프로그래밍

이은섭
(Eunseop Lee)

Computer Science and Engineering,
Pohang University of Science and Technology
eseop90@gmail.ac.kr

리스트

리스트

- 1부터 10까지 숫자 중 홀수의 집합을 표현하면?
  - 문자열: "1 3 5 7 9"
  - 숫자: ?

- 홀수의 집합에 1을 더하려면?
  - 문자열: ?
  - 숫자: ?

- 리스트(list)를 사용하면 쉽게 표현할 수 있음

```
>>> odd = [1, 3, 5, 7, 9]
```

리스트

- 리스트는 대괄호 '[]'로 감싸 주고 각 요소값은 쉼표 ','로 구분함

```
리스트명 = [요소1, 요소2, 요소3, ...]
```

- 리스트의 요소값은 어떠한 자료형도 포함할 수 있음
- 비어 있는 리스트를 만들 수 있음

```
>>> a = []
>>> b = [1, 2, 3]
>>> c = ['Life', 'is', 'too', 'short']
>>> d = [1, 2, 'Life', 'is']
>>> e = [1, 2, ['Life', 'is']]
```

리스트 인덱싱

- 리스트는 문자열과 유사하게 인덱싱을 적용할 수 있음

```
>>> a = [1, 2, 3]
>>> a
[1, 2, 3]
```

```
>>> a[0]
1
```

```
>>> a[0] + a[2]
4
```

- 마이너스(-)를 이용하여 인덱싱할 수 있음

```
>>> a[-1]
3
```

리스트 슬라이싱

- 리스트는 문자열과 유사하게 슬라이싱을 적용할 수 있음

```
>>> a = [1, 2, 3, 4, 5]
>>> a[0:2]
[1, 2]
>>> b = a[:2]
>>> c = a[2:]
>>> b
[1, 2]
>>> c
[3, 4, 5]
```

```
>>> a = [1,2,3,4,5]
>>> a[::]
[1, 2, 3, 4, 5]
>>> a[::2]
[1, 3, 5]
```

리스트 연산

- 더하기(+)
  - 리스트 사이 '+'는 리스트끼리 합치는 기능을 함

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a + b
[1, 2, 3, 4, 5, 6]
```

- 곱하기(*)
  - 리스트에 '* N'를 해주면 N회 반복되는 새로운 리스트를 만들어 냄

```
>>> a = [1, 2, 3]
>>> a * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

리스트 연산

- len() 함수
  - 리스트의 길이를 구할 수 있음

```
>>> a = [1, 2, 3]
>>> len(a)
3
```

리스트 수정과 삭제

- 리스트 요소 수정

```
>>> a = [1, 2, 3]
>>> a[2] = 4
>>> a
[1, 2, 4]
```

리스트 수정과 삭제

- 리스트 요소 삭제(del() 함수)
  - del a[X]는 X번째 요소값을 삭제

```
>>> a = [1, 2, 3]
>>> del a[1]
>>> a
[1, 3]
```

  - 슬라이싱 기법을 사용하여 리스트의 요소 여러 개를 한꺼번에 삭제할 수 있음

```
>>> a = [1, 2, 3, 4, 5]
>>> del a[2:]
>>> a
[1, 2]
```

리스트 관련 함수

- append()
  - 리스트의 맨 마지막에 새로운 요소를 추가하는 함수

```
>>> a = [1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
```

- sort()
  - 리스트 내 요소를 순서대로 정렬
  - 숫자뿐만 아니라 알파벳도 정렬할 수 있음

```
>>> a = [1, 4, 3, 2]
>>> a.sort()
>>> a
[1, 2, 3, 4]
```

```
>>> a = ['a', 'c', 'b']
>>> a.sort()
>>> a
['a', 'b', 'c']
```

리스트 관련 함수

- reverse()
  - 리스트의 요소들을 역순으로 뒤집어 줌

```
>>> a = ['a', 'c', 'b']
>>> a.reverse()
>>> a
['b', 'c', 'a']
```

- index()
  - index(x) 함수는 리스트에 x 값이 있으면 x의 인덱스 값을 리턴함
  - 리스트에 해당 값이 존재하지 않으면 오류 발생

```
>>> a = [1, 2, 3]
>>> a.index(3)
2
>>> a.index(1)
0
```

```
>>> a.index(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 0 is not in list
```

리스트 관련 함수

- insert()
  - insert(a, b)는 리스트의 a번째 위치에 b를 삽입하는 함수

```
>>> a = [1, 2, 3]
>>> a.insert(0, 4)
>>> a
[4, 1, 2, 3]
```

```
>>> a.insert(3, 5)
>>> a
[4, 1, 2, 5, 3]
```

- remove()
  - remove(x)는 리스트에서 첫 번째로 나오는 x를 삭제하는 함수

```
>>> a = [1, 2, 3, 1, 2, 3]
>>> a.remove(3)
>>> a
[1, 2, 1, 2, 3]
```

리스트 관련 함수

- pop()
  - pop()은 리스트의 맨 마지막 요소를 리턴하고, 해당 요소는 삭제

```
>>> a = [1, 2, 3]
>>> a.pop()
3
>>> a
[1, 2]
```

  - pop(x)는 리스트의 x번째 요소를 리턴하고 그 요소는 삭제

```
>>> a = [1, 2, 3]
>>> a.pop(1)
2
>>> a
[1, 3]
```

리스트 관련 함수

- count()
  - count(x)는 리스트 안에 x가 몇 개 있는지 리턴하는 함수

```
>>> a = [1, 2, 3, 1]
>>> a.count(1)
2
```

- extend()
  - extend(x)에서 x에는 리스트만 올 수 있으며, 기존의 리스트에 x 리스트를 더함
  - append()와 extend() 구분 필수!

```
>>> a = [1,2,3]
>>> a.extend([4,5])
>>> a
[1, 2, 3, 4, 5]
>>> a.append([6, 7])
>>> a
[1, 2, 3, 4, 5, [6, 7]]
```

튜플

튜플

- 튜플(tuple)은 리스트와 유사하게 순서가 있는 자료들의 나열
- 몇 가지 점을 제외하곤 리스트와 거의 유사하지만, 다른 부분이 존재
  - 리스트는 [], 튜플은 ()로 둘러쌈(생략 가능)
  - 리스트는 요소값의 생성, 삭제, 수정이 가능하지만, 튜플은 요소값을 바꿀 수 없음

```
t1 = ()
t2 = (1,)
t3 = (1, 2, 3)
t4 = 1, 2, 3
t5 = ('a', 'b', ('ab', 'cd'))
```

튜플

- 요소값을 삭제할 때

```
>>> t1 = (1, 2, 'a', 'b')
>>> del t1[0]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

- 요소값을 변경할 때

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0] = 'c'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

튜플

- 인덱싱

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0]
1
>>> t1[3]
'b'
```

- 슬라이싱

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[1:]
(2, 'a', 'b')
```

튜플

- 더하기

```
>>> t1 = (1, 2, 'a', 'b')
>>> t2 = (3, 4)
>>> t3 = t1 + t2
>>> t3
(1, 2, 'a', 'b', 3, 4)
```

- 곱하기

```
>>> t2 = (3, 4)
>>> t3 = t2 * 3
>>> t3
(3, 4, 3, 4, 3, 4)
```

튜플

- len() 함수

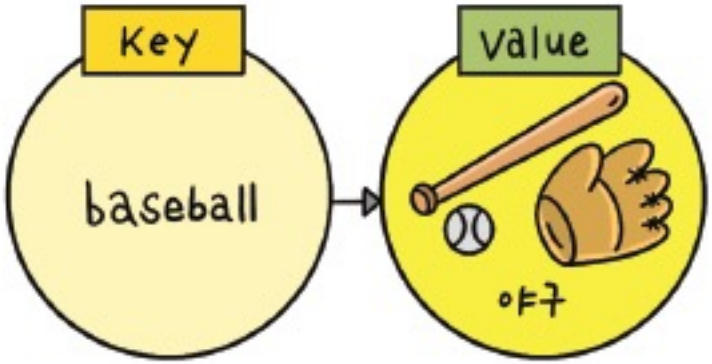
```
>>> t1 = (1, 2, 'a', 'b')
>>> len(t1)
4
```

- 튜플은 요소값을 변경할 수 없기에 sort(), insert(), remove(), pop()과 같은 내장 함수가 없음

딕셔너리

딕셔너리

- 딕셔너리(dictionary)는 key와 value를 한 쌍으로 가지는 자료형으로, 리스트나 튜플처럼 순차적으로 인덱스를 할당하지 않고, key를 통해 value를 호출함



딕셔너리

- 딕셔너리는 각각 요소가 key:value 형태로 이루어져 있고 쉼포로 구분됨

```
{Key1: Value1, Key2: Value2, Key3: Value3, ...}
```

```
>>> dic = {'name': 'pey', 'phone': '010-9999-1234', 'birth':
'1118'}
```

- Key에는 변경 불가능한 자료형(정수, 실수, 문자열, 튜플)만 올 수 있으며, value에는 모든 자료형이 올 수 있음

딕셔너리

- key를 이용하여 value 리턴(1)

```
>>> dic = {'name':'pey', 'phone':'010-9999-1234', 'birth': '1118'}
>>> dic['name']
'pey'
>>> dic['phone']
'010-9999-1234'
>>> dic['birth']
'1118'
```

```
>>> grade = {'pey': 10, 'julliet': 99}
>>> grade['pey']
10
>>> grade['julliet']
99
```

```
>>> a = {1:'a', 2:'b'}
>>> a[1]
'a'
>>> a[2]
'b'
```

딕셔너리

- key를 이용하여 value 리턴(2)
  - get() 함수를 이용

```
>>> a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
>>> a.get('name')
'pey'
>>> a.get('phone')
'010-9999-1234'
```

  - 딕셔너리에 존재하지 않는 키로 값을 가져오려고 할 경우, 딕셔너리['key'] 방법은 오류를 발생시키지만, get() 함수는 None을 리턴함

```
>>> a = {'name':'pey', 'phone':'010-9999-1234', 'birth': '1118'}
>>> print(a.get('nokey'))
None
>>> print(a['nokey'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'nokey'
```

딕셔너리

- key를 이용하여 value 리턴(2)
  - get() 함수를 이용하여 딕셔너리 안에 찾으려는 key가 없는 경우, 미리 정해 둔 디폴트 값을 대신 가져올 수 있음

```
>>> a.get('nokey', 'foo')
'foo'
```

딕셔너리

- 딕셔너리 만들 때 주의사항
  - key는 고유한 값이므로, 중복된 key값을 설정하면, 하나를 제외한 나머지 값들은 무시됨

```
>>> a = {1:'a', 1:'b'}
>>> a
{1: 'b'}
```

딕셔너리

- 요소 추가하기
  - 딕셔너리[key]=value를 입력하면 딕셔너리에 {key:value}가 추가됨

```
>>> a = {1: 'a'}
>>> a[2] = 'b'
>>> a
{1: 'a', 2: 'b'}
```

```
>>> a['name'] = 'pey'
>>> a
{1: 'a', 2: 'b', 'name': 'pey'}
```

```
>>> a[3] = [1, 2, 3]
>>> a
{1: 'a', 2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

딕셔너리

- 요소 삭제하기
  - del 딕셔너리[key]를 입력하면 지정한 key에 해당하는 {key:value} 쌍이 삭제됨

```
>>> del a[1]
>>> a
{2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

- in을 이용하여 해당 key가 딕셔너리 안에 있는지 조사하기

```
>>> a = {'name':'pey', 'phone':'010-9999-1234', 'birth': '1118'}
>>> 'name' in a
True
>>> 'email' in a
False
```

딕셔너리 관련 함수

- keys() 함수
  - 딕셔너리의 key값들을 모아 리턴함

```
>>> a = {'name': 'pey', 'phone': '010-9999-1234', 'birth': '1118'}
>>> a.keys()
dict_keys(['name', 'phone', 'birth'])
```

- values() 함수
  - 딕셔너리의 value값들을 모아 리턴함

```
>>> a.values()
dict_values(['pey', '010-9999-1234', '1118'])
```

딕셔너리 관련 함수

- items() 함수
  - 딕셔너리의 key와 value 쌍들을 리턴함

```
>>> a.items()
dict_items([('name', 'pey'), ('phone', '010-9999-1234'), ('birth', '1118')])
```

- clear() 함수
  - 딕셔너리 안의 모든 요소값들을 삭제함

```
>>> a.clear()
>>> a
{}
```

Q&A