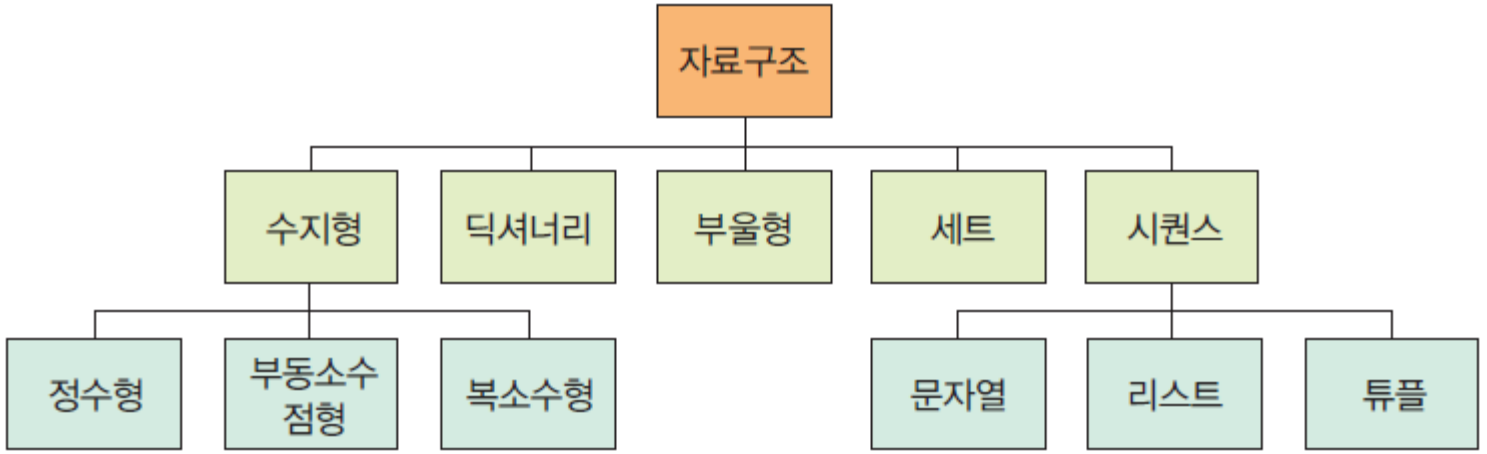
8장 튜플, 세트, 딕셔너리

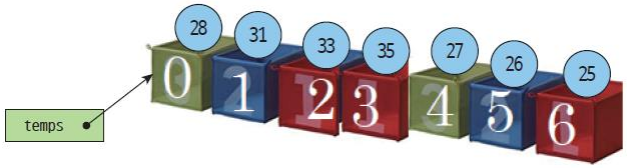
자료구조란. p269

- 자료구**조(data structure), 또는 데이터 구조 :** 자료들을 저장하는 여러 가지 구조



자료구조란. p269

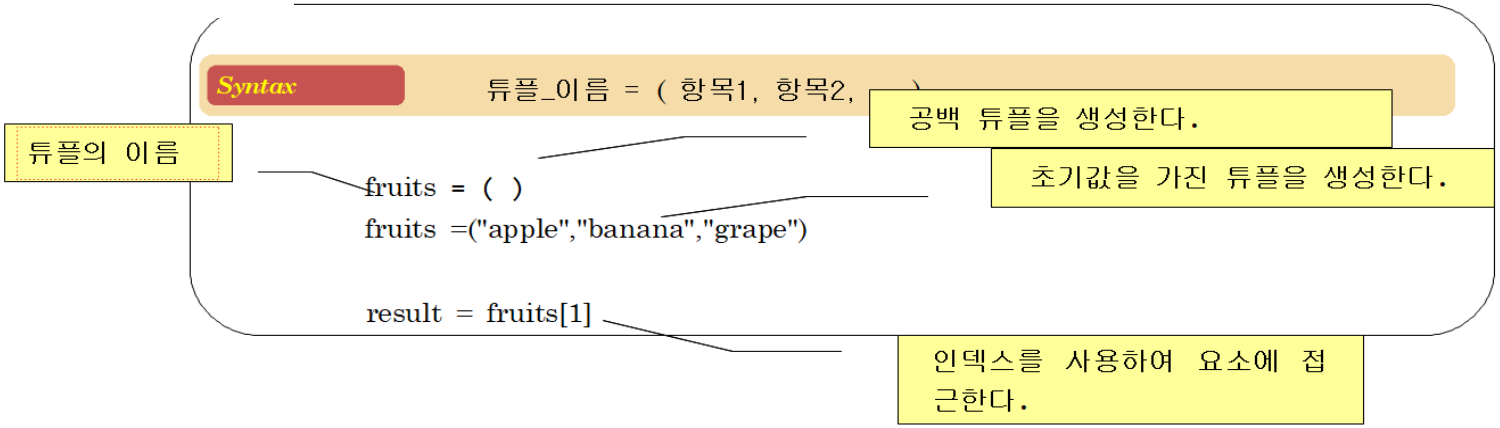
- 시퀀스(sequence)
  - 요소(element)로 구성되어 있고 요소 간에는 순서가 있다.
  - 시퀀스의 요소들은 번호를 부여 받는다. -> 인덱스(index)
  - 내장 시퀀스 : str, bytes, bytearray, list, tuple, range



  - 동일한 연산을 공유 : 인덱싱(indexing), 슬라이싱(slicing), 덧셈 연산(adding), 곱셈 연산(multiplying), len(), max(), min()

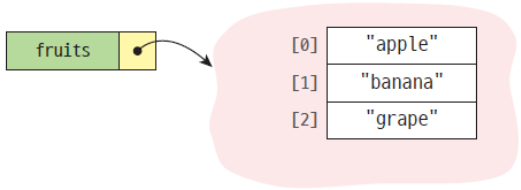
튜플. p270

- 리스트와 유사. 튜플은 변경이 불가능.




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

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



튜플. p271

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

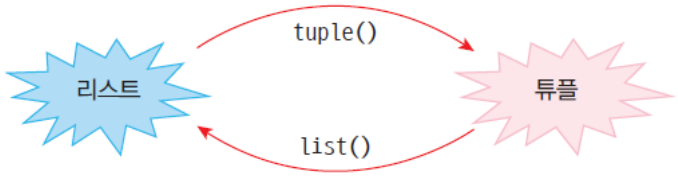
```
>>> fruits = ("apple", "banana", "grape") # 인덱스 사용
>>> fruits[1]
banana
```

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

```
fruits = ("apple", "banana", "grape")
for i in fruits:                           # 반복 루프 사용
    print(f, end=“ “)
```

튜플 <-> 리스트. p271

```
>>> myList = [1, 2, 3, 4]
>>> myTuple = tuple(myList)          # tuple()는 튜플을 생성하는 함수이다.
>>> myTuple
(1, 2, 3, 4)
```




```
>>> myTuple = (1, 2, 3, 4)
>>> myList = list(myTuple)           # list()는 리스트를 생성하는 함수이다.
>>> myList
[1, 2, 3, 4]
```

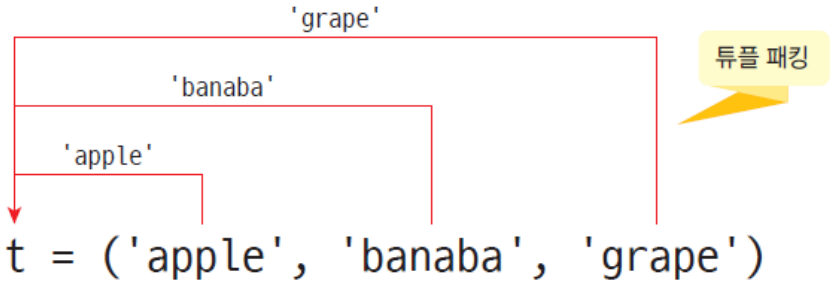
튜플 연산들. p272

- 튜플은 변경 불가 – 추가, 삭제 할 수 없다.
- 하지만 += 연산자를 이용하여 다른 튜플을 추가하는 것은 가능. 기존 튜플을 변경하는 것이 아니고 새로운 튜플을 생성하는 것이기 때문.

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

튜플 패킹과 언패킹. p272

- 튜플 패킹(packing) : 여러 개의 항목으로 튜플을 생성하는 것
- 튜플 언패킹(unpacking) : 튜플 안에 저장된 데이터를 풀어서 개별 변수에 저장하는 것




```
# 서로 다른 자료형에 대해서도 패킹, 언패킹 가능
>>> student = (“Kim”, [3.1, 3.6, 4.0, 0.0])
>>> name, grades = student
>>> name
Kim
>>> grades
[3.1, 3.6, 4.0, 0.0]
```

튜플 패킹과 언패킹. p273

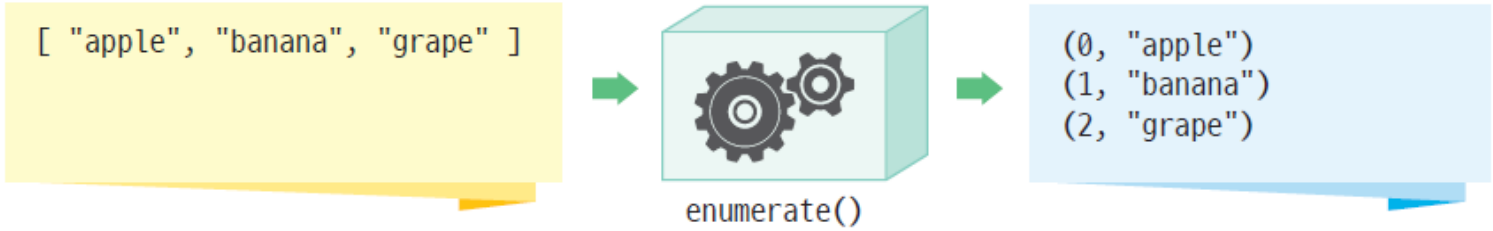
- 패킹과 언패킹을 이용해서 데이터의 순서를 바꾸기도 한다.

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

```
90  10
```

enumerate() 사용하기. p274

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




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

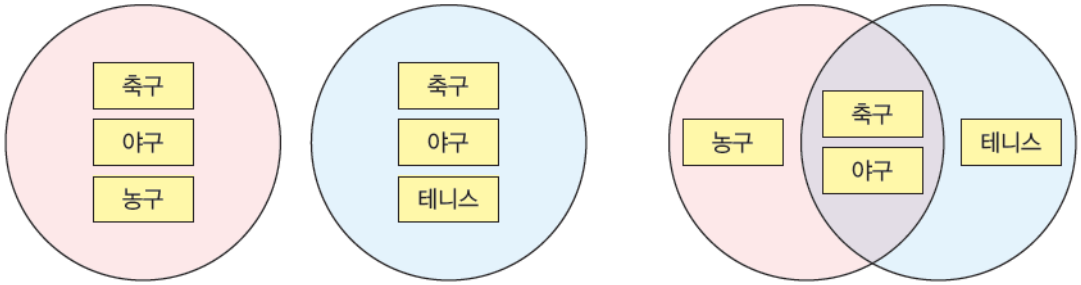
```
0 apple
1 banana
2 grape
```

튜플의 장점. p274

	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.

세트. p275

- 세트(set) : 집합.
- 고유한 값들을 저장하는 자료구조 -> 중복이 없다.
- 순서를 가지지 않는다.



세트 생성하기. p276

Syntax 세트_이름 = { 항목1, 항목2, 항목3, ... }

```
numbers = {1, 2, 3}
values = set( )
```

초기화된 세트를 생성한다.

공백 세트를 생성한다.

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

{ 1, 2, 3}

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

{ ‘a’, ‘b’, ‘c’}

세트의 연산. p276

- all(), any(), enumerate(), len(), max(), min(), sorted(), sum() 등

```
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.
```

```
fruits ={"apple","banana","grape"}
for x in fruits:                                     # 출력 순서가 다를 수 있다.
        print(x, end=" ")       .
```

```
grape banana apple
```

```
fruits ={"apple","banana","grape"}
for x in sorted(fruits):                             # 정렬된 순서로 출력한다.
        print(x, end=" ")
```

```
apple banana grape
```

요소 추가하고 삭제하기. p277

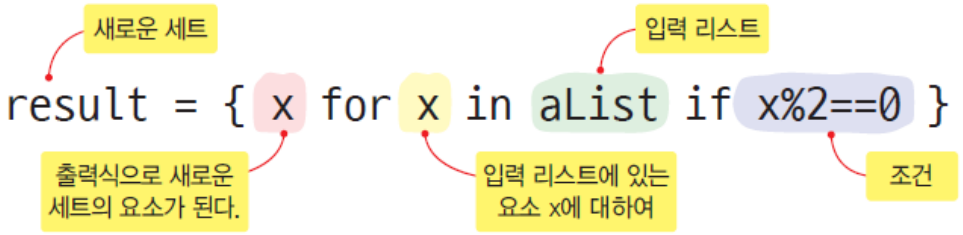
```
fruits ={"apple","banana","grape"}
fruits.add("kiwi") 된다.
```

```
# 요소를 삭제할 때는 remove() 메소드를 사용할 수 있다.
fruits ={"apple","banana","grape", "kiwi"}
fruits.remove("kiwi")
```

```
#세트의 모든 요소를 삭제하려면 clear() 메소드를 사용한다.
fruits ={"apple","banana","grape","kiwi"}
fruits.clear()                    # 공백 세트가 된다.
```

세트 함축 연산. p274

- 리스트 함축과 동일하고 [...] 대신에 { ... } 가 사용되는 것만 다른다.




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

{2, 4}

부분 집합 연산. p27개

- < 연산자나 issubset() 메소드를 사용

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

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

A는 B의 부분 집합입니다.

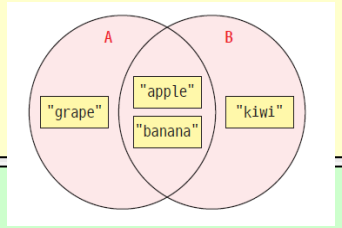
```
A ={"apple","banana","grape"}
B ={"apple","banana","grape","kiwi"}
if A == B :
        print("A와 B는 같습니다.")
else :
        print("A와 B는 같지 않습니다.")
```

A와 B는 같지 않습니다.

교집합, 합집합, 차집합 연산. p279

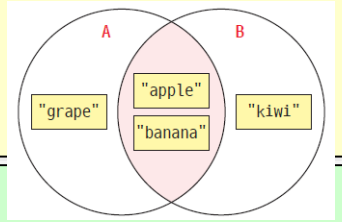
```
A = {"apple", "banana", "grape"}
B = {"apple", "banana", "kiwi"}
```

```
# 합집합
C = A | B                    # 또는 C = A.union(B)
print(C)
```




```
{'banana', 'grape', 'apple', 'kiwi'}
```

```
# 교집합
C = A & B                    # 또는 C = A.intersection(B)
print(C)
```

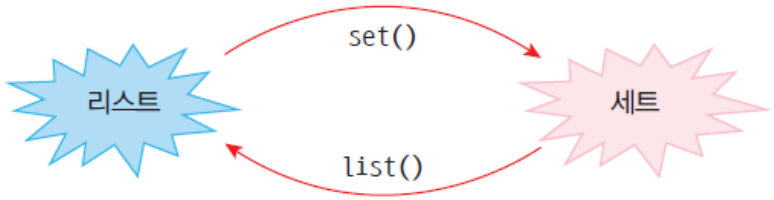



```
{'banana', 'apple'}
```

```
# 차집합
C = A - B                    # 또는 C = A.difference(B)
print(C)
```

```
{'grape'}
```

리스트 <-> 세트. p280




```
# 리스트 안에 서로 다른 정수가 몇 개 있을까?
>>> list1 =[1,2,3,4,5,1,2,4 ]
>>> len(set(list1))
5
```

```
# 2개의 리스트에 공통적으로 들어 있는 숫자는 ?
>>> list1 =[1,2,3,4,5 ]
>>> list2 =[3,4,5,6,7 ]
>>> set(list1)&set(list2)
{3, 4, 5}
```

```
# (추가) 세트는 순서 관계가 없다. 따라서 세트를 정렬하면 자동으로 리스트로 바뀐다.
>>> x = sorted({3, 4, 9, 7, 1})
>>> x
[1, 3, 4, 7, 9]
```

세트 메소드. p281

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

Lab 간단한 표절 검사 프로그램. p282

- 사용자로부터 2개의 문자열을 받아서 두 문자열의 공통 단어를 카운팅하여 표절률을 계산해보자.

첫 번째 문자열:Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do
두 번째 문자열:Kim was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do

표절률 = 85.71428571428571

Lab 중복되지 않은 단어의 개수 세기. p283

- 작문을 할 때 다양한 단어를 사용하면 높은 점수를 받는다. 단어를 얼마나 다양하게 사용하여 테스트를 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"}

딕셔너리. p284

- 딕셔너리(dictionary) : 중괄호 안에 항목을 쉼표로 분리시켜서 나열. 항목은 키(key)와 값(value)로 구성. 키와 값 사이에는 콜론(:)

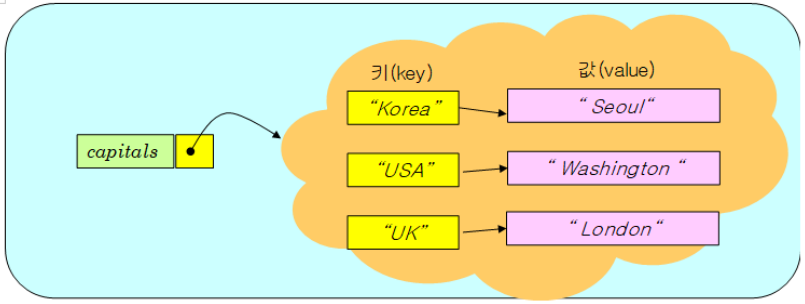
공백 딕셔너리를 생성한다.

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

```
capitals = { }
capitals = { “Korea”: ”Seoul” , ”USA”: "Washington", "UK": ”London” } # ②
```

값

키



항목 탐색하기. p285

- 리스트에서는 인덱스를 가지고 항목을 찾을 수 있지만 딕셔너리에서는 키를 가지고 값을 찾는다.

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

```
Seoul
```

```
# 존재하지 않는 키를 제시하면 KeyError 예외가 발생
print( capitals[“France”] )
```

```
...
KeyError: "France"
```

```
# get() 을 사용하면 해당 키를 찾을 수 없으면 두번 째 인수를 반환
print( capitals.get("France", "해당 키가 없습니다." ) )
```

```
해당 키가 없습니다.
```

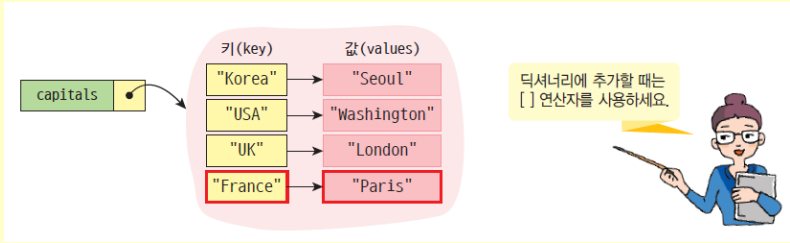
```
# in과 not in을 사용하면 지정된 키가 포함되어 있는지 확인할 수 있다.
if "France" in capitals :
        print( "딕셔너리에 포함됨" )
else:
        print( "딕셔너리에 포함되지 않음" )
```

```
딕셔너리에 포함되지 않음
```

항목 추가하기. p286

```
capitals ={"Korea":"Seoul","USA":"Washington","UK":"London"}
capitals["France"] = "Paris"
```

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




```
{'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'}
```

항목 삭제하기. p287

- 특정 항목을 삭제하려면 pop() 메소드를 사용

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

```
# 만약 주어진 키를 가진 항목이 없으면 KeyError 예외가 발생. 방지하려면 먼저 검사한다.
if "UK" in capitals :
        capitals.pop("UK")
```

```
# 모든 키-값을 삭제하는 함수는 clear()
# 딕셔너리가 비어 있는지 검사하려면 len()을 사용하면 된다.
capitals.clear()
if len(capitals)==0 :
    print("딕셔너리가 비어 있음")
else:
    print("딕셔너리가 비어 있지 않음")
```

```
딕셔너리가 비어 있음
```

항목 방문하기. p287

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

```
Korea  USA  UK
```

```
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
```

항목 방문하기. p288

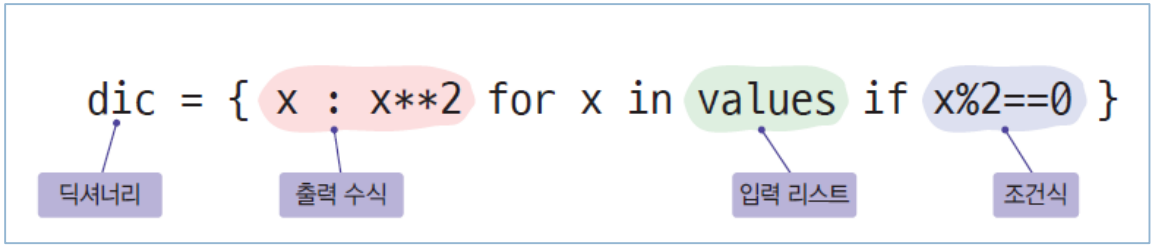
```
capitals ={"Korea":"Seoul","USA":"Washington","UK":"London"}
print(capitals.keys())
print(capitals.values())
```

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

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

```
Korea UK USA
```

딕셔너리 함축. p288




```
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 메소드. p289

연산	설명
d = dict()	공백 딕셔너리를 생성한다.
$d=\{k_1:v_1,k_2:v_2, \ldots,k_n:v_n\}$	초기값으로 딕셔너리를 생성한다.
len(d)	딕셔너리에 저장된 항목의 개수를 반환한다.
k in d	k가 딕셔너리 d 안에 있는지 여부를 반환한다.
k not in d	k가 딕셔너리 d 안에 없으면 True를 반환한다.
d[key] = value	딕셔너리에 키와 값을 저장한다.
v = d[key]	딕셔너리에서 key에 해당되는 값을 반환한다.
d.get(key, default)	주어진 키를 가지고 값을 찾는다. 만약 없으면 default 값이 반환된다.
d.pop(key)	항목을 삭제한다.
d.values()	딕셔너리 안의 모든 값의 시퀀스를 반환한다.
d.keys()	딕셔너리 안의 모든 키의 시퀀스를 반환한다.
d.items()	딕셔너리 안의 모든 (키, 값)을 반환한다.

Lab 영한 사전. p290

```
단어를 입력하시오: one
하나
```

```
단어를 입력하시오: two
둘
```

```
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. p291

- 3가지 과목에서 각 학생의 성적을 딕셔너리에 저장해 보자. 딕셔너리에서 각 학생의 성적을 꺼내서 각 학생들의 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= 63.666666666666664
Choi 의 평균성적= 50.0
```

- (도전문제) 교사가 학생들의 성적을 딕셔너리에 입력할 수 있도록 프로그램을 변경해보자. 어떤 방법을 사용해야 하는가?

Mini Project 주소록 작성. p292

- 주소록을 작성해보자. 딕셔너리를 사용하여서 연락처들을 저장한다. 사용자에게 메뉴를 제시하고 연락처 추가, 삭제, 검색을 지원하는 전체 프로그램을 작성해보자.

```
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 번화번호: 123-4567
...
```

연습문제. p294

Programming. p298