10. 튜플 자료형

1. 튜플 만들기
2. 튜플 인덱싱(indexing), 슬라이싱(slicing)
3. 튜플은 immutable 객체입니다
4. 튜플에 +, *, in, not in, del 연산자 사용하기
5. 튜플에 함수 적용하기 - len(), max(), min(), sum(), sorted(), reversed()
6. 튜플 메소드
7. 튜플의 이용
8. 정리

1. 튜플 만들기

- 튜플은 ()를 이용하여 데이터를 모아서 관리함
- 튜플은 immutable한 객체임
- immutable하기 때문에 객체 수정과 관련된 메소드들은 없음

튜플	리스트
()	[ ]
immutable	mutable
메소드 2개 count(), index()	메소드 11개

1. 튜플 만들기

- 튜플은 () 안에 콤마를 이용하여 데이터를 모아서 관리함
- 괄호없이 콤마로 데이터들을 분리해도 튜플로 간주함

```
>>> T1 = (3, 2, 7, 1)
>>> type(T1)
<class 'tuple'>
>>> T2 = 5, 3, 7     # 괄호가 없이 콤마로 여러 데이터를 적는 경우도 튜플입니다.
>>> type(T2)
<class 'tuple'>
>>> T3 = 6, 8, 9,    # 맨 마지막에 콤마가 있기도 합니다.
>>> type(T3)
<class 'tuple'>
```

```
>>> a = 3; b = 4; c = 5
>>> a, b, c       # 튜플로 출력하게 되니까 괄호로 묶어서 출력합니다.
(3, 4, 5)
```

1. 튜플 만들기

- 원소가 1개인 튜플 생성하기 (괄호에 데이터가 1개 있으면 괄호 무시함, 튜플 아님)

```
>>> T1 = (5)              # 괄호 안에 정수만 하나 있으면 튜플이 아니라 정수입니다.
>>> type(T1)
<class 'int'>
>>> T2 = ('apple')        # 괄호 안에 문자열만 하나 있으면 튜플이 아니라 문자열입니다.
>>> type(T2)
<class 'str'>
>>> len(T2)               # 'apple'의 길이가 나와요.
5
>>> T3 = ([1,2,3])        # 괄호 안에 리스트만 하나 있으면 튜플이 아니라 리스트입니다.
>>> type(T3)
<class 'list'>
>>> len(T3)               # 리스트 [1,2,3]의 길이가 나와요.
3
>>> T4 = ((1,3,5,7,9)) # 괄호 안에 튜플이 하나 있으면 역시 그냥 튜플입니다.
>>> type(T4)
<class 'tuple'>
>>> len(T4)               # 튜플 (1,3,5,7,9)의 길이가 나와요.
5
```

1. 튜플 만들기

- 원소가 1개인 튜플을 만들려면, 반드시 콤마를 넣어야 한다

```
〉〉〉 S1 = (5,)            # S1 = 5, 라고 해도 똑같습니다.
〉〉〉 type(S1)
〈class 'tuple'〉
〉〉〉 len(S1)
1
〉〉〉 S2 = ('hello',)      # S2 = ‘hello’, 라고 해도 똑같습니다.
〉〉〉 type(S2)
〈class 'tuple'〉
〉〉〉 len(S2)
1
〉〉〉 S3 = ([1,2,3],)      # S3 = [1,2,3], 라고 해도 똑같습니다.
〉〉〉 type(S3)
〈class 'tuple'〉
〉〉〉 len(S3)
1
```

1. 튜플 만들기

- 빈 튜플 만들기

빈 괄호 이용하기	tuple() 함수 이용하기

>>> T = ()
>>> type(T)
<class 'tuple'>
	>>> T = tuple()
	>>> type(T)
	<class 'tuple'>

- 다른 자료형을 튜플로 변환하기 (tuple(iterable) 함수 사용)

튜플 ← 문자열	〉〉〉 book = 'Harry Potter' 〉〉〉 T1 = tuple(book) 〉〉〉 T1 ('H', 'a', 'r', 'r', 'y', ' ', 'P', 'o', 't', 't', 'e', 'r')
튜플 ← 리스트	〉〉〉 L = [1, 3, 5, 7] 〉〉〉 T2 = tuple(L) 〉〉〉 T2 (1, 3, 5, 7)

1. 튜플 만들기

튜플 ← 집합	>>> S = {'red', 'blue', 'white'} # 집합은 순서 개념이 없어요.
	>>> T3 = tuple(S)
	>>> T3
	('white', 'blue', 'red')

튜플 ← 사전	>>> area_code = {'서울':'02', '경기':'031', '인천':'032'}
	>>> T4 = tuple(area_code) # 키만 튜플에 저장합니다.
	>>> T4
	('서울', '경기', '인천')

튜플 ← range()	>>> T5 = tuple(range(10, 60, 10))
	>>> T5
	(10, 20, 30, 40, 50)

튜플 ← reversed()	>>> T6 = tuple(reversed([13, 25, 11, 12]))
	>>> T6
	(12, 11, 25, 13)

2. 튜플 인덱싱(indexing), 슬라이싱(slicing)

- 인덱싱, 슬라이싱은 리스트와 똑같음

```
>>> primes = (2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
>>> print(primes[5])
13
>>> print(primes[-1])          # -1은 맨 마지막 자리입니다.
29
>>> print(primes[3:7])         # 인덱스 3에서 인덱스 6까지입니다.
(7, 11, 13, 17)
>>> print(primes[8:2:-3])
(23, 13)
```

3. 튜플은 immutable 객체입니다

- 튜플 객체는 만든 후에 내용을 수정할 수 없음

```
〉〉〉 T = (1, 3, 6, 7)
〉〉〉 T[2] = 5                # 인덱스 2에 있는 6을 5로 바꾸려고 하는데 에러가 발생해요.
... ...
TypeError: 'tuple' object does not support item assignment
〉〉〉 del T[2]                # T[2]에 있는 6을 삭제하려고 하는데 에러가 발생해요.
... ...
TypeError: 'tuple' object doesn't support item deletion
```

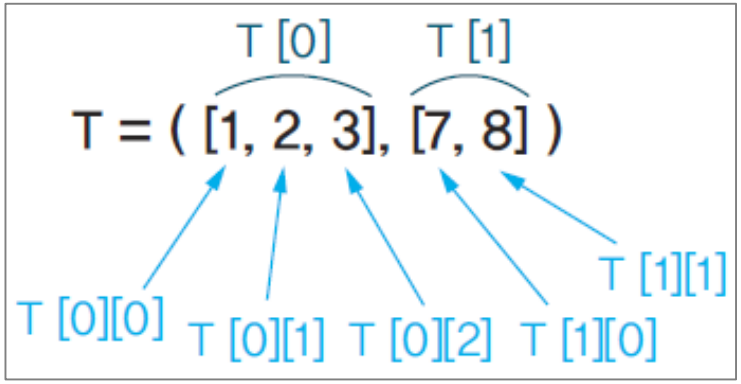
- del 연산자를 이용해서 튜플을 통째로 삭제하는 것은 가능함

```
〉〉〉 T = ('apple', 'banana', 'grape', 'orange')
〉〉〉 del T
〉〉〉 T
... ...
NameError: name 'T' is not defined
```

3. 튜플은 immutable 객체입니다

- 튜플 안에는 9가지 자료형을 모두 저장할 수 있음
- 튜플에 mutable 객체인 리스트, 집합, 사전도 저장 가능함

```
〉〉〉 T = ([1,2,3], [7,8])
〉〉〉 T[0] [2] = 100      # T[0]는 리스트
〉〉〉 T
([1, 2, 100], [7, 8])
〉〉〉 T[1] [0] = 200      # T[1]은 리스트
〉〉〉 T
([1, 2, 100], [200, 8])
```



3. 튜플은 immutable 객체입니다

```
>>> T = ([1,2,3], (4,5), 'hello')
>>> T[0] = [10, 20, 30]
... ...
T[0] = [10, 20, 30]
TypeError: 'tuple' object does not support item assignment
>>> T[0][0] = 10                    # 튜플 안에 저장된 리스트는 수정이 가능합니다.
>>> T[0][1] = 20
>>> T[0][2] = 30
>>> T
([10, 20, 30], (4, 5), 'hello')
>>> T[1][0] = 40                    # 튜플 안에 저장된 튜플은 수정할 수 없습니다.
... ...
T[1][0] = 40
TypeError: 'tuple' object does not support item assignment
>>> T[2][0] = 'H'                   # 튜플 안에 저장된 문자열도 수정할 수 없습니다.
... ...
T[2][0] = 'H'
TypeError: 'str' object does not support item assignment
```

4. 튜플에 +, *, in, not in, del 연산자 사용하기

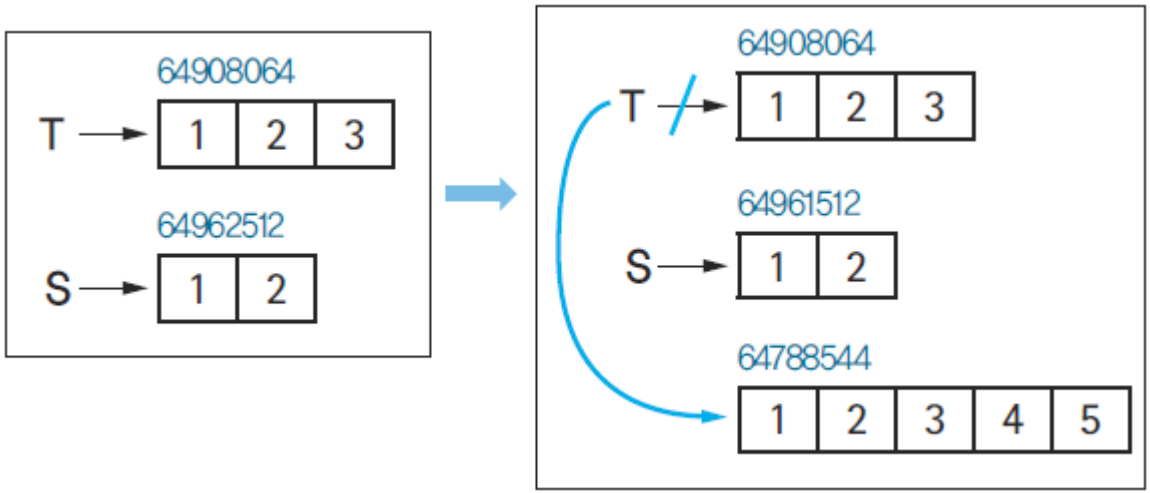
- 문자열, 리스트에서 사용하던 것과 같음

```
〉〉〉 T = (7, 8, 2, 5, 4)
〉〉〉 S = (6,9)
〉〉〉 T + S                    # 튜플 T와 S를 연결한 새로운 튜플을 만듭니다.
(7, 8, 2, 5, 4, 6, 9)
〉〉〉 S * 5                    # 튜플 S를 5번 반복해서 만든 새로운 튜플을 만듭니다.
(6, 9, 6, 9, 6, 9, 6, 9, 6, 9)
〉〉〉 4 in T
True
〉〉〉 10 not in S
True
```

4. 튜플에 +, *, in, not in, del 연산자 사용하기

- +=, *= 도 사용 가능함

```
〉〉〉 T = (1, 2, 3)
〉〉〉 S = (4, 5)
〉〉〉 id(T), id(S)
(64908064, 64962512)
〉〉〉 T += S
〉〉〉 id(T), id(S)
(64788544, 64962512)
〉〉〉 T
(1, 2, 3, 4, 5)
〉〉〉 S
(4, 5)
〉〉〉 A = (7, 9)
〉〉〉 A *= 3
〉〉〉 A
(7, 9, 7, 9, 7, 9)
```



5. 튜플에 함수 적용하기

- len(), max(), min(), sum(), sorted(), reversed()

```
>>> primes = (5, 13, 2, 7, 9)
>>> len(primes), max(primes), min(primes), sum(primes)
(5, 13, 2, 36)
>>> sorted(primes)                    # sorted( ) 함수의 결과는 리스트입니다.
[2, 5, 7, 9, 13]
>>> reversed(primes)
<reversed object at 0x01660B90>
>>> tuple(reversed(primes))      # reversed( ) 함수 결과를 tuple로 변환합니다.
(9, 7, 2, 13, 5)
>>> list(reversed(primes))        # reversed( ) 함수 결과를 list로 변환합니다.
[9, 7, 2, 13, 5]
```

6. 튜플 메소드

- 튜플은 immutable하기 때문에 객체, 변수, 변수명

```
〉〉〉 dir(tuple)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__',
'__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'count',
'index']
```

메소드	설명	반환값	발생 가능 에러
count(x)	데이터 x의 개수를 반환합니다.	있음	없음
index(x)	데이터 x의 인덱스를 반환합니다. x가 없는 데이터이면, ValueError가 발생합니다.	있음	ValueError

6. 튜플 메소드

- 튜플에 있는 데이터 개수 세기 - count(x)
  - 반드시 1개의 데이터 x를 인수로 넣어야 하고, x가 튜플에 없으면 0을 반환함.

```
>>> T = (3, 7, 9, 3, 2, 7, 3, 7)
>>> T.count(7)              # 튜플 T에는 7이 3개 있습니다.
3
>>> T.count(5)              # 튜플 T에는 5가 없습니다.
0
```

6. 튜플 메소드

- 튜플에 있는 데이터 위치 찾기 – index()

〉〉〉 T = (3, 7, 9, 3, 2, 7, 3, 7)	

인수 1개인 경우	>>> T.index(2)
	4
	>>> T.index(7) # 데이터가 여러 개인 경우 첫 번째 인덱스 반환
	1
	>>> T.index(5) # 없는 데이터에 대해서는 ValueError 발생
	……
	ValueError: tuple.index(x): x not in tuple

인수 2개인 경우	>>> T.index(7, 5) # T[5:]에서 처음으로 7이 있는 인덱스 반환
	5
	>>> T.index(3, 2) # T[2:]에서 처음으로 3이 있는 인덱스 반환
	3
	>>> T.index(9, 5) # T[5:]에서 9가 없으므로 ValueError 발생
	……
	ValueError: tuple.index(x): x not in tuple

인수 3개인 경우	>>> T.index(7, 3, 6) # T[3:6]에서 7의 인덱스를 반환
	5
	>>> T.index(2, -6, -1) # T[-6:-1]에서 2의 인덱스를 반환
	4

6. 튜플 메소드

CODE 59 튜플의 count 메소드와 똑같이 동작하는 코드

```
T = (89, 90, 85, 99, 77, 58, 85, 77)
score = int(input('Enter score : '))
cnt = 0                  # cnt는 개수를 세기 위한 변수입니다.
for x in T:              # 튜플 T에서 원소를 하나씩 차례대로 가져와서 루프를 수행합니다.
    if x == score:       # x와 score가 같으면 cnt 값을 1 증가시킵니다.
        cnt += 1
print("{} - There are {}.".format(score, cnt))
```

[결과 1]	[결과 2]	[결과 3]
Enter score : 85 85 - There are 2.	Enter score : 90 90 - There are 1.	Enter score : 100 100 - There are 0.

7. 튜플의 이용

- print() 함수에서 튜플 사용하였음

```
>>> a = 10; b = 20
>>> print('a = %d b = %d' % (a,b))     # (a,b)가 튜플임
a = 10 b = 20
```

- 간단히 swap하기
  - swap은 두 변수의 값을 바꾸는 일을 의미함.
  - 리스트를 이용해서도 swap을 할 수 있음.

```
>>> a = 10; b = 20
>>> b, a = a, b              # (b, a) = (a, b)와 같은 표현입니다.
>>> print(a, b)              # 출력해 보면 a의 값과 b의 값이 바뀌어 있어요.
20 10
```

7. 튜플의 이용

- IDLE에서 간단히 변수 출력하기

```
〉〉〉 a = 10; b = 20; c = 30
〉〉〉 a, b, c          # 괄호가 없더라도 데이터가 콤마로 분리되어 나열되면 튜플로 인식함.
(10, 20, 30)
```

8. 정리

◆ 리스트와 튜플 비교

	리스트	튜플
기호	[]	()
저장되는 데이터	아홉 가지 자료형 모두 저장 가능	아홉 가지 자료형 모두 저장 가능
변경 가능성	변경 가능 (mutable 자료형)	변경 불가능 (immutable 자료형)
메소드	append(), insert(), clear() 등 11개의 메소드가 있음.	count(), index() 두 개만 있음.