단국대학교
SW중심대학

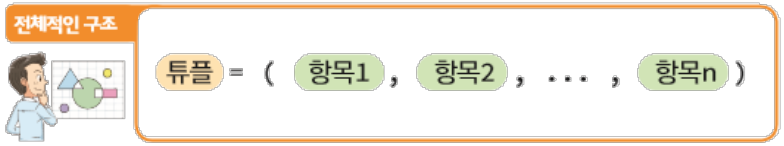
대학기초 소프트웨어 입문

Lecture 1-2. 자료형과 자료구조 (Python Basics)

튜플 (Tuple)

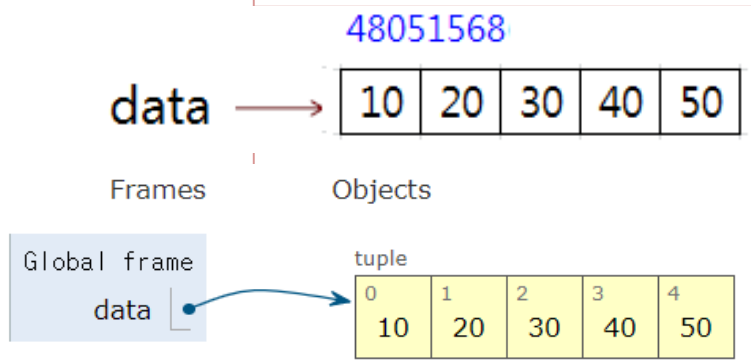
파이썬 자료구조

튜플(tuple)



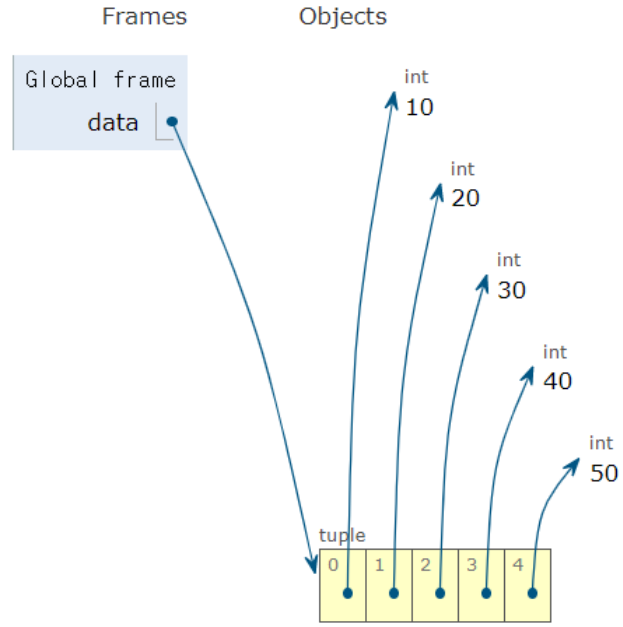
- 괄호 ( )로 표현
- 리스트와 유사, 데이터를 변경할 필요가 없는 경우 사용
  - 인덱싱, 슬라이싱 이용할 수 있음
  - +, *, in, not in, len() 사용가능

```
>>> data = (10, 20, 30, 40, 50)
>>> print(data)
(10, 20, 30, 40, 50)
>>> type(data)
<class 'tuple'>
>>> id(data)
48051568
>>>
```




```
>>> numbers = (1, 2, 3, 4, 5)
>>> colors = ('red', 'green', 'blue')
>>> t = numbers + colors
>>> t
(1, 2, 3, 4, 5, 'red', 'green', 'blue')
```

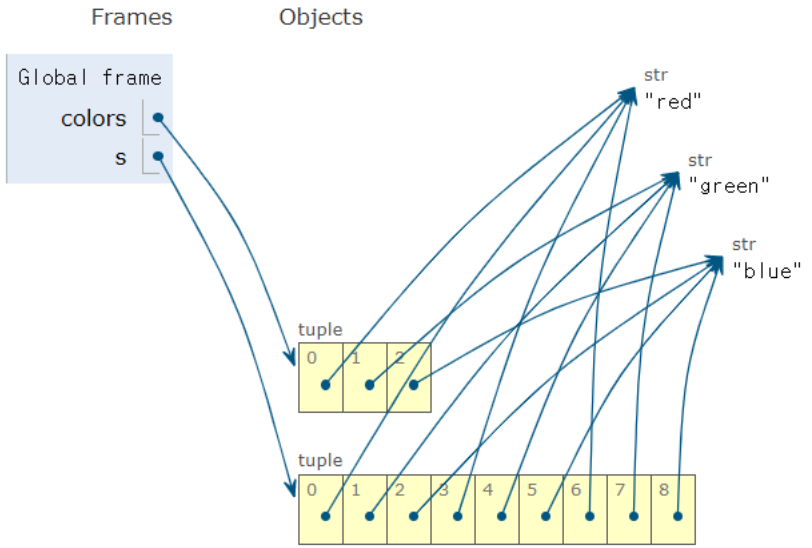
```
>>> s = colors * 3
>>> s
('red', 'green', 'blue', 'red', 'green', 'blue', 'red', 'green', 'blue')
```



- 빈 튜플

```
>>> t1 = () # 빈 튜플
>>> t2 = tuple()
```

```
>>> t = tuple('dankook')
>>> t
('d', 'a', 'n', 'k', 'o', 'o', 'k')
>>> t = tuple([1,2,3,4,5])
>>> t
(1, 2, 3, 4, 5)
>>> t = tuple(range(10))
>>> t
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```



튜플(tuple)

- immutable

```
>>> t1 = (1, 2, 3, 4, 5)
>>> print(t1[2])
3
>>> t1[0] = 100
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    t1[0] = 100
TypeError: 'tuple' object does not support item assignment
```

```
>>> my_tuple = (4, 2, 3, [6, 5])
>>> my_tuple[3][0] = 9
>>> my_tuple
(4, 2, 3, [9, 5])
```

할당된 튜플의 요소는 변경할 수 없지만 요소 자체가 리스트와 같은 변경 가능한 데이터 유형인 경우 중첩된 요소를 변경할 수 있다.

- 튜플 삭제

```
>>> my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
>>> del my_tuple[3]  #can't delete items
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    del my_tuple[3]  #can't delete items
TypeError: 'tuple' object doesn't support item deletion
```

```
>>> del my_tuple  # Can delete an entire tuple
>>> print(my_tuple)  # my_tuple이 삭제되어 NameError 발생됨
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    print(my_tuple)  # my_tuple이 삭제되어 NameError 발생됨
NameError: name 'my_tuple' is not defined
```

튜플(tuple)

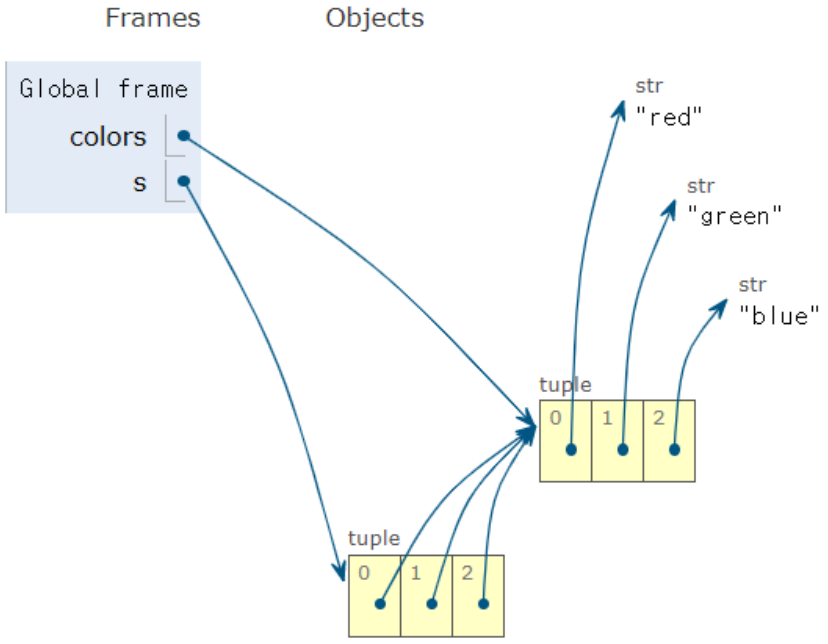
- 원소가 하나인 튜플 만들기

```
>>> t1 = (1,)   # 콤마 , 필요
>>> type(t1)
<class 'tuple'>
>>> s1 = (1)    # s1 = 1 과 동일
>>> type(s1)
<class 'int'>
>>> a = 1, 2, 3   #콤마를 사용하면 괄호를 생략해도 튜플로 인식
>>> type(a)
<class 'tuple'>
>>> b = 1
>>> c = 2,
>>> type(b)
<class 'int'>
>>> type(c)
<class 'tuple'>
```

```
>>> t2 = ('a',)
>>> t3 = ('a')
>>> type(t2), type(t3)
(<class 'tuple'>, <class 'str'>)
```

Syntactically, a tuple is a comma-separated list of values

```
>>> colors = ('red', 'green', 'blue')
>>> s = (colors,) * 3
>>> s
(('red', 'green', 'blue'), ('red', 'green', 'blue'), ('red', 'green', 'blue'))
```



튜플(tuple)

- 튜플 대입 연산 : 튜플은 여러 변수에 값을 동시에 할당할 수 있다.

```
>>> student1 = ('철수', 19, 'CS')
>>> (name, age, major) = student1
>>> name
'철수'
>>> age
19
>>> major
'CS'
```

```
>>> x, y, z = 1, 2, 3
>>> (x, y, z) = (1, 2, 3)
```

=

```
>>> x = 1
>>> y = 2
>>> z = 3
```

```
>>> my = ['yun', 20]
>>> name, age = my
>>> name
'yun'
>>> age
20
```

```
>>> addr = 'yun@dankook.ac.kr'
>>> name, domain = addr.split('@')
>>> name
'yun'
>>> domain
'dankook.ac.kr'
```

튜플(tuple)

- 튜플을 이용한 swap

```
>>> a = 10
>>> b = 20
>>> print(a, b)
10 20
>>> a, b = b, a
>>> print(a,b)
20 10
```

- 퓨플도 리스트와 같이 내부에 다른 튜플을 가질 수 있다.

```
>>> t = (1, 2, 'hello')
>>> u = t, ( 4,5,6,7)
>>> u
((1, 2, 'hello'), (4, 5, 6, 7))
```

u = t, 4,5,6,7 과의 차이는?

```
>>> u = t, 4,5,6,7
>>> u
((1, 2, 'hello'), 4, 5, 6, 7)
```

튜플(tuple)

- Method : count(), index()

```
>>> dir(tuple)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt
__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmul_
_', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'count', 'index']
```

```
>>> my = ('a', 'p', 'p', 'l', 'e')
>>> my.count('p')
2
>>> my.index('l')
3
```

- 튜플 비교 : 비교 연산자 ==, !=, >, <를 사용하여 2개의 튜플 비교할 수 있음

```
>>> (0, 1, 2) < ( 0, 3, 4)
True
>>> (0, 3) > (0, 2, 7)
True
```

파이썬 자료구조

튜플(tuple)

- 기본적인 튜플 연산들

파이썬 수식	결과	설명
`len((1, 2, 3))`	`3`	튜플의 길이
`(1, 2, 3) + (4, 5, 6)`	`(1, 2, 3, 4, 5, 6)`	접합
`('Hi!',) * 4`	`('Hi!', 'Hi!', 'Hi!', 'Hi!')`	반복
`3 in (1, 2, 3)`	`True`	멤버쉽
`for x in (1, 2, 3): print x,`	`1 2 3`	반복

함수	설명
`len(t)`	튜플의 길이를 반환한다.
`max(t)`	튜플에 저장된 최대값을 반환한다.
`min(t)`	튜플에 저장된 최소값을 반환한다.
`tuple(seq)`	리스트를 튜플로 변환한다.

t = ((1, 2, [10, 20]), 4, 5, 6, 7)

len(t) 의 결과는?

For lists or tuples the number of elements are counted, whereas a sublist counts as one element.

Lab: 함수의 튜플 반환 예제

- 원의 넓이와 둘레를 동시에 반환하는 함수를 작성, 테스트해보자.

```
원의 반지름을 입력하시오: 10
원의 넓이는 314.1592653589793이고 원의 둘레는62.831853071795860이다.
```

Solution

```
import math

def calCircle(r):
    # 반지름이 r인 원의 넓이와 둘레를 동시에 반환하는 함수  (area, circum)
    area  = math.pi * r * r
    circum = 2 * math.pi * r
    return (area, circum)

radius = float(input("원의 반지름을 입력하시오: "))
(a, c) = calCircle(radius)
print("원의 넓이는 "+str(a)+"이고 원의 둘레는"+str(c)+"이다.")
```

파이썬 자료구조

```
for x, y in [ (7, 2), (5, 8), (6, 4) ]:
    print(x,y)

a, b, c, d = range(7, 11)
print('a=%d, b=%d, c=%d, d=%d'%(a,b,c,d))
```

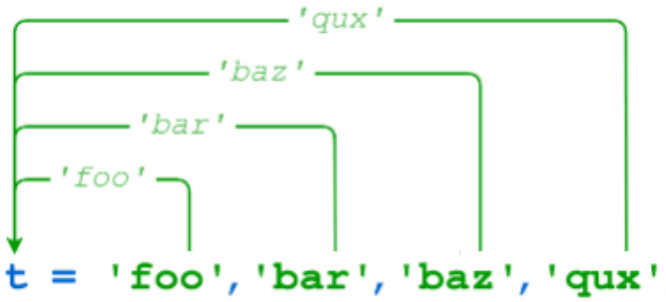
```
7 2
5 8
6 4
```

```
a=7, b=8, c=9, d=10
```

튜플(tuple)

- Tuple Packing : 괄호를 사용하지 않고 튜플을 생성하는 것 (Auto packing)

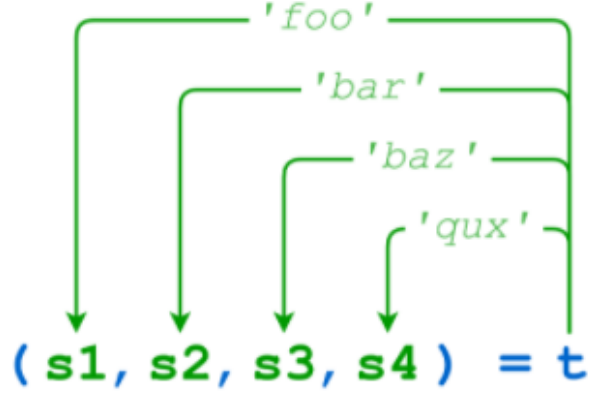
```
>>> t = 'foo', 'bar', 'baz', 'qux'   #packing
>>> t
('foo', 'bar', 'baz', 'qux')
```




```
>>> my = 1, #packing
>>> my
(1,)
```

- Tuple Unpacking : 튜플에서 값을 꺼내서 변수에 대입하는 것

```
>>> (s1, s2, s3, s4) = t  # unpacking
>>> print(f's1 = {s1}, s2 = {s2}, s2 = {s2}, s3 = {s3}')
s1 = foo, s2 = bar, s2 = bar, s3 = baz
```



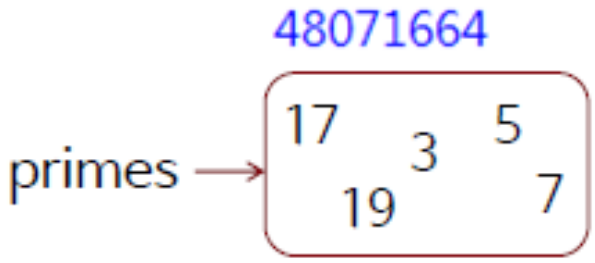

```
>>> t = ('spam', 'egg', 'bacon', 'tomato')
>>> s1, s2, s3 = t
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    s1, s2, s3 = t
ValueError: too many values to unpack (expected 3)
```

unpacking 할 때에 왼쪽의 변수 개수는 튜플의 요소 개수와 일치해야 한다.

집합

파이썬 자료구조

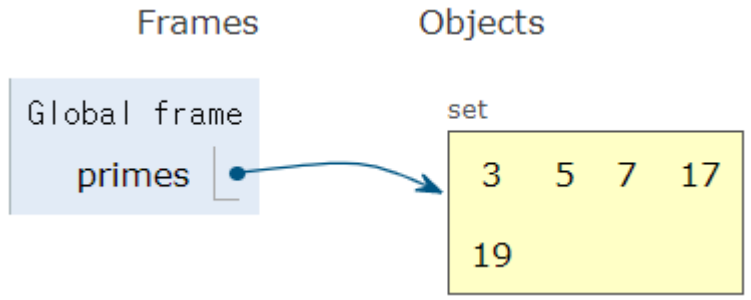
집합(set)

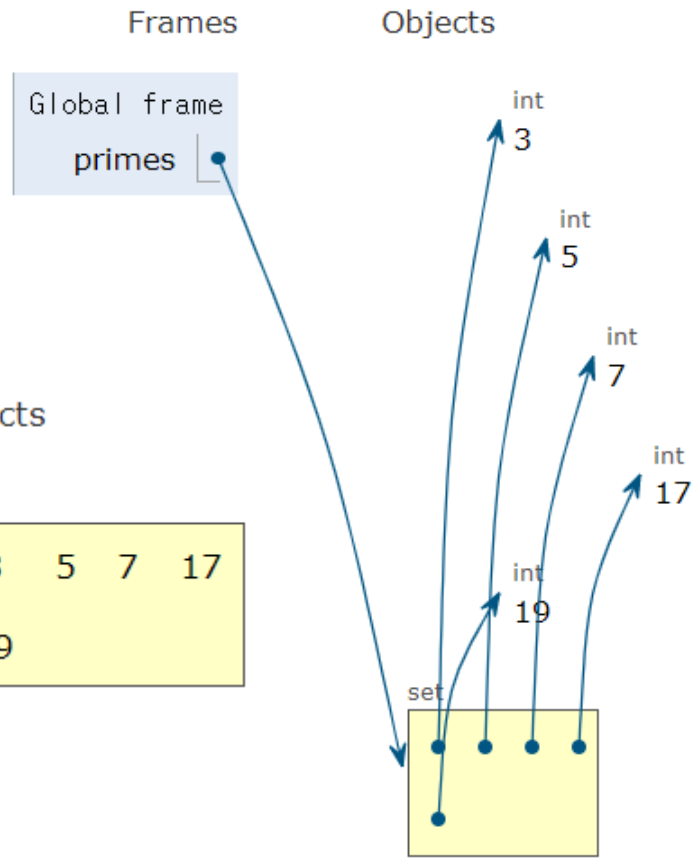



```
>>> primes = {7, 17, 3, 5, 3, 19}
>>> print(primes)
{3, 5, 7, 17, 19}
>>> type(primes)
<class 'set'>
>>> id(primes)
48071664
```

- 중괄호 { }로 표현
- 중복되지 않는 여러 개의 자료들을 모아서 저장해야 하는 경우 사용
- 순서가 없는 데이터 구조
- In, not in, len() 사용 가능
- mutable
  - 집합 자체는 변경가능(요소를 추가, 제거 가능), 각 요소들은 변경 불가능(immutable) 해야함.
- 집합 연산 제공(합집합, 교집합, 차집합 등)
- 빈 집합

```
>>> s = set()  # 빈 집합 생성
>>> type(s)
<class 'set'>
>>> t = {}  # 빈 사전 생성
>>> type(t)
<class 'dict'>
```





yunyi96@naver.com

집합(set)

- 집합 생성 { }
  - set( ) 함수 : sequence, iterable object를 사용하여 집합을 생성
    - 요소의 개수에 제한이 없으며 다른 자료형 (정수, 실수, 튜플, 문자열 등)을 요소로 가질 수 있다.
    - 집합은 모든 요소들은 해싱을 이용하여 저장하고 관리한다. 해싱 : 각 객체에 식별 숫자 코드를 부여하여 객체를 테이블에 저장하는 것
      ➔ 집합은 리스트 , 집합 또는 사전 과 같은 변경 가능한 객체를 요소로 가질 수 없다 .

```
>>> fruits = { 'apple', 'banana', 'pineapple'}
>>> myset = {1.0, 2.0, 'Hello world', (1, 2, 3)}
>>> type(fruits), type(myset)
(<class 'set'>, <class 'set'>)
>>> myset = { 1, 2, [3, 4] }
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    myset = { 1, 2, [3, 4] }
TypeError: unhashable type: 'list'
```

```
>>> myset = {1, 2, 3, 4, 3, 2}
>>> myset
{1, 2, 3, 4}
```

```
>>> x = set('python best')
>>> x
{'s', 'o', 'n', 't', ' ', 'p', 'b', 'e', 'h', 'y'}
>>> type(x)
<class 'set'>
>>> x = set([1,2,3,4,5,6])
>>> x
{1, 2, 3, 4, 5, 6}
>>> type(x)
<class 'set'>
>>> x = set([1,2,3,1,2,3])
>>> x
{1, 2, 3}
>>> x = set((1,2,3,4,5))
>>> x
{1, 2, 3, 4, 5}
```

```
>>> x = set(((1,2), (3,4), (5,6)))
>>> x
{(1, 2), (3, 4), (5, 6)}
>>> type(x)
<class 'set'>
>>> y = set([[1,2], [3,4], [5,6]])
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    y = set([[1,2], [3,4], [5,6]])
TypeError: unhashable type: 'list'
```

Every set element is unique (no duplicates) and must be immutable (cannot be changed).

집합(set)

- In 연산자

```
>>> my_set = set("apple")
>>> 'a' in my_set
True
>>> 'p' not in my_set
False
```

```
>>> numbers = {2, 1, 3}
>>> if 1 in numbers:
        print("집합 안에 1이 있습니다.")


집합 안에 1이 있습니다.
```

- for 문을 사용하여 집합의 각 요소를 반복

```
>>> numbers = {2, 1, 3}
>>> for x in numbers:
        print(x, end=" ")


1 2 3
```

```
>>> for letter in set("apple"):
        print(letter)


e
a
p
l
```

집합(set)

- 메소드

메소드	설명
add(x)	집합에 원소 x를 추가
clear()	집합에 있는 모든 원소 제거, 공집합으로 만든다.
copy()	집합을 복사
discard(x)	x가 집합의 구성원이면 제거하고 아니면 아무것도 하지 않는다.
pop()	임의의 집합 요소를 제거하고 반환한다. 어떤 요소를 가져올 지 알 수 없다. 집합이 비어 있는 경우 KeyError 발생.
remove(x)	집합에서 x 요소 제거, 만약 x 요소가 구성원이 아닌 경우 KeyError 오류 발생

```
>>> dir(set)
['__and__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__
', '__getattribute__', '__gt__', '__hash__', '__iand__', '__init__', '__init_subclass__', '__ior__', '__isub__',
'__iter__', '__ixor__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__or__', '__rand__', '__reduce__', '
__reduce_ex__', '__repr__', '__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub_
_', '__subclasshook__', '__xor__', 'add', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'int
ersection', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric_diff
erence', 'symmetric_difference_update', 'union', 'update']
```

집합(set)

- 세트에 요소 추가하기

```
>>> numbers = { 2, 1, 3 }
>>> numbers[0]
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    numbers[0]
TypeError: 'set' object is not subscriptable
>>> numbers.add(4)  # 집합 numbers에 데이터 4를 추가
>>> numbers
{1, 2, 3, 4}
>>> numbers.add(2)  # 이미 있는 데이터를 추가하면 변동이 없다.
>>> numbers
{1, 2, 3, 4}
```

순서가 없으므로 위치를 가지고 세트의 항목에 접근할 수 없음

```
>>> t = {'foo', 'bar', 'baz', 'qux' }
>>> t.update(['a','b'])
>>> t
{'b', 'baz', 'foo', 'qux', 'a', 'bar'}
```

- 공집합 만들기

```
>>> a = {1,2,5,3,7}
>>> a.clear() # a를 공집합으로 만들기
>>> print(a)
set()
```

집합(set)

- 세트에 요소 삭제하기

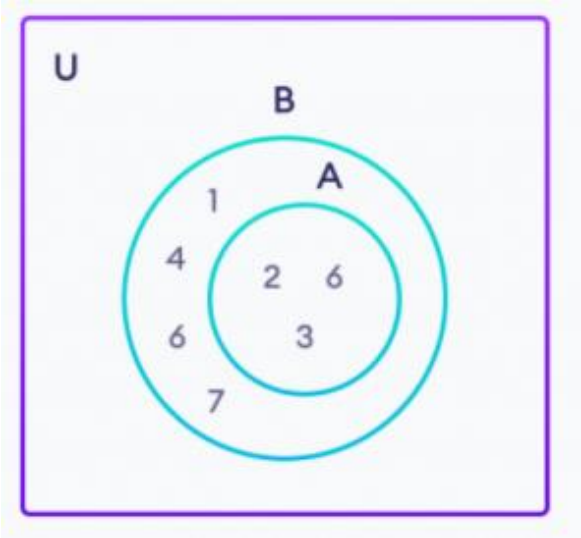
```
>>> numbers = { 1,2,3,4}
>>> numbers.discard(4) # 집합 numbers에 데이터 4를 삭제
>>> numbers
{1, 2, 3}
>>> numbers.discard(5)  # 집합에 없는 원소를 삭제해도 error 없음
>>> numbers
{1, 2, 3}
>>> numbers.remove(2)   # 2 삭제
>>> numbers
{1, 3}
>>> numbers.remove(5)   # 집합에 없는 원소를 삭제하면 error 발생
Traceback (most recent call last):
  File "<pyshell#44>", line 1, in <module>
    numbers.remove(5)   # 집합에 없는 원소를 삭제하면 error 발생
KeyError: 5
```

```
>>> a = {1,2,5,3,7}
>>> a.pop() # 집합에서 임의의 원소를 반환하고 삭제
1
>>> print(a)
{2, 3, 5, 7}
```

집합(set)

- 부분 집합 연산

```
>>> A = {1, 2, 3}
>>> B = {1, 2, 3}
>>> A == B
True
>>>
>>> A = {1, 2, 3, 4, 5}
>>> B = {1, 2, 3}
>>> B < A
True
>>>
>>> A = {1, 2, 3, 4, 5}
>>> B = {1, 2, 3}
>>> B.issubset(A)
True
```

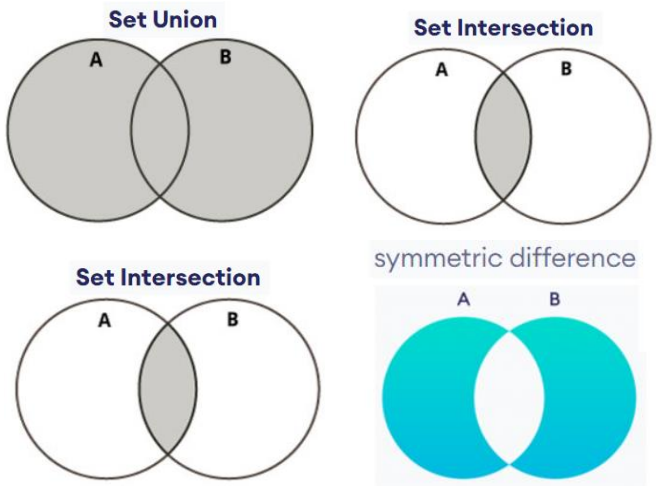


set A is a subset of B.

A.issubset(B)

집합(set)

- 집합 연산




```
>>> A = {1, 2, 3}
>>> B = {3, 4, 5}
>>> A.union(B)
{1, 2, 3, 4, 5}
>>>
>>> A.intersection(B)
{3}
>>>
>>> A.difference(B)
{1, 2}
>>>
>>> A.symmetric_difference(B)
{1, 2, 4, 5}
```

```
>>> A = {1, 2, 3}
>>> B = {3, 4, 5}
>>> A | B
{1, 2, 3, 4, 5}
>>>
>>> A & B
{3}
>>>
>>> A - B
{1, 2}
>>>
>>> A ^ B
{1, 2, 4, 5}
```

```
>>> x = {'a', 'b', 'c', 'd', 'e'}
>>> y = {'b', 'c' }
>>> z = {'c', 'd'}
>>> x.difference(y,z)  # x - y - z
{'a', 'e'}
>>> x.union(y, z) # x | y | z
{'c', 'd', 'b', 'a', 'e'}
>>> x.intersection(y, z)  # x & y & z
{'c'}
```

Method Name	Operator	Use	Explanation
union(*others)	|	A.union(B) A.union(B, C, ...)	set | other | ... 집합과 모든 others에 있는 원소들로 구성된 새 집합을 반환
intersection(*others)	&	A.intersection(B) A.intersection(B, C, ...)	set & other & ... 집합과 모든 others의 공통 원소들로 구성된 새 집합을 반환
difference(*others)	-	A.difference(B) A.difference(B, C, ...)	set - other - ... 집합에는 포함되었으나 others에는 포함되지 않은 원소들로 구성된 새 집합을 반환
symmetric_difference(other)	^	A.symmetric_difference(B)	set ^ other 집합이나 other에 포함되어 있으나 둘 모두에 포함되지는 않은 원소들로 구성된 새 집합을 반환
issubset(other)	<=		set <= other 집합의 모든 원소가 *other* 에 포함되는지 검사

Lab: 파티 동시 참석자 알아내기

- 파티에 참석한 사람들의 명단이 세트 A와 B에 각각 저장되어 있다. 2개 파티에 모두 참석한 사람들의 명단을 출력하려면 어떻게 해야 할까?

```
2개의 파티에 모두 참석한 사람은 다음과 같습니다.
{'Park'}
```

Solution

```
partyA = set(["Park", "Kim", "Lee"])
partyB = set(["Park", "Choi"])

print("2개의 파티에 모두 참석한 사람은 다음과 같습니다. ")
print ( partyA.intersection(partyB))
```

```
A = input("파티 A 명단: ")
B = input("파티 B 명단: ")

partyA = set()
partyB = set()

for s in A.split():
    partyA.add(s)

for s in B.split():
    partyB.add(s)

print("2개의 파티에 모두 참석한 사람은 다음과 같습니다. ")
print ( partyA.intersection(partyB))
```

Lab: 파일에서 중복되지 않은 단어의 개수

- 텍스트 파일을 읽어서 단어를 얼마나 다양하게 사용하여 문서를 작성하였는지를 계산하는 프로그램을 작성해보자.

```
입력 파일 이름: proverbs.txt
사용된 단어의 개수= 18
{'travels', 'half', 'that', 'news', 'alls', 'well', 'fast',
'feather', 'flock', 'bad', 'together', 'ends', 'is', 'a', 'done',
'begun', 'birds', 'of'}
```

proverbs.txt

All's well that ends well.
Bad news travels fast.
Well begun is half done.
Birds of a feather flock together.

Solution

```
# 단어에서 구두점을 제거하고 소문자로 만든다.
def process(w):
        output =""
        for ch in w:
                if( ch.isalpha() ):
                        output += ch
        return output.lower()

words = set()

#문자열에 저장, 작은따옴표 3개 혹은 큰따옴표 3개 -> 여러 줄을 하나의 문자열로
string = '''
AllW's well that ends well.
Bad news travels fast.
Well begun is half done.
Birds of a feather flock together.
'''

# 문자열을 스페이스나 개행으로 나누어 읽기
for s in string.split():
        words.add(process(s))   # 단어를 세트에 추가한다.

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

사전(DICTIONARY)

파이썬 자료구조

전체적인 구조

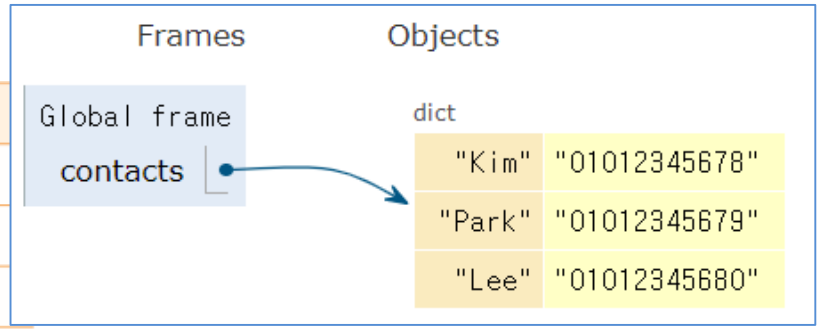
딕셔너리 = { 키1 : 값1 , 키2 : 값2 , ... }

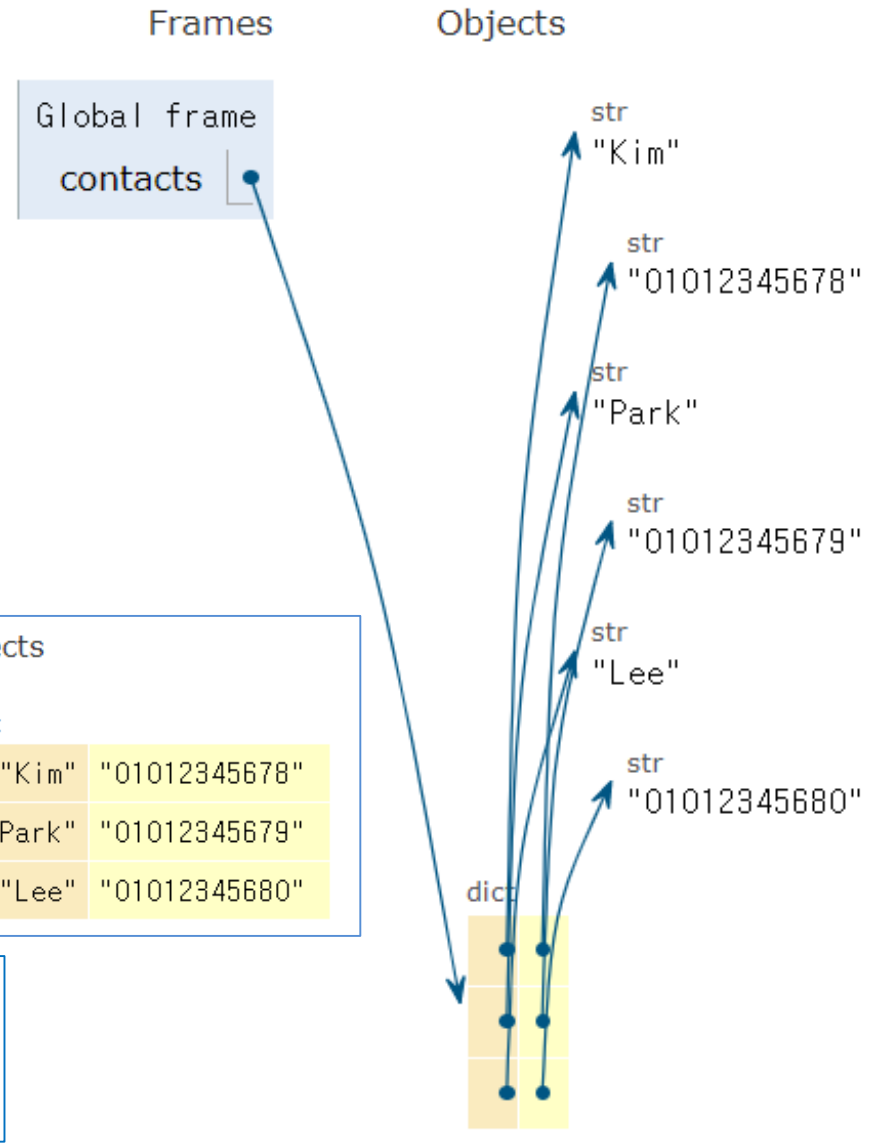
딕셔너리(dict)

- 키(key)와 값(value)의 쌍으로 구성되는 집합의 일종
  - 사전에는 (키:값)의 쌍으로 하나의 데이터가 저장된다.
- 키는 다른 데이터와 중복을 허용하지 않음.
- 중괄호 { }로 표현 – 항목에 키와 값의 쌍을 Key:value 으로 표현
- +, * 을 사용할 수 없다. 인덱스 기호([ ])는 사용
- In, not in , len() 사용 가능
- 순서 개념이 없다
- 공백 딕셔너리

```
emptyA = { }
emptyB = dict()
```

키(key)	값(value)
"Kim"	"01012345678"
"Park"	"01012345679"
"Lee"	"01012345680"






```
>>> contacts = {'Kim':'01012345678',  'Park':'01012345679', 'Lee':'01012345680' }
>>> contacts
{'Kim': '01012345678', 'Park': '01012345679', 'Lee': '01012345680'}
```

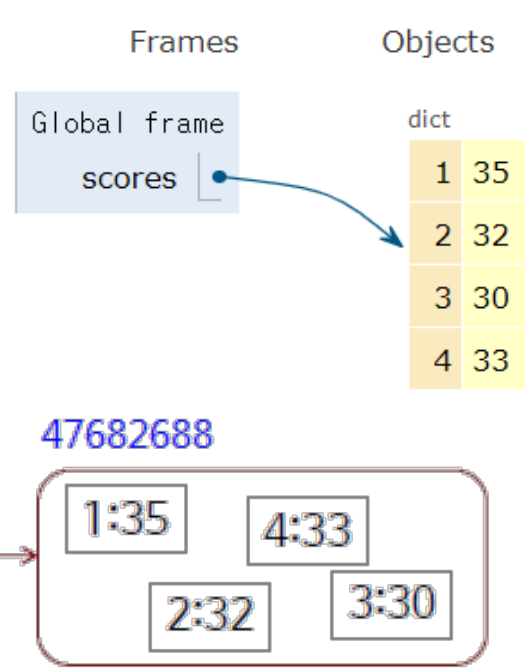
파이썬 자료구조

딕셔너리(dict)

- 사전의 키
  - mutable 자료형은 키가 될 수 없음 (리스트, 집합, 사전)
  - 정수, 실수, bool, 복소수, 문자열, 튜플은 키가 될 수 있음
- 사전의 값 : 모든 자료형이 사전의 값이 될 수 있음
- dict() 생성자
  - 키와 값의 쌍을 갖는 Tuple 리스트를 받아들이거나 dict(key=value, key=value, ...) 식의 키-값을 직접 파라미터로 지정하는 방식을 사용하여 사전을 생성

```
>>> scores = {1:35, 2:32, 3:30, 4:33}
>>> print(scores)
{1: 35, 2: 32, 3: 30, 4: 33}
>>> type(scores)
<class 'dict'>
>>> id(scores)
47682688
>>>
```

scores




```
>>> numbers = {x: x**2 for x in (2, 4, 6)}
>>> numbers
{2: 4, 4: 16, 6: 36}
```

```
>>> scores =dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
>>> scores
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>>
>>> students = dict(sape=4139, guido=4127, jack=4098)
>>> students
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> type(students)
<class 'dict'>
```

파이썬 자료구조

- 딕셔너리(dict)

```
>>> dir(dict)
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

메소드	설명
clear ()	사전의 내용을 모두 지운다.
copy ()	사전을 복사한다.
items()	사전에 있는 모든 데이터의 (키,값)을 모두 반환한다.
keys()	사전에서 키만 반환한다.
values()	사전에서 값만 반환한다.
update(D)	사전 D를 추가한다.

```
>>> scores = {1:35, 2:32, 3:30, 4:33}
>>> scores.items()
dict_items([(1, 35), (2, 32), (3, 30), (4, 33)])
>>>
>>> scores.keys()
dict_keys([1, 2, 3, 4])
>>> scores.values()
dict_values([35, 32, 30, 33])
```

```
>>> s_key = list(scores.keys())
>>> s_key
[1, 2, 3, 4]
```

```
>>> scores = {1:35, 2:32, 3:30, 4:33}
>>> scores2 = {5:40, 6:42}
>>> scores.update(scores2)
>>> scores
{1: 35, 2: 32, 3: 30, 4: 33, 5: 40, 6: 42}
>>>
>>> scores2
{5: 40, 6: 42}
```

파이썬 자료구조

딕셔너리(dict)

- 항목 접근하기 : [ ] , get( )메소드 사용
  - **[ ]** : 사전명[키] , 키가 없으면 KeyError 발생
  - **get()**메소드
    - 딕셔너리명.get(key, default=None)
    - [ ] 사용과 유사하나 key가 없을 경우
      에러 대신 None 혹은 default value 리턴

```
>>> dic = {1:'a', 2:'b', 3:'c', 4:'d'}
>>> dic[5]
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    dic[5]
KeyError: 5
>>> print(dic.get(5))
None
>>> print(dic.get(2))
b
>>> print(dic.get(5,'해당 키가 없음'))
해당 키가 없음
```

```
>>> contacts = {'Kim':'01012345678',  'Park':'01012345679', 'Lee':'01012345680' }
>>> contacts['Kim']   # 특정 요소 읽기 ★ key를 index 처럼 사용
'01012345678'
>>> contacts.get('Kim')  #get()메소드 => 딕셔너리명.get(키)
'01012345678'
```

```
>>> if "Kim" in contacts:
        print("키가 딕셔너리에 있음")


키가 딕셔너리에 있음
```

딕셔너리(dict)

- 항목 추가, 수정하기
  - **[ ]** 이용 : 키가 이미 존재 하는 경우 업데이트하고, 키가 없는 경우 새 쌍이 사전에 추가

```
>>> contacts = {'Kim':'01012345678',  'Park':'01012345679', 'Lee':'01012345680' }
>>> contacts['Choi'] = '01056781234'   # 추가하기
>>> contacts
{'Kim': '01012345678', 'Park': '01012345679', 'Lee': '01012345680', 'Choi': '01056781234'}
>>>
>>> contacts['Kim'] = '01012340000' # 수정하기
>>> contacts
{'Kim': '01012340000', 'Park': '01012345679', 'Lee': '01012345680', 'Choi': '01056781234'}
```

  - **update( )** : 사전 또는 키:값 쌍의 반복 가능한 객체 사용

```
>>> d = {1: "one", 2: "three"}
>>> d.update({2:'two'})
>>> d
{1: 'one', 2: 'two'}
>>> items = {3:'three', 4:'four'}
>>> d.update(items)
>>> d
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

```
>>> d = {'x': 2}
>>> d.update(y = 3, z = 0)
>>> d
{'x': 2, 'y': 3, 'z': 0}
```

파이썬 자료구조

딕셔너리(dict)

- 항목 삭제하기
  - **pop(key[, default])** : 키에 해당하는 값을 리턴후 제거
    default : 키가 사전에 없을 때 반환되는 값을 지정, 지정하지 않은 경우 키가 없으면 KeyError 발생
  - **popitem( )** : 임의의 항목 (key, value)을 제거하고 반환, 사전이 비어있는 경우 KeyError 발생
    - LIFO (Last In, First Out) 순서로 사전에서 (키, 값) 쌍을 제거하고 반환(가장 최근에 삽입된 요소(키,값) 쌍을 반환)
  - **del** 문 : 사전의 개별 항목 또는 사전 자체를 제거

```
>>> contacts = {'Kim':'01012345678',  'Park':'01012345679', 'Lee':'01012345680' }
>>> contacts.pop("Kim") #pop()를 호출하여 삭제 값을 리턴후 제거
'01012345678'
>>> contacts
{'Park': '01012345679', 'Lee': '01012345680'}
>>> del contacts['Lee']    #삭제 contacts['Lee']
>>> contacts
{'Park': '01012345679'}
```

```
>>> contacts.pop("Kim")
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    contacts.pop("Kim")
KeyError: 'Kim'
>>> contacts.pop("Kim", '찾는 항목 없음')
'찾는 항목 없음'
```

```
>>> person = {'name': 'Phill', 'age': 22, 'salary': 3500.0}
>>> person.popitem()
('salary', 3500.0)
>>> person.popitem()
('age', 22)
>>> person.popitem()
('name', 'Phill')
>>> person.popitem()
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    person.popitem()
KeyError: 'popitem(): dictionary is empty'
```

딕셔너리(dict)

- 항목 삭제하기

```
>>> prices = {'apple': 0.40, 'orange': 0.35, 'banana': 0.25}
>>> for key in list(prices.keys()):  # Use a list instead of a view
        if key == 'orange':
            del prices[key]  # Delete a key from prices


>>> prices
{'apple': 0.4, 'banana': 0.25}
```

Python에서 사전을 반복하는 동안 키를 수정하는 안전한 방법

```
>>> # Python 3. dict.keys() returns a view object, not a list
>>> prices = {'apple': 0.40, 'orange': 0.35, 'banana': 0.25}
>>> for key in prices.keys():
        if key == 'orange':
            del prices[key]


Traceback (most recent call last):
  File "<pyshell#179>", line 1, in <module>
    for key in prices.keys():
RuntimeError: dictionary changed size during iteration
```

파이썬 자료구조

딕셔너리(dict)

- 항목 순회하기
  - items( ) : 사전의 모든 항목을 (키, 값) 튜플 쌍의 목록을 표시하는 뷰 객체로 반환
  - keys() : 사전의 모든 키 목록을 표시하는 뷰 객체로 반환
  - values() : 사전의 모든 값 목록을 표시하는 뷰 객체로 반환

```
>>> sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }
>>> sales.items()
dict_items([('apple', 2), ('orange', 3), ('grapes', 4)])
>>> sales.keys()
dict_keys(['apple', 'orange', 'grapes'])
>>> sales.values()
dict_values([2, 3, 4])
```

```
>>> scores = { 'Korean': 80, 'Math': 90, 'English': 80}
>>> for item in scores.items():
        print(item)


('Korean', 80)
('Math', 90)
('English', 80)
```

```
>>> for item in sorted(scores.items()) : print(item)  #키를 기준으로 정렬

('English', 80)
('Korean', 80)
('Math', 90)
```

```
>>> scores = { 'Korean': 80, 'Math': 90, 'English': 80}
>>> for i in scores: print(i)

Korean
Math
English
```

```
>>> for i in scores.items(): print('%s - %d'% i)

Korean - 80
Math - 90
English - 80
```

```
>>> for i in scores.keys() : print(i)

Korean
Math
English
```

```
>>> for i in scores.values() : print(i)

80
90
80
```

파이썬 자료구조

```
>>> help(sorted)
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

딕셔너리(dict)

정렬

- **sorted(**iterable**) 내장함수 :** 사전의 키에 대해 오름차순 정렬하여 리스트로 반환 (Python 3.6 버전 이상인 경우 사용 가능)
- **reversed( ) :** 사전의 키에 대해 역방향으로 반복가능한 객체(iterator object) 반환(Python 3.8 버전 이상인 경우)

```
>>> a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}
>>> sorted(a_dict)
['color', 'fruit', 'pet']
>>> for key in sorted(a_dict):
        print(key,'->', a_dict[key])

color -> blue
fruit -> apple
pet -> dog
>>> type(sorted(a_dict))
<class 'list'>
```

```
>>> for key in sorted(a_dict, reverse=True):
        print(key, '→', a_dict[key])

pet → dog
fruit → apple
color → blue
```

```
>>> for i in reversed(a_dict.keys()):
        print(i)

pet
fruit
color
```

```
>>> reversed(a_dict)
<dict_reversekeyiterator object at 0x03E8A5A0>
>>> for key in reversed(a_dict):
        print(key,'->', a_dict[key])

pet -> dog
fruit -> apple
color -> blue
>>> type(reversed(a_dict))
<class 'dict_reversekeyiterator'>
```

```
>>> def by_value(item):
        return item[1]

>>> for k, v in sorted(a_dict.items(), key=by_value):
        print(k, '→', v)

fruit → apple
color → blue
pet → dog
```

딕셔너리(dict)

언팩킹 연산자 *, ** : 도움말

- **Dictionary Unpacking Operator ** 사용하기 ➔ ** 연산자를 사용하여 여러 사전을 새 사전으로 병합**
  - **병합하려는 사전에 반복되거나 공통된 키가 있는 경우 가장 오른쪽에 있는 사전의 값이 우선함**

```
>>> fruit_prices = {'apple': 0.40, 'orange': 0.35}
>>> vegetable_prices = {'pepper': 0.20, 'onion': 0.55}
>>> {**vegetable_prices, **fruit_prices}
{'pepper': 0.2, 'onion': 0.55, 'apple': 0.4, 'orange': 0.35}
```

```
>>> vegetable_prices = {'pepper': 0.20, 'onion': 0.55}
>>> fruit_prices = {'apple': 0.40, 'orange': 0.35, 'pepper': .25}
>>> {**vegetable_prices, **fruit_prices}
{'pepper': 0.25, 'onion': 0.55, 'apple': 0.4, 'orange': 0.35}
```

```
>>> for k, v in {**vegetable_prices, **fruit_prices}.items():
        print(k, '→', v)

pepper → 0.2
onion → 0.55
apple → 0.4
orange → 0.35
```

- **iterable unpacking operator * (**tuple, list, set 등 …)

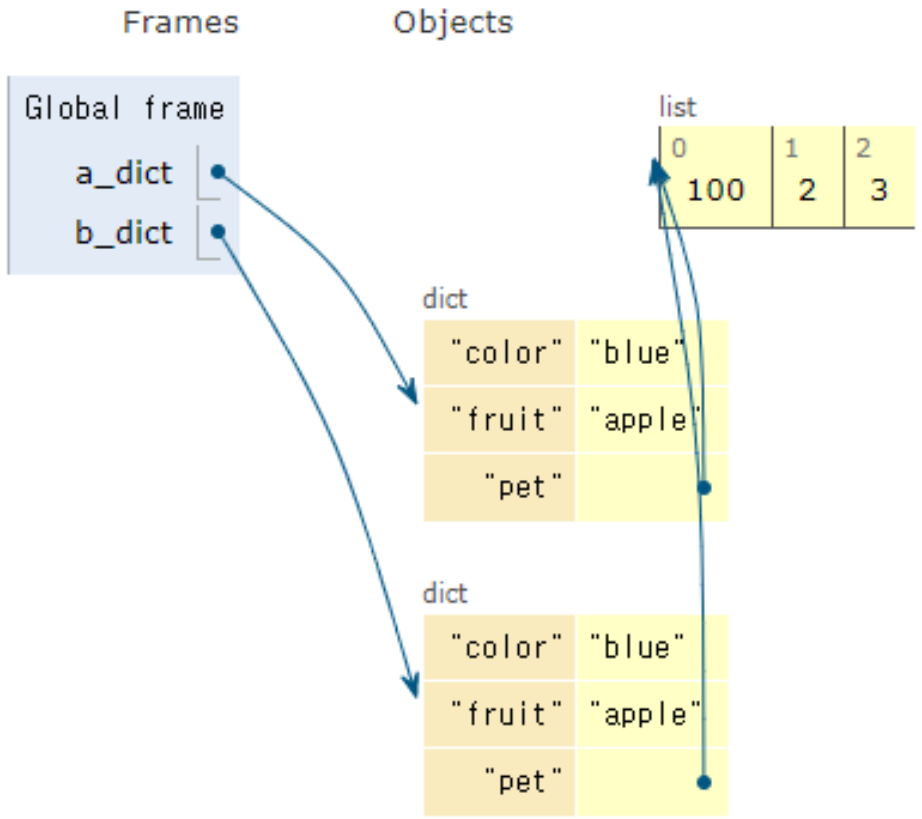
```
>>> a = {1,2,5,4}
>>> b= {10,20,30}
>>> {*a, *b}
{1, 2, 4, 5, 20, 10, 30}
>>> [*a]
[1, 2, 4, 5]
>>> [*a, *b]
[1, 2, 4, 5, 10, 20, 30]
```

```
>>> (*a, *b)
(1, 2, 4, 5, 10, 20, 30)
>>> c = range(1,10)
>>> {*a, *b, *c}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30}
```

딕셔너리(dict)

- 복사 : copy( ) method
  - Returns a **shallow copy** of the dictionary.

```
>>> a_dict = {'color': 'blue', 'fruit': 'apple', 'pet': [1,2,3]}
>>> b_dict = a_dict.copy()
>>> a_dict['pet'][0] = 100
>>> a_dict
{'color': 'blue', 'fruit': 'apple', 'pet': [100, 2, 3]}
>>> b_dict
{'color': 'blue', 'fruit': 'apple', 'pet': [100, 2, 3]}
```



Lab: 영한 사전 만들기

- 우리는 영한 사전을 구현하여 보자. 어떻게 하면 좋은가? 공백 딕셔너리를 생성하고 여기에 영어 단어를 키로 하고 설명을 값으로 하여 저장하면 될 것이다.

```
단어를 입력하시오: one
하나
```

```
단어를 입력하시오: python
없음
```

Solution

```
english_dict = dict()

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

word = input("단어를 입력하시오: ");
print (english_dict.get(word, "없음"))
```

Lab: 단어 카운터

- 사용자가 지정하는 파일을 읽어서 파일에 저장된 각각의 단어가 몇 번이나 나오는지를 계산하는 프로그램을 작성하여 보자.

```
파일 이름: proverbs.txt
{'a': 1, 'done.': 1, 'that': 1, 'well.': 1, 'ends': 1, 'Well': 1,
'flock': 1, 'feather': 1, "All's": 1, 'Birds': 1, 'together.': 1,
'of': 1, 'fast.': 1, 'begun': 1, 'half': 1, 'well': 1, 'travels':
1, 'news': 1, 'is': 1, 'Bad': 1}
```

proverbs.txt

All's well that ends well.
Bad news travels fast.
Well begun is half done.
Birds of a feather flock together.

Solution

```
fname = input("파일 이름: ")
file = open(fname, "r")

table = dict()
for line in file:
    words = line.split()
    for word in words:
        if word not in table:
            table[word] = 1
        else:
            table[word] += 1

print(table)
```

```
proverbs = '''All's well that ends well.
Bad news travels fast.
Well begun is half done.
Birds of a feather flock together.'''

table = dict()

for word in proverbs.split() :
    if word not in table:
        table[word] = 1
    else:
        table[word] += 1

print(table)
```

Lab: 축약어 풀어쓰기

- 현대인들은 축약어를 많이 사용한다. 예를 들어서 "B4(Before)" "TX(Thanks)" "BBL(Be Back Later)" "BCNU(Be Seeing You)" "HAND(Have A Nice Day)"와 같은 축약어들이 있다. 축약어를 풀어서 일반적인 문장으로 변환하는 프로그램을 작성하여 보자.

```
번역할 문장을 입력하시오: TX Mr. Park!
Thanks Mr.Park!
```

Solution

```
table = { "B4": "Before",
          "TX": "Thanks",
          "BBL": "Be Back Later",
          "BCNU":"Be Seeing You",
          "HAND":"Have A Nice Day" }

message = input('번역할 문장을 입력하시오: ')
words = message.split()
result = ""
for word in words:
    if word in table:
        result += table[word] + " "
    else:
        result += word

print(result)
```

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}
```

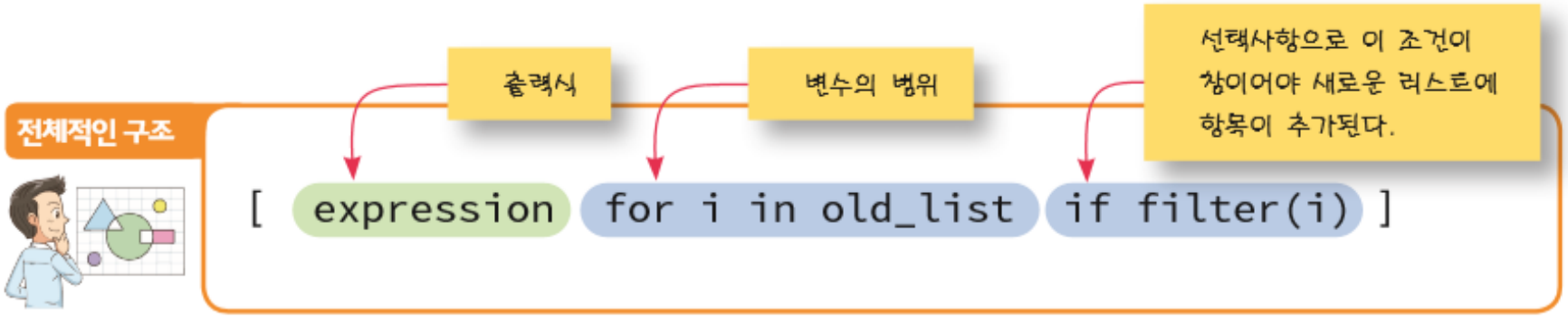
Solution

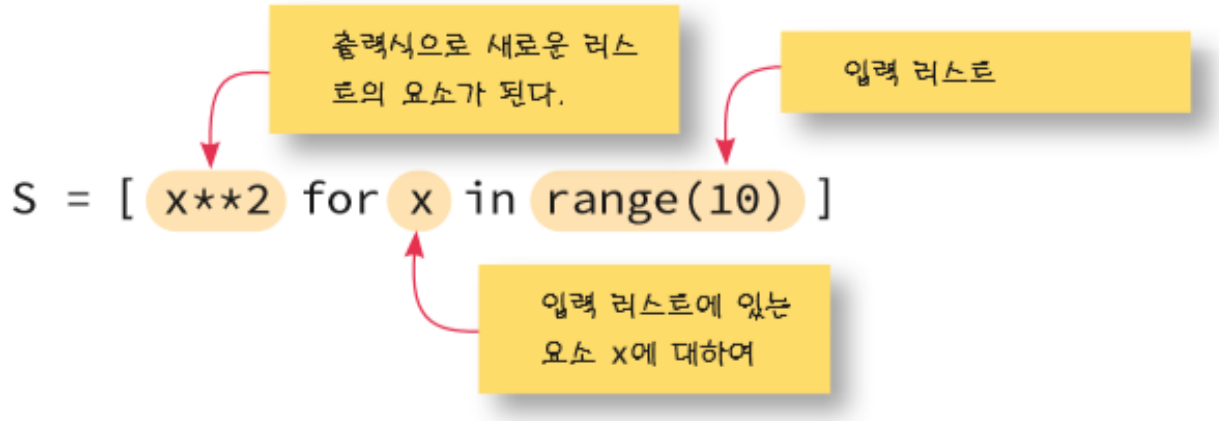
```
sentence = input("문자열을 입력하시오: ")

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

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

print(table)
```

함축 COMPREHENSION

리스트 함축(List Comprehension)

- 리스트를 수학자들이 집합을 정의하는 것과 유사하게 생성하는 것
- 한 Sequence로 부터 새로운 Sequence(iterable Object)를 만들어내는 간결한 문법





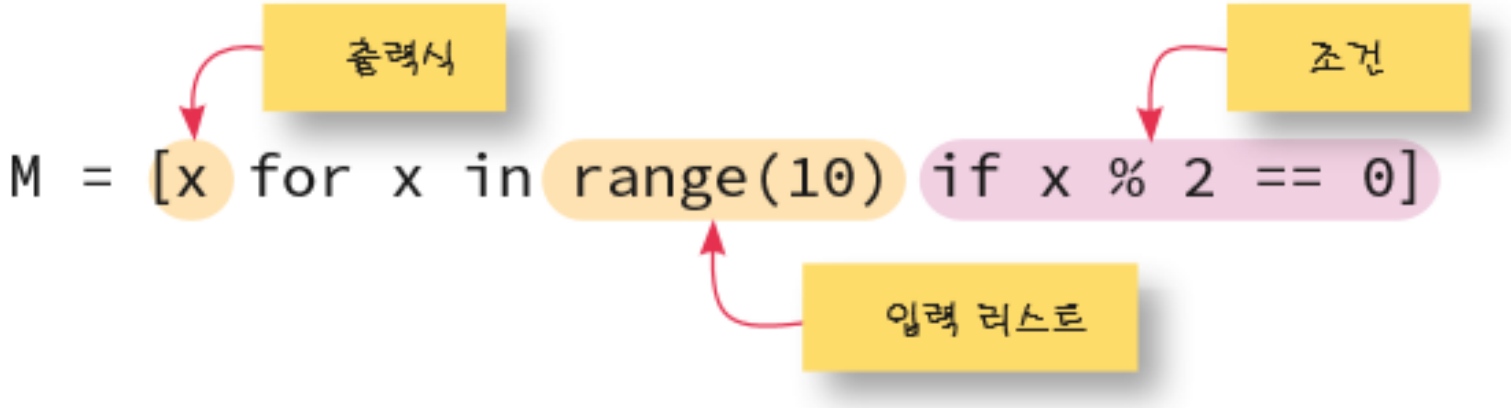
예제

```
results = []
li = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in li:
    result = i * i
    results.append(result)
print(results)
```

```
# Comprehension 으로 표현
li = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
results = [result*result for result in li]
print(results)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

조건이 붙는 리스트 함축



[0, 2, 4, 6, 8]

리스트 압축의 중복 표현

- **컴프리헨션은 내부에서 for 키워드와 if 키워드를 몇번이고 반복할 수 있다.**
  - for문 작동순서는 왼쪽 for문부터 순서대로 작동

```
>>> [ (x, y) for x in ['쌈밥', '치킨', '피자'] for y in ['사과', '아이스크림', '커피']]
[('쌈밥', '사과'), ('쌈밥', '아이스크림'), ('쌈밥', '커피'), ('치킨', '사과'), ('치킨', '아이스크림')
, ('치킨', '커피'), ('피자', '사과'), ('피자', '아이스크림'), ('피자', '커피')]
```

```
for x in ['쌈밥', '치킨', '피자']:
    for y in ['사과', '아이스크림', '커피']:
        print(x, y)
```

```
>>> [ x for x in range(10) if x < 5 if x % 2 == 0 ]
[0, 2, 4]
```

리스트 함축(List Comprehension)

if ... else With List Comprehension 예제

- i가 0~9까지의 숫자를 체크하며, i가 2로 나눈 나머지가 0이면 Even, 그 외는 Odd 로 obj list에 추가됨

```
obj = ["Even" if i%2==0 else "Odd" for i in range(10)]

print(obj)
```

```
['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']
```

리스트 함축(List Comprehension)

Python supports a ***conditional expression*** syntax that can replace a simple control structure. The general syntax is an expression of the form:

expr1 **if** *condition* **else** *expr2*

This compound expression evaluates to *expr1* if the condition is true, and otherwise evaluates to *expr2*. For those familiar with Java or C++, this is equivalent to the syntax, *condition* ? *expr1* : *expr2*, in those languages.

```
if n >= 0:
    param = n
else:
    param = -n
```

```
param = n if n >= 0 else -n
```

Set Comprehension

사용법

{ 출력표현식 for 요소 in 입력Sequence [if 조건식]}

- [if 조건식] : [ ]는 옵션 , 조건이 있을때 만 입력

```
>>> int_data = [1, 1, 2, 3, 3, 4]
>>> # 예: num * num 의 set 컬렉션 만들기
square_data_set = {num * num for num in int_data}
>>> square_data_set
{16, 1, 4, 9}
>>>
>>> # 예: num * num 의 set 컬렉션 만들기 (조건 붙여보기)
square_data_set = {num * num for num in int_data if num > 3}
>>>
>>> square_data_set
{16}
```

Dictionary Comprehension

사용법

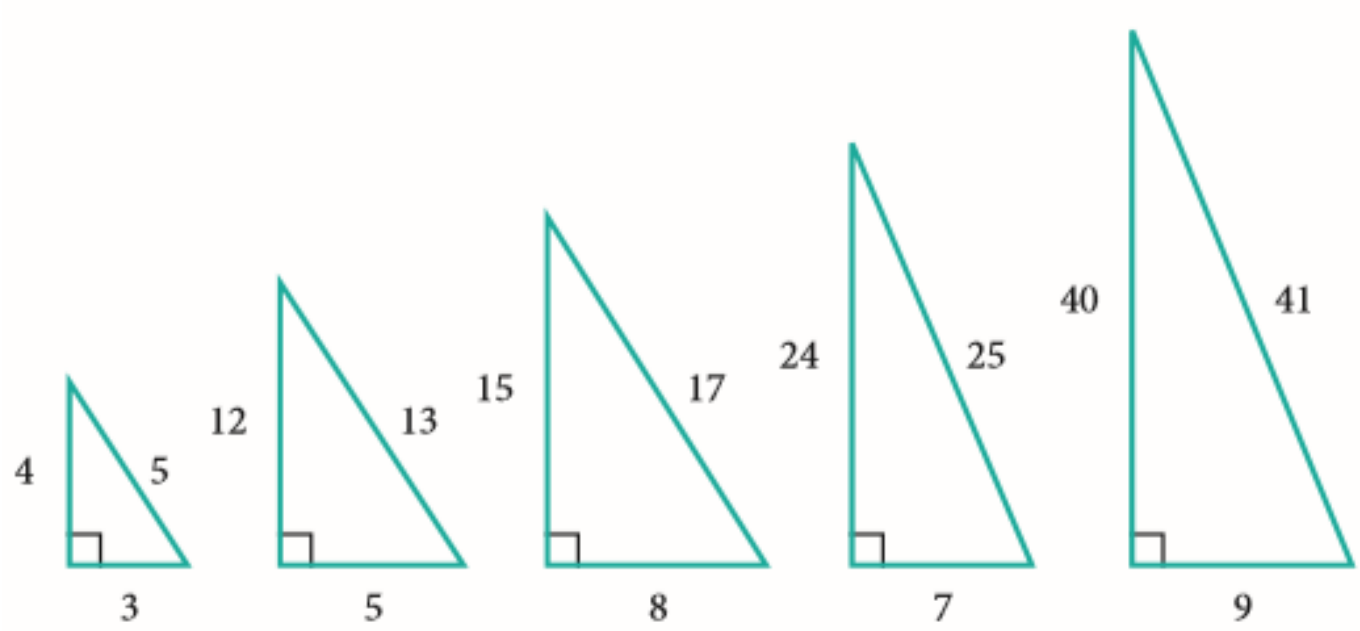
{ Key:Value for 요소 in 입력Sequence [if 조건식]}

- [if 조건식] : [ ]는 옵션 , 조건이 있을때 만 입력

```
>>> id_name = {1: 'Dave', 2: 'David', 3: 'Anthony'}
>>> id_name.items()
dict_items([(1, 'Dave'), (2, 'David'), (3, 'Anthony')])
>>>
>>> # 아이디가 1이상인 데이터를 이름:아이디 형식으로 새로운 set 만들기
name_id = {val:key for key,val in id_name.items() if key > 1}
>>> name_id
{'David': 2, 'Anthony': 3}
>>>
>>> # 아이디를 10단위로 한번에 바꾸기
name_id = {key * 10:val for key,val in id_name.items()}
>>>
>>> name_id
{10: 'Dave', 20: 'David', 30: 'Anthony'}
```

Lab: 피타고라스 삼각형

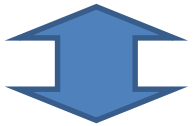
- 피타고라스의 정리를 만족하는 삼각형들을 모두 찾아보자. 삼각형 한 변의 길이는 1부터 30 이하이다.



[(3, 4, 5), (5, 12, 13), (6, 8, 10), (7, 24, 25), (8, 15, 17), (9, 12, 15), (10, 24, 26), (12, 16, 20), (15, 20, 25), (20, 21, 29)]

Solution

```
new_list = [(x,y,z) for x in range(1,30) for y in range(x,30) for z in range(y,30) if x**2 + y**2 == z**2]
print(new_list)
```



```
new_list = []
for x in range(1, 30):
        for y in range(x, 30):
                for z in range(y, 30):
                        if x**2+y**2==z**2:
                                new_list.append((x, y, z))

print(new_list)
```

Q & A