12. 자료 구조

자료 구조란 간단하게, 어떤 *자료*를 담는 *구조*를 말합니다. 다른 말로 하면, 서로 연관있는 어떤 자료들의 집합을 저장하는 데 사용됩니다.

파이썬에는 네 종류의 자료 구조가 있는데, 각각 _리스트, 튜플, 사전(dict), 집합(set)입니다. 이제 앞으로 각각의 사용법에 대해 알아보고 또 각각이 얼마나 편리한지 확인해보도록 하겠습니다.

12.1. 리스트

리스트란 순서대로 정리된 항목들을 담고 있는 자료 구조입니다. 즉, 리스트에는 항목의 *목록*을 저장할 수 있습니다. 이것은 쉽게 말하자면 장 보러 갈 때 적는 일종의 장바구니 목록 같은 것인데, 아마도 여러분은 각 품목들을 한줄 한줄 적겠지만 파이썬에서는 쉼표로 각 항목을 구분한다는 것만 다릅니다.

리스트를 정의할 때는 대괄호 [] 를 이용해서 파이썬에게 이것이 리스트를 의미한다는 것을 알려줍니다. 한번 리스트를 만들어 두면 여기에 새로운 항목을 추가하거나 삭제할 수 있으며, 특정 항목이 존재하는지 검색할 수도 있습니다. 이 때 항목을 추가 및 삭제가 가능하다는 것을 *비정적(mutable)*이라고 하며, 리스트는 비정적 자료구조로 내부 항목을 변경할 수 있습니다.

List.append: List의 마지막에 원소를 추가

```
In [384]: b_list.append('dwarf')
In [385]: b_list
Out[385]: ['foo', 'peekaboo', 'baz', 'dwarf']
```

List.insert(위치,원소): 원하는 위치에 원소를 추가

```
In [386]: b_list.insert(1, 'red')
In [387]: b_list
Out[387]: ['foo', 'red', 'peekaboo', 'baz', 'dwarf']
```

List.pop(인덱스) : 인덱스 위치의 원소를 삭제 (비효율적 방법)

```
In [388]: b_list.pop(2)
Out[388]: 'peekaboo'
In [389]: b_list
```

```
Out[389]: ['foo', 'red', 'baz', 'dwarf']
```

List.remove(원소) : 리스트에 있는 특정 원소 중 가장 앞에 있는 값을 리스트에서 삭제

```
In [390]: b_list.append('foo')
In [391]: b_list.remove('foo')
In [392]: b_list
Out[392]: ['red', 'baz', 'dwarf', 'foo']
```

리스트 + 리스트 = 긴리스트

```
In [394]: [4, None, 'foo'] + [7, 8, (2, 3)]
Out[394]: [4, None, 'foo', 7, 8, (2, 3)]
```

List.sort() : 리스트를 정렬한다.

```
In [398]: a = [7, 2, 5, 1, 3]

In [399]: a.sort()
In [400]: a
Out[400]: [1, 2, 3, 5, 7]
```

Sort(key=) sort에서 key를 사용하여 정렬방식을 바꿔줄 수 있다.

```
In [401]: b = ['saw', 'small', 'He', 'foxes', 'six']
In [402]: b.sort(key=len)
In [403]: b
Out[403]: ['He', 'saw', 'six', 'small', 'foxes']
```

bisect.bisect(list. 원소) : list에 원소를 어디에 넣어야 정렬이 유지되는지 주소값 출력

bisect.insort(list. 원소): 정렬이 유지되게 list에 원소를 삽입

```
In [404]: import bisect
In [405]: c = [1, 2, 2, 2, 3, 4, 7]
In [406]: bisect.bisect(c, 2)
Out[406]: 4
In [407]: bisect.bisect(c, 5)
Out[407]: 6
In [408]: bisect.insort(c, 6)
In [409]: c
Out[409]: [1, 2, 2, 2, 3, 4, 6, 7]
```

Sequence의 범위로 주소 지정하기

```
In [410]: seq = [7, 2, 3, 7, 5, 6, 0, 1]
In [411]: seq[1:5]
Out[411]: [2, 3, 7, 5]
```

A `step` can also be used after a second colon to, say, take every other element:

```
In [418]: seq[::2]
Out[418]: [7, 3, 3, 6, 1]
```

A clever use of this is to pass `-1` which has the useful effect of reversing a list or tuple:

```
In [419]: seq[::-1]
Out[419]: [1, 0, 6, 5, 3, 6, 3, 2, 7]
```

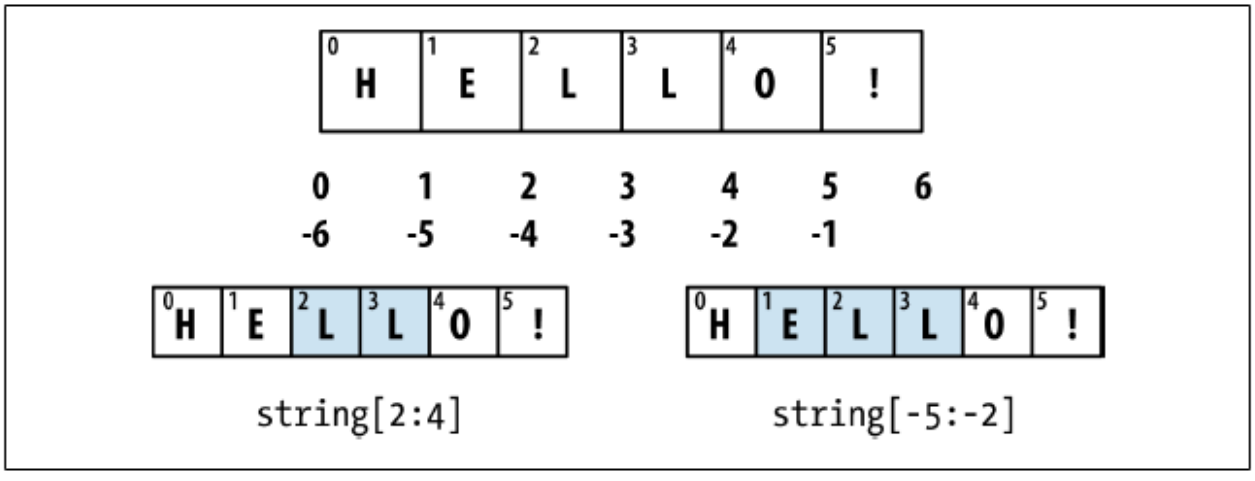

Figure A-2. Illustration of Python slicing conventions

enumerate

list에서 어떤 명령을 반복할 때 반복수를 계산하기 위해서 직접 아래와 같은 구문을 사용할 수 있다.

```
i = 0
for value in collection:
# do something with value
i += 1
```

Enumerate: 어떤 구문의 반복 시행시 반복수를 index로 자동으로 구해준다.

```
for i, value in enumerate(collection):
# do something with value
```

Dict를 mapping하는데 응용할 수 있다.

```
In [420]: some_list = ['foo', 'bar', 'baz']
In [421]: mapping = dict((v, i) for i, v in enumerate(some_list))
In [422]: mapping
Out[422]: {'bar': 1, 'baz': 2, 'foo': 0}
```

Sorted: 어떤 배열을 정렬된 상태의 새배열로 만들어 준다.

```
In [423]: sorted([7, 1, 2, 6, 0, 3, 2])
Out[423]: [0, 1, 2, 2, 3, 6, 7]
In [424]: sorted('horse race')
Out[424]: [' ', 'a', 'c', 'e', 'e', 'h', 'o', 'r', 'r', 's']

In [425]: sorted(set('this is just some string'))
Out[425]: [' ', 'e', 'g', 'h', 'i', 'j', 'm', 'n', 'o', 'r', 's', 't', 'u']
```

zip

zip은 서로 다른 두 list, set으로부터 각 원소가 짝지어진 새로운 tuple의 조합을 가지는 배열을 만들어준다.

```
In [426]: seq1 = ['foo', 'bar', 'baz']
In [427]: seq2 = ['one', 'two', 'three']
In [428]: zip(seq1, seq2)
Out[428]: [('foo', 'one'), ('bar', 'two'), ('baz', 'three')]
```

만약 여러 sequence들의 크기가 다르면 가장 작은 크기의 배열을 따른다.

```
In [429]: seq3 = [False, True]
In [430]: zip(seq1, seq2, seq3)
Out[430]: [('foo', 'one', False), ('bar', 'two', True)]
```

example

```
In [431]: for i, (a, b) in enumerate(zip(seq1, seq2)):
.....: print('%d: %s, %s' % (i, a, b))
.....:
0: foo, one
1: bar, two
2: baz, three
```

Zip을 이용하여 짝지어진 배열을 다시 푸는 것에 사용할 수 있다.

```
In [432]: pitchers = [('Nolan', 'Ryan'), ('Roger', 'Clemens'),
.....: ('Schilling', 'Curt')]
In [433]: first_names, last_names = zip(*pitchers) # *은 pitchers안의 모든 원소 의미
In [434]: first_names
Out[434]: ('Nolan', 'Roger', 'Schilling')
In [435]: last_names
Out[435]: ('Ryan', 'Clemens', 'Curt')
```

reversed

list안의 원소들을 거꾸로 배치해준다.

```
In [436]: list(reversed(range(10)))
Out[436]: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

예제 (ds_using_list.py 로 저장하세요):

```
# This is my shopping list

shoplist = ['apple', 'mango', 'carrot', 'banana']



print 'I have', len(shoplist), 'items to purchase.'
```

```
print 'These items are:',
for item in shoplist:
    print item,

print '₩nI also have to buy rice.'
shoplist.append('rice')
print 'My shopping list is now', shoplist

print 'I will sort my list now'
shoplist.sort()
print 'Sorted shopping list is', shoplist

print 'The first item I will buy is', shoplist[0]
olditem = shoplist[0]
del shoplist[0]
print 'I bought the', olditem
print 'My shopping list is now', shoplist
My shopping list is now ['banana', 'carrot', 'mango', 'rice']
```

12.3. 튜플

튜플은 여러 개의 객체를 모아 담는 데 사용됩니다. 튜플은 리스트와 비슷하지만, 리스트 클래스에 있는 여러가지 기능이 없습니다. 또 튜플은 수정이 불가능하며, 그래서 주로 문자열과 같이 *비정적*인 객체들을 담을 때 사용됩니다.

튜플은 생략할 수 있는 괄호로 묶인 쉼표로 구분된 여러 개의 항목으로 정의됩니다.

```
In [356]: tup = 4, 5, 6
```

```
In [357]: tup

Out[357]: (4, 5, 6)



In [358]: nested_tup = (4, 5, 6), (7, 8)

In [359]: nested_tup

Out[359]: ((4, 5, 6), (7, 8))
```

튜플에 저장된 값들은 수정이 불가능하기 때문에, 단순 값들의 목록을 다루는 구문이나 사용자 정의 함수에서 주로 사용됩니다.

```
In [364]: tup = tuple(['foo', [1, 2], True])

In [365]: tup[2] = False

---------------------------------------------------------------------------

TypeError Traceback (most recent call last)

<ipython-input-365-c7308343b841> in <module>()

----> 1 tup[2] = False

TypeError: 'tuple' object does not support item assignment
```

 # however 아래와 같은 식으로 추가할 수 있음

```
In [366]: tup[1].append(3)

In [367]: tup

Out[367]: ('foo', [1, 2, 3], True)
```

Tuple 덧셈

```
In [368]: (4, None, 'foo') + (6, 0) + ('bar',)

Out[368]: (4, None, 'foo', 6, 0, 'bar')
```

Tuple 곱셈

```
In [369]: ('foo', 'bar') * 4

Out[369]: ('foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'bar')
```

예제

```
# I would recommend always using parentheses

# to indicate start and end of tuple

# even though parentheses are optional.

# Explicit is better than implicit.


zoo = ('python', 'elephant', 'penguin')

print 'Number of animals in the zoo is', len(zoo)

new_zoo = 'monkey', 'camel', zoo

print 'Number of cages in the new zoo is', len(new_zoo)

print 'All animals in new zoo are', new_zoo

print 'Animals brought from old zoo are', new_zoo[2]

print 'Last animal brought from old zoo is', new_zoo[2][2]

print 'Number of animals in the new zoo is', ₩

      len(new_zoo)-1+len(new_zoo[2])
```

실행 결과:

```
$ python ds_using_tuple.py

Number of animals in the zoo is 3

Number of cages in the new zoo is 3

All animals in new zoo are ('monkey', 'camel', ('python', 'elephant', 'penguin'))
```

```
Animals brought from old zoo are ('python', 'elephant', 'penguin')

Last animal brought from old zoo is penguin

Number of animals in the new zoo is 5
```

NOTE 빈 튜플과 한 개짜리 튜플

빈 튜플은 괄호 안에 아무것도 적지 않고 myempty = () 와 같이 생성할 수 있습니다. 그러나, 항목 한 개만 담고 있는 튜플을 정의할 때는 주의해야 합니다. 이 경우 첫 번째 항목의 뒤에 쉼표를 붙여 주어 파이썬에게 이것이 숫자 연산에 사용되는 괄호가 아니라 객체를 담는 튜플을 의미하는 것이라는 것을 구분할 수 있도록 단서를 주어야 합니다. 예를 들어, 항목 2 를 담고 있는 튜플을 정의하려면 singleton = (2 ,) 와 같이 합니다.

12.4. 사전(Dict)

사전은 이를테면 전화번호부 같은 것인데, 누군가의 이름을 찾으면 그 사람의 주소와 연락처를 알 수 있는 것과 같습니다. 이 때 그 사람의 이름에 해당하는 것을 키 라 부르고, 주소와 연락처 등에 해당하는 것을 값 이라 부릅니다. 전화번호부에 동명이인이 있을 경우 어떤 정보가 맞는 정보인지 제대로 알아낼 수 없듯이, 사전의 키는 사전에서 유일한 값이어야 합니다.

사전의 키는 정적 객체(문자열 등등)이어야 하지만, 값으로는 정적 객체나 비정적 객체 모두 사용할 수 있습니다. 이것을 간단하게 다시 말하면 사전의 키로는 단순 객체만 사용할 수 있다고 표현합니다.

사전을 정의할 때 키와 값의 쌍은 d = {key1 : value1, key2 : value2 } 와 같이 지정해 줍니다. 이 때 키와 값은 콜론으로 구분하며 각 키-값 쌍은 쉼표로 구분하고 이 모든 것을 중괄호 `{}`로 묶어 준다는 것을 기억하시기 바랍니다.

```
In [437]: empty_dict = {}
In [438]: d1 = {'a' : 'some value', 'b' : [1, 2, 3, 4]}



In [440]: d1[7] = 'an integer'
In [441]: d1
Out[441]: {7: 'an integer', 'a': 'some value', 'b': [1, 2, 3, 4]}
```

```
In [442]: d1['b']
Out[442]: [1, 2, 3, 4]

In [443]: 'b' in d1
Out[443]: True
```

```
In [449]: d1.keys() In [450]: d1.values()
Out[449]: ['a', 'b', 7] Out[450]: ['some value', [1, 2, 3, 4], 'an integer']
```

```
In [451]: d1.update({'b' : 'foo', 'c' : 12})
In [452]: d1
Out[452]: {7: 'an integer', 'a': 'some value', 'b': 'foo', 'c': 12}
```

여기서 사전의 키-값 쌍은 자동으로 정렬되지 않습니다. 이를 위해서는 사용하기 전에 먼저 직접 정렬을 해 주어야 합니다.

sequence로부터 Dict만들기

```
mapping = {}
for key, value in zip(key_list, value_list):
mapping[key] = value
```

dict는 보통 2가지 tuple의 조합으로 이루어진다

```
In [453]: mapping = dict(zip(range(5), reversed(range(5))))
In [454]: mapping
Out[454]: {0: 4, 1: 3, 2: 2, 3: 1, 4: 0}
```

Dict의 논리 사용 예

```
if key in some_dict:
        value = some_dict[key]
else:
        value = default_value
```

get, pop을 이용해 위의 구문을 아래와 같이 단순화 할 수 있다.

```
value = some_dict.get(key, default_value)
```

Dict 생성하기

```
In [455]: words = ['apple', 'bat', 'bar', 'atom', 'book']
In [456]: by_letter = {}
In [457]: for word in words:
.....: letter = word[0]
.....: if letter not in by_letter:
.....: by_letter[letter] = [word]
.....: else:
.....: by_letter[letter].append(word)
.....:
In [458]: by_letter
Out[458]: {'a': ['apple', 'atom'], 'b': ['bat', 'bar', 'book']}
```

setdefault dict

```
by_letter.setdefault(letter, [ ]).append(word)
```

defaultdict 각각의 slot의 key들을 이미 설정 되어있는 기본값으로 설정해 주는 기능

```
from collections import defaultdict
by_letter = defaultdict(list)
for word in words:
        by_letter[word[0]].append(word)
```

default dict의 사용예시

```
counts = defaultdict(lambda: 4)
```

```
예제

# 'ab' is short for 'a'ddress'b'ook



ab = {   'Swaroop'    : 'swaroop@swaroopch.com',

         'Larry'      : 'larry@wall.org',
```

```
        'Matsumoto' : 'matz@ruby-lang.org',
        'Spammer'    : 'spammer@hotmail.com'
    }

print "Swaroop's address is", ab['Swaroop']

# Deleting a key-value pair
del ab['Spammer']

print '₩nThere are {} contacts in the address-book₩n'.format(len(ab))

for name, address in ab.items():
    print 'Contact {} at {}'.format(name, address)

# Adding a key-value pair
ab['Guido'] = 'guido@python.org'

if 'Guido' in ab:
    print "₩nGuido's address is", ab['Guido']
```

TIP 키워드 인수와 사전의 관계

함수를 호출할 때 키워드 인수를 사용해 보셨다면, 이미 사전을 사용해 보신 것입니다! 여기서 여러분이 지정해준 키-값 쌍은 각각 함수를 정의할 때 지정해준 매개 변수들의 이름과 각 매개 변수에 넘겨줄 값에 대응하는 하나의 사전에 접근하는 것입니다 (이것을 _심볼 테이블_이라고 부릅니다).

12.6. 집합(Set)

집합은 정렬되지 않은 단순 객체의 묶음입니다. 집합은 포함된 객체들의 순서나 중복에 상관없이 객체를 묶음 자체를 필요로 할 때 주로 사용합니다.

집합끼리는 멤버십 테스트를 통해 한 집합이 다른 집합의 부분집합인지 확인할 수 있으며, 두 집합의 교집합 등을 알아낼 수도 있습니다.

```
>>> bri = set(['brazil', 'russia', 'india'])

>>> 'india' in bri

True

>>> 'usa' in bri

False

>>> bric = bri.copy()

>>> bric.add('china')

>>> bric.issuperset(bri)

True

>>> bri.remove('russia')

>>> bri & bric # OR bri.intersection(bric)

{'brazil', 'india'}
```

```
In [465]: set([2, 2, 2, 1, 3, 3])
Out[465]: set([1, 2, 3])

In [466]: {2, 2, 2, 1, 3, 3}
Out[466]: set([1, 2, 3])
```

Function	Alternate Syntax	Description
a.add(x)	N/A	Add element x to the set a
a.remove(x)	N/A	Remove element x from the set a
a.union(b)	a | b	All of the unique elements in a and b.
a.intersection(b)	a & b	All of the elements in *both* a and b.
a.difference(b)	a - b	The elements in a that are not in b.
a.symmetric_difference(b)	a ^ b	All of the elements in a or b but *not both*.
a.issubset(b)	N/A	True if the elements of a are all contained in b.
a.issuperset(b)	N/A	True if the elements of b are all contained in a.
a.isdisjoint(b)	N/A	True if a and b have no elements in common.

예제

```
print 'Simple Assignment'

shoplist = ['apple', 'mango', 'carrot', 'banana']

# mylist is just another name pointing to the same object!

mylist = shoplist


# I purchased the first item, so I remove it from the list

del shoplist[0]


print 'shoplist is', shoplist

print 'mylist is', mylist

# Notice that both shoplist and mylist both print

# the same list without the 'apple' confirming that

# they point to the same object


print 'Copy by making a full slice'
```

```
# Make a copy by doing a full slice

mylist = shoplist[:]

# Remove first item

del mylist[0]


print 'shoplist is', shoplist

print 'mylist is', mylist

# Notice that now the two lists are different


실행 결과:

$ python ds_reference.py

Simple Assignment

shoplist is ['mango', 'carrot', 'banana']

mylist is ['mango', 'carrot', 'banana']

Copy by making a full slice

shoplist is ['mango', 'carrot', 'banana']

mylist is ['carrot', 'banana']
```

리스트와 같은 어떤 열거형이나 복잡한 객체 (정수형과 같이 단순 객체 를 제외하고)의 복사본을 생성하고 싶을 때에는, 슬라이스 연산자를 이용하여 복사본을 생성해야 합니다. 단순히 한 변수를 다른 변수에 할당하게 되면 두 변수는 같은 객체를 ''참조'' 하게 되며 실제 복사본이 생성되지 않습니다. 따라서 이것을 조심하지 않으면 문제가 발생할 수 있습니다.

자료형에서 사용되는 편리한 구문들

list

```
[expr for val in collection if condition]
```

위의 명령은 아래의 명령과 같은 기능을 한다.

```
result = []
for val in collection:
if condition:
result.append(expr)
```

Dict

```
dict_comp = {key-expr : value-expr for value in collection
if condition}
```

Set

```
set_comp = {expr for value in collection if condition}
```

```
In [481]: loc_mapping = {val : index for index, val in enumerate(strings)}
In [482]: loc_mapping
Out[482]: {'a': 0, 'as': 1, 'bat': 2, 'car': 3, 'dove': 4, 'python': 5}

같은 기능, 다른 구문
loc_mapping = dict((val, idx) for idx, val in enumerate(strings)}
```

Nested list comprehensions

다음과 같은 남자/여자 이름으로 이루어진 list가 있다.

```
In [483]: all_data = [['Tom', 'Billy', 'Jefferson', 'Andrew', 'Wesley', 'Steven', 'Joe'],
.....: ['Susie', 'Casey', 'Jill', 'Ana', 'Eva', 'Jennifer', 'Stephanie']]
```

'E'를 두개 가지고 있는 이름들을 따로 정리하고싶다.

```
names_of_interest = []
for names in all_data:
enough_es = [name for name in names if name.count('e') > 2]
names_of_interest.extend(enough_es)
```

Nested list comprehensions을 이용하여 위의 기능을 아래와 같이 사용할 수 있다.

```
In [484]: result = [name for names in all_data for name in names
.....: if name.count('e') >= 2]

In [485]: result
Out[485]: ['Jefferson', 'Wesley', 'Steven', 'Jennifer', 'Stephanie']
```

Example

```
In [486]: some_tuples = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]
In [487]: flattened = [x for tup in some_tuples for x in tup]

In [488]: flattened
Out[488]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

위의 구문은 단순히 for루프로 풀어 아래와 같이 쓸 수 있다.

```
flattened = []
for tup in some_tuples:
for x in tup:
flattened.append(x)
```