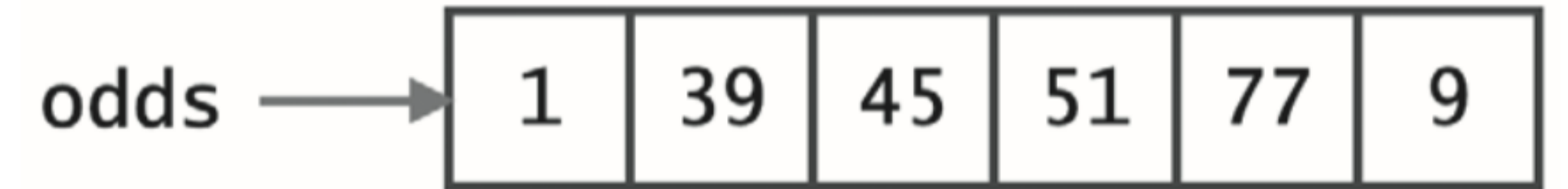
```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28, 22+23, 23+28]
```

```
>>> odds2 = odds
```



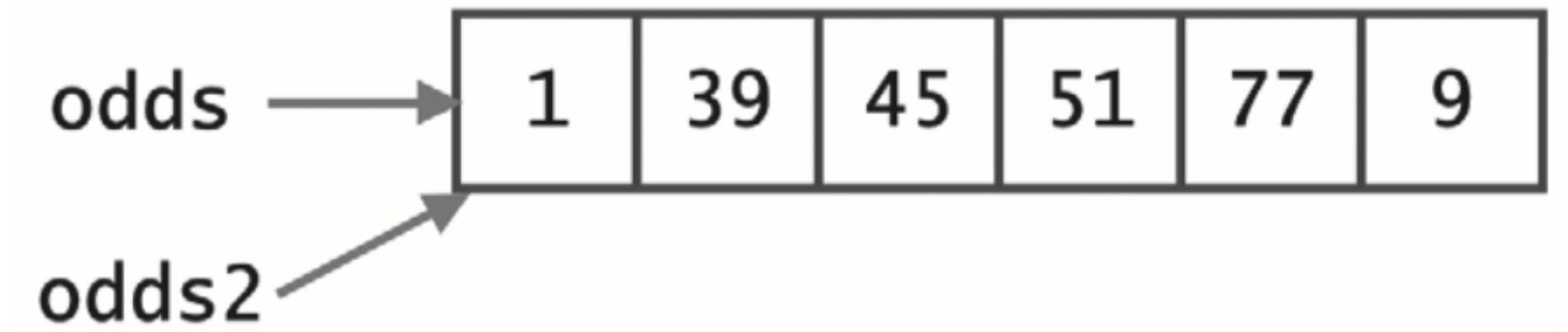
```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28, 22+23, 23+28]
```

```
>>> odds2 = odds
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

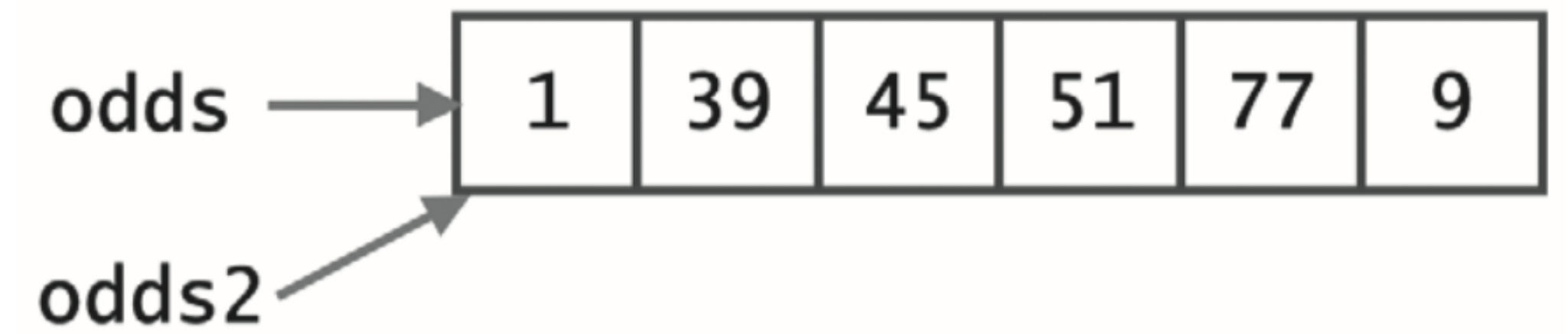
```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28, 22+23, 23+28]
```

```
>>> odds2 = odds
```

```
>>> odds2[2] = 5
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

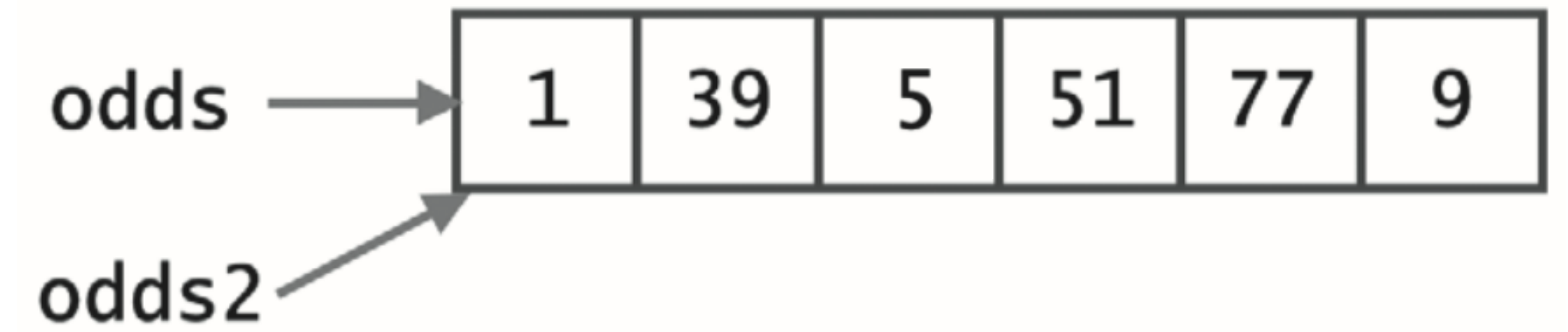
```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28, 22+23, 23+28]
```

```
>>> odds2 = odds
```

```
>>> odds2[2] = 5
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

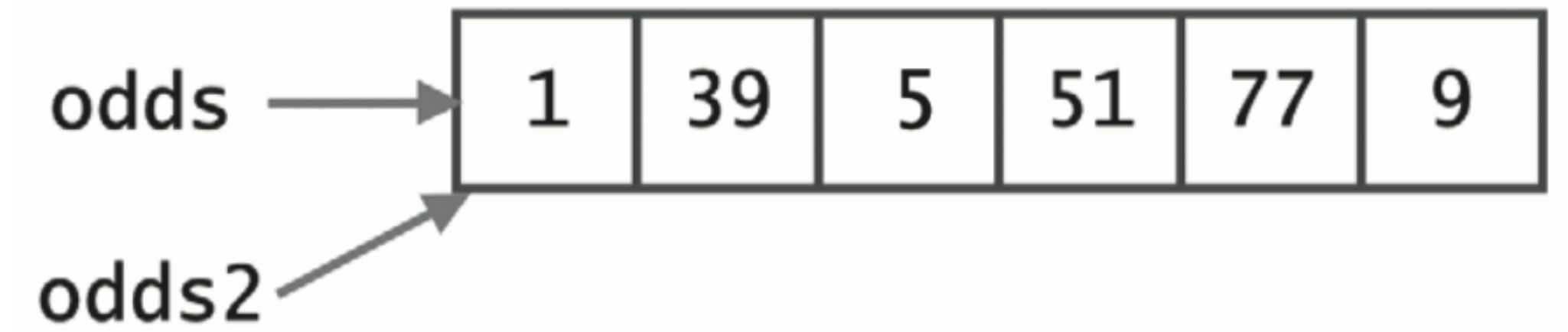
```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28, 22+23, 23+28]
```

```
>>> odds2 = odds
```

```
>>> odds2[2] = 5
```

```
>>> odds2 = odds[:]
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

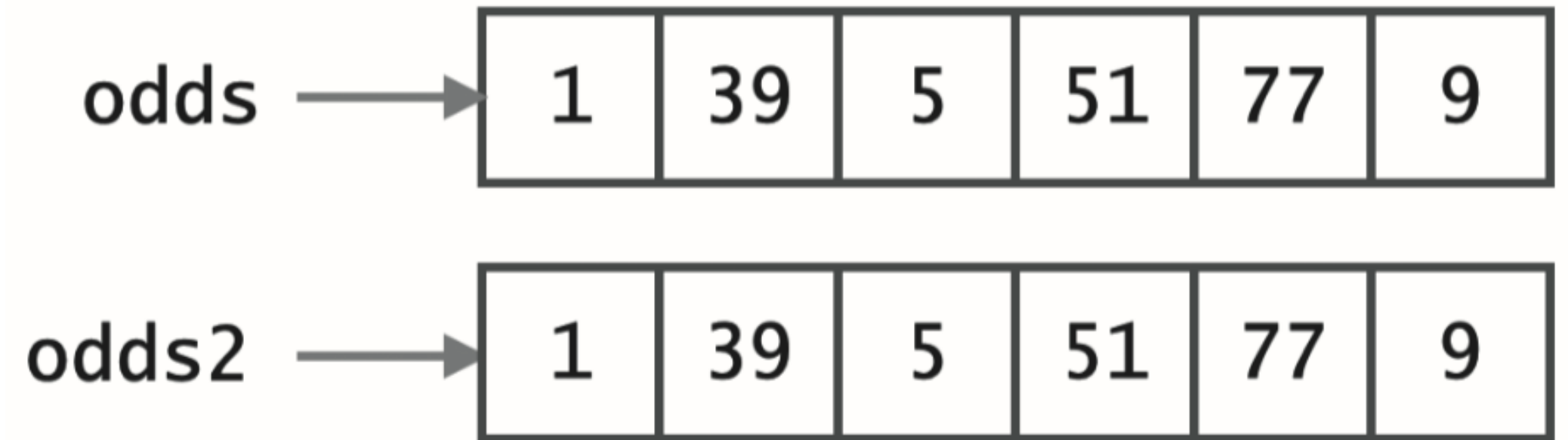
```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28, 22+23, 23+28]
```

```
>>> odds2 = odds
```

```
>>> odds2[2] = 5
```

```
>>> odds2 = odds[:]
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

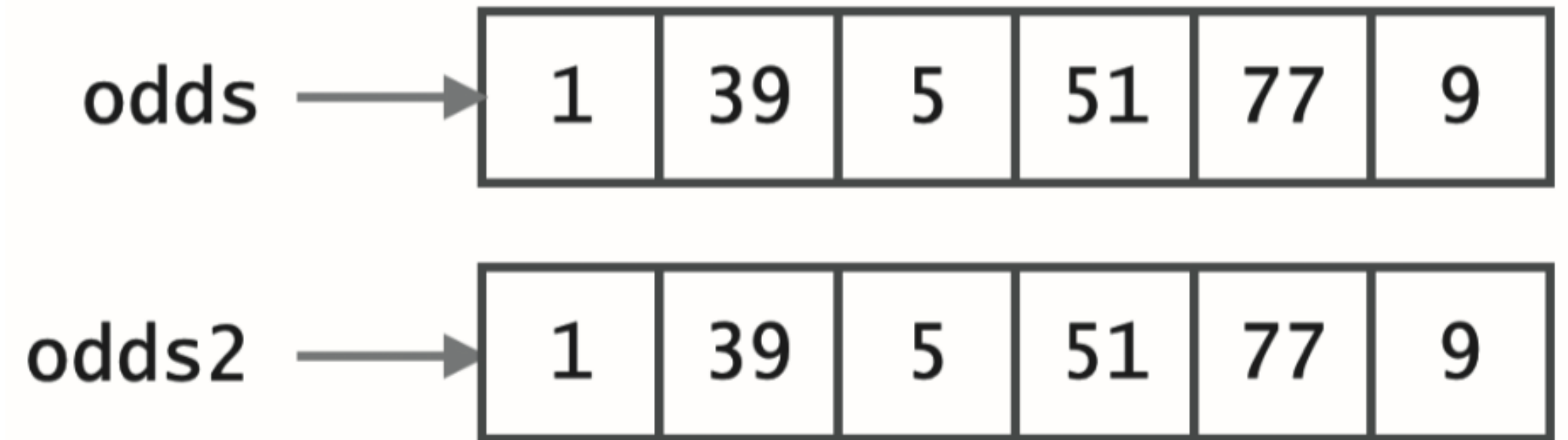
```
>>> odds[1:3] = [11+28, 22+23, 23+28]
```

```
>>> odds2 = odds
```

```
>>> odds2[2] = 5
```

```
>>> odds2 = odds[:]
```

```
>>> odds2[4] = 7
```




```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

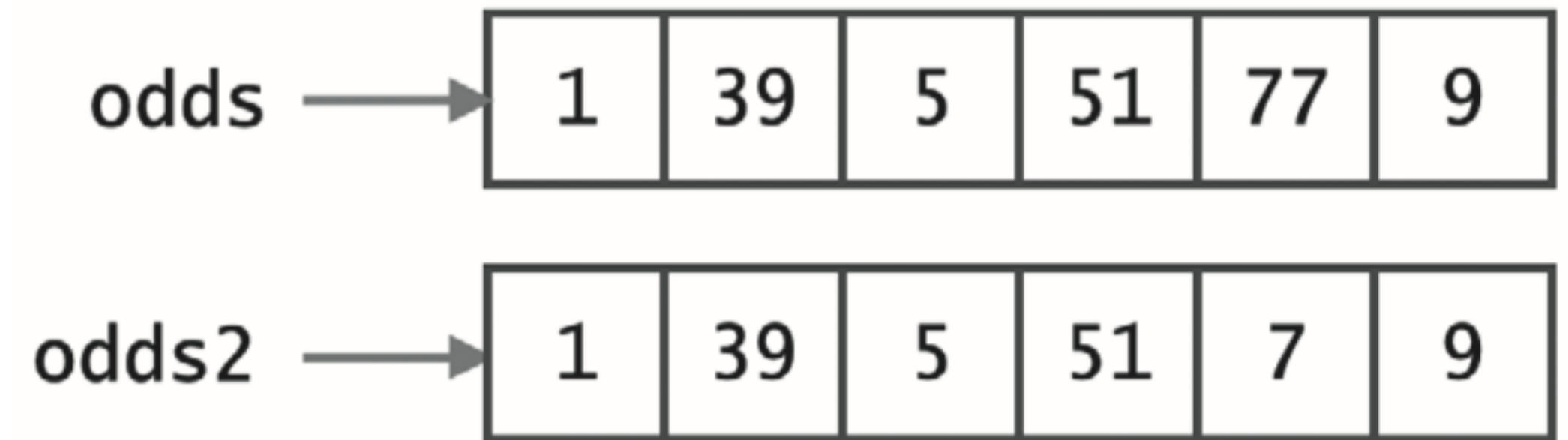
```
>>> odds[1:3] = [11+28, 22+23, 23+28]
```

```
>>> odds2 = odds
```

```
>>> odds2[2] = 5
```

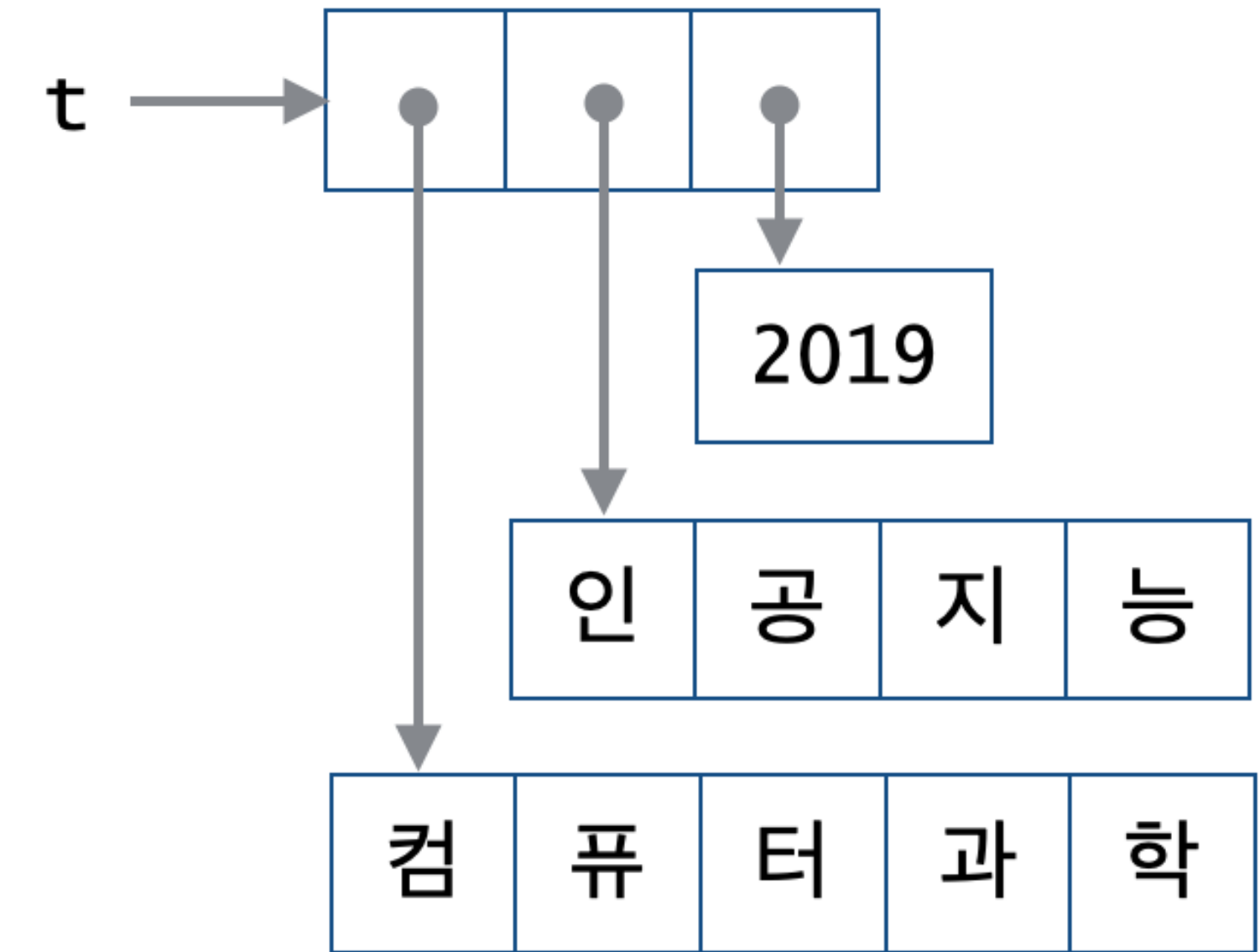
```
>>> odds2 = odds[:]
```

```
>>> odds2[4] = 7
```



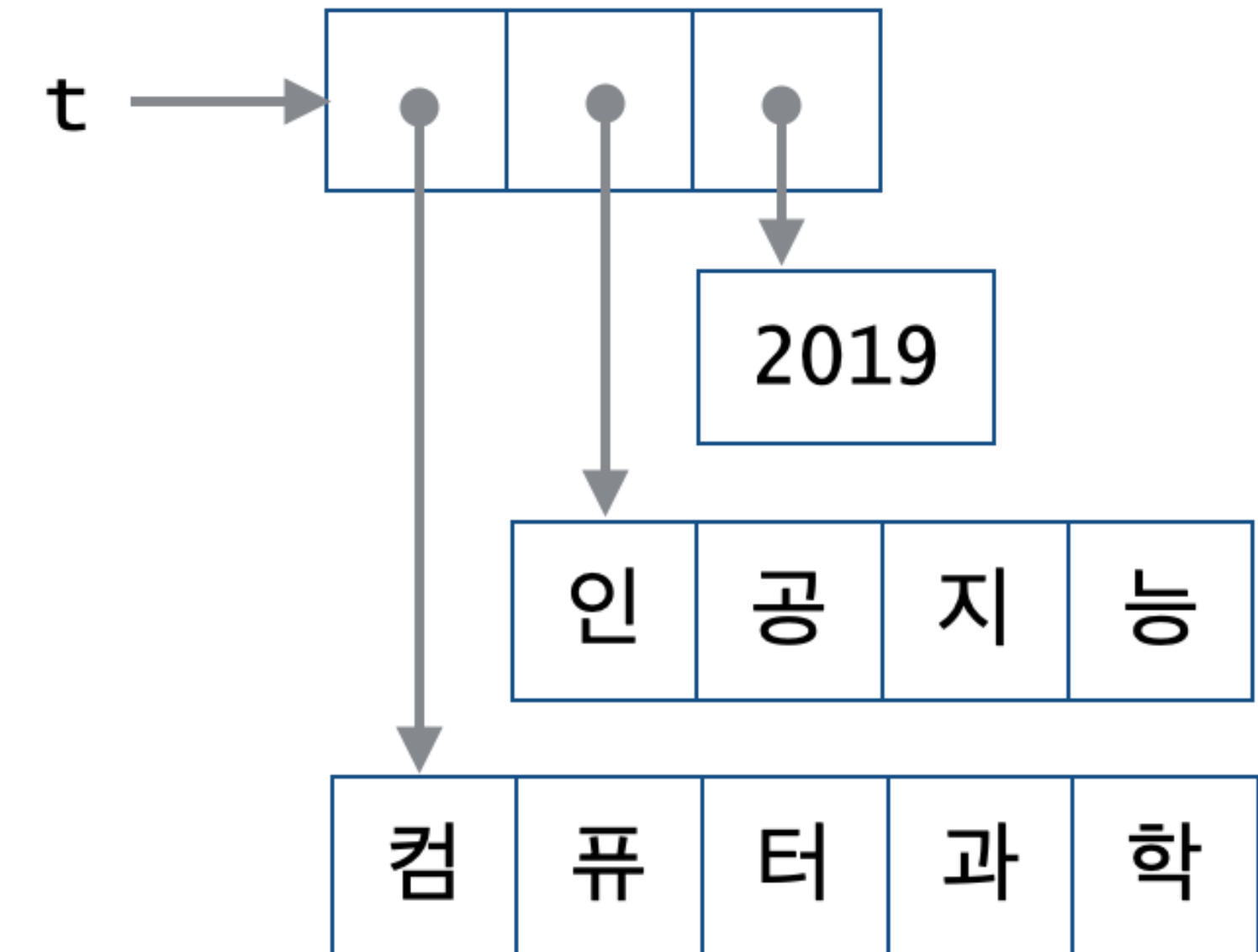
```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```



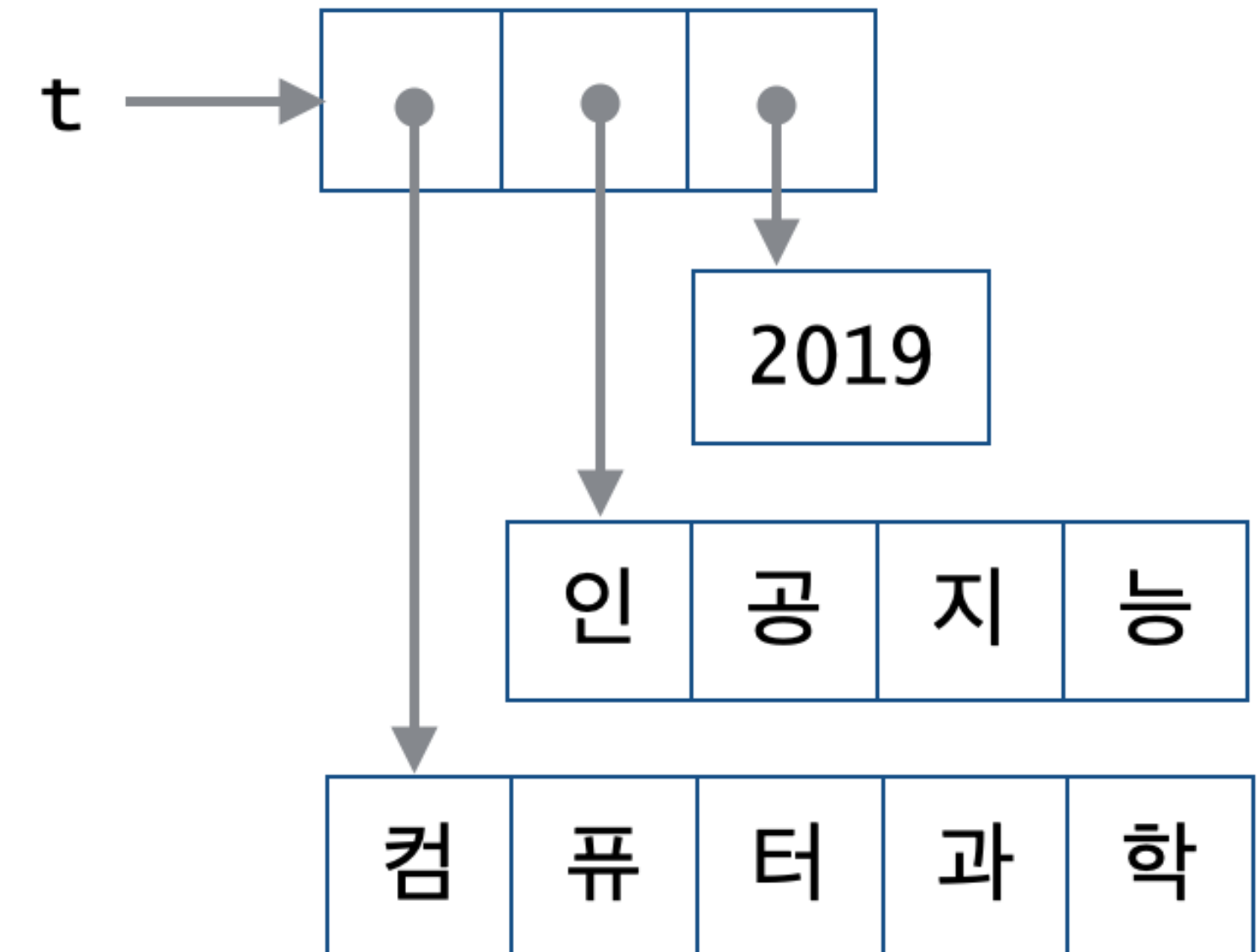
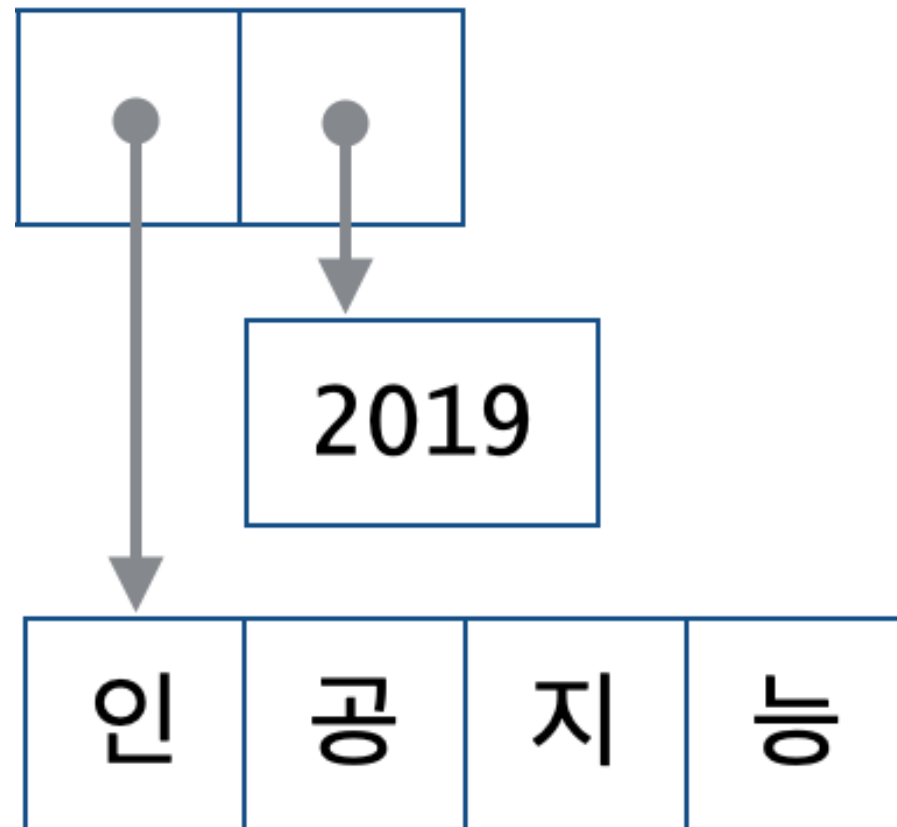
```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

```
>>> t[1:]
```



```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

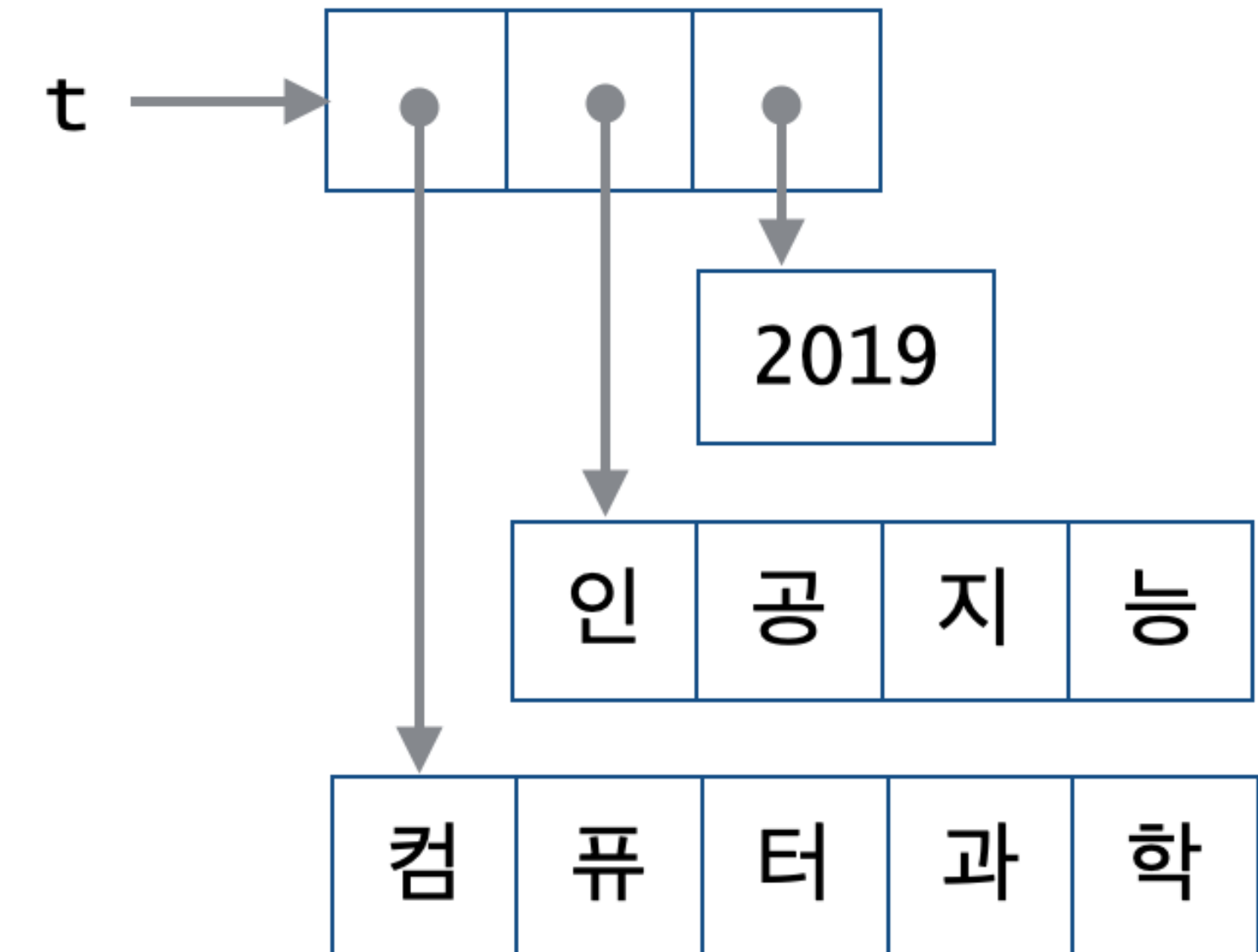
```
>>> t[1:]
```



```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

```
>>> t[1:]
```

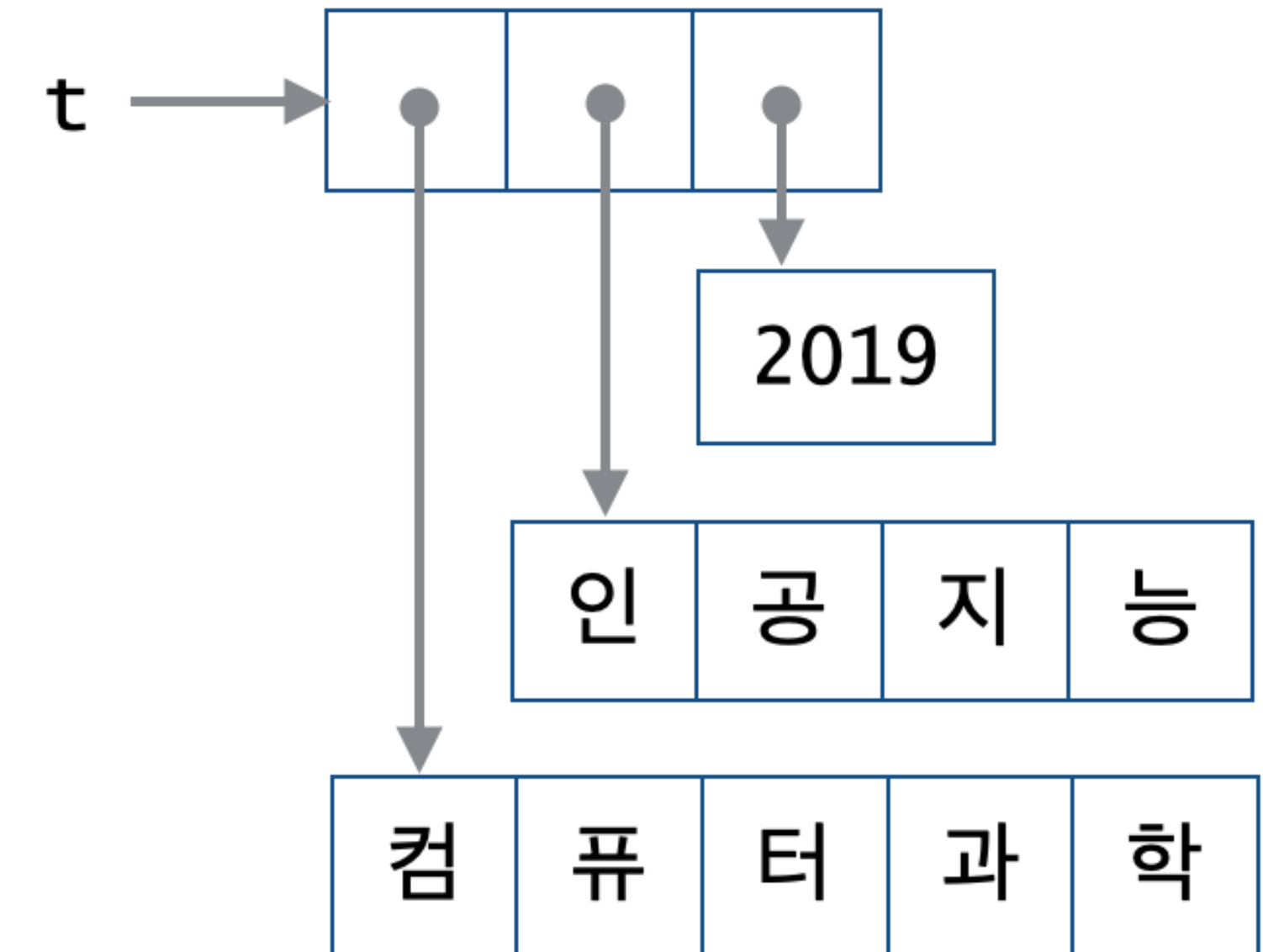
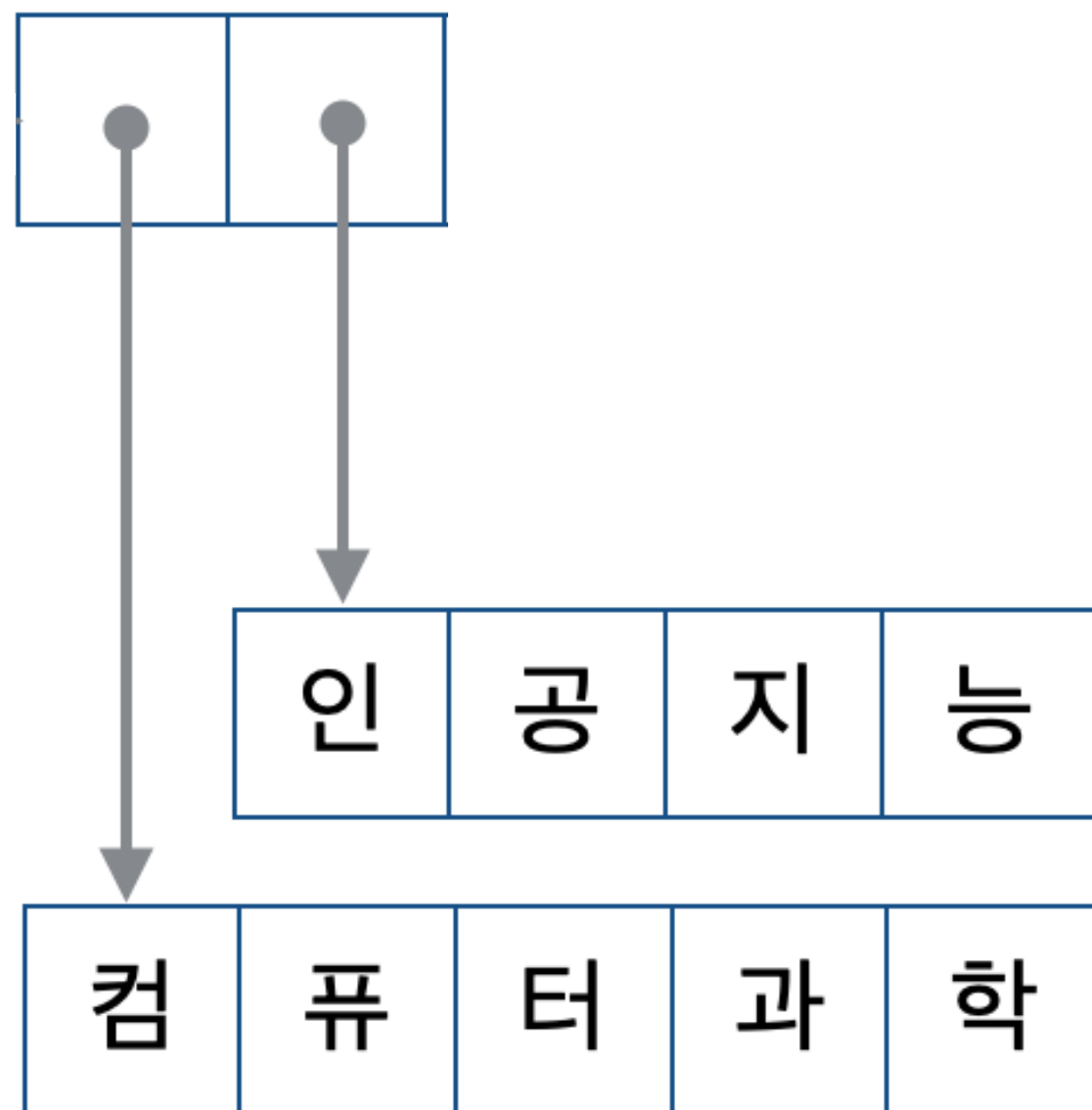
```
>>> t[:2]
```



```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

```
>>> t[1:]
```

```
>>> t[:2]
```

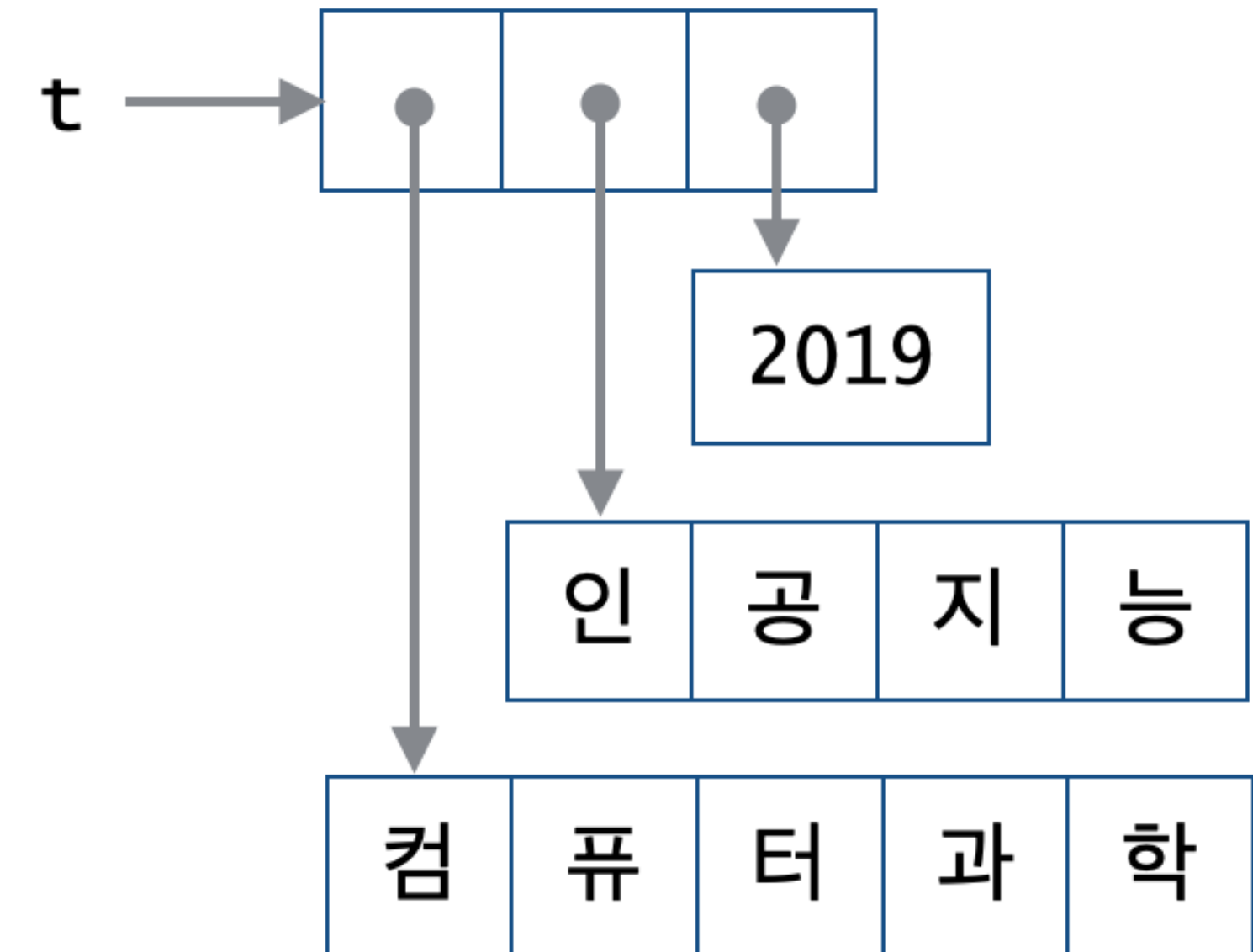


```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

```
>>> t[1:]
```

```
>>> t[:2]
```

```
>>> t[2] = '꽝'
```



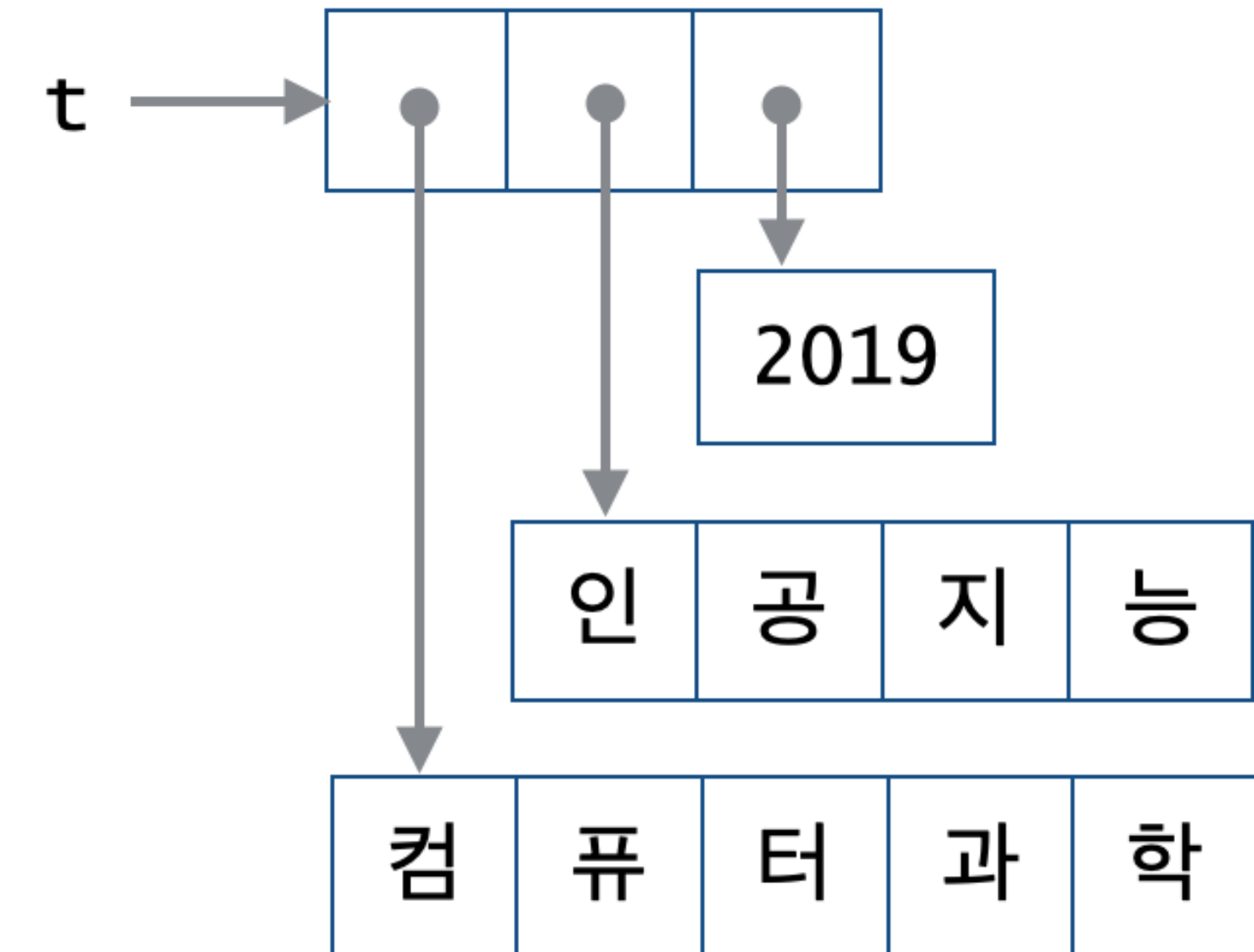

```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

```
>>> t[1:]
```

```
>>> t[:2]
```

```
>>> t[2] = '꽝'
```

```
>>> ("alone")
```



```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

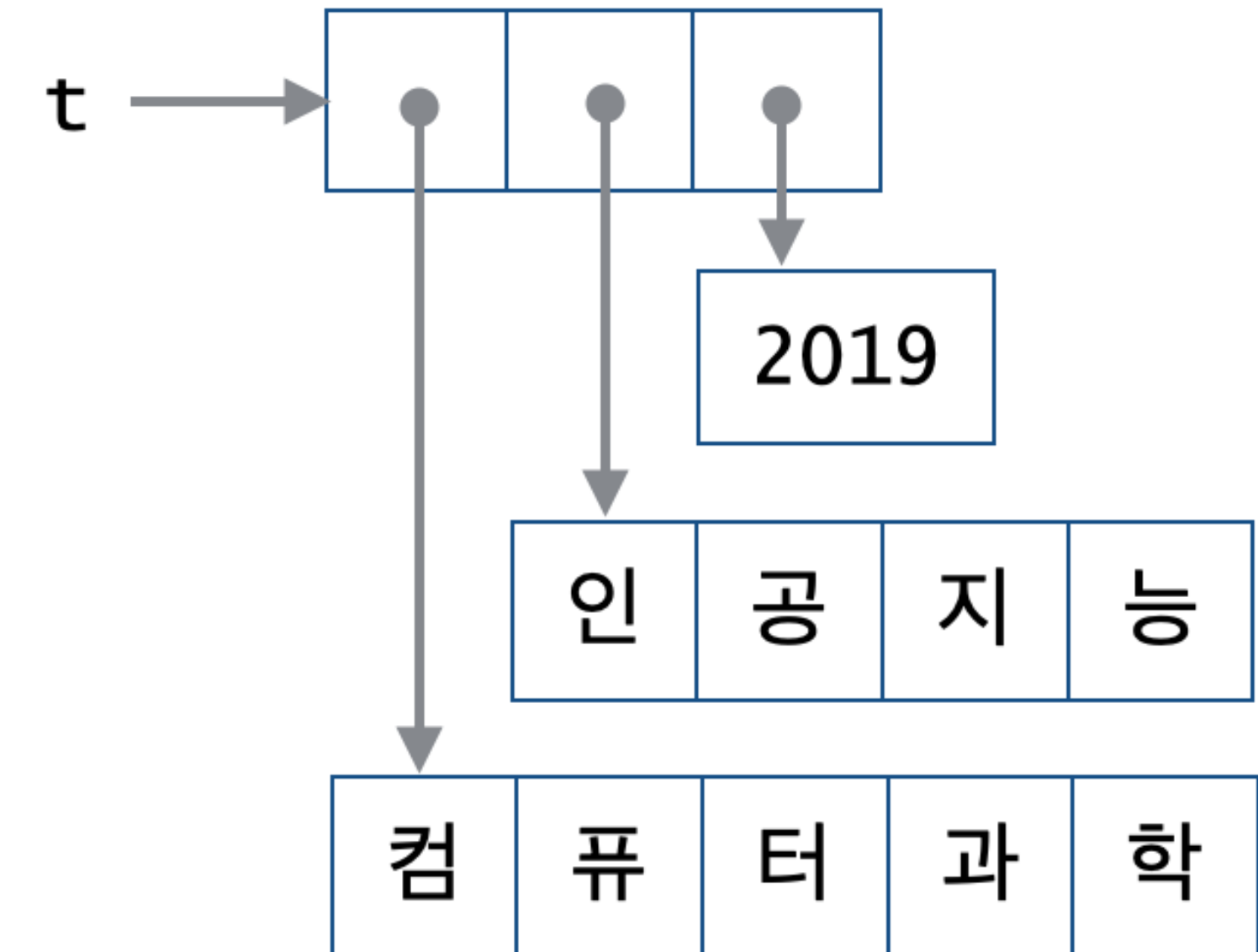
```
>>> t[1:]
```

```
>>> t[:2]
```

```
>>> t[2] = '꽝'
```

```
>>> ("alone")
```

a	l	o	n	e
---	---	---	---	---



```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

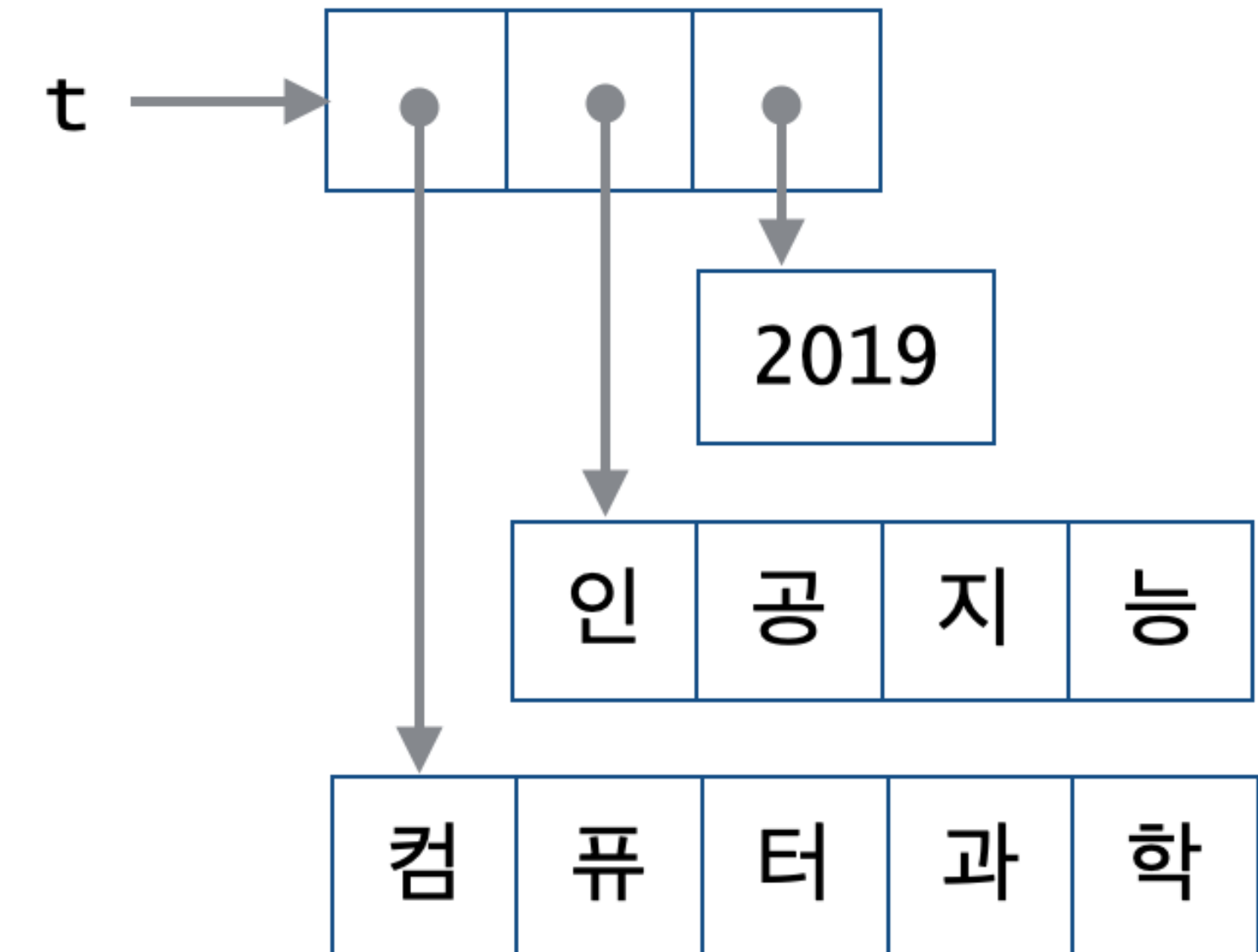
```
>>> t[1:]
```

```
>>> t[:2]
```

```
>>> t[2] = '꽝'
```

```
>>> ("alone")
```

```
>>> ("alone",)
```



```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

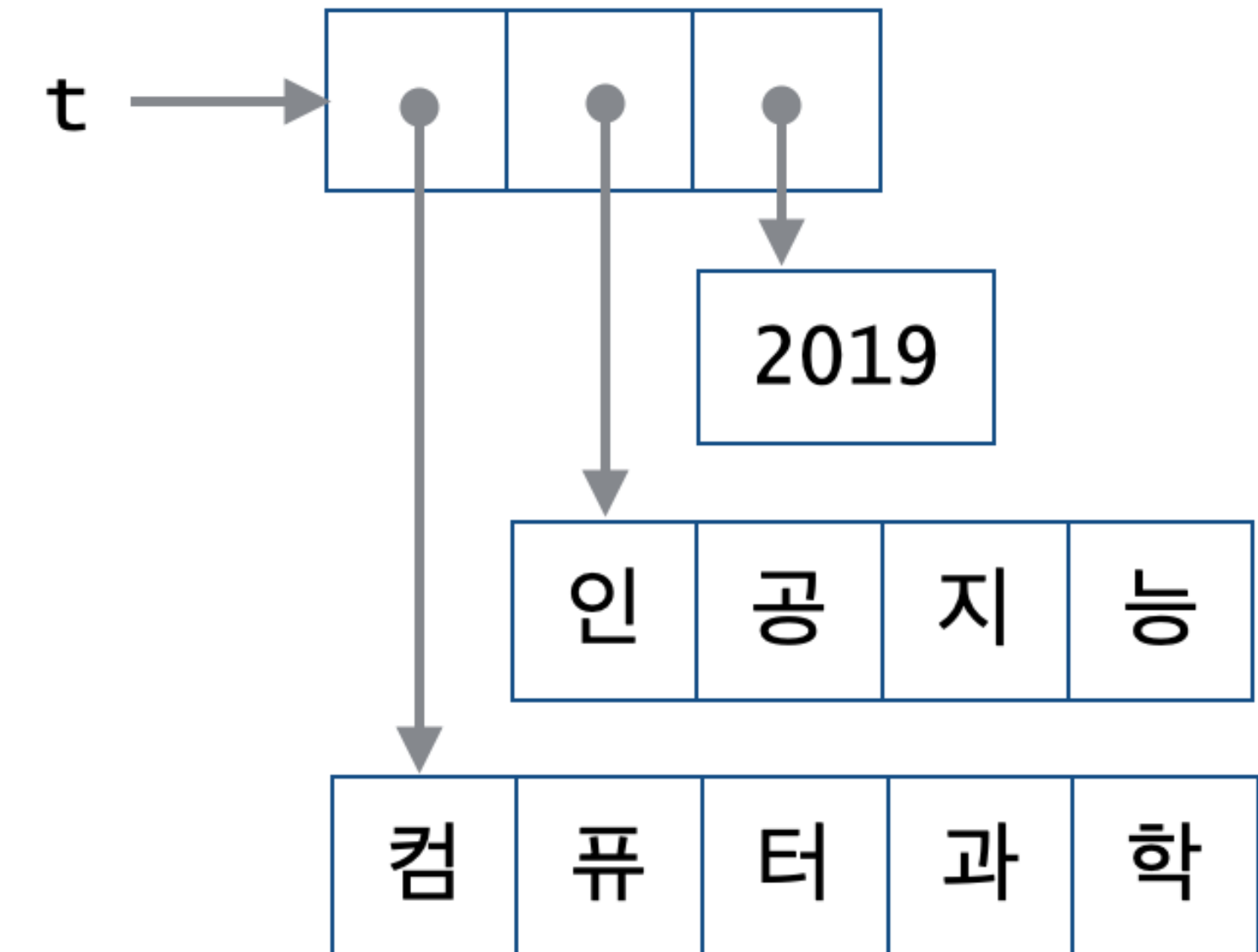
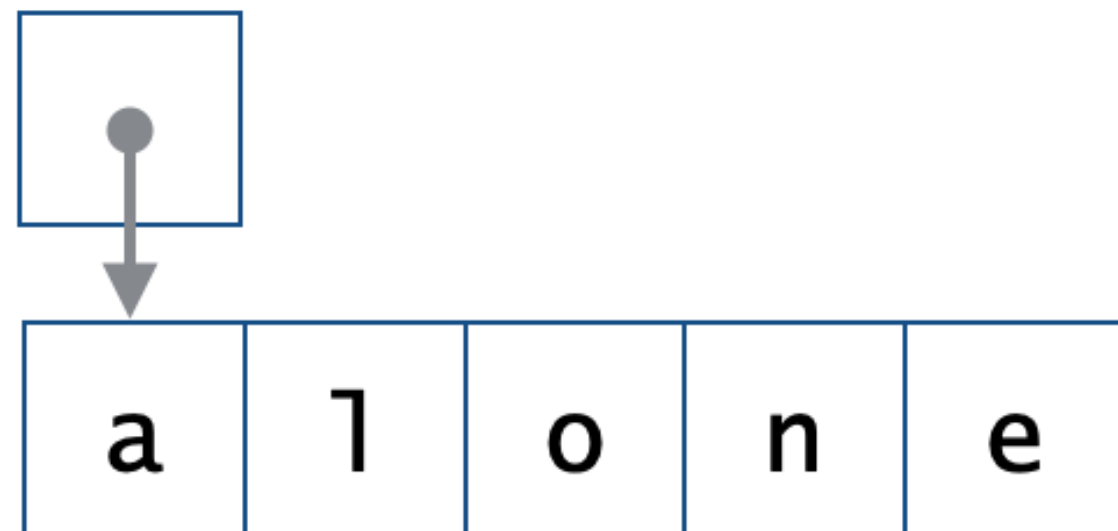
```
>>> t[1:]
```

```
>>> t[:2]
```

```
>>> t[2] = '꽝'
```

```
>>> ("alone")
```

```
>>> ("alone",)
```



```
>>> s = "컴퓨터과학"
```

```
>>> s = "컴퓨터과학"
```



```
>>> s = "컴퓨터과학"
```

```
>>> s[3] = "공"
```



```
>>> s = "컴퓨터과학"
```

```
>>> s[3] = "공"
```

```
>>> s = s + "도"
```




```
>>> s = "컴퓨터과학"
```

```
>>> s[3] = "공"
```

```
>>> s = s + "도"
```



```
>>> s = "컴퓨터과학"
```

```
>>> s[3] = "공"
```

```
>>> s = s + "도"
```



- `range(n)`은 정수 0부터 $n-1$ 까지를 간격 1로 나열한 정수 범위를 나타낸다.

`range(5)`

- `range(m,n)`은 정수 m 부터 $n-1$ 까지를 간격 1로 나열한 정수 범위를 나타낸다.

`range(3,10)`

- `range(m,n,k)`는 정수 m 부터 $n-1$ 까지를 간격 k 로 나열한 정수 범위를 나타낸다.

`range(3,11,2)`

`range(10,3,-1)`

- `range(n)`은 정수 0부터 $n-1$ 까지를 간격 1로 나열한 정수 범위를 나타낸다.

`range(5)`

0	1	2	3	4
---	---	---	---	---

- `range(m,n)`은 정수 m 부터 $n-1$ 까지를 간격 1로 나열한 정수 범위를 나타낸다.

`range(3,10)`

- `range(m,n,k)`는 정수 m 부터 $n-1$ 까지를 간격 k 로 나열한 정수 범위를 나타낸다.

`range(3,11,2)`

`range(10,3,-1)`

- `range(n)`은 정수 0부터 $n-1$ 까지를 간격 1로 나열한 정수 범위를 나타낸다.

`range(5)`

0	1	2	3	4
---	---	---	---	---

- `range(m,n)`은 정수 m 부터 $n-1$ 까지를 간격 1로 나열한 정수 범위를 나타낸다.

`range(3,10)`

3	4	5	6	7	8	9
---	---	---	---	---	---	---

- `range(m,n,k)`는 정수 m 부터 $n-1$ 까지를 간격 k 로 나열한 정수 범위를 나타낸다.

`range(3,11,2)`

`range(10,3,-1)`

- `range(n)`은 정수 0부터 $n-1$ 까지를 간격 1로 나열한 정수 범위를 나타낸다.

`range(5)`

0	1	2	3	4
---	---	---	---	---

- `range(m,n)`은 정수 m 부터 $n-1$ 까지를 간격 1로 나열한 정수 범위를 나타낸다.

`range(3,10)`

3	4	5	6	7	8	9
---	---	---	---	---	---	---

- `range(m,n,k)`는 정수 m 부터 $n-1$ 까지를 간격 k 로 나열한 정수 범위를 나타낸다.

`range(3,11,2)`

3	5	7	9
---	---	---	---

`range(10,3,-1)`

- `range(n)`은 정수 0부터 $n-1$ 까지를 간격 1로 나열한 정수 범위를 나타낸다.

`range(5)`

0	1	2	3	4
---	---	---	---	---

- `range(m,n)`은 정수 m 부터 $n-1$ 까지를 간격 1로 나열한 정수 범위를 나타낸다.

`range(3,10)`

3	4	5	6	7	8	9
---	---	---	---	---	---	---

- `range(m,n,k)`는 정수 m 부터 $n-1$ 까지를 간격 k 로 나열한 정수 범위를 나타낸다.

`range(3,11,2)`

3	5	7	9
---	---	---	---

`range(10,3,-1)`

10	9	8	7	6	5	4
----	---	---	---	---	---	---

시퀀스 연산

s, t : 시퀀스
x : 원소
i, j, k : 인덱스
n : 자연수

연산	의미
x in s	x가 s에 있으면 True, 없으면 False
x not in s	x가 s에 없으면 True, 있으면 False
s[i]	s에서 i에 있는 원소
s[i:j]	i 에서 j 까지 시퀀스 조각 (i 포함, j 제외)
s[i:j:k]	i 에서 j 까지 k간격으로 띄운 시퀀스 조각 (i 포함, j 제외)
len(s)	s의 길이
min(s)	s에서 가장 작은 원소
max(s)	s에서 가장 큰 원소
s.index(x)	s에서 가장 앞에 나오는 x의 인덱스
s.index(x,i)	s의 i 에서 시작하여 가장 앞에 나오는 x의 인덱스
s.index(x,i,j)	s의 i에서 j까지 범위에서 가장 앞에 나오는 x의 인덱스 (i 포함, j 제외)
s.count(x)	s에서 x의 빈도수
s + t	s와 t 나란히 붙이기
s * n	s를 n번 반복하여 나란히 붙이기
n * s	s를 n번 반복하여 나란히 붙이기

for 루프

for-loop

for <변수> in <시퀀스>:

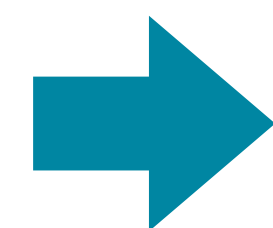
<블록>

for 루프

for-loop

```
for x in s:
```

⟨블록⟩



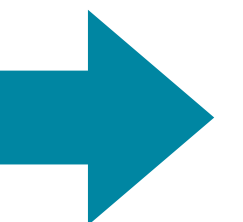
for 루프

for-loop

```
for x in s:
```

⟨블록⟩

x = s[0]



for 루프

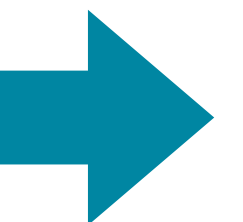
for-loop

```
for x in s:
```

⟨블록⟩

x = s[0]

x = s[1]



for 루프

for-loop

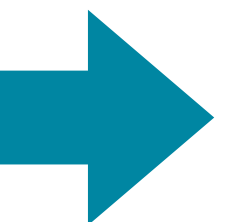
```
for x in s:
```

⟨블록⟩

x = s[0]

x = s[1]

x = s[2]



for 루프

for-loop

```
for x in s:
```

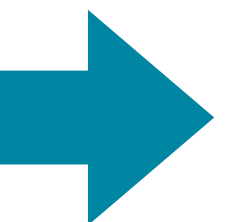
⟨블록⟩

x = s[0]

x = s[1]

x = s[2]

...



for 루프

for-loop

for x in s:

〈블록〉

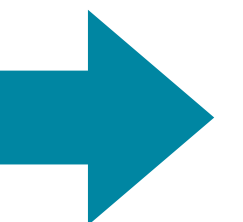
x = s[0]

x = s[1]

x = s[2]

...

x = s[len(s)-1]



>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

프로그래밍의 정석 파이썬

도경구 지음



pp.222~224



실습 5.1 for 루프 작성 연습

>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

프로그래밍의 정석

파이썬

도경구 지음



CHAPTER 5

재귀와 반복 : 정렬

프로그래밍의 정석
파이썬

5

재귀와 반복 : 정렬

5.1 시퀀스 · 5.2 리스트 정렬

CHAPTER 5

재귀와 반복 : 정렬

5.1 시퀀스

✓ 5.2 리스트 정렬

순서를 매길 수 있는 원소로 구성된 리스트



정렬
Sorting



순서대로 정렬한 리스트

리스트 vs. 배열

	인덱스로 직접 접근	길이	원소의 타입
리스트 List	불가능	가변	달라도 됨
배열 Array	가능	고정	같아야 함

파이썬의 리스트

	인덱스로 직접 접근	길이	원소의 타입
리스트 List	불가능	가변	달라도 됨
배열 Array	가능	고정	같아야 함

리스트 List

구조 귀납 歸納, 인덕 Structural Induction

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, $x :: xs$ 도 리스트이다.</p> <p>여기서, $::$는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>
(3)		그 외에 다른 리스트는 없다.

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, x :: xs 도 리스트이다.</p> <p>여기서, ::는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

[]

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, x :: xs 도 리스트이다.</p> <p>여기서, ::는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

2 :: []

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, x :: xs 도 리스트이다.</p> <p>여기서, ::는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

4 :: 2 :: []

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, $x :: xs$ 도 리스트이다.</p> <p>여기서, $::$는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

5 :: 4 :: 2 :: []

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, $x :: xs$ 도 리스트이다.</p> <p>여기서, $::$는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

3 :: 5 :: 4 :: 2 :: []

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, $x :: xs$ 도 리스트이다.</p> <p>여기서, $::$는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

3 :: 5 :: 4 :: 2 :: []

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, $x :: xs$ 도 리스트이다.</p> <p>여기서, $::$는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

3 :: 5 :: 4 :: 2 :: []

[2]

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, $x :: xs$ 도 리스트이다.</p> <p>여기서, $::$는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

3 :: 5 :: 4 :: 2 :: []

[4, 2]

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, $x :: xs$ 도 리스트이다.</p> <p>여기서, $::$는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

3 :: 5 :: 4 :: 2 :: []

[5, 4, 2]

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, x :: xs 도 리스트이다.</p> <p>여기서, ::는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

3 :: 5 :: 4 :: 2 :: []

[3, 5, 4, 2]

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, x :: xs 도 리스트이다.</p> <p>여기서, ::는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

3 :: 5 :: 4 :: 2 :: []

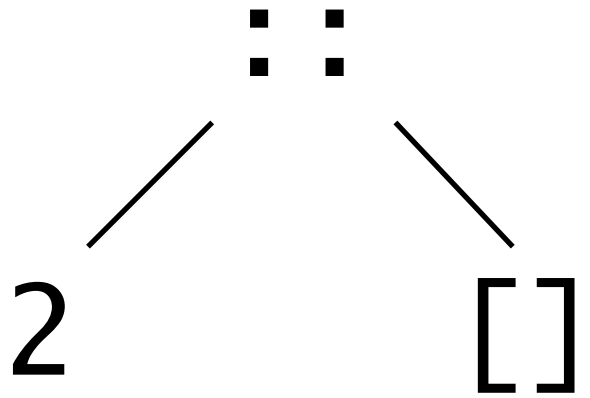
[3, 5, 4, 2]

[]

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, x :: xs 도 리스트이다.</p> <p>여기서, ::는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

3 :: 5 :: 4 :: 2 :: []

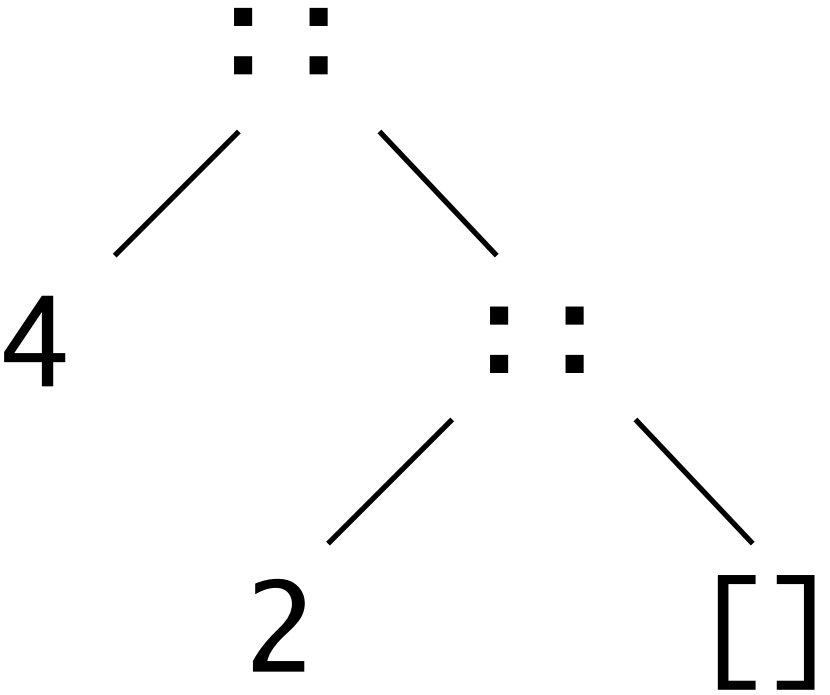
[3, 5, 4, 2]



(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, x :: xs 도 리스트이다.</p> <p>여기서, ::는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

3 :: 5 :: 4 :: 2 :: []

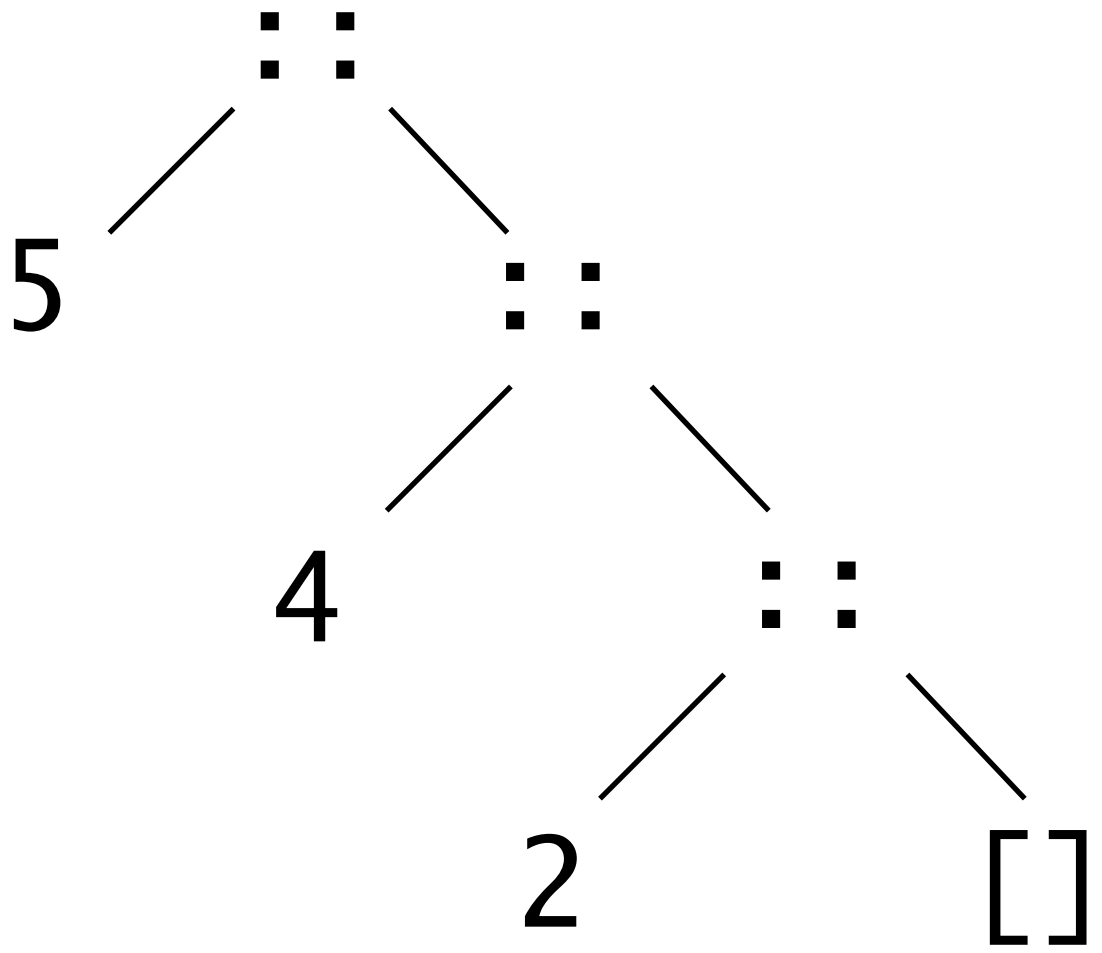
[3, 5, 4, 2]



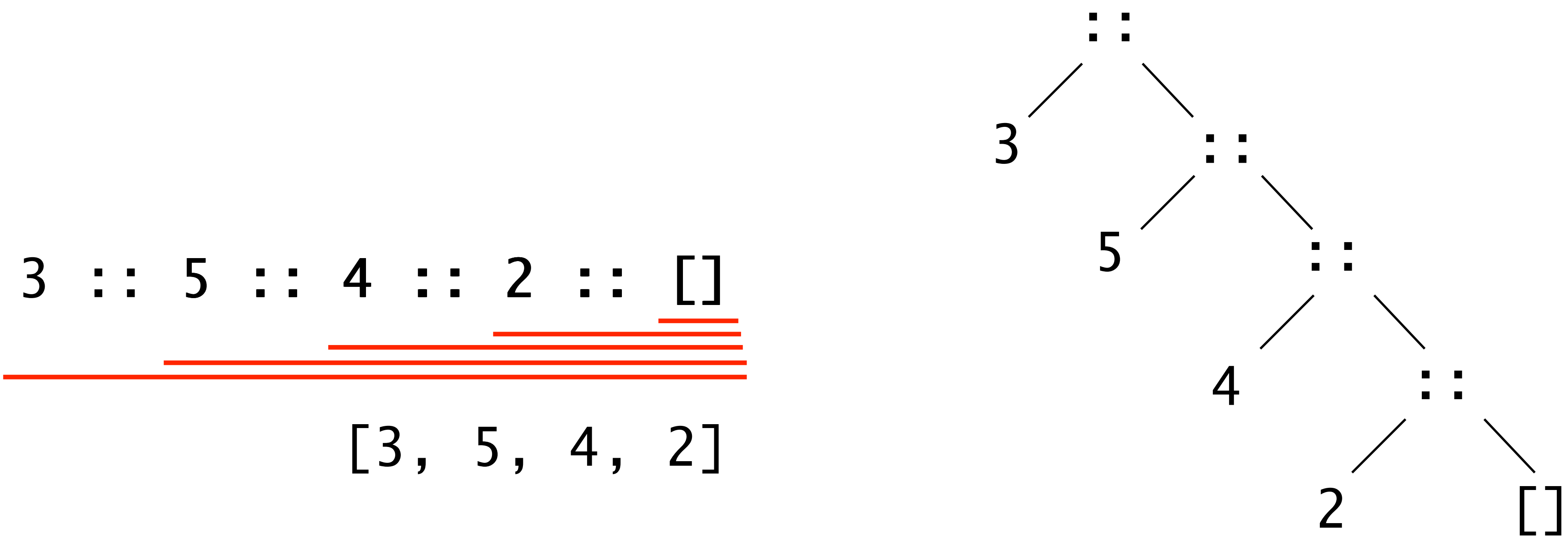
(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, x :: xs 도 리스트이다.</p> <p>여기서, ::는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

3 :: 5 :: 4 :: 2 :: []

[3, 5, 4, 2]



(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, $x :: xs$ 도 리스트이다.</p> <p>여기서, $::$는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>



(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, $x :: xs$ 도 리스트이다.</p> <p>여기서, $::$는 리스트 생성 연산자로 constructor 라고 한다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, $[x] + xs$ 도 리스트이다.</p> <p>x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, [x] + xs 도 리스트이다.</p> <p>x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

[]

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	x가 임의의 원소이고 xs가 임의의 리스트이면, [x] + xs 도 리스트이다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.

$$\underline{[2]} + \underline{[]}$$

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	x가 임의의 원소이고 xs가 임의의 리스트이면, [x] + xs 도 리스트이다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.

$$\underline{[2] + []}$$

$$[2]$$

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	x가 임의의 원소이고 xs가 임의의 리스트이면, [x] + xs 도 리스트이다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.

$$\underline{\underline{[4] + [2] + []}}$$

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	<p>x가 임의의 원소이고 xs가 임의의 리스트이면, $[x] + xs$ 도 리스트이다.</p> <p>x는 선두원소(head), xs은 후미리스트(tail)라고 한다.</p>

$$\underline{[4] + [2] + []}$$

$$[4, 2]$$

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	x가 임의의 원소이고 xs가 임의의 리스트이면, [x] + xs 도 리스트이다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.

$$\underline{\underline{[5] + [4] + [2] + []}}$$

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	x가 임의의 원소이고 xs가 임의의 리스트이면, [x] + xs 도 리스트이다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.

$$\begin{array}{ccccccc} [5] & + & [4] & + & [2] & + & [] \\ & & & & \underline{\hspace{1cm}} & & \underline{\hspace{1cm}} \\ \underline{\hspace{1cm}} & & \underline{\hspace{1cm}} & & \underline{\hspace{1cm}} & & \underline{\hspace{1cm}} \end{array}$$


$$[5, 4, 2]$$

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	x가 임의의 원소이고 xs가 임의의 리스트이면, [x] + xs 도 리스트이다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.

[3] + [5] + [4] + [2] + []

(1)	기초 Basis	빈 리스트 []은 리스트 이다.
(2)	귀납 Induction	x가 임의의 원소이고 xs가 임의의 리스트이면, [x] + xs 도 리스트이다. x는 선두원소(head), xs은 후미리스트(tail)라고 한다.

[3] + [5] + [4] + [2] + []



[3, 5, 4, 2]

선택정렬

Selection Sort

선택정렬 알고리즘

Selection Sort

리스트 xs를 정렬 하려면		
반복 조건	<code>xs != []</code>	<ul style="list-style-type: none">xs에서 가장 작은 원소를 찾아서 <code>smallest</code>로 지정하고,xs에서 <code>smallest</code>를 제거하고,xs를 재귀로 정렬하고,<code>smallest</code>가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	<code>xs == []</code>	<ul style="list-style-type: none">정렬할 원소가 없으므로 <code>[]</code>를 그대로 리턴한다.

리스트 xs를 정렬 하려면		
반복 조건	xs != []	<ul style="list-style-type: none">xs에서 가장 작은 원소를 찾아서 smallest로 지정하고,xs에서 smallest를 제거하고,xs를 재귀로 정렬하고,smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	xs == []	<ul style="list-style-type: none">정렬할 원소가 없으므로 []를 그대로 리턴한다.

[]



정렬

[]

리스트 xs를 정렬 하려면		
반복 조건	xs != []	<ul style="list-style-type: none">xs에서 가장 작은 원소를 찾아서 smallest로 지정하고,xs에서 smallest를 제거하고,xs를 재귀로 정렬하고,smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	xs == []	<ul style="list-style-type: none">정렬할 원소가 없으므로 []를 그대로 리턴한다.

정렬 대상 리스트

xs

리스트 xs를 정렬 하려면		
반복 조건	<code>xs != []</code>	<ul style="list-style-type: none">xs에서 가장 작은 원소를 찾아서 <code>smallest</code>로 지정하고,xs에서 <code>smallest</code>를 제거하고,xs를 재귀로 정렬하고,<code>smallest</code>가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	<code>xs == []</code>	<ul style="list-style-type: none">정렬할 원소가 없으므로 []를 그대로 리턴한다.



리스트 xs를 정렬 하려면		
반복 조건	<code>xs != []</code>	<ul style="list-style-type: none">xs에서 가장 작은 원소를 찾아서 <code>smallest</code>로 지정하고,xs에서 <code>smallest</code>를 제거하고,xs를 재귀로 정렬하고,<code>smallest</code>가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	<code>xs == []</code>	<ul style="list-style-type: none">정렬할 원소가 없으므로 []를 그대로 리턴한다.



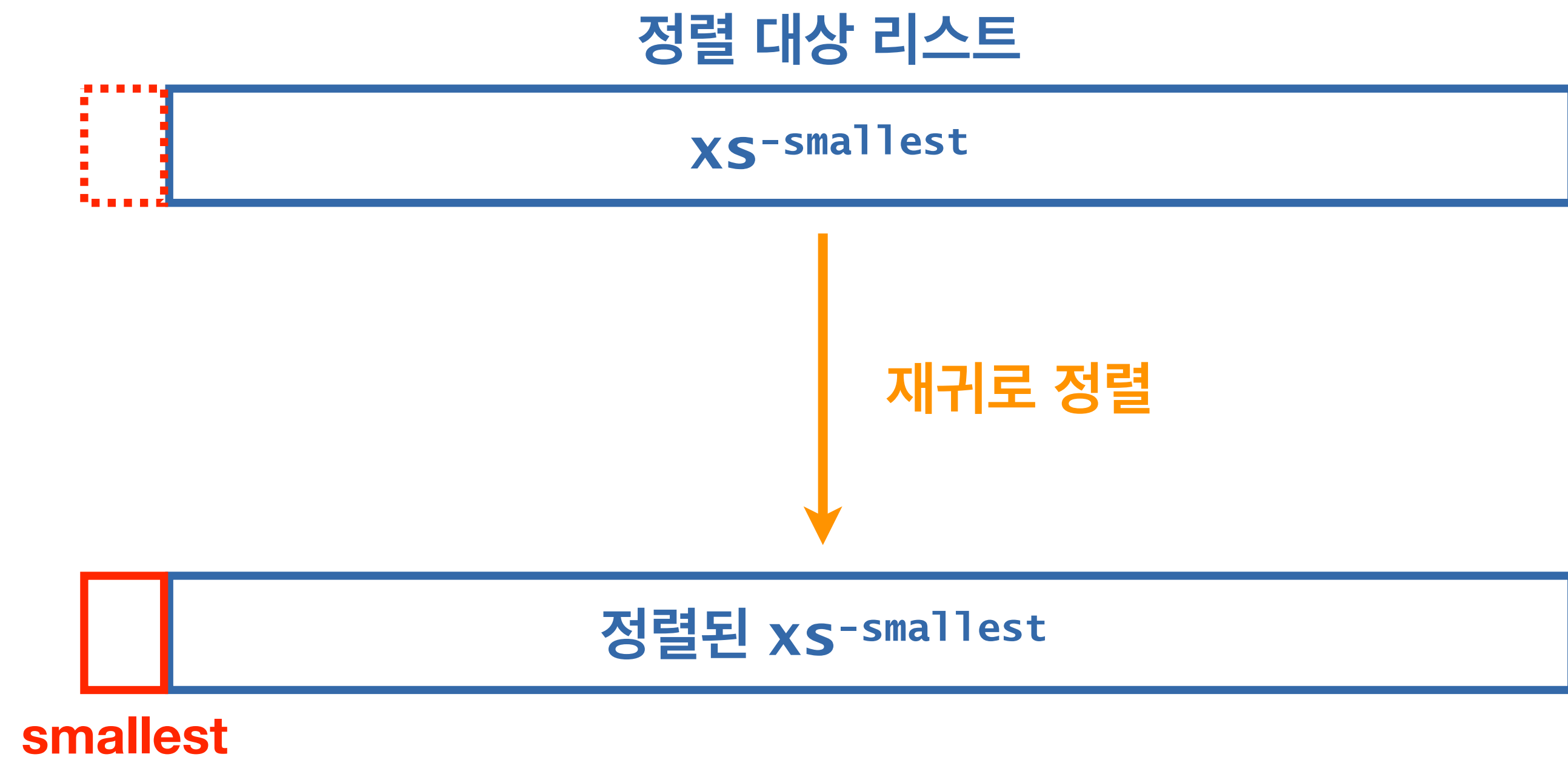
리스트 xs를 정렬 하려면		
반복 조건	<code>xs != []</code>	<ul style="list-style-type: none">xs에서 가장 작은 원소를 찾아서 <code>smallest</code>로 지정하고,xs에서 <code>smallest</code>를 제거하고,xs를 재귀로 정렬하고,<code>smallest</code>가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	<code>xs == []</code>	<ul style="list-style-type: none">정렬할 원소가 없으므로 []를 그대로 리턴한다.

정렬 대상 리스트

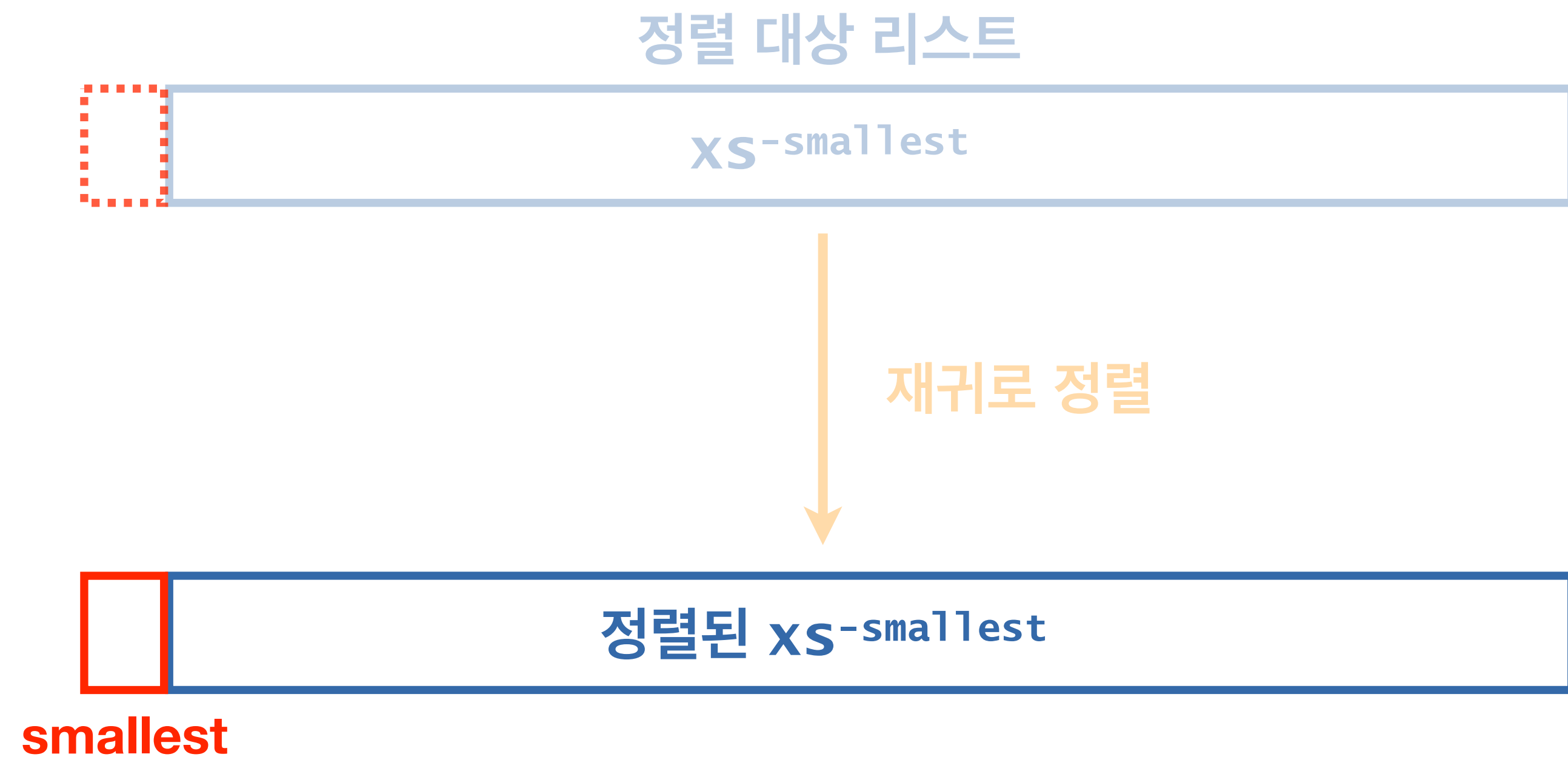



smallest

리스트 xs를 정렬 하려면		
반복 조건	<code>xs != []</code>	<ul style="list-style-type: none">xs에서 가장 작은 원소를 찾아서 <code>smallest</code>로 지정하고,xs에서 <code>smallest</code>를 제거하고,xs를 재귀로 정렬하고,<code>smallest</code>가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	<code>xs == []</code>	<ul style="list-style-type: none">정렬할 원소가 없으므로 []를 그대로 리턴한다.



리스트 xs를 정렬 하려면		
반복 조건	<code>xs != []</code>	<ul style="list-style-type: none">xs에서 가장 작은 원소를 찾아서 <code>smallest</code>로 지정하고,xs에서 <code>smallest</code>를 제거하고,xs를 재귀로 정렬하고,<code>smallest</code>가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	<code>xs == []</code>	<ul style="list-style-type: none">정렬할 원소가 없으므로 []를 그대로 리턴한다.



리스트 xs를 정렬 하려면		
반복 조건	xs != []	<ul style="list-style-type: none">xs에서 가장 작은 원소를 찾아서 smallest로 지정하고,xs에서 smallest를 제거하고,xs를 재귀로 정렬하고,smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	xs == []	<ul style="list-style-type: none">정렬할 원소가 없으므로 []를 그대로 리턴한다.

정렬된 xs

리스트 xs를 정렬 하려면		
반복 조건	xs != []	<ul style="list-style-type: none">xs에서 가장 작은 원소를 찾아서 smallest로 지정하고,xs에서 smallest를 제거하고,xs를 재귀로 정렬하고,smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	xs == []	<ul style="list-style-type: none">정렬할 원소가 없으므로 []를 그대로 리턴한다.

code : 5-11.py

```
1 def selection_sort(xs):
2     if xs != []:
3         smallest = min(xs)
4         xs.remove(smallest)
5         return [smallest] + selection_sort(xs)
6     else:
7         return []
```

리스트 메소드

연산	의미	비고
<code>xs.remove(x)</code>	xs에서 가장 앞에 나오는 원소 x를 제거한다.	x이 xs에 없으면 ValueError 가 발생한다.

```
1 def selection_sort(xs):  
2     if xs != []:  
3         smallest = min(xs)  
4         xs.remove(smallest)  
5         return [smallest] + selection_sort(xs)  
6     else:  
7         return []
```

```
selection_sort([3,5,4,2])
```

```
1 def selection_sort(xs):  
2     if xs != []:  
3         smallest = min(xs)  
4         xs.remove(smallest)  
5         return [smallest] + selection_sort(xs)  
6     else:  
7         return []
```

```
selection_sort([3,5,4,2])  
=> [2] + selection_sort([3,5,4])
```

```
1 def selection_sort(xs):  
2     if xs != []:  
3         smallest = min(xs)  
4         xs.remove(smallest)  
5         return [smallest] + selection_sort(xs)  
6     else:  
7         return []
```

```
    selection_sort([3,5,4,2])  
=> [2] + selection_sort([3,5,4])  
=> [2] + [3] + selection_sort([5,4])
```

```
1 def selection_sort(xs):  
2     if xs != []:  
3         smallest = min(xs)  
4         xs.remove(smallest)  
5         return [smallest] + selection_sort(xs)  
6     else:  
7         return []
```

```
    selection_sort([3,5,4,2])  
=> [2] + selection_sort([3,5,4])  
=> [2] + [3] + selection_sort([5,4])  
=> [2] + [3] + [4] + selection_sort([5])
```

```
1 def selection_sort(xs):  
2     if xs != []:  
3         smallest = min(xs)  
4         xs.remove(smallest)  
5         return [smallest] + selection_sort(xs)  
6     else:  
7         return []
```

```
selection_sort([3,5,4,2])  
=> [2] + selection_sort([3,5,4])  
=> [2] + [3] + selection_sort([5,4])  
=> [2] + [3] + [4] + selection_sort([5])  
=> [2] + [3] + [4] + [5] + selection_sort([])
```



```
1 def selection_sort(xs):  
2     if xs != []:  
3         smallest = min(xs)  
4         xs.remove(smallest)  
5         return [smallest] + selection_sort(xs)  
6     else:  
7         return []
```

```
    selection_sort([3,5,4,2])  
=> [2] + selection_sort([3,5,4])  
=> [2] + [3] + selection_sort([5,4])  
=> [2] + [3] + [4] + selection_sort([5])  
=> [2] + [3] + [4] + [5] + selection_sort([])  
=> [2] + [3] + [4] + [5] + []
```

```
1 def selection_sort(xs):  
2     if xs != []:  
3         smallest = min(xs)  
4         xs.remove(smallest)  
5         return [smallest] + selection_sort(xs)  
6     else:  
7         return []
```

```
    selection_sort([3,5,4,2])  
=> [2] + selection_sort([3,5,4])  
=> [2] + [3] + selection_sort([5,4])  
=> [2] + [3] + [4] + selection_sort([5])  
=> [2] + [3] + [4] + [5] + selection_sort([])  
=> [2] + [3] + [4] + [5] + []  
== [2,3,4,5]
```

재귀

code : 5-11.py

```
1 def selection_sort(xs):
2     if xs != []:
3         smallest = min(xs)
4         xs.remove(smallest)
5         return [smallest] + selection_sort(xs)
6     else:
7         return []
```

꼬리 재귀

code : 5-13.py

```
1 def selection_sort(xs):
2     def loop(xs, ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs, ss+[smallest])
7         else:
8             return ss
9     return loop(xs, [])
```

꼬리 재귀

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs,ss+[smallest])
7         else:
8             return ss
9     return loop(xs,[])
```

selection_sort([3,5,4,2])

=>

꼬리 재귀

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs,ss+[smallest])
7         else:
8             return ss
9     return loop(xs,[])
```

```
    selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=>
```

꼬리 재귀

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs,ss+[smallest])
7         else:
8             return ss
9     return loop(xs,[])
```

```
selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[2])
```

꼬리 재귀

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs,ss+[smallest])
7         else:
8             return ss
9     return loop(xs,[])
```

```
    selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[2]) == loop([3,5,4],[2])
=>
```

꼬리 재귀

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs,ss+[smallest])
7         else:
8             return ss
9     return loop(xs,[])
```

```
selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[2]) == loop([3,5,4],[2])
=> loop([5,4],[2]+[3]) ==
```


꼬리 재귀

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs,ss+[smallest])
7         else:
8             return ss
9     return loop(xs,[])
```

```
selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[2]) == loop([3,5,4],[2])
=> loop([5,4],[2]+[3]) == loop([5,4],[2,3])
=>
```

꼬리 재귀

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs,ss+[smallest])
7         else:
8             return ss
9     return loop(xs,[])
```

```
selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[2]) == loop([3,5,4],[2])
=> loop([5,4],[2,3]) == loop([5,4],[2,3])
=> loop([5],[2,3,4]) ==
```

꼬리 재귀

```

1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs,ss+[smallest])
7         else:
8             return ss
9     return loop(xs,[])

```

```

    selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[2]) == loop([3,5,4],[2])
=> loop([5,4],[2,3]) == loop([5,4],[2,3])
=> loop([5],[2,3,4]) == loop([5],[2,3,4])
=>

```

꼬리 재귀

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs,ss+[smallest])
7         else:
8             return ss
9     return loop(xs,[])
```

```
selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[2]) == loop([3,5,4],[2])
=> loop([5,4],[2,3]) == loop([5,4],[2,3])
=> loop([5],[2,3,4]) == loop([5],[2,3,4])
=> loop([], [2,3,4,5]) ==
```

꼬리 재귀

```

1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs,ss+[smallest])
7         else:
8             return ss
9     return loop(xs,[])

```

```

    selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[2]) == loop([3,5,4],[2])
=> loop([5,4],[2]+[3]) == loop([5,4],[2,3])
=> loop([5],[2,3]+[4]) == loop([5],[2,3,4])
=> loop([], [2,3,4]+[5]) == loop([], [2,3,4,5])
=>

```

꼬리 재귀

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs,ss+[smallest])
7         else:
8             return ss
9     return loop(xs,[])
```

```
selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[2]) == loop([3,5,4],[2])
=> loop([5,4],[2]+[3]) == loop([5,4],[2,3])
=> loop([5],[2,3]+[4]) == loop([5],[2,3,4])
=> loop([], [2,3,4]+[5]) == loop([], [2,3,4,5])
=> [2,3,4,5]
```

리스트 메소드

연산	의미	비고
<code>xs.append(x)</code>	<code>xs</code> 의 맨 뒤에 <code>x</code> 을 붙인다.	-

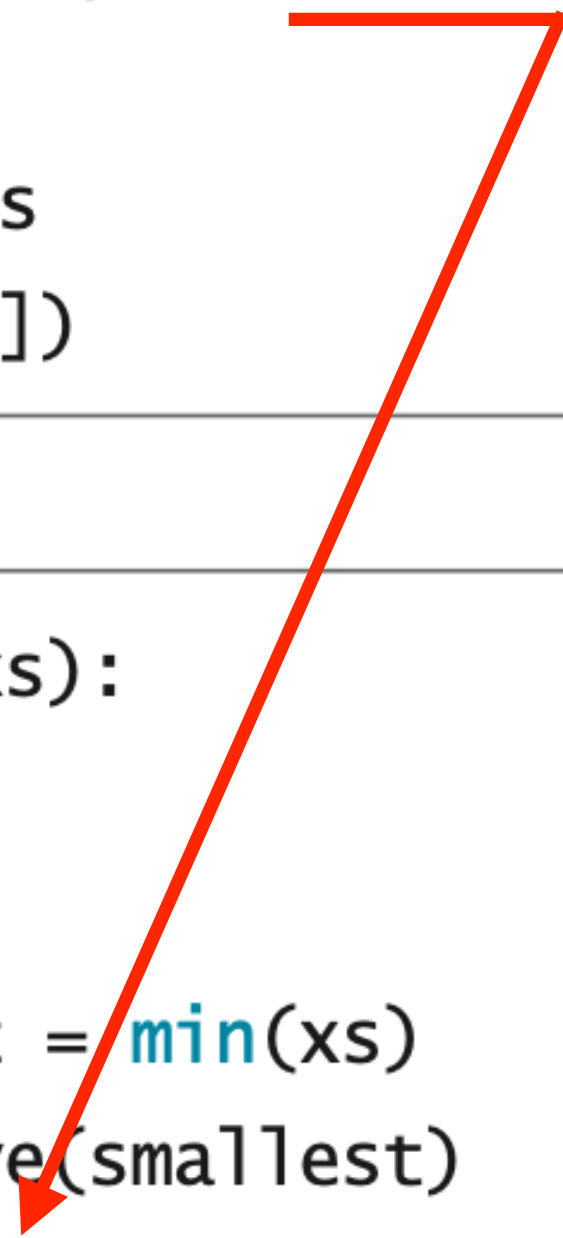
꼬리 재귀

code : 5-13.py

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             return loop(xs,ss+[smallest])
7         else:
8             return ss
9     return loop(xs,[])
```

code : 5-14.py


```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             ss.append(smallest)
7             return loop(xs,ss)
8         else:
9             return ss
10    return loop(xs,[])
```



꼬리 재귀

code : 5-14.py

```
1 def selection_sort(xs):
2     def loop(xs, ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             ss.append(smallest)
7             return loop(xs, ss)
8         else:
9             return ss
10    return loop(xs, [])
```



while 루프

code : 5-15.py

```
1 def selection_sort(xs):
2     ss = []
3     while xs != []:
4         smallest = min(xs)
5         xs.remove(smallest)
6         ss.append(smallest)
7     return ss
```

꼬리 재귀

code : 5-14.py

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             ss.append(smallest)
7             return loop(xs,ss)
8         else:
9             return ss
10    return loop(xs,[])
```

while 루프

code : 5-15.py

```
1 def selection_sort(xs):
2     ss = []
3     while xs != []:
4         smallest = min(xs)
5         xs.remove(smallest)
6         ss.append(smallest)
7     return ss
```

꼬리 재귀

code : 5-14.py

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             ss.append(smallest)
7             return loop(xs,ss)
8         else:
9             return ss
10    return loop(xs,[])
```

while 루프

code : 5-15.py

```
1 def selection_sort(xs):
2     ss = []
3     while xs != []:
4         smallest = min(xs)
5         xs.remove(smallest)
6         ss.append(smallest)
7     return ss
```

꼬리 재귀

code : 5-14.py

```
1 def selection_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             smallest = min(xs)
5             xs.remove(smallest)
6             ss.append(smallest)
7             return loop(xs,ss)
8         else:
9             return ss
10    return loop(xs,[])
```

while 루프

code : 5-15.py

```
1 def selection_sort(xs):
2     ss = []
3     while xs != []:
4         smallest = min(xs)
5         xs.remove(smallest)
6         ss.append(smallest)
7     return ss
```

삽입정렬

Insertion Sort

삽입정렬 알고리즘

Insertion Sort

리스트 xs를 <u>정렬</u> 하려면		
반복 조건	$xs \neq []$	<ul style="list-style-type: none">• xs의 후미리스트인 $xs[1:]$를 재귀로 <u>정렬</u>하고,• xs의 선두원소인 $xs[0]$를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	$xs == []$	<ul style="list-style-type: none">• 정렬할 원소가 없으므로 $[]$를 그대로 리턴한다.

리스트 xs를 <u>정렬</u> 하려면		
반복 조건	<code>xs != []</code>	<ul style="list-style-type: none"> xs의 후미리스트인 <code>xs[1:]</code>를 재귀로 <u>정렬</u>하고, xs의 선두원소인 <code>xs[0]</code>를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	<code>xs == []</code>	<ul style="list-style-type: none"> 정렬할 원소가 없으므로 <code>[]</code>를 그대로 리턴한다.

[]



정렬

[]

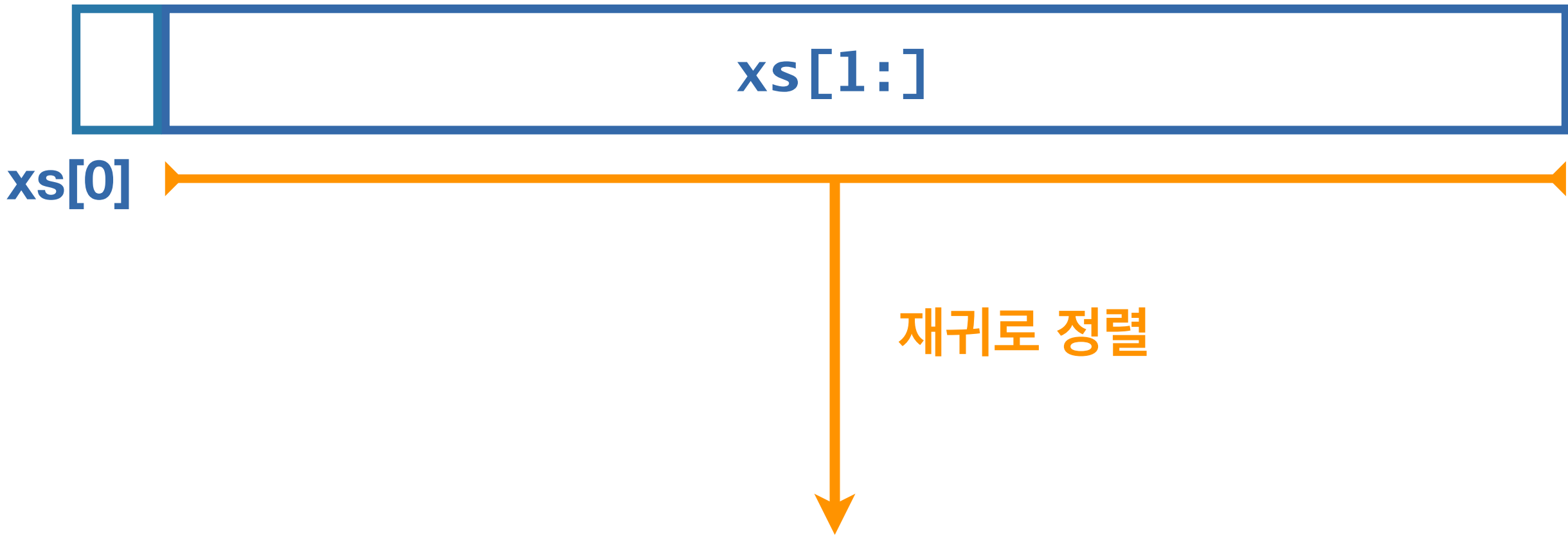
리스트 xs를 <u>정렬</u> 하려면		
반복 조건	$xs \neq []$	<ul style="list-style-type: none">xs의 후미리스트인 $xs[1:]$를 재귀로 <u>정렬</u>하고,xs의 선두원소인 $xs[0]$를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	$xs == []$	<ul style="list-style-type: none">정렬할 원소가 없으므로 $[]$를 그대로 리턴한다.

정렬 대상 리스트

xs

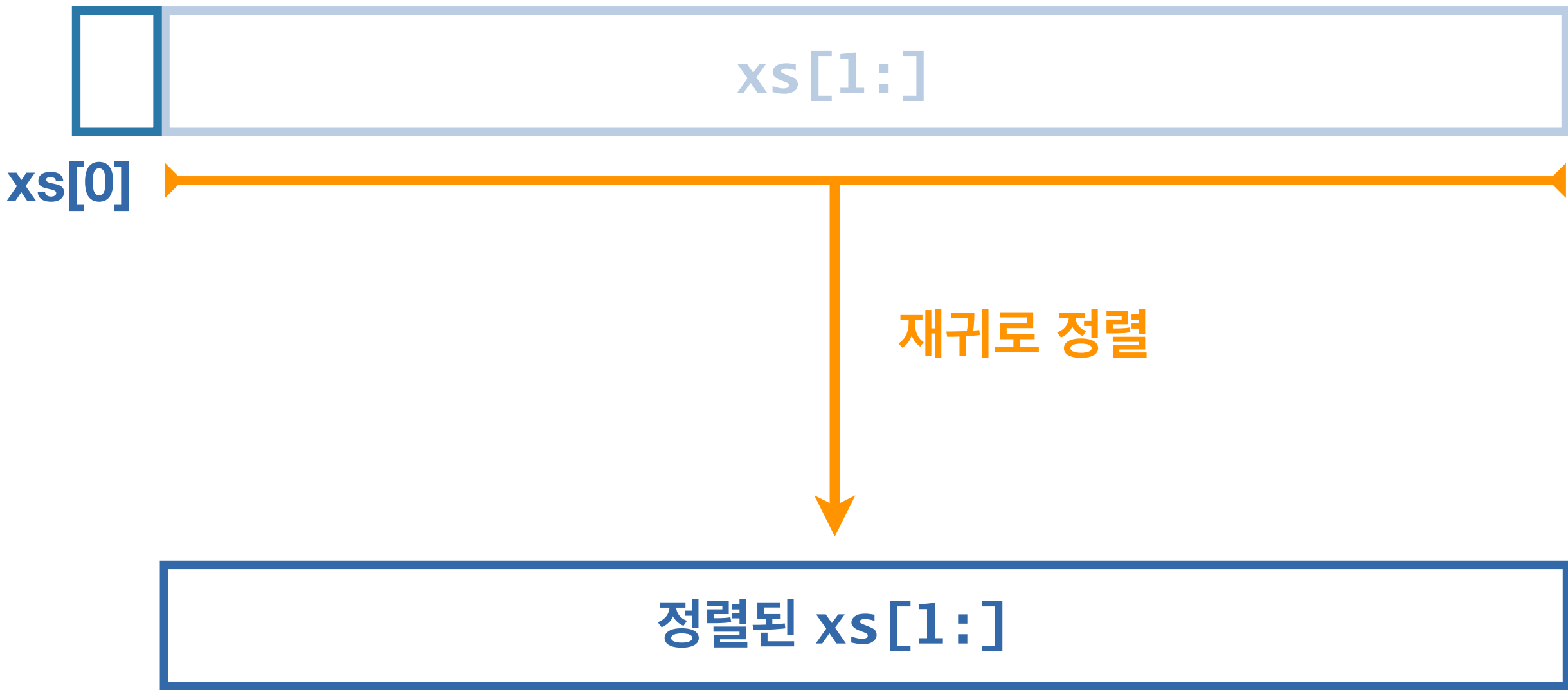
리스트 xs를 정렬하려면		
반복 조건	<code>xs != []</code>	<ul style="list-style-type: none">xs의 후미리스트인 <code>xs[1:]</code>를 재귀로 정렬하고,xs의 선두원소인 <code>xs[0]</code>를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	<code>xs == []</code>	<ul style="list-style-type: none">정렬할 원소가 없으므로 <code>[]</code>를 그대로 리턴한다.

정렬 대상 리스트

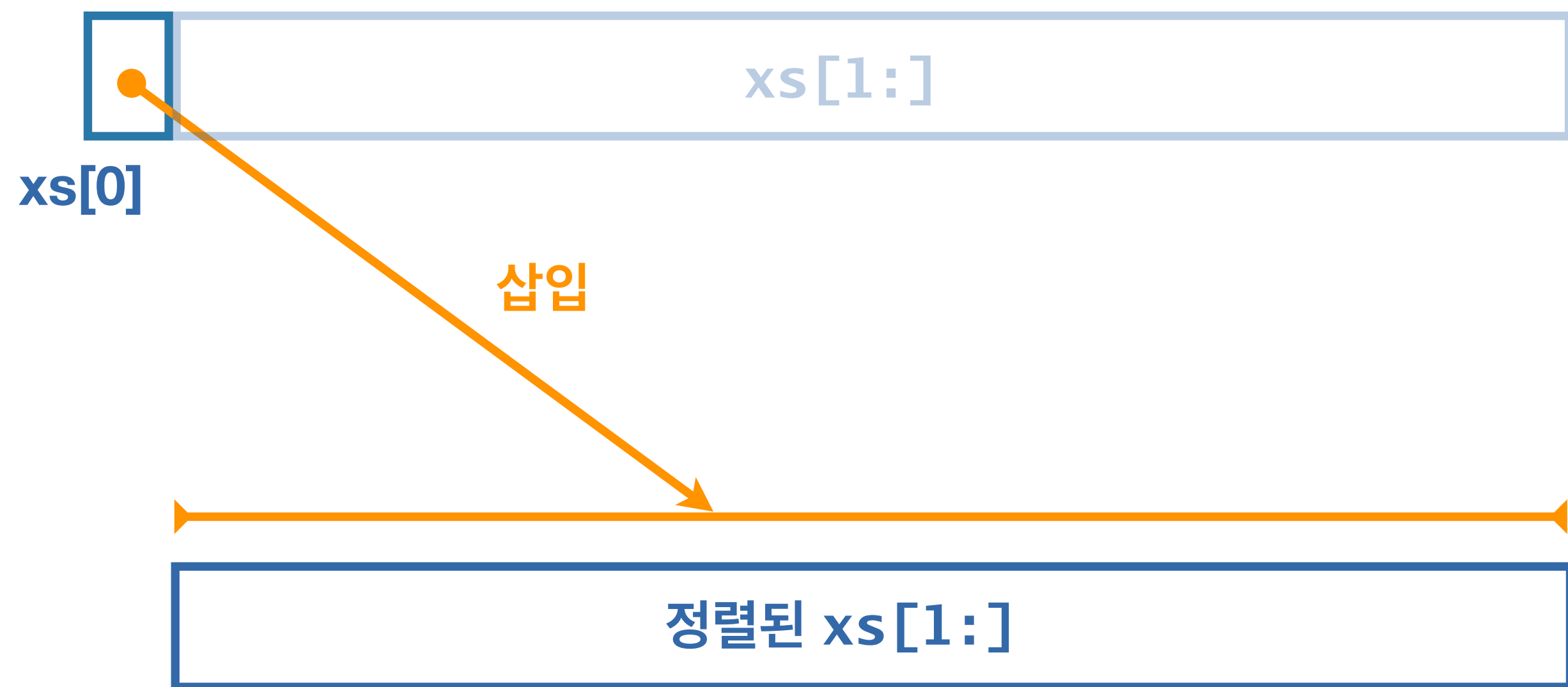


리스트 xs를 <u>정렬</u> 하려면		
반복 조건	<code>xs != []</code>	<ul style="list-style-type: none"> xs의 후미리스트인 <code>xs[1:]</code>를 재귀로 <u>정렬</u>하고, xs의 선두원소인 <code>xs[0]</code>를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	<code>xs == []</code>	<ul style="list-style-type: none"> 정렬할 원소가 없으므로 <code>[]</code>를 그대로 리턴한다.

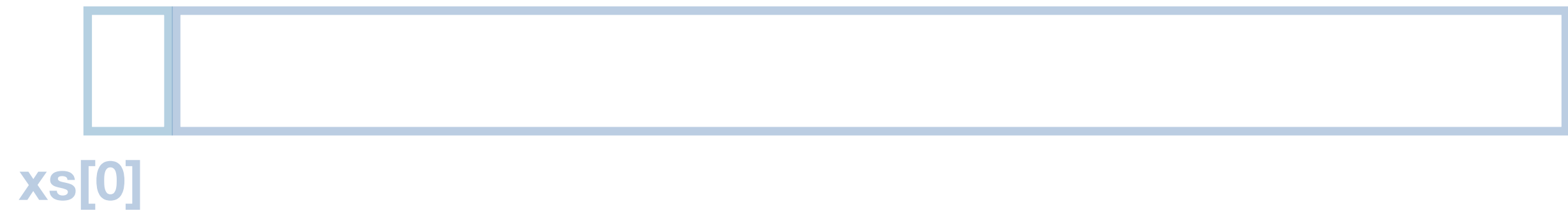
정렬 대상 리스트



리스트 xs를 정렬하려면		
반복 조건	<code>xs != []</code>	<ul style="list-style-type: none"> xs의 후미리스트인 <code>xs[1:]</code>를 재귀로 정렬하고, xs의 선두원소인 <code>xs[0]</code>를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	<code>xs == []</code>	<ul style="list-style-type: none"> 정렬할 원소가 없으므로 <code>[]</code>를 그대로 리턴한다.



리스트 xs를 <u>정렬</u> 하려면		
반복 조건	<code>xs != []</code>	<ul style="list-style-type: none">xs의 후미리스트인 <code>xs[1:]</code>를 재귀로 <u>정렬</u>하고,xs의 선두원소인 <code>xs[0]</code>를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	<code>xs == []</code>	<ul style="list-style-type: none">정렬할 원소가 없으므로 <code>[]</code>를 그대로 리턴한다.



리스트 xs를 정렬하려면		
반복 조건	<code>xs != []</code>	<ul style="list-style-type: none"> xs의 후미리스트인 <code>xs[1:]</code>를 재귀로 정렬하고, xs의 선두원소인 <code>xs[0]</code>를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	<code>xs == []</code>	<ul style="list-style-type: none"> 정렬할 원소가 없으므로 <code>[]</code>를 그대로 리턴한다.

code : 5-16.py

```

1 def insertion_sort(xs):
2     if xs != []:
3         return insert(xs[0], insertion_sort(xs[1:]))
4     else:
5         return []

```

```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])
```

```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))
```

```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))  
=> insert(3,insert(5,insertionsort([4,2])))
```



```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))  
=> insert(3,insert(5,insertionsort([4,2])))  
=> insert(3,insert(5,insert(4,insertionsort([2])))))
```

```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))  
=> insert(3,insert(5,insertionsort([4,2])))  
=> insert(3,insert(5,insert(4,insertionsort([2])))  
=> insert(3,insert(5,insert(4,insert(2,insertionsort([])))))
```

```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))  
=> insert(3,insert(5,insertionsort([4,2])))  
=> insert(3,insert(5,insert(4,insertionsort([2])))  
=> insert(3,insert(5,insert(4,insert(2,insertionsort([]))))  
=> insert(3,insert(5,insert(4,insert(2,[]))))
```

```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))  
=> insert(3,insert(5,insertionsort([4,2])))  
=> insert(3,insert(5,insert(4,insertionsort([2])))  
=> insert(3,insert(5,insert(4,insert(2,insertionsort([]))))  
=> insert(3,insert(5,insert(4,insert(2,[]))))
```

```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))  
=> insert(3,insert(5,insertionsort([4,2])))  
=> insert(3,insert(5,insert(4,insertionsort([2])))  
=> insert(3,insert(5,insert(4,insert(2,insertionsort([]))))  
=> insert(3,insert(5,insert(4,insert(2,[]))))  
=> insert(3,insert(5,insert(4,[2])))
```

```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))  
=> insert(3,insert(5,insertionsort([4,2])))  
=> insert(3,insert(5,insert(4,insertionsort([2])))  
=> insert(3,insert(5,insert(4,insert(2,insertionsort([]))))  
=> insert(3,insert(5,insert(4,insert(2,[]))))  
=> insert(3,insert(5,insert(4,[2])))
```

```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))  
=> insert(3,insert(5,insertionsort([4,2])))  
=> insert(3,insert(5,insert(4,insertionsort([2])))  
=> insert(3,insert(5,insert(4,insert(2,insertionsort([]))))  
=> insert(3,insert(5,insert(4,insert(2,[]))))  
=> insert(3,insert(5,insert(4,[2])))  
=> insert(3,insert(5,[2,4]))
```

```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))  
=> insert(3,insert(5,insertionsort([4,2])))  
=> insert(3,insert(5,insert(4,insertionsort([2])))  
=> insert(3,insert(5,insert(4,insert(2,insertionsort([]))))  
=> insert(3,insert(5,insert(4,insert(2,[]))))  
=> insert(3,insert(5,insert(4,[2])))  
=> insert(3,insert(5,[2,4]))
```



```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))  
=> insert(3,insert(5,insertionsort([4,2])))  
=> insert(3,insert(5,insert(4,insertionsort([2])))  
=> insert(3,insert(5,insert(4,insert(2,insertionsort([]))))  
=> insert(3,insert(5,insert(4,insert(2,[]))))  
=> insert(3,insert(5,insert(4,[2])))  
=> insert(3,insert(5,[2,4]))  
=> insert(3,[2,4,5])
```

```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))  
=> insert(3,insert(5,insertionsort([4,2])))  
=> insert(3,insert(5,insert(4,insertionsort([2])))  
=> insert(3,insert(5,insert(4,insert(2,insertionsort([]))))  
=> insert(3,insert(5,insert(4,insert(2,[]))))  
=> insert(3,insert(5,insert(4,[2])))  
=> insert(3,insert(5,[2,4]))  
=> insert(3,[2,4,5])
```

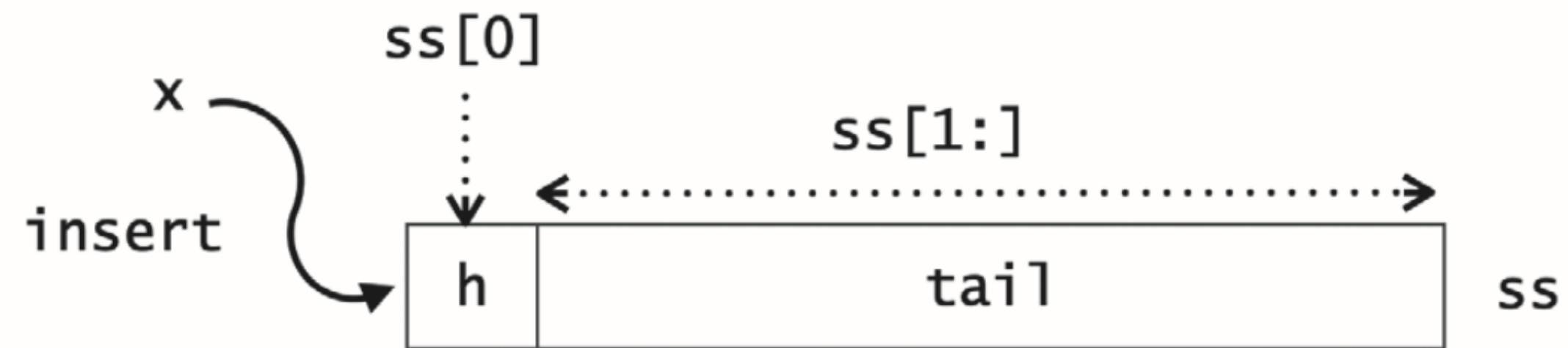
```
1 def insertion_sort(xs):  
2     if xs != []:  
3         return insert(xs[0],insertion_sort(xs[1:]))  
4     else:  
5         return []
```

```
insertionsort([3,5,4,2])  
=> insert(3,insertionsort([5,4,2]))  
=> insert(3,insert(5,insertionsort([4,2])))  
=> insert(3,insert(5,insert(4,insertionsort([2])))  
=> insert(3,insert(5,insert(4,insert(2,insertionsort([]))))  
=> insert(3,insert(5,insert(4,insert(2,[]))))  
=> insert(3,insert(5,insert(4,[2])))  
=> insert(3,insert(5,[2,4]))  
=> insert(3,[2,4,5])  
=> [2,3,4,5]
```

insert(x, ss)

원소 x 를 정렬된 리스트 ss 의 적절한 위치에 끼워넣기

$ss \neq []$



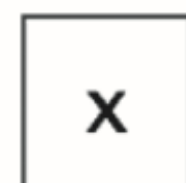
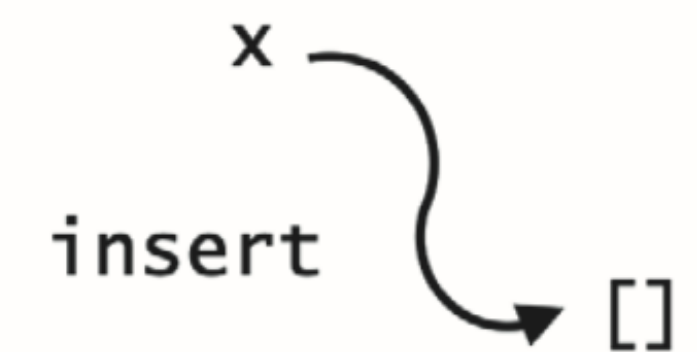
• $x \leq h$



• $x > h$



$ss == []$



`[x]`

insert 알고리즘

정수 x 을 정렬된 리스트 ss 의 제 위치에 끼워 넣으려면,

반복 조건

$ss \neq []$

- $x \leq ss[0]$ 이면, x 를 ss 의 앞에 붙여서 리턴한다.
- $x > ss[0]$ 이면, 재귀로 x 를 $ss[1:]$ 의 제 위치에 끼워넣고, 그 앞에 $ss[0]$ 를 붙여서 리턴한다.

종료 조건

$ss == []$

그냥 x 만 가지고 리스트를 만든다.

insert 실행추적 사례

```
insert(9, [])
```

insert 실행추적 사례

```
insert(9, [])  
=> [9]
```

insert 실행추적 사례

```
insert(1,[2,4,5,7,8])
```


insert 실행추적 사례

```
insert(1,[2,4,5,7,8])  
=> [1, 2, 4, 5, 7, 8]
```

insert 실행추적 사례

```
insert(6,[2,4,5,7,8])
```

insert 실행추적 사례

```
insert(6, [2, 4, 5, 7, 8])  
=> [2] + insert(6, [4, 5, 7, 8])
```

insert 실행추적 사례

```
insert(6,[2,4,5,7,8])
```

```
=> [2] + insert(6,[4,5,7,8])
```

```
=> [2] + [4] + insert(6,[5,7,8])
```

insert 실행추적 사례

```
insert(6,[2,4,5,7,8])  
=> [2] + insert(6,[4,5,7,8])  
=> [2] + [4] + insert(6,[5,7,8])  
=> [2] + [4] + [5] + insert(6,[7,8])
```

insert 실행추적 사례

```
insert(6,[2,4,5,7,8])  
=> [2] + insert(6,[4,5,7,8])  
=> [2] + [4] + insert(6,[5,7,8])  
=> [2] + [4] + [5] + insert(6,[7,8])  
=> [2] + [4] + [5] + [6] + [7,8]
```

insert 실행추적 사례

```
insert(6, [2, 4, 5, 7, 8])  
=> [2] + insert(6, [4, 5, 7, 8])  
=> [2] + [4] + insert(6, [5, 7, 8])  
=> [2] + [4] + [5] + insert(6, [7, 8])  
=> [2] + [4] + [5] + [6] + [7, 8]  
    . . .  
=> [2, 4, 5, 6, 7, 8]
```

프로그래밍의 정석

파이썬

도경구 지음



실습 5.4 insert : while 루프 버전

>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

프로그래밍의 정석 파이썬

도경구 지음



pp.236~238



실습 5.5 insertion_sort : 꼬리재귀 함수 버전



실습 5.6 insertion_sort : while 루프 버전



실습 5.7 insertion_sort : for 루프 버전

합병정렬

Merge Sort

합병정렬 알고리즘

Merge Sort

리스트 xs 를 합병정렬하려면

반복 조건

$\text{len}(xs) > 1$

- xs 를 반으로 나누어, 각각 재귀로 합병정렬 완료하고,
- 정렬된 두 리스트를 앞에서부터 차례로 훑어가며
- 두 선두원소 중에서 작은 원소를 먼저 하나씩 취하는 방식으로 하나로 합병_{merge}하여 리턴한다.

종료 조건

$\text{len}(xs) \leq 1$

- 정렬할 필요가 없으므로 그대로 리턴한다.

리스트 xs를 합병정렬하려면		
반복 조건	$\text{len}(xs) > 1$	<ul style="list-style-type: none">xs를 반으로 나누어, 각각 재귀로 합병정렬 완료하고,정렬된 두 리스트를 앞에서부터 차례로 훑어가며두 선두원소 중에서 작은 원소를 먼저 하나씩 취하는 방식으로 하나로 합병_{merge}하여 리턴한다.
종료 조건	$\text{len}(xs) \leq 1$	<ul style="list-style-type: none">정렬할 필요가 없으므로 그대로 리턴한다.



리스트 xs를 합병정렬하려면		
반복 조건	$\text{len}(xs) > 1$	<ul style="list-style-type: none">xs를 반으로 나누어, 각각 재귀로 합병정렬 완료하고,정렬된 두 리스트를 앞에서부터 차례로 훑어가며두 선두원소 중에서 작은 원소를 먼저 하나씩 취하는 방식으로 하나로 합병_{merge}하여 리턴한다.
종료 조건	$\text{len}(xs) \leq 1$	<ul style="list-style-type: none">정렬할 필요가 없으므로 그대로 리턴한다.

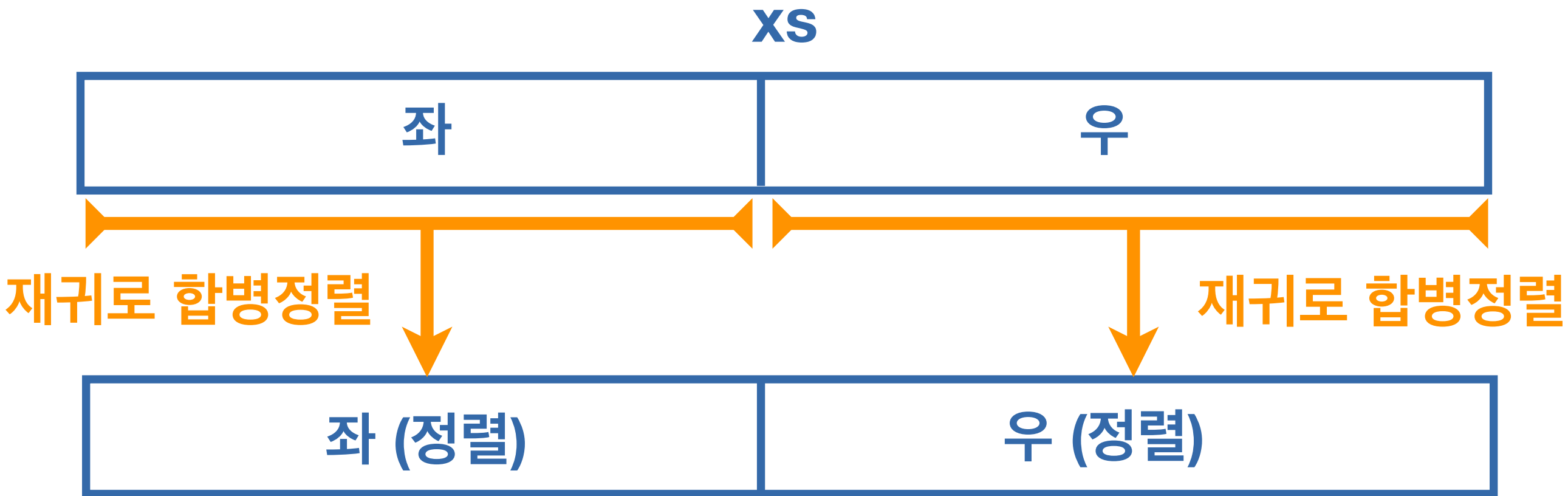
정렬 대상 리스트

xs

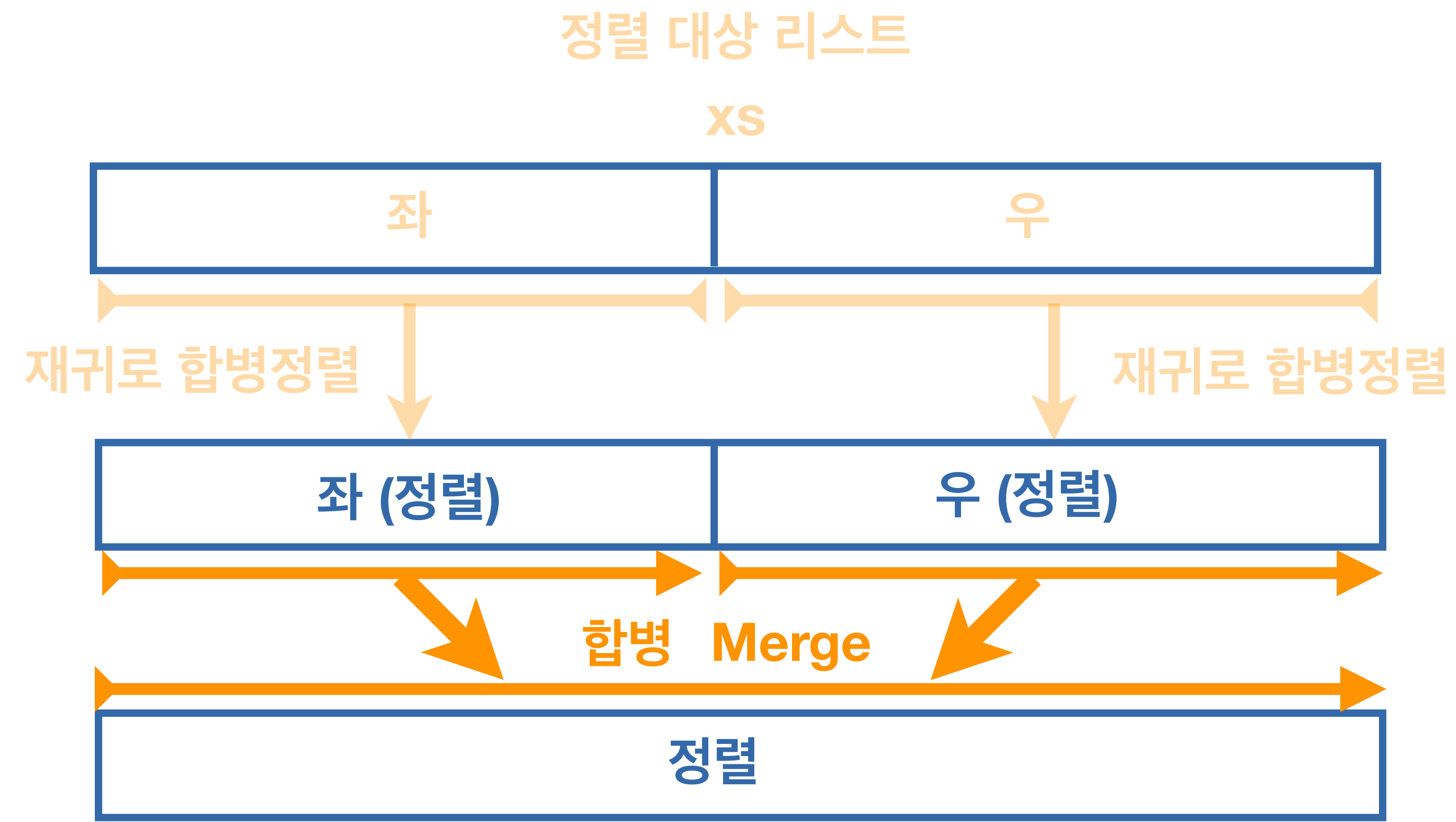
좌	우
---	---

리스트 xs를 합병정렬하려면		
반복 조건	$\text{len}(xs) > 1$	<ul style="list-style-type: none">xs를 반으로 나누어, 각각 재귀로 합병정렬 완료하고,정렬된 두 리스트를 앞에서부터 차례로 훑어가며두 선두원소 중에서 작은 원소를 먼저 하나씩 취하는 방식으로 하나로 합병_{merge}하여 리턴한다.
종료 조건	$\text{len}(xs) \leq 1$	<ul style="list-style-type: none">정렬할 필요가 없으므로 그대로 리턴한다.

정렬 대상 리스트



리스트 xs를 합병정렬하려면		
반복 조건	$\text{len}(xs) > 1$	<ul style="list-style-type: none">xs를 반으로 나누어, 각각 재귀로 합병정렬 완료하고,정렬된 두 리스트를 앞에서부터 차례로 훑어가며두 선두원소 중에서 작은 원소를 먼저 하나씩 취하는 방식으로 하나로 합병_{merge}하여 리턴한다.
종료 조건	$\text{len}(xs) \leq 1$	<ul style="list-style-type: none">정렬할 필요가 없으므로 그대로 리턴한다.

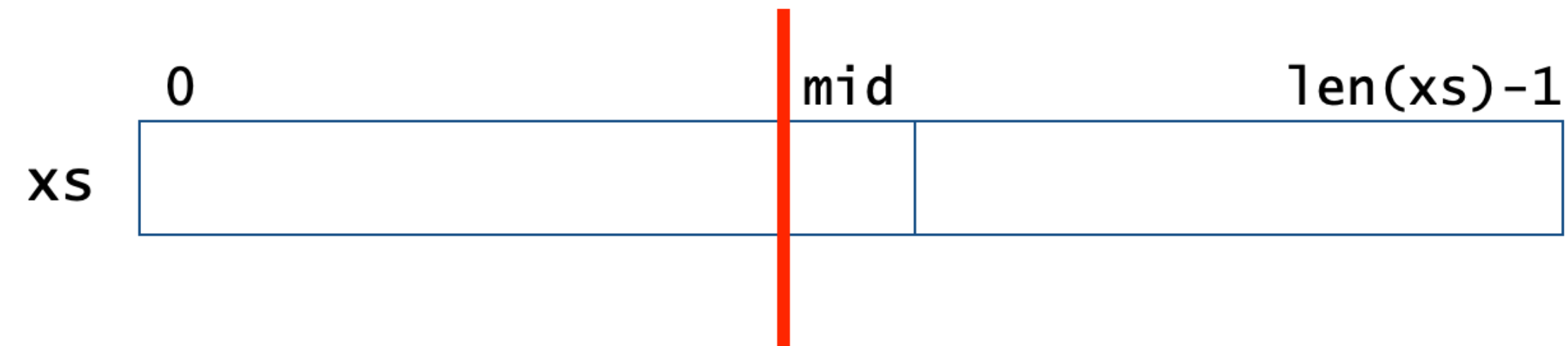


합병정렬

Merge Sort

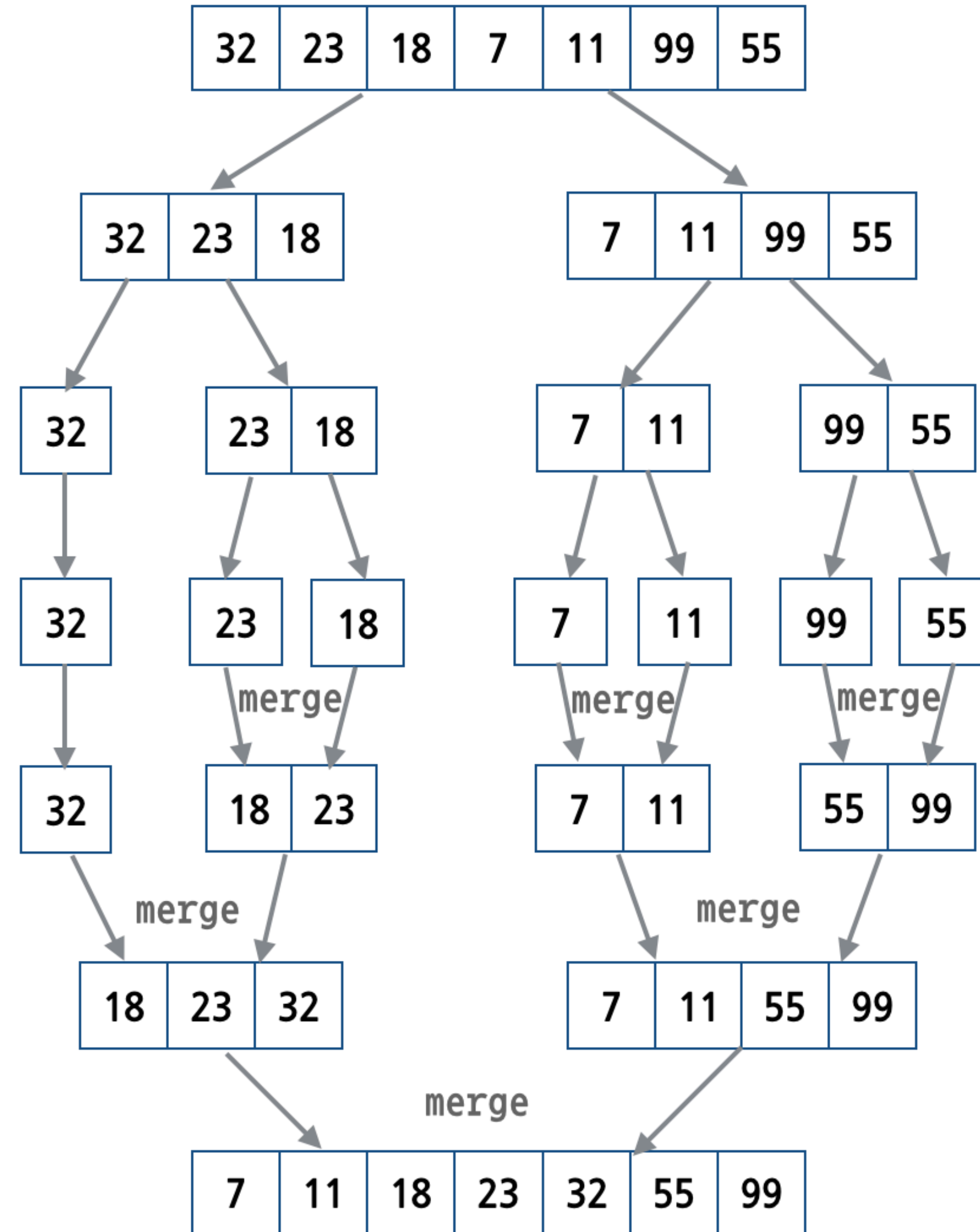
code : 5-24.py

```
1 def merge_sort(xs):  
2     if len(xs) > 1:  
3         mid = len(xs) // 2  
4         return merge(merge_sort(xs[:mid]), merge_sort(xs[mid:]))  
5     else:  
6         return xs
```



합병정렬

Merge Sort



합병 Merge



합병 Merge

정렬된 리스트 left와 right를 합병하는 재귀 함수 merge의 반복조건

```
left와 right 양쪽에 원소가 최소한 하나씩은 있음  
= left != [] and right != []  
= not (left == []) and not (right == [])  
= not (left == [] or right == [])
```

code : 5-25.py

```
1 def merge(left, right):  
2     if not (left == [] or right == []):  
3         if left[0] <= right[0]:  
4             return [left[0]] + merge(left[1:], right)  
5         else:  
6             return [right[0]] + merge(left, right[1:])  
7     else:  
8         return left + right
```

>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

프로그래밍의 정석 파이썬

도경구 지음



pp.242~243



실습 5.8

merge : 꼬리재귀 함수 버전



실습 5.9

merge : while 루프 버전

퀵정렬

Quicksort

퀵정렬 알고리즘

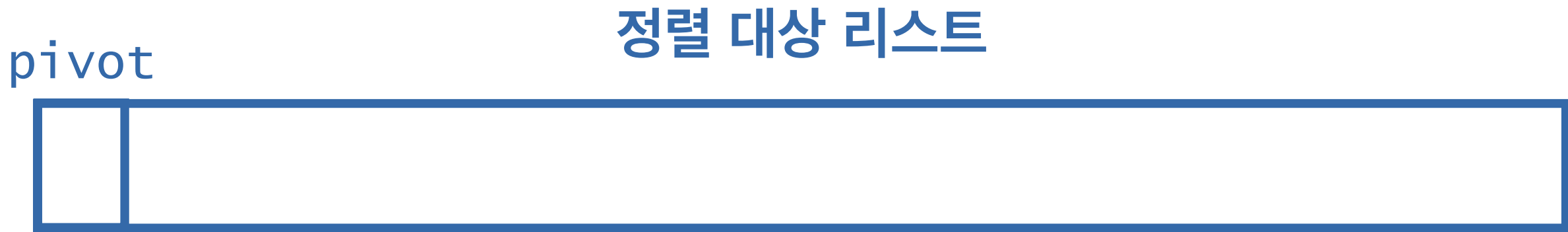
Quicksort

리스트 xs를 퀵정렬하려면		
반복 조건	$\text{len}(xs) > 1$	<ul style="list-style-type: none">● 기준으로 사용할 피벗 원소 <code>pivot</code>을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다.● <code>pivot</code>을 기준으로 작은 원소는 왼쪽 리스트 <code>left</code>로, 큰 원소는 오른쪽 리스트 <code>right</code>로 옮긴다.● 왼쪽 리스트 <code>left</code>와 오른쪽 리스트 <code>right</code>를 각각 재귀로 정렬하고, <code>left</code>와 <code>pivot</code>과 <code>right</code>를 나란히 붙여서 리턴한다.
종료 조건	$\text{len}(xs) \leq 1$	<ul style="list-style-type: none">● 정렬할 필요가 없으므로 그대로 리턴한다.

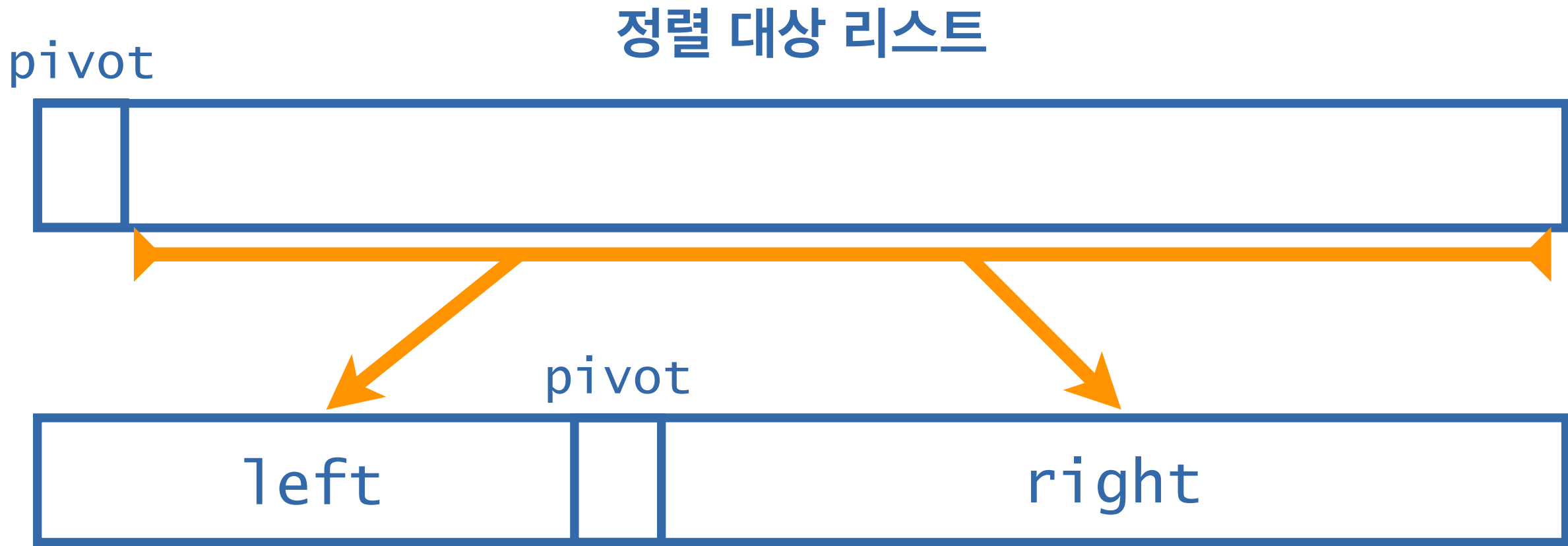
리스트 xs를 퀵정렬하려면		
반복 조건	$\text{len}(xs) > 1$	<ul style="list-style-type: none">● 기준으로 사용할 피벗 원소 <code>pivot</code>을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다.● <code>pivot</code>을 기준으로 작은 원소는 왼쪽 리스트 <code>left</code>로, 큰 원소는 오른쪽 리스트 <code>right</code>로 옮긴다.● 왼쪽 리스트 <code>left</code>와 오른쪽 리스트 <code>right</code>를 각각 재귀로 정렬하고, <code>left</code>와 <code>pivot</code>과 <code>right</code>를 나란히 붙여서 리턴한다.
종료 조건	$\text{len}(xs) \leq 1$	<ul style="list-style-type: none">● 정렬할 필요가 없으므로 그대로 리턴한다.



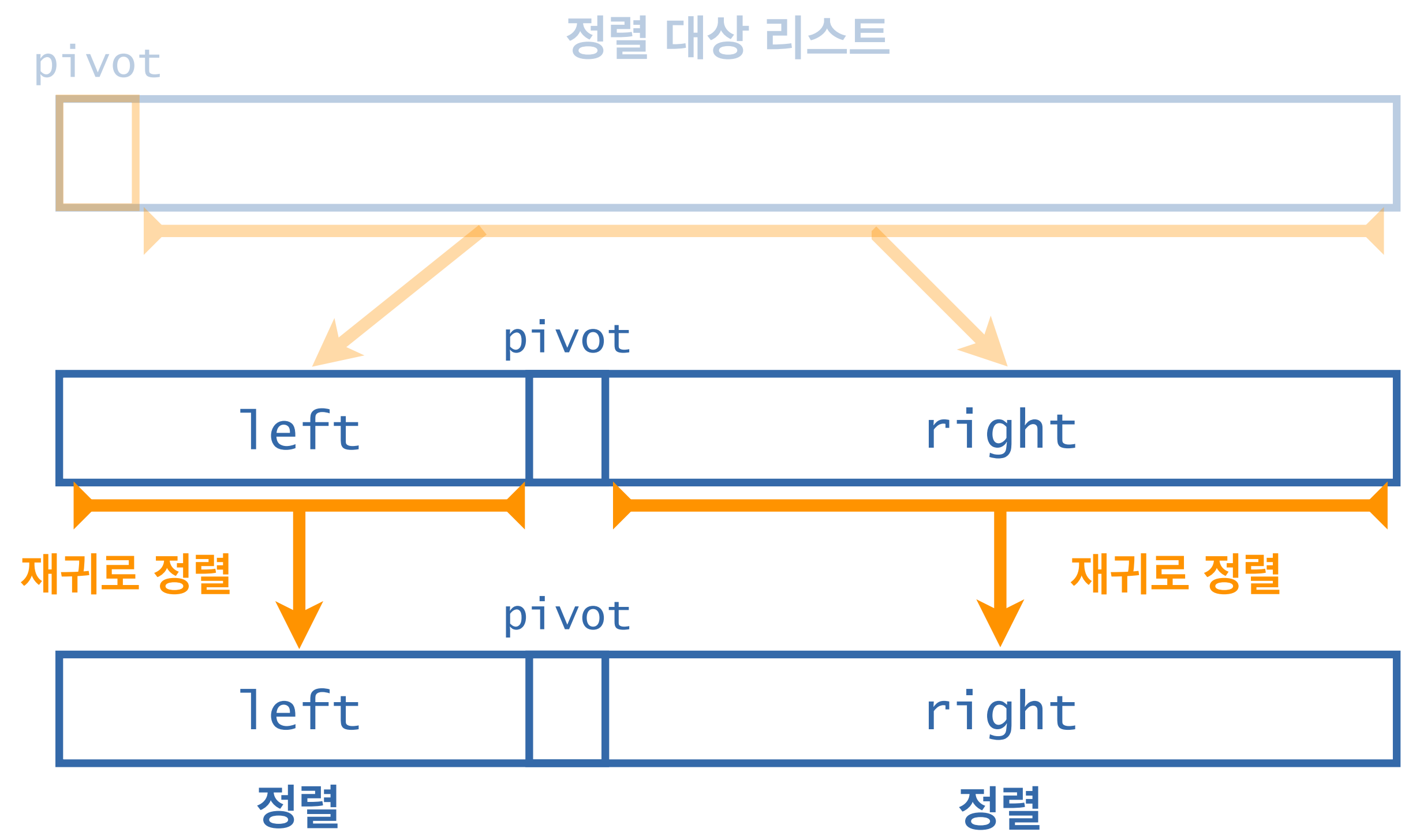
리스트 xs를 퀵정렬하려면		
반복 조건	$\text{len}(xs) > 1$	<ul style="list-style-type: none">● 기준으로 사용할 피벗 원소 <code>pivot</code>을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다.● <code>pivot</code>을 기준으로 작은 원소는 왼쪽 리스트 <code>left</code>로, 큰 원소는 오른쪽 리스트 <code>right</code>로 옮긴다.● 왼쪽 리스트 <code>left</code>와 오른쪽 리스트 <code>right</code>를 각각 재귀로 정렬하고, <code>left</code>와 <code>pivot</code>과 <code>right</code>를 나란히 붙여서 리턴한다.
종료 조건	$\text{len}(xs) \leq 1$	<ul style="list-style-type: none">● 정렬할 필요가 없으므로 그대로 리턴한다.



리스트 xs를 퀵정렬하려면		
반복 조건	$\text{len}(xs) > 1$	<ul style="list-style-type: none">● 기준으로 사용할 피벗 원소 <code>pivot</code>을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다.● <code>pivot</code>을 기준으로 작은 원소는 왼쪽 리스트 <code>left</code>로, 큰 원소는 오른쪽 리스트 <code>right</code>로 옮긴다.● 왼쪽 리스트 <code>left</code>와 오른쪽 리스트 <code>right</code>를 각각 재귀로 정렬하고, <code>left</code>와 <code>pivot</code>과 <code>right</code>를 나란히 붙여서 리턴한다.
종료 조건	$\text{len}(xs) \leq 1$	<ul style="list-style-type: none">● 정렬할 필요가 없으므로 그대로 리턴한다.

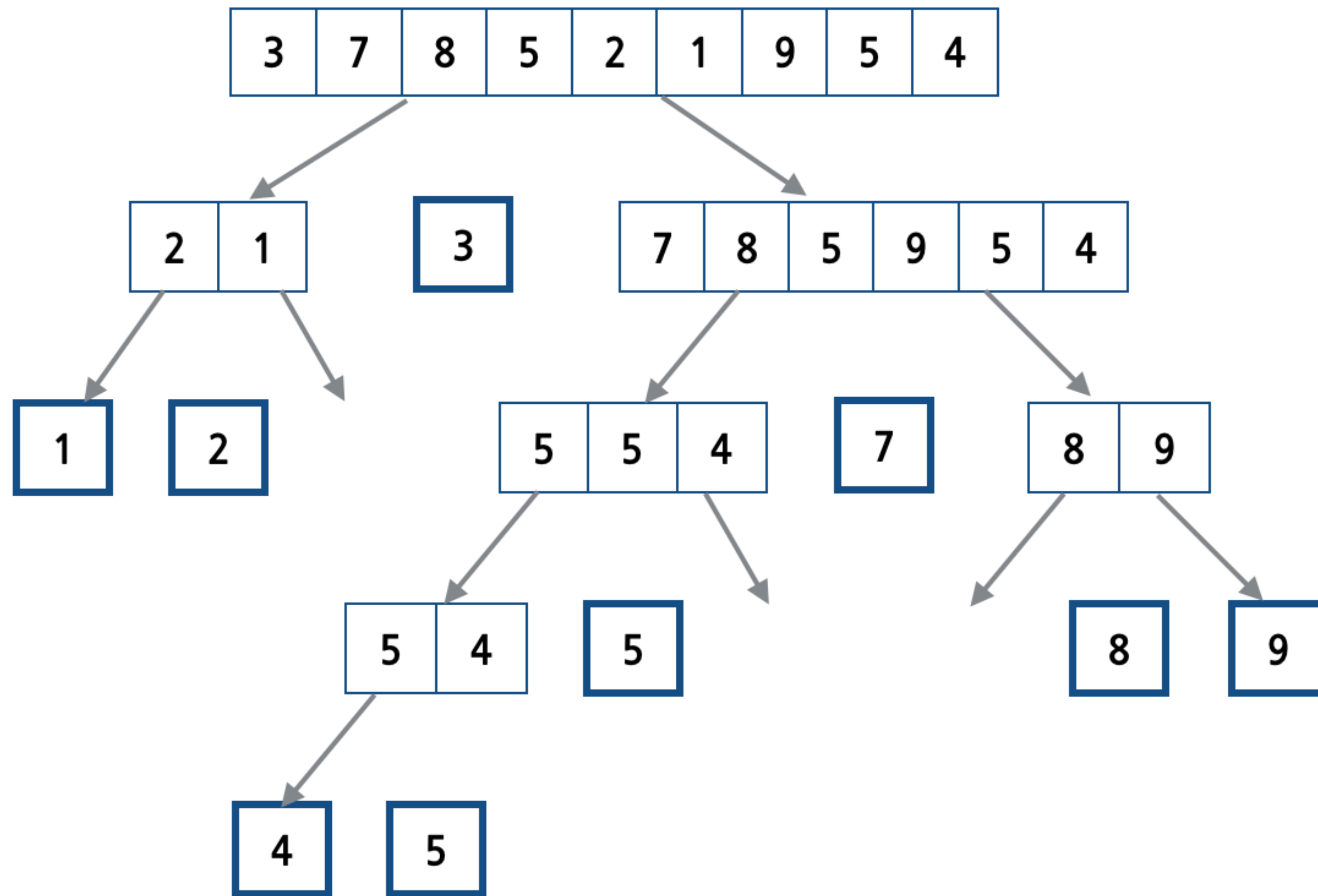


리스트 xs를 퀵정렬하려면		
반복 조건	$\text{len}(xs) > 1$	<ul style="list-style-type: none">● 기준으로 사용할 피벗 원소 <code>pivot</code>을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다.● <code>pivot</code>을 기준으로 작은 원소는 왼쪽 리스트 <code>left</code>로, 큰 원소는 오른쪽 리스트 <code>right</code>로 옮긴다.● 왼쪽 리스트 <code>left</code>와 오른쪽 리스트 <code>right</code>를 각각 재귀로 정렬하고, <code>left</code>와 <code>pivot</code>과 <code>right</code>를 나란히 붙여서 리턴한다.
종료 조건	$\text{len}(xs) \leq 1$	<ul style="list-style-type: none">● 정렬할 필요가 없으므로 그대로 리턴한다.



퀵정렬

Quicksort



리스트 xs를 퀵정렬하려면		
반복 조건	<code>len(xs) > 1</code>	<ul style="list-style-type: none"> ● 기준으로 사용할 피벗 원소 <code>pivot</code>을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다. ● <code>pivot</code>을 기준으로 작은 원소는 왼쪽 리스트 <code>left</code>로, 큰 원소는 오른쪽 리스트 <code>right</code>로 옮긴다. ● 왼쪽 리스트 <code>left</code>와 오른쪽 리스트 <code>right</code>를 각각 재귀로 정렬하고, <code>left</code>와 <code>pivot</code>과 <code>right</code>를 나란히 붙여서 리턴한다.
종료 조건	<code>len(xs) <= 1</code>	<ul style="list-style-type: none"> ● 정렬할 필요가 없으므로 그대로 리턴한다.

code : 5-28.py

```

1 def quicksort(xs):
2     if len(xs) > 1:
3         pivot = xs[0]
4         (left, right) = partition(pivot,xs[1:])
5         return quicksort(left) + [pivot] + quicksort(right)
6     else:
7         return xs

```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10        return [], []
```

partition(5,[7,2,1,9,4])

=>

```
1 def partition(pivot,xs):  
2     if xs != []:  
3         left, right = partition(pivot,xs[1:])  
4         if xs[0] <= pivot:  
5             left.append(xs[0])  
6         else:  
7             right.append(xs[0])  
8         return left, right  
9     else:  
10        return [], []
```

```
partition(5,[7,2,1,9,4])  
=> left, right = partition(5,[2,1,9,4])  
=>
```

```
1 def partition(pivot,xs):  
2     if xs != []:  
3         left, right = partition(pivot,xs[1:])  
4         if xs[0] <= pivot:  
5             left.append(xs[0])  
6         else:  
7             right.append(xs[0])  
8         return left, right  
9     else:  
10        return [], []
```

```
partition(5,[7,2,1,9,4])  
=> left, right = partition(5,[2,1,9,4])  
=>
```

```
1 def partition(pivot,xs):  
2     if xs != []:  
3         left, right = partition(pivot,xs[1:])  
4         if xs[0] <= pivot:  
5             left.append(xs[0])  
6         else:  
7             right.append(xs[0])  
8         return left, right  
9     else:  
10        return [], []
```

```
partition(5,[7,2,1,9,4])  
=> left, right = partition(5,[2,1,9,4])  
    => left, right = partition(5,[1,9,4])  
        =>
```



```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10        return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        =>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10        return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            =>
```

```
1 def partition(pivot,xs):  
2     if xs != []:  
3         left, right = partition(pivot,xs[1:])  
4         if xs[0] <= pivot:  
5             left.append(xs[0])  
6         else:  
7             right.append(xs[0])  
8         return left, right  
9     else:  
10        return [], []
```

```
partition(5,[7,2,1,9,4])  
=> left, right = partition(5,[2,1,9,4])  
    => left, right = partition(5,[1,9,4])  
        => left, right = partition(5,[9,4])  
            =>
```

```
1 def partition(pivot,xs):  
2     if xs != []:  
3         left, right = partition(pivot,xs[1:])  
4         if xs[0] <= pivot:  
5             left.append(xs[0])  
6         else:  
7             right.append(xs[0])  
8         return left, right  
9     else:  
10        return [], []
```

```
partition(5,[7,2,1,9,4])  
=> left, right = partition(5,[2,1,9,4])  
    => left, right = partition(5,[1,9,4])  
        => left, right = partition(5,[9,4])  
            => left, right = partition(5,[4])  
                =>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10        return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                =>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10    ✓ return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    =>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10    ✓ return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    =>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10        return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    => [], []
=>
```



```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10        return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    => [], []
=>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8     return left, right
9 else:
10    return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    => [], []
                => [4], []
            =>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8     return left, right
9 else:
10    return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    => [], []
                => [4], []
            =>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10        return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    => [], []
                => [4], []
            => [4], [9]
        =>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10        return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    => [], []
                => [4], []
            => [4], [9]
        =>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10        return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    => [], []
                => [4], []
            => [4], [9]
        => [4,1], [9]
    =>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10        return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    => [], []
                => [4], []
            => [4], [9]
        => [4,1], [9]
    =>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8     return left, right
9 else:
10    return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    => [], []
                => [4], []
            => [4], [9]
        => [4,1], [9]
    => [4,1,2], [9]
=>
```



```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8     return left, right
9 else:
10    return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    => [], []
                => [4], []
            => [4], [9]
        => [4,1], [9]
    => [4,1,2], [9]
=>
```

```
1 def partition(pivot,xs):
2     if xs != []:
3         left, right = partition(pivot,xs[1:])
4         if xs[0] <= pivot:
5             left.append(xs[0])
6         else:
7             right.append(xs[0])
8         return left, right
9     else:
10        return [], []
```

```
partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
        => left, right = partition(5,[9,4])
            => left, right = partition(5,[4])
                => left, right = partition(5,[])
                    => [], []
                => [4], []
            => [4], [9]
        => [4,1], [9]
    => [4,1,2], [9]
=> [4,1,2], [9,7]
```

>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

프로그래밍의 정석 파이썬

도경구 지음



pp.247~248



실습 5.10 partition: 꼬리재귀 함수 버전



실습 5.11 partition: while 루프 버전



실습 5.12 partition: for 루프 버전

>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>

프로그래밍의 정석

파이썬

도경구 지음



CHAPTER 5

재귀와 반복 : 정렬