

Python 소개

목차

파이썬

1. 파이썬의 특징
 2. 코테 용어로서의 장단점
- 자료 구조
 1. stack
 2. queue
 3. deque
 4. priority queue

파이썬 내장 기능

- 내장 함수
- 라이브러리
 1. itertools
 2. bisect
 3. math

파이썬의 특징

- 급부상중인 언어!

Year		Quarter	
2021		1	
# Ranking	Programming Language	Percentage (Change)	Trend
1	JavaScript	18.756% (+0.053%)	
2	Python	16.628% (+0.390%)	
3	Java	11.680% (+0.742%)	
4	Go	7.829% (-1.176%)	

https://madnight.github.io/github/#/pull_requests/2021/1

- 간결한, 직관적인 언어!

```
a, b = input().split()
print(int(a) + int(b))
```

```
#include <stdio.h>

int main(){
    int a, b;

    scanf("%d %d", &a, &b);
    printf("%d", a+b);

    return 0;
}
```

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws Exception {
        Scanner scan = new Scanner(System.in);

        String n1 = scan.next();
        String n2 = scan.next();

        int a = Integer.parseInt(n1);
        int b = Integer.parseInt(n2);

        System.out.println(a+b);
    }
}
```

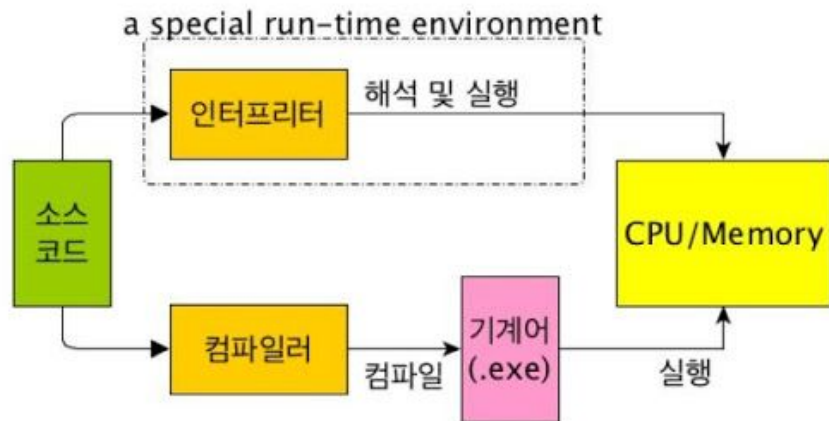
파이썬의 특징

1. 인터프리터 언어 (스크립트 언어)
2. 동적 타입(자료형) 언어
3. 절차/객체지향/함수형/관접형 프로그래밍 다 가능
4. 변수와 함수 모두가 거의 다 객체
5. 변수 할당 방식 - mutable과 immutable

```
>>> a1 = [[0]*2] * 2
>>> a1
[[0, 0], [0, 0]]
>>> a1[0][0] = 1
>>> a1
[[1, 0], [1, 0]]
>>> a1[1][1] = 3
>>> a1
[[1, 3], [1, 3]]
```

???

```
>>> x=28
>>> type(x)
<class 'int'>
>>> x=28.0
>>> type(x)
<class 'float'>
>>> x='SOPTAC'
>>> type(x)
<class 'str'>
```



파이썬의 특징

1. 인터프리터 언어 (스크립트 언어)
2. 동적 타입(자료형) 언어
3. 절차/객체지향/함수형/관접형 프로그래밍 다 가능
4. 변수와 함수 모두가 거의 다 객체
5. 변수 할당 방식 - mutable과 immutable

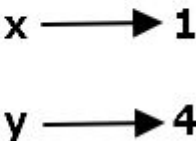
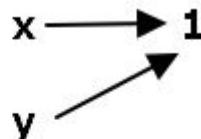
```
>>> a1 = [[0]*2] * 2
>>> a1
[[0, 0], [0, 0]]
>>> a1[0][0] = 1
>>> a1
[[1, 0], [1, 0]]
>>> a1[1][1] = 3
>>> a1
[[1, 3], [1, 3]]
>>> a2 = [[0]*2 for i in range(2)]
>>> a2
[[0, 0], [0, 0]]
>>> a2[0][0] = 1
>>> a2
[[1, 0], [0, 0]]
>>>
```

???

!!!

immutable

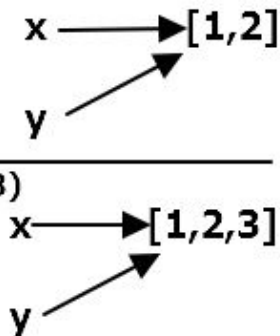
```
>>> x = 1
>>> y = x
>>> y += 3
>>> x
1
>>> y
4
```



문자열,
숫자,
튜플

mutable

```
>>> x = [1,2]
>>> y = x
>>> y.append(3)
>>> x
[1,2,3]
>>> y
[1,2,3]
```



딕셔너리,
리스트

파이썬의 장점

1. 배우기 쉬움
2. 기본 라이브러리로 제공되는 라이브러리 많음
(라이브러리 못쓰게 하는 코테도 많음)
3. 작성할 byte수가 다른 주류언어(C++,Java)에 비해 적음
4. 책, 블로그 등 다량의 레퍼런스 자료

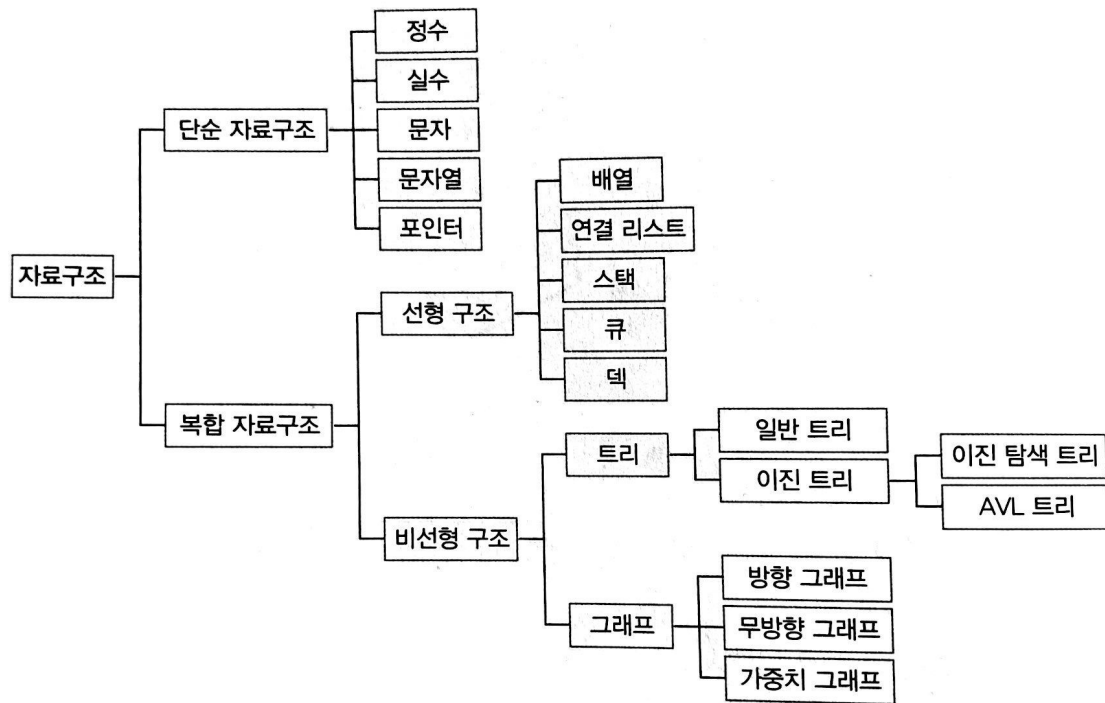
파이썬의 단점

1. 주류 언어중에 가장 느림
(기업 코테에서는 무시 가능한 특징)
2. 지원하지 않는 기업도 존재
(ex. 삼성 SW 역량테스트 B형, NHN enter.)

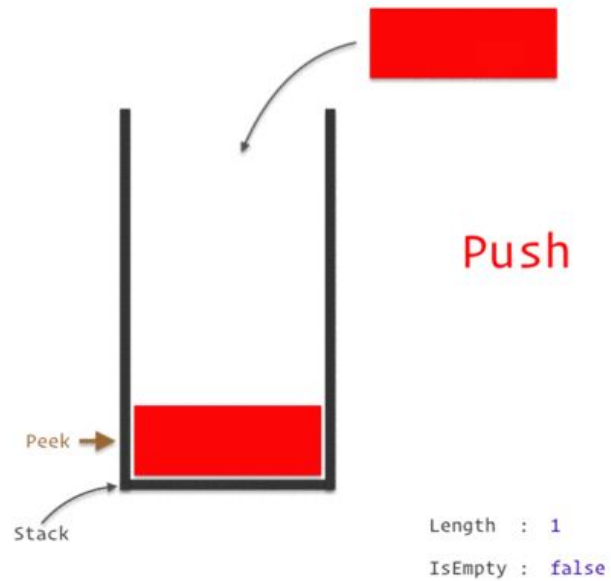
메모리	실행 시간	사용 언어	코드 길이
29284 KB	56 ms	Python 3 / 수정	46 B
14188 KB	104 ms	Java / 수정	314 B
1116 KB	0 ms	C / 수정	110 B

파이썬으로 보는 자료구조

자료구조



자료구조 1. 스택



자료구조 1. 스택

스택은 **LIFO**(Last In First Out) 자료구조 🖐️ 가장 마지막에 들어간 원소가 가장 먼저 나온다

리스트를 사용해 구현은 하지만, 보통 알 수 있는 값은 맨 위에 있는 값뿐이고

스택은 맨 위에 새 값을 쌓아올리거나, 맨 위에 있는 값을 빼는 삽입/삭제 연산을 합니다.

삽입을 **push** 연산, 삭제를 **pop** 연산이라 하며 맨 위의 값을 보는 연산을 보통 **top** 연산이라 칭합니다.

[출처] 스택(Stack) (수정 2019-05-14) | 작성자 라이

자료구조 1. 스택

파이썬에는 따로 라이브러리가
있진 않고 **리스트**를 스택처럼
사용하면 됩니다.

왜냐면 원래 스택은 리스트
(연결리스트)로 구현하니까요
^~^

옆의 연산이 모두 $O(1)$ 입니다.

init

```
st = []  
st = list()
```

push

```
st.append(1) # li = [1]  
st.append(2) # li = [1, 2]
```

pop

```
# st = [1, 2]  
top = st.pop()  
print(top) # 2
```

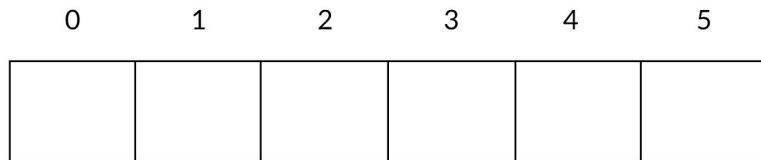
peek (top)

```
st = [1, 2, 3, 4]  
top = st[-1]  
print(top) # 4
```

자료구조 2. 큐

Queue Operations

Front = Rear = -1



Empty Queue



스택과 반대로 **FIFO**(First In First Out) 자료구조인데, **가장 먼저 들어갔던 값이 가장 먼저 나온다**는 겁니다.

역시 스택처럼 가장 끝부분에서만 삽입과 삭제 연산이 이루어지며 중간의 값은 알 필요가 없으나,

스택은 삽입과 삭제가 같은 쪽에서 이루어지고 **큐는 서로 다른 쪽에서 이루어진다**는 것이 다릅니다.

[출처] 큐(Queue), 덱(Deque) (수정: 2016-12-26)|작성자 라이

파이썬에서 큐를 어떻게 쓰는지 설명하기 전에,
덱부터 봐야해요 🐼

자료구조 3. 덱

앞 뒤로 모두 출입이 가능한~


우리가 문제 풀 때는 큐를 따로 구현하지 않고(해도 되긴해요 하고싶음)

파이썬에서 기본으로 제공해주는 collections 모듈의 deque 클래스를 쓸거예요.

Deque: Double Ended Queue

파이썬에서는 collections라는 모듈에서 deque라는 클래스를 제공한다. deque는 양방향 연결리스트로 구현이 돼있어 리스트보다 출입 연산이 더 효율적이다.

As **deque** provides an $O(1)$ time complexity for append and pop operations as **compared to list** which provides $O(n)$ time complexity.

그냥 리스트를 써서 양쪽에서 push, pop하는 것보다 deque를 사용하는 게
바람직하겠종 ~ 

자료구조 3. 덱

다른 메서드도 많은데
`appendleft(item)`, `popleft()` 이
있다는 걸 꼭 아셔야해용 ㅎㅎ

리스트에는 없는 메서드죠.

`append(x)`

Add x to the right side of the deque.

`appendleft(x)`

Add x to the left side of the deque.

...

`pop()`

Remove and return an element from the right side of the deque. If no elements are present, raises an `IndexError`.

`popleft()`

Remove and return an element from the left side of the deque. If no elements are present, raises an `IndexError`.

자료구조 3. 덱

의 메서드를 써서

큐 쓰기!

init

```
from collection import deque  
q = deque()
```

push

```
q.append(1)  
q.append(2)  
print(q) # [1, 2]
```

pop

큐는 먼저 들어간 것 (왼쪽에 있는 것)이 먼저 나와야하기 때문에, pop()이 아니라 popleft()를 사용해야한다!

```
popped = q.popleft()  
print(popped) # 1
```

peek

큐의 head를 확인하려면 그냥 인덱스로 0번째 값에 접근하면 된다.

```
q = [2, 3, 4, 5]  
head = q[0]  
print(head) # 2
```


자료구조 4. 우선순위 큐

WHAT: 우선순위 큐가 뭔데?’

우선순위 큐(Priority Queue)

우선순위 큐 그림 무엇일까요??
앞에서 큐에 대해서 잠깐 소개했었죠.

큐는 가장 먼저 들어간 데이터가 가장 먼저 나오는 구조를 띄고 있었습니다.

그와 다르게 우선순위 큐는 우선순위가 가장 높은 데이터를 가장 먼저 삭제하는 자료구조입니다!

말 그대로 데이터를 우선순위에 따라 처리하고 싶을 때 사용하는데요.

예를 들어, 물건 데이터를 자료구조에 넣었다가 가치가 높은 물건부터 꺼내서 확인해야 하는 경우 사용합니다.
[예시 출처. <https://www.youtube.com/watch?v=AjFlp951nz0>]

우리의 관심사, 코딩테스트 문제에서는 최소값, 최대값을 계속해서 호출해야 하는 경우 사용될 수 있을 것 같습니다.

자료구조 4. 우선순위 큐

HOW: 우선순위 큐 어떻게 만드는데?

우선순위 큐(Priority Queue)

1. 리스트를 이용하는 방법
2. 힙(heap)을 이용하는 방법

구현 방식	삽입 시간	삭제 시간
리스트	$O(1)$	$O(N)$
힙 (Heap)	$O(\log N)$	$O(\log N)$

힙으로 구현되는 경우 N 개의 데이터를 힙에 넣었다가 모두 꺼내는 작업은 힙 정렬과 동일합니다.

간단하게 말해서, 시간 복잡도가 $O(N \log N)$ 이라는 이야기에요!

자료구조 4. 우선순위 큐

HOW: 그래서 어떻게 사용하나요!!

heap에 대해 자세히 알아보면 좋겠지만,, (대신 [자료구조: 우선순위 큐\(Priority Queue\)와 힙\(Heap\) 10분 핵심 요약](#)을 봐주세요)

(👍 진짜로 10분정도 길이에 제가 전달하고 싶은 내용이 너무 잘 정리되어 있어요!! 참고해주세요)

(원하시면 제가 나중에 따로 정리해서 제공해드리겠습니다)



자 이론이 너무 길어지면 빨리 지칠 것 같으니까 거두절미하고
파이썬에서는 우선순위 큐를 어떻게 사용할 수 있을지 알아보죠-

파이썬에서 기본으로 제공하는 heapq 라이브러리를 사용할 겁니다!

heapq는 일반적인 리스트와 다르게 가지고 있는 요소를 push, pop 할때마다 자동으로
정렬해주는 녀석입니다.

(🔥 파이썬에서 제공하는 heap 라이브러리는 **값이 낮은 순(우선순위가 높다)**으로 먼저 뽑아내는 **최소 힙**입니다~)

(파이썬 너란 녀석 도대체 어디까지 도와주는거니...👍)

자료구조 4. 우선순위 큐

HOW: 메서드

기본적으로 아래 4가지를 꼭

기억해주세요. (사용방법은 알아야 하니까요!)

```
import heapq
```

힙 큐(우선순위 큐)를 사용하기 위해 heapq 모듈을 import한다.

`heapify()`

기존 배열을 heap 구조로 만든다.

`heappush(배열이름, 요소)`

heap(배열)에 요소를 추가한다.

`heappop(배열이름)`

heap(배열)에서 요소를 제거한다.

+ 우선순위큐·heapq를 사용할 수 있는 괜찮은 문제들입니다!!!
(이 문제들은 나중에 같이 한 번 풀어보죠!)

1. [15903번: 카드 합체 놀이](#)
2. [코딩테스트 연습 - 더 맵게](#)

```
import heapq

# 기존 데이터 heap 구조로 변환
heap_data = [1,3,2,6,8,0,6]
heapq.heapify(heap_data)
# [0,3,1,6,8,2,6]

# 요소 추가하기
heap_data = []
heap.heappush(heap_data, 3)
heap.heappush(heap_data, 5)
heap.heappush(heap_data, 1)
heap.heappush(heap_data, -3)
# [-3,1,3,5]

# 요소 제거하기
heap.heappop(heap_data)
heap.heappop(heap_data)
# [3,5]
```

유용한 내장함수

유용한 내장함수 - map과 split

map 함수와 split 함수

알고리즘 문제들을 접하다보면, 입력 한 줄에 여러 요소들을 입력받아야 하는 경우가 대부분입니다.

이 때 활용할 수 있는 파이썬 내장함수가 바로 map 함수와 split 함수 !!!

문제를 풀며 정말 자주 사용하게 될 테니 잘 익혀두는 것이 좋아요.

유용한 내장함수 - map과 split

map 함수란?

iterable객체(list, tuple, dict, set)를 받아서 각 요소에 함수를 적용해주는 함수입니다.

※ 사용법 ※

map(적용시킬 함수, 적용할 요소들)

사용 예시

```
# 전달받은 매개변수에 1을 더해 돌려주는 함수 선언
def add_one(n):
    return n + 1

# 함수에 적용할 임의의 리스트 선언
target = [1, 2, 3, 4, 5]

# map 함수를 활용해 리스트의 각 요소에 add_one 함수 적용
result = list(map(add_one, target))
print(result)
```

출력 결과

```
[2, 3, 4, 5, 6]

Process finished with exit code 0
```

유용한 내장함수 - map과 split

split 함수란?

특정 문자를 기준으로 문자열을 분리해주는 함수입니다.

※ 사용법 ※

문자열.split()

사용 예시

```
# 특정 문자열 선언
temp1 = "I L00000VE SOPTAC !!!"
temp2 = "I♥LOVE♥SOPTAC♥!!!"
# split 함수를 활용해 공백을 기준으로 문자열 분리
print(temp1.split())
# split 함수를 활용해 ♥를 기준으로 문자열 분리
print(temp2.split("♥"))
```

출력 결과

```
['I', 'L00000VE', 'SOPTAC', '!!!']
['I', 'LOVE', 'SOPTAC', '!!!']

Process finished with exit code 0
```


유용한 내장함수 - map과 split

알고리즘 문제에서 map과 split을 활용하는 법!

문제 예시

세 자연수 A, B, C가 주어졌을 때, A, B, C의 합을 구하는 프로그램을 작성하시오.

입력 조건

첫째 줄에 A, B, C가 순서대로 주어진다.

출력 조건

첫째 줄에 A, B, C의 합을 출력한다.

유용한 내장함수 - map과 split

알고리즘 문제에서 map과 split을 활용하는 법!

풀이 예시

int 형으로 타입을 변환 해주는 함수

```
A, B, C = map(int, input().split())  
print(A+B+C)
```

공백을 기준으로
입력 값들을 분리

입력과 출력

```
3 5 7  
15
```

유용한 내장함수 - sorted

sorted 함수는 iterable객체가 들어왔을 때, 정렬된 결과를 반환해주는 함수입니다.

정렬을 하는 방법에는 여러가지 방법이 있지만, 시간복잡도를 고려하지 않아도 되는 간단한 경우,

내장함수를 활용하여 빠르고 쉽게 결과를 얻을 수 있습니다.

또한 'key'속성으로 정렬 기준을 명시할 수 있으며, 'reverse'속성으로 역정렬도 가능한 강력한 내장함수입니다.

```
numbers = [7, 1, 8, 5, 4]

result = sorted(numbers)
print(result)

result = sorted(numbers, reverse=True)
print(result)

SOPTAC = [('화이팅', 486), ('습탁', 2021), ('여러분', 1004)]
result = sorted(SOPTAC, key=lambda x:x[1], reverse=True)
print(result)
```

```
[1, 4, 5, 7, 8]
```

```
[8, 7, 5, 4, 1]
```

```
[('습탁', 2021), ('여러분', 1004), ('화이팅', 486)]
```

유용한 내장함수 - 연산 관련 내장함수들

sum 함수 : iterable객체가 입력으로 주어졌을 때 모든 원소의 합을 반환한다.

```
numbers = [1, 2, 3, 4, 5]
result = sum(numbers)
print(result)
```

15

min 함수 & max 함수 : 매개변수가 2개 이상 들어왔을 때 가장 작은/큰 값을 반환한다.

```
result = min(7, 3, 5, 2)
print(result)
result = max(7, 3, 5, 2)
print(result)
```

2

7

round 함수 : 반올림을 수행할 수 있다. 두 번째 매개변수로 반올림할 소숫점 위치를 결정할 수 있다.

```
print(round(123.456))
print(round(123.456, 2))
print(round(123.456, -1))
```

123

123.46

120.0

유용한 내장함수 - 기타

abs 함수 : 정수가 입력되었을 때, 해당하는 절대값을 반환해줌

```
print(abs(-77)) 77
print(abs(77)) 77
```

ord 함수 & chr 함수 : 문자/아스키코드가 입력되었을 때, 해당하는 아스키코드/문자를 반환해줌

```
print(ord('U')) 85
print(chr(85)) U
```

enumerate 함수 : 순서가 있는 자료형(list, tuple, 문자열)을 입력으로 받아 인덱스 값을 포함하는 객체를 반환해줌 ※주로 for문과 함께 사용함※

```
soptac = ['S', 'O', 'P', 'T', 'A', 'C']
for i, name in enumerate(soptac):
    print(i, name)
```

```
0 S
1 O
2 P
3 T
4 A
5 C
```

유용한 라이브러리 - math : 수학 관련 모든 것

삼각함수 친구들

math.pi	π
math.degrees(라디안)	라디안 \rightarrow 도
math.radians(도)	도 \rightarrow 라디안
math.cos(라디안)	cosin
math.sin(라디안)	sin
math.tan(라디안)	tangent
math.acos(라디안)	secant
math.asin(라디안)	cosecant
math.atan(라디안)	cotangent

로그 친구들

math.e	자연상수 e
math.log(밑.진수)	로그함수
math.log10(진수)	밑이 10의 로그함수

기타등등 친구들

math.trunc(실수)	실수의 정수 부분
math.factorial(정수)	1부터 해당 수까지 곱한 값
math.pow(밑.지수)	밑의 지수제곱
math.sqrt(실수)	실수의 제곱근

유용한 라이브러리 - itertools : 반복 관련 모든 것

확통 친구들

itertools.permutations(**p**,**r**)

순열 (순서o 반복x)

permutations('ABCD', 2) --> AB AC AD BA BC BD CA CB CD DA DB DC

itertools.combinations(**p**,**r**)

조합 (순서x 반복x)

combinations('ABCD', 2) --> AB AC AD BC BD CD

itertools.combinations_with_replacement(**p**,**r**)

중복조합 (순서x 반복o)

combinations_with_replacement('ABC', 2) --> AA AB AC BB BC CC

itertools.product(**p**,**q**...repeat=**r**)

곱집합

product('ABCD', 'xy', repeat=1) --> Ax Ay Bx By Cx Cy Dx Dy

유용한 라이브러리 - bisect : 정렬된 배열에서의 탐색

인덱스 찾는 친구들

`bisect.bisect_left(리스트, 데이터)` 리스트가 정렬된 순서를 유지하도록 데이터를 삽입할 왼쪽 위치의 인덱스

`bisect.bisect_right(리스트, 데이터)` 리스트가 정렬된 순서를 유지하도록 데이터를 삽입할 오른쪽 위치의 인덱스

```
>>> import bisect
>>> a = [1,2,4,4,4,9]
>>> bisect.bisect_left(a,4)
2
>>> bisect.bisect_right(a,4)
5
>>> bisect.bisect_left(a,8)
5
>>> bisect.bisect_right(a,8)
5
.
```



원소의 개수를 구할 수 있다!

유용한 라이브러리 - collections : 새로운 데이터형

deque 덱

큐: popleft() / 스택: pop() 맨 앞 원소 제거 / 맨 뒤 원소 제거

append(x) 맨 뒤에 원소 삽입

appendleft(x) 맨 앞에 원소 삽입



Queue로 쓰일 때는 list보다 빠름!

인덱스 접근이 안 되니까 중간 접근 필요한 건 list로!

Stack은 차이 없음!

Counter

등장 횟수 세는 기능 제공

```
from collections import Counter

counter = Counter(['red', 'blue', 'red', 'green', 'blue', 'blue'])
print(counter['red']) # 'red'가 등장한 횟수 출력
print(counter['blue']) # 'blue'가 등장한 횟수 출력
print(dict(counter)) # 딕셔너리형으로 변환
```



2

3

{'red': 2, 'blue': 3, 'green': 1}

파이썬 최강자 이다은이 제작한 노션

<https://www.notion.so/SOPTAC-1st-e871b2c187b34bf1bea0abc478ecd51a>