산업 인공지능 - 실습 3

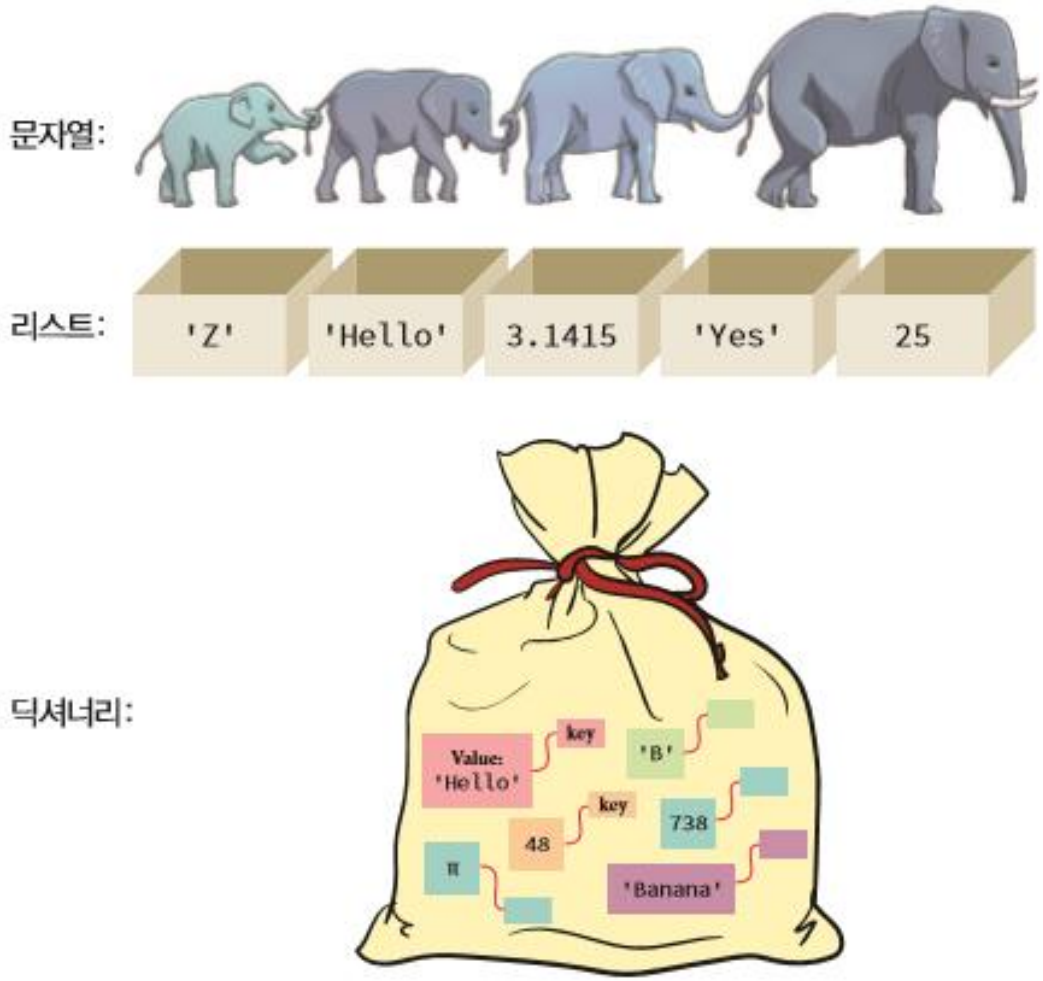
Python 프로그래밍

2021 Spring

1. 자료구조

❖ **자료구조 (data structure)**

- 프로그램에서 자료들을 저장하는 여러 가지 구조



자료구조

❖ 시퀀스(sequence)

- **요소(element)**로 **구성**되어 있고 요소 간에는 **순서**가 있는 파이썬이 제공하는 기초적인 **자료 구조**
- 각 요소는 0부터 시작하는 인덱스(index)를 통해 접근 가능

- 파이썬은 **6가지 내장 시퀀스** 제공
  - **str, bytes, bytearray, list, tuple, range**
  - 주로 사용되는 시퀀스 : 리스트(**list**), **튜플**(**tuple**)
  - 동일한 연산 지원
    - **인덱싱(indexing)**
    - **슬라이싱(slicing)**
    - **덧셈 연산(adding)**
    - **곱셈 연산(multiplying)**

2. 튜플(tuple)

❖ **튜플(tuple)**

- 만들어지고 나서 **변경될 수 없는 리스트**

튜플 = (항목1 , 항목2 , ... , 항목n)

- **튜플 생성 : 소괄호 사용**

```
>>> colors = ("red", "green", "blue")
>>> colors
('red', 'green', 'blue')

>>> numbers = (1, 2, 3, 4, 5 )
>>> numbers
(1, 2, 3, 4, 5)
```

튜플(tuple)

❖ 리스트로부터 튜플 생성

```
>>> t = tuple([1, 2, 3, 4, 5])
```

❖ 공백 튜플 생성

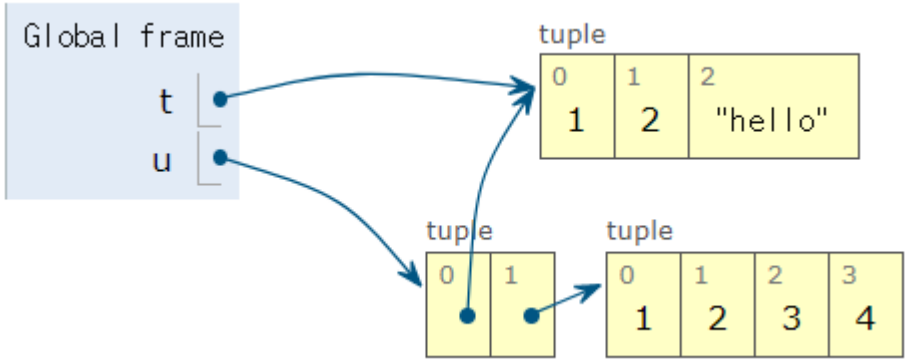
```
>>> t = ( )
```

❖ 하나의 값을 갖는 튜플 : 반드시 값 다음에 쉼표 추가

```
>>> t = (10, )
```

❖ 중첩된 튜플

```
>>> t = (1, 2, 'hello')
>>> u = t, (1, 2, 3, 4)
>>> u
((1, 2, 'hello'), (1,2,3,4))
```



튜플(tuple)

❖ 튜플은 변경 불가

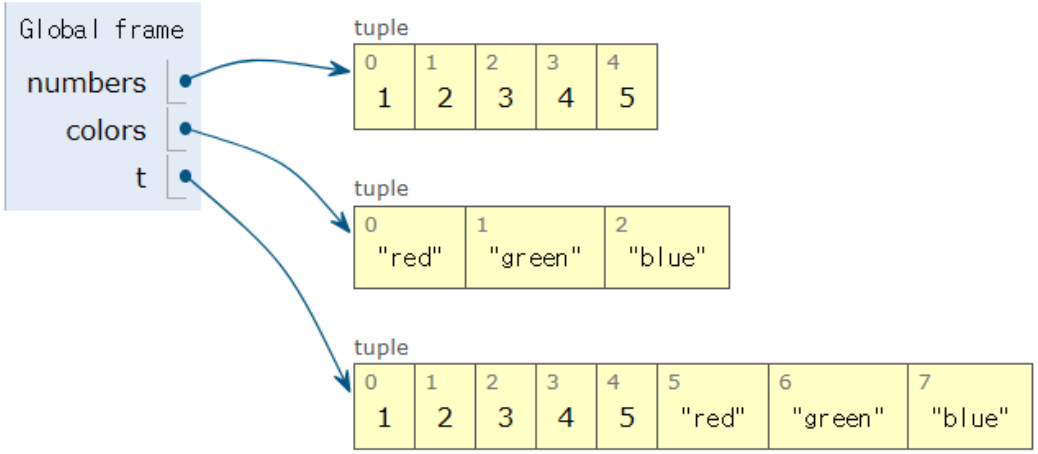
```
>>> t1 = (1, 2, 3, 4, 5);
>>> t1[0] = 100;
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    t1[0]=100
TypeError: 'tuple' object does not support item assignment
```

튜플(tuple)

❖ 튜플 연산

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

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



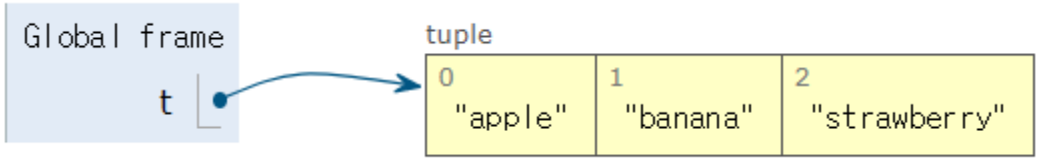
튜플(tuple)

❖ **인덱싱과 슬라이싱**

- 문자열, 리스트 등과 동일하게 동작

```
>>> t = ('apple', 'banana', 'strawberry')
>>> t[1]
banana
>>> t[-2]
banana
>>> t[1:]
('banana', 'strawberry')
```

음수 인덱스는 오른쪽 끝에서 시작

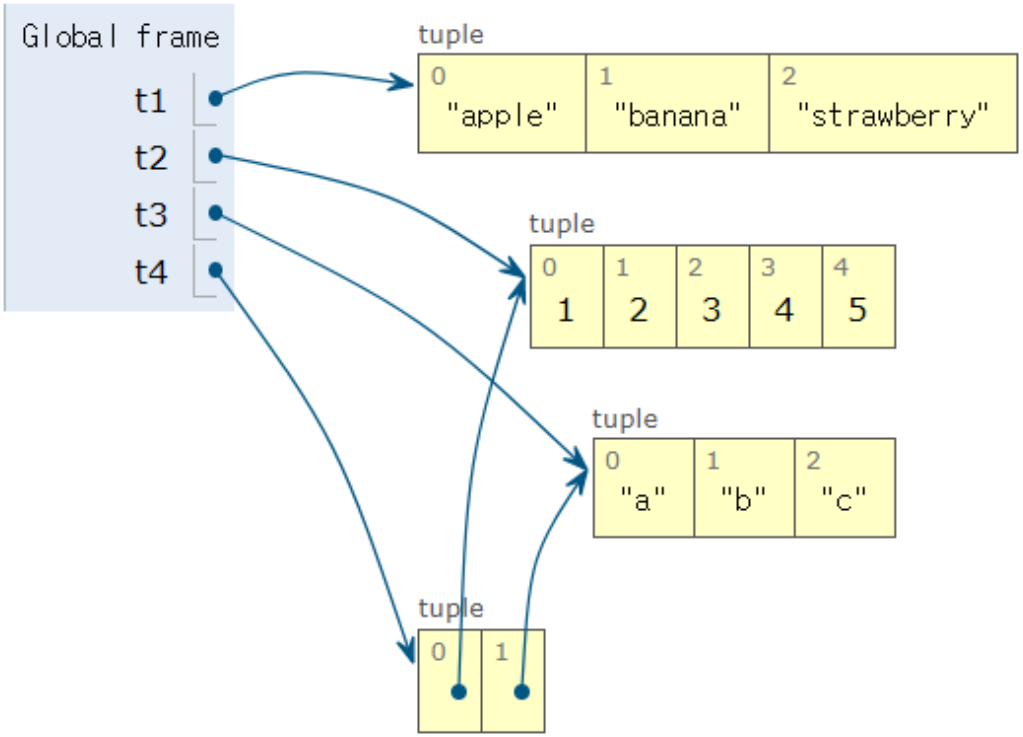


튜플(tuple)

❖ 괄호가 없는 튜플

- 괄호없이 나열된 객체는 튜플로 간주

```
>>> t1 = 'apple', 'banana', 'strawberry'
>>> t2 = 1, 2, 3, 4, 5
>>> t3 = "a", "b", "c"
>>> t4 = t2, t3
```

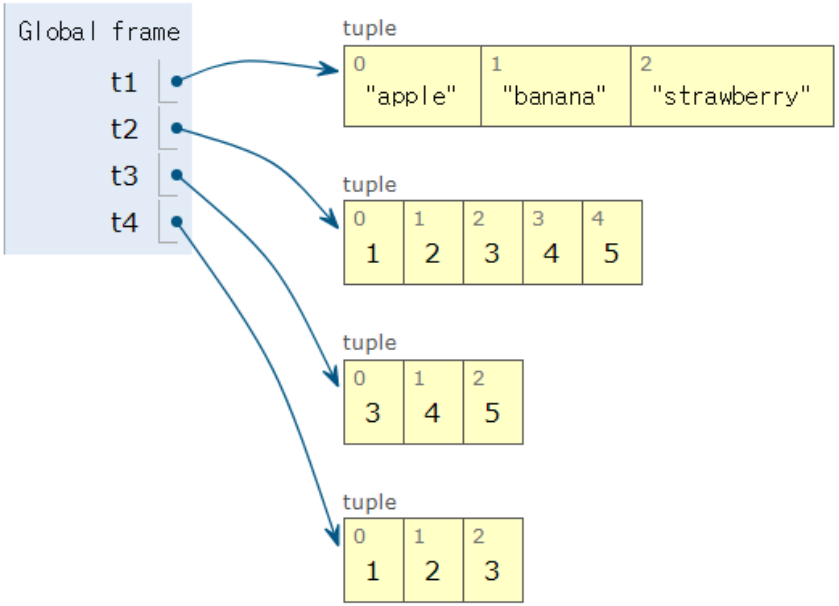


튜플(tuple)

❖ 튜플의 내장 함수

함수	설명
t1 > t2	2개의 튜플을 비교한다.
len(t)	튜플의 길이를 반환한다.
max(t)	튜플에 저장된 최대값을 반환한다.
min(t)	튜플에 저장된 최소값을 반환한다.
tuple(seq)	리스트를 튜플로 변환한다.

```
>>> t1 = 'apple', 'banana', 'strawberry'
>>> t2 = 1, 2, 3, 4, 5
>>> t3 = 3, 4, 5
>>> t2>t3
False
>>> len(t1)
3
>>> max(t2)
5
>>> min(t2)
1
>>> t4 = tuple([1, 2, 3])
```

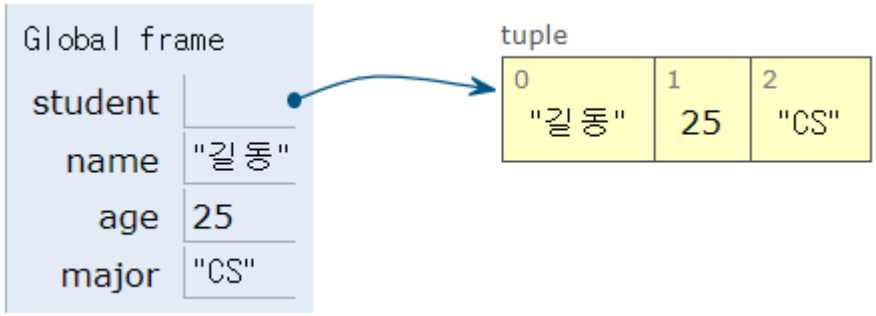


튜플(tuple)

❖ 튜플 대입 연산 (tuple assignment)

- 여러 개의 변수로 한 번에 값을 대입하는 기능

```
>>> student = ('길동', 25, 'CS')
>>> (name, age, major) = student
```



- **튜플 패킹(tuple packing)**
  - 튜플에 값을 저장하는 과정

- **튜플 언패킹(tuple unpacking)**
  - 튜플에서 값을 꺼내서 변수에 대입하는 과정

예: 원의 면적과 둘레 계산

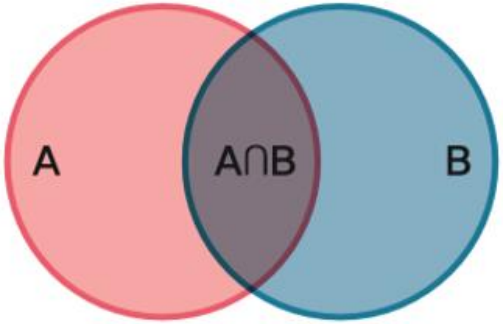
```
원의 반지름을 입력하시오: 10
원의 넓이는 314.1592653589793이고 원의 둘레는 62.83185307179586이다.
```

```
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.")
```

3. 세트(set, 집합)

❖ **세트(set)**

- **집합**을 나타내는 자료구조
- 중복되지 않은 항목들이 모인 것
- 항목 간에는 순서가 없음

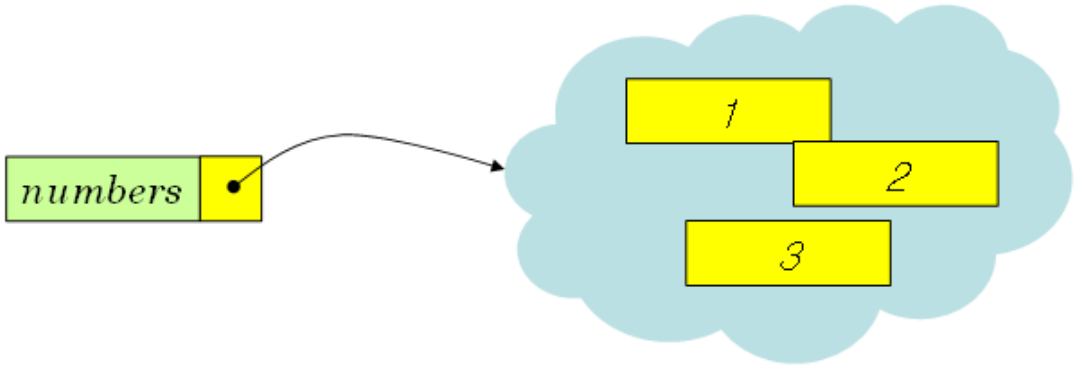


세트 = { 항목1 , 항목2 , ... , 항목n }

세트(set)

❖ 세트의 정의

```
>>> numbers = {2, 1, 3}
>>> numbers
{1, 2, 3}

>>> len(numbers)
3

>>> fruits = { "Apple", "Banana", "Pineapple" }
>>> mySet = { 1.0, 2.0, "Hello World", (1, 2, 3) }
```



세트(set)

❖ in 연산자

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


집합 안에 1이 있습니다.
```

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


1 2 3
```

세트(set)

❖ 세트에 요소 추가 및 삭제 하기

- 순서가 없기 때문에 인덱싱 불가
- add() 사용 요소 추가
- update() 사용 여러 요소 추가
- remove() / discard() 사용 요소 삭제
- clear() 사용 전체 요소 삭제

```
>>> numbers = { 2, 1, 3 }
>>> numbers[0]
...
TypeError: 'set' object does not support indexing

>>> numbers.add(4)
>>> numbers
{1, 2, 3, 4}
>>> numbers.update([5,6])
{1, 2, 3, 4, 5, 6}
>>> numbers.remove(5)
{1, 2, 3, 4, 6}
>>> numbers.discard(1)
{2, 3, 4, 6}
```

세트(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

>>> B.isuperset(A)
False
```

세트(set)

❖ 집합 연산

```
>>> A = {1, 2, 3}
>>> B = {3, 4, 5}

>>> A | B
{1, 2, 3, 4, 5}
>>> A.union(B)
{1, 2, 3, 4, 5}

>>> A & B
{3}
>>> A.intersection(B)
{3}

>>> A - B
{1, 2}
>>> A.difference(B)
{1,2}
```

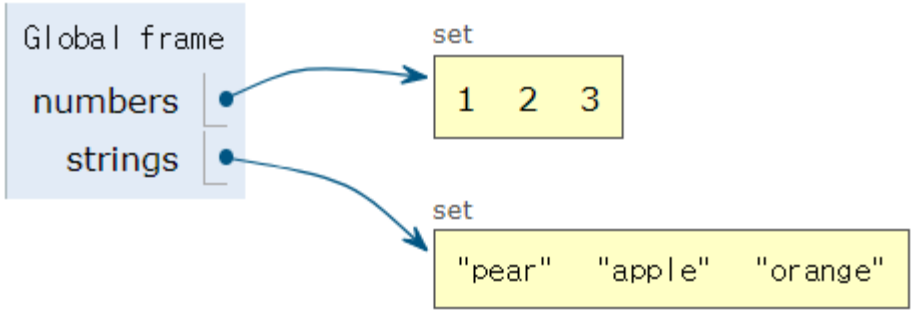
세트(set)

❖ 세트의 저장

- 해싱(hashing)을 이용하여 저장하고 관리
  - **해싱** : 객체를 식별할 수 있는 숫자코드를 생성하여 객체 관리

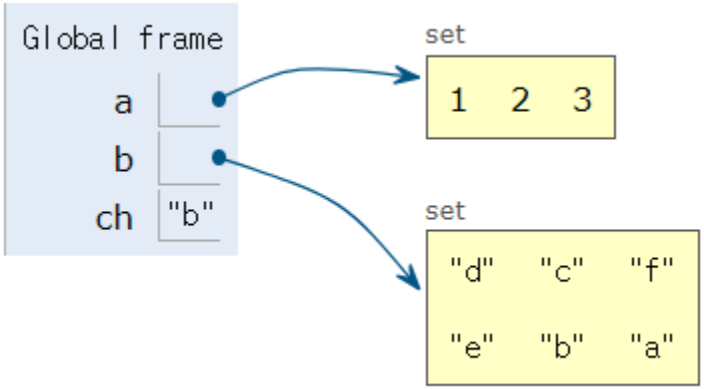
- **해싱가능**(hashable) 요소만 허용

```
>>> numbers = {1, 2, 3}
>>> strings = {"apple", "orange", "pear"}
>>> wrong = {1, 2, [3,4]}    # 오류
```



세트(set)

```
>>> a = set([1,2,3,1,2,3])
>>> b = set("abcdefab")
>>> for ch in set("banana"):
    print(ch)
a
n
b
```



세트(set)

```
>>> any(l == 't' for l in 'python')
True

>>> all(l == 't' for l in 'python')
False
```

4. 딕셔너리(dictionary)

❖ **딕셔너리(dictionary, 사전)**

- **키(key)**와 **값(value)**의 쌍을 저장할 수 있는 객체

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

딕셔너리 = { 키1 : 값1 , 키2 : 값2 , ... }

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

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

딕셔너리(dictionary)

❖ 디셔너리 생성 및 접근

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

>>> contacts['Kim']
'01012345678’

>>> contacts.get('Kim')
'01012345678'
```

❖ 항목 확인

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

딕셔너리(dictionary)

❖ 항목 추가

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

❖ 항목 삭제

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

>>> contacts.pop("Kim")
'01012345678'

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

딕셔너리(dictionary)

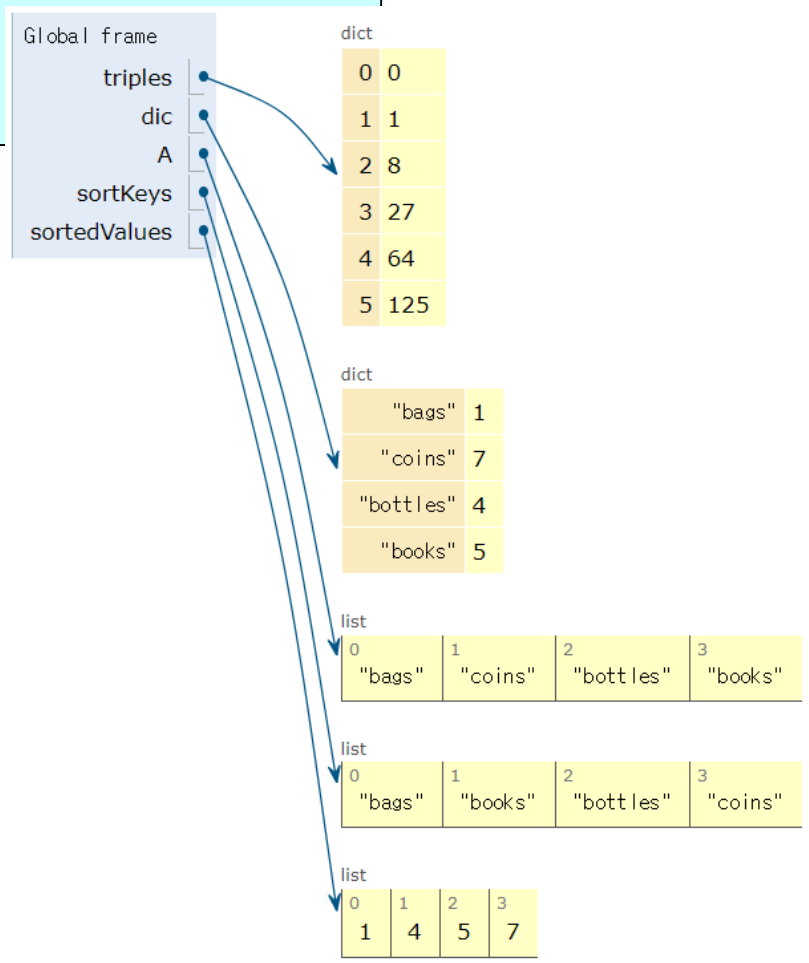
❖ 항목 순회하기

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


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

딕셔너리(dictionary)

❖ 정렬

```
>>> triples = { x: x*x*x for x in range(6)}
>>> dic = {"bags":1, "coins":7, "bottles":4, "books":5}
>>> A = list(dic)
>>> sortKeys = sorted(dic)
>>> sortedValues = sorted(dic.values())
```



예. 영한 사전

```
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"))
```

```
단어를 입력하시오: one
하나
```

```
단어를 입력하시오: python
없음
```

예. 단어 세기

```
파일 이름: proverbs.txt
Birds of a feather flock together
Bad news travel fast
Well begun is half done
All's well that ends well
```

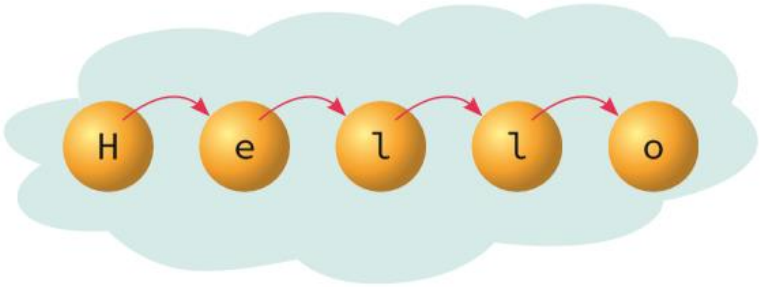
```
{'a': 1, 'done.': 1, 'that': 1, 'well.': 1, 'ends': 1, 'Well': 1, 'flock': 1, 'feather': 1, "All's": 1, 'Birds': 1, 'together.': 1, 'of': 1, 'fast.': 1, 'begun': 1, 'half': 1, 'well': 1, 'travels': 1, 'news': 1, 'is': 1, 'Bad': 1}
```

```
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)
```

5. 문자열 (string)

❖ **문자열 (string)**

- 문자들의 시퀀스
- "hello" 'hello'




```
s1 = str("Hello")
s2 = "Hello"

s3 = "Hello"
s4 = "World"
s5 = "Hello "+"World"
```

Global frame

s1	"Hello"
s2	"Hello"
s3	"Hello"
s4	"World"
s5	"Hello World"

문자열

❖ 개별 문자 접근하기

```
>>> word = 'abcdef'
>>> word[0]
'a'
>>> word[5]
'f'
```

a	b	c	d	e	f
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

문자열

❖ 슬라이싱

```
>>> word = 'Python'
>>> word[0:2]
'Py'
>>> word[2:5]
'tho’
```

문자열

❖ in 연산자와 not in 연산자

```
>>> s = "Love will find a way."
>>> "Love" in s
True
>>> "love" in s
False
```

```
s = input("문자열을 입력하시오")

if 'c' in s:
    print('c가 포함되어 있음')
else:
    print('c가 포함되어 있지 않음')
```

문자열

❖ 문자열 비교하기

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

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

문자열

❖ 문자열에서 단어 분리

```
>>> s = 'Never put off till tomorrow what you can do today.'
>>> s.split()
['Never', 'put', 'off', 'till', 'tomorrow', 'what', 'you',
'can', 'do', 'today.']
```

예: CSV 파일 분석

```
# 파일을 오픈한다.

f = open("E:\\students.txt", "r")

# 파일의 각 줄에 대하여 반복한다.
for line in f.readlines():

    # 양 끝의 공백 문자를 제거한다.
    line = line.strip()

    # 줄을 출력한다.
    print(line)

    # 줄을 단어로 분리한다.
    words = line.split(",")

    # 줄의 단어를 출력한다.
    for word in words:
        print("   ", word)
```

홍길동,2018,생활관A,312,3.18
김철수,2017,생활관B,102,3.25

```
홍길동,2018,생활관A,312,3.18
홍길동
2018
생활관A
312
3.18
김철수,2017,생활관B,102,3.25
김철수
2017
생활관B
102
3.25
...
```

Durable Rules

규칙기반 시스템

프로그래밍 실습: 규칙기반 시스템

❖ **Durable Rules 패키지**

- Redis DB 상에서 RETE 구조를 C언어로 구현한 규칙기반 시스템
- Python, Node.js, Ruby 등 지원

```
!pip install durable_rules
```

```
from durable.lang import *

with ruleset('testRS'):  # 규칙집합
    # antecedent(조건부). @when_all, @when_any를 사용하여 표기
    @when_all(m.subject == 'World')  # m: rule이 적용되는 데이터
    def say_hello(c):
        # consequent (결론부)
        print ('Hello {0}'.format(c.m.subject))

post('testRS', { 'subject': 'World' }) # 규칙 집합에 데이터 'subject': 'World' 전달
```

```
Hello World
{'$s': 1, 'id': 'sid-0', 'sid': '0'}
```

#https://github.com/jruizgit/rules/blob/master/docs/py/reference.md

```python
from durable.lang import *

with ruleset('animal'):
    @when_all(c.first << (m.predicate == 'eats') & (m.object == 'flies'),  # << 해당 조건을 만족하는 대상 지시하는 이름
              (m.predicate == 'lives') & (m.object == 'water') & (m.subject == c.first.subject))
    def frog(c):
        c.assert_fact({ 'subject': c.first.subject, 'predicate': 'is', 'object': 'frog' })
        # 사실(fact)의 추가

    @when_all(c.first << (m.predicate == 'eats') & (m.object == 'flies'),
              (m.predicate == 'lives') & (m.object == 'land') & (m.subject == c.first.subject))
    def chameleon(c):
        c.assert_fact({ 'subject': c.first.subject, 'predicate': 'is', 'object': 'chameleon' })

    @when_all((m.predicate == 'eats') & (m.object == 'worms'))
    def bird(c):
        c.assert_fact({ 'subject': c.m.subject, 'predicate': 'is', 'object': 'bird' })

    @when_all((m.predicate == 'is') & (m.object == 'frog'))
    def green(c):
        c.assert_fact({ 'subject': c.m.subject, 'predicate': 'is', 'object': 'green' })

    @when_all((m.predicate == 'is') & (m.object == 'chameleon'))
    def grey(c):
        c.assert_fact({ 'subject': c.m.subject, 'predicate': 'is', 'object': 'grey' })

    @when_all((m.predicate == 'is') & (m.object == 'bird'))
    def black(c):
        c.assert_fact({ 'subject': c.m.subject, 'predicate': 'is', 'object': 'black' })

    @when_all(+m.subject)  # m.subject가 한번 이상
    def output(c):
        print('Fact: {0} {1} {2}'.format(c.m.subject, c.m.predicate, c.m.object))

assert_fact('animal', { 'subject': 'Kermit', 'predicate': 'eats', 'object': 'flies' })
```

```
35
36 assert_fact('animal', { 'subject': 'Kermit', 'predicate': 'eats', 'object': 'flies' })
37 assert_fact('animal', { 'subject': 'Kermit', 'predicate': 'lives', 'object': 'water' })
38 assert_fact('animal', { 'subject': 'Greedy', 'predicate': 'eats', 'object': 'flies' })
39 assert_fact('animal', { 'subject': 'Greedy', 'predicate': 'lives', 'object': 'land' })
40 assert_fact('animal', { 'subject': 'Tweety', 'predicate': 'eats', 'object': 'worms' })
```

```
Fact: Kermit eats flies
Fact: Kermit is green
Fact: Kermit is frog
Fact: Kermit lives water
Fact: Greedy eats flies
Fact: Greedy is grey
Fact: Greedy is chameleon
Fact: Greedy lives land
Fact: Tweety is black
Fact: Tweety is bird
Fact: Tweety eats worms
{'$s': 1, 'id': 'sid-0', 'sid': '0'}
```

```
with ruleset('risk'):
    @when_all(c.first << m.t == 'purchase',
              c.second << m.location != c.first.location)
    def fraud(c):
        print('이상거래 탐지 -> {0}, {1}'.format(c.first.location, c.second.location))

post('risk', {'t': 'purchase', 'location': 'US'})
post('risk', {'t': 'purchase', 'location': 'CA'})
```

```
이상거래 탐지 -> CA, US
{'$s': 1, 'id': 'sid-0', 'sid': '0'}
```

```
with ruleset('bookstore'):
    @when_all(+m.status) # status를 갖는 것에 대해서 실행되는 규칙
    def event(c):
        print('bookstore-> Reference {0} status {1}'.format(c.m.reference, c.m.status))

    @when_all(+m.name)
    def fact(c):
        print('bookstore-> Added "{0}"'.format(c.m.name))

    @when_all(none(+m.name)) # name이 없는 것(삭제되는 것)에 호출
    def empty(c):
        print('bookstore-> No books')

# 새로운 fact 추가하는 경우
assert_fact('bookstore', {
    'name': 'The new book',
    'seller': 'bookstore',
    'reference': '75323',
    'price': 500
})

# 기존의 fact를 다시 추가하는 경우 MessageObservedError 발생
try:
    assert_fact('bookstore', {
        'reference': '75323',
        'name': 'The new book',
        'price': 500,
        'seller': 'bookstore'
    })
except BaseException as e:
    print('Error: {0}'.format(e.message))

```

```
post('bookstore', {
    'reference': '75323',
    'status': 'Active'
})

retract_fact('bookstore', {
    'reference': '75323',
    'name': 'The new book',
    'price': 500,
    'seller': 'bookstore'
})
```

```
bookstore-> Added "The new book"
Error: Message has already been observed: {"reference": "75323", "name": "The new book
bookstore-> Reference 75323 status Active
bookstore-> No books
{'$s': 1, 'id': 'sid-0', 'sid': '0'}
```