

7장 파이썬 자료구조 II (튜플, 딕셔너리, 세트, 문자열)

학습 목표

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



이번장에서 만들 프로그램

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

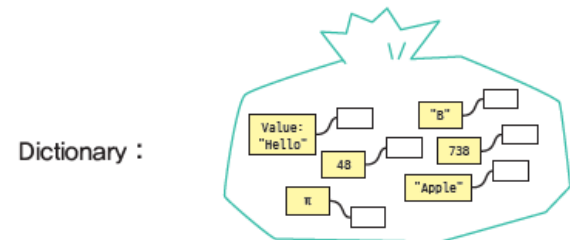
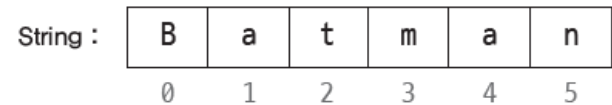
메뉴 항목을 선택하시오: 4

KIM 의 전화번호: 123-4567

...

자료구조

- 자료구조(data structure) = 데이터 구조 : 자료들을 저장하는 구조
- 가장 기초적인 자료구조 -> 시퀀스(sequence)
 - ▣ 리스트가 대표적 예
 - ▣ 요소(element)로 구성.
 - ▣ 인덱스(index)가 있다.
 - ▣ 파이썬에는 6개의 내장 시퀀스가 있다. - str, bytes, bytearray, list, tuple, range
 - ▣ 동일한 연산을 지원 : 인덱싱(indexing), 슬라이싱(slicing), 덧셈 연산(adding), 곱셈 연산(multiplying)
 - ▣ 공통적인 내장함수 적용 : len(), max(), min() 등



추가점검 종교

1. 리스트는 시퀀스에 속하는가?
2. 시퀀스의 특징에는 어떤 것들이 있는가?



튜플(tuple)

- 리스트와 유사. 튜플은 변경이 불가능.

Syntax: 튜플

형식 튜플_이름 = (항목1, 항목2, ...)

공백 튜플을 생성한다.

예 fruits = ()

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

초기값을 가진 튜플을 생성한다.

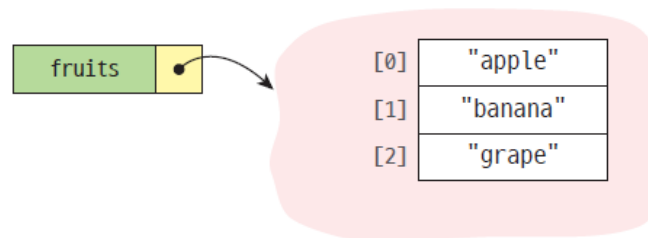
result = fruits[1]

인덱스를 사용하여 요소에 접근한다.

튜플의 이름

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

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



튜플(tuple)

p311_tuple1.py

- 주의할 점 – 요소가 하나 뿐인 튜플을 만들 때는 요소의 끝에 반드시 쉼표(,)를 추가하여야 한다.

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

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

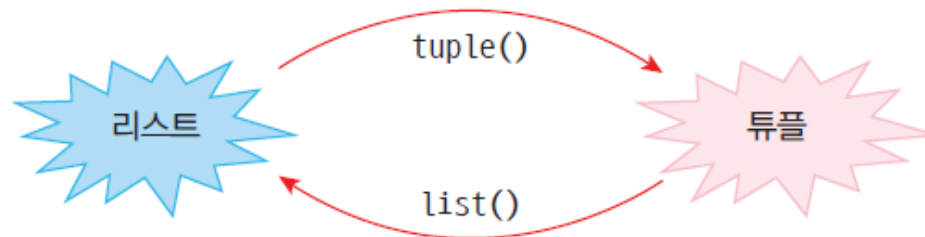
>>> fruits[1] = "pear"             # 오류 발생!
TypeError: "tuple" object does not support item assignment
```

튜플 <.-> 리스트

p311_tuple1.py

```
>>> 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.

- 튜플을 수정, 삭제하는 것은 불가능하지만 튜플을 합치는 것은 가능

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

다른 튜플에 합치는 것은 가능

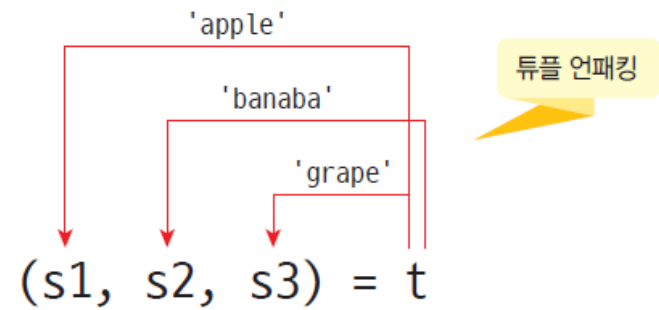
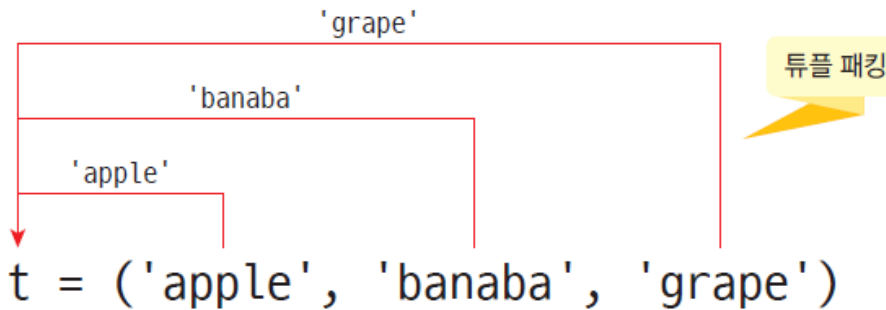
```
>>> numbers = [10, 20, 30]  
>>> numbers += (40, 50)  
>>> numbers  
[10, 20, 30, 40, 50]
```

리스트에 튜플을 합치는 것은 가능

- 튜플 안의 리스트 변경이나 튜플을 리스트에 추가하는 것은 허용
-> 즉 튜플의 내용 변경만 아니면 모든 것이 허용

튜플 패킹과 언패킹

p314_packing_unpacking.py



```
>>> x = ("apple", "banana", "grape")  
>>> (s1, s2, s3) = x
```

튜플 패킹
튜플 언패킹

```
>>> student = ("Kim", [3.1, 3.6, 4.0, 0.0])  
>>> name, grades = student  
>>> name  
Kim  
>>> grades  
[3.1, 3.6, 4.0, 0.0]
```

서로 다른 자료형에 대해서도
패킹과 언패킹이 가능하다

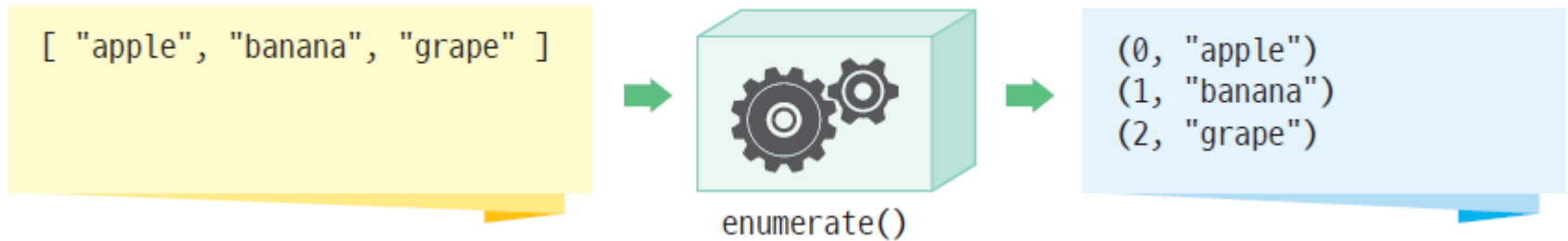
```
n1 = 10  
n2 = 90  
n1, n2 = (n2, n1)
```

튜플을 이용하여 데이터의 값을 바꿀 수 있다.

(90 10)

enumerate() 사용하기

- 반복 가능한 객체(리스트나 튜플)를 받아서, 각 요소에 대해 (인덱스, 값) 형태의 튜플을 반환



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

p315.py

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

튜플의 장점

- 튜플을 통하여 반복하는 것이 리스트 반복보다 빠르다. 튜플은 변경 불가능 객체이기 때문이다. 그만큼 처리 코드가 간단해진다.
- 변경 불가능한 요소들로 이루어진 튜플은 딕셔너리 키가 될 수 있다.
- 만약 다른 개발자가 특정 데이터를 변경하는 것을 금지하려면 튜플을 사용하면 된다.

	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.

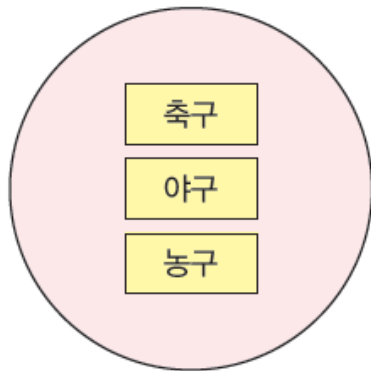
추가점

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

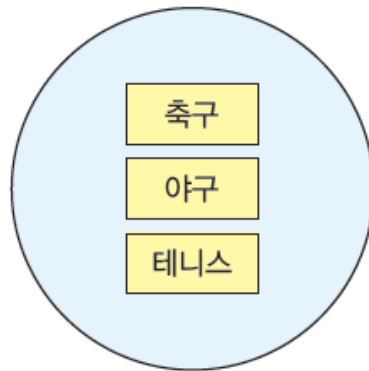


세트

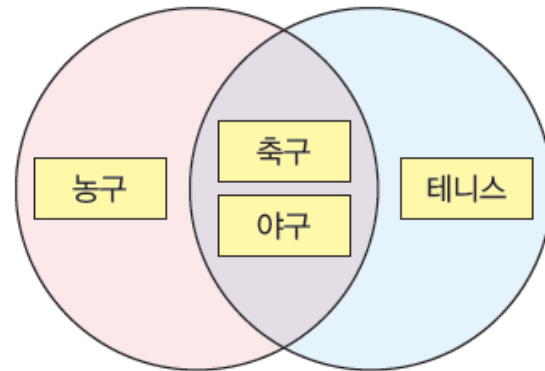
- 세트(set)는 우리가 수학에서 배웠던 집합. 고유한 값들을 저장하는 자료구조. 중복이 없다.
- 리스트와는 다르게 세트의 요소는 특정 순서로 저장되지 않으며 위치별로 액세스할 수 없다.
- 세트는 특정 순서를 유지할 필요가 없으므로 세트 연산은 리스트 연산보다 훨씬 빠르다.



세트 #1



세트 #2



세트 #1 ∩ 세트 #2

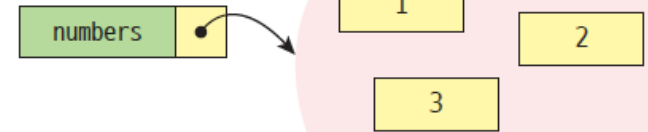
세트 생성하기

Syntax: 세트

p317_set.py

형식 세트_이름 = { 항목1, 항목2, 항목3, ... }

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

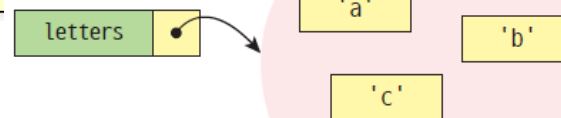


```
numbers = set([1,2,3,1,2,3])  
print(numbers)
```

리스트로부터 세트 생성도 가능.
set() 함수 사용

{ 1, 2, 3 }

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



문자열을 분해하여
세트로 만들 수 있어요.



세트의 연산

- all(), any(), enumerate(), len(), max(), min(), sorted(), sum() 사용 가능
- all() – 세트의 모든 요소가 True인 경우에 True
- any() – 하나의 요소라도 True이면 True

p318.py

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

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

어떤 항목이 세트 안에 있는지를
검사하려면 in 연산자 사용

집합 안에 apple이 있습니다.

세트의 연산

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

for 반복문을 이용하여 각 항목에 접근할 수 있다. 순서없이 출력

grape banana apple

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

정렬을 한 후 출력한다.

apple banana grape

요소 추가하고 삭제하기

- 한 개 추가 – add()
- 삭제 – remove(), discard(), clear()
- 여러 개 추가 – update()

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

p319.py

```
fruits.add("kiwi")          #add() 메소드를 이용하여 요소를 추가한다
print(fruits)
```

```
fruits.remove("kiwi")       #remove() 메소드로 요소를 삭제한다.
print(fruits)
```

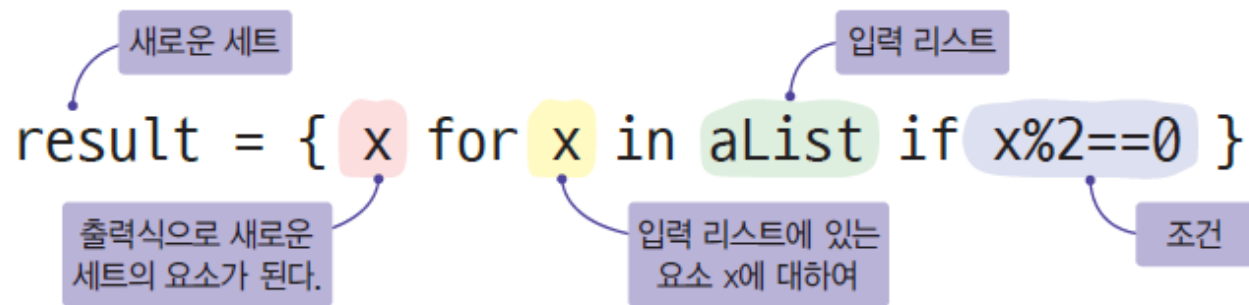
```
#fruits.remove("mango")     #remove() 메소드는 삭제하고자 하는 요소가 없으면 오류를 생성한다.
fruits.discard("mango")     #discard() 메소드는 삭제하고자 하는 요소가 없으면 그냥 넘어간다.
```

```
....
```

```
....
```

세트 합축 연산

- 리스트 안의 숫자가 짝수이면 세트에 추가하는 함축 코드 예



p320_SetCompre.py

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

세트이기 때문에 동일한 숫자는
딱 한번만 추가된다.

🔍 실행결과

{2, 4}

부분 집합 연산

- 부분집합인지를 검사하려면 < 연산자나 `issubset()` 메소드를 사용

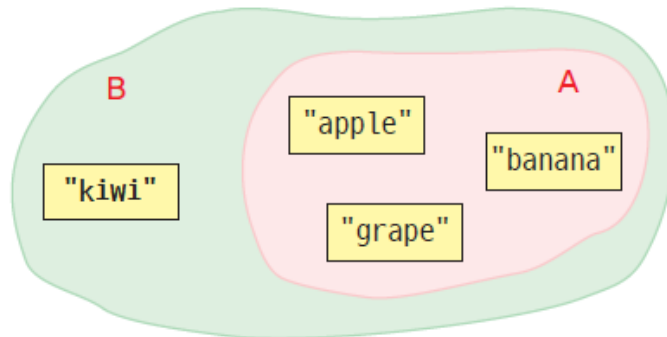
```
A = {"apple", "banana", "grape"}  
B = {"apple", "banana", "grape", "kiwi"}
```

p320_subset.py

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

실행결과

A는 B의 부분 집합입니다.



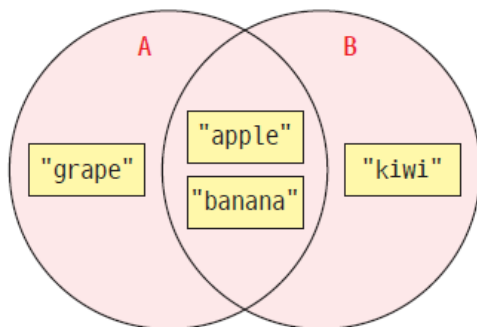
부분 집합은 < 으로
검사할 수 있어요!



교집합, 합집합, 차집합

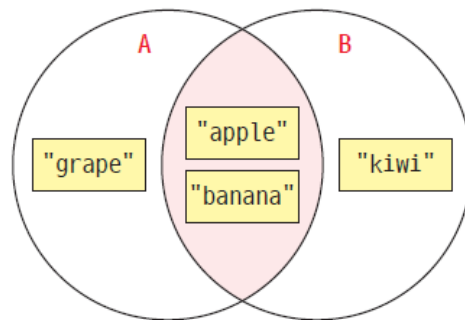
$$C = A \cup B$$

또는 $C = A.union(B)$



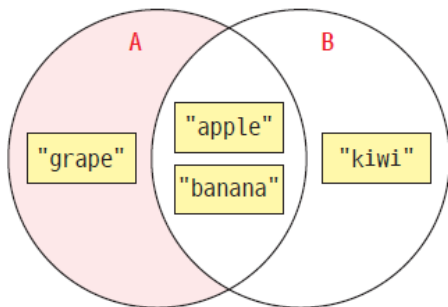
$$C = A \cap B$$

또는 $C = A.intersection(B)$



$$C = A - B$$

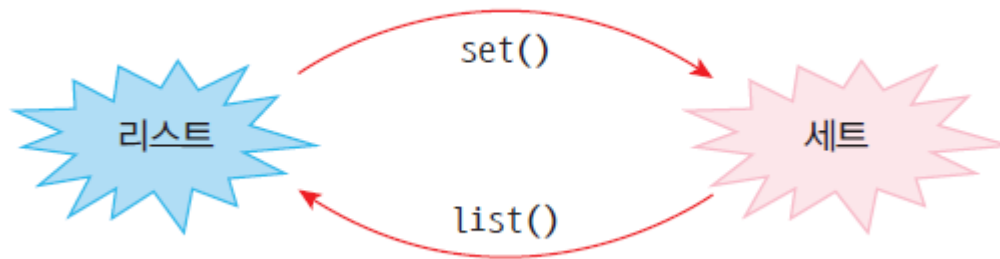
또는 $C = A.difference(B)$



p321_union.py

리스트 <-> 세트

- 리스트에 set() 함수를 적용하여 세트를 만들면 쉽게 풀 수 있는 문제들이 많다.



p322_set.py

```
>>> 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 들어있는 숫자는 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

정가점거 공간점거

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
```

```
s1=input("첫 번째 문자열:")  
s2=input("두 번째 문자열:")  
  
list1 = list( set(s1) & set(s2) )  
  
print("\n공통적인 글자:", end=" ")  
for i in list1:  
    print(i, end=" ")
```

p324_str_ex1.py

세트로 만들고 교집합 연산을 한다.

- 입력한 문자열 단위가 아니라 문자 단위로 처리하는데 -> 문자열 단위로 하려면 ? -> 아이디어는 다음 LAB(p.324)에서 얻을 수 있다.

도전문제

- 2개의 문자열을 비교했을 때, 공통적인 글자가 많으면 표절이라고 의심할 수도 있다. 공통적인 문자의 비율이 전체 문자의 **70%** 이상이 되면 “표절 의심 ” 이라고 출력해보자.

Lab: 중복되지 않은 단어의 개수 세기

- 단어를 얼마나 다양하게 사용하여 텍스트를 작성하였는지를 계산하는 프로그램을 작성해보자.

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"}

```
txt = input("입력 텍스트: ")
```

p325_str_ex2.py

```
words = txt.split(" ")
```

```
unique = set(words)
```

집합으로 만들면 자동적으로 중복을 제거한다.

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

```
print(unique)
```

도전문제

- 학생의 작문 리포트를 입력으로 받아서 중복되지 않은 단어가 많으면 점수를 높게 매기는 프로그램으로 변경해보자. 예를 들어서 중복되지 않은 단어의 개수가 전체 단어 개수의 **50%**가 넘으면 학점 **A**를 부여한다.

딕셔너리

- 딕셔너리(dictionary) : 사전처럼 키(key)와 값(value)으로 구성.
- 값은 어떤 유형이든지 상관없지만 키는 반드시 불변이고 유일해야. 학번처럼.

Syntax: 딕셔너리

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

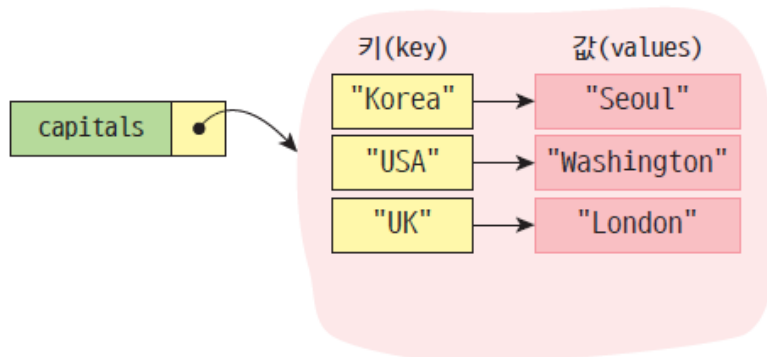
예 capitals = { } # ①
capitals = { "Korea": "Seoul" , "USA": "Washington", "UK": "London" } # ②

공백 딕셔너리를 생성한다.

키

값

p326.py



딕셔너리는 키와 값으로 이루어집니다.



항목 탐색하기

- 리스트에서는 인덱스를 가지고 항목을 찾을 수 있지만 딕셔너리에서는 키가 있어야 값을 찾을 수 있다.

p326.py

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

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

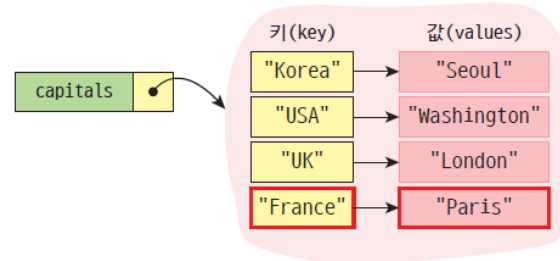
>>> print( capitals.get("France", "해당 키가 없습니다." ) )
해당 키가 없습니다.
```

키를 찾을 수 없어 프로그램이 오류로
종단되지 않게 하려면 이렇게 해야 함!

함수 추가하기

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

p328.py



딕셔너리에 추가할 때는
[] 연산자를 사용하세요.



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

```
capitals = {"Korea": "Seoul", "USA": "Washington", "UK": "London"}  
capitals.update({"France": "Paris", "Germany": "Berlin"})  
print(capitals)
```

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

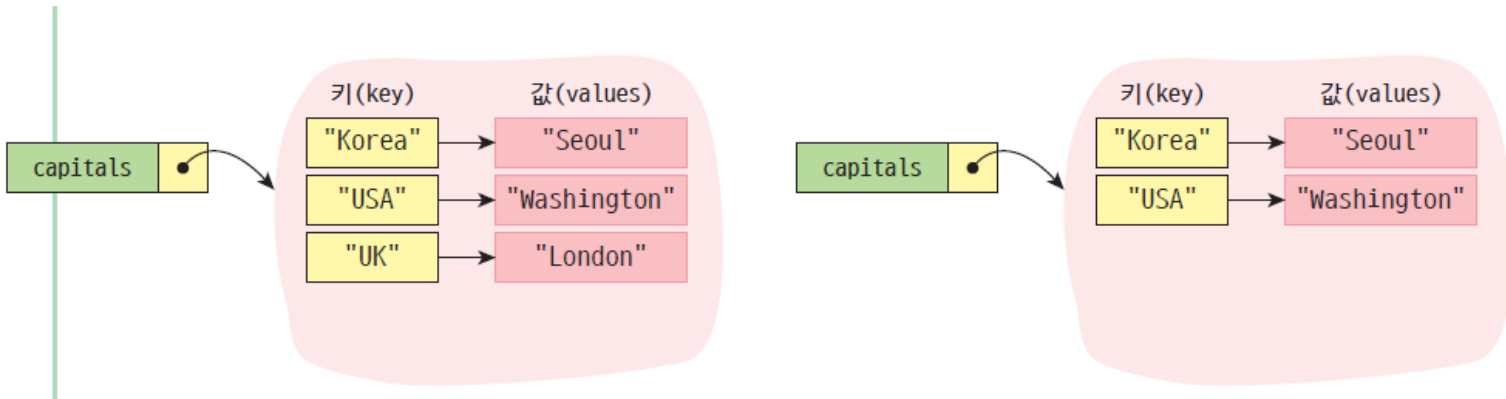
다른 딕셔너리 전체를 현재의 딕셔너리에
추가하려면 update() 사용

항목 삭제하기

p329.py

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

특정항목을 삭제하려면 pop() 사용.
삭제되는 항목은 반환된다.



```
if "UK" in capitals :  
    capitals.pop("UK")
```

주어진 키를 가진 항목이 없을 때 에러
발생을 방지하려면 먼저 검사한 후.

항목 삭제하기

```
capitals.clear()
if len(capitals)==0 :
    print("딕셔너리에 항목이 없음")
else:
    print("딕셔너리가 비어 있음")
```

딕셔너리에 저장된 모든 키-값 쌍을
삭제하려면 clear() 사용

하모 바꿈하기

p329.py

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

키와 값 쌍을 출력 - 딕셔너리[키]
형식 사용

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

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

키와 값 쌍을 출력 - items()
메소드 사용

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

딕셔너리 합성

- values 리스트에서 값을 하나씩 꺼내서 x에 대입하고 x가 짝수이면 x**2를 계산하여 x와 묶어서 딕셔너리를 생성하는 코드 예

```
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]
```

p330.py

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

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

```
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}
```

```
dic1 = {"One":1, "Two":2, "Three":3 }  
dic2 = { n : w for w , n in dic1.items()}  
print( dic2 )
```

```
{1: 'One', 2: 'Two', 3: 'Three'}
```

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.

추가점검

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

p332.py

단어를 입력하시오: one
하나

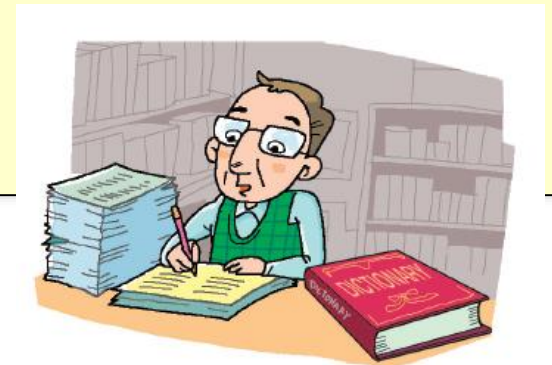
단어를 입력하시오: two
둘

```
english_dict = {} # 공백 딕셔너리를 생성한다.

english_dict["one"]="하나"
english_dict["two"]="둘"
english_dict["three"]="셋"

word = input("단어를 입력하시오: ");
print (english_dict[word])
```

딕셔너리에 단어와 의미를 추가한다.



Lab: 주소록 작성

p333.py

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

메뉴 항목을 선택하시오: 1

이름: KIM

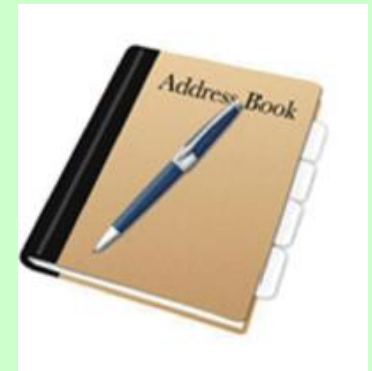
전화번호: 123-4567

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

p334.py

- 3가지 과목(국어, 영어, 수학)에서의 각 학생의 성적을 딕셔너리에 저장해보자. 학생이 이름(문자열)이 키가 되고 해당 학생의 성적이 포함된 정수 리스트가 값이 된다. 딕셔너리에서 각 학생의 성적을 꺼내서 평균 성적을 계산해서 출력해보자.

```
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))
```

```
Kim 의 평균성적= 92.33333333333333  
Lee 의 평균성적= 63.66666666666666  
Choi 의 평균성적= 50.0
```

Lab: 단어 카운터 만들기

p335.py

- 단어가 나타나는 횟수인 빈도를 계산하는 기법은 종종 텍스트를 분석하는데 사용된다. 출판물의 단어 빈도를 다른 출판물과 비교하면 유사점을 분석할 수 있다.
- 두가지 코드
 - (1) 직접 코딩
 - (2) 이미 구현된 클래스 **Counter** 사용 – 분석결과는 딕셔너리로 생성

Solution:

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

```
word_dic = {}
for w in text_data.split():
    if w in word_dic:
        word_dic[w] += 1
    else:
        word_dic[w] = 1

for w, count in sorted(word_dic.items()):
    print(w, "의 등장횟수=", count)
```

단어들과 출현 횟수를 저장하는 딕셔너리를 생성
텍스트를 단어들로 분리하여 반복한다.
단어가 이미 딕셔너리에 있으면
출현 횟수를 1 증가한다.
처음 나온 단어이면 1로 초기화한다.
키와 값을 정렬하여 반복 처리한다.

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

Solution:

```
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

- 파이썬은 다른 언어에 비해 문자열 처리 기능이 탁월한 언어
- 수많은 원천 데이터들은 문자열 형태로 저장되어 있다. 따라서 문자열을 읽어서 처리하는 것은 아주 중요.
- 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")   #ASCII 코드값
97
>>> ord("가")
44032
```

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


인덱싱

- 문자열에서 개별 문자들을 추출

0	1	2	3	4	5	6	7	8	9	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

[6:10]

[-12:-7]

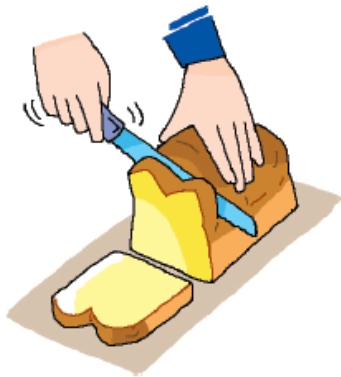
```
>>> s = "Monty Python"
>>> s[0]
"M"
>>> s[-1]
"n"
```

슬라이싱

- 슬라이싱(slicing)이란 문자열의 일부를 잘라서 서브 문자열을 만드는 연산

```
>>> s = "Monty Python"  
>>> s[6:10]  
"Pyth"
```

s[6]부터 s[9]까지의 문자를 선택



슬라이싱은 문자열의 일부를
추출하는 기능입니다.



```
>>> 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"
```

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
```

슬라이드

- **주의 !** - 문자열은 변경 불가능한 객체이다. 따라서 문자열의 일부 글자를 바꾸려고 하면 오류가 발생한다.

```
>>> word = "abcdef"
>>> word[0]="A'
...
TypeError: "str" object does not support item assignment
```

- 문자열 변경이 필요하면 연산을 통해 문자열을 다시 만드는 방식으로 하면 된다.

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

문자열 비교하기

- ==, !=, <, > 연산자 사용

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

p341.py

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

문자열 출력하기

p342.py

```
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

정가점거 공간점거

1. 문자열에 포함된 글자들의 코드값을 얻으려면 어떤 함수를 호출하는가?
2. 문자열의 맨 끝에 있는 글자를 추출하는 명령문을 작성해보자.
3. 문자열 **A**와 문자열 **B**의 순서를 비교하려면 어떤 명령문을 사용하는가?



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.
```

```
s = input("문자열을 입력하시오: ")

s1 = s[::-1]                                # 문자열을 거꾸로 만든다.

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

문자열 메소드: 대소문자 변환하기

p344.py

- `capitalize()` : 첫 글자만 대문자로 변환
- `lower()` : 소문자로 변환
- `upper()` : 대문자로 변환

```
>>> s = "i am a student."  
>>> s.capitalize()  
"I am a student."
```

```
>>> s = "Breakfast At Tiffany"  
>>> s.lower()  
"breakfast at tiffany"  
>>> s.upper()  
"BREAKFAST AT TIFFANY"
```

문자열 메소드: 찾기 및 바꾸기

- `startswith(s)` : 문자열 `s`로 시작되는 문자열이면 `True` 반환
- `endswith(s)` : 문자열 `s`로 종료되는 문자열이면 `True` 반환
- `replace()` : 부분 문자열을 다른 문자열로 변환
- `find()` : 문자열에서 부분 문자열을 찾아 인덱스를 반환
- `s.rfind(<sub>[, <start>[, <end>]]]` : 끝에서 시작하여 지정된 문자열을 대상 문자열 안에서 검색
- `count()` : 부분 문자열이 등장하는 횟수를 반환

p344.py

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

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

문자열 메소드: 찾기 및 바꾸기

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

```
>>> 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

- `s.isalpha()` : 문자열 `s` 안의 모든 문자가 알파벳(글자)이면 **True** 반환
- `s.isdigit()` : 문자열 `s` 가 숫자로 구성되어 있으면 **True** 반환
- `s.islower()` : 모든 알파벳 문자가 소문자로 되어 있으면 **True** 반환
- `s.isupper()` : 모든 알파벳 문자가 대문자로 되어 있으면 **True** 반환

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

```
True
```

p344.py

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

```
True
```

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

```
True
```

문자열 메소드: 공백문자 제거

- `strip()` : 문자열의 첫 부분과 끝부분에서 공백문자를 제거
- `strip('x')` : 문자열 앞뒤의 특정한 문자 'x'를 삭제
- `rstrip()`, `lstrip()` : 문자열의 오른쪽 끝과 왼쪽 끝에서 원치 않는 문자 삭제

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

p344.py

```
>>> 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

- `split()` : 주어진 분리자를 이용하여 문자열을 토큰들의 리스트로 반환

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

p344.py

```
>>> 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", "!"]
```

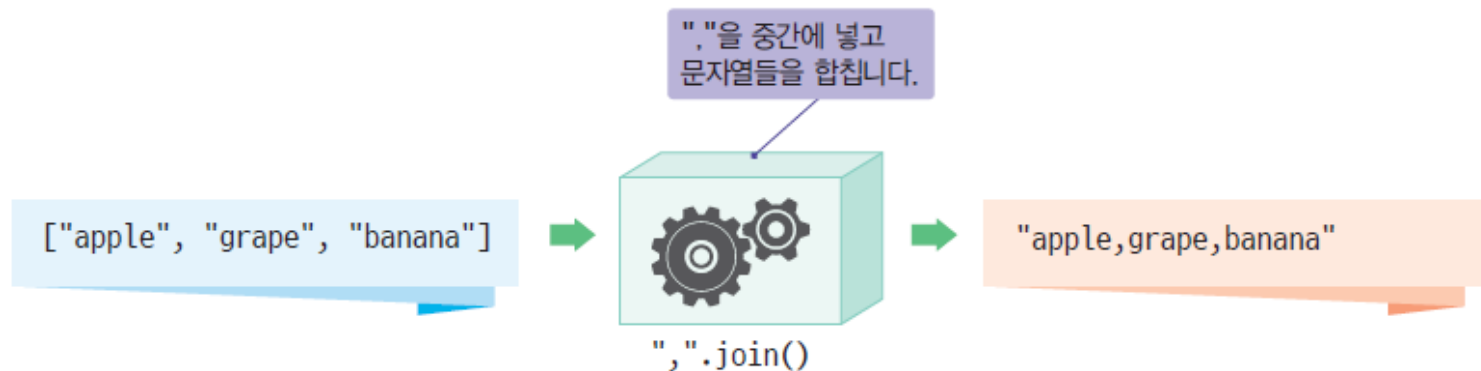
문자열 메소드: 문자열 결합하기

- `join()` : 단어들을 모아서 하나의 문자열로 만드는 역할. 결합문자를 지정하여 사용할 수 있다.

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

p344.py

```
>>> "-".join("010.1234.5678".split("."))  
"010-1234-5678"
```



Lab: 머리 글자어 만들기

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

문자열을 입력하시오: North Atlantic Treaty Organization
NATO

```
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 )
```

p349.py

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

```
address=input("이메일 주소를 입력하시오: ")  
(id, domain) = address.split("@")
```

p350.py

```
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}

```
sentence = input("문자열을 입력하시오: ")
```

p351.py

```
table = {"alphas":0,"digits":0,"spaces":0 }
```

```
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

- skip

Lab: OTP 발생 프로그램

- 일회용 암호 (OTP) 프로그램을 작성해보자.

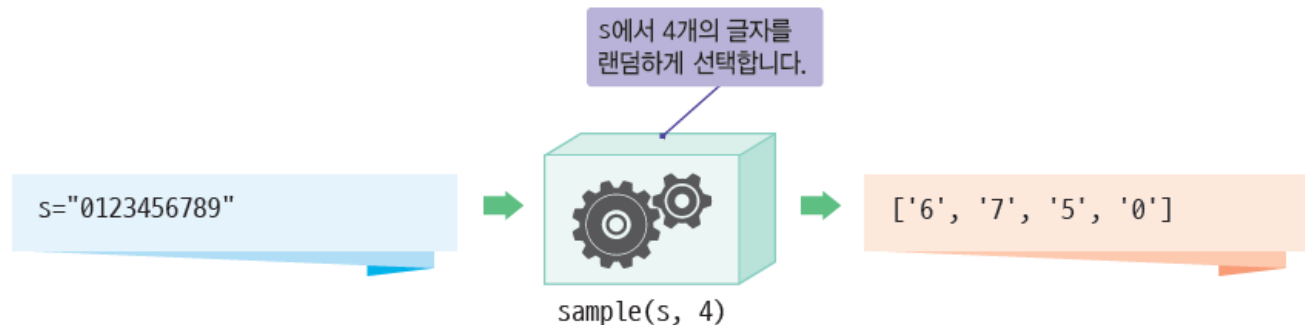
3482

```
import random
```

p353.py

```
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.

