Programming in Python

07 – 리스트와 사전

2016년 8월, 국민대학교 컴퓨터공학부

리스트 (Lists)

Python 리스트의 특징들

- 임의의 객체를 모아 순서를 부여한 것
  - 객체들을 그룹화하여 왼쪽부터 오른쪽으로 순서가 매겨진 또다른 객체
- 오프셋 (인덱스) 을 이용하여 접근
  - 순서 (0 부터 시작함) 에 해당하는 정수 오프셋을 이용하여 각 객체에 접근 가능
- 가변 길이이며 중첩 가능
  - 동적으로 자라나고 줄어들 수 있으며, 리스트 안에 또다른 리스트를 포함할 수 있음
- “mutable sequence”
  - 문자열과는 달리 이미 생성된 객체에 대하여 변경을 가하는 것이 가능
- 객체 참조의 배열
  - 내부적으로는 리스트를 구성하는 각 객체에 대한 참조 (포인터) 의 모음

기본적 리스트 연산

```
>>> len([1, 2, 3])                  # 리스트의 길이: 포함된 원소의 수
3

>>> [1, 2, 3] + [4, 5, 6]           # 병합 (concatenation)
[1, 2, 3, 4, 5, 6]

>>> ['Ni!'] * 4                     # 반복
['Ni!', 'Ni!', 'Ni!', 'Ni!']



>>> str([1, 2]) + "34"              # "[1, 2]" + "34" 와 동일
'[1, 2]34'

>>> [1, 2] + list("34")             # [1, 2] + ["3", "4"] 와 동일
[1, 2, '3', '4']
```

문자열의 경우와 비교!

리스트 반복과 내포

```
>>> 3 in [1, 2, 3]                # 원소가 리스트에 포함?
True

>>> for x in [1, 2, 3]:           # for x in "123"?
...     print(x, end=' ')
...
1 2 3
```

문자열의 경우와 비교!

```
>>> res = [c * 4 for c in 'SPAM']
>>> res
['SSSS', 'PPPP', 'AAAA', 'MMMM']

>>> res = []
>>> for c in 'SPAM':
...     res.append(c * 4)
...
>>> res
['SSSS', 'PPPP', 'AAAA', 'MMMM']

>>> list(map(abs, [-1, -2, 0, 1, 2]))
[1, 2, 0, 1, 2]
```

생성된 리스트를 변경하는 것이 가능

```
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[1] = 'eggs'                    # 인덱스를 이용한 치환
>>> L
['spam', 'eggs', 'SPAM!']

>>> L[0:2] = ['eat', 'more']         # 슬라이스를 치환
>>> L
['eat', 'more', 'SPAM!']



>>> L.append('please')               # append 는 리스트가 제공하는 메서드
>>> L
['eat', 'more', 'SPAM!', 'please']

>>> L.sort()                         # sort 또한 메서드 (note: 'S' < 'e')
>>> L
['SPAM!', 'eat', 'more', 'please']
```

리스트의 정렬 (Sort)

```
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort()
>>> L
['ABD', 'aBe', 'abc']

>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort(key=str.lower)                    # sort 에 이용될 키를 지정
>>> L
['abc', 'ABD', 'aBe']

>>> str.lower                                # str.lower 가 무엇이길래?
<method 'lower' of 'str' objects>

>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort(key=str.lower, reverse=True)      # 역순으로의 정렬도 가능
>>> L
['aBe', 'ABD', 'abc']
```

정렬된 리스트 객체를 새로 생성

리스트가 제공하는 sort() 메서드를 이용하는 것과의 차이점은 무엇인가?

```
>>> L = ['abc', 'ABD', 'aBe']
>>> sorted(L, key=str.lower, reverse=True)
['aBe', 'ABD', 'abc']

>>> sorted
<built-in function sorted>

>>> L = ['abc', 'ABD', 'aBe']
>>> sorted([x.lower() for x in L], reverse=True)
['abe', 'abd', 'abc']
```

이 결과는 같지 않음에 주의!
왜 그럴까?

리스트 메서드들

```
>>> L = [1, 2]
>>> L.extend([3, 4, 5])
>>> L
[1, 2, 3, 4, 5]

>>> L.pop()
5
>>> L
[1, 2, 3, 4]

>>> L.reverse()
>>> L
[4, 3, 2, 1]

>>> list(reversed(L))
[1, 2, 3, 4]
```

```
>>> L = []
>>> L.append(1)
>>> L.append(2)
>>> L
[1, 2]
>>> L.pop()
2
>>> L
[1]
```

이것은 어떤 자료구조를 구현한 것인가요?

리스트에의 삽입과 삭제

```
>>> L = ['spam', 'eggs', 'ham']
>>> L.index('eggs')                # 특정 오브젝트의 인덱스
1

>>> L.insert(1, 'toast')           # 특정 포지션에 삽입
>>> L
['spam', 'toast', 'eggs', 'ham']

>>> L.remove('eggs')               # 오브젝트의 값으로 삭제
>>> L
['spam', 'toast', 'ham']

>>> L.pop(1)                       # 특정 포지션에서 pop
'toast'
>>> L
['spam', 'ham']
```

```
>>> L
['SPAM!', 'eat', 'more', 'please']
>>> del L[0]
>>> L
['eat', 'more', 'please']

>>> del L[1:]
>>> L
['eat']

>>> L = ['Already', 'got', 'one']
>>> L[1:] = []
>>> L
['Already']

>>> L[0] = []
>>> L
[[]]
```

사전 (Dictionaries)

Python 사전의 특징들

리스트와 비교!

- 오프셋 (인덱스) 이 아닌 키 값으로 접근
  - 키 (key) 와 그에 해당하는 값 (value) 를 짝지어 둔 것
  - 연관 배열 (associative array) 또는 해쉬 (hash) 라고 부르기도 함
- 객체의 모음이지만 순서는 정해져 있지 않음
  - 임의의 객체의 집합으로서, 순서를 가지지 않고 키 값에 의하여 객체가 참조되어 있음
- 가변 길이이며 중첩 가능
  - 동적으로 자라나고 줄어들 수 있으며, 사전 안에 다른 사전이나 리스트 또한 포함할 수 있음
- “mutable mapping”
  - 문자열과는 달리 이미 생성된 객체에 대하여 변경을 가하는 것이 가능
- 객체 참조에 대한 해쉬 테이블
  - 내부적으로는 각 객체를 참조하는 해쉬 테이블로 구현됨

기본적 사전 연산

```
>>> D = {'spam': 2, 'ham': 1, 'eggs': 3}        # 사전 객체를 생성
>>> D['spam']                                   # 키에 의하여 객체 접근
2

>>> D                                           # 순서는 지켜지지 않음
{'eggs': 3, 'ham': 1, 'spam': 2}

>>> len(D)                          # 사전에 포함된 객체의 개수
3

>>> 'ham' in D                      # 특정 원소가 사전의 키로 포함되어 있는지?
True

>>> list(D.keys())                  # 키의 집합을 리스트로 구성
['eggs', 'ham', 'spam']
```

리스트의 경우와 비교!

생성된 사전을 변경하는 것이 가능

```
>>> D
{'eggs': 3, 'ham': 1, 'spam': 2}

>>> D['ham'] = ['grill', 'bake', 'fry']                    # 원소의 대체
>>> D
{'eggs': 3, 'ham': ['grill', 'bake', 'fry'], 'spam': 2}

>>> del D['eggs']                                          # 원소의 삭제 (키 이용)
>>> D
{'ham': ['grill', 'bake', 'fry'], 'spam': 2}

>>> D['brunch'] = 'Bacon'                                  # 새로운 원소의 추가
>>> D
{'brunch': 'Bacon', 'ham': ['grill', 'bake', 'fry'], 'spam': 2}
```

사전 메서드들

```
>>> D = {'spam': 2, 'ham': 1, 'eggs': 3}
>>> list(D.values())                    # 원소의 값들을 모아 리스트를 생성
[3, 1, 2]

>>> list(D.items())                     # 원소들 자체를 리스트로 생성
[('eggs', 3), ('ham', 1), ('spam', 2)]

>>> type(list(D.items())[0])            # 각 원소의 타입은?
<type 'tuple'>

>>> D.get('spam')                       # 존재하는 키에 대하여 참조
2
>>> D.get('toast')                      # 존재하지 않는 키에 대하여 참조하면?
>>> print(D.get('toast'))
None
>>> D.get('toast', 88)                  # 키에 대하여 디폴트 값을 에러 없이 구현하고자
88
>>> D                                   # 사전 객체에는 변화가 없음
{'eggs': 3, 'ham': 1, 'spam': 2}
```

사전 메서드들

```
>>> D
{'eggs': 3, 'ham': 1, 'spam': 2}
>>> D2 = {'toast': 4, 'muffin': 5}
>>> D.update(D2)                          # 리스트 병합과 비교 / 만약 동일한 키가 있다면?
>>> D
{'toast': 4, 'muffin': 5, 'eggs': 3, 'ham': 1, 'spam': 2}

>>> D.pop('muffin')                       # 특정 키를 가지는 원소의 값을 반환하고 그 원소를 삭제
5
>>> D.pop('toast')
4
>>> D
{'eggs': 3, 'ham': 1, 'spam': 2}
```

사전의 응용

```
>>> table = {'Python': 'Guido van Rossum',
...          'Perl':   'Larry Wall',
...          'Tcl':    'John Ousterhout' }
>>> language = 'Python'
>>> creator = table[language]
>>> creator
'Guido van Rossum'

>>> for lang in table:
...     print(lang, '\t', table[lang])
...
Python      Guido van Rossum
Tcl         John Ousterhout
Perl        Larry Wall
```

L = [0] * 100

```
>>> L = []
>>> L[99] = 'spam'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
>>>
>>> D = {}
>>> D[99] = 'spam'
>>> D[99]
'spam'
>>> D
{99: 'spam'}
```

사전의 응용 - 희소 행렬 (Sparse Matrix)

```
>>> Matrix = {}
>>> Matrix[(2, 3, 4)] = 88
>>> Matrix[(7, 8, 9)] = 99

>>> X = 2; Y = 3; Z = 4
>>> Matrix[(X, Y, Z)]
88
>>> Matrix
{(2, 3, 4): 88, (7, 8, 9): 99}
```

존재하지 않는 좌표를 접근하려 하면:

```
>>> Matrix[(2, 3, 6)]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: (2, 3, 6)
```

서로 다른 세 가지의 해결책:

```
>>> if (2, 3, 6) in Matrix:
...     print(Matrix[(2, 3, 6)])
... else:
...     print(0)
...
0
```

```
>>> try:
...     print(Matrix[(2, 3, 6)])
... except KeyError:
...     print(0)
...
0
```

```
>>> Matrix.get((2, 3, 4), 0)
88
>>> Matrix.get((2, 3, 6), 0)
0
```

Quiz

- L = [1, 2, 6, 7, 8] 일 때, 이 리스트를 [1, 2, 3, 4, 5, 6, 7, 8] 이 되게 하려면?
  - L = L[:2] + [3, 4, 5] + L[2:] ?
  - L.insert(2, [3, 4, 5]) ?
  - L[2:2] = [3, 4, 5] ?

- 사전의 키는 항상 문자열이어야 하는가?
  - D = {}
  - D[1] = 3 ?
  - L = [1, 2]; D[L] = 5 ?
  - from math import sqrt; D[sqrt] = 7 ?

- 리스트 내의 원소가 사전일 수 있는가?

- 사전 내의 원소의 값이 리스트일 수 있는가?

Exercise

(1) 학생의 이름과 점수를 입력으로 받아
그 쌍을 기록해 두는 클래스를 작성해 보자.

- 클래스 이름: Score
- 입력 메서드: putScore(self, name, score)
- 출력 메서드: printScore(self)
  - 학생 이름의 알파벳 순서로 아래 예제처럼 출력
    - John: 87
    - Mary: 92
    - Peter: 63

Answers

```
class Score:

    def __init__(self):
        self.scores = {}

    def putScore(self, name, score):
        self.scores[name] = score

    def printScore(self):
        names = list(self.scores.keys())
        for name in sorted(names):
            print('%s\t%s' % (name, self.scores[name]))
```

```
>>> from score import Score
>>> s = Score()
>>> s.putScore('Mary', 92)
>>> s.putScore('Peter', 63)
>>> s.printScore()
Mary        92
Peter       63
>>> s.putScore('John', 87)
>>> s.printScore()
John        87
Mary        92
Peter       63
```

Exercise

(1) 앞의 Score 클래스에 메서드를 추가하여 보자.

- 메서드: getScore(self, name)
  - 이름이 주어진 학생의 기록이 존재하면 그 점수를 리턴
  - 이름이 주어진 학생의 기록이 존재하지 않으면 -1 을 리턴

Answers

```
class Score:

    def __init__(self):
        self.scores = {}

    def putScore(self, name, score):
        self.scores[name] = score

    def printScore(self):
        names = list(self.scores.keys())
        for name in sorted(names):
            print('%s\t%s' % (name, self.scores[name]))

    def getScore(self, name):
        return self.scores.get(name, -1)
```

```
>>> from score import Score
>>> s = Score()
>>> s.putScore('Mary', 92)
>>> s.printScore()
Mary        92
>>> s.getScore('Mary')
92
>>> s.getScore('Frank')
-1
```

Q & A