컬렉션 자료 이해하기

백석대학교 강윤희

자료형

- List
  - 값의 나열로 여러 자료형의 값을 담을 수 있음
  - [ ] 에 요소를 , 로 분리하여 유지함
  - 인덱싱과 슬라이싱 지원
  - 리스트의 두번째 자료를 출력하고자함

  fruits = ["apple", "banana", "cherry"]
  print( )
  type(fruits) # 출력 내용은

- 복잡한 데이터를 다루기 위해 사용됨

  a = [1, 2, ['a', 'b', ['Life', 'is']]]
  a[2][2][0] 의 값은

자료형

```
>>> lst = ["easy", "simple", "cheap", "fre
e"]
 >>> lst[-1]
'free’
>>> lst = [3, 5, 7]
>>> lst.append(42)
>>> lst
[3, 5, 7, 42]
 >>> lst = [3, 5, 7]
>>> lst = lst.append(42)
>>> print(lst)
None
```

b와 c는 어떤 값을 갖는가 ?

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

리스트 a를 정렬하기 위한 함수는 무엇인가 ?

리스트(List)

- 복합자료형

[0]
[2.3, 4.5]
[5, "Hello", "there", 9.8]
[]

- len() 를 사용하여 리스트의 길이를 얻음

>>> names = ["Kang", “Yun", “Hee"]
>>> len(names)

[ ]로 리스트 아이템 사용

```
>>> names[0]
'Kang'
>>> names[1]
'Myung'
>>> names[2]
'Ju'
>>> names[3]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

```
>>> names[-1]
'Ju'
>>> names[-2]
'Myung'
>>> names[-3]
'Kang'
```

리스트(List)

```
# list_sort.py
import sys

input_list = sys.argv[1:]
input_list.sort()
print (input_list)
```

```
$ python list_sort.py  Apple Orange Graph lemon
['Apple', 'Graph', 'Orange', 'lemon']
```

리스트(List)

- 리스트의 자료 원소에 대해 제곱값을 구함

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)   # Prints [0, 1, 4, 9, 16]
```

- 단순화한 리스트 처리

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)   # Prints [0, 1, 4, 9, 16]
```

리스트(List)

- 리스트의 짝수 자료 원소에 대해 제곱값을 구함

```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Prints "[0, 4, 16]"
```

(심화)자료형

```
>>> cities = ["Hamburg", "Linz", "Salzburg", "Vienna"]
>>> cities.pop(0)
'Hamburg'
>>> cities
['Linz', 'Salzburg', 'Vienna']
>>> cities =  ['Linz', 'Salzburg', 'Vienna']
>>> cities.pop(1)
'Salzburg’
>>> cities
```

```
>>> cities2 = ["Seoul", "Busan", "Incheon", "Vienna"]
>>> cities2.pop(-1)
‘Vienna’
>>> cities
```

(심화)자료형

```
>>> cities = ["Amsterdam", "The Hague", "Strasbourg"]
>>> cities.pop()
'Strasbourg'
>>> cities
['Amsterdam', 'The Hague']
>>>
```

스택의 pop() 연산과 동일함

자료형

- Tuple, 튜플 : 리스트(list)와 거의 비슷하며, 리스트와 다른 점
  - 리스트는 [ ]으로 둘러싸지만 튜플은 ( )으로 표현됨
  - 리스트는 그 값의 생성, 삭제, 수정이 가능하지만 튜플은 그 값을 바꿀 수 없음

Thistuple = (" apple " , " banana " , " cherry ")

Thistuple[1] = " blackcurrant "
TypeError: ' tuple ' object does not support item assignment
print(thistuple)

자료형

- Set, 집합
  - 순서가 없으며 중복 허용이 안됨

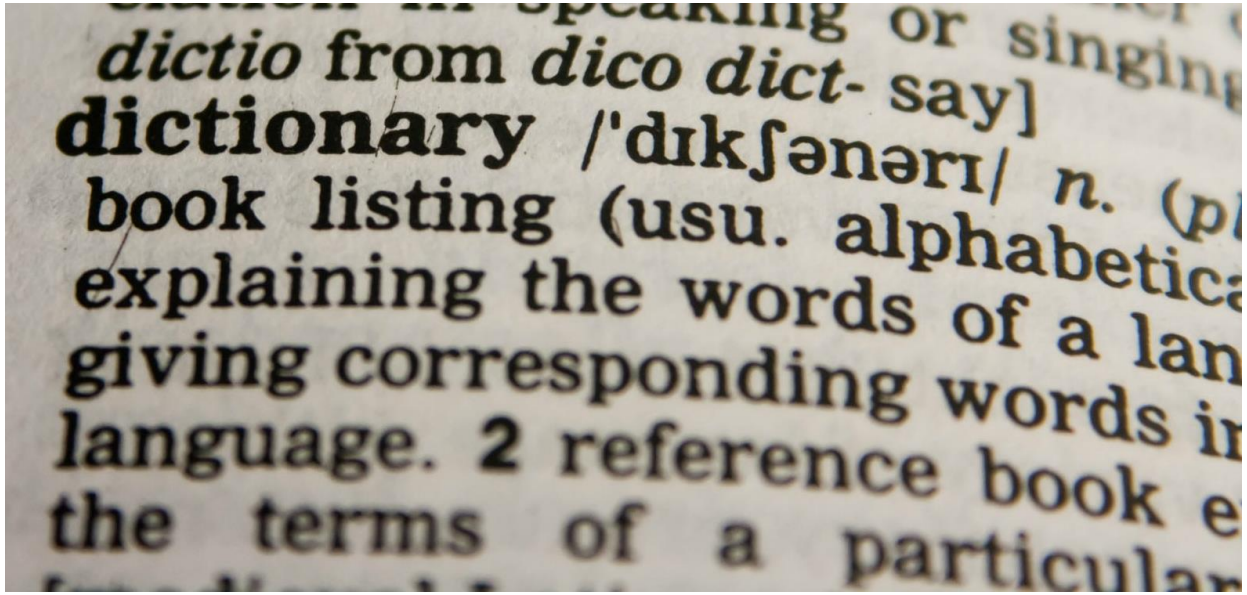
```
thisset = {"apple","banana","cherry", "apple"}
thisset
{'banana', 'apple', 'cherry'}
```

자료형

- 딕셔너리(사전)
  - Key와 Value 쌍으로 이루어진 자료형, 연관 배열

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)


x = thisdict["model"]
```



(생각하기) 올바른 자료구조

```
def buildList():
    bigList = [str(i) for i in range(10000000)]
def findList():
    "abc" in bigList
```

```
def buildSet():
    bigSet = set(bigList)
def findSet():
    "abc" in bigSet
```

buildList : 30.978904340999975 milliseconds findList : 1.344825822999951 milliseconds

buildSet : 13.866673436999918 milliseconds findSet : 5.642000019179 25e-06 milliseconds

(문제풀이) 길이 얻기

- 함수 len 을 사용하여 리스트 또는 문자열의 길이를 얻음

```
name = "Jamie"
print(len(name)) # 5

names = ["Bob", "Jane", "James", "Alice"]
print(len(names)) # 4
```

(문제풀이)

세개의 센서에서 생성된 자료를 튜플로 구성한 후 리스트로 구성

```
a = b = c = range(20)
zipped = zip(a, b, c) # 튜풀 구성
result = list(zipped) # 튜플로 구성된 리스트 생성
print(result)
```

(문제풀이) 컬렉션 자료형 특징 이해하기

- ( )는 배열 처럼 취급되며 인덱싱과 슬라이싱을 지원
- ( )은 변형이 불가능한 리스트임
- ( )은 색인을 갖지 않으며 순서가 없음, 중복된 내용을 갖지 않음
- 연관 배열(Associative array) 을 표현하기 위해 ( )가 사용되며, 자료항목은 키와 값의 쌍으로 이루어짐