



데이터 처리 프로그래밍

Data Processing Programming

04

리스트, 튜플, 세트, 딕셔너리

목차

1. 리스트 list
2. 튜플 tuple
3. 세트 set
4. 딕셔너리 dictionary



1. 리스트

- 변경 가능한 시퀀스 자료형
- 하나의 변수에 여러 값 할당 가능
- 같은 자료형이 아니라 다른 자료형을 가지는 값들을 포함 가능
- 일반적으로 비슷한 항목들의 모음을 순서대로 저장하는데 사용

['one', 'two', 'three']

'one'	'two'	'three'
-------	-------	---------

```
>>> []  
[]  
>>> [1, 2, 3]  
[1, 2, 3]  
>>> ['One', 'Two', 'Three']  
['One', 'Two', 'Three']  
>>> [1, 'One', 2, 'Two', 3, 'Three']  
[1, 'One', 2, 'Two', 3, 'Three']  
>>> [1, 2, 3, ['One', 'Two', 'Three']]  
[1, 2, 3, ['One', 'Two', 'Three']]
```

- 리스트의 각 위치에 해당하는 값에 접근하기 위해서 주소 개념의 숫자를 사용

'one'	'two'	'three'
[0]	[1]	[2]
[-3]	[-2]	[-1]

```
>>> list = ['One', 'Two', 'Three']
>>> list
['One', 'Two', 'Three']
>>> list[0]
'One'
>>> list[1]
'Two'
>>> list[2]
'Three'
>>> list[-1]
'Three'
>>> list[-2]
'Two'
>>> list[-3]
'One'
```

- 리스트 안에 리스트가 있을 경우, 인덱스 접근 방법

```
>>> list = [1, 2, 3, ['One', 'Two', 'Three']]
>>> list
[1, 2, 3, ['One', 'Two', 'Three']]
>>> list[3]
['One', 'Two', 'Three']
>>> list[3][0]
'One'
>>> list[3][1]
'Two'
>>> list[3][2]
'Three'
```

1	2	3	'one'	'two'	'three'
[0]	[1]	[2]	[3][0]	[3][1]	[3][2]

- 리스트의 인덱스를 통해서 부분만 가져올 때 사용

```
>>> list = ['One', 'Two', 'Three']
>>> list[0:]
['One', 'Two', 'Three']
>>> list[:]
['One', 'Two', 'Three']
>>> list = ['One', 'Two', 'Three']
>>> list[1:2]
['Two']
>>> list[1:]
['Two', 'Three']
```

- 중첩된 리스트에서 일부분만 인덱스를 통해서 가져올 때 사용

```
>>> list = [1, 2, 3, ['One', 'Two', 'Three']]
>>> list[2:4]
[3, ['One', 'Two', 'Three']]
>>> list[3][2:]
['Three']
>>> list[3][:2]
['One', 'Two']
```

- 더하기 + 연산
- 곱하기 * 연산
- 리스트 길이 len() 함수

```
>>> list_1 = ['One', 'Two', 'Three']
>>> list_2 = ['Four', 'Five', 'Six']
>>> list_1 + list_2
['One', 'Two', 'Three', 'Four', 'Five', 'Six']
>>> list_1 * 3
['One', 'Two', 'Three', 'One', 'Two', 'Three', 'One', 'Two', 'Three']
>>> len(list_1)
3
>>> len(list_1 + list_2)
6
```

- 리스트 인덱스 주소를 이용한 값 수정

```
>>> list = ['One', 'Two', 'Three']
>>> list[2] = 3
>>> list
['One', 'Two', 3]
>>> list[1] = 2
>>> list
['One', 2, 3]
>>> list[0] = 1
>>> list
[1, 2, 3]
```

- 리스트 요소 추가 함수 `append()`

```
>>> list = ['One', 'Two', 'Three']
>>> list.append('Four')
>>> list
['One', 'Two', 'Three', 'Four']
>>> list.append([1, 2, 3, 4])
>>> list
['One', 'Two', 'Three', 'Four', [1, 2, 3, 4]]
```

- 리스트 정렬 함수 `sort()`

```
>>> list = [10, 40, 20, 30]
>>> list.sort()
>>> list
[10, 20, 30, 40]
>>> list = ['orange', 'apple', 'banana', 'strawberry']
>>> list.sort()
>>> list
['apple', 'banana', 'orange', 'strawberry']
```

- 리스트 요소 반전 함수 reverse()

```
>>> list = ['orange', 'apple', 'banana', 'strawberry']
>>> list.reverse()
>>> list
['strawberry', 'banana', 'apple', 'orange']
```

- 리스트의 요소 값에 대한 인덱스를 반환하는 함수 index()

```
>>> list = [10, 40, 20, 30]
>>> list.index(10)
0
>>> list.index(20)
2
```

- 리스트 요소 삽입 함수 insert()

```
>>> list = [10, 40, 20, 30]
>>> list.insert(4, 50)
>>> list
[10, 40, 20, 30, 50]
>>> list.insert(0, 60)
>>> list
[60, 10, 40, 20, 30, 50]
```

- 리스트 요소 제거 함수 remove()

```
>>> list = [10, 40, 20, 30]
>>> list.remove(40)
>>> list
[10, 20, 30]
>>> list.remove(30)
>>> list
[10, 20]
```

- 리스트 요소 제거 함수 del

```
>>> list = [10, 40, 20, 30]
>>> del list[0]
>>> del list[2]
>>> list
[40, 20]
```

- 리스트 요소를 꺼내는 함수 pop()

```
>>> list = [10, 40, 20, 30]
>>> list.pop()
30
>>> list
[10, 40, 20]
>>> list.pop(0)
10
>>> list
[40, 20]
```

- 리스트 요소의 개수 세는 함수 count()

```
>>> list = [10, 20, 20, 30, 30, 30]
>>> list.count(30)
3
>>> list.count(20)
2
```

- 리스트 확장 함수 extend()

```
>>> list = [10, 40, 20, 30]
>>> list.extend([50, 60])
>>> list
[10, 40, 20, 30, 50, 60]
```



2. 튜플

- 리스트와 유사하지만 변경 불가능한 시퀀스 자료형
- 하나의 변수에 여러 값 할당 가능
- ‘(와)’를 사용하여 표현

```
>>> ()  
()  
>>> (1, 2, 3)  
(1, 2, 3)  
>>> ('One', 'Two', 'Three')  
('One', 'Two', 'Three')  
>>> (1, 'One', 2, 'Two', 3, 'Three')  
(1, 'One', 2, 'Two', 3, 'Three')  
>>> (1, 2, 3, ('One', 'Two', 'Three'))  
(1, 2, 3, ('One', 'Two', 'Three'))
```

- 튜플의 각 위치에 해당하는 값에 접근하기 위해서 주소 개념의 숫자를 사용
- 튜플 안에 튜플이 중첩되어 있을 경우, 인덱스 접근 방법

```
>>> tuple = ('One', 'Two', 'Three')
>>> tuple
('One', 'Two', 'Three')
>>> tuple[0]
'One'
>>> tuple[-1]
'Three'
>>> tuple = (1, 2, 3, ('One', 'Two', 'Three'))
>>> tuple
(1, 2, 3, ('One', 'Two', 'Three'))
>>> tuple[3]
('One', 'Two', 'Three')
>>> tuple[3][1]
'Two'
```

- 튜플의 인덱스를 통해서 부분만 가져올 때 사용
- 중첩된 튜플에서 일부분만 인덱스를 통해서 가져올 때 사용

```
>>> tuple = ('One', 'Two', 'Three')
>>> tuple[0:]
('One', 'Two', 'Three')
>>> tuple[:]
('One', 'Two', 'Three')
>>> tuple[1:2]
('Two',)
>>> tuple[1:]
('Two', 'Three')
>>> tuple = (1, 2, 3, ('One', 'Two', 'Three'))
>>> tuple[2:4]
(3, ('One', 'Two', 'Three'))
>>> tuple[3][2:]
('Three',)
>>> tuple[3][:2]
('One', 'Two')
```

- 더하기 + 연산
- 곱하기 * 연산
- 리스트 길이 len() 함수

```
>>> tuple_1 = ('One', 'Two', 'Three')
>>> tuple_2 = ('Four', 'Five', 'Six')
>>> tuple_1 + tuple_2
('One', 'Two', 'Three', 'Four', 'Five', 'Six')
>>> tuple_1 * 3
('One', 'Two', 'Three', 'One', 'Two', 'Three', 'One', 'Two', 'Three')
>>> len(tuple_1)
3
>>> len(tuple_1 + tuple_2)
6
```



3. 세트

- 데이터 중복을 허용하지 않는 구조
- 순서가 없는 데이터 집합을 위한 구조
- 인덱싱으로 값을 접근할 수 없음

```
>>> {}  
{}  
>>> {'One', 'Two', 'Three'}  
{'Three', 'One', 'Two'}  
>>> {10, 20, 30, 40}  
{40, 10, 20, 30}
```

- 교집합: `&`, `intersection()`
- 합집합: `|`, `union()`
- 차집합: `-`, `difference()`
- 여집합: `^`, `symmetric_difference()`

```
>>> set_1 = {10, 20, 20, 30}
>>> set_2 = {30, 30, 40, 50}
>>> set_1 & set_2
{30}
>>> set_1.intersection(set_2)
{30}
>>> set_1 | set_2
{50, 20, 40, 10, 30}
>>> set_1.union(set_2)
{50, 20, 40, 10, 30}
>>> set_1 - set_2
{10, 20}
>>> set_1.difference(set_2)
{10, 20}
>>> set_1 ^ set_2
{40, 10, 50, 20}
>>> set_1.symmetric_difference(set_2)
{40, 10, 50, 20}
```

- 요소를 추가하는 add() 함수
- 여러 요소를 추가하는 update() 함수
- 요소를 제거하는 remove() 함수

```
>>> set = {10, 20, 30, 40}
>>> set.add(50)
>>> set
{40, 10, 50, 20, 30}
>>> set.update([60, 70, 80])
>>> set
{70, 40, 10, 80, 50, 20, 60, 30}
>>> set.remove(80)
>>> set.remove(70)
>>> set.remove(60)
>>> set
{40, 10, 50, 20, 30}
```

- 요소를 제거하는 `discard()` 함수
- 모든 요소를 제거하는 `clear()` 함수

```
>>> set = {10, 20, 30, 40}
>>> set
{40, 10, 20, 30}
>>> set.discard(30)
>>> set
{40, 10, 20}
>>> set.clear()
>>> set
set()
```



4. 딥셔너리

- 키^{key}와 값^{value}의 쌍으로 구성된 데이터
- 순서가 없는 데이터
- 키를 통해 값을 얻음
- 동일한 키가 있을경우 덮어씀

```
>>> dic = {1:'One', 2:'Two', 3:'Three'}  
>>> dic  
{1: 'One', 2: 'Two', 3: 'Three'}
```

```
>>> dic = {1:'One', 2:'Two', 3:'Three'}  
>>> dic  
{1: 'One', 2: 'Two', 3: 'Three'}  
>>> dic = {1:'One', 2:'Two', 3:'Three', 1:'One1', 2:'Two2', 3:'Three3'}  
>>> dic  
{1: 'One1', 2: 'Two2', 3: 'Three3'}
```

- 딕셔너리의 해당 키 값에 값을 추가하여 요소 추가
- del 함수를 이용하여 요소 제거

```
>>> dic = {1: 'One', 2: 'Two', 3: 'Three'}
>>> dic
{1: 'One', 2: 'Two', 3: 'Three'}
>>> dic[2]
'Two'
>>> dic[4] = 'Four'
>>> dic
{1: 'One', 2: 'Two', 3: 'Three', 4: 'Four'}
>>> dic[5] = 'Five'
>>> dic
{1: 'One', 2: 'Two', 3: 'Three', 4: 'Four', 5: 'Five'}
>>> del dic[4]
>>> dic
{1: 'One', 2: 'Two', 3: 'Three', 5: 'Five'}
```

- 딕셔너리의 키만 가져오는 keys() 함수
- 딕셔너리의 값만 가져오는 values() 함수
- 딕셔너리의 키와 값을 모두 가져오는 items() 함수
- 딕셔너리에서 키에 해당하는 값을 가져오는 get() 함수
- 딕셔너리에서 키에 해당하는 값을 빼오는 pop() 함수
- 딕셔너리의 모든 요소를 제거하는 clear() 함수

```
>>> dic = {1: 'One', 2: 'Two', 3: 'Three'}
>>> dic.keys()
dict_keys([1, 2, 3])
>>> dic.values()
dict_values(['One', 'Two', 'Three'])
>>> dic.items()
dict_items([(1, 'One'), (2, 'Two'), (3, 'Three')])
>>> dic.get(2)
'Two'
>>> dic.pop(3)
'Three'
>>> dic
{1: 'One', 2: 'Two'}
>>> dic.clear()
>>> dic
{}
```

Q & A