튜플형(Tuple Type)

- **불변자료형**(immutable)이다.
  - 생성한 후 내용의 변경이 불가능하다. 즉, 담고 있는 객체를 삭제, 변경, 삽입하는 것이 불가능하다.
- 순서를 가지는 0개 이상의 객체를 참조하는 **시퀀스형**이다.
  - 각 객체는 쉼표(,)로 구분한다.
- 튜플 자신을 포함해 어떠한 자료형도 담을 수 있다.
- 출력 형식
  - 튜플은 항상 **( )**(소괄호) 형태로 출력한다.

튜플 생성

튜플을 만드는 방법으로는 다음 세 가지가 있다.

- 소괄호 **( )** 안에 쉼표(,)로 구분한 객체
- 소괄호 없이 쉼표(,)로 구분한 객체
- **tuple()** 생성자(클래스)

소괄호 **( )**를 사용해서 튜플를 만들어보자.

- 각 객체는 쉼표(,)로 구분한다.

In [ ]:

```
t1 = (5, -3, 3.14, 'red', '드럼')
print(t1)
```

소괄호 없이 튜플를 만들어보자.

- 각 객체는 쉼표(,)로 구분한다.

In [ ]:

```
t2 = 5, -3, 3.14, 'red', '드럼'
print(t2)
```

앞서 만든 튜플의 자료형을 확인해보자.

In [ ]:

```
type(t1)
```

In [ ]:

```
type(t2)
```

tuple() 생성자를 사용해서 튜플을 만들어 보자.

- **tuple()**의 전달인자로 **순회형**만 올 수 있다.
- 순회형(iterable)이란 담고 있는 객체들에 하나씩 순서대로 접근할 수 있는 자료형으로 문자열, 리스트, 튜플, 딕셔너리, 세트가 있다.

In [ ]:

```
string_tuple= tuple('가나다라마바사')
```

In [ ]:

```
print(string_tuple)
```

모든 복합자료형은 어떠한 자료형도 담을 수 있다.

- 따라서 튜플은 튜플 자신을 포함해 어떠한 자료형도 담을 수 있다.

In [ ]:

```
complex_tuple = False, (), t2, ['x', 3, string_tuple], None
```

In [ ]:

```
print(complex_tuple)
```

앞서 만든 튜플의 길이, 즉 각 튜플이 담고 있는 객체의 개수를 확인해보자.

In [ ]:

```
# (5, -3, 3.14, 'red', '드럼')
len(t1)
```

In [ ]:

```
# 5, -3, 3.14, 'red', '드럼'
len(t2)
```

In [ ]:

```
# tuple('가나다라마바사')
len(string_tuple)
```

In [ ]:

```
# False, (), tuple2, ['x', 3, string_tuple], None
len(complex_tuple)
```

튜플은 불변자료형이라 생성한 후 내용의 변경이 불가능하다.

In [ ]:

```
# (5, -3, 3.14, 'red', '드럼')
t1[-1] = '기타'    # 수정 불가능
```

형변환

튜플은 불변자료형이기 때문에 객체를 변경할 필요가 있는 경우에는 **list()** 생성자를 사용하여 튜플을 리스트로 형변환해야 한다.

튜플은 리스트로 형변환이 가능하고 리스트도 튜플로 형변환이 가능하다.

In [ ]:

```
T = 1, 2, 3
```

In [ ]:

```
type(T)
```

In [ ]:

```
print(T)
```

In [ ]:

```
L = list(T)    # 튜플을 리스트로 형변환한다.
```

In [ ]:

```
type(L)
```

In [ ]:

```
print(L)
```

In [ ]:

```
T = tuple(L)  # 리스트를 튜플로 다시 형변환한다.
```

In [ ]:

```
type(T)
```

In [ ]:

```
print(T)
```

튜플 할당

(x, y, z) = (i, j, k)

x, y, z = i, j, k

튜플 할당(tuple assignment)이란?

등호를 기준으로 우변의 값 혹은 표현식 각각을 좌변의 같은 위치에 해당하는 변수로 할당하는 것이다.

- 좌변 : 튜플 변수
- 우변 : 표현식 튜플(문자열과 리스트 같은 시퀀스형 자료도 가능)

원리

- 좌변의 변수에 대응하는 우변의 값을 할당한다.
- 우변의 모든 표현식은 좌변으로 할당하기 전에 평가(evaluation) 또는 계산을 해야 한다.
- 좌변의 변수 개수와 우변의 표현식 값의 개수는 반드시 일치해야 한다.

이점

- 튜플 할당은 매우 편리하고 유용하게 사용할 수 있다.
- 예를 들어, 한 줄에서 여러 개의 변수를 한꺼번에 할당하는 것이 가능해져 코드가 간소해진다.
- 두 변수 간의 값을 서로 바꿀 때(swap)에도 유용하다.

In [ ]:

```
i, j, k = '드럼', '기타', '피아노'
print(i, j, k)
```

In [ ]:

```
# --- i, j, k = '드럼', '기타', '피아노'
# 스왑(swap operation)
i, j = j, i
print(i, j, k)
```

튜플 할당 기능이 없었다면?

In [ ]:

```
# i, j, k = '드럼', '기타', '피아노' 대신 ...
i = '드럼'
j = '기타'
k = '피아노'
```

In [ ]:

```
# --- i, j, k = '드럼', '기타', '피아노'
# i, j = j, i 대신...
tmp = j
j = i
i = tmp
print(i, j, k)
```

In [ ]:

```
email = 'abc@xyz.kr'
user_name, domain = email.split('@')
```

In [ ]:

```
print(user_name)
print(domain)
```

In [ ]:

```
x, y = 1, 2, 3
```

좌변의 변수 개수와 우변의 객체 개수가 같지 않으면 오류가 발생한다.

이런 경우 우변의 불필요한 객체 값을 좌변에서 더미 변수(dummy variable)인 밑줄(_)로 받아 처리하면 된다.

In [ ]:

```
# 2와 4가 불필요한 경우
x, _, y, _ = 1, 2, 3, 4
```

In [ ]:

```
print(x, y)
```

튜플 패킹과 언패킹

튜플 패킹

- 우변의 객체들을 좌변에 있는 변수 하나에 할당하는 것이다.

In [ ]:

```
# 튜플 패킹
t = 3.14, [5, -3], 'green', ('드럼', '기타')
```

In [ ]:

```
type(t)   # t의 자료형은 튜플이다.
print(t)
```

다음 코드를 보면 알 수 있듯이, 튜플을 생성할 때 우리가 이미 했던 일이다.

In [ ]:

```
t1 = (5, -3, 3.14, 'red', '드럼')
t2 = 5, -3, 3.14, 'red', '드럼'
```

In [ ]:

```
print(t1)
print(t2)
```

튜플 언패킹

- 우변의 패킹한 튜플 변수에서 여러 개의 값을 좌변으로 꺼내오는 것이다.
- 즉, 튜플 안의 객체들을 여러 개의 변수에 한 번에 할당하는 것이라 할 수 있다.
- 이때 좌변의 변수 개수와 우변의 튜플 길이가 같아야 한다.

다음과 같은 형식으로 사용한다.

```
(x, y, z) = t

x, y, z = t
```

In [ ]:

```
# --- 튜플 패킹 : t = 3.14, [5, -3], 'green', ('드럼', '기타')
# 튜플 언패킹
print(t)
a, b, c, d = t
print(a)
print(b)
print(c)
print(d)
```

시퀀스형 패킹/언패킹 연산자 *

모든 시퀀스 자료형(리스트, 튜플 등)은 *를 사용하여 담고 있는 객체들을 패킹(packing) 또는 언패킹(unpacking) 할 수 있다.

여기에서는 시퀀스형 패킹만 살펴보고 *를 시퀀스형 언패킹 연산자로 사용하는 경우는 뒤(함수)에서 다루기로 한다.

시퀀스형 패킹

좌변의 변수 중 어느 한 변수 앞에 * 부호를 붙여 다음과 같은 형식으로 사용한다.

```
x, ..., *y = i, j, k,...
```

다음과 같은 특징이 있다.

- 좌변의 변수는 반드시 두 개 이상이어야 한다.
- ***y**는 좌변의 맨 앞이나 맨 뒤, 중간 어디에 와도 상관없다.
- 우변의 객체를 좌변의 변수에 할당할 때
  - 우변의 객체 개수가 좌변의 변수 개수보다 많으면
  - 우변의 객체를 좌변의 변수에 **순서대로 할당**한다.
- 우변의 남은 객체는 모두 변수 ***y**에 리스트(**list**)로 할당한다.
- 남은 객체는 좌변의 ***y**의 위치에 따라 달라진다.
- 패킹한 ***y***의 자료형은 리스트(**list**)다.

In [ ]:

```
x, y, *z = 3.14, [5, -3], 'green', ('드럼', '기타')
```

In [ ]:

```
print(x)
```

In [ ]:

```
print(y)
```

In [ ]:

```
print(z)  # 리스트다.
```

In [ ]:

```
*x, y, z = 3.14, [5, -3], 'green', ('드럼', '기타')
```

In [ ]:

```
print(z)
```

In [ ]:

```
print(y)
```

In [ ]:

```
print(x)  # 리스트다.
```

In [ ]:

```
x, *y, z = 3.14, [5, -3], 'green', ('드럼', '기타')
```

In [ ]:

```
print(x)
```

In [ ]:

```
print(z)
```

In [ ]:

```
print(y)  # 리스트다.
```