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