06. 튜플

튜플은 변경 가능하지 않은 데이터의 묶음이다. 리스트와 유사하지만 튜플이 한번 만들어지면 값은 변경할 수 없다.

6.1 뉴플의 연산

튜플은 임의의 객체들이 순서를 가지는 모음으로 리스트와 유사한 면이 많다. 차이점은 변경 불가능한 자료형이라는 것이다. 또한, 튜플은 리스트가 가지고 있는 것만큼 다양한 메서드가 없다. 튜플은 시퀀스 자료형이므로 시퀀스형의 인덱싱과 슬라이싱, 연결하기, 반복하기, 길이정보 등의 일반적인 연산을 모두 가진다. 튜플은 ()로 표현한다.

```
>>> t = ()              # 빈튜플
>>> t = (1, 2, 3)       # 괄호 사용
```

사실 괄호()를 사용하지 않아도 쉼표 ,로 데이터가 구분되면 튜플로 처리한다.

```
>>> t = 1, 2, 3
```

괄호()는 수식에서의 ()와 혼동될 가능성이 있기 때문에 주의해야 한다. 예를 들어 r=(1)은 r=1로 해석된다. 따라서 데이터가 하나인 튜플은 괄호의 사용과 관계없이 반드시 r=(1,)와 같이 쉼표,를 포함해야 한다.

```
>>> r = (1,)            # 데이터가 한 개일 때는 반드시 쉼표가 있어야 한다.
>>> r = 1,              # 괄호는 없어도 쉼표는 있어야 한다. 한다.
```

다음은 시퀀스 자료형의 일반적인 연산을 사용한 예이다.

```
>>> t * 2                           # 반복하기
(1, 2, 3, 1, 2, 3)
>>> t + ('pyKUNG','users')          # 연결하기
(1, 2, 3, 'pyKUNG', 'users')
>>> print(t[0], t[1:3])             # 인덱싱과 슬라이싱
1 (2, 3)
>>> len(t)                          # 길이 정보
3
>>> 1 in t                          # 멤버 검사
True
```

튜플은 변경 불가능한 자료형이어서 값을 변경할 수는 없다.

```
>>> t[0] = 100          # 허용이 안되며 에러가 발생한다.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

튜플은 검색에 관련된 메서드 두 개를 갖는다.

```
>>> t = (1, 2, 3, 2, 2, 3)
>>> t.count(2)          # 2가 몇 개 있는가?
3
>>> t.index(2)          # 첫 번째 2의 우치는?
1
>>> t.index(2, 1)       # 1 위치부터 검색해 나간다.
1
```

튜플을 중첩하는 것도 물론 가능하다.

```
>>> t = (12345, 54321, 'hello')
>>> u = t, (1, 2, 3, 4, 5)              # 중첩된 튜플
>>> u
((12345, 54321, 'hello'), (1, 2, 3, 4, 5))
```

튜플을 이용하여 좌우 변에 복수 개의 데이터를 치환할수 있다.

```
>>> x, y, z = 1, 2, 3
>>> (x1, y1), (x2, y2) = (1, 2), (3, 4)
```

튜플을 이용하면 두 개의 값도 쉽게 바꿀수 있다.

```
>>> x, y = 1, 2
>>> x, y = y, x
>>> x, y
(2, 1)
```

6.2 패킹과 언패킹

한 데이터에 여러개의 데이터를 넣는 것을 패킹이라고 한다.

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

패킹과 반대로 한 데이터에서 데이터를 각각 꺼내오는 것을 언패킹이라고 한다.

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

리스트로 언패킹을 지원한다.

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

학장된 언패킹은 좀더 자유로운 형태로 이용할수 있다.

```
>>> T =(1, 2, 3, 4, 5)
>>>
>>> a, *b = T
>>> print(a, b)
1 [2, 3, 4, 5]
>>>
>>> *a, b = T
>>> print(a, b)
[1, 2, 3, 4] 5
>>>
>>> a, b, *c = T
>>> print(a, b, c)
1 2 [3, 4, 5]
```

여기서 * a와 같은 식의 표현은 나머지 전부를 의미한다. 따라서 다음과 같은 표현은 있을수 없다.

```
>>> a, *b, *c = T
  File "<stdin>", line 1
SyntaxError: two starred expressions in assignment
```

6.3 리스트와의 차이점

리스트와의 공통점은 임의의 객체를 저장할수 있다는 것과 시퀀스 자료형이라고는 것이다. 이에 반해 다음과 같은 차이점이 있다.

● 변경 불가능한 시퀀스 자료형이다.
● 함수의 가변 인수를 지원한다.

변경해야 할 데이터들은 리스트에, 변경하지 말아야 할 데이터는 튜플에 저장한다. 리스트와 튜플은 list()와 tuple() 내장 함수를 사용하여 상호 변환할수 있다.

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

튜플은 다음과 같은 경우에 특별히 활용된다. 첫 번째는 함수에 있어서 하나 이상의 값을 반환할때이다.

```
>>> def calc(a, b):
...     return a + b, a* b              # 튜플을 반환한다
...
>>> x, y = calc(5, 4)
```

두 번째는 튜플에 있는 값들을 함수의 인수로 사용할 때이다.

```
>>> args = (4, 5)
>>> calc(*args)         # calc(4, 5) 와 동일하다
(9, 20)
```

세 번째는 파이썬 2 형식의 서식 문자열에 데이터를 공급할 때이다.

```
>>> "%d %f %s" %(12, 3.456, 'hello')
'12 3.456000 hello'
```

6.4 이름있는 튜플

이름있는 튜플은 튜플에 이름으로 접근할 수 있도록 관련 기능을 추가한 것이다. 모듈 collections의 namedtuple() 함수로 객체를 생성한다.

```
namedtuple(typename, field_names, verbose=False, rename=False)
```

인수 field_names는 이름들을 공백으로 구분하는 문자열로 전달한다.

```
>>> from collections import namedtuple
>>> Point = namedtuple('Point', 'x y')
>>> Point
<class '__main__.Point'>
>>> Point.__name__
'Point'
```

예에서 namedtuple() 함수는 Point 클래스를 만든다.

```
>>> pt1 = Point(1.0, 5.0)
>>> pt2 = Point(2.5, 1.5)
>>> pt1
Point(x=1.0, y=5.0)
>>>
>>> from math import sqrt
>>>
>>> length = sqrt((pt1.x - pt2.x) ** 2 + (pt1.y - pt2.y) ** 2)
>>> # 첨자 참조 가능으로
...
>>> length = sqrt((pt1[0] - pt2[0]) ** 2 + (pt1[1] - pt2[1]) ** 2)
```

6.5 예제: 경로명 다루기

투플을 사용하는 예로 os.path 모듈의 경로명을 다루는 함수를 살펴보자.

```
>>> import os
>>> p = os.path.abspath('NEWS.txt')     # 상태 경로를 절대 경로로 반환한다.
>>> p
'C:₩₩Program Files₩₩Python36₩₩NEWS.txt'
>>> os.path.exists(p)                   # 파일 존재 여부를 검사한다.
True
>>> os.path.getsize(p)# 파일 크기를 확인한다.....
379756
>>> os.path.split(p)                    # (head, tail)로 분리한다(디렉터리명, 파일이름).
('C:₩₩Program Files₩₩Python36', 'NEWS.txt')
>>> os.path.join('c:\\work','t.hwp')    # 디렉터리와 파일 이름을 결합한다.합한다.
'c:\\work₩₩t.hwp'
>>> os.path.normpath('c:\\work\\.\\t.hwp')              # 파일 이름을 정규화한다.
'c:\\work\\.\\t.hwp'
>>> os.path.splitext('c:\\work\\t.hwp') )
('c:\\work\\t', '.hwp')
```

6.6 예제 : URL 다루기

urlib.parse 모듈은 Uniform Resource Locator(URL)을 성분별로 분해하거나 결합하는 인터페이스를 제공한다.

urlparse() 함수

urlib.parse 모듈의 urlparse() 함수는 URL을 다음과 같이 분리하여 튜플을 반환한다.

(addressing scheme, network location, path, parameters, query, fragment identifier)

사용한 예를 다음과 같다.

```
>>> from urllib.parse import urlparse
>>> a = 'http://some.where.or.kr:8080/a/b/c.html;param?x=1&y=2#fragment'
>>> r = urlparse(a)
>>> r
ParseResult(scheme='http',    netloc='some.where.or.kr:8080',    path='/a/b/c.html',    params='param',
query='x=1&y=2', fragment='fragment')
```

urlunparse() 함수

urlunparse() 함수는 튜플로 표현된 성분들을 하나의 URL로 역변환한다. 튜플의 구성 순서는 urlparse 함수의 출력과 같다. 사용한 예는 다음과 같다.

```
>>> from urllib.parse import urlparse, urlunparse
>>> a = 'http://some.where.or.kr:8080/a/b/c.html;param?x=1&y=2#fragment'
>>> u = urlparse(a)
>>> urlunparse(u)
'http://some.where.or.kr:8080/a/b/c.html;param?x=1&y=2#fragment'
```

urljoin() 함수

urljoin() 함수는 URL과 상대 URL을 연결하여 절대 URL을 만든다. 다음은 사용한 예이다.

```
>>> from urllib.parse import urljoin
>>> b = 'http://some.where.or.kr:8080/a/b/c.html?x=1'
>>> urljoin(b, 'd.html')
'http://some.where.or.kr:8080/a/b/d.html'
```

시킴(Schmen, 프로토콜명, 예:http)과 네트워크 위치만 얻으려면 다음과 같이 /를 이용한다.

```
>>> urljoin(b, '/')
'http://some.where.or.kr:8080/'
```