

자료구조란

- 여러 개의 값들을 모아서 관리하는 데이터 타입.
 - 한 개의 변수는 한 개의 값 밖에는 가지지 못한다. 그러나 하나의 변수로 여러 개의 값 묶어서 저장해 관리해야 할 경우가 있다.
 - 예) 고객의 정보의 경우 이름, 나이, 주소, 전화번호 등 여러개의 값이 모여서 하나의 값이 된다.
- 파이썬은 데이터를 모으는 방식에 따라 다음과 같이 4개의 타입을 제공한다.
 - List: 순서가 있으며 중복된 값들을 모으는 것을 허용하고 구성하는 값들(원소)을 변경할 수 있다.
 - Tuple: 순서가 있으며 중복된 값들을 모으는 것을 허용하는데 구성하는 값들을 변경할 수 없다.
 - Dictionary: key-value 형태로 값들을 저장해 관리한다.
 - Set: 중복을 허용하지 않고 값들의 순서가 없다.
- 원소, 성분, 요소, element
 - 자료구조의 값들을 구성하는 개별 값들을 말한다.
 - len(자료구조) 함수
 - 자료구조 내의 원소의 개수를 반환한다.

List (리스트)

- 값을 순서대로 모아서 관리하는 자료구조. 원소(element)들을 순번을 이용해 식별한다.
 - 각각의 원소가 어떤 값인지를 순번을 가지고 식별하기 때문에 순서가 매우 중요하다. 즉 같은 값에 대해 순서가 바뀌면 안된다.
- 각 원소들은 순번을 index라고 하며 값을 조회하거나 변경할 때 index를 이용해 식별한다.
 - index는 문자열과 마찬가지로 양수 index와 음수 index 두개가 각 값에 생긴다.
 - 양수 index는 앞에서부터 음수 index는 뒤에서 부터 값을 식별할 때 사용하는 것이 편리하다.
- 중복된 값들을 저장할 수 있다.
- 각 원소들의 데이터 타입은 달라도 상관없다.
 - 보통은 같은 타입의 데이터를 모은다.
- 리스트를 구성하는 원소들을 변경할 수 있다. (추가, 삭제, 변경이 가능)

List 생성 구문

[값, 값, 값, ...]

In [1]:

```
l1 = [10, 20, 30, 40, 50, ]
print(type(l1))
```

<class 'list'>

In [2]:

```
print(l1)
```

[10, 20, 30, 40, 50]

In [3]:



```
l2 = [10, 50.8, 7., True, False, "안녕하세요"]
print(l2)
```

```
[10, 50.8, 7.0, True, False, '안녕하세요']
```

In [4]:



```
l3 = [10, [1,2,3], ('a', 'b'), {"이름":"홍길동", "나이":20}]
# 리스트(자료구조) 안에 다른 자료구조들을 원소로 넣을 수 있다.
l3
```

Out[4]:

```
[10, [1, 2, 3], ('a', 'b'), {'이름': '홍길동', '나이': 20}]
```

In [7]:



```
# 자료구조의 원소들의 개수 조회
print(len(l1))
print(len(l2))
print(len(l3))
```

5

6

4

Indexing과 Slicing을 이용한 원소(element) 조회 및 변경

Indexing

- 하나의 원소를 조회하거나 변경할 때 사용
- 리스트[index]
 - index의 원소를 조회
- 리스트[index] = 값
 - index의 원소를 변경

Slicing

- 범위로 조회하거나 그 범위의 값들을 변경한다.
- 기본구문: 리스트[시작 index : 종료 index : 간격]
 - 시작 index ~ (종료 index - 1)
 - 간격을 지정하면 간격만큼 index를 증/감한다. (생략 시 1이 기본 간격)
- 0번 index 부터 조회 할 경우 시작 index는 생략가능
 - 리스트[:5] => 0 ~ 4 까지 조회
- 마지막 index까지 (끝까지) 조회 할 경우 종료 index는 생략 가능
 - 리스트[2:] => 2번 index 에서 끝까지
- 명시적으로 간격을 줄 경우
 - 리스트[: : 3] => 0, 3, 6, 9.. index의 값 조회
 - 리스트[1 : 9 : 2] => 1, 3, 5, 7 index의 값 조회

- 시작 index > 종료 index, 간격을 음수로 하면 역으로 반환한다.(Reverse)

- 리스트[5: 1: -1] => 5, 4, 3, 2 index의 값 조회
- 리스트[: -1] => 마지막 index ~ 0번 index 까지 의미. Reverse 한다.

slicing을 이용한 값 변경

- slicing 을 이용할 경우 slicing된 원소 개수와 동일한 개수의 값들을 대입한다.
 - 리스트[1:5] = 10,20,30,40 : index 1, 2, 3, 4의 값을 각각 10, 20, 30, 40 으로 변경

In [9]:



```
l = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
# indexing
l[0] # 첫번째 값. index 0의 값 조회
l[3] # 네번째 값. index 3의 값 조회
```

Out[9]:

3

In [10]:



```
l[3] + l[7]
```

Out[10]:

10

In [12]:



```
# 음수 index
l[0], l[-10]
l[-3], l[-5]
```

Out[12]:

(7, 5)

In [18]:



```
# 특정 index의 값을 변경 -> 대입연산자
print(l)
l[0] = 1000
print(l)
```

[1000, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[1000, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [17]:



```
# s = "abc"
# s[0] = "가"
```

...

In [20]:



```
l[3] = 30000  
print(l)
```

```
[1000, 1, 2, 30000, 4, 5, 6, 7, 8, 9]
```

In [22]:



```
# slicing  
l[1:8:3] # 1, 4, 7
```

Out[22]:

```
[1, 4, 7]
```

In [23]:



```
l[1:7] #간격: 1
```

Out[23]:

```
[1, 2, 30000, 4, 5, 6]
```

In [24]:



```
l[1:-2]
```

Out[24]:

```
[1, 2, 30000, 4, 5, 6, 7]
```

In [25]:



```
l[ : 5] # 0 ~ 4
```

Out[25]:

```
[1000, 1, 2, 30000, 4]
```

In [26]:



```
l[5 : ] # 5 ~ 끝까지
```

Out[26]:

```
[5, 6, 7, 8, 9]
```

In [27]:



```
l[::-1]
```

Out[27]:

```
[9, 8, 7, 6, 5, 4, 30000, 2, 1, 1000]
```

In [28]:



```
l[6:1:-2]
```

Out[28]:

```
[6, 4, 2]
```

In [30]:



```
# 변경
l[:3] = 500, 900, 800 #0: 500, 1: 900, 2: 800
```

In [31]:



```
l
```

Out[31]:

```
[500, 900, 800, 30000, 4, 5, 6, 7, 8, 9]
```

List 연산자

- 리스트 + 리스트
 - 두 리스트의 원소들을 합친 리스트를 반환한다.
- 리스트 * 정수
 - 같은 리스트의 원소들을 정수번 합친 리스트를 반환한다.
- in, not in 연산자
 - 값 in 리스트
 - 리스트의 원소로 값이 **있으면** True, 없으면 False 반환
 - 값 not in 리스트
 - 리스트의 원소로 값이 **없으면** True, 있으면 False 반환
- len(리스트)
 - 리스트 내의 원소수를 반환.

In [32]:



```
a = [1, 2, 3]
b = ['가', '나']
c = a + b
print(a)
print(b)
print(c)
```

```
[1, 2, 3]
['가', '나']
[1, 2, 3, '가', '나']
```

In [33]:



```
d = a + b + c
print(d)
```

```
[1, 2, 3, '가', '나', 1, 2, 3, '가', '나']
```

In [35]:



```
e = a * 3
print(e)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

In [39]:



```
print(1 in a)
print(10 in a)
```

```
True
False
```

In [40]:



```
print(1 not in a)
print(10 not in a)
```

```
False
True
```

In [42]:



```
print([1, 2] in a) # a안에 [1,2] 가 있나?
```

```
False
```

In [44]:



```
[1,2] in [[1,2], [3,4,5]]
```

Out[44]:

```
True
```

In [49]:



```
(1 in a) and (2 in a)
```

Out[49]:

```
False
```

중첩 리스트 (Nested List)

- List가 원소로 List를 가지는 것을 말한다.
 - List를 포함한 모든 자료구조 타입들도 다 값이므로 다른 자료구조의 원소로 들어갈 수 있다.

In [3]:

```
l2 = [  
    [10, 20, 30, 40],  
    [100, 200, [1, 2, [1, 3, 4]]],  
    [1000, 2000]  
]
```

In [5]:

```
l2[1][2][2][1]
```

Out[5]:

3

In [51]:

```
len(l2)
```

Out[51]:

3

In [53]:

```
a = l2[1]  
a
```

Out[53]:

[100, 200, 300]

In [54]:

```
a[1]
```

Out[54]:

200

In [55]:

```
l2[1][1]
```

Out[55]:

200

List 대입

리스트의 원소들을 각각 다른 변수에 대입하는 표현식

In [57]:

```
# num1 = 10
# num2 = 20
# num3 = 30

num1, num2, num3 = [10, 20, 30]  #변수의 개수와 리스트원소의 개수는 동일해야한다.
print(num1)
print(num2)
print(num3)
```

10
20
30

In [58]:

```
# 튜플대입
a, b, c = 1, 2, 3
print(a)
print(b)
print(c)
```

...

List 주요 메소드

메소드	설명
append(value)	value를 추가한다.
extend(List)	List의 원소들을 추가한다.
sort([reverse=False])	원소들을 오름차순 정렬한다. reverse=True로 하면 내림차순정렬 한다.
insert(index, 삽입할값)	지정한 index에 '삽입할값'을 삽입한다.
remove(삭제할값)	'삭제할값' 값과 같은 원소를 삭제한다.
index(찾을값[, 시작index])	'찾을값'의 index를 반환한다.
pop([index])	index의 값을 반환하면서 삭제한다. index 생략하면 가장 마지막 값을 반환하며 삭제한다.
count(값)	'값'이 리스트의 원소로 몇개 있는지 반환한다.
clear()	리스트 안의 모든 원소들을 삭제한다.

In [59]:



```
result_list = []  
result_list.append(1 + 1) # 처리결과를 결과리스트에 추가.  
result_list.append(2 * 3)  
result_list.append(10/5)
```

In [60]:



```
result_list
```

Out[60]:

```
[2, 6, 2.0]
```

In [61]:



```
l = [100, 200, 300]  
result_list + l # 두 리스트의 원소들을 합친 새로운 리스트를 생성
```

Out[61]:

```
[2, 6, 2.0, 100, 200, 300]
```

In [64]:



```
result_list.extend(l) # result_list에 리스트 l의 원소들을 추가.
```

In [65]:



```
result_list
```

Out[65]:

```
[2, 6, 2.0, 100, 200, 300]
```

In [66]:



```
l2 = [100, 5, 20, 70, 23, 54]  
# 정렬 - 오름차순  
l2.sort()
```

In [67]:



```
print(l2)
```

```
[5, 20, 23, 54, 70, 100]
```

In [68]:



```
# 정렬 - 내림차순
l2.sort(reverse=True)
```

In [69]:



```
print(l2)
```

```
[100, 70, 54, 23, 20, 5]
```

In [72]:



```
l3 = [100, 5, 20, 70, 23, 54]
# 정렬 함수
sorted(l3, reverse=True) # l3를 정렬한 값들을 원소로 가지는 새로운 리스트를 생성.
```

Out[72]:

```
[100, 70, 54, 23, 20, 5]
```

In [73]:



```
l3
```

Out[73]:

```
[100, 5, 20, 70, 23, 54]
```

In [77]:



```
# 삭제
l = [1, 2, 3, 4, 5, 1]
print(len(l))
# 값으로 삭제
l.remove(1) # l의 원소들중 1을 삭제
print(len(l))
```

```
6
```

```
5
```

In [79]:



```
l[0]
```

Out[79]:

```
2
```

In [81]:



```
# index로 삭제
print(l)
v = l.pop(1)
print(v)
print(l)
```

```
[2, 3, 4, 5, 1]
3
[2, 4, 5, 1]
```

In [84]:



```
print(l)
del l[1]
print(l)
```

```
[2, 4, 5, 1]
[2, 5, 1]
```

In [86]:



```
# 다지우기
l.clear()
print(l)
```

```
[]
```

In [88]:



```
l = [1, 1, 1, 3, 2, 3, 3, 2, 1, 1, 3, 2]
l.count(1)
l.count(3)
```

Out[88]:

```
4
```

In []:



Tuple (튜플)

- List와 같이 순서대로 원소들을 관리한다. 단 저장된 원소를 변경할 수 없다.
- Tuple 은 각 위치(Index) 마다 정해진 의미가 있고 그 값이 한번 설정되면 바뀌지 않는 경우에 사용한다.
 - Tuple은 값의 변경되지 않으므로 안전하다.

Tuple 생성

- (value, value, value, ...)
- 소괄호를 생략할 수 있다.

- 원소가 하나인 Tuple 표현식
 - (value,) 또는 value,
 - 값 뒤에 , 를 붙여준다. , 를 붙이지 않으면 ()가 연산자 우선순위 괄호가 된다.

Indexing과 Slicing을 이용한 원소(element) 조회

- 리스트와 동일하다.
- 단 튜플은 조회만 가능하고 원소를 변경할 수 없다.

Tuple 연산자

- tuple + tuple
 - 두 tuple의 원소들을 합친 tuple을 반환한다.
- tuple * 정수
 - 같은 tuple의 원소들을 정수번 합친 tuple을 반환한다.
- in, not in 연산자
 - 값 in tuple
 - tuple의 원소로 값이 **있으면** True, 없으면 False 반환
 - 값 not in tuple
 - tuple의 원소로 값이 **없으면** True, 있으면 False 반환
- len(tuple)
 - tuple의 원소 개수 반환

In [7]:

```
# 생성
t1 = (1, 2, 3, 4, 5)
t2 = (1, True, 3.5, "가나다")
t3 = (1, 1, 2, 3, 1, 1, 2,) # 중복 허용
t4 = (1)
t5 = (1,)
```

In [9]:

```
print(type(t1))
print(t1)
print(t2)
print(t3)
print(type(t4)) # 값이 하나인 튜플로 묶을때 쉼표를 뒤에 해줘야함. -> 연산 처리결과를 튜플로 전달
print(type(t5), len(t5))
```

```
<class 'tuple'>
(1, 2, 3, 4, 5)
(1, True, 3.5, '가나다')
(1, 1, 2, 3, 1, 1, 2)
<class 'int'>
<class 'tuple'> 1
```

In [101]:



```
# tuple 생성시 ( )는 생략가능.
t1 = 1, 2, 3, 4, 5
t2 = 1, True, 3.5, "가나다"
t3 = 1, 1, 2, 3, 1, 1, 2, # 중복 허용
t4 = 1
t5 = 1,
```

In [102]:



```
print(type(t1))
print(t1)
print(t2)
print(t3)
print(type(t4))
print(type(t5), len(t5))
```

```
<class 'tuple'>
(1, 2, 3, 4, 5)
(1, True, 3.5, '가나다')
(1, 1, 2, 3, 1, 1, 2)
<class 'int'>
<class 'tuple'> 1
```

In [105]:



```
# indexing
t1[0], t1[-2]
```

Out[105]:

(1, 4)

In [106]:



```
t1[0] = 1000 #변경은 할 수 없다.
```

```
-----
-----
TypeError                                Traceback (most recent call last)
Cell In[106], line 1
----> 1 t1[0] = 1000
```

```
TypeError: 'tuple' object does not support item assignment
```

In [108]:



```
t1[1:4]
```

Out[108]:

(2, 3, 4)

In [109]:



```
t1[:]
```

Out[109]:

```
(1, 2, 3, 4, 5)
```

In [110]:



```
t1[::2]
```

Out[110]:

```
(1, 3, 5)
```

In [111]:



```
# 연산자
t1 = (1, 2, 3, 4, 5)
t2 = 10, 20

t3 = t1 + t2
print(t3)
```

```
(1, 2, 3, 4, 5, 10, 20)
```

In [112]:



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

```
(1, 2, 3, 4, 5)
(10, 20)
```

In [113]:



```
t4 = t2 * 3
print(t4)
```

```
(10, 20, 10, 20, 10, 20)
```

In [114]:



```
print(t2)
```

```
(10, 20)
```

In [116]:



```
10 in t4, 300 in t4
```

Out[116]:

```
(True, False)
```

In [119]:



```
10 not in t4, 300 not in t4
```

Out[119]:

```
(False, True)
```

Tuple 대입

- Tuple의 원소들을 각각 다른 변수에 대입하는 표현식

In [121]:



```
a, b, c = 10, 20, 30
# a, b, c = (10, 20, 30)
print(a, b, c)
```

```
10 20 30
```

In []:



Tuple의 주요 메소드

메소드	설명
index(찾을값 [, 시작index])	'찾을값'이 몇번 index인지 반환한다.
count(값)	원소로 '값'이 몇개 있는지 반환한다.

In [124]:



```
t1.index(2)
t1.index(100) # 없는 것 찾을 경우 에러발생.
```

```
-----
-----
ValueError                                Traceback (most recent call last)
Cell In[124], line 2
      1 t1.index(2)
----> 2 t1.index(100)

ValueError: tuple.index(x): x not in tuple
```

Dictionary

- 값을 키(key)-값(value) 쌍으로 묶어서 저장하는 자료구조이다.
 - 리스트나 튜플의 index의 역할을 하는 key를 직접 지정한다.
 - 서로 의미가 다른 값들을 하나로 묶을 때 그 값의 의미를 key로 가질 수 있는 dictionary를 사용한다.
 - cf) 값의 의미가 같을 경우 List나 Tuple을 사용한다.
 - key-value 쌍으로 묶은 데이터 한개를 **item 또는 entry**라고 한다.
 - key는 중복을 허용하지 않고 value는 중복을 허용한다.

Dictionary 생성

- 구문
 1. { 키 : 값, 키 : 값, 키 : 값 }
 2. dict(key=value, key=value) 함수 이용
 - 키(key)는 불변(Immutable)의 값들만 사용 가능하다. (숫자, 문자열, 튜플) 일반적으로 문자열을 사용한다.
 - dict() 함수를 사용할 경우 key는 변수로 정의한다

In []:



Dictionary 원소 조회 및 변경

- 조회: index에 key값을 식별자로 지정한다.
 - dictionary[key]
 - 없는 키로 조회 시 KeyError 발생
- 변경
 - dictionary[key] = 값
 - 있는 key값에 값을 대입하면 변경이고 없는 key 일 경우는 새로운 item을 추가하는 것이다.

In []:

▶

Dictionary 연산자

- in, not in 연산자
 - 값 in dictionary
 - dictionary의 **Key**로 값이 **있으면** True, 없으면 False 반환
 - 값 not in tuple
 - dictionary의 **Key**로 값이 **없으면** True, 있으면 False 반환
- len(dictionary)
 - dictionary의 **Item**의 개수 반환

In []:

▶

Dictionary 주요 메소드

메소드	설명
get(key[, 기본값])	key의 item의 값을 반환한다. 단 key가 없을 경우 None또는 기본값을 반환한다.
pop(key)	key의 item의 값을 반환하면서 dictionary에서 삭제한다. 없는 key일 경우 KeyError발생
clear()	dictionary의 모든 item들을 삭제한다.
del dict[key]	key의 item을 제거한다.
items()	item의 key, value를 튜플로 묶어 모아 반환한다.
keys()	key값들만 모아 반환한다.
values()	value값들만 모아 반환한다.

In []:

▶

Set

- Set은 중복되는 값을 허용하지 않고 순서를 신경 쓰지 않는다.
 - 원소를 식별할 수 있는 식별자가 없기 때문에 Set은 indexing과 slicing을 지원하지 않는다

Set 생성

- 구문
 - {값, 값, 값 }

-빈 Dictionary 만들기

- info = {}

In []:



Set 연산자

- in, not in 연산자
 - 값 in Set
 - Set의 원소로 값이 **있으면** True, 없으면 False 반환
 - 값 not in Set
 - Set의 원소로 값이 **없으면** True, 있으면 False 반환
- len(Set)
 - Set의 원소의 개수 반환

In []:



Set의 주요 메소드

메소드	설명
add(값)	집합에 값 추가
update(자료구조)	자료구조내의 원소들을 모두 집합에 추가
pop()	원소를 반환하고 Set에서 삭제한다.
remove(값)	값을 찾아서 Set에서 삭제한다.

In []:



Set의 집합연산 연산자 및 메소드

- 합집합
 - 집합A | 집합B
 - 집합A.union(집합B)
- 교집합
 - 집합A & 집합B
 - 집합A.intersection(집합B)
- 차집합
 - 집합A - 집합B
 - 집합A.difference(집합B)

In []:



자료구조 변환 함수

- list(자료구조)
 - 대상 자료구조/Iterable을 List로 변환한다.
- tuple(자료구조)
 - 대상 자료구조/Iterable을 Tuple로 변환
- set(자료구조)
 - 대상 자료구조/Iterable을 Set으로 변환
 - 다른 자료구조의 원소 중 중복을 빼고 조회할 때 set()를 이용해 Set으로 변환한다.
- Dictionary로 변환하는 함수는 없다.
- 변경할 대상이 Dictionary 일 경우에는 key값들만 모아서 변환한다.

- Iterable
 - 반복가능한 객체. 반복문(for in)을 이용해 일련의 값들을 반복적으로 각각 제공하는 객체를 말한다.
 - 대표적으로 자료구조, 문자열 등이 있다.

In []:



TODO



In []:

```
# 문제 1 ~ 7
jumsu = [100, 90, 100, 80, 70, 100, 80, 90, 95, 85]
# 위 리스트는 학생번호 1번 ~ 10번까지 10명의 시험 점수이다.

#(1) 7번의 점수를 출력하세요

#(2) 1번부터 5번까지의 점수를 출력하세요.

#(3) 4, 5, 6, 7번의 점수를 출력하세요.

#(4) 짝수번째 점수를 출력하세요.

#(5) 홀수번째 점수를 출력하세요.

#(6) 9번의 점수를 20으로 변경하고 전체 출력하세요.

#(7) 중복된 점수는 제거하고 하나씩만 나오도록 출력하세요.

# 문제 8 ~ 9
fruits = ["복숭아", "수박", "딸기"]

#(8) fruits 리스트에 마지막 원소로 "사과", "귤"을 추가하세요.

#(9) fruits 리스트에서 "복숭아"를 제거하세요.

# 문제 10 ~ 15
#(10)본인의 이름, 나이, email주소, 취미, 결혼유무를 사전(딕셔너리)으로 생성.
# 취미는 2개 이상의 값을 넣는다..

#(11) 위 딕셔너리에서 이름과 email주소를 조회해서 출력하세요.

#(12) 위 딕셔너리에서 취미중 두번째 취미를 조회해서 출력하세요.

#(13) 위 딕셔너리에 몸무게와 키 항목을 추가하세요.

#(14) 위 딕셔너리에서 나이를 제거하세요.

#(15) 위 딕셔너리에서 email 주소를 다른 값으로 변경하세요.
```

