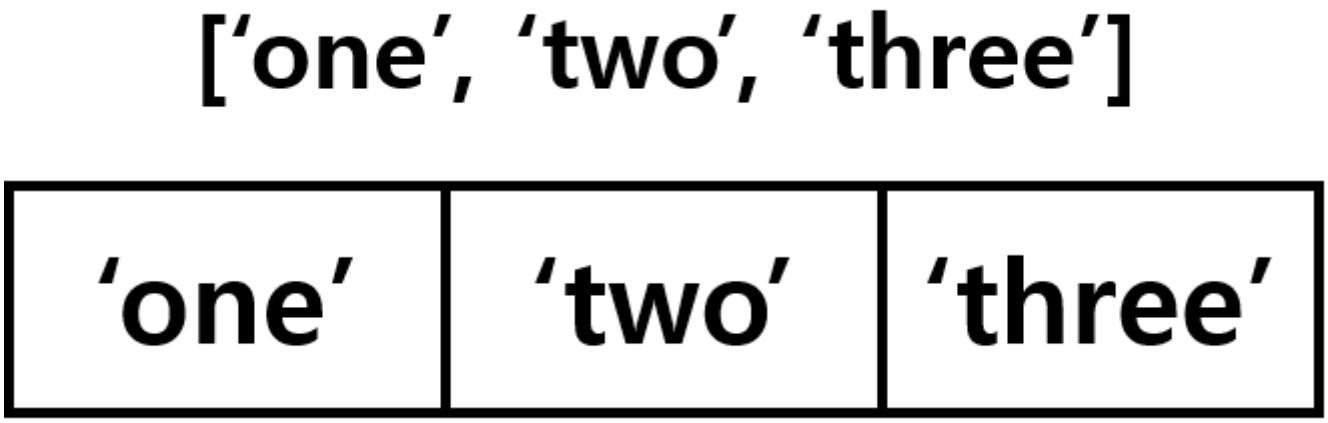
리스트, 튜플, 세트, 딕셔너리

리스트(List)

- 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

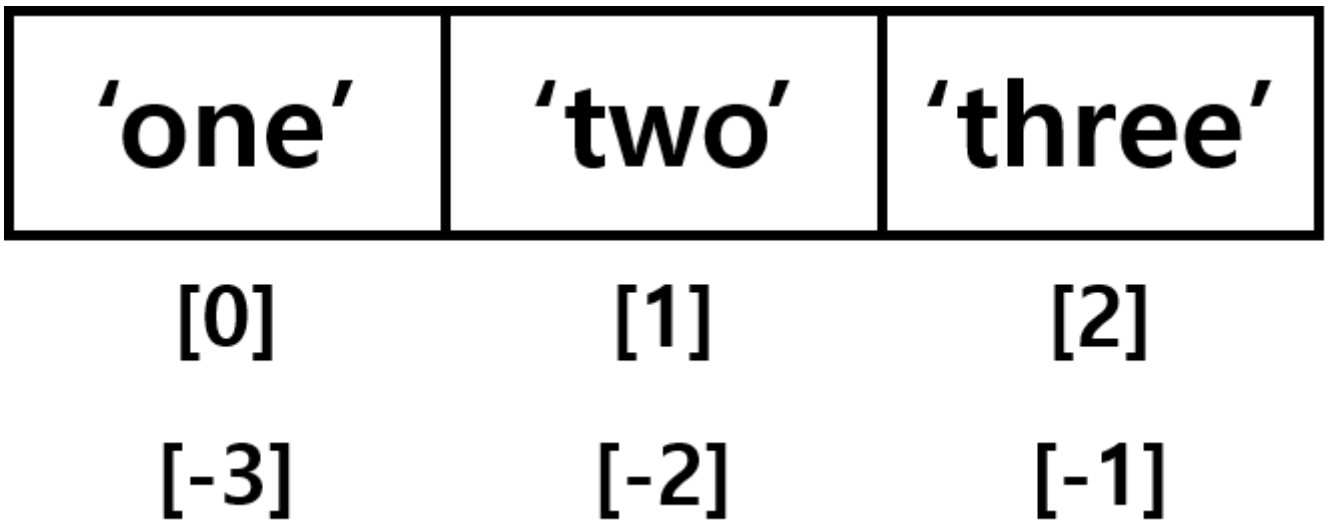



```
print([])
print([1, 2, 3])
print(['One', 'Two', 'Three'])
print([1, 'One', 2, 'Two', 3, 'Three'])
print([1, 2, 3, ['One', 'Two', 'Three']])
```

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

리스트 인덱싱(List Indexing)

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

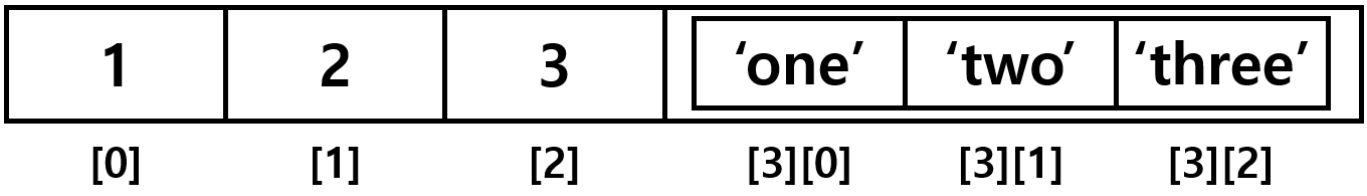



```
list = ['One', 'Two', 'Three']
print(list)
print(list[0])
print(list[1])
print(list[2])
print(list[-1])
print(list[-2])
print(list[-3])
```

```
['One', 'Two', 'Three']
One
Two
Three
Three
Two
One
```

중첩 리스트 인덱싱(Nested List Indexing)

- 리스트 안에 리스트가 있을 경우 인덱스 접근 방법




```
list = [1, 2, 3, ['One', 'Two', 'Three']]
print(list)
print(list[3])
print(list[3][0])
print(list[3][1])
print(list[3][2])
```

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

```
One
Two
Three
```

리스트 슬라이싱(List Slicing)

- 리스트의 인덱스를 통해서 부분만 가져올 때 사용

```
list = ['One', 'Two', 'Three']
print(list)
print(list[0:])
print(list[1:2])
print(list[1:])
```

```
['One', 'Two', 'Three']
['One', 'Two', 'Three']
['Two']
['Two', 'Three']
```

중첩 리스트 슬라이싱(Nested List Slicing)

- 중첩된 리스트에서 일부분만 인덱스를 통해서 가져올 때 사용

```
list = [1, 2, 3, ['One', 'Two', 'Three']]
print(list)
print(list[2:4])
print(list[3][2:])
print(list[3][:2])
```

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

리스트 연산자/함수(List Operators/Function)

- 더하기 + 연산자
- 곱하기 * 연산자
- 리스트 길이를 구하는 `len()` 함수

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

```
['One', 'Two', 'Three']
```

```
['Four', 'Five', 'Six']
['One', 'Two', 'Three', 'Four', 'Five', 'Six']
['One', 'Two', 'Three', 'One', 'Two', 'Three', 'One', 'Two', 'Three']
3
9
```

리스트 수정(List Modify)

- 리스트 인덱스 주소를 이용한 값 수정

```
list = ['One', 'Two', 'Three']
print(list)
list[2] = 3
print(list)
list[1] = 2
print(list)
list[0] = 1
print(list)
```

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

리스트 메소드(List Methods)

append()

- 리스트 요소 추가

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

```
['One', 'Two', 'Three', 'Four']
['One', 'Two', 'Three', 'Four', [1, 2, 3, 4]]
```

sort()

- 리스트 정렬

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

```
print(list)
```

```
[10, 40, 20, 30]
[10, 20, 30, 40]
['orange', 'apple', 'banana', 'strawberry']
['apple', 'banana', 'orange', 'strawberry']
```

reverse()

- 리스트 요소 반전

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

```
['orange', 'apple', 'banana', 'strawberry']
['strawberry', 'banana', 'apple', 'orange']
```

index()

- 리스트의 요소 값에 대한 인덱스를 반환

```
list = [10, 40, 20, 30]
print(list)
print(list.index(10))
print(list.index(20))
```

```
[10, 40, 20, 30]
0
2
```

insert()

- 리스트 요소 삽입

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

```
[10, 40, 20, 30]
[10, 40, 20, 30, 50]
[60, 10, 40, 20, 30, 50]
```

remove()

- 리스트 요소 제거

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

```
[10, 40, 20, 30]
[10, 20, 30]
[10, 20]
```

del

- 리스트 요소 제거 연산

```
list = [10, 40, 20, 30]
print(list)
del list[0]
print(list)
del list[2]
print(list)
```

```
[10, 40, 20, 30]
[40, 20, 30]
[40, 20]
```

pop()

- 리스트 요소를 방출

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

```
[10, 40, 20, 30]
[10, 40, 20]
[40, 20]
```

count()

- 리스트 요소의 갯수 계산

```
list = [10, 20, 20, 30, 30, 30]
print(list)
print(list.count(30))
print(list.count(20))
```

```
[10, 20, 20, 30, 30, 30]
3
2
```

extend()

- 리스트 확장

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

```
[10, 40, 20, 30]
[10, 40, 20, 30, 50, 60]
```

튜플(Tuple)

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

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

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

튜플 인덱싱(Tuple Indexing)

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

```
tuple = ('One', 'Two', 'Three')
print(tuple)
print(tuple[0])
print(tuple[-1])
```

```
('One', 'Two', 'Three')
One
Three
```

중첩 튜플 인덱싱(Nested Tuple Indexing)

- 튜플 안에 튜플이 중첩되어 있을 경우, 인덱스 접근 방법

```
tuple = (1, 2, 3, ('One', 'Two', 'Three'))
print(tuple)
print(tuple[3])
print(tuple[3][1])
```

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

튜플 슬라이싱(Tuple Slicing)

- 튜플의 인덱스를 통해서 부분만 가져올 때 사용

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

```
('One', 'Two', 'Three')
('One', 'Two', 'Three')
('Two',)
('Two', 'Three')
```

중첩 튜플 슬라이싱(Nested Tuple Slicing)

- 중첩된 튜플에서 일부분만 인덱스를 통해서 가져올 때 사용

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

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

튜플 연산자/함수(Tuple Operators/Function)

- 더하기 + 연산
- 곱하기 * 연산
- 튜플 길이를 구하는 `len()` 함수

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

```
('One', 'Two', 'Three')
('Four', 'Five', 'Six')
('One', 'Two', 'Three', 'Four', 'Five', 'Six')
('One', 'Two', 'Three', 'One', 'Two', 'Three', 'One', 'Two', 'Three')
3
6
```

세트(Set)

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

```
print({})
print({'One', 'Two', 'Three'})
print({10, 20, 30, 40})
```

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

세트 연산자(Set Operators)

- 교집합: &
- 합집합: |
- 차집합: -
- 여집합: ^

```
set_1 = {10, 20, 20, 30}
set_2 = {30, 30, 40, 50}
print(set_1)
print(set_2)
print(set_1 & set_2)
print(set_1 | set_2)
print(set_1 - set_2)
print(set_1 ^ set_2)
```

```
{10, 20, 30}
{40, 50, 30}
{30}
{50, 20, 40, 10, 30}
{10, 20}
{40, 10, 50, 20}
```

세트 메소드(Set Methods)

- 교집합: intersection()
- 합집합: union()
- 차집합: difference()
- 여집합: symmetric_difference()

```
set_1 = {10, 20, 20, 30}
set_2 = {30, 30, 40, 50}
print(set_1)
print(set_2)
print(set_1.intersection(set_2))
print(set_1.union(set_2))
print(set_1.difference(set_2))
print(set_1.symmetric_difference(set_2))
```

```
{10, 20, 30}
{40, 50, 30}
{30}
{50, 20, 40, 10, 30}
{10, 20}
{40, 10, 50, 20}
```

- 요소 추가: add()
- 여러 요소 추가: update()
- 요소 제거: remove()
- 요소 제거: discard()
- 모든 요소 제거: clear()

```
set = {10, 20, 30, 40}
print(set)

set.add(50)
print(set)

set.update([60, 70])
print(set)

set.remove(70)
set.remove(60)
print(set)

set.discard(30)
print(set)
```

```
print(set)

set.clear()
print(set)
```

```
{40, 10, 20, 30}
{40, 10, 50, 20, 30}
{70, 40, 10, 50, 20, 60, 30}
{40, 10, 50, 20, 30}
{40, 10, 50, 20}
set()
```

딕셔너리(Dictionary)

- 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'}
print(dic)

dic = {1:'One', 2:'Two', 3:'Three', 1:'One', 2:'Two', 3:'Three'}
print(dic)
```

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

딕셔너리 요소 추가/삭제

- 딕셔너리의 해당 키 값에 값을 추가하여 요소 추가
- del을 이용하여 요소 제거

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

```
{1: 'One', 2: 'Two', 3: 'Three'}
Two
{1: 'One', 2: 'Two', 3: 'Three', 4: 'Four'}
{1: 'One', 2: 'Two', 3: 'Three', 4: 'Four', 5: 'Five'}
{1: 'One', 2: 'Two', 3: 'Three', 5: 'Five'}
```

딕셔너리 메소드(Dictionary Methods)

메소드	설명
keys()	딕셔너리의 키 가져오기
values()	딕셔너리의 값 가져오기
items()	딕셔너리의 키와 값을 모두 가져오기
get()	딕셔너리에서 키에 해당하는 값 가져오기
pop()	딕셔너리에서 키에 해당하는 값 추출하기
clear()	딕셔너리의 모든 요소를 제거하기

```
dic = {1:'One', 2:'Two', 3:'Three'}
print(dic)
print(dic.keys())
print(dic.values())
print(dic.items())
print(dic.get(2))
print(dic.pop(3))
print(dic)
dic.clear()
print(dic)
```

```
{1: 'One', 2: 'Two', 3: 'Three'}
dict_keys([1, 2, 3])
dict_values(['One', 'Two', 'Three'])
dict_items([(1, 'One'), (2, 'Two'), (3, 'Three')])
Two
Three
{1: 'One', 2: 'Two'}
{}
```