콜렉션 자료형

Update : 2019. 7

Contents

리스트 자료형

- 특징

  다수개의 데이터 값을 함께 저장하고 호출할 수 있는 컨테이너.

  자바, 자바스크립트의 배열과는 다르게 **서로 다른 데이터 형을 함께 사용할 수 있다**

- 기본 구조 :

  리스트명 = [요소1, 요소2, 요소3, ...]

  리스트명[index]를 이용하여 개별 요소에 접근

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

```
>>> a = [1, 2, 3]
>>> a[0]
1
>>> a[0] + a[2]
4
>>> a[-1]
3
```

리스트 요소끼리 더하기

리스트 자료형

리스트의 슬라이싱

listName[:]
listName[start:end]
listName[start:]
listName[:end]
n번째 배수 추출
listName[::n]
마지막 리스트 추출
listName[-1] : 인덱스값이 음수(-)인 경우에는 뒤에서부터 요소를 가르킨다.

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

리스트 자료형

리스트 더하기

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

리스트 반복하기

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

리스트 자료형

리스트에서 하나의 값 수정하기

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

리스트에서 연속된 범위의 값 수정하기

```
>>> a[1:2]
[2]
>>> a[1:2] = ['a', 'b', 'c']
>>> a
[1, 'a', 'b', 'c', 4]
```

리스트 자료형

[ ] 사용해 리스트 요소 삭제하기

```
>>> a = [1, 'a', 'b', 'c', 4]
>>> a[1:3] = []
>>> a
[1, 'c', 4]
```

del 함수 사용해 리스트 요소 삭제하기

```
>>> a
[1, 'c', 4]
>>> del a[1]
>>> a
[1, 4]
```

리스트 자료형

- 리스트 조작 함수

함수	설명	사용법
append()	리스트 맨 뒤에 항목을 추가한다.	리스트명.append(값)
pop()	리스트 맨 뒤의 항목을 빼낸다(리스트에서 해당 항목이 삭제된다).	리스트명.pop()
sort()	리스트의 항목을 정렬한다.	리스트명.sort()
reverse()	리스트 항목의 순서를 역순으로 만든다.	리스트명.reverse()
index()	지정한 값을 찾아 해당 위치를 반환한다.	리스트명.index(찾을값)
insert()	지정된 위치에 값을 삽입한다.	리스트명.insert(위치, 값)
remove()	리스트에서 지정한 값을 삭제한다. 단 지정한 값이 여러 개면 첫 번째 값만 지운다.	리스트명.remove(지울값)
extend()	리스트 뒤에 리스트를 추가한다. 리스트의 더하기(+) 연산과 기능이 동일하다.	리스트명.extend(추가할리스트)
count()	리스트에서 해당 값의 개수를 센다.	리스트명.count(찾을값)
clear()	리스트의 내용을 모두 지운다.	리스트명.clear()
del()	리스트에서 해당 위치의 항목을 삭제한다.	del(리스트명[위치])
len()	리스트에 포함된 전체 항목의 개수를 센다.	len(리스트명)
copy()	리스트의 내용을 새로운 리스트에 복사한다.	새리스트=리스트명.copy()
sorted()	리스트의 항목을 정렬해서 새로운 리스트에 대입한다.	새리스트=sorted(리스트)

리스트 자료형

리스트에 요소 추가(append) : 위치 맨 뒤

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

리스트 정렬(sort)

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

리스트 자료형

리스트 뒤집기(reverse)

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

위치 반환(index)

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

리스트 자료형

리스트에 요소 삽입(insert)
: insert(index,data)리스트의 index 위치에 요소를 삽입한다.

listName.insert(index,data)

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

리스트 요소 제거(remove)
: remove(value) – 첫번째 값만 제거

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

리스트 자료형

리스트 요소 끄집어내기(pop)
: 마지막 값 부터 제거

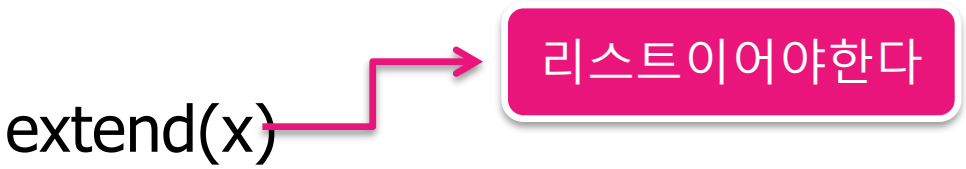
```
>>> a = [1,2,3]
>>> a.pop()
3
>>> a
[1, 2]
```

리스트에 포함된 요소 x의 개수 세기(count)
: count(아이템값)

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

리스트 자료형

리스트 확장(extend)




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

리스트 자료형

문자열로 변환하기 : join()

`' '.join(x)` → 리스트이어야한다

```
>>> a = ['Life', 'is', 'too', 'short']
>>> result = " ".join(a)
>>> print(result)
Life is too short
```

리스트 자료형

다차원 리스트 : 리스트 안에 리스트 정의

```
# 1차원 리스트 정의
kor = [ 55, 66, 34, 60]
math = [ 78, 90, 45, 88]
eng = [ 56, 98, 78, 90]
# 2차원 리스트 정의
grade = [kor, math, eng]
print("grade : ", grade)
print("grade[0][0] : ", grade[0][0])
print("grade[1][1] : ", grade[1][1])
print("grade[2][2] : ", grade[2][2])
```

```
#결과
grade :  [[55, 66, 34, 60], [78, 90, 45, 88], [56, 98, 78, 90]]
grade[0][0] :  55
grade[1][1] :  90
grade[2][2] :  78
```

튜플 자료형

▪ 튜플의 특징

- 리스트는 대괄호 []로 생성, 튜플은 소괄호 ()로 생성

- 튜플은 값을 수정할 수 없으며, 읽기만 가능해 읽기 전용 자료를 저장할 때 사용

▪ 튜플 선언 1

```
# 여러 개 있는 경우
tupleName = (item1, ... )
# 한 개만 있는 경우 쉼표(,)로 마무리
tupleName = (item,)
# 빈값이 있는 경우
tupleName = ()
```

튜플 자료형

- 튜플 선언2

()를 생략

tupleName = item1, ...

한 개만 있는 경우 쉼표(,)로 마무리

tupleName = item,

- 튜플 선언3

공백 튜플

tupleName = ()

튜플 자료형

인덱싱

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

슬라이싱

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

튜플 자료형

튜플 요소값 삭제 시 오류

```
>>> t1 = (1, 2, 'a', 'b')
>>> del t1[0]
Traceback (innermost last):
File "", line 1, in ?del t1[0]
TypeError: object doesn't support item deletion
```

튜플 요소값 변경 시 오류

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0] = 'c'
Traceback (innermost last):
File "", line 1, in ?t1[0] = 'c'
TypeError: object doesn't support item assignment
```

튜플 자료형

더하기

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

곱하기

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

튜플 자료형

튜플 요소 값 추가하기 : 새로 튜플이 생성됨

: += (값,)

```
>>> t = (1, 2)
>>> t += (3,)
>>> print(t)
(1,2,3)
```

딕셔너리 자료형

- 연관 배열(Associative array) 또는 해시(Hash)
- 단어 그대로 해석하면 사전이라는 뜻
- Key를 통해 Value를 얻는다
- 중괄호 {}로 묶어 구성, 키(Key)와 값(Value)의 쌍으로 구성
- 주의할 점

  : 딕셔너리에는 순서가 없어 생성한 순서대로 딕셔너리가 구성되어 있다는 보장 없음

  : 키 값이 같은 경우 마지막 아이템만 유지된다.

  : 키 값으로 리스트는 쓸 수 없다.

```
딕셔너리변수 = {키1:값1, 키2:값2, 키3:값3, …}
```

```
>>> dic = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
```

딕셔너리 자료형

- **딕셔너리의 생성**

```
dic1 = {1 : 'a', 2 : 'b', 3 : 'c'}
dic1
```

출력 결과

```
{1 : 'a', 2 : 'b', 3 : 'c'}
```

- **키와 값을 반대로 딕셔너리 생성**

```
dic2 = {'a': 1, 'b': 2, 'c': 3}
dic2
```

출력 결과

```
{'a': 1, 'b': 2, 'c': 3}
```

키와 값은 사용자가 지정하는 것이지 규정은 없음
주의할 점 : 딕셔너리에는 순서가 없어 생성한 순서대로 딕셔너리가 구성되어 있다는 보장 없음

딕셔너리 자료형

딕셔너리 쌍 추가하기

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

딕셔너리 요소 삭제하기

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

딕셔너리 자료형

딕셔너리에서 Key 사용해 Value 얻기

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

```
>>> sports = {"축구":"박지성", "야구":"강정호", "체조":"손연제"}
>>> sports["축구"]
'박지성'
>>> sports["야구"]
'강정호'
```

딕셔너리 만들 때 주의할 사항

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

- 키 값이 같은 경우 마지막 아이템만 유지된다.

딕셔너리 자료형

Key 리스트 만들기(keys) : keys()

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth':
'1118'}
>>> a.keys()
dict_keys(['name', 'phone', 'birth'])
```

Value 리스트 만들기(values) : values()

```
>>> a.values()
dict_values(['pey', '0119993323', '1118'])
```

딕셔너리 자료형

Key, Value 쌍 얻기(items)

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth':
'1118'}
>>> a.items()
dict_items([('name', 'pey'), ('phone', '0119993323'),
('birth', '1118')])
```

Key: Value 쌍 모두 지우기(clear)

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

딕셔너리 자료형

Key로 Value얻기(get)

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> a.get('name')
'pey'
>>> a.get('phone')
'0119993323‘
>>> a.get(‘addr’)
>>> dict.get('f','없다')
‘없다’
```

값이 없는 경우 None 리턴

값이 없는 경우 디폴트값 리턴

딕셔너리 자료형

해당 Key가 딕셔너리 안에 있는지 조사하기(in)

Key in dictName

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> 'name' in a
True
>>> 'email' in a
False
```

딕셔너리 자료형

dict() 함수로 딕셔너리 생성하기
: 딕셔너리변수 = dict()

```
#항목(key:value)이 없는 딕셔너리를 만든다.
aa = dict()
print(aa)
>> {}
aa['one'] = "첫번째"
print (aa)
>> {‘one’:’첫번째’}
```

딕셔너리 자료형

pop()함수를 이용해서 value값을 가져오기
: 딕셔너리변수.pop('키값' [, '기본값'])
:딕셔너리에 항목을 제거한다.

```
bb = {"name":"홍길동", "hp":"010-1234-1234", "age":24}
m = bb.pop('name')
print(m)
>> "홍길동"
print(bb)
>> {'hp': '010-1234-1234', 'age': 24}
m = bb.pop('gender','없음')
>>없음
```

#키가 없는 경우에는 디폴트값으로 대체 :pop(키 [,디폴트값])

집합 자료형

- 집합에 관련된 것들을 쉽게 처리하기 위해 만들어진 자료형
- 중복을 허용하지 않는다.
- 순서가 없다(Unordered)
- 집합 자료형 선언하기

  set() 을 이용하여 객체 선언

  집합이름 = set([data1, data2…])
- 집합 자료형 출력하기

  집합이름

  { data1, data2… }

```
>>> s1 = set([1,2,3])
>>> s1
{1, 2, 3}
```

```
>>> s2 = set("Hello")
>>> s2
{'e', 'l', 'o', 'H'}
```

집합 자료형

- 리스트 튜플은 순서가 있다. 따라서, 인덱싱을 이용하여 값을 얻어올 수 있다. 집합은 순서가 없기 때문에 인덱싱으로 값을 얻어올 수 없다.
- 집합과 딕셔너리는 인덱싱을 지원하지 않는다.
- 집합에 저장된 값을 인덱싱으로 접근하기 위해서는 리스트나 튜플로 변환해야한다.

집합 자료형

교집합 1 - setName1 & setName2

```
>>> s1 = set([1, 2, 3, 4, 5, 6])
>>> s2 = set([4, 5, 6, 7, 8, 9])
>>> s1 & s2
{4, 5, 6}
```

교집합 2 - setName1.intersection(setName2)

```
>>> s1.intersection(s2)
{4, 5, 6}
```

집합 자료형

합집합 1 - setName1 | setName2

```
>>> s1 = set([1, 2, 3, 4, 5, 6])
>>> s2 = set([4, 5, 6, 7, 8, 9])
>>> s1 | s2
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

합집합 2 - setName1.union(setName2)

```
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

집합 자료형

차집합 1 - setName1 - setName2

```
>>> s1 = set([1, 2, 3, 4, 5, 6])
>>> s2 = set([4, 5, 6, 7, 8, 9])
>>> s1 - s2
{1, 2, 3}
>>> s2 - s1
{8, 9, 7}
```

차집합 2 - setName1.difference(setName2)

```
>>> s1.difference(s2)
{1, 2, 3}
>>> s2.difference(s1)
{8, 9, 7}
```

집합 자료형

값 1개 추가하기(add) – setName.add(data)

```
>>> s1 = set([1, 2, 3])
>>> s1.add(4)
>>> s1
{1, 2, 3, 4}
```

값 여러 개 추가하기(update) – setName.update([data1 , data2 ...])

```
>>> s1 = set([1, 2, 3])
>>> s1.update([4, 5, 6])
>>> s1
{1, 2, 3, 4, 5, 6}
```

집합 자료형

특정 값 제거하기(remove) – setName.remove(data)

```
>>> s1 = set([1, 2, 3])
>>> s1.remove(2)
>>> s1
{1, 3}
```

특정 값 제거하기(discard) – setName. discard(data)

```
>>> s1 = set([1, 2, 3])
>>> s1
{1,2,3}
>>> s1.discard(1)
>>> s1
{2,3}
```

집합내에 없는 항목을 제거 할때는 에러가 발생하지 않는다.

집합 자료형

모든 값 제거하기(clear) – setName. clear()

```
>>> s1 = set([1, 2, 3])
>>> s1
{1,2,3}
>>> s1.clear()
>>> s1
{}
```

집합 자료형

대칭 차집합 (^)

: 두개의 집합이 있을 때 둘 중 한집합에만 있는 항목들

: 집합1 ^ 집합2

```
>>> s = set("Good Morning")
>>> t = set("Good Night")
>>> print(s ^ t)
>>> {'N', 't', 'h', 'r', 'M', 'n'}
```

자료형의 참과 거짓

- 불(Boolean)형
  - 참과 거짓
  - True / 1
  - False / 0

```
a = (100 == 100)
b = (10 > 100)
print(a, b)
```

출력 결과

```
True False
```

자료형의 참과 거짓

값	참 or 거짓
"python"	참
""	거짓
[1, 2, 3]	참
[]	거짓
()	거짓
{}	거짓
1	참
0	거짓
None	거짓

자료형의 참과 거짓
: 모든 자료형에는 참과 거짓이 있다.