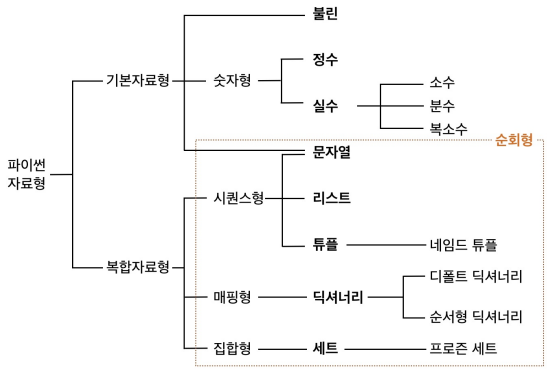

시퀀스 자료형

- 0개 이상의 객체를 배열 형태로 참조하는 자료형을 말한다.
- 따라서 담고 있는 각 객체는 순서를 가지고 있다.

가장 많이 사용하는 시퀀스형

- 문자열(**str**)
- 리스트(**list**)
- 튜플(**tuple**)

리스트(list)

- **가변자료형**(mutable)이다.
  - 생성한 후 내용의 변경이 가능하다. 즉, 담고 있는 객체를 삭제, 변경, 삽입하는 것이 가능하다.
- **순서를 가지는** 0개 이상의 객체를 참조하는 **시퀀스형**이다.
  - 각 객체는 쉼표(,)로 구분한다.
- 리스트 자신을 포함해 어떠한 자료형도 담을 수 있다.
- 유연성 높아 파이썬에서 가장 흔히 사용되는 자료형이다.
- 출력 형식
  - 리스트는 항상 **[ ]**(대괄호) 형태로 출력한다.

리스트 생성

리스트를 만드는 방법으로는 다음 세 가지가 있다.

- **[ ]**(대괄호)
- **list()** 생성자(클래스)
- 리스트 축약(list comprehension)

따라해보기

대괄호 **[ ]**를 사용하여 리스트 생성

- 각 객체는 쉼표(,)로 구분한다.

In [ ]:

```
number_list = [1, 0, -5, -1.23, 27]
```

In [ ]:

```
print(number_list)
```

list() 생성자를 사용하여 리스트 생성

- **list()**의 전달인자로 **순회형**만 올 수 있다.
- 순회형(iterable)이란 담고 있는 객체들에 하나씩 순서대로 접근할 수 있는 자료형으로 문자열, 리스트, 튜플, 딕셔너리, 세트가 있다.

In [ ]:

```
string_list= list('가나다라마바사')
```

In [ ]:

```
print(string_list)
```

list comprehension을 사용하여 리스트 생성

- [**x** for **x** in ***iterabale***]

In [ ]:

```
s = '가나다라마'

[x for x in s]
```

In [ ]:

```
[x for x in range(1,6)]
```

In [ ]:

```
[x**2 for x in range(1,6)]
```

모든 복합자료형은 어떠한 자료형도 담을 수 있다.

- 따라서 리스트는 리스트 자신을 포함해 어떠한 자료형도 담을 수 있다.

In [ ]:

```
complex_list = [list(), number_list, ['x', 3, string_list], True, None]
```

In [ ]:

```
print(complex_list)
```

앞서 만든 리스트의 길이, 즉 각 리스트가 담고 있는 객체의 개수를 확인해보자.

- **len()** 함수는 순회형(**str**, **list**, **tuple**, **dict**, **set**) 객체의 길이(담고 있는 객체의 수)를 반환한다.

In [ ]:

```
# [1, 0, -5, -1.23, 27]
len(number_list)
```

In [ ]:

```
# list('가나다라마바사')
len(string_list)
```

In [ ]:

```
# [list(), number_list, ['x', 3, string_list], True, None]
len(complex_list)
```

문자열은 불변자료형이라 생성한 후 내용의 변경이 불가능하다.

In [ ]:

```
s = 'abcde'
s[0] = 'A'  # a를 A로 바꾼다.
```

리스트는 가변자료형이라 생성한 후 내용의 변경이 가능하다.

In [ ]:

```
L = ['a', 'b', 'c', 'd', 'e']
L[0] = 'A'  # 'a'를 'A'로 바꾼다.

print(L)
```

리스트 인덱스

문자열 인덱스와 같은 원리로 작동한다.

In [ ]:

```
band = ['보컬', ['기타1', '기타2', '베이스'], '드럼', '키보드']
```

In [ ]:

```
# 첫 번째 객체를 추출한다.
band[0]
```

In [ ]:

```
# --- ['보컬', ['기타1', '기타2', '베이스'], '드럼', '키보드']
# 마지막 객체를 추출한다.
band[-1]
```

In [ ]:

```
# --- ['보컬', ['기타1', '기타2', '베이스'], '드럼', '키보드']
# 두 번째 객체를 추출한다.
band[1]
```

In [ ]:

```
# --- ['보컬', ['기타1', '기타2', '베이스'], '드럼', '키보드']
# 두 번째 객체의 두 번째 객체를 추출한다.
band[1][1]
```

In [ ]:

```
# --- ['보컬', ['기타1', '기타2', '베이스'], '드럼', '키보드']
# 두 번째 객체의 마지막 객체(문자열)의 첫 번째 문자를 추출한다.
band[1][-1][0]
```

리스트 범위 밖의 객체를 지정해서는 안 된다.

In [ ]:

```
# --- ['보컬', ['기타1', '기타2', '베이스'], '드럼', '키보드']
band[99]
```

기존 객체를 새로운 객체로 바꿀 수도 있는데 교체하고자 하는 자리의 인덱스 번호를 지정한 후 새로운 객체를 할당하면 된다.

교체하는 형식은 다음과 같다.

리스트[인덱스번호] = 새로운-객체

In [ ]:

```
# --- ['보컬', ['기타1', '기타2', '베이스'], '드럼', '키보드']
# 리스트의 세 번째 객체를 교체한다.
band[2] = '카혼'

print(band)
```

리스트 안의 리스트에 있는 객체도 바꿀 수 있다.

리스트 관련 연산자

리스트와 관련된 연산자로는 다음과 같은 것이 있다.

결합 연산자 : **+**
반복 결합 연산자 : *****
확장 연산자 : **+=**

분할 연산자 : **[:]**
삭제 연산자 : **del**
멤버십 연산자 : **in/not in**

결합 연산자

In [ ]:

```
[1, 2, 3] + [4, 5, 6]
```

In [ ]:

```
['a', 'b'] + ['c', 'd']
```

반복 연산자

In [ ]:

```
[0] * 5
```

In [ ]:

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

확장 연산자

우변의 객체를 병합해서 좌변의 리스트를 확장한다.

- **+=**의 우변에는 순회형이 와야 한다.
- **extend(순회형)** 메소드도 같은 기능을 수행한다.

In [ ]:

```
list1 = ['프로도', '네오']
list2 = ['라이언', '콘']
list1 += list2

print(list1)
```

In [ ]:

```
# --- ['프로도', '네오', '라이언', '콘']
# 좌변 리스트에 정수 307을 추가한다.
list1 += 307
```

왜 오류가 나는 것일까?

+=의 우변에는 순회형이 와야 하는데 정수형이 왔기 때문에 오류가 난다.

어떻게 하면 307을 좌변의 리스트에 넣을 수 있을까?

In [ ]:

```
# 우변에 순회형만 올 수 있기 때문에 리스트 안에 추가한 후 좌변으로 병합하면 된다.
list1 += [307]

print(list1)
```

In [ ]:

```
list1+= '제이지'

print(list1)
```

어떻게 된 일인가?

+=의 우변에 순회형 중 문자열이 왔기 때문에 문자열의 문자를 하나씩 꺼내어 좌변 리스트에 추가하게 된다.

어떻게 하면 '제이지'를 하나의 문자열로 좌변의 리스트에 넣을 수 있을까?

In [ ]:

```
# --- ['프로도', '네오', '라이언', '콘', 307]
# 좌변 리스트에 문자열 '제이지'를 추가하려면
# 먼저 '제이지'를 리스트의 객체로 만든 후 좌변으로 병합하면 된다.
list1 += ['제이지']

print(list1)
```

분할 연산자

분할 연산자 **[:]**와 **[: :]**는 리스트 분할을 통해 리스트의 객체를 추출한다.

- 앞서 배운 문자열 분할 연산자와 같은 기능을 한다.

In [ ]:

```
s = ['a', 'b', 'c', 'd', ['x', 'y', 'z'], 'e', 'f']
```

In [ ]:

```
# 전체를 추출한다. s[0:len(s)]와 같다.
s[:]
```

In [ ]:

```
# --- ['a', 'b', 'c', 'd', ['x', 'y', 'z'], 'e', 'f']
# 처음 3개를 추출한다.
s[:3]
```

In [ ]:

```
# --- ['a', 'b', 'c', 'd', ['x', 'y', 'z'], 'e', 'f']
# 마지막 3개를 추출한다.
s[-3:]
```

In [ ]:

```
# --- ['a', 'b', 'c', 'd', ['x', 'y', 'z'], 'e', 'f']
# 3~4번째 객체 'c'와 'd'를 '가'과 '나'로 교체한다.
s[2:4] = ['가', '나']

print(s)
```

In [ ]:

```
# --- ['a', 'b', '가', '나', ['x', 'y', 'z'], 'e', 'f']
# 두 번째 객체 'b'를 'l', 'm', 'n'으로 교체한다.
# 이때, 분할 연산자를 사용해서 교체한다.
s[1:2] = ['l', 'm', 'n']

print(s)
```

In [ ]:

```
# --- ['a', 'l', 'm', 'n', '가', '나', ['x', 'y', 'z'], 'e', 'f']
# 두 번째 객체 'l'을 ['p', 'q']로 교체한다.
# 이때, 인덱스를 사용해서 교체한다.
s[1] = ['p', 'q']

print(s)
```

In [ ]:

```
# --- ['a', ['p', 'q'], 'm', 'n', '가', '나', ['x', 'y', 'z'], 'e', 'f']
# 두 번째 객체부터 마지막 객체 앞까지를 '다'와 '라'로 교체한다.
s[1:-1] = ['다', '라']

print(s)
```

In [ ]:

```
# --- ['a', '다', '라', 'f']
# 리스트의 모든 객체를 빈 리스트로 교체한다.
s[:] = []

print(s)
```

삭제 연산자

del은 객체의 참조를 삭제하는 연산자다.

- 따라서 리스트 객체뿐만 아니라 파이썬에서 사용하는 모든 자료형에서 객체 참조를 삭제하는 데 사용할 수 있다.

형식은 다음과 같다.

- **del *객체***

In [ ]:

```
L = ['a', 'b', 'c', 'd', ['x', 'y', 'z'], 'e', 'f']
```

In [ ]:

```
# 다섯 번째 객체의 마지막 객체를 삭제한다.
del L[4][-1]
print(L)
```

In [ ]:

```
# --- ['a', 'b', 'c', 'd', ['x', 'y'], 'e', 'f']
# 다섯 번째 객체를 삭제한다.
del L[4]
print(L)
```

In [ ]:

```
# --- ['a', 'b', 'c', 'd', 'e', 'f']
# 두 번째와 세 번째 객체를 삭제한다.
del L[1:3]
print(L)
```

In [ ]:

```
# --- ['a', 'd', 'e', 'f']
# 리스트의 모든 객체를 삭제한다.
del L[:]
print(L)
```

멤버십 연산자

멤버십 연산은 순회형 값 안에 객체가 존재하는지 여부를 '참(**True**)' 또는 '거짓(**False**)'으로 알려준다.

형식은 다음과 같다.

- ***객체* in *순회형***
- ***객체* not in *순회형***

In [ ]:
```
L = ['a', 'b', ['c', 'd', 'e']]
```

In [ ]:
```
'a' in L
```

In [ ]:
```
# --- ['a', 'b', ['c', 'd', 'e']]
'c' in L
```

In [ ]:
```
# ['a', 'b', ['c', 'd', 'e']]
'c' in L[-1]
```

In [ ]:
```
# ['a', 'b', ['c', 'd', 'e']]
'b' not in L
```

In [ ]:
```
# ['a', 'b', ['c', 'd', 'e']]
'f' not in L
```

참고로 멤버십 연산자는 모든 ***순회형***에 적용할 수 있기 때문에 문자열에 일부 문자가 포함되어 있는지 확인할 때도 사용할 수 있다.

In [ ]:
```
text = 'Don’t just play on your phone, program it'
```

In [ ]:
```
'j' in text
```

In [ ]:
```
# --- 'Don’t just play on your phone, program it'
'd' in text
```

In [ ]:
```
# --- 'Don’t just play on your phone, program it'
'program' in text
```

리스트 관련 메소드

복사 메소드 : **copy()**
추가 메소드 : **append()**, **extend()**, **insert()**
삭제 메소드 : **pop()**, **remove()**
질의 메소드 : **count()**, **index()**
정렬 메소드 : **reverse()**, **sort()**

복사 메소드

L.**copy()**

- 리스트 *L*의 얕은 복사(shallow copy)를 반환한다.

In [ ]:

```
x = [1, 2, 3, ['x', 'y', 'z']]

# 리스트 x를 얕은 복사해서 변수 y에 할당한다.
y = x.copy()
print(y)
```

추가 메소드

L.**append**(***x***)

- 리스트 *L*의 마지막에 객체 ***x***를 추가한다.

In [ ]:

```
L = []            # 빈 리스트를 초기화 한다.
L.append('a')     # 'a'를 리스트에 추가한다.
L.append('b')     # 'b'를 리스트에 추가한다.
print(L)
```

In [ ]:

```
# --- ['a', 'b']
# ['d', 'e']를 리스트에 추가한다.
L.append(['d', 'e'])
print(L)
```

L.**extend**(***m***)

- 순회형 ***m***의 모든 객체를 리스트 *L*의 끝에 추가한다.
- *L* **+=** ***m***과 같다.

In [ ]:

```
# --- ['a', 'b', ['d', 'e']]
# ['f', 'g']를 리스트에 확장해서 추가한다.
L.extend(['f', 'g'])
print(L)
```

L.**insert**(***i***, ***x***)

- 리스트 *L*의 인덱스 ***i***에 객체 ***x***를 삽입한다.

In [ ]:

```
# --- ['a', 'b', ['d', 'e'], 'f', 'g']
# 리스트의 다섯 번째 위치에 'b'를 추가한다.
L.insert(4, 'b')
print(L)
```

삭제 메소드

L.**pop()** : 리스트 *L*의 가장 오른쪽에 있는 객체(마지막 객체)를 반환한 후 삭제한다.

- 빈 리스트거나 설정한 인덱스 번호가 리스트 범위 밖이면 **IndexError**가 발생한다.

In [ ]:

```
# --- ['a', 'b', ['d', 'e'], 'f', 'b', 'g']
# 리스트의 가장 오른쪽에 있는 마지막 객체를 반환한 후 삭제한다.
L.pop()
print(L)
```

L.**pop**(***i***) : 리스트 *L*의 인덱스 ***i***에 있는 객체를 반환한 후 삭제한다.

- 빈 리스트거나 설정한 인덱스 번호가 리스트 범위 밖이면 **IndexError**가 발생한다.

In [ ]:

```
# --- ['a', 'b', ['d', 'e'], 'f', 'b']
# 리스트의 세 번째 객체를 반환한 후 삭제한다.
L.pop(2)
print(L)
```

In [ ]:

```
# --- ['a', 'b', 'f', 'b']
# 리스트의 다섯 번때 객체를 반환한 후 삭제한다.
# 하지만 다섯 번째 객체는 존재하지 않기 때문에 오류가 난다.
L.pop(4)
```

L.**remove**(***x***)

- 리스트 *L*의 **가장 왼쪽**에 있는 객체(첫 번째 객체) ***x***를 삭제한다.
- 객체 ***x***가 존재하지 않으면 **ValueError**가 발생한다.

In [ ]:

```
# --- ['a', 'b', 'f', 'b']
# 리스트의 첫 번째 나타나는 'b'를 삭제한다.
# [주의] 삭제하는 객체를 반환하지는 않는다.
L.remove('b')
print(L)
```

In [ ]:

```
# --- ['a', 'f', 'b']
# 리스트의 첫 번째 'c'를 삭제한다.
# 하지만 'c'는 리스트에 존재하지 않기 때문에 오류가 난다.
L.remove('c')
```

질의 메소드

L.**count**(***x***)

- 리스트 *L*에서 객체 ***x***의 개수를 반환한다.

In [ ]:

```
L = ['a', 'b', ['c', 'd', 'e'], 'f', 'b', 'c', 'g']
```

In [ ]:

```
# 'b'가 리스트에 몇 개 들어 있는지 확인한다.
L.count('b')
```

In [ ]:

```
# --- ['a', 'b', ['c', 'd', 'e'], 'f', 'b', 'c', 'g']
# 'c'가 리스트에 몇 개 들어 있는지 확인한다.
L.count('c')
```

L.**index**(***x, 시작번호, 끝번호***)

- 리스트 *L*에 있는 객체 ***x*** 중 **가장 왼쪽**에 있는 객체 ***x***의 인덱스를 반환한다.
- 리스트 *L*의 ***시작번호***와 ***끝번호*** 인덱스가 정해지면 인덱스 범위 안에 있는 객체 ***x*** 중 **가장 왼쪽**에 있는 객체 ***x***의 인덱스를 반환한다.
- 분할 연산자처럼 ***시작번호***부터 ***끝번호* 바로 앞**까지만 검색한다.
- 객체 ***x***가 없으면 **ValueError**가 발생한다.

In [ ]:

```
# --- ['a', 'b', ['c', 'd', 'e'], 'f', 'b', 'c', 'g']
# 리스트에서 가장 먼저 나타나는 'b'의 인덱스 번호를 확인한다.
L.index('b')
```

In [ ]:

```
# --- ['a', 'b', ['c', 'd', 'e'], 'f', 'b', 'c', 'g']
# 리스트의 네 번째부터 검색해서
# 가장 먼저 나타나는 'b'의 인덱스 번호를 확인한다.
L.index('b', 3)
```

In [ ]:

```
# --- ['a', 'b', ['c', 'd', 'e'], 'f', 'b', 'c', 'g']
# 리스트의 네 번째부터 여섯 번째 사이에서
# 가장 먼저 나타나는 'a'의 인덱스 번호를 확인한다.
# 'a'가 리스트의 지정한 범위 안에는 없기 때문에 오류가 난다.
L.index('a', 3, 6)
```

정렬 메소드

L.**reverse**()

- 리스트 *L*에 있는 객체들의 순서를 거꾸로 뒤집는다.

In [ ]:

```
# --- ['a', 'b', ['c', 'd', 'e'], 'f', 'b', 'c', 'g']
# 리스트에 들어 있는 객체들의 순서를 거꾸로 정렬한다.
print(L)
L.reverse()
print(L)
```

L.**sort**(***key=None, reverse=False***)

- 리스트 *L*을 정렬한다.
- 기본값으로 오름차순 정렬한다.

In [ ]:

```
# --- ['g', 'c', 'b', 'f', ['c', 'd', 'e'], 'b', 'a']
# 리스트에 들어 있는 객체들을 오름차순 정렬한다.
L.sort()
```

왜 오류가 날까?

정렬하는 모든 객체를 비교할 수 있어야 한다.

따라서, 만약 *L*이 담고 있는 객체 중 **비교가 불가능한 객체**가 있어 정렬을 할 수 없으면 **TypeError**가 발생한다.

In [ ]:

```
# --- ['g', 'c', 'b', 'f', ['c', 'd', 'e'], 'b', 'a']
# 정렬을 가능하게 하기 위해 다섯 번째 객체인 리스트를 삭제하여 문자열만 리스트에 포함되도록 한다.
L.pop(4)
print(L)
```

이제 리스트에는 비교가 가능한 객체들만 있기 때문에 정렬을 할 수 있다.

In [ ]:

```
# --- ['b', 'c', 'f', 'g', 'b', 'a']
# 'M'을 리스트의 세 번째 위치에 추가한다.
L.insert(2, 'M')
print(L)
```

In [ ]:

```
# --- ['b', 'c', 'M', 'f', 'g', 'b', 'a']
# 리스트를 오름차순으로 정렬한다.
print(L)
L.sort()
print(L)
```

왜 대문자 'M'이 정렬한 리스트의 맨 앞에 있을까?

Unicode에서 영어 대문자는 영어 소문자보다 앞 쪽에 위치하기 때문에 오름차순으로 정렬을 하면 대문자가 소문자보다 먼저 온다.

그려면 대소문자 구분없이 정렬할 수는 없을까?

sort() 메소드는 키워드 전달인자 ***key***와 ***reverse***를 선택적으로 사용할 수 있다.

key의 전달인자로는 함수가 온다.

- 전달인자로 함수의 이름만 오고 함수 뒤에 오는 소괄호 **()**는 생략한다.
- 이 함수를 사용해서 원하는 방식으로 리스트 *L*을 정렬할 수 있다.

In [ ]:

```
# --- ['M', 'a', 'b', 'b', 'c', 'f', 'g']
# 대소문자 구분없이 정렬하기 위해 모든 문자열으로 소문자로 바꾼 후 정렬한다.
L.sort(key=str.lower)
```

In [ ]:

```
print(L)
```

reverse의 기본값은 '거짓(**False**)'이라 오름차순이 기본 정렬 방식이다.

하지만 전달인자를 '참(**True**)'으로하면 정렬이 내림차순으로 이루어진다.

In [ ]:

```
# --- ['a', 'b', 'b', 'c', 'f', 'g', 'M']
# reverse=True로 내림차순 정렬을 했기 때문에 소문자가 대문자보다 먼저 온다.
L.sort(reverse=True)
print(L)
```

In [ ]:

```
# --- ['g', 'f', 'c', 'b', 'b', 'a', 'M']
# 대소문자 구분없이 내림차순으로 정렬한다.
L.sort(key=str.lower, reverse=True)
print(L)
```

정렬 방법: sort() 메소드 vs. sorted() 함수

리스트의 **sort()** 매소드는 리스트 자체를 정렬한다.

따라서, 원본 리스트를 정렬하지 않으려면, **sorted()** 함수를 사용해서 새로운 리스트 객체를 생성하는 것이 좋다.

In [ ]:

```
x = ['드럼', '기타', '키보드', '베이스']
```

In [ ]:

```
# 리스트 x를 오름차순으로 정렬해서 변수 y에 할당한다.
y = sorted(x)
print(y)
```

In [ ]:

```
# x는 바뀌지 않았다.
print(x)
```

In [ ]:

```
# --- x = ['드럼', '기타', '키보드', '베이스']
# 리스트 x 자체를 오름차순으로 정렬한다.
x.sort()
```

In [ ]:

```
# x 자체가 바뀌었다.
print(x)
```

기본적으로 **sort()** 메소드와 **sorted()** 함수는 값이 작은 것부터 큰 것으로, 즉 **오름차순**으로 담고 있는 객체를 정렬한다.

만약 반대로 가장 값이 큰 것부터 작은 순, 즉 **내림차순**으로 정렬을 원하면, 매개변수 ***reverse***에 **True** 전달인자를 사용하면 된다.

또한 기본 정렬 규칙으로 정렬하지 않고 다른 정렬 규칙으로 객체들을 정렬하고자 한다면 ***key*** 매개변수에 정렬 규칙을 구현한 함수(사용자 정의 함수 포함)를 전달인자로 사용하면 된다.

- 사용자 정의 함수는 주로 람다 함수를 많이 사용한다.

In [ ]:

```
students = [
    ['John', 'A', 15],
    ['Jane', 'B', 17],
    ['Dave', 'B', 16]
]

students.sort(key=lambda s: s[2])
students
```

list comprehension

기존 리스트를 기반으로 새로운 리스트를 생성하는 기능

for i **in** ***iterable***:\ **if** ***conditional***:\ ***expression***

new_list = **[** ***expression*** i **in** ***iterable*** **if** ***conditional*** **]**

In [ ]:

```
# 1부터 20 사이의 짝수들을 리스트로 생성
list1 = [x*2 for x in range(1, 11)]
list2 = [x for x in range(1,21) if x % 2 == 0]

print(list1)
print(list2)

# 36의 약수들을 리스트로 생성
list3 = [x for x in range(1, 36) if 36 % x == 0]
print(list3)
```

In [ ]:

```
# 2차원 매트릭스를 1차원화
matrix =[
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12]
]

list4 = []
for row in matrix:
    for e in row:
        list4.append(e)

list5 = [e for row in matrix for e in row]

print(list4)
print(list5)
```