CH3. 복합데이터

3-2. 튜플

3-3. 딕셔너리

3-4. 집합

목차

3. 복합데이터

- 복합데이터 타입의 종류

리스트(List)

- 여러 개의 데이터를 가진 순서형 자료
- 수정할 수 있다
- 파이썬에서는 리스트에 서로 다른 데이터 타입들을 저장할 수 있다

```
x = [1, 2, 3, 4, 5]
```

튜플(Tuple)

- 여러 개의 데이터를 가진 순서형 자료
- 수정할 수 없다
- 리스트와 마찬가지로 인덱싱이 가능하다. 인덱싱은 대괄호로 하도록 주의한다.

```
x = (1, 2, 3, 4, 5)
```

딕셔너리(Dictionry)

- 여러 개의 데이터를 키(key)와 값(value)을 쌍으로 갖는 자료형
- 딕셔너리는 사전이라는 뜻으로, 키는 실마리, 밸류는 값을 뜻한다.
- 밸류는 딕셔너리 변수에 담긴 정보, 키는 리스트의 인덱스와 비슷한 역할이다.

```
x = {'이름':'홍길동', '나이' : 20}
```

집합(Set)

- 여러 개의 데이터를 중복으로 허용하지 않는 자료형
- 중복으로 넣은 요소는 하나로 표현된다.
- 합집합, 교집합, 차집합 연산을 할 수 있다.

```
x = {1, 2, 3, 4, 5}
```

3. 복합 데이터

3-2. 튜플

개념 - 튜플은 소괄호 ( ) 와 콤마 , 를 사용해서 만든다.

〈튜플 기본 구조〉

변수명 = (값1, 값2, 값3, ……)

- 튜플은 리스트와 비슷한 기능을 하지만, ***값을 수정하거나 삭제하는 것이 불가능한 형태의 자료형***이다.
- 튜플의 요소를 추가하거나 수정, 삭제 하고 싶다면, 추가•변경•삭제 데이터를 자기 자신에게 저장하는 형태로 한다.
- 튜플과 함께 사용되는 메소드는 count(), index() 두 개이니 참고하자.

〈사용예시1〉

```
a = ( 1, 2, 3, 4 )
print(a[0]) ····▸ 1
```

〈사용예시1〉

```
a = (2, )
print(a[0]) ····▸ 2
```

튜플이 요소 하나를 가질 때는, 쉼표를 넣어야 함에 유의하자!!
(머신러닝 등을 할 때 유용하게 사용됨)

〈사용예시3〉

```
a  = (3, 4, '가' )
```

튜플도 리스트와 마찬가지로 여러 자료형을 담을 수 있다

- 튜플은 인덱싱 및 슬라이싱으로 값의 요소에 접근할 수 있다. 값의 수정은 불가능하기 때문에 값을 가져오거나 출력하는 것만 가능하다.

종류	설명	사용예시
인덱싱	인덱스(index)를 통해 개별적으로 접근 할 수 있다.	**x = (1, 2, 3) print(x[0]) → 1**
음수 인덱싱	음수 인덱스를 사용하여 인덱싱 가능하다	**x = (1, 2, 3) print(x[-1]) → 3**
슬라이싱	슬라이싱 기법으로 한 번에 여러 개 항목을 추출	**x = (1, 2, 3, 4, 5, 6) print(x[0:3]) -> 1, 2, 3**
음수 슬라이싱	음수 인덱스를 사용하여 슬라이싱이 가능하다	**x = (1, 2, 3, 4, 5, 6) print(x[-4:-1]) -> 3, 4, 5**
인덱스 생략 슬라이싱	시작인덱스와 끝인덱스는 각각 생략 가능	**x = (1, 2, 3, 4, 5, 6) print(x[:3]) -> 1, 2, 3 print(x[3:]) → 4, 5, 6 print(x[:]) → 1, 2, 3, 4, 5, 6**
특정 간격 슬라이싱	슬라이싱 세번째 항목에 간격을 입력해주면 특정간격으로 슬라이싱 가능하다	**x = (1, 2, 3, 4, 5, 6) print(x[0:6:2]) -> 1, 3, 5 print(x[::2]) → 1, 3, 5 print(x[::-1]) → 6, 5, 4, 3, 2, 1**

연습 문제

1 아래의 순서에 맞춰 튜플을 만들고 다루는 연습을 해보자.

1) tu1에 10, 20, 30의 값을 넣은 튜플을 만든다.
2) tu1 튜플의 요소 중 20을 출력해보자. (인덱스 사용)
3) tu1 튜플의 요소 개수를 출력해보자.
4) tu1 에 10이라는 값이 있는지 확인해보자.

5) tu2에 ' forty ' , ' fifty ' 라는 값을 넣은 튜플을 만든다.
6) tu3에 tu1 튜플과 tu2튜플을 연결하여 저장한다.
7) tu3 튜플을 출력해보자.
8) tu1 튜플을 3번 반복해 출력해보자.

2 tu1 = (1, 5 3) 이라는 튜플이 있을 때, 이 튜플을 (1, 2, 3)으로 변경하고 싶다면 슬라이싱과 더하기 연산자를 이용하면 된다.

```
tu1 = (1, 5, 3)
tu1 = tu1 [0:1] + (2,) + tu1 [2:]
#튜플은 요소가 하나일 때 콤마를 사용한다
(숫자 자료형과 구분하기 위해서)
```

위의 코드를 참고 한 후 아래의 문제를 풀어보자.

1) tu1 = (1, 5, 3) 을 (1, 5, 3, 4)로 변경해보자.
2) tu1 = (1, 5, 3) 을 (5, 3)으로 변경해보자
3) tu1 = (1, 5, 3) 을 (1, 3)으로 변경해보자

3 다음의 튜플을 출력 결과와 똑같이 출력되도록 차례로 출력해보자.

menu = ('짜장면', '우동', '짬뽕', '볶음밥')

출력 결과 1 〉 ('짜장면', '우동', '짬뽕', '볶음밥')

출력 결과 2〉 짜장면

출력 결과 3〉 ('우동', '짬뽕', '볶음밥')

연습 문제

4 아래의 조건에 맞춰 순서대로 코딩을 해보자.

1) tuTest라는 이름의 '가', '나', '다' 라는 요소를 가진 튜플이 있다.

2) tuTest에 '라', '마' 요소를 추가해 보자.

3) tuTest의 요소 개수를 출력해보자.

4) tuTest에 '다'라는 값이 있는지 확인하여 있을 경우 해당 요소를 출력해보자.

3. 복합 데이터

3-3. 딕셔너리

개념

- 리스트와 튜플이 여러 개의 자료를 모아서 저장하는 자료형이라면 딕셔너리는 모아 놓은 자료, 즉 각각의 값들에 이름을 붙여서 저장하는 자료형이다.

- 딕셔너리는 사전(dictionary), 키는 실마리(key), 값은 'value'를 뜻한다.

- 딕셔너리는 중괄호 { }와 '키 : 값' 형태의 자료, 쉼표를 사용해 만든다.

〈딕셔너리 기본 구조〉

변수명 = { 키1:값1, 키2:값2, 키3:값3, ….. }

〈사용예시1〉

```
a = { "name":"어벤져스",
      "type":"히어로 무비"}
```

len(a)를 사용하여 길이를 확인해보면 2가 반환된다.
딕셔너리는 **키:값** 형태가 한 쌍으로 인식되는 것에 유의하자

〈사용예시2〉

```
a = { "apple":"사과",
      "banana":"바나나"}
```

〈사용예시3〉

```
a = { 1:"김영희",
      2: "박철수"}
```

키와 값은 숫자,문자,불 타입 다양한 형태로 섞어서 사용 가능하다
키는 변경 불가한 값이 와야한다는 것에 유의

- 리스트와 튜플은 인덱스로 값에 접근하는 방식이지만 딕셔너리의 경우는 ***키를 인덱스처럼 사용하여*** 값에 접근한다.

〈키를 이용해서 값에 접근하는 법〉

```
a = { 'name' : '어벤져스', 'type' : '히어로 무비'}

print( a['name'] ) ·····▶ 어벤져스
print( a.['type'] ) ·····▶ 히어로 무비
```

〈키를 이용해서 값을 변경하는 법〉

```
a = { 'name' : '어벤져스', 'type' : '히어로 무비'}

a['name'] = '아이언맨'
print( a['name'] ) ·····▶ 아이언맨
```

〈키를 이용해서 값을 추가하는 법〉

키값이 기존에 없는 새로운 값이어야 값이 추가된다.
기존에 있는 값이라면 해당 키의 밸류(값)이 변경된다.

```
a = { 'name' : '아이언맨', 'type' : '히어로 무비'}

a ['hero'] = '토니 스타크'
print( a ) ·····▶ { 'name' : '아이언맨', 'type' : '히어로 무비', 'hero' : '토니 스타크 ' }
```

3. 복합 데이터

3-3. 딕셔너리

개념 - 딕셔너리 또한 리스트, 튜플과 마찬가지로 여러 가지의 데이터 타입을 한번에 담을 수 있다. (키에는 변경 불가한 데이터 타입만 들어감에 유의)

- 아래의 사용예시처럼 밸류에 리스트가 있다면, 인덱스를 지정하면 리스트 안의 특정값에 접근할 수 있다.

〈사용예시〉

```
a = { "name" : "어벤져스",
      "type" : "히어로 무비",
      "hero" : "토니 스타크",
      "characters" : ["페퍼 포츠", "제임스 로드", " 오베디아 스탠"]
    }
print(a["characters"][0]) ·····▶ 페퍼 포츠
```

key

- 딕셔너리는 키를 인덱스처럼 사용하기 때문에 키값을 확인하는 것이 중요하다. in 연산자와 get()함수를 사용하면 키값을 확인할 수 있다.

〈in 연산자를 이용해서 키의 유무 확인하는 법〉

```
a = { 'name' : '어벤져스', 'type' : '히어로 무비'}

print('name' in a)  ·····▶ True
```

〈get()함수를 이용해서 키값을 확인하는 법〉

```
a = { 'name' : '어벤져스', 'type' : '히어로 무비'}

print(a.get('name')) ·····▶ 어벤져스
print(a.get('hero')) ·····▶ None
```

get()함수를 사용할 때는 괄호안에 키값을 적고, 키가 존재하면 안의 밸류를, 존재하지 않으면 None을 리턴받는다.

- 딕셔너리의 길이를 구할 때도 len()함수를 사용할 수 있다.

〈len() 함수를 이용해서 딕셔너리의 길이 구하는 법〉

```
a = { 'name' : '어벤져스', 'type' : '히어로 무비'}

print(len(a)) ·····▶ 2
```

딕셔너리의 경우 키:값 형태의 자료가 하나임을 명심하자! (그러므로 키 개수 = 값 개수)가 된다

3. 복합 데이터

3-3. 딕셔너리

개념 - 딕셔너리의 경우, 키와 값에 접근할 수 있는 다양한 메소드를 제공한다.

종류	설명	사용예
딕셔너리명.keys()	딕셔너리의 모든 키를 가져옴	**d = { 'a' : '사과', 'b' : '바나나' } d.keys() → dict_keys(['a', 'b'])**
딕셔너리명.values()	딕셔너리의 모든 값을 가져옴	**d = { 'a' : '사과', 'b' : '바나나' } d.values() → dict_values(['사과', '바나나'])**
딕셔너리명.items()	딕셔너리의 모든 키.값을 가져옴	**d = { 'a' : '사과', 'b' : '바나나' } dct.items() → d_items([{ 'a' : '사과', 'b' : '바나나'])**
딕셔너리명.get(키, 기본값)	해당 키의 해당 값가져옴 기본값 생략하면 True, False 반환 키가 없을 때는 기본값 반환	**d = { 'a' : '사과', 'b' : '바나나' } d.get('a', 1) → '사과' d.get('c', 1) → 1**
딕셔너리명.setdefault(키, 값)	키-값을 추가함 값 생략하면 none으로 저장됨	**d = { 'a' : '사과', 'b' : '바나나' } d.setdefault('c', '당근;) → 'c':'당근'추가 d.setdefault('c') → 'c':None 추가**
딕셔너리명.update(키=값) ---------- 딕셔너리명.update(딕셔너리명)	해당 키의 값을 변경 새로운 키일 경우 키:값 추가	**d = { 'a' : '사과', 'b' : '바나나' } d.update(a='apple') → 사과 대신 apple 저장 d.update(c='당근') -> 'c':'당근 ' 추가 ---------- E = {'c':'당근', 'd':'두리안'} d.update(e)**
딕셔너리명.pop(키)	해당 키의 키-값 쌍을 삭제 후 삭제 값을 반환	**d = { 'a' : '사과', 'b' : '바나나' } d.popitem('b') → '바나나' 반환 print(d) = {'a' : '사과'}**
딕셔너리명.popitem()	마지막 키-값 쌍 삭제 후 삭제 키-값쌍을 튜플로 반환	**d = { 'a' : '사과', 'b' : '바나나' } d.popitem() → ('b', '바나나') 반환 print(d) = {'a' : '사과'}**
del 딕셔너리명[키]	해당 키의 키-값 쌍 삭제	**d = { 'a' : '사과', 'b' : '바나나' } del d['b'] → 'b':'바나나' 삭제 print(d) = {'a' : '사과'}**
딕셔너리명.copy()	딕셔너리 통째로 복사하기	**d = { 'a' : '사과'} d2 = d.copy() → d2 = { 'a' : '사과'}**
딕셔너리명.clear()	딕셔너리 통째로 삭제하기	**d = { 'a' : '사과'} d.clear()**

- 딕셔너리에서 키는 리스트의 인덱스 역할을 하는데 사용자가 이름을 붙일 수 있는 점이 다르다고 보면 된다. 리스트에서 리스트 길이 이상의 인덱스를 사용하면 에러가 발생하듯이, 딕셔너리에서도 존재하지 않는 키를 사용하면 에러가 발생하니 주의하자.

3. 복합 데이터

3-3. 딕셔너리

개념 - 반복문을 이용하면 딕셔너리의 요소들을 출력할 수 있다.

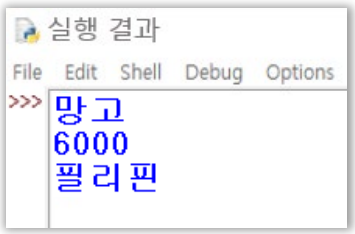
〈딕셔너리와 반복문(for)〉

for 변수 in 딕셔너리:
    코드

```
dct = { "name":"망고",
        "price":6000,
        "origin":"필리핀"}

for key in dct  :
        print(key)
```

for 변수 in 딕셔너리.values():
    코드

```
dct = { "name":"망고",
        "price":6000,
        "origin":"필리핀"}

for value in dct.values()  :
        print(value)
```



for 변수1, 변수2 in 딕셔너리.items():
    코드

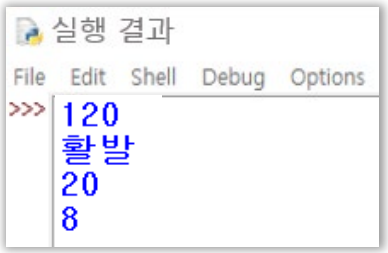
```
dct = { "name":"망고",
        "price":6000,
        "origin":"필리핀"}

for key, value in dct.items()  :
        print(key, value)
```

연습 문제

1 '나의 특징'에 대해서 딕셔너리를 만들려고 한다. 아래 표를 토대로 딕셔너리를 만든 후 출력 결과 처럼 출력해보자.

['키']	['성격']	['몸무게']	['나이']	['성별']
120	'활발'	23	8	'남'

출력 결과〉〉



2 다음과 같이 아이스크림 이름과 가격 정보가 담겨있는 딕셔너리가 있을 때, 아래 아이스크림 가격 정보를 추가한 뒤 출력해보자.

ice = {"메로나": 1000, "폴라포": 1200, "빵빠레": 1800}

이름	가격
죠스바	1200
월드콘	1500

3 (위의 문제에 이어서) 딕셔너리에서 메로나의 가격을 1300으로 수정해보자.

4 (위의 문제에 이어서) 딕셔너리에서 메로나를 삭제해보자.

연습 문제

5 아래의 표에서 아이스크림 이름을 키 값으로, 가격과 재고를 리스트 형태로 값으로 저장한 뒤 출력해보자.

이름	가격	재고
메로나	300	20
비비빅	400	3
죠스바	250	100

6 (위의 문제에 이어서) 메로나의 가격을 화면에 출력해보자.

7 딕셔너리를 이용하여 삼성의 휴대폰과 출시년도를 알아보려고 한다.
빈 칸을 채워 프로그램을 완성해보자.

출력 결과〉〉
```
갤럭시 S5 => 2014
갤럭시 S7 => 2016
갤럭시 노트8 => 2017
갤럭시 S9 => 2018
```

phones = {"갤럭시 S5": 2014, "갤럭시 S7": 2016, "갤럭시 노트8": 2017, "갤럭시 S9": 2018}

```
for key in _____ :
    print('_____')
```

8 아래와 같은 딕셔너리가 있을 때, 'd' :4 키-값 쌍을 삭제 후 출력해보자.
phones = {"a": 1, "b": 2, "c": 3, "d": 4}

연습 문제

9 아래에 4명 학생의 성적이 담긴 딕셔너리가 있다. 반복문을 사용하여 학생들 성적의 합계와 평균, 최대점수, 최소점수를 구해보자.

#. 단, 최대점수, 최소점수를 구할 때 내장함수를 사용하지 말 것
#. 반복문을 사용해서 최대, 최소를 구한다.
#. 점수는 0~100점 사이 값이다.

dctScores = {'황다은':52, '이승빈':55, '이창우':87, '유은재':65}

출력 예〉
합　　계: 259
평　　균: 64.75
최소점수: 52
최대점수: 87

10 위의 문제를 딕셔너리의 메소드 : 딕셔너리명.values() 를 사용하여 반복문을 사용하지 않고 풀어보자.

11 다음의 딕셔너리로부터 key값으로만 구성된 리스트를 생성해보자.

ice = {'탱크보이':1200, '폴라포':900, '월드콘':1500, '메로나':1000}

12 (위의 문제에 이어서) 이번에는 딕셔너리로부터 value 값만 가져와 sum() 메소드를 이용하여 총합을 출력해보자.

13 (위의 문제에 이어서) 아래의 새로운 딕셔너리가 있을 때, 딕셔너리명.update() 메소드를 사용하여 항목을 추가해보자.

연습 문제

15 게임 캐릭터를 만들려고 한다. 아래 순서에 맞춰 캐릭터의 속성을 코딩해보자.

이름	Warrior
힘	88
마 법	65
민 첩	80
체 력	90

1) 위 표를 기반으로 Warrior 캐릭터의 딕셔너리를 생성한다.

2) Warrior의 모든 능력치를 화면에 아래와 같이 출력
해보자
(#반복문 사용, 딕셔너리명.items() 메소드를 사용한다)

('이름', 'Warrior')
('힘', 88)
('마법', 65)
('민첩', 80)
('체력', 90)

3) 몬스터를 잡느라 '체력'이 30 소진 되었다. 체력 값을 수정하자.

4) 몬스터를 잡은 보상으로 방패를 획득하였다. '방어' key를 가진 요소가 있는지 확인하고, 없다면 (Key '방어', Value 70) 요소를 추가하자.

5) 현재 Warrior의 능력치를 다시 출력하여 확인하자.

연습 문제

16 아이디와 비밀번호를 체크하는 프로그램을 만들려고 한다. 아이디와 비밀번호의 정보가 다음과 같은 딕셔너리에 담겨있을 때, 사용자로부터 아이디와 비밀번호를 입력받은 뒤 정보에 접근 권한이 있는지 없는지 확인해주도록 프로그램을 만들어보자.

ad = { " id " : " admin " , " password":"12345"}

출력 결과〉〉

```
아이디를 입력하세요: admin
비밀번호를 입력하세요: 12345
모든 정보에 접근 권한이 있습니다!
```

출력 결과〉〉

```
아이디를 입력하세요: ssss
비밀번호를 입력하세요: 12343
정보에 접근 권한이 없습니다!
```

17 다음은 딕셔너리를 이용하여 학생 성적의 합계와 평균을 구하는 프로그램이다. 밑줄 친 부분을 채워 코드를 완성해보자.

```
scores = {"김채린": 85, "박수정": 98, "함소희": 94, "안예린": 90, "연수진": 93}

sum = 0

for key in scores :
    sum = sum + _____

avg = sum/_____

print(f"합계 : {sum}, 평균 : {avg}" )
```

연습 문제

17 다음은 딕셔너리를 이용하여 영어 단어 퀴즈를

```
words = {"꽃":"flower", "나비":"butterfly", "학교":"school", "자동차":"car",
"비행기":"airplane"}

print("〈영어 단어 맞추기 퀴즈〉")

for ____________ in words :
    in_word = input(f'{____} 에 해당되는 영어 단어를 입력해주세요: ')

    if ____________ == ________________ :
        print("정답입니다!")
    else :
        print("틀렸습니다!")
```

3. 복합 데이터

3-4. 집합(set)

개념

- 집합은 수학에서 사용하는 집합의 개념과 동일하다.
- 집합은 중괄호 { }와 쉼표를 사용해 만든다.

〈딕셔너리 기본 구조〉

변수명 = { 값1, 값2, 값3, ….. }

- 집합 자료형은 중복되는 요소가 모두 하나로 표현된다.
- 집합 자료형은 요소들이 오름차순으로 저장된다.
- 합집합(|), 교집합(&), 차집합(-)을 사용할 수 있다.

〈사용예시1〉

```
a = { 1, 2, 3, 3, 3}
print(a)
-〉 {1, 2, 3}
```

〈사용예시2〉

```
a = { 3, 2, 1 }
print(a)
-〉 {1, 2, 3}
```

〈사용예시3〉

```
a = { 1, 2, 3 }
B = { 3, 4, 5 }
print(a | b) -〉 {1, 2, 3, 4, 5}
print(a & b) -〉 {3}
print(a - b) -〉 {1, 2}
```

종류	설명	사용예
집합명.add(요소)	집합에 요소 추가하기	**a = {1, 2, 3, 4} a.add(5) -> {1, 2, 3, 4, 5}**
집합명.remove(요소)	집합에서 특정 요소 삭제	**a = {1, 2, 3, 4} a.remove(3) -> {1, 2, 4}**
집합명.discard(요소)	집합에서 특정 요소를 삭제하는데 없으면 그냥 넘어감	**a = {1, 2, 3, 4} a.discard(5) -> {1, 2, 3, 4}**
집합명.pop()	집합에서 임의의 요소 삭제하고 반환	**a = {1, 2, 3, 4} print(a.pop()) -> { 2, 3, 4}**
집합명.clear()	집합에서 모든 요소 삭제	**a = {1, 2, 3, 4} a.clear -> set()**

- 집합은 a={} 형태로 사용하게 되면 딕셔너리로 정의되기 때문에 빈 세트의 경우에는 a = set() 의 형태로 사용한다.

연습 문제

1 다음 중 집합을 만드는 방법으로 틀린 것은?

```
a = {1, 2, 3, 4, 5}
a = {}
a = set('hello')
a = set(range(10))
a = set()
```

2 다음과 같은 집합이 있을 때, 5 라는 요소를 추가해보자.

```
a = { 1, 2, 3, 4}
```

3 (위의 문제에 이어서) 요소 3을 삭제해보자.

4 (위의 문제에 이어서) 집합의 개수를 출력해보자.

5 (위의 문제에 이어서) 다음과 같은 집합이 있을 때, a와 b의 합집합, 차집합, 교집합을 차례대로 출력해보자.