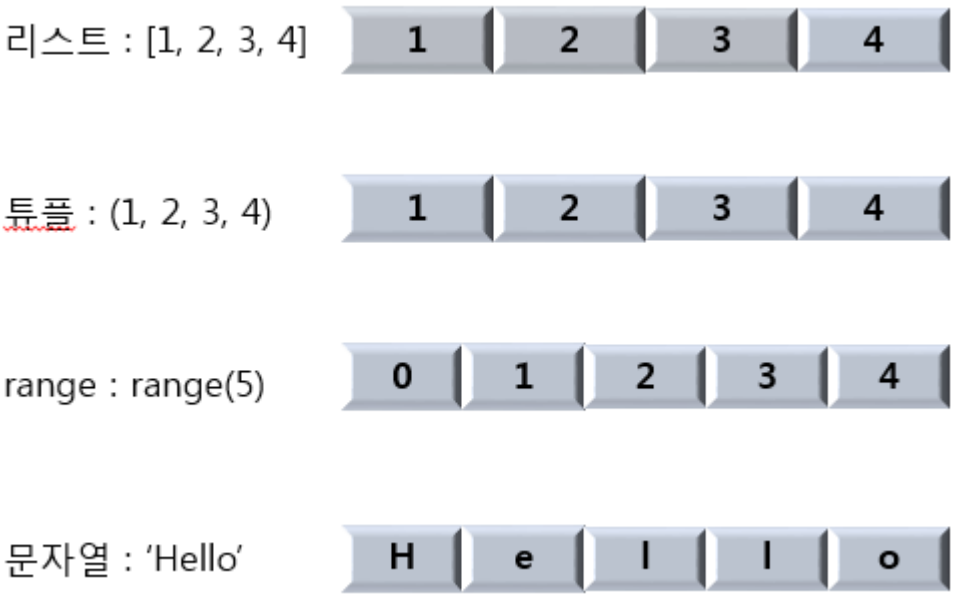
Day 4 - 시퀀스 자료형

시퀀스는 데이터의 순서 있는 나열을 의미합니다. 주로 문자열, 리스트, 튜플 등이 시퀀스 자료형에 속합니다.



Python 시퀀스 자료형

시퀀스 소개

파이썬 표준 라이브러리는 C로 구현된 다음과 같은 시퀀스 자료형을 제공합니다.

- **컨테이너 시퀀스 (Container sequence)**
  - 객체에 대한 참조를 담고 있습니다
  - 어떤 자료형이든 담을 수 있습니다
  - list, tuple, collection.deque
- **플랫 시퀀스 (Flat sequence)**
  - 단 하나의 자료형만 담을 수 있습니다
  - 자신의 메모리 공간에 각 항목의 값을 직접 담습니다.
  - str, bytes, bytearray, memoryview, array.array

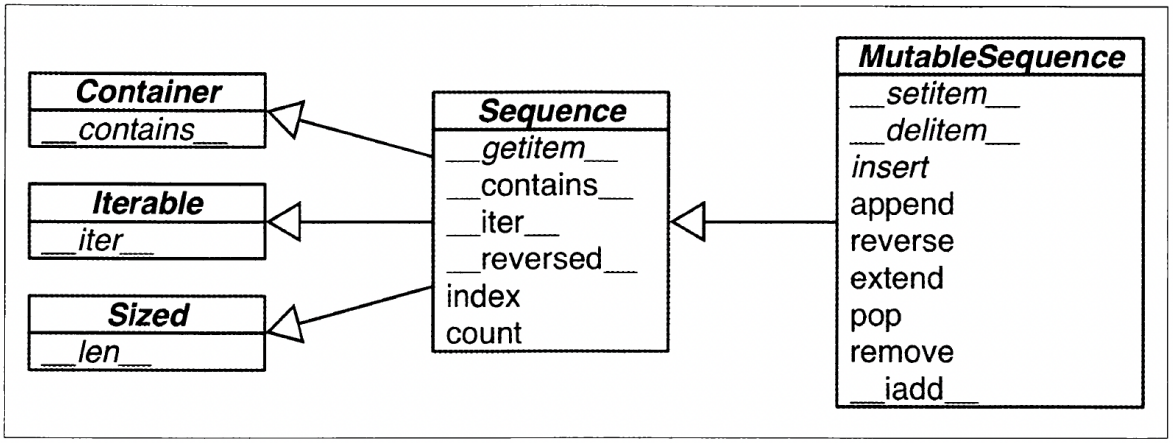
시퀀스는 다음과 같이 가변성(mutability)에 따라 분류할 수도 있습니다

- **가변 시퀀스 (mutable sequence)**
  - list, bytearray, array.array, collections.deque, memoryview
- **불변 시퀀스 (immutable sequences)**
  - tuple, str, bytes

아래 그림은 파이썬이 어떤 식으로 자료형에 대한 기능을 구현하는지 알려주고 있습니다.

가변 시퀀스는 불변 자료형의 모든 기능을 구현하고, 추가로 insert 같은 메소드를 구현하고 있습니다.

그림 2-1 collections.abc의 일부 클래스에 대한 UML 다이어그램. 슈퍼클래스는 왼쪽에 있으며, 상속 관계를 나타내는 화살표는 서브클래스에서 슈퍼클래스를 향한다. 이탤릭체로 표시된 이름은 추상 클래스와 추상 메서드를 나타낸다.



실제로 시퀀스가 `collections.abc`를 통해 구현되는 것은 아니지만, 어떤 기능을 제공할지 예측하는데 도움이 됩니다. (abc는 파이썬에서 제공하는 추상화 클래스 라이브러리인데, 당장 알 필요는 없습니다)

시퀀스 기능

여기서는 세 가지 시퀀스 자료형으로 공통 기능에 대해 알아볼 것입니다

- 문자열 : `Hello`
- 리스트 : `[1, 2, 3]`

- 튜플 : `('GOOD', 100, 10.1)`

모든 시퀀스는 순서가 유지되고, 정수로 인덱싱하고, 길이가 있습니다.

```
# 변수를 초기화 합니다
a = 'Hello'
b = [1,2,3]
c = ('GOOD', 100, 10.1)

# 정수로 인덱싱 가능합니다
a[0]    # 'H'
b[-1]   # 3
c[2]    # 10.1

# 길이
len(a)   # 5
len(b)   # 3
len(c)   # 3
```

시퀀스에 덧셈, 곱셈 연산자를 사용할 수 있습니다

```
# 곱셈
a = 'Hello'
a * 3     # 'HelloHelloHello'

b = [1,2,3]
b * 3     # [1,2,3,1,2,3,1,2,3]

# 덧셈
a = (1,2,3)
b = (4,5)
c = [6,7]

a + b    # (1,2,3,4,5)
```

```
# 같은 자료형끼리만 붙일 수 있습니다
a + c    # Error
```

단, 이때 시퀀스의 성격에 따라 결과가 조금 다를 수 있습니다.

```
# 곱셈
a = [1, 2, 3] # list는 가변 자료형입니다
b = (1, 2, 3) # tuple는 불변 자료형입니다

# id를 확인해봅시다
id(a)  # 139743876029952
id(b)  # 139743894783488

# 곱셈 연산자를 사용해봅니다
a *= 2
b *= 2

# id를 다시 확인
id(a) # 139743876029952
id(b) # 139743912383424  -> 새로운 객체가 생겼다
```

슬라이싱(Slicing)

시퀀스의 일부(subsequence)를 취하는 것을 슬라이싱이라고 합니다.

슬라이싱을 진행하면 새로운 객체를 생성하게 됩니다.

```
a[start:end:step]
```

다음 예제를 확인해봅시다.

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

a[2:5] # [2, 3, 4]
```

```
a[-5:] # [6, 7, 8, 9, 0]
a[::2] # [1, 3, 5, 7, 9]
```

- start, end, step은 모두 정수입니다
- step은 생략 가능합니다
- 슬라이스는 end 값을 포함하지 않습니다
- 값을 생략하면, 각각 시작과 끝을 기본값으로 사용합니다

슬라이싱 재할당(re-assignment)

다시 할당하거나 삭제할 수 있습니다

```
# 재할당
a = list(range(10))
a[2:4] = [99]       # [0,1,99,4,5,6,7,8]

# 오른쪽에도 반복 가능한 객체가 와야 합니다
a[1:3] = 100        # Error
```

일부분을 삭제할 수 있습니다

```
# 삭제
a = list(range(10))
del a[2:4]                # [0,1,4,5,6,7,8,9,0]
```

시퀀스 순회(iteration)

for 루프를 통해 시퀀스 자료형 내부의 요소들을 순회할 수 있습니다

```
>> s = [1, 4, 9, 16]
>> for i in s:
...     print(i)
...
1
```

```
4
9
16
```

`enumerate()` 함수를 사용하면 카운터 값을 같이 가져올 수 있습니다

```
>> names = ['Elwood', 'Jake', 'Curtis']
>> for i, name in enumerate(names):
...     print(f'idx = {i}, name = {name}')

idx = 0, name = Elwood
idx = 1, name = Jake
idx = 2, name = Curtis
```

여러개의 이터레이터 변수를 사용해서 루프를 수행할 수도 있다

```
>> a = [[1,2],[3,4],[4,5]]
>> for x, y in a:
..     print(f'x: {x}, y:{y}')

a: 1, b:2
a: 3, b:4
a: 4, b:5
```

zip() 함수 사용하기

여러개의 시퀀스를 결합해 이터레이터를 만들 수도 있습니다.

`zip()` 내장함수는 여러 개의 순회 가능한(iterable) 객체를 인자로 받고, 각 객체가 담고 있는 원소를 튜플의 형태로 차례로 접근할 수 있는 반복자(iterator)를 반환합니다.

```
>> a = list(range(10))
>> b = tuple(range(10))
>> for pair in zip(a,b): # 추가로, list(zip(a,b)) 로 사용하면 lis
..     print(pair)
```

```
(0, 0)
(1, 1)
(2, 2)
(3, 3)
(4, 4)
(5, 5)
(6, 6)
(7, 7)
(8, 8)
(9, 9)

# 여러개의 변수로 사용해 튜플을 언팩할수도 있습니다
>> for x,y in zip(a,b):
>>     print(f'x: {x}, y: {y}')

x: 0, y: 0
x: 1, y: 1
x: 2, y: 2
x: 3, y: 3
x: 4, y: 4
x: 5, y: 5
x: 6, y: 6
x: 7, y: 7
x: 8, y: 8
x: 9, y: 9
```