ILE3-006 리스트, 딕셔너리

리스트

데이터를 순차적으로 저장하는 자료구조
배열 or C++의 vector와 유사

생성

```
ret = list()
ret = []
ret = [1,2,3,4]
```

사용 가능한 형식

별도로 사용할 형식을 지정하지 않으며, 서로 다른 형식의 데이터도 사용 가능

```
ret = [1, 2, 3, 4, 'five', True, None]
ret = [ [1, 2, 3, 4], 'five', True, None] # 리스트도 넣을 수 있음
```

기본 기능

삽입

`insert()`, `append()` 함수를 이용하여 데이터를 삽입

맨 뒤에 삽입
`append()` 함수 사용

```
>>> ret = [1, 2, 3, 4]
>>> ret.append(5)
>>> ret
[1, 2, 3, 4, 5]
```

지정한 Index에 삽입
`insert()` 함수 사용

```
>>> ret = [1, 2, 3, 4]
>>> ret.insert(3, True)
>>> print(ret)
[1, 2, 3, True, 4]
```

삽입된 항목 개수 확인

`len()` 함수를 이용하여 삽입된 데이터의 개수 확인

```
>>> ret = [1,2,3,4]
>>> len(ret)
4
```

읽기

배열과 동일하게 `대괄호 []`를 이용하여 Index를 지정하여 읽음

```
list[INDEX]
```

```
>>> ret = [1,2,3,4]
>>> ret[2]
3
```

수정

배열과 동일하게 `대괄호 []`를 이용하여 Index를 지정하여 데이터를 수정

```
>>> ret = [1, 2, 3, 4]
>>> ret[2] = True
>>> print(ret)
[1, 2, True, 4]
```

삭제

전부 다 삭제

`clear()` 함수 사용

```
list.clear()
```

```
>>> ret = [1, 2, 3, 2, 4, 2, 5]
>>> ret.clear()
>>> ret
[]
```

Index를 이용하여 삭제: del

`del`과 `대괄호 []`를 이용하여 삭제

범위를 지정하여 삭제도 가능

```
del list[INDEX]
```

```
>>> ret = [1, 2, 3, 4]
>>> del ret[1]
>>> ret
[1, 3, 4]

>>> ret = [1, 2, 3, 4]
>>> del ret[1:3]
>>> ret
[1, 4]

>>> ret = [1, 2, 3, 4]
>>> del ret[2:]
>>> ret
[3, 4]

>>> ret = [1, 2, 3, 4]
>>> del ret[:2]
>>> ret
[3, 4]
```

Index를 이용하여 삭제: pop

`pop(INDEX)` 함수는 데이터를 삭제하나, 삭제 되는 데이터를 반환 함

매개변수를 제공하지 않는 경우에는 마지막 데이터를 삭제

```
list.pop(INDEX)
```

```
>>> ret = [1, 2, 3, 4]
>>> ret.pop(1)
2
>>> ret
[1, 3, 4]
>>> ret.pop()
4
>>>ret
[1, 3]
```

값을 이용하여 삭제

`remove()` 함수를 이용하여 데이터를 삭제하며, 중복된 데이터가 존재하는 경우에는 맨 앞에 존재하는 데이터만 제거 됨

```
list.remove(VALUE)
```

```
>>> ret = [1, 2, 3, 2, 4, 2, 5]
>>> ret.remove(2)
>>> ret
[1, 3, 2, 4, 2, 5]
>>> ret.remove(2)
>>> ret
[1, 3, 4, 2, 5]
```

리스트의 연산

두 개의 리스트의 병합

첫번째 List에 두번째 List를 모두 삽입

`+` 연산

두개의 리스트를 합쳐 새로운 리스트를 생성

> 피연산자는 변경되지 않음

```
new_list = list1 + list2
```

```
>>> l1 = [1, 2, 3]
>>> l2 = [3, 4, 5]
>>> ret = l1 + l2
>>> ret
[1, 2, 3, 3, 4, 5]
>>> l1
[1, 2, 3]
>>> l2
[3, 4, 5]
```

`extend()` 함수

리스트의 뒤에 전달받은 리스트를 추가

```
list1.extend(list2)
```

```
>>> ret = [1, 2, 3]
>>> l1 = [3, 4, 5]
>>> ret.extend(l1)
>>> ret
[1, 2, 3, 3, 4, 5]
```

```
>>> l1
[3, 4, 5]
```

데이터 분석

최대, 최소값

`min()`, `max()` 함수를 이용하며, 문자열이 들어있는 리스트의 경우에는 사용 불가

> 정확하게는 `<` 연산자를 사용 할 수 있는 형식이면 가능

```
min(list)
max(list)
```

```
>>> ret = [1, 2, 3, 3.2, 4]
>>> min(ret)
1
>>> max(ret)
4
>>> ret = [1, 2, 3, 3.2, 4, 'ABCD']
>>> min(ret)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and 'int'
```

합계

`sum()` 함수를 이용하며, 정수,소수 이외의 형식이 들어있는 리스트의 경우에는 사용 불가

```
min(list)
max(list)
```

```
>>> ret = [1, 2, 3, 3.2, 4]
>>> sum(ret)
13.2
>>> ret = [1, 2, 3, 3.2, 4, 'ABCD']
>>> sum(ret)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'float' and 'str'
```

반복문에서 사용

`for` 문에서 리스트를 입력으로 사용 가능

```
for cur in list:
    pass
```

```
>>> ret = [1, 2, 3, 3.2, 4]
>>> for cur in ret: print(ret)
1
2
3
3.2
4
```

정렬

`sort()` 함수를 이용하며, 정렬 방향에 대한 정의가 가능

```
list.sort(reverse=False)
```

```
>>> ret = [2, 3, 1, 4]
>>> ret.sort()
[1, 2, 3, 4]
>>> ret.sort(reverse=True)
[4, 3, 2, 1]
```

튜플

리스트와 거의 비슷하나, 데이터의 편집이 불가능한 상수

```
t1 = ()
t2 = (1,)
t3 = (1, 2, 3)
t4 = 1, 2, 3
t5 = ('a', 'b', ('ab', 'cd'))
```

삭제 불가

```
>>> t = (1, 2, 3)
>>> t.remove(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'remove'
```

수정 불가

```
>>> t = (1, 2, 3)
>>> t[1] = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

딕셔너리

키를 기반으로 값 저장
키, 값 쌍으로 저장 됨

생성

```
ret = dict()
ret = {}
```

```
ret = { 'name': '홍길동', 'grade': '1학년' }
```

사용 가능한 데이터

키와 값 모두 데이터형을 구분하지 않음

- 선언되지 않은 변수는 사용 불가
- 중복되는 키값이 있을 경우에는 마지막 값이 출력 됨

```
ret = { 1: 'One', 'Two': 2, None: 'NULL', False: 0, 'list': [1,2,3,4], 'dict': { 'a':1 } }
```

```
>>> ret = { 1: 'One', 1: '하나' }
>>> print(ret)
{ 1: '하나' }
```

기본 기능

항목의 개수

```
len(dict)
```

```
>>> ret = { 'One': 1, 'Two': 2, 'Three': 3, 'Four': 4 }
>>> len(ret)
4
```

Read

키 이름으로 찾기

```
dict[읽을 키]
```

> 존재하지 않는 키를 입력한 경우에는 KeyError 발생

```
>>> ret = { 'One': 1, 'Two': 2, 'Three': 3, 'Four': 4 }
>>> print(ret['One'])
1
```

안전하게 읽기

```
dict.get([읽을 키])
```

> 존재하지 않는 키를 입력한 경우에는 None을 반환

```
>>> ret = { 'One': 1, 'Two': 2, 'Three': 3, 'Four': 4 }
>>> ret.get('One')
1
>>> ret.get('Zero')
None
```

데이터가 있는지만 확인

```
[확인할 키] in dict
[확인할 키] not in list
```

```
>>> ret = { 'One': 1, 'Two': 2, 'Three': 3, 'Four': 4 }
>>> 'One' in ret
True
```

```
>>> 'Zero' in ret
False
>>> 'One' not in ret
False
>>> 'Zero' not in ret
True
```

Insert or Update

```
dict[기록할 키] = [기록할 값]
```

```
>>> ret = {}
>>> ret['One'] = 2
>>> print(ret)
{ 'One': 2 }
>>> ret['One'] = 1
>>> print(ret)
{ 'One': 1 }
```

Delete

```
del dict[삭제할 키]
```

> 존재하지 않는 키 입력 시 Key Error 발생

```
>>> ret = { 'One': 1, 'Two': 2, 'Three': 3, 'Four': 4 }
>>> del ret['Four']
>>> print(ret)
{ 'One': 1, 'Two': 2, 'Three': 3 }
```

딕셔너리 함수

키 리스트

```
dict.keys()
```

```
>>> ret = { 'One': 1, 'Two': 2, 'Three': 3, 'Four': 4 }
>>> ret.keys()
dict_keys(['One', 'Two', 'Three', 'Four'])
```

값 리스트

```
dict.values()
```

> 존재하지 않는 키 입력 시 Key Error 발생

```
>>> ret = { 'One': 1, 'Two': 2, 'Three': 3, '하나': 1}
>>> ret.values()
dict_values([1, 2, 3, 1])
```

키/값 리스트

```
dict.items()
```

```
>>> ret = { 'One': 1, 'Two': 2, 'Three': 3, 'Four': 4 }
>>> ret.items()
dict_items([('One', 1), ('Two', 2), ('Three', 3), ('Four', 4)])
```

딕셔너리를 이용한 반복문

딕셔너리를 입력으로 사용 한 경우

```
ret = { 'One': 1, 'Two': 2, 'Three': 3, 'Four': 4 }
for v in ret:
    print(v)
```

```
One
Two
Three
Four
```

키, 값 리스트를 입력으로 사용한 경우

키 리스트

```
ret = { 'One': 1, 'Two': 2, 'Three': 3, 'Four': 4 }
for v in ret.keys():
    print(v)
```

```
One
Two
Three
Four
```

값 리스트

```
ret = { 'One': 1, 'Two': 2, 'Three': 3, 'Four': 4 }
for v in ret.values():
    print(v)
```

```
1
2
3
4
```

키,값 쌍을 입력으로 사용하는 경우

```
ret = { 'One': 1, 'Two': 2, 'Three': 3, 'Four': 4 }
for v in ret.items():
    print(v)
```

```
('One', 1)
('Two', 2)
('Three', 3)
('Four', 4)
```

```python
ret = { 'One': 1, 'Two': 2, 'Three': 3, 'Four': 4 }
for k, v in ret.items():
    print(f'{k}: {v}')
```

```
One: 1
Two: 2
Three: 3
Four: 4
```

과제

입력된 데이터의 값에 따른 개수 출력
출력값은 값으로 정렬

Input

```
[1, 2, 1, 3, 3, 3]
```

Output

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

과제

검색 기능을 사용할 때는 항상 문자열의 전체를 이용하여 검색을 수행하지 않는다.
예를 들어 "판교역 신분당선"을 검색하는 경우 "판교역", "판교" 등등으로 입력이 가능하다.

이를 구현하기 위해서는 단순하게는 각 문자열에 대하여 검색을 수행하는 방법이 존재하지만, 입력 데이터의 개수가 늘어나는 경우에는 검색 속도가 매우 느려질 수 있다.

이를 위하여 문자열을 아래와 같이 분리하여 저장한 데이터를 이용하여 빠르게 검색을 수행할 수 있다.

```
"판교역신분당선"
-> "판", "교", "역", "신", "분", "당", "선"
-> "판교", "교역", "역신", "신분", "분당", "당선"
-> "판교역", "교역신", "역신분", "신분당", "분당선"
-> "판교역신", "교역신분", "역신분당", "신분당선"
-> "판교역신분", "교역신분당", "역신분당선"
-> "판교역신분당", "교역신분당선"
-> "판교역신분당선",
```

입력 데이터

입력 데이터는 아래와 같이 구성됩니다.

- 검색을 수행할 명칭 (1줄)
- 검색을 수행할 명칭과 검색 대상을 각각 한줄 씩 사용한다.

> 띄워쓰기는 없습니다.

```
분당
판교역신분당선
판교역장항선
정자역분당선
```

출력 데이터

검색된 결과 데이터를 `output.txt`로 출력

```
판교역신분당선
정자역분당선
```