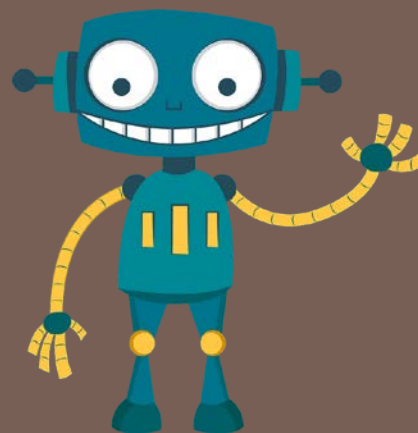
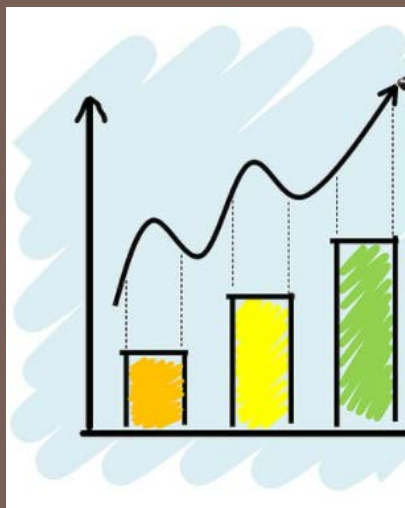


파이썬 익스프레스



7장 파이썬 자료구조 II

학습 목표

- 튜플을 이해하고 사용할 수 있다.
- 세트를 이해하고 활용할 수 있다.
- 딕셔너리를 이해하고 활용할 수 있다.
- 문자열의 각종 연산을 이해하고 활용할 수 있다.



이번 장에서 만들 프로그램

1. 연락처 추가
2. 연락처 삭제
3. 연락처 검색
4. 연락처 출력
5. 종료

메뉴 항목을 선택하시오: 1

이름: KIM

전화번호: 123-4567

1. 연락처 추가
2. 연락처 삭제
3. 연락처 검색
4. 연락처 출력
5. 종료

메뉴 항목을 선택하시오: 4

KIM 의 전화번호: 123-4567

...

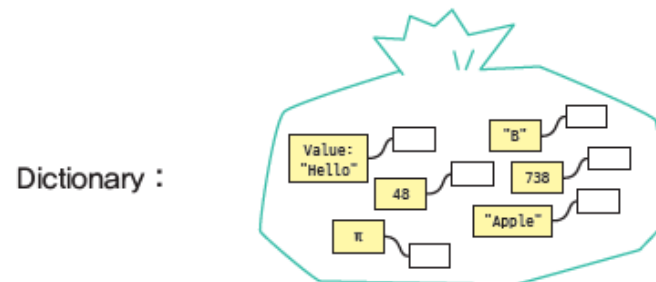
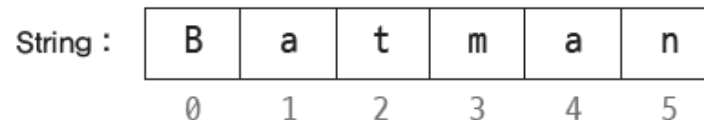
자료구조

- 자료들을 저장하는 여러 가지 구조들을 **자료구조(data structure)**, 또는 **데이터구조**라 부른다.
- **시퀀스(sequence):**
 - ▣ 요소(**element**)로 구성
 - ▣ 요소 간에는 순서가 있다.
 - ▣ 시퀀스의 요소들은 번호가 붙여져 있다.
 - ▣ 내장 시퀀스(**str, bytes, bytearray, list, tuple, range**)



시퀀스

- 동일한 연산 지원
 - ▣ 인덱싱(indexing), 슬라이싱(slicing), 덧셈 연산(adding), 곱셈 연산(multiplying)
- 내장함수 적용가능
 - ▣ 시퀀스의 길이를 반환하는 len() 함수, 최대값과 최소값을 찾는 max()와 min() 함수



튜플

- 리스트와 **튜플(tuple)**은 아주 유사하다.
- 하지만 리스트와는 다르게 **튜플은 변경이 불가능**하다.

Syntax: 튜플

형식 튜플_이름 = (항목1, 항목2, ...)

공백 튜플을 생성한다.

예 fruits = ()

fruits = ("apple", "banana", "grape")

초기값을 가진 튜플을 생성한다.

result = fruits[1]

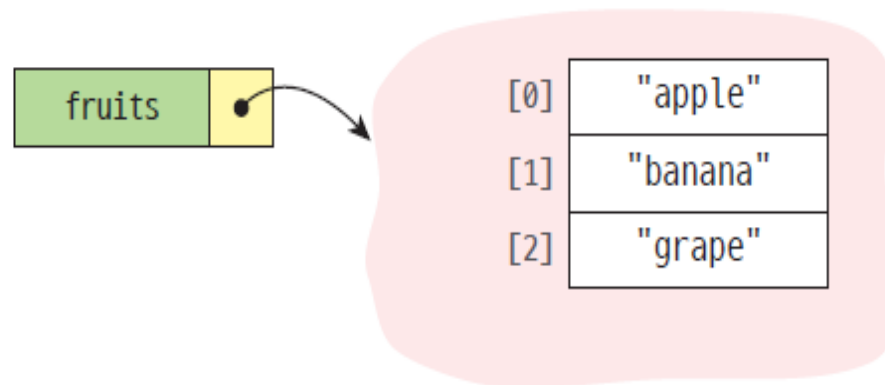
인덱스를 사용하여 요소에 접근한다.

튜플의 이름

튜플 생성

```
fruits = ("apple", "banana", "grape")
```

```
fruits = "apple", "banana", "grape"
```



주의할 점

```
>>> single_tuple = ("apple",)      # 심표가 끝에 있어야 한다.
>>> single_tuple
("apple",)                        # 튜플
>>> no_tuple = ("apple")           # 심표가 없으면 튜플이 아니라 문자열이 된다.
>>> no_tuple
"apple"                           # 문자열
```

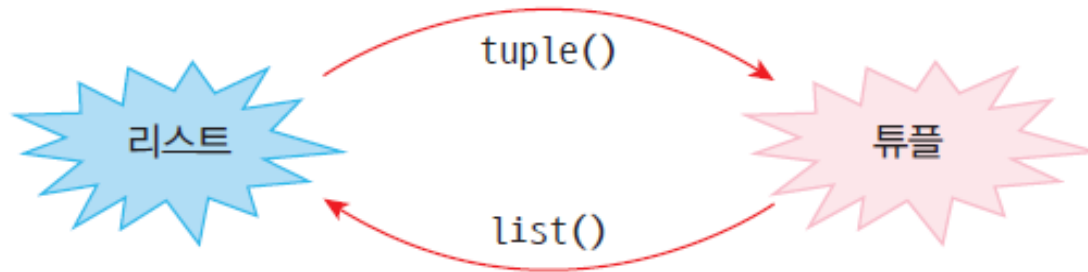
```
>>> fruits = ("apple", "banana", "grape")
>>> fruits[1]
banana

>>> fruits[1] = "pear"             # 오류 발생!
TypeError: "tuple" object does not support item assignment
```


튜플 ↔ 리스트

```
>>> myList = [1, 2, 3, 4]
>>> myTuple = tuple(myList)
>>> myTuple
(1, 2, 3, 4)
```

tuple()는 튜플을 생성하는 함수이다.



```
>>> myTuple = (1, 2, 3, 4)
>>> myList = list(myTuple)
>>> myList
[1, 2, 3, 4]
```

list()는 리스트를 생성하는 함수이다.

튜플 추가 연산들

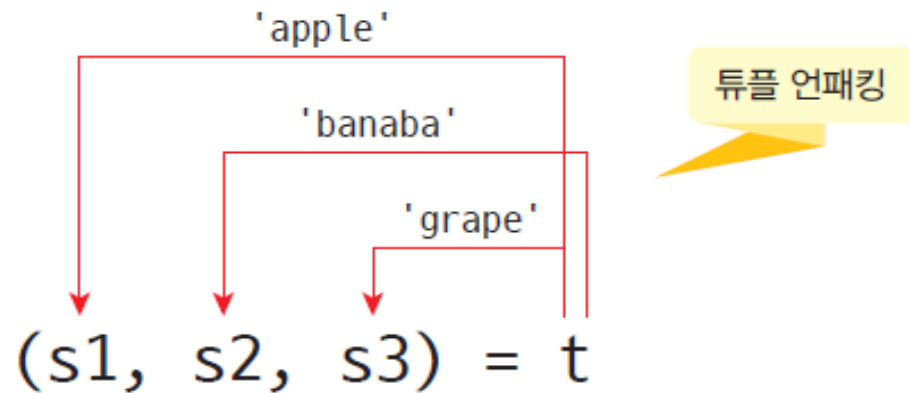
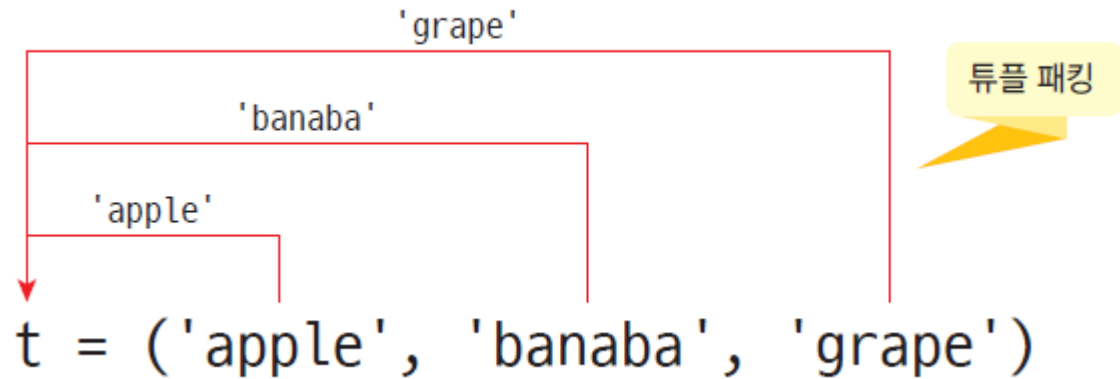
```
>>> fruits = ("apple", "banana", "grape")
>>> fruits += ("pear", "kiwi")
>>> fruits
("apple", "banana", "grape", "pear", "kiwi")
```

- 다른 튜플에 합치는 것은 가능
- id 함수로 참조값을 확인해 보면?
print(id(fruits))

```
>>> numbers = [10, 20, 30]
>>> numbers += (40, 50)
>>> numbers
[10, 20, 30, 40, 50]
```

- 리스트에 튜플을 합치는 것은 가능
- id 함수로 참조값을 확인해 보면?
print(id(fruits))

튜플 패킹과 언패킹



예제

n1 = 10

n2 = 90

n1, n2 = (n2, n1)

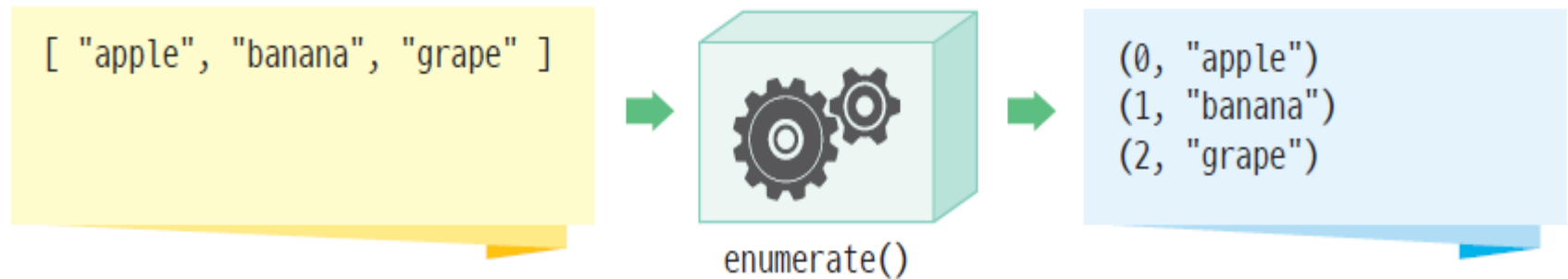
튜플을 이용하여 데이터의 순서를
바꾼다.

(90 10)

n1, n2 = sub()

함수로부터 2개 이상의 값을 반환받는
것도 튜플을 통하여 구현된다.

enumerate() 사용하기



```
fruits=["apple","banana","grape"]  
for index, value in enumerate(fruits):  
    print(index, value)
```

```
0 apple  
1 banana  
2 grape
```

튜플의 장점

	리스트	튜플
문법	항목을 []으로 감싼다.	항목을 ()으로 감싼다.
변경여부	변경 가능한 객체	변경 불가능한 객체
메소드	약 46개의 메소드 지원	약 33개의 메소드 지원
용도	딕셔너리에서 키로 이용할 수 없다.	딕셔너리에서 키로 이용할 수 있다.

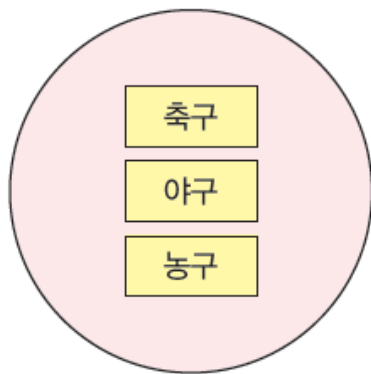
중간점검

1. 리스트와 튜플의 다른 점은 무엇인가?
2. 리스트를 튜플로 바꾸려면 어떤 함수를 사용하는가?
3. 패킹과 언패킹을 설명해보자.
4. `enumerate()` 함수가 하는 역할은 무엇인가?

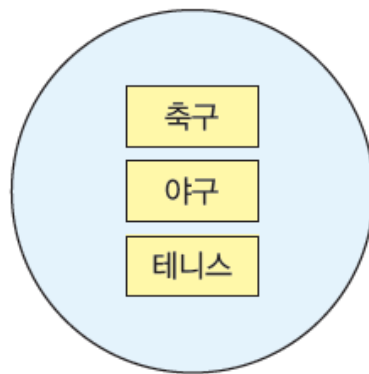


세트

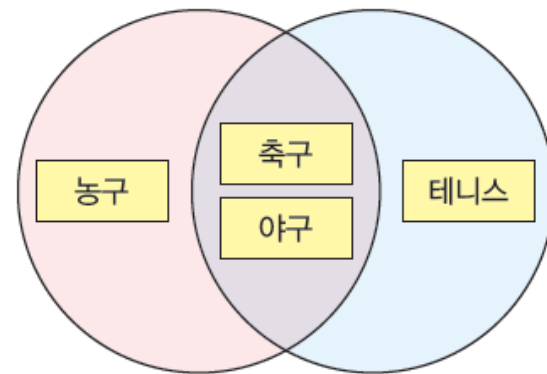
- **세트(set)**는 우리가 수학에서 배웠던 **집합**이다.
 - ▣ 세트는 **고유한 값들을** 저장하는 자료구조라고 할 수 있다.
- 리스트와는 다르게 세트의 요소는 **특정 순서로** 저장되지 않으며 **위치별로 액세스할 수 없다**.



세트 #1



세트 #2



세트 #1 ∩ 세트 #2

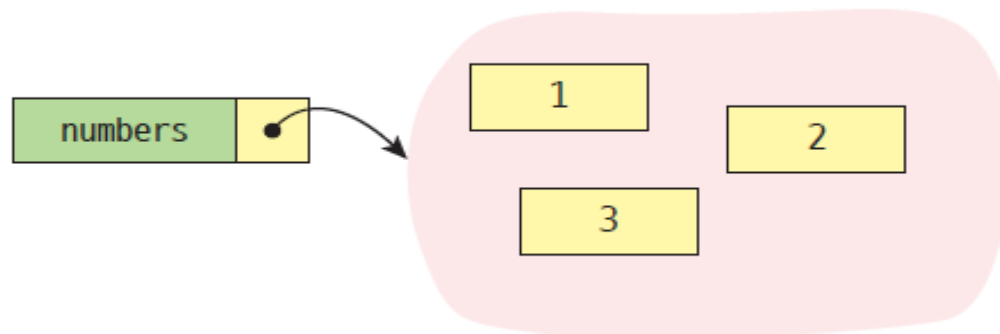
세트 생성하기

Syntax: 세트

형식 세트_이름 = { 항목1, 항목2, 항목3, ... }

예 numbers = {1, 2, 3} 초기화된 세트를 생성한다.
values = set() 공백 세트를 생성한다.

공백 세트가 { }가 아니다.



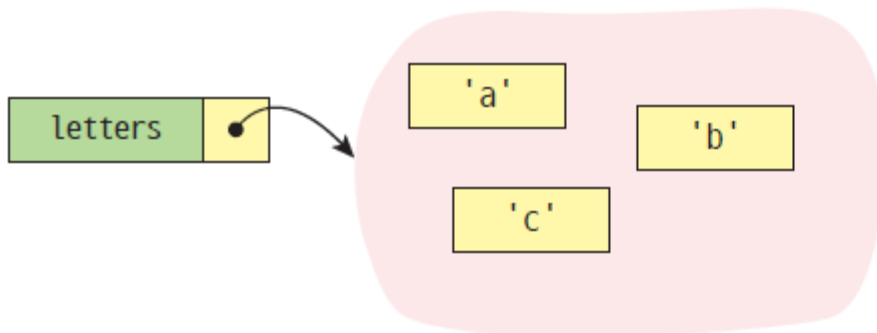
리스트 ↔ 세트

```
numbers = set([1,2,3,1,2,3])
```

```
print(numbers)
```

```
{ 1, 2, 3}
```

```
letters = set("abc")
```



문자열을 분해하여
세트로 만들 수 있어요.



세트의 연산

- **all(), any(), enumerate(), len(), max(), min(), sorted(), sum()** 사용 가능

```
fruits = {"apple", "banana", "grape"}  
size = len(fruits)           # size는 3이 된다.
```

```
fruits = { "apple", "banana", "grape" }  
if "apple" in fruits:  
    print("집합 안에 apple이 있습니다.")
```

```
집합 안에 apple이 있습니다.
```

세트의 연산

```
fruits={"apple", "banana", "grape"}  
for x in fruits:  
    print(x, end=" ")
```

grape banana apple # 실행할 때마다 다른 출력 결과 나옴

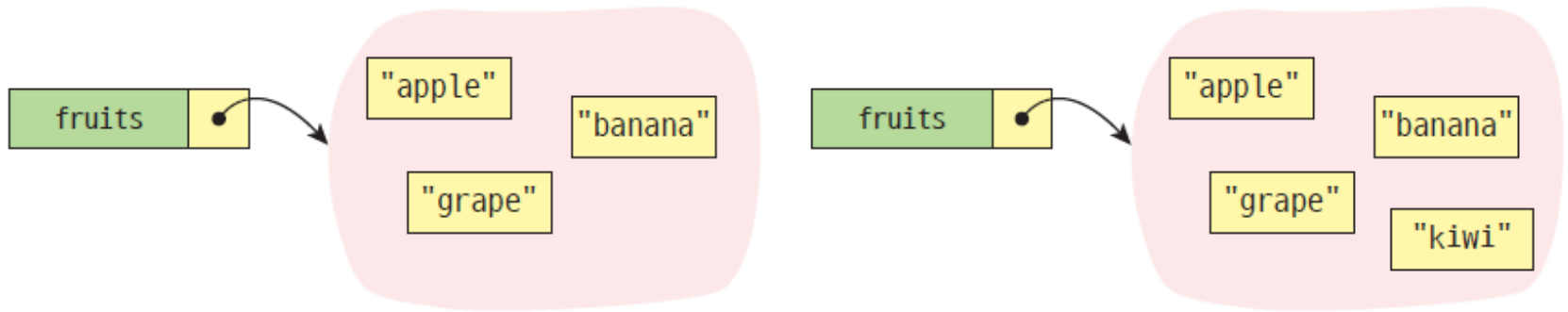
```
fruits={"apple", "banana", "grape"}  
for x in sorted(fruits):  
    print(x, end=" ")
```

apple banana grape

세트에 요소 추가하기

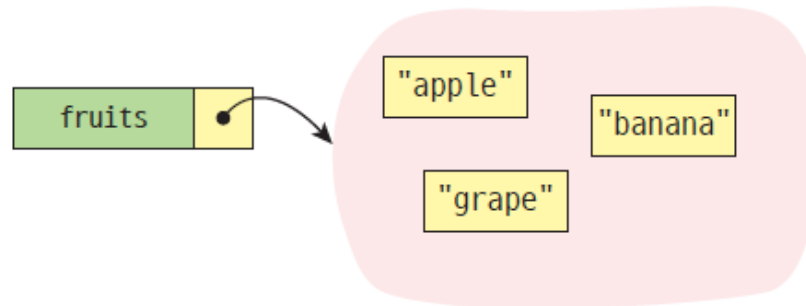
```
fruits = {"apple", "banana", "grape"}
```

```
fruits.add("kiwi")
```

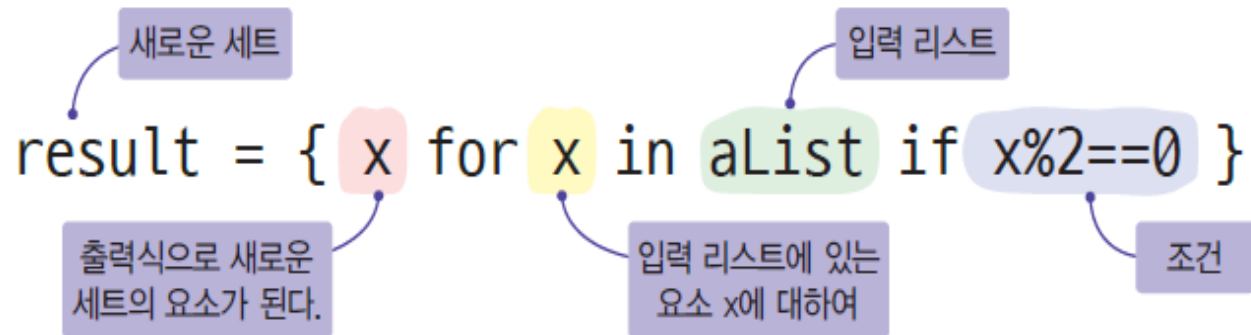


```
fruits.remove("kiwi")
```

삭제하는 요소가 없으면 예외 발생, 예외 발생시키지
않으려면 `discard()` 사용!



세트 함축 연산



```
aList =[1,2,3,4,5,1,2 ]  
result ={ x for x in aList if x%2==0 }  
print(result)
```

🔍 실행결과

{2, 4}

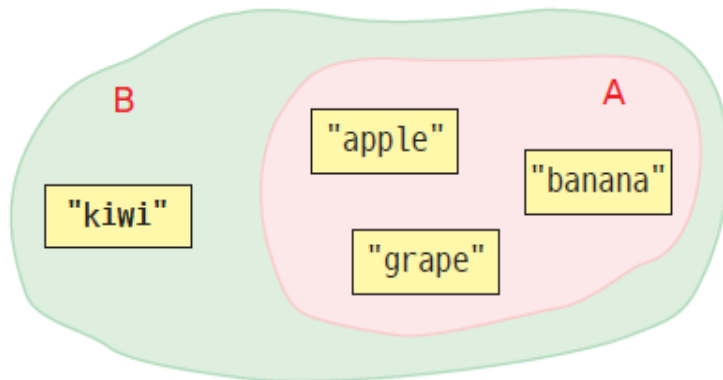
부분 집합 연산

```
A = {"apple", "banana", "grape"}  
B = {"apple", "banana", "grape", "kiwi"}
```

```
if A < B :                # 또는 A.issubset(B) :  
    print("A는 B의 부분 집합입니다.")
```

실행결과

A는 B의 부분 집합입니다.



부분 집합은 < 으로
검사할 수 있어요!



==, != 연산

```
A={"apple","banana","grape"}  
B={"apple","banana","grape","kiwi"}
```

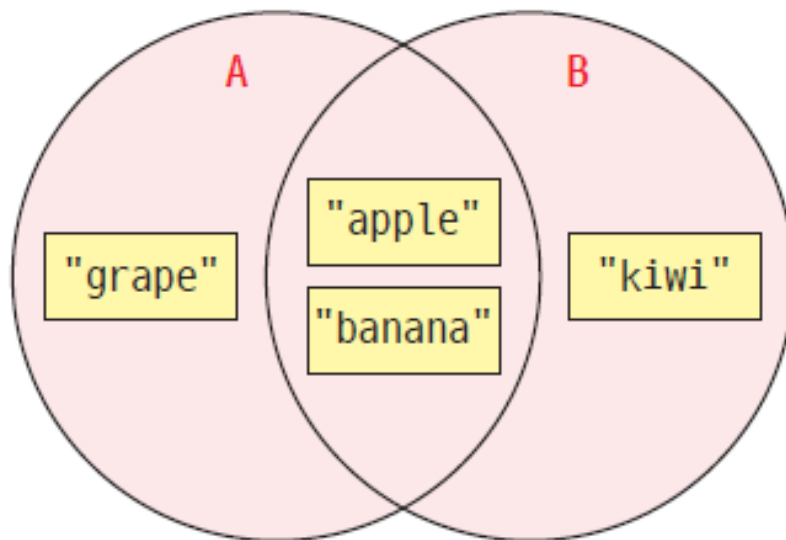
```
if A == B :  
    print("A와 B는 같습니다.")  
else :  
    print("A와 B는 같지 않습니다.")
```

A와 B는 같지 않습니다.

합집합

$C = A \cup B$

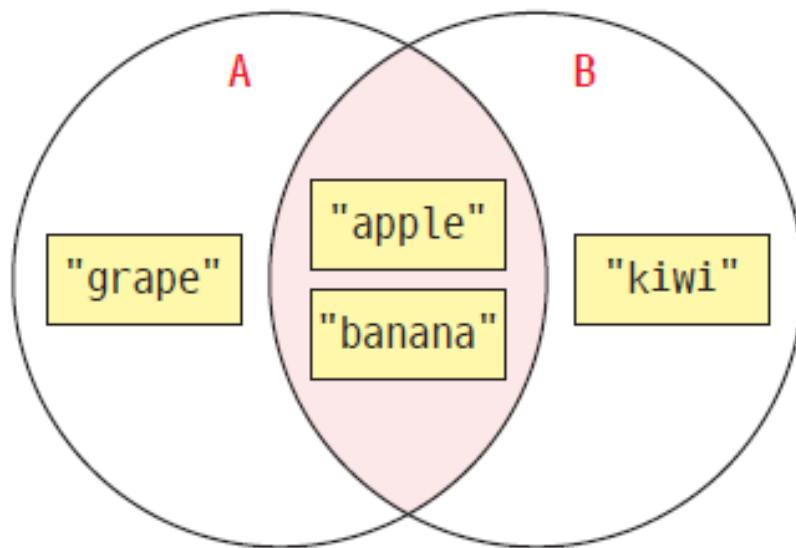
또는 $C = A.union(B)$



교집합

$C = A \& B$

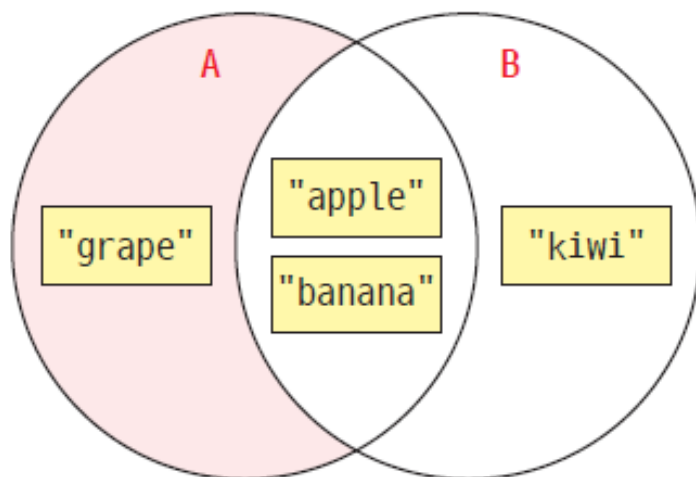
또는 $C = A.intersection(B)$



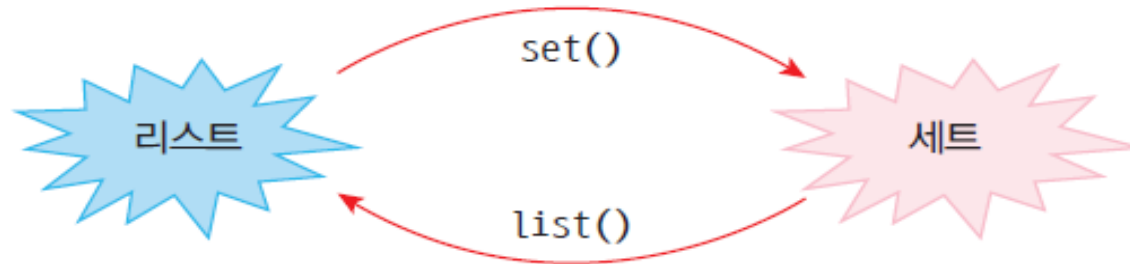
차집합

$$C = A - B$$

또는 $C = A.\text{difference}(B)$



리스트 ↔ 세트



```
>>> list1 =[1,2,3,4,5,1,2,4 ]  
>>> len(set(list1))  
5
```

서로 다른 정수는 몇 개나 있을까?

```
>>> list1 =[1,2,3,4,5 ]  
>>> list2 =[3,4,5,6,7 ]  
>>> set(list1) & set(list2)  
{3, 4, 5}
```

공통적인 정수는 무엇일까?

세트 연산 정리

연산	설명
<code>set()</code>	공백 세트 생성
<code>set(seq)</code>	시퀀스에서 요소를 꺼내서 세트를 만든다.
<code>s1 = { e1, e2, e3, ... }</code>	초기값이 있는 세트는 중괄호로 만든다.
<code>len(s1)</code>	세트에 있는 요소의 수
<code>e in s1</code>	e가 세트 안에 있는지 여부
<code>add(e)</code>	e를 세트에 추가한다.
<code>remove(e)</code> <code>discard(e)</code>	e를 세트에서 삭제한다.
<code>clear()</code>	세트의 모든 요소를 삭제한다.
<code>s1.issubset(s2)</code>	부분 집합인지를 검사한다.
<code>s1 == s2</code> <code>s1 != s2</code>	동일한 집합인지를 검사한다.
<code>s1.union(s2)</code> <code>s1 s2</code>	합집합
<code>s1.intersection(s2)</code> <code>s1 & s2</code>	교집합
<code>s1.difference(s2)</code> <code>s1 - s2</code>	차집합

중간점검

1. 리스트와 세트의 차이점은 무엇인가?
2. 세트에 저장된 항목에 접근할 때 인덱스를 사용할 수 있는가?
3. 세트 A와 세트 B의 교집합을 계산하는 수식을 만들어보자.
4. 세트에 항목을 추가하는 함수는?



Lab: 문자열의 공통 문자

- 사용자로부터 2개의 문자열을 받아서 두 문자열의 공통 문자를 출력하는 프로그램을 작성해보자.

첫 번째 문자열: **Hello World!**

두 번째 문자열: **How are you?**

공통적인 글자: o H r e

Solution:

```
s1 = input("첫 번째 문자열:")
s2 = input("두 번째 문자열:")

list1 = list( set(s1) & set(s2) )    # 세트로 만들고 교집합 연산을 한다.

print("\n공통적인 글자:", end=" ")
for i in list1:
    print(i, end=" ")
```


Lab: 문자열의 공통 문자

- 중복되지 않은 단어의 개수 세기

입력 텍스트: **I have a dream that one day every valley shall be exalted and every hill and mountain shall be made low**

사용된 단어의 개수 = 17

{"be", "and", "shall", "low", "have", "made", "one", "exalted", "every", "mountain", "I", "that", "valley", "hill", "day", "a", "dream"}

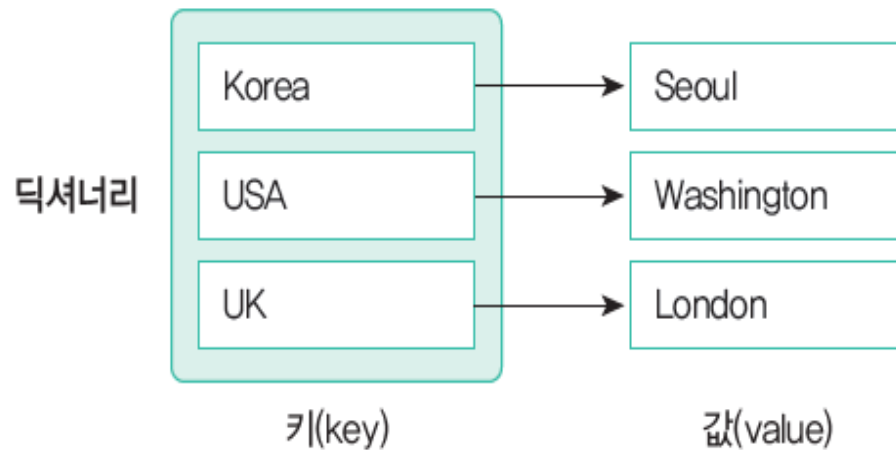
Solution:

```
txt = input("입력 텍스트: ")
words = txt.split(" ")
unique = set(words)          # 집합으로 만들면 자동적으로 중복을 제거한다.

print("사용된 단어의 개수= ", len(unique))
print(unique)
```

딕셔너리

- 딕셔너리(dictionary)도 값을 저장하는 자료구조이다.
- 하지만 딕셔너리에는 값(value)과 관련된 키(key)도 저장된다.



딕셔너리 생성

Syntax: 딕셔너리

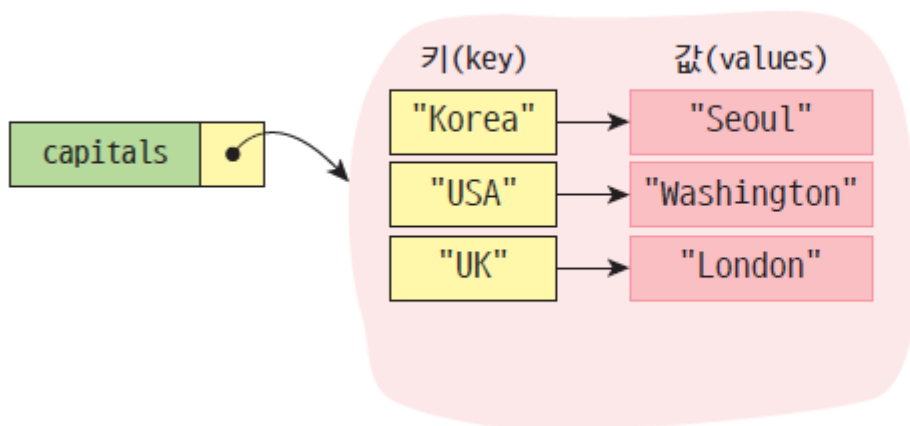
형식 딕셔너리_이름 = { 키1:값1, 키2:값2, 키3:값3, ... }

예 capitals = { } # ①
capitals = { "Korea": "Seoul", "USA": "Washington", "UK": "London" } # ②

공백 딕셔너리를 생성한다.

키

값



딕셔너리는 키와
값으로 이루어집니다.



항목 탐색하기

```
>>> capitals = {"Korea": "Seoul", "USA": "Washington", "UK": "London"}
>>> print( capitals["Korea"])
Seoul

>>> print( capitals["France"] )
...
KeyError: "France"

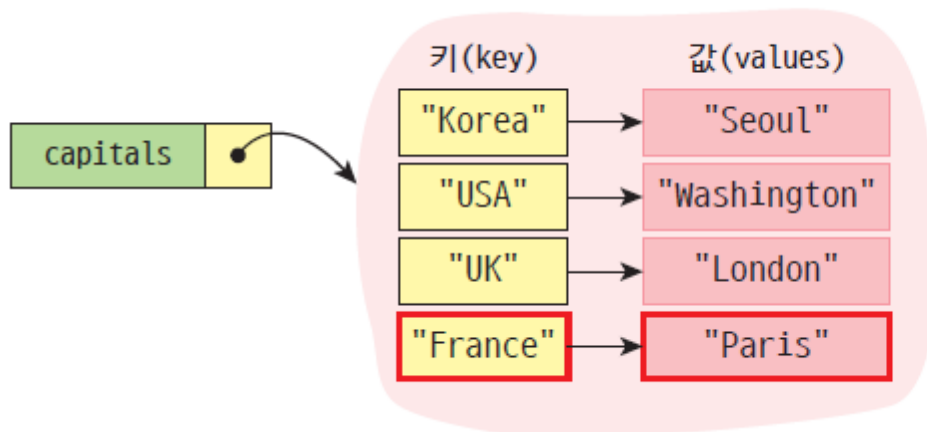
>>> print( capitals.get("France", "해당 키가 없습니다." ) )
해당 키가 없습니다.
```

프로그램이 오류로 중단되지 않게
하려면 이렇게 해야 함!

항목 추가하기

```
capitals = { }  
capitals["Korea"] = "Seoul"  
capitals["USA"] = "Washington"  
capitals["UK"] = "London"  
capitals["France"] = "Paris"  
print(capitals)
```

```
{'Korea': 'Seoul', 'USA': 'Washington', 'UK': 'London', 'France': 'Paris'}
```



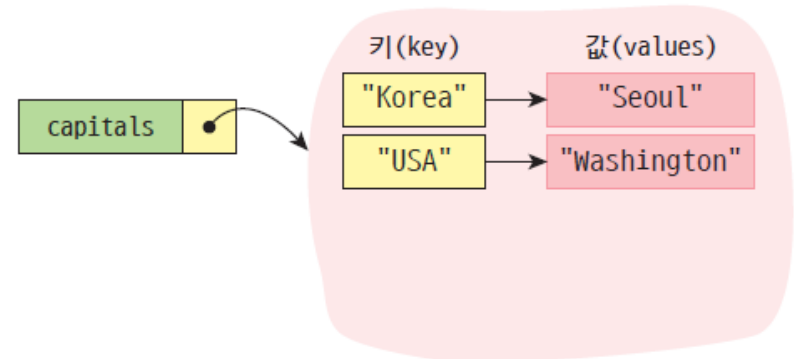
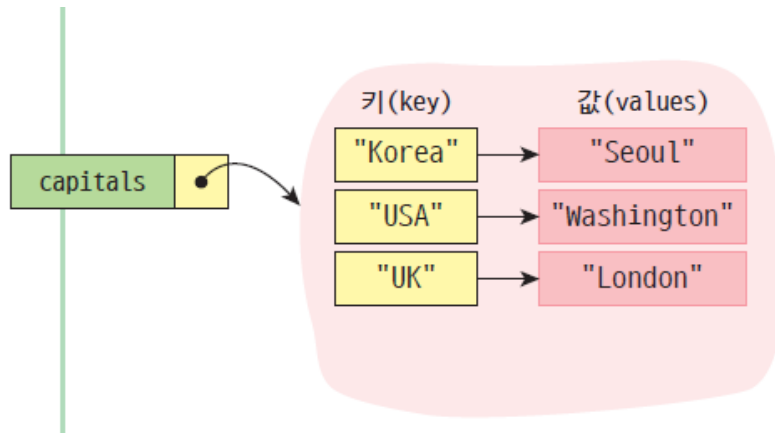
딕셔너리에 추가할 때는
[] 연산자를 사용하세요.



항목 삭제하기

만약 주어진 키를 가진 항목이 없으면 `KeyError` 예외가 발생한다

```
city = capitals.pop("UK")
```



```
if "UK" in capitals :      # 조건문 추가하여 예외 발생하지 않게  
    capitals.pop("UK")
```

항목 방문하기

```
capitals = {"Korea": "Seoul", "USA": "Washington", "UK": "London"}  
for key in capitals :  
    print( key, ":", capitals[key])
```

```
Korea : Seoul  
USA : Washington  
UK : London
```

```
capitals = {"Korea": "Seoul", "USA": "Washington", "UK": "London"}  
for key, value in capitals.items():  
    print( key, ":", value )
```

```
Korea : Seoul  
USA : Washington  
UK : London
```


기타 연산

```
capitals = {"Korea": "Seoul", "USA": "Washington", "UK": "London"}  
print( capitals.keys() )  
print( capitals.values() )
```

```
dict_keys(['Korea', 'USA', 'UK'])  
dict_values(['Seoul', 'Washington', 'London'])
```

```
for key in sorted( capitals.keys() ):  
    print(key, end=" ")
```

```
Korea UK USA
```

딕셔너리 함축

```
dic = { x : x**2 for x in values if x%2==0 }
```

딕셔너리

출력 수식

입력 리스트

조건식

```
values =[1,2,3,4,5,6]
```

```
dic = { x : x**2 for x in values if x%2==0 }
```

```
print(dic)
```

```
{2: 4, 4: 16, 6: 36}
```

딕셔너리 함축의 예

```
dic = { i:str(i) for i in [1,2,3,4,5]}  
print( dic )
```

```
{1: "1", 2: "2", 3: "3", 4: "4", 5: "5"}
```

```
fruits = ["apple", "orange", "banana"]
```

```
dic = { f:len(f) for f in fruits }  
print( dic )
```

```
{"apple": 5, "orange": 6, "banana": 6}
```

딕셔너리 메소드

연산	설명
<code>d = dict()</code>	공백 딕셔너리를 생성한다.
<code>d = {k₁:v₁, k₂:v₂, ..., k_n:v_n}</code>	초기값으로 딕셔너리를 생성한다.
<code>len(d)</code>	딕셔너리에 저장된 항목의 개수를 반환한다.
<code>k in d</code>	k가 딕셔너리 d 안에 있는지 여부를 반환한다.
<code>k not in d</code>	k가 딕셔너리 d 안에 없으면 True를 반환한다.
<code>d[key] = value</code>	딕셔너리에 키와 값을 저장한다.
<code>v = d[key]</code>	딕셔너리에서 key에 해당되는 값을 반환한다.
<code>d.get(key, default)</code>	주어진 키를 가지고 값을 찾는다. 만약 없으면 default 값이 반환된다.
<code>d.pop(key)</code>	항목을 삭제한다.
<code>d.values()</code>	딕셔너리 안의 모든 값의 시퀀스를 반환한다.
<code>d.keys()</code>	딕셔너리 안의 모든 키의 시퀀스를 반환한다.
<code>d.items()</code>	딕셔너리 안의 모든 (키, 값)을 반환한다.

중간점검

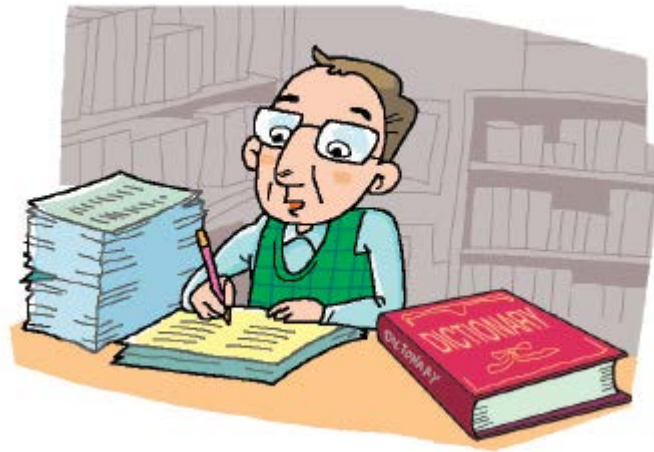
1. 공백 딕셔너리를 생성하는 명령문을 만들어보자.
2. 딕셔너리에 존재하는 모든 키를 방문하는 코드를 작성해보자.
3. 딕셔너리 d 에 (k, v) 를 저장하는 명령문을 만들어보자.



Lab: 영한 사전

단어를 입력하시오: **one**
하나

단어를 입력하시오: **two**
둘



Solution:

```
english_dict = {}
```

공백 딕셔너리를 생성한다.

```
english_dict["one"] = "하나"
```

딕셔너리에 단어와 의미를 추가한다.

```
english_dict["two"] = "둘"
```

```
english_dict["three"] = "셋"
```

```
word = input("단어를 입력하시오: ");
```

```
print (english_dict[word])
```

Lab: 학생 성적 처리

1. 연락처 추가
2. 연락처 삭제
3. 연락처 검색
4. 연락처 출력
5. 종료

메뉴 항목을 선택하시오: 1

이름: KIM

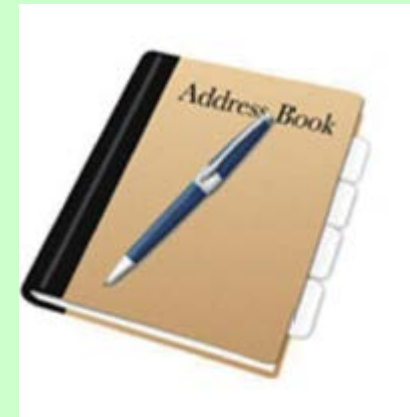
전화번호: 123-4567

1. 연락처 추가
2. 연락처 삭제
3. 연락처 검색
4. 연락처 출력
5. 종료

메뉴 항목을 선택하시오: 4

KIM 의 전화번호: 123-4567

...



Solution:

```
def main():  
    address_book = {}                                # 공백 딕셔너리를 생성한다.  
    while True :  
        user = display_menu();  
        if user == 1 :  
            name, number = get_contact()  
            address_book[name] = number              # name과 number를 추가한다.  
        elif user == 2 :  
            name, number = get_contact()  
            address_book.pop(name)                   # name을 키로 가지고 항목을 삭제한다.  
        elif user == 3 :  
            pass                                       # 도전 문제 참조  
        elif user == 4 :  
            for key in sorted(address_book):  
                print(key, "의 전화번호:", address_book[key])  
        else :  
            break
```

Solution:

이름과 전화번호를 입력받아서 반환한다.

```
def get_contact():
```

```
    name = input("이름: ")
```

```
    number = input("전화번호:")
```

```
    return name, number
```

튜플로 반환한다.

메뉴를 화면에 출력한다.

```
def display_menu() :
```

```
    print("1. 연락처 추가")
```

```
    print("2. 연락처 삭제")
```

```
    print("3. 연락처 검색")
```

```
    print("4. 연락처 출력")
```

```
    print("5. 종료")
```

```
    select = int(input("메뉴 항목을 선택하시오: "))
```

```
    return select
```

```
main()
```

Lab: 학생 성적 처리

```
score_dic = {  
    "Kim": [99, 83, 95],  
    "Lee": [68, 45, 78],  
    "Choi": [25, 56, 69]  
}
```

Kim 의 평균성적= 92.33333333333333

Lee 의 평균성적= 63.666666666666664

Choi 의 평균성적= 50.0

Solution:

```
score_dic = {  
    "Kim":[99,83,95],  
    "Lee":[68,45,78],  
    "Choi":[25,56,69]  
}  
  
for name, scores in score_dic.items():  
    print(name,"의 평균성적=", sum(scores) / len(scores))
```

Lab: 단어 카운터 만들기

```
text_data = "Create the highest, grandest vision possible for your life, because  
you become what you believe"
```

```
word_dic = {}                                # 단어들과 출현 횟수를 저장하는 딕셔너리를 생성
for w in text_data.split():                  # 텍스트를 단어들로 분리하여 반복
    if w in word_dic:                        # 단어가 이미 딕셔너리에 있으면
        word_dic[w] += 1                    # 출현 횟수를 1 증가
    else:                                    # 처음 나온 단어이면 1로 초기화하여 등록
        word_dic[w] = 1
for w, count in sorted(word_dic.items()):    # 키와 값을 정렬하여 반복 처리
    print(w, "의 등장횟수=", count)
```

```
Create 의 등장횟수= 1
because 의 등장횟수= 1
become 의 등장횟수= 1
believe 의 등장횟수= 1
...
```

Solution (2):

```
from collections import Counter
```

```
text_data = "Create the highest, grandest vision possible for your life, because  
you become what you believe"
```

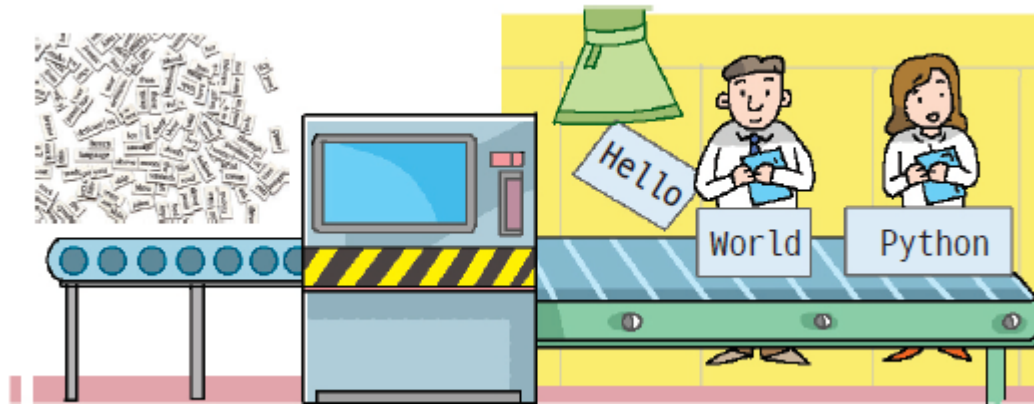
```
a = Counter( text_data.split() )
```

```
print(a)
```

```
Counter({'you': 2, 'Create': 1, 'the': 1, 'highest': 1, 'grandest': 1, 'vision': 1,  
'possible': 1, 'for': 1, 'your': 1, 'life': 1, 'because': 1, 'become': 1, 'what': 1,  
'believe': 1})
```

문자열

- 파이썬의 문자열 함수들만 이용하여도 어느 정도 데이터를 처리할 수 있지만,
- 우리가 사용하고 있는 개발 환경인 아나콘다는 **BeautifulSoup, csv, json, nltk**와 같은 우수한 모듈을 제공하기 때문에 우리는 쉽게 텍스트를 처리하고 분석할 수 있다.



문자열 내장 함수

함수	설명
chr()	정수를 문자로 변환
ord()	문자를 정수로 변환
len()	문자열의 길이를 반환
str()	객체의 문자열 표현을 반환

```
>>> ord("a")  
97  
>>> ord("가")  
44032
```

```
>>> chr(97)  
"a"  
>>> chr(44032)  
"가"
```


문자열의 인덱싱

- 문자열도 크게 보면 시퀀스(sequence)라는 자료 구조에 속한다.

0	1	2	3	4	5	[6:10]				10	11
M	o	n	t	y		P	y	t	h	o	n
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
[-12:-7]											

```
>>> s = "Monty Python"
```

```
>>> s[0]
```

```
"M"
```

```
>>> s[-1]
```

```
"n"
```

문자열 슬라이싱

- 슬라이싱이란 문자열의 일부를 잘라서 부분 문자열을 만드는 연산으로 파이썬의 두드러진 장점 중의 하나이다.

```
>>> s = "Monty Python"
```

```
>>> s[6:10]
```

```
"Pyth"
```



슬라이싱은 문자열의 일부를 추출하는 기능입니다.



문자열 슬라이싱

```
>>> s = "Monty Python"
>>> s[:2]
"Mo"
>>> s[4:]
"y Python"
```

```
>>> s = "Monty Python"
>>> s[:2] + s[2:]
"Monty Python"
>>> s[:4] + s[4:]
"Monty Python"
```

```
>>> s = "Monty Python"
>>> s[:]
"Monty Python"
```

문자열 슬라이싱

```
>>> message="see you at noon"
>>> low = message[:5]
>>> high = message[5:]
>>> low
"see y"
>>> high
"ou at noon"
```

```
>>> reg= "980326"
>>> print(reg[0:2]+"년")
98년
>>> print(reg[2:4]+"월")
03월
>>> print(reg[4:6]+"일")
26일
```

문자열은 불변 객체

```
>>> word = "abcdef"
```

```
>>> word[0]="A'
```

```
...
```

```
TypeError: "str" object does not support item assignment
```

```
>>> word = "abcdef"
```

```
>>> word = "A" + word[1:]
```

```
>>> word
```

```
"Aabcdef"
```

문자열 비교

- 마찬가지로 ==, !=, <, > 연산자를 문자열에도 적용할 수 있다.

```
a = input("문자열을 입력하시오: ")
b = input("문자열을 입력하시오: ")
if ( a < b ):
    print(a, "가 앞에 있음")
else:
    print(b, "가 앞에 있음")
```

```
문자열을 입력하시오: apple
문자열을 입력하시오: orange
apple 가 앞에 있음
```

문자열 출력하기

```
x = 25  
y = 98  
prod = x * y  
print(x, "과", y, "의 곱은", prod)
```

25 과 98 의 곱은 2450

```
x = 25  
y = 98  
prod = x * y  
print(f"{x}과 {y}의 곱은 {prod}")
```

f-문자열

25과 98의 곱은 2450

Lab: 회문 검사하기

- 회문(**palindrome**)은 앞으로 읽으나 뒤로 읽으나 동일한 문장이다.
- 예를 들어서 "mom", "civic", "dad" 등이 회문의 예이다.
- 사용자로부터 문자열을 입력받고 회문인지를 검사하는 프로그램을 작성하여 보자.

문자열을 입력하시오: **dad**

회문입니다.

Solution:

```
s = input("문자열을 입력하시오: ")

s1 = s[::-1] # 문자열을 거꾸로 만든다.

if( s == s1 ):
    print("회문입니다.")
else:
    print("회문이 아닙니다.")
```

문자열 메소드: 대소문자 변환하기

```
>>> s = "i am a student."
```

```
>>> s.capitalize()
```

s 문자열이 수정되는가??

```
"I am a student."
```

```
>>> s = "Breakfast At Tiffany"
```

```
>>> s.lower()
```

```
"breakfast at tiffany"
```

```
>>> s.upper()
```

```
"BREAKFAST AT TIFFANY"
```

문자열 메소드: 찾기 및 바꾸기

```
s = input("파이썬 소스 파일 이름을 입력하시오: ")

if s.endswith(".py"):
    print("올바른 파일 이름입니다")
else :
    print("올바른 파일 이름이 아닙니다.")
```

파이썬 소스 파일 이름을 입력하시오: **aaa.py**
올바른 파일 이름입니다

문자열 메소드: 찾기 및 바꾸기

```
>>> s = "www.naver.com"
>>> s.replace("com", "co.kr")
"www.naver.co.kr"
```

문자열 내부에서 특정문자가 어디에 위치하는지 찾을 때 사용한다.
find()함수는 왼쪽부터 찾고, **rfind()**함수는 오른쪽부터 찾는다.

```
>>> s = "www.naver.co.kr"
>>> s.find(".kr")
12
```

```
>>> s = "Let it be, let it be, let it be"
>>> s.rfind("let")
22
```

```
>>> s = "www.naver.co.kr"
>>> s.count(".")
3
```

문자열 메소드: 문자 분류

```
>>> "ABCAbc".isalpha()
```

```
True
```

```
>>> "123".isdigit()
```

```
True
```

```
>>> "abc".islower()
```

```
True
```

문자열 메소드: 공백 제거

```
>>> s = " Hello, World! "
```

```
>>> s.strip()
```

```
"Hello, World!"
```

```
>>> s = "#####this is example#####"
>>> s.strip("#")
```

```
"this is example"
```

```
>>> s = "#####this is example#####"
>>> s.lstrip("#")
```

```
"this is example#####"
>>> s.rstrip("#")
```

```
"#####this is example"
```

문자열 메소드: 문자열 분해하기

```
>>> s = "Welcome to Python"
```

```
>>> s.split()
```

```
["Welcome", "to", "Python"]
```

```
>>> s = "Hello, World!"
```

```
>>> s.split(",")
```

```
["Hello", " World!"]
```

```
>>> s = "Hello, World!"
```

```
>>> s.split(", ")
```

```
["Hello", "World!"]
```

```
>>> list("Hello, World!")
```

```
['H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!']
```

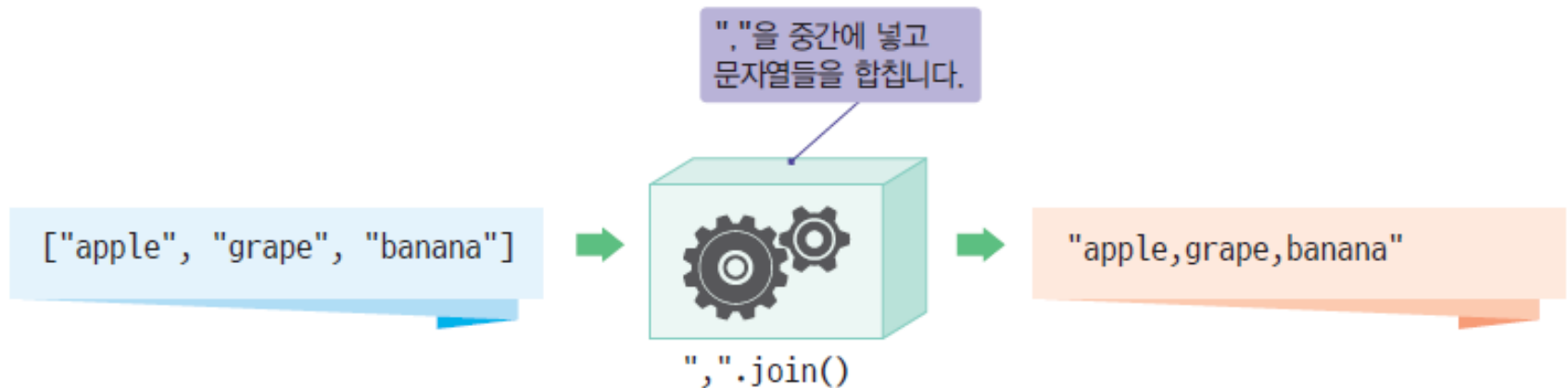
문자열 메소드: 문자열 결합하기

```
>>> ", ".join(["apple", "grape", "banana"])
```

```
"apple,grape,banana"
```

```
>>> "-".join("010.1234.5678".split("."))
```

```
"010-1234-5678"
```



Lab: 머리 글자어 만들기

- 머리 글자어(acronym)은 NATO(North Atlantic Treaty Organization)처럼 각 단어의 첫 글자를 모아서 만든 문자열이다.
- 사용자가 문장을 입력하면 해당되는 머리 글자어를 출력하는 프로그램을 작성하여 보자.

문자열을 입력하시오: **North Atlantic Treaty Organization**

NATO

Solution:

```
phrase = input("문자열을 입력하시오: ")

acronym = ""

# 대문자로 만든 후에 단어들로 분리한다.
for word in phrase.upper().split():
    acronym += word[0]      # 단어를 첫 글자만을 acronym에 추가한다.

print( acronym )
```

Lab: 이메일 주소 분석

- 이메일 주소에서 아이디와 도메인을 구분하는 프로그램을 작성하여 보자.

이메일 주소를 입력하시오: **aaa@google.com**

aaa@google.com

아이디:aaa

도메인:google.com

Solution:

```
address=input("이메일 주소를 입력하시오: ")
```

```
(id, domain) = address.split("@")
```

```
print(address)
```

```
print("아이디:"+id)
```

```
print("도메인:"+domain)
```

Lab: 문자열 분석

- 문자열 안에 있는 문자의 개수, 숫자의 개수, 공백의 개수를 계산하는 프로그램을 작성하여 보자.

문자열을 입력하시오: **A picture is worth a thousand words.**

```
{"digits": 0, "spaces": 6, "alphas": 29}
```

Solution:

```
sentence = input("문자열을 입력하시오: ")

table = {"alphas":0,"digits":0,"spaces":0 }      # 딕셔너리 자료구조

for i in sentence:
    if i.isalpha():
        table["alphas"]+=1
    if i.isdigit():
        table["digits"]+=1
    if i.isspace():
        table["spaces"]+=1

print(table)
```

Lab: 트위터 메시지 처리

- 일반적으로 부정적인 감정은 긍정적인 것보다 적은 양의 단어를 포함한다고 한다. 트윗에서 단어의 개수를 추출하여서 발신자의 감정을 판단해보자.

```
t = "Python is very easy and powerful!"
```

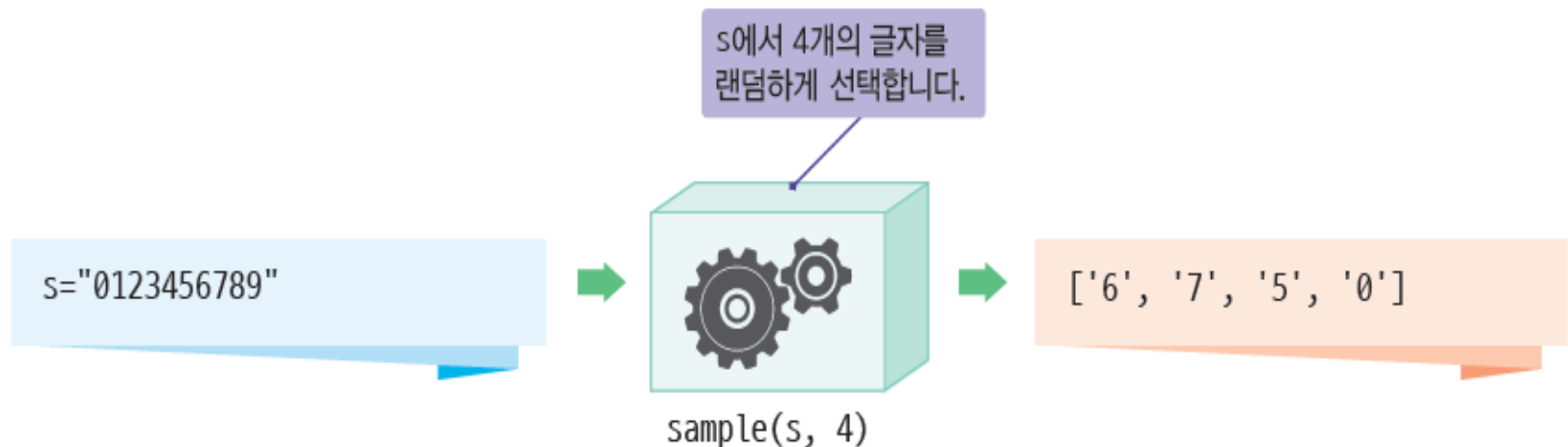
```
length = len(t.split(" "))
```

```
print(length)
```

Lab: OTP 발생 프로그램

- 일회용 암호 (OTP) 프로그램을 작성해보자.

3482



Solution:

```
import random

s = "0123456789"          # 대상 문자열
passlen = 4               # 패스워드 길이

# sample()은 주어진 개수만큼의 글자를 문자열 s에서 임의로 선택한다.
# join()은 이들 글자들을 결합한다.

p = "".join(random.sample( s, passlen ))
print(p)
```

이번 장에서 배운 것

- 튜플은 변경 불가능한 항목들을 모아둔 곳이다.
- ()을 이용하여 공백 튜플을 만들 수 있다.
- 딕셔너리는 키와 값으로 이루어진다.
- 딕셔너리에서 [] 연산자를 사용하여 키와 관련된 값을 액세스할 수 있다.
- 딕셔너리에서 pop 메소드를 사용하여 항목을 제거한다.
- 세트는 고유한 값들을 저장한다.
- 세트는 set() 함수를 사용하여 생성할 수 있다.
- 세트의 add() 메소드를 사용하여 새 요소를 추가할 수 있다.

