리스트 자료형과 튜플 자료형

1 리스트(List) 자료형

① 리스트 자료형이란?

다양한 자료형을 순차적으로 저장하는 집합적 자료형

- 문자열이 지닌 대부분의 연산은 리스트도 지원함
- 대괄호로 정의함 ➡ l = [1, 2, 3]
- 다른 프로그래밍 언어(C, C++) 등과 달리 동적 배열, 다차원 배열,
  인덱싱 등을 훨씬 쉽고 편리하게 사용할 수 있음

```
l = list()
print(l, type(l))
```

```
[] <class 'list'>
```

```
l = [1,2,3]
print(type(l))
print(l)
```

```
<class 'list'>
[1, 2, 3]
```

리스트 자료형과 튜플 자료형

1 리스트(List) 자료형

② 리스트 자료형의 특징

1 문자열과 같이 인덱스와 슬라이싱 연산 가능

```
l = [1,2,3,4,5,6,7,8,9]
print(l[0])
```

```
1
```

```
print(l[0:4])
```

```
[1, 2, 3, 4]
```

```
print(l[5])
```

```
6
```

인덱스는 0부터 시작하기 때문에 리스트의 길이를 구한 다음 -1을 함

```
print(l[len(l)-1])
```

```
9
```

리스트 자료형과 튜플 자료형

1 리스트(List) 자료형

② 리스트 자료형의 특징

2 요소의 값 변경 가능

- 인덱스를 활용해 요소 값을 접근·변경할 수 있음

```
l = [1,2,3,4,5,6,7,8,9]
l[0] = 99
print(l)
```

```
[99, 2, 3, 4, 5, 6, 7, 8, 9]
```

리스트 자료형과 튜플 자료형

1 리스트(List) 자료형

② 리스트 자료형의 특징

2 요소의 값 변경 가능

- 리스트의 요소로 다른 자료형이 올 수 있음

```
l[1] = [1,2,3]
l[2] = "문자"
print(l)
```

```
[99, [1, 2, 3], '문자', 4, 5, 6, 7, 8, 9]
```

리스트 자료형과 튜플 자료형

1 리스트(List) 자료형

2 리스트 자료형의 특징

3 여러 함수 활용 가능

- 선언한 리스트에서 . + tab 키를 눌러 사용할 수 있는 함수 확인 가능

```
l.append
l.clear
l.copy
l.count
l.extend
l.index
l.insert
l.pop
l.remove
l.reverse
l.
```

```
l = [1,2,3,4,5]
print(l)
l.append(6)
print(l)
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 6]
```

```
l = ['a','b','c','d']
print(l)
l.remove('b')
print(l)
```

```
['a', 'b', 'c', 'd']
['a', 'c', 'd']
```

리스트 자료형과 튜플 자료형

2 튜플(Tuple) 자료형

① 튜플 자료형이란?

다양한 자료형을 순차적으로 저장하는 집합적 자료형

- 리스트와 비슷하지만 값을 변경할 수 없는 특징이 있음
- 소괄호로 정의함 ➡ t = (1, 2, 3)

리스트 자료형과 튜플 자료형

2 튜플(Tuple) 자료형

① 튜플 자료형이란?

```
t = tuple()
print(t, type(t))
```

```
() <class 'tuple'>
```

```
t = (1,2,3)
print(type(t))
print(t)
```

```
<class 'tuple'>
(1, 2, 3)
```

리스트 자료형과 튜플 자료형

2 튜플(Tuple) 자료형

② 튜플 자료형의 특징

1 리스트와 비슷한 자료형
:인덱싱, 슬라이싱 등의 연산 가능

- 선언한 튜플에서 . + tab 키를 눌러 사용할 수 있는 함수 확인 가능

```
l = [1,2,3]
t = (1,2,3)
print(l, type(l))
print(t, type(t))
```

```
[1, 2, 3] <class 'list'>
(1, 2, 3) <class 'tuple'>
```

```
print(l[0], l[0:2])
print(t[0],t[0:2])
```

```
1 [1, 2]
1 (1, 2)
```

```
print(l + l)
print(t + t)
```

```
[1, 2, 3, 1, 2, 3]
(1, 2, 3, 1, 2, 3)
```

리스트 자료형과 튜플 자료형

2 튜플(Tuple) 자료형

② 튜플 자료형의 특징

2 리스트와의 차이점 : 값의 변경이 불가능

- 리스트에서 활용할 수 있는 여러 함수들도 튜플에서 사용할 수 없음
- 상수적인 특징을 가지고 있기 때문에 리스트보다 연산에 빠른 장점이 있음

리스트 자료형과 튜플 자료형

2 튜플(Tuple) 자료형

② 튜플 자료형의 특징

2 리스트와의 차이점 : 값의 변경이 불가능

```
l = [1,2,3]
t = (1,2,3)
print(l, type(l))
print(t, type(t))
```

```
[1, 2, 3] <class 'list'>
(1, 2, 3) <class 'tuple'>
```

```
l[0] = 5
print(l)
```

```
[5, 2, 3]
```

```
t[0] = 5
print(t)
```

```
----------------------------------------------
TypeError                                 Tra
<ipython-input-27-0fff53691999> in <module>()
----> 1 t[0] = 5
      2 print(t)

TypeError: 'tuple' object does not support it
```

사전 자료형과 집합 자료형

1 사전(Dict) 자료형

① 사전 자료형이란?

키를 이용하여 값을 저장하는 자료형

- 정수형 인덱스가 아닌 키로 값을 저장하기 때문에 저장된 자료의 순서는 의미가 없음
- 중괄호로 정의함 ➡ d = {'a' = 1, 'b' = 2, 'c' = 3}

사전 자료형과 집합 자료형

1 사전(Dict) 자료형

① 사전 자료형이란?

```
d = dict()
print(d, type(d))
```

```
{} <class 'dict'>
```

```
d = {
    'a' : 1,
    'b' : 2,
    'c' : 3
}
print(type(d))
print(d)
```

```
<class 'dict'>
{'a': 1, 'b': 2, 'c': 3}
```

사전 자료형과 집합 자료형

1 사전(Dict) 자료형

② 사전 자료형의 특징

1 정수형 인덱스가 아닌 **키**와 **값**으로 자료를 저장

```
d = {
    1 : 1,
    'a' : [1,2,3],
    (1,2) : "aaa"
}

print(d)
```

```
{1: 1, 'a': [1, 2, 3], (1, 2): 'aaa'}
```

사전 자료형의 키 값은 변경 불가능한 객체 (튜플 자료형 포함)가 올 수 있음

사전 자료형과 집합 자료형

1 사전(Dict) 자료형

② 사전 자료형의 특징

2 값의 추가, 수정이 매우 용이함

- 기존 사전에 있는 키에 새로운 값을 선언하면 새로운 값으로 변경됨
- 사전에 없는 키라면, 새로운 값이 추가됨

```
d = {'a' : 1, 'b' : 2}
print(d)
```

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

```
d['a'] = 2
print(d)
```

```
{'a': 2, 'b': 2}
```

```
d['c'] = 3
print(d)
```

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

사전 자료형과 집합 자료형

1 사전(Dict) 자료형

② 사전 자료형의 특징

2 값의 추가, 수정이 매우 용이함

- 기존에 없는 키를 단순히 참조한다면 에러 발생

```
print(d['d'])
```

```
---------------------------
KeyError
<ipython-input-38-c19e1
----> 1 print(d['d'])

KeyError: 'd'
```

사전 자료형과 집합 자료형

1 사전(Dict) 자료형

② 사전 자료형의 특징

3 사전 자료형의 함수 활용(Keys, Values, Items)

- 함수를 활용해 키, 값을 따로 뽑아 리스트에 반환할 수 있음
- Json, XML 형식과 유사하기 때문에 NoSQL, OPEN API , 빅데이터 분석 등에 많이 활용됨

사전 자료형과 집합 자료형

1 사전(Dict) 자료형

2 사전 자료형의 특징

3 사전 자료형의 함수 활용(Keys, Values, Items)

```
d = {
      'a' : 1,
      'b' : 2,
      'c' : 3
}
print(type(d))
print(d)
```

```
<class 'dict'>
{'a': 1, 'b': 2, 'c': 3}
```

```
print(d.keys())
print(d.values())
print(d.items())
```

```
dict_keys(['a', 'b', 'c'])
dict_values([1, 2, 3])
dict_items([('a', 1), ('b', 2), ('c', 3)])
```

사전 자료형과 집합 자료형

2 집합(Set) 자료형

① 집합 자료형이란?

중복과 순서가 없는 자료형

- 순서가 없기 때문에 인덱싱 또한 지원하지 않음
- 중괄호로 정의함 ➡ s = {1,2,3,4,5}

집합! 파이썬의 집합은 교집합, 합집합 등 수학의 집합론을 따르고 있음

사전 자료형과 집합 자료형

2 집합(Set) 자료형

① 집합 자료형이란?

```
s = {1,2,3,4}
print(s, type(s))
```

```
{1, 2, 3, 4} <class 'set'>
```

```
s = set("Hello")
print(s)
```

```
{'o', 'e', 'H', 'l'}
```

```
s = set()
print(s, type(s))
```

```
set() <class 'set'>
```

사전 자료형과 집합 자료형

2 집합(Set) 자료형

② 집합 자료형의 특징

1 교집합, 합집합, 차집합의 연산도 지원

- 중복, 순서가 없음

```
s1 = set([1, 2, 3, 4, 5, 6])
s2 = set([4, 5, 6, 7, 8, 9])
print(s1)
print(s2)
```

```
{1, 2, 3, 4, 5, 6}
{4, 5, 6, 7, 8, 9}
```

```
print(s1&s2)          #교집합
```

```
{4, 5, 6}
```

```
print(s1|s2)          #합집합
print(s1.union(s2))
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
print(s1-s2)          #차집합
print(s1.difference(s2))
```

```
{1, 2, 3}
{1, 2, 3}
```

사전 자료형과 집합 자료형

2 집합(Set) 자료형

② 집합 자료형의 특징

2 추가, 삭제 방법

- 하나의 값을 추가하려면 add 함수 활용

```
s.add(4) # 값 추가
print(s)
```

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

사전 자료형과 집합 자료형

2 집합(Set) 자료형

② 집합 자료형의 특징

2 추가, 삭제 방법

- 여러 값을 추가할 때는 update 함수로 리스트에 값을 넣어 추가

```
s.update([5]) #값 추가
print(s)
```

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

```
s.update([6,7])
print(s)
```

```
{1, 2, 3, 4, 5, 6, 7}
```

사전 자료형과 집합 자료형

2 집합(Set) 자료형

② 집합 자료형의 특징

2 추가, 삭제 방법

- 삭제는 remove 함수를 활용

```
s.remove(5) #값 삭제
print(s)
```

```
{1, 2, 3, 4, 6, 7}
```

Run! 프로그래밍

Mission 1 | 리스트의 중복된 값을 제거하기

L = [1,1,1,1,2,2,2,3,3,4,5,6,4,4,5,7,7,8,9,7,7,7,5,6,7]

정답

```
temp = set(L)
L = list(temp)

print(L)
```

Mission 2 | 튜플의 값을 추가, 수정, 삭제하기

T = (1, 2, 3, 4, 5, 6, 7, 8)

정답

```
L = list(T)
L[0] = 0
L.remove(8)
L.append(9)

print(L)
```