



# 시퀀스 데이터타입과 딕셔너리

소프트웨어개발단 박경규

# 목차

- 시퀀스 데이터 타입
- 리스트(List)
- 인덱싱과 슬라이싱
- 리스트 메소드(Method)
- 리스트 코딩실습
- 튜플(Tuple)
- 집합(Set)
- 딕셔너리(Dictionary)
- 딕셔너리 메소드(Method)

## 학습목표

- 파이썬의 시퀀스 데이터 타입에 대해 이해를 하고 시퀀스 데이터 타입 사용법을 숙지한다.
- 파이썬의 딕셔너리 데이터 타입을 이해 하고, 딕셔너리 데이터 타입의 각종 메서드 사용법을 학습한다.

# 시퀀스 데이터 타입

- 파이썬은 데이터를 묶어서 처리할 수 있는 시퀀스 데이터 타입을 제공합니다. 과목별 시험점수 처리 등 데이터를 묶어서 관리하면 편합니다.
- 시퀀스 데이터 타입으로 문자열, 리스트, 튜플이 있으며, 시퀀스 데이터의 항목(item) 또는 엘리먼트(element)는 정의된 순서로 정렬되며, 인덱스를 통해 액세스 할 수 있습니다.

```
>>> text = "Python is powerful"
>>> print(text[0], text[10], text[-1])
P i l
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P	y	t	h	o	n		i	s		p	o	w	e	r	f	u	l
-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> names = ['Minho', 'Seoyun', 'Dongsu']
>>> print(names[0], names[2])
Minho Dongsu
```

# 시퀀스 데이터 타입

- 동일한 구문 및 함수를 사용하여 시퀀스 데이터 타입을 처리합니다.
- 문자열, 리스트, 튜플의 길이는 len()이라는 함수로 알 수 있습니다.

## 문자열(String)

```
>>> text = "Python is powerful"
>>> print(len(text))
18
```

## 리스트(List)

```
>>> names = ["Minho", "Seoyun", "Dongsu"]
>>> print(len(names))
3
```

## 튜플(Tuple)

```
>>> names = ("Minho", "Seoyun", "Dongsu")
>>> print(len(names))
3
```

# 리스트(List)

- 리스트 데이터 타입 유형은 파이썬 프로그램에 필수적입니다.
- 파이썬의 리스트는 정렬된 아이템이나 엘리먼트의 그룹입니다.
- 리스트의 엘리먼트는 동일한 타입일 필요는 없으며, 숫자, 문자열, 기타 리스트 등의 엘리먼트가 임의로 혼합(mixed) 될 수 있습니다.
- 리스트는 대괄호([ ])를 이용해 만듭니다.

리스트 표기	설명
<code>[]</code>	빈 리스트
<code>[1, 2, 3, 4, 5]</code>	정수 리스트
<code>[1, "Math exam score", 95]</code>	믹스드 데이터 타입 리스트
<code>["Minho", "Seoyun", "Dongsu"]</code>	문자열 리스트
<code>[["Minho", "Math", 95], ["Seoyun", "Math", 85], ["Dongsu", "Math", 75]]</code>	중첩 리스트

# 인덱싱과 슬라이싱

- 리스트도 문자열처럼 인덱싱과 슬라이싱이 가능합니다.

```
>>> a = [1, 2, 3, 4, 5]
>>> a[0]
1
>>> a[-1]
5

>>> a[0:2]
[1, 2]

>>> a = a + [6, 7, 8, 9, 10]
>>> a
[1, 2, 3, 4, 5, 10, 6, 7, 8, 9, 10]

>>> a[10] = 0
>>> a
[1, 2, 3, 4, 5, 10, 6, 7, 8, 9, 0]
```

# 리스트 메서드(Method)

리스트 메서드	설명	사용예
append(x)	아이템 하나를 리스트에 추가	todo.append('study')
extend( )	아이템 여러 개를 리스트에 추가	todo.extend(['homework', 'workout', 'game'])
insert(i, x)	아이템을 리스트 특정위치에 삽입	todo.insert(1, 'cleaning')
remove(x)	일치하는 첫 아이템을 리스트에서 제거	todo.remove('game')
pop()	리스트의 마지막 항목을 제거	todo.pop()
index(x)	리스트에서 값이 x와 같은 첫 번째 아이템의 인덱스를 반환	n = todo.index('study')
count(x)	리스트에서 아이템이 나타나는 횟수를 반환	i = todo.count('study')
sort()	리스트 아이템을 정렬	todo.sort()
reverse()	리스트 아이템을 역순으로 정렬	todo.reverse()
clear()	리스트의 모든 아이템을 제거	todo.clear()



# 리스트 코딩 실습

```
IPython console
Console 1/A x

In [1]: todo = []

In [2]: todo.append('study')

In [3]: todo
Out[3]: ['study']

In [4]: todo.extend(['homework', 'workout', 'game'])

In [5]: todo
Out[5]: ['study', 'homework', 'workout', 'game']

In [6]: todo.insert(1, 'cleaning')

In [7]: todo
Out[7]: ['study', 'cleaning', 'homework', 'workout', 'game']

In [8]: todo.remove('game')

In [9]: todo
Out[9]: ['study', 'cleaning', 'homework', 'workout']

In [10]: todo.pop()
Out[10]: 'workout'
```

```
In [11]: todo
Out[11]: ['study', 'cleaning', 'homework']

In [12]: n = todo.index('study')

In [13]: print(n)
0

In [14]: i = todo.count('study')

In [15]: print(i)
1

In [16]: todo.sort()

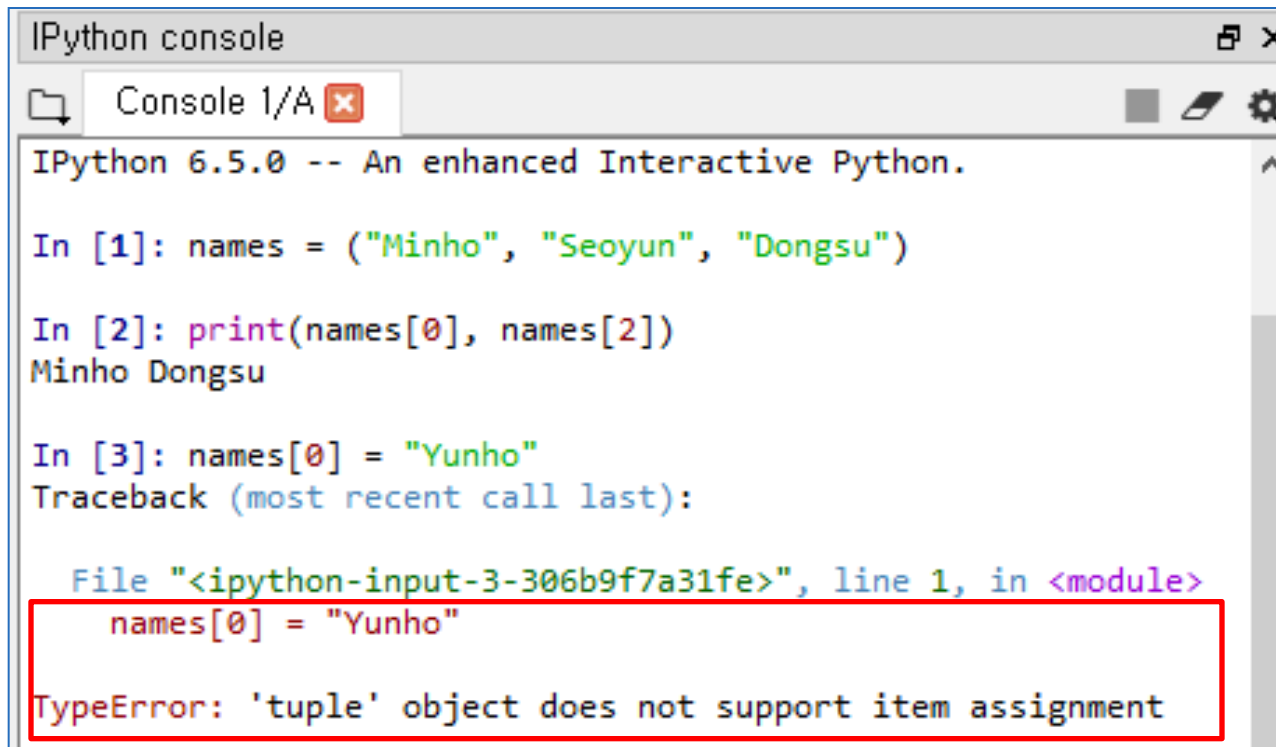
In [17]: todo
Out[17]: ['cleaning', 'homework', 'study']

In [18]: todo.reverse()

In [19]: todo
Out[19]: ['study', 'homework', 'cleaning']
```

# 튜플(Tuple)

- 튜플은 불변 리스트(immutable list)입니다.  
튜플이 생성되면 어떤 방법으로든 변경할 수 없습니다.
- 튜플은 엘리먼트 집합이 대괄호 대신 괄호로 묶여 있다는 점을 제외하고는 목록과 유사하게 정의됩니다.



```
IPython console
Console 1/A x
IPython 6.5.0 -- An enhanced Interactive Python.

In [1]: names = ("Minho", "Seoyun", "Dongsu")

In [2]: print(names[0], names[2])
Minho Dongsu

In [3]: names[0] = "Yunho"
Traceback (most recent call last):

  File "<ipython-input-3-306b9f7a31fe>", line 1, in <module>
    names[0] = "Yunho"
TypeError: 'tuple' object does not support item assignment
```

# 세트(Set)

- 파이썬은 집합을 위한 데이터 타입 세트(Set)를 제공합니다.
- 세트는 중복되는 엘리먼트가 없는 순서없는 컬렉션입니다.
- 세트는 합집합, 교집합, 차집합과 같은 수학적인 연산을 지원합니다.
- 집합을 만들 때는 중괄호({ })나 set() 함수를 사용합니다.  
빈 집합을 만들려면 set() 을 사용해야 합니다.

```
>>> fruits = {'apple', 'orange', 'apple', 'orange', 'banana'}  
>>> print(fruits)  
{'banana', 'apple', 'orange'}
```

# 집합(Set)

- 세트는 메서드(Method) 또는 연산자(Operator)를 이용하여 수학 집합 연산을 할 수 있습니다.

연산	메서드	연산자
합집합	A.union(B)	A   B
교집합	A.intersection(B)	A & B
차집합	A.difference(B)	A - B

```
IPython console
Console 1/A x

In [11]: A = {0, 1, 2, 3, 4, 5, 6}
In [12]: B = {3, 4, 5, 6, 7, 8, 9}

In [13]: A.union(B)
Out[13]: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

In [14]: A | B
Out[14]: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

In [15]: A.intersection(B)
Out[15]: {3, 4, 5, 6}

In [16]: A & B
Out[16]: {3, 4, 5, 6}

In [17]: A.difference(B)
Out[17]: {0, 1, 2}

In [18]: A - B
Out[18]: {0, 1, 2}

In [19]:
```

# 딕셔너리(Dictionary)

- 딕셔너리는 파이썬을 효과적이고 탁월하게 만드는 데이터 타입입니다.
- 딕셔너리는 데이터 전체를 중괄호({ })로 감싸고 키와 값의 구분은 콜론(:)으로, 키와 값으로 이루어진 각 쌍은 콤마(,)로 구분합니다.  
`{key1: value1, key2: value2, key3: value3 ...}`
- 리스트와 마찬가지로 쉽게 변경할 수 있으며 실행시에 축소되거나 확장될 수 있습니다.
- 리스트와 딕셔너리의 차이점은
  - 리스트는 정렬 된 객체의 시퀀스이며 위치를 통해 액세스 됩니다.
  - 딕셔너리는 정렬되지 않은 집합이며 키(key)를 통해 액세스 됩니다.

# 딕셔너리 메소드

메서드	설명	사용예
len(d)	저장된 항목 수, 즉 (키, 값) 쌍 수를 반환합니다.	len(fruits)
del d [k]	키 k를 값과 함께 삭제합니다.	del fruits['apple']
k in d	사전 d에 키 k가있는 경우 참	'banana' in fruits
k not in d	키 k가 사전 d에 존재하지 않으면 참	'apple' not in fruits
keys()	딕셔너리의 키 전체를 리스트로 반환	fruits.keys()
values()	딕셔너리의 값 전체를 리스트로 반환	fruits.values()
items()	딕셔너리의 아이템을 튜플(키, 값)로 반환	fruits.items()
clear()	딕셔너리의 모든 항목 삭제	fruits.clear()

# 딕셔너리 코딩 실습

```
IPython console
Console 1/A

In [1]: exam_score = {'Minho': 95, 'Seoyun': 85, 'Dongsu': 75}

In [2]: exam_score['Minho']
Out[2]: 95

In [3]: exam_score['Seoyun']
Out[3]: 85

In [4]: exam_score['Dongsu'] = 90

In [5]: exam_score['Dongsu']
Out[5]: 90

In [6]: exam_score
Out[6]: {'Minho': 95, 'Seoyun': 85, 'Dongsu': 90}

In [7]: exam_score['Yunsu'] = 100

In [8]: exam_score
Out[8]: {'Minho': 95, 'Seoyun': 85, 'Dongsu': 90, 'Yunsu': 100}

In [9]: del exam_score['Yunsu']

In [10]: exam_score
Out[10]: {'Minho': 95, 'Seoyun': 85, 'Dongsu': 90}

In [11]:
```

```
In [11]: fruits = {'apple': '사과', 'banana': '바나나', 'orange': '오렌지'}

In [12]: fruits
Out[12]: {'apple': '사과', 'banana': '바나나', 'orange': '오렌지'}

In [13]: len(fruits)
Out[13]: 3

In [14]: del fruits['apple']

In [15]: fruits
Out[15]: {'banana': '바나나', 'orange': '오렌지'}

In [16]: 'banana' in fruits
Out[16]: True

In [17]: 'apple' not in fruits
Out[17]: True

In [18]: fruits['apple'] = '사과'

In [19]: fruits
Out[19]: {'banana': '바나나', 'orange': '오렌지', 'apple': '사과'}

In [20]: fruits.keys()
Out[20]: dict_keys(['banana', 'orange', 'apple'])

In [21]: fruits.values()
Out[21]: dict_values(['바나나', '오렌지', '사과'])

In [22]: fruits.items()
Out[22]: dict_items([('banana', '바나나'), ('orange', '오렌지'), ('apple', '사과')])

In [23]: fruits.clear()

In [24]: fruits
Out[24]: {}
```



# Comprehension

- 파이썬의 Comprehension은 입력시퀀스로부터 다른 시퀀스(리스트, 세트, 딕셔너리)를 만드는 간결한 방법을 제공합니다.
- 리스트 Comprehension은 입력 시퀀스로부터 지정된 표현식에 따라 새로운 리스트를 빌드하는 것으로, 문법은 아래와 같습니다.

[출력표현식 for 요소 in 입력시퀀스 [if 조건식]]

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```



리스트  
Comprehension

```
>>> squares = [x**2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

# Comprehension

- 세트 Comprehension은 리스트 Comprehension과 비슷하며, 결과가 Set {...}으로 리턴된다는 점이 다릅니다.  
{출력표현식 for 요소 in 입력Sequence [if 조건식]}
- 딕셔너리 Comprehension는 출력표현식이 key:value pair로 표현되며 결과로 dictionary가 리턴됩니다.  
{key:value for 요소 in 입력시퀀스 [if 조건식]}

## 세트 Comprehension

```
>>> squares = (x**2 for x in range(10))
>>> squares
(0, 1, 4, 9, 16, 25, 36, 49, 64, 81)
```

## 딕셔너리 Comprehension

```
>>> dict1 = {1: 'one', 2: 'two', 3: 'three'}
>>> dict2 = {v: k for k, v in dict1.items()}
>>> dict1
{1: 'one', 2: 'two', 3: 'three'}
>>> dict2
{'one': 1, 'two': 2, 'three': 3}
```



danny.park@kt.com  
kgpark88@gmail.com