파이썬 프로그래밍

튜플과 집합

한국기술교육대학교
온라인평생교육원

▣ 튜플 활용법

1. 튜플 연산

- 튜플(Tuples): 순서있는 임의의 객체 모음 (시퀀스형)
- 튜플은 변경 불가능(Immutable)
- 시퀀스형이 가지는 다음 연산 모두 지원
  - 인덱싱, 슬라이싱, 연결, 반복, 멤버쉽 테스트 .

```
t1 = ()      # 비어있는 튜플
t2 = (1,2,3) # 괄호 사용

t3 = 1,2,3   # 괄호가 없어도 튜플이 됨
print type(t1), type(t2), type(t3)
```

```
<type 'tuple'> <type 'tuple'> <type 'tuple'>
```

- 시퀀스형 → 리스트와 튜플이 공통으로 가져가는 특징
- 리스트와의 차별점 → 튜플은 변경 불가능
- 튜플 → ( ) 을 사용하여 정의
- ( ) 을 사용하지 않아도 튜플이 될 수 있음
- ( ) 가 없이 연속적인 객체를 콤마(,)를 이용하여 나열하면 튜플 가능

▣ 튜플 활용법

1. 튜플 연산

```
r1 = (1,) # 자료가 한 개일 때는 반드시 콤마가 있어야 한다.
r2 = 1,   # 괄호는 없어도 콤마는 있어야 한다.
print type(r1)
print type(r2)
```

```
<type 'tuple'>
<type 'tuple'>
```

- 튜플 안 원소가 하나 → 튜플로 보지 않음
- 원소가 하나인 튜플 → 콤마(,)를 사용
- 괄호가 없어도 콤마 있으면 튜플로 인식

▣ 튜플 활용법

1. 튜플 연산

```
t = (1, 2, 3)
print t * 2                 # 반복
print t + ('PyKUG', 'users')  # 연결
print t
print

print t[0], t[1:3]          # 인덱싱, 슬라이싱
print len(t)                # 길이
print 1 in t                # 멤버십 테스트
```

```
(1, 2, 3, 1, 2, 3)
(1, 2, 3, 'PyKUG', 'users')
(1, 2, 3)

1 (2, 3)
3
True
```

- t 자체가 변경되는 것이 아니라 새로운 튜플이 반환
- t[0] → 인덱싱 연산
- t[1:3] → 슬라이싱 연산
- len(t) → t 안에 있는 원소 개수
- 1 in t → 1이라는 객체가 t 안에 존재 하는지 확인

▣ 튜플 활용법

1. 튜플 연산

```
t[0] = 100 # 튜플은 변경 불가능, 에러발생
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-ded308ebf8b1> in <module>()
----> 1 t[0] = 100 # 튜플은 변경 불가능, 에러발생

TypeError: 'tuple' object does not support item assignment
```

- 튜플은 변경할 수 없음

▣ 튜플 활용법

1. 튜플 연산

```
t = (12345, 54321, 'hello!')
u = t, (1, 2, 3, 4, 5) # 튜플 내부 원소로 다른 튜플을 가질 수 있음
print u

t2 = [1, 2, 3] # 튜플 내부 원소로 리스트 가질 수 있음
u2 = t2, (1, 2, 4)
print u2

t3 = {1:"abc", 2:"def"} # 튜플 내부 원소로 사전 가질 수 있음
u3 = t3, (1, 2, 3)
print u3
```

```
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
([1, 2, 3], (1, 2, 4))
({1: 'abc', 2: 'def'}, (1, 2, 3))
```

- t = (정수, 정수, 문자열)
- u = t, (1, 2, 3, 4, 5) → 2개의 원소를 가진 튜플 → 원소로 튜플 가능
- u2 = t2, (1, 2, 3) → 원소로 리스트 가능
- 원소로 사전도 가능

▣ 튜플 활용법

1. 튜플 연산

```
x, y, z = 1, 2, 3 # 튜플을 이용한 복수 개의 자료 할당
print type(x), type(y), type(z)
print x
print y
print z
```

```
<type 'int'> <type 'int'> <type 'int'>
1
2
3
```

- 튜플을 이용한 복수 개의 자료 할당
- x → 1, y → 2, z → 3
- 1, 2, 3 = (1, 2, 3) → 튜플
- 튜플을 활용하여 원소의 변수 값 치환 가능

```
x = 1
y = 2
x, y = y, x # 튜플을 이용한 두 자료의 값 변경
print x, y
```

```
2 1
```

- x → y, y → x = x값과 y값 서로 교환

▣ 튜플 활용법

2. 패킹과 언패킹

• 패킹 (Packing): 하나의 튜플 안에 여러 개의 데이터를 넣는 작업

```
t = 1, 2, 'hello'
```

- 1, 2,'hello' = (1, 2, 'hello')
- t = (1, 2, 'hello') → 패킹 = 3개의 객체를 하나의 변수에 묶는 것

• 언패킹 (Unpacking): 하나의 튜플에서 여러 개의 데이터를 한꺼번에 꺼내와 각각 변수에 할당하는 작업

```
x, y, z = t
```

- 언패킹 = 묶여 있는 객체는 푸는 것

• 리스트로도 비슷한 작업이 가능하지만, 단순 패킹/언패킹 작업만을 목적으로 한다면 튜플 사용 추천

```
a = ['foo', 'bar', 4, 5]
[x, y, z, w] = a
```

- 단순 패킹/언패킹 작업 목적 → 리스트 보단 튜플 사용

▣ 튜플 활용법

2. 패킹과 언패킹

- 튜플과 리스트와의 공통점
  - 원소로서 임의의 객체를 저장
  - 시퀀스 자료형
    * 인덱싱, 슬라이싱, 연결, 반복, 멤버쉽 테스트 연산 지원
- 리스트와 다른 튜플만의 특징
  - 변경 불가능 (Immutable)
  - 메소드를 가지지 않는다.
- list() 와 tuple() 내장 함수를 사용하여 리스트와 튜플을 상호 변환할 수 있음

```
T = (1,2,3,4,5)
L = list(T)
L[0] = 100
print L

T = tuple(L)
print T
```

```
[100, 2, 3, 4, 5]
(100, 2, 3, 4, 5)
```

- 튜플은 메소드를 가지지 않음
- 리스트 메소드 → append(), extend(), pop().....
- 튜플은 변경이 불가능하므로 메소드 필요 X
- L = list(T) → T라는 튜플이 리스트로 변환됨

▣ 튜플 활용법

3. 튜플의 사용 용도

- 튜플을 사용하는 경우 1: 함수가 하나 이상의 값을 리턴하는 경우

```
f calc(a, b):
    return a+b, a*b

x, y = calc(5, 4)
```

- T = tuple(L) → L 이라는 리스트가 튜플로 변환됨
- 리턴을 할 때 2개의 원소 같이 리턴 → 튜플 괄호 숨어있음
- 리턴한 결과가 튜플한 뒤 언패킹 진행
- 튜플은 동시에 여러 개의 값 리턴 가능

- 튜플을 사용하는 경우 2: 문자열 포멧팅

```
print 'id : %s, name : %s' % ('gslee', 'GangSeong')
```

```
id : gslee, name : GangSeong
```

- %s → 문자열을 이 자리에 위치
- 문자열 포맷팅 시 % 뒤에 튜플 사용

▣ 튜플 활용법

3. 튜플의 사용 용도

- 튜플을 사용하는 경우 3: 고정된 값을 쌍으로 표현하는 경우

```
d = {'one':1, 'two':2}
print d.items()
```

```
[('two', 2), ('one', 1)]
```

- 사전의 원소 기준 → 콤마(,)
- 원소 : item
- 결과는 리스트, 원소는 튜플

파이썬 프로그래밍

튜플과 집합

한국기술교육대학교
온라인평생교육원

▣ 집합 자료형

1. 집합 자료형 생성

- set 내장 함수를 사용한 집합 자료 생성
  - 변경 가능(Mutable)한 객체이다.
  - 각 원소간에 순서는 없다.
  - 각 원소는 중복될 수 없다.
  - [note] 시퀀스 자료형이 아니다.

- set 내장함수를 활용하여 생성하는 자료형
- 순서가 없음 = 시퀀스 자료형이 아님 → 인덱싱, 슬라이싱 X
- 1, 2, 2, 3 = 1, 2, 3

▣ 집합 자료형

1. 집합 자료형 생성

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

b = set((1, 2, 3))
print type(b)
print b
print

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

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

```
<type 'set'>
set([1, 2, 3])

<type 'set'>
set([1, 2, 3])

<type 'set'>
set(['a', 'c', 'b'])

<type 'set'>
set([1, 2, 3])
```

▣ 집합 자료형

1. 집합 자료형 생성

- set 내장함수 인자 → 리스트, 튜플, 사전 가능
- set 안에 튜플이 들어가도 출력은 리스트로 됨
- 인자에 사전이 들어가면 value는 무시되고 key 값만 가져옴
- 사전에는 key 값이 value보다 중요한 역할
- Set 안에 사전이 들어가도 출력은 리스트
- 사전 자료형에 values() 함수 사용시 → 사전의 key값은 무시되고 value 값만 출력

▣ 집합 자료형

1. 집합 자료형 생성

- set의 원소로는 변경 불가능(Immutable)한 것만 할당 가능하다.

```
print set() # 빈 set 객체 생성
print set([1,2,3,4,5]) # 초기 값은 일반적으로 시퀀스 자료형인 리스트를 넣어준다.
print set([1,2,3,2,3,4]) # 중복된 원소는 한 번만 표현
print set('abc') # 문자열은 각 문자를 집합 원소로 지닌다.
print set([(1,2,3),(4,5,6)]) # 각 튜플은 원소로 가질 수 있음
print set([[1,2,3],[4,5,6]]) # 변경 가능 자료인 리스트는 집합의 원소가 될 수 없다.
```

```
[set([])
set([1, 2, 3, 4, 5])
set([1, 2, 3, 4])
set(['a', 'c', 'b'])
set([(4, 5, 6), (1, 2, 3)])

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-7f19fe538c1f> in <module>()
      4 print set('abc')              # 문자열은 각 문자를 집합 원소로 지닌다.
      5 print set([(1,2,3),(4,5,6)]) # 각 튜플은 원소로 가질 수 있음
----> 6 print set([[1,2,3],[4,5,6]]) # 변경 가능 자료인 리스트는 집합의 원소가 될 수 없다.

TypeError: unhashable type: 'list'
```

- Set의 원소로 변경 불가능한 것만 가능
- 중복된 원소는 하나로 인식
- set 내장함수 인자로 문자열도 가능 → 문자 하나하나가 나누어짐
- 리스트는 변경 가능한 자료이므로 원소 X

▣ 집합 자료형

1. 집합 자료형 생성

- set의 기본 연산

set 연산	동일 연산자	내용
len(s)		원소의 개수
x in s		x가 집합 s의 원소인가?
x not in s		x가 집합 s의 원소가 아닌가?

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

print len(A)      # 집합의 원소의 수
print 5 in A      # 멤버십 테스트
print 10 not in A # 멤버십 테스트
```

```
9
True
True
```

- 기본 연산 = 슬라이싱, 인덱싱..... → set 적용 불가능
- len(A) → A 집합의 원소 개수
- in, not in → 원소가 존재 하는지 하지 않는지 확인

▣ 집합 자료형

2. 집합 자료형 메소드

- set의 주요 메소드
  - 다음 연산은 원래 집합은 변경하지 않고 새로운 집합을 반환한다.

set 연산	동일 연산자	내용
len(s)		원소의 개수
x in s		x가 집합 s의 원소인가?
x not in s		x가 집합 s의 원소가 아닌가?

```
B = set([4,5,6,10,20,30])
C = set([10,20,30])

print C.issubset(B)    # C가 B의 부분집합?
print C <= B
print B.issuperset(C)  # B가 C를 포함하는 집합?
print B >= C
print
```

```
True
True
True
True
```

- C의 원소들이 B의 원소에 포함됨
- C.Issubset(B) → C가 B의 부분집합인지를 확인
- .issubset = '<='
- C.Issuperset(B) → B가 C를 포함되는지 확인
- .issuperset = '>='

▣ 집합 자료형

2. 집합 자료형 메소드

```
A = set([1,2,3,4,5,6,7,8,9])
B = set([4,5,6,10,20,30])

print A.union(B) # A와 B의 합집합
print A
print

print A.intersection(B) # A와 B의 교집합
print A
print

print A.difference(B) # A - B (차집합)
print A
print

print A.symmetric_difference(B) # 베타집합. A와 B의 합집합에서 교집합의 원소를
                                제외한 집합
print A
```

```
set([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30])
set([1, 2, 3, 4, 5, 6, 7, 8, 9])

set([4, 5, 6])
set([1, 2, 3, 4, 5, 6, 7, 8, 9])

set([1, 2, 3, 7, 8, 9])
set([1, 2, 3, 4, 5, 6, 7, 8, 9])

set([1, 2, 3, 7, 8, 9, 10, 20, 30])
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

▣ 집합 자료형

2. 집합 자료형 메소드

- A.union(B) → A와 B의 합집합
- union을 쓰면 합쳐서 반환되는 것이지 A는 변경 X
- intersection → 교집합
- A.difference(B) → 차집합(A-B)
- A.symmetric_difference(B) → 베타집합
- 베타집합 = A와 B의 합집합에서 교집합 원소를 제외한 집합

▣ 집합 자료형

2. 집합 자료형 메소드

```
A = set([1,2,3,4,5,6,7,8,9])
B = set([4,5,6,10,20,30])

print A | B # A와 B의 합집합
print A
print

print A & B # A와 B의 교집합
print A
print

print A - B #A - B (차집합)
print A
print

print A.symmetric_difference(B) # 베타집합. A와 B의 합집합에서 교집합의 원소를
                                제외한 집합
print A ^ B
print A
```

```
set([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30])
set([1, 2, 3, 4, 5, 6, 7, 8, 9])

set([4, 5, 6])
set([1, 2, 3, 4, 5, 6, 7, 8, 9])

set([1, 2, 3, 7, 8, 9])
set([1, 2, 3, 4, 5, 6, 7, 8, 9])

set([1, 2, 3, 7, 8, 9, 10, 20, 30])
set([1, 2, 3, 7, 8, 9, 10, 20, 30])
set([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

▣ 집합 자료형

2. 집합 자료형 메소드

- union = ' | ' (오와 연산)
- intersection = ' & '
- difference = ' - '
- symmetric differnce → ' ^ '

▣ 집합 자료형

2. 집합 자료형 메소드

```
A = set([1,2,3,4,5,6,7,8,9])
D = A.copy()
print D
print

print A == D #자료값 비교
print A is D #객체 동등성 비교
```

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

True
False
```

- D = A.copy( ) A 내용은 그대로 가져오나 D라는 새로운 집합 생성
- A == D → A와 D의 자료값 비교
- is → 객체 동등성 비교
- copy는 객체가 새로 생성되는 것이므로 A와 B는 다른 객체

▣ 집합 자료형

2. 집합 자료형 메소드

- set은 시퀀스 자료형이 아니므로 인덱싱, 슬라이싱, 정렬 등을 지원하지 않는다.

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

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-a0fca37959a3> in <module>()
      1 A = set([1,2,3,4,5,6,7,8,9])
----> 2 print A[0]

TypeError: 'set' object does not support indexing
```

- A[0] → 인덱싱 연산 → A에서 0번째 인덱스 반환
- set은 시퀀스 자료형이 아니라 순서가 없음

▣ 집합 자료형

2. 집합 자료형 메소드

```
print A[1:4]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-795e14598526> in <module>()
----> 1 print A[1:4]

TypeError: 'set' object has no attribute '__getitem__'
```

- 슬라이싱 연산도 지원되지 않음

```
print A.sort()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-11-9fbe6777337f> in <module>()
----> 1 print A.sort()

AttributeError: 'set' object has no attribute 'sort'
```

- sort( ) 도 마찬가지로 지원되지 않음

▣ 집합 자료형

2. 집합 자료형 메소드

- 집합을 리스트나 튜플로 변경가능
  - 집합에 인덱싱, 슬라이싱, 정렬 등을 적용하기 위해서는 리스트나 튜플로 변경한다.

```
print list(A)
print tuple(A)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

- list, tuple 함수 지원

- 하지만 집합에 for ~ in 연산은 적용 가능하다.

```
A = set([1,2,3,4,5,6,7,8,9])
for ele in A:
    print ele,
```

```
1 2 3 4 5 6 7 8 9
```

- for문은 지원

▣ 집합 자료형

2. 집합 자료형 메소드

- set은 변경 가능(Mutable)한 자료 구조 객체
- 다음 메소드들은 set을 변경하는 집합 자료 구조 메소드들임

set 연산	동일 연산자	내용
s.update(t)	s |= t	s와 t의 합집합을 s에 저장
s.intersection_update(t)	s &= t	s와 t의 교집합을 s에 저장
s.difference_update(t)	s -= t	s와 t의 차집합을 s에 저장
s.symmetric_difference_update(t)	s ^= t	s와 t의 배타집합을 s에 저장
s.add(x)		원소 x를 집합 s에 추가
s.remove(x)		원소 x를 집합 s에서 제거, 원소 x가 집합 s에 없으면 예외 발생
s.discard(x)		원소 x를 집합 s에서 제거
s.pop()		임의의 원소를 집합 s에서 제거, 집합 s가 공집합이면 예외 발생
s.clear()		집합 s의 모든 원소 제거

- update → . 앞에 있는 집합 객체의 내용 변경

▣ 집합 자료형

2. 집합 자료형 메소드

```
A = set([1,2,3,4])
B = set([3,4,5,6])

A.update(B) # A에 B 집합의 원소를 추가 시킴
print A

A.intersection_update([4,5,6,7,8]) # &=
print A

A.difference_update([6,7,8]) # -=
print A

A.symmetric_difference_update([5,6,7]) # ^=
print A

A.add(8)    # 원소 추가
print A

A.remove(8) # 원소 제거
print A
```

```
set([1, 2, 3, 4, 5, 6])
set([4, 5, 6])
set([4, 5])
set([4, 6, 7])
set([8, 4, 6, 7])
set([4, 6, 7])
```

▣ 집합 자료형

2. 집합 자료형 메소드

- update → A에 B의 내용 추가
- A와 B에 중복으로 있는 내용은 제외하고 추가
- update = union (합집합)
- update는 union과 다르게 A가 변경된다는 것이 중요
- intersection_update → 교집합의 결과가 반영됨
- _update 없으면 A는 변경 X
- _update가 있어야 A 변경 가능
- symetric_difference_update → 합집합에서 교집합 내용 빼기
- add( ) → 단순히 원소를 더하기
- remove( ) → 단순히 원소 없애기
- ' s.update(t) ' = ' s |= t '
- 'A. intersection_update(t)' = ' A &= t '
- difference_update = ' -= '
- symmetric_difference_update = ' ^= '

▣ 집합 자료형

2. 집합 자료형 메소드

```
A.remove(10) # 없는 원소를 제거하면 KeyError 발생
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-21-1bcb28846ad3> in <module>()
----> 1 A.remove(10) #없는 원소를 제거하면 KeyError 발생

KeyError: 10
```

- remove는 원소가 없으면 error, discard는 error X

```
A.discard(10) # remove와 같으나 예외가 발생하지 않음
A.discard(6)  # 원소 6제거
print A

A.pop() # 임의의 원소 하나 꺼내기
print A

A = set([1,2,3,4])
A.clear() # 원소들 없애기
print A
```

```
set([4, 7])
set([7])
set([])
```

- pop( ) → 임의의 원소 하나 꺼내기 (랜덤으로 꺼내기)
- clear() → 원소 모두 없애기