Ch4. 리스트와 튜플

- **리스트와 튜플 사용하기**
  - 지금까지 변수에는 값을 한 개씩만 저장함

```
a = 10
b = 20
```

  - 값을 30개 저장하려면 다음과 같이 변수 30개에 값 30개를 저장하면 됨

```
a1 = 10
a2 = 20
# ... (생략)
a29 = 60
a30 = 40
```

  - 변수 30개를 일일이 타이핑하기는 쉽지 않으니 리스트를 사용하면 편리
  - **리스트는 말 그대로 목록이라는 뜻이며 값을 일렬로 늘어놓은 형태**

    (보통 리스트의 값은 코드로 생성하는 경우가 많아서 타이핑할 일이 거의 없음)
  - 유닛부터 리스트를 만드는 방법과 기본 사용 방법을 알아보자

- **리스트 만들기**
  - 변수에 값을 저장할 때 [ ](대괄호)로 묶어주면 리스트가 되며
    각 값은 ,(콤마)로 구분해줌

    • 리스트 = [값, 값, 값]
  - 숫자가 5개 들어있는 리스트

```
>>> a = [38, 21, 53, 62, 19]
>>> a
[38, 21, 53, 62, 19]
```

  - a = [38, 21, 53, 62, 19]와 같이 변수에 [ ]로 값을 저장하여 리스트 생성
  - 리스트에 저장된 각 값 요소(element)라고 부름

- **리스트에 여러 가지 자료형 저장하기**
  - 리스트는 문자열, 정수, 실수, 불 등 모든 자료형을 저장할 수 있으며 자료형을 섞어서 저장해도 됨

```
>>> person = ['james', 17, 175.3, True]
>>> person
['james', 17, 175.3, True]
```

  - 리스트에 여러 가지 자료형을 사용하면 관련 정보를 하나로 묶기 좋음

- **빈 리스트 만들기**
  - 빈 리스트를 만들 때는 [ ]만 지정하거나 list를 사용하면 됨

```
• 리스트 = []
• 리스트 = list()
```

```
>>> a = []
>>> a
[]
>>> b = list()
>>> b
[]
```

  - **빈 리스트는 쓸모가 없을 것 같지만, 보통 빈 리스트를 만들어 놓은 뒤에 새 값을 추가하는 방식으로 사용함**

- **range를 사용하여 리스트 만들기**
  - range는 연속된 숫자를 생성하는데 range에 10을 지정하면 0부터 9까지 숫자를 생성함
  - 지정한 횟수 숫자는 생성되는 숫자에 포함되지 않음

• range(횟수)

```
>>> range(10)
range(0, 10)
```

  - range(0, 10)이라고 나와서 10까지 생성될 것 같지만 10은 포함되지 않음
  - **list에 range(10)을 넣어보면 0부터 9까지 들어있는 리스트가 생성**

• 리스트 = list(range(횟수))

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

- **range를 사용하여 리스트 만들기**
  - range는 시작하는 숫자와 끝나는 숫자를 지정할 수도 있음
  - 이때도 끝나는 숫자는 생성되는 숫자에 포함되지 않음
  - **list에 range(5, 12)를 넣으면 5부터 11까지 들어있는 리스트가 생성**

• 리스트 = list(range(시작, 끝))

```
>>> b = list(range(5, 12))
>>> b
[5, 6, 7, 8, 9, 10, 11]
```

- **range를 사용하여 리스트 만들기**
  - **증가폭을 사용하는 방법**
  - **range에 증가폭을 지정하면 해당 값만큼 증가하면서 숫자를 생성**
  - **range(-4, 10, 2)는 -4부터 8까지 2씩 증가함**
  - 여기서 끝나는 값은 10이므로 10까지 증가하지 않고 8까지 생성
  - 증가폭을 음수로 지정하면 해당 값만큼 숫자가 감소함

• 리스트 = list(range(시작, 끝, 증가폭))

```
>>> c = list(range(-4, 10, 2))
>>> c
[-4, -2, 0, 2, 4, 6, 8]
```

  - **range(10, 0, -1)은 10부터 1씩 감소하며 0은 포함되지 않으므로 1까지 생성**

```
>>> d = list(range(10, 0, -1))
>>> d
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

- **튜플 사용하기**
  - **튜플은 리스트처럼 요소를 일렬로 저장하지만, 안에 저장된 요소를 변경, 추가, 삭제를 할 수 없음**
  - 간단하게 **읽기 전용 리스트**라고 할 수 있음
  - 변수에 값을 저장할 때 ( )(괄호)로 묶어주면 튜플이 되며 각 값은 ,(콤마)로 구분
  - 묶지 않고 값만 콤마로 구분해도 튜플이 됨

```
• 튜플 = (값, 값, 값)
• 튜플 = 값, 값, 값
```

- **튜플 사용하기**
  - a = (38, 21, 53, 62, 19)와 같이 값을 괄호로 묶은 뒤 변수에 저장하여 튜플 생성
  - a = 38, 21, 53, 62, 19와 같이 괄호를 사용하지 않아도 튜플을 만들 수 있음

```
>>> a = 38, 21, 53, 62, 19
>>> a
(38, 21, 53, 62, 19)
```

  - 튜플도 리스트처럼 여러 자료형을 섞어서 저장해도 됨
  - 저장된 요소를 변경, 추가, 삭제할 수도 없는 튜플을 만든 이유는 파이썬 프로그래밍에서 튜플을 사용하는 쪽이 더 유리한 경우도 있기 때문임
  - **튜플은 요소가 절대 변경되지 않고 유지되어야 할 때 사용함**
  - **튜플을 만든 상태에서 요소를 변경하게 되면 에러가 발생하게 됨**
  - 요소를 실수로 변경하는 상황을 방지할 수 있음
  - 요소를 자주 변경해야 할 때는 리스트를 사용함
  - **실무에서는 요소를 변경하는 경우가 많기 때문에 튜플보다 리스트를 더 자주 사용하는 편임**

- 튜플 과 리스트 차이는?
  - 사전적 의미는 Tuple과 List가 비슷
  - 리스트는 “목록”
  - 튜플은 “N개의 요소로 된 집합”

- 파이썬의 List와 Tuple의 차이
  - List는 데이터 변경 가능(리스트 생성 후 추가/수정/삭제 가능)
  - Tuple은 데이터 변경 불가능(튜플 생성 후 추가/수정/삭제 불가능)
  - List는 이름 그대로 목록 형식의 데이터를 다루는 데 적합
  - Tuple은 위경도 좌표나 RGB 색상처럼 작은 규모의 자료구조를 구성하기에 적합

- 변경이 불가능한 자료형이 필요한 이유?
  - 성능 : 변경 가능한 자료형과는 달리 데이터를 할당할 공간의 내용이나 크기가 달라지지 않기 때문에 생성 과정이 간단
  - 데이터가 오염되지 않을 것이라는 보장이 있기 때문에 복사본을 만드는 대신 그냥 원본을 사용
  - 신뢰 가능한 코드 : 변경되지 않아야 할 데이터를 오염시키는 버그를 만들 가능성 제거
  - 코드를 설계할 때부터 변경이 가능한 데이터와 그렇지 않은 데이터를 정리해서 코드에 반영

- 문자열도 변경이 불가능한 자료형

- **요소가 한 개 들어있는 튜플 만들기**
  - 요소가 1개 들어있는 튜플은 괄호로 묶으면 튜플이 아니라 그냥 값이 됨

```
>>> (38)
38
```

  - **요소가 한 개인 튜플을 만들 때는 ( )(괄호) 안에 값 한 개를 넣고 ,(콤마)를 붙임**
  - 괄호로 묶지 않고 값 한 개에 ,를 붙여도 됨

- 튜플 = (값,)
- 튜플 = 값,

```
>>> (38, )
(38,)
>>> 38,
(38,)
```

  - 튜플은 요소를 변경, 추가, 삭제할 수도 없는데 값 한 개짜리 튜플은 함수(클래스)를 사용하다 보면 값이 아닌 튜플을 넣어야 할 경우가 생김
  - 이때 값은 한 개지만 튜플을 넣어야 할 때 (값,)과 같은 형식을 사용해야 함

- **range를 사용하여 튜플 만들기**
  - range를 사용하여 튜플을 만드는 방법
  - **tuple 안에 range를 넣으면 튜플이 생성**

• 튜플 = tuple(range(횟수))

```
>>> a = tuple(range(10))
>>> a
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

  - 5부터 11까지 들어있는 튜플 생성

• 튜플 = tuple(range(시작, 끝))

```
>>> b = tuple(range(5, 12))
>>> b
(5, 6, 7, 8, 9, 10, 11)
```

- **range를 사용하여 튜플 만들기**
  - range에 증가폭을 지정하는 방법도 가능함

• 튜플 = tuple(range(시작, 끝, 증가폭))

```
>>> c = tuple(range(-4, 10, 2))
>>> c
(-4, -2, 0, 2, 4, 6, 8)
```

  - range(-4, 10, 2)는 -4부터 2씩 증가하며 10은 포함되지 않으므로 8까지 들어있는 튜플을 생성

- **튜플을 리스트로 만들고 리스트를 튜플로 만들기**
  - **튜플과 리스트는 요소를 변경, 추가, 삭제할 수 있는지 없는지만 다를 뿐 기능과 형태는 같음**
  - 튜플을 리스트로 만들거나 리스트를 튜플로 만들 수도 있음
  - tuple 안에 리스트를 넣으면 새 튜플이 생김

```
>>> a = [1, 2, 3]
>>> tuple(a)
(1, 2, 3)
```

  - 반대로 list 안에 튜플을 넣으면 새 리스트가 생성됨

```
>>> b = (4, 5, 6)
>>> list(b)
[4, 5, 6]
```

  - **리스트를 생성할 때는 [ ](대괄호)를 사용하고, 튜플을 생성할 때는 ( )(괄호)를 사용한다는 점이 중요**
  - **튜플은 안에 저장된 요소를 변경, 추가, 삭제할 수 없으므로 요소가 그대로유지되어야 할 때 사용한다는 점도 기억**

- **cf) 리스트와 튜플로 변수 생성**
  - 변수 여러 개를 한번에 생성
  - 변수의 개수와 리스트, 튜플의 요소의 개수는 동일해야함

```
>>> a,b,c=[1,2,3]
>>> print(a,b,c)
1 2 3
>>> d,e,f=(4,5,6)
>>> print(d,e,f)
4 5 6
>>>
```

  - **패킹 : 변수에 리스트, 튜플을 할당, 여러 데이터를 1개로 묶는 것**
  - **언패킹 : 리스트와 튜플의 요소를 변수 여러 개에 할당**

```
>>> a=[1,2,3]
>>> b=(1,2,3)
>>> c=1,2,3
```

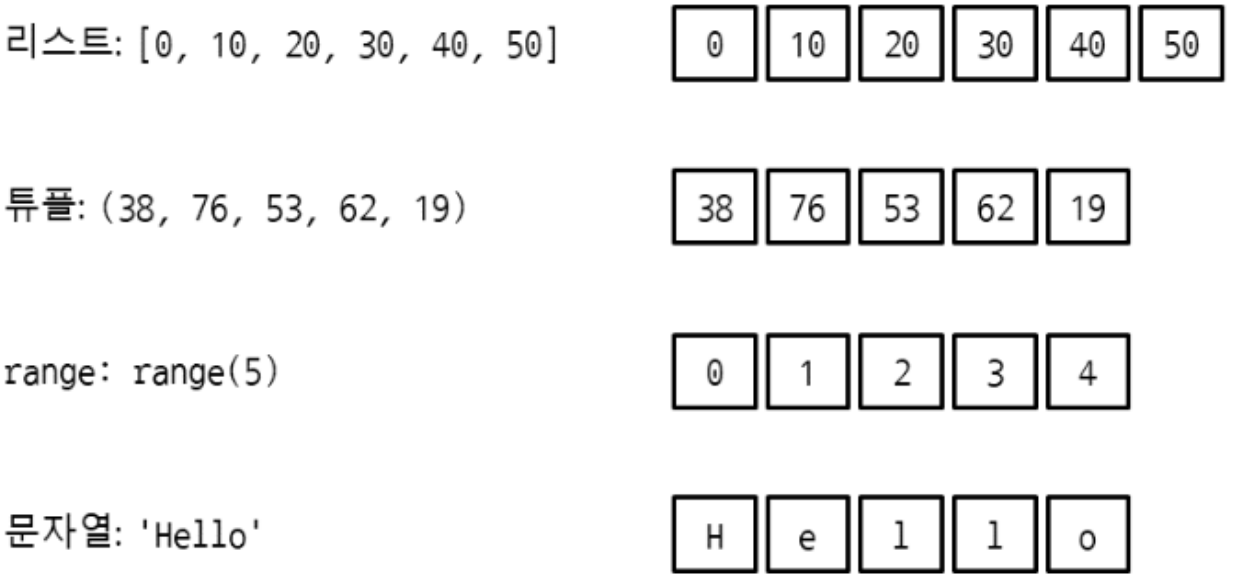
```
>>> x=[1,2,3]
>>> a,b,c=x
>>> print(a,b,c)
1 2 3
>>> y=(4,5,6)
>>> d,e,f=y
>>> print(d,e,f)
4 5 6
```

- **cf) 언패킹 실패**

```
>>> a = 1, 2, 3
>>> one, two = a
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    one, two = a
ValueError: too many values to unpack (expected 2)
```

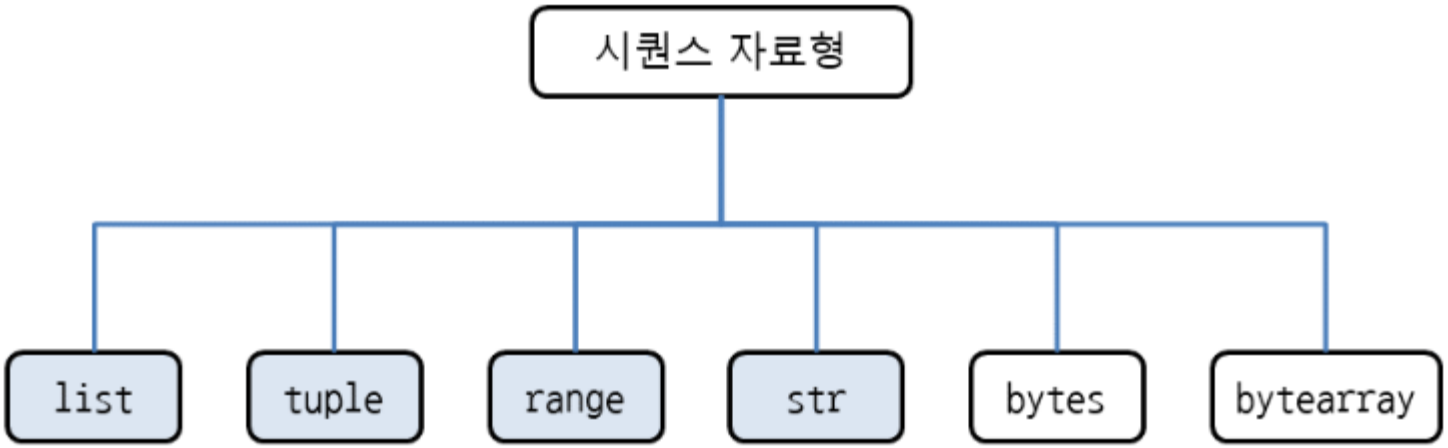
  - 튜플 요소의 수와 언패킹할 요소의 수가 일치 하지 않음
  - a는 3개의 요소로 이루어진 튜플이지만, 변수는 2개만 존재

- **시퀀스 자료형 활용하기**
  - **리스트, 튜플, range, 문자열의 공통점: 연속적(sequence)**



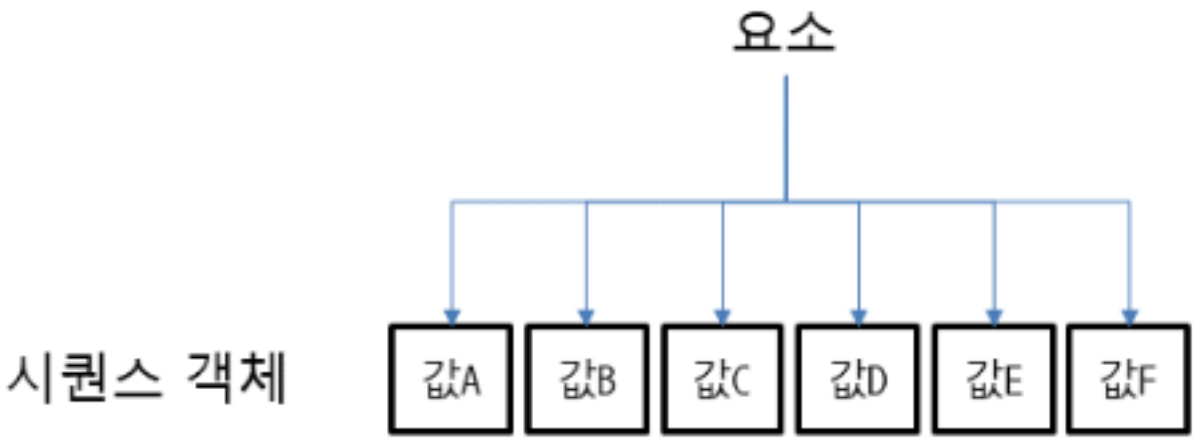
값이 연속적으로 이어진 자료형

- **시퀀스 자료형 활용하기**
  - 값이 연속적으로 이어진 자료형을 시퀀스 자료형(sequence types)라고 부름



  - 이 시퀀스 자료형 중에서 list, tuple, range, str을 주로 사용하며 bytes, bytearray 라는 자료형도 있음

- **시퀀스 자료형 활용하기**
  - 시퀀스 자료형의 특징: 공통 동작과 기능을 제공
  - 시퀀스 객체: 시퀀스 자료형으로 만든 객체
  - 요소(element): 시퀀스 객체에 들어있는 각 값



시퀀스 객체와 요소

- **특정 값이 있는지 확인하기**
  - 리스트 a에서 30과 100이 있는지 확인함

• 값 in 시퀀스객체

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> 30 in a
True
>>> 100 in a
False
```

  - 시퀀스 객체에 in 연산자: 값이 있으면 True, 없으면 False
  - 시퀀스 객체에 not in 연산자: 특정 값이 없는지 확인함

• 값 not in 시퀀스객체

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> 100 not in a
True
>>> 30 not in a
False
```

- **특정 값이 있는지 확인하기**
  - not in은 특정 값이 없으면 True, 있으면 False
  - 튜플, range, 문자열도 같은 방법으로 활용할 수 있음

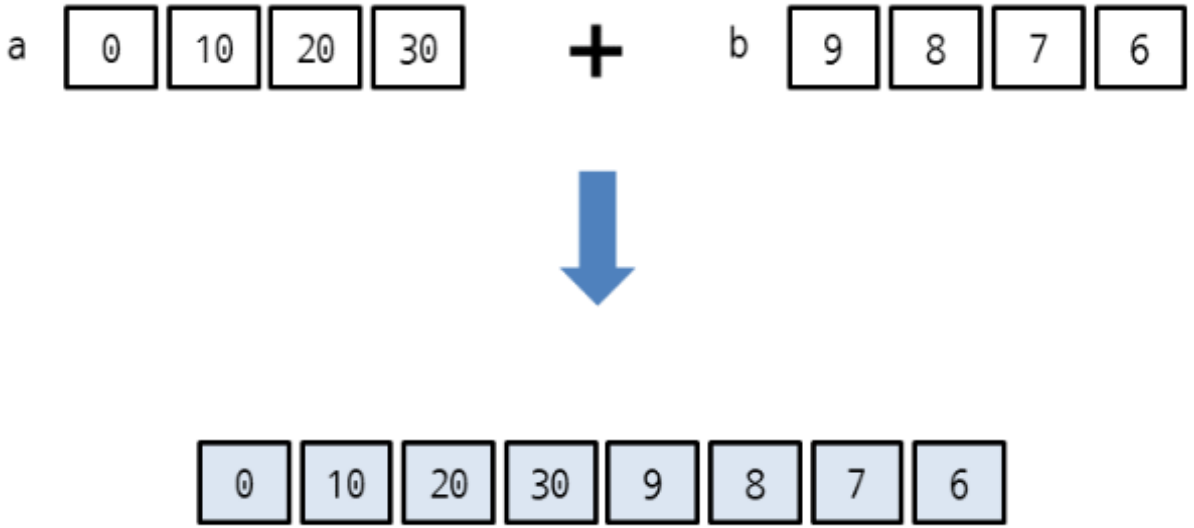
```
>>> 43 in (38, 76, 43, 62, 19)
True
>>> 1 in range(10)
True
>>> 'P' in 'Hello, Python'
True
```

- **시퀀스 객체 연결하기**
  - \+ 연산자로 두 객체를 연결해 새 객체를 만들 수 있음

• 시퀀스객체1 + 시퀀스객체2

```
>>> a = [0, 10, 20, 30]
>>> b = [9, 8, 7, 6]
>>> a + b
[0, 10, 20, 30, 9, 8, 7, 6]
```

  - 리스트 a와 b를 더하니 두 리스트가 연결되었음
  - 변수를 만들지 않고 리스트 여러 개를 직접 연결해도 상관없음



- **시퀀스 객체 연결하기**
  - **시퀀스 자료형 중에서 range는 + 연산자로 객체를 연결할 수 없음**

```
>>> range(0, 10) + range(10, 20)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    range(0, 10) + range(10, 20)
TypeError: unsupported operand type(s) for +: 'range' and 'range'
```

  - **range를 리스트 또는 튜플로 만들어서 연결하면 됨**

```
>>> list(range(0, 10)) + list(range(10, 20))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
>>> tuple(range(0, 10)) + tuple(range(10, 20))
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19)
```

  - **문자열은 + 연산자로 여러 문자열을 연결할 수 있음**

```
>>> 'Hello, ' + 'world!'
'Hello, world!'
```

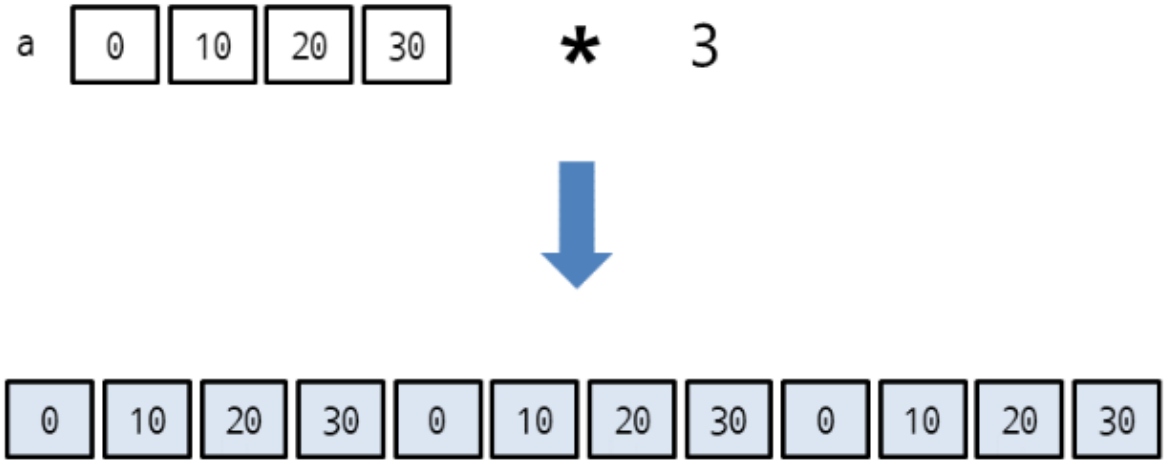
  - 따옴표로 묶은 문자열 'Hello, '와 'world!'를 연결하여 'Hello, world!'가 나옴
  - 파이썬에서 문자열 연결은 여러 가지 결과를 묶어서 한 번에 출력할 때 자주 사용함

- **시퀀스 객체 반복하기**
  - * 연산자: 시퀀스 객체를 특정 횟수만큼 반복하여 새 시퀀스 객체를 만듦
    (0 또는 음수를 곱하면 빈 객체가 나오며 실수는 곱할 수 없음)

  - 시퀀스객체 * 정수
  - 정수 * 시퀀스객체

```
>>> [0, 10, 20, 30] * 3
[0, 10, 20, 30, 0, 10, 20, 30, 0, 10, 20, 30]
```

  - 요소 0, 10, 20, 30이 들어있는 리스트를 3번 반복해서 새 리스트를 만들었음



- 시퀀스 객체 반복하기
  - range는 + 연산자로 객체를 연결할 수 없고 range는 * 연산자를 사용하여 반복할 수 없음

```
>>> range(0, 5, 2) * 3
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    range(0, 5, 2) * 3
TypeError: unsupported operand type(s) for *: 'range' and 'int'
```

  - range를 리스트 또는 튜플로 만들어서 반복하면 됨

```
>>> list(range(0, 5, 2)) * 3
[0, 2, 4, 0, 2, 4, 0, 2, 4]
>>> tuple(range(0, 5, 2)) * 3
(0, 2, 4, 0, 2, 4, 0, 2, 4)
```

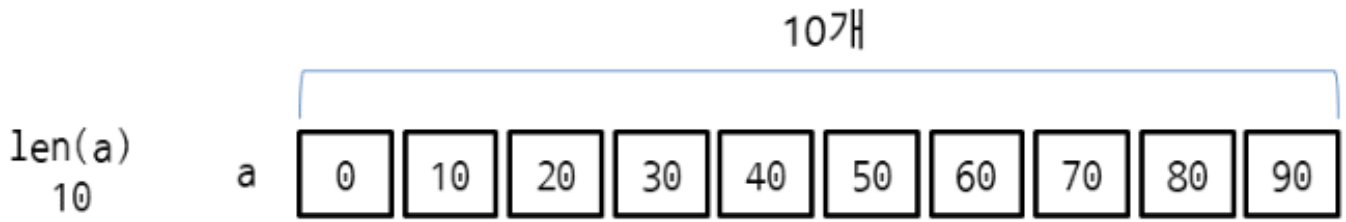
  - 문자열은 * 연산자를 사용하여 반복할 수 있음

```
>>> 'Hello, ' * 3
'Hello, Hello, Hello, '
```

- **시퀀스 객체의 요소 개수 구하기**
  - **시퀀스 객체에는 요소가 여러 개 들어있는 이 요소의 개수(길이)를 구할 때는 len 함수를 사용함**
    - len(시퀀스객체)
  - 리스트와 튜플의 요소 개수 구하기
    - 리스트의 요소 개수부터 구해보자
    - 리스트 a에는 요소가 10개 들어있으므로 len(a)는 10이 나옴

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> len(a)
10
```

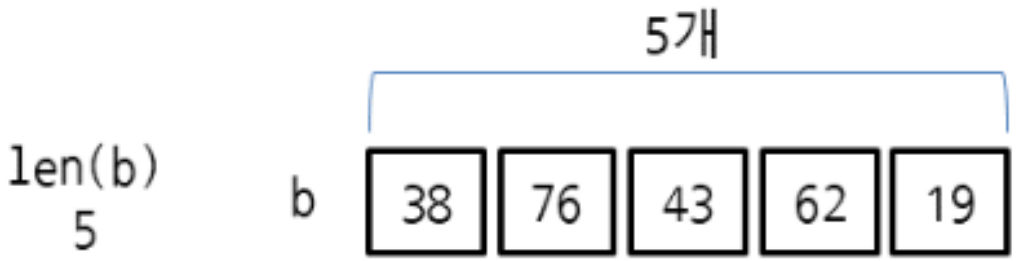
```
a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```



- **리스트와 튜플의 요소 개수 구하기**
  - 튜플 b에는 요소가 5개 들어있으므로 len(b)는 5가 나옴

```
>>> b = (38, 76, 43, 62, 19)
>>> len(b)
5
```

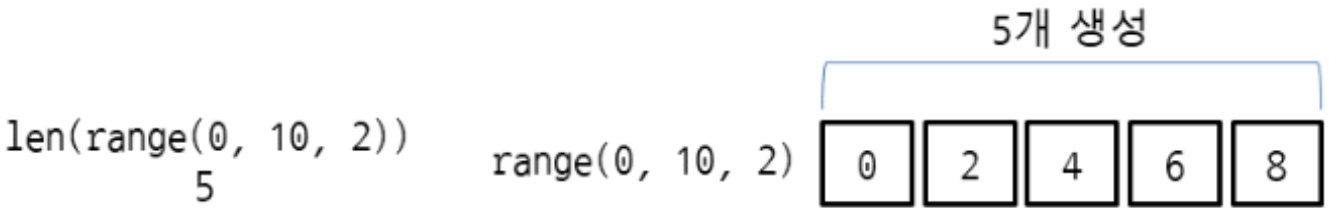
b = (38, 76, 43, 62, 19)



- **range의 숫자 생성 개수 구하기**
  - range에 len 함수를 사용하면 숫자가 생성되는 개수를 구함

```
>>> len(range(0, 10, 2))
5
```

  - range(0, 10, 2)는 0부터 10까지 2씩 증가하므로 0, 2, 4, 6, 8임
  - 따라서 5가 나옴

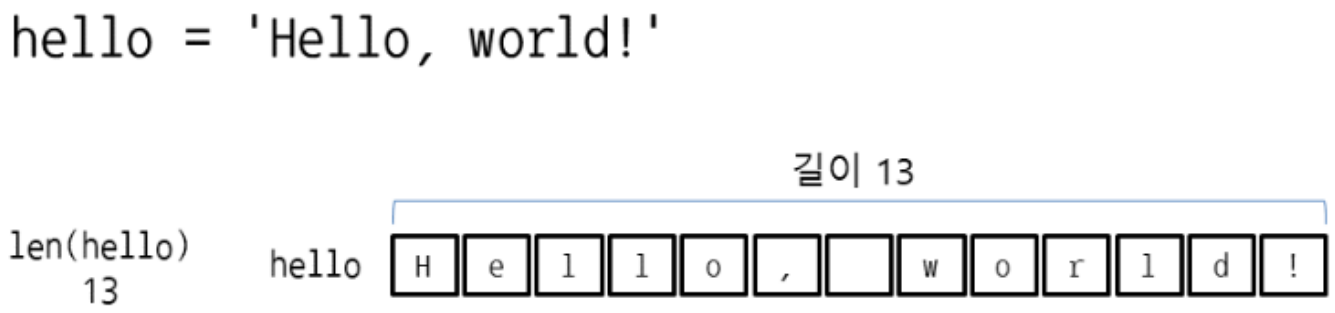


  - 리스트(튜플)의 값을 직접 타이핑해서 요소의 개수를 알기 쉬웠음
  - 실무에서는 range 등을 사용하여 리스트(튜플)를 생성하거나,
    다양한 방법으로 요소를 추가, 삭제, 반복하므로
    요소의 개수가 한눈에 보이지 않음
  - 요소의 개수를 구하는 len 함수를 자주 활용하게 됨

- **문자열의 길이 구하기**
  - 문자열의 길이(문자의 개수)를 구해보자
  - 문자열도 시퀀스 자료형이므로 len 함수를 사용하면 됨

```
>>> hello = 'Hello, world!'
>>> len(hello)
13
```

  - len으로 'Hello, world!' 문자열이 들어있는 hello의 길이를 구해보면 13이 나옴



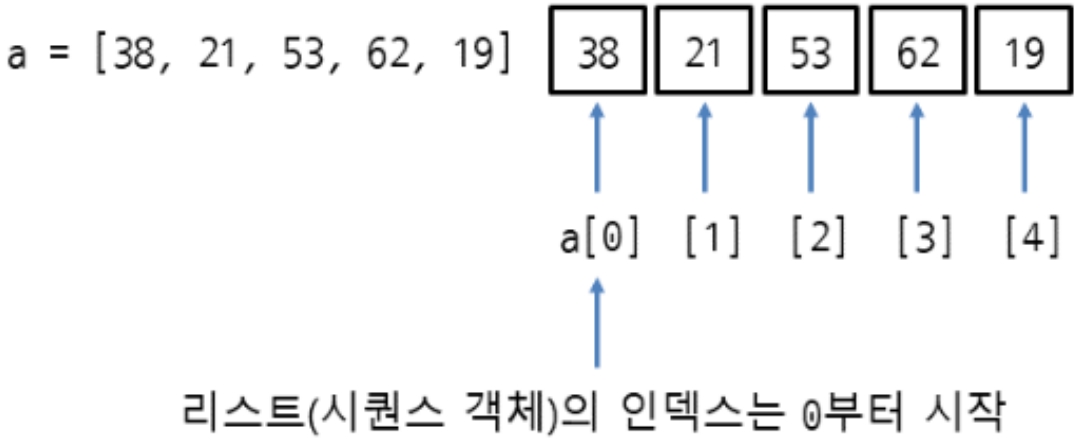
- **문자열의 길이 구하기**
  - 문자열의 길이는 공백까지 포함
  - 단, 문자열을 묶은 따옴표는 제외함
  - 이 따옴표는 문자열을 표현하는 문법일 뿐 문자열 길이에는 포함되지 않음 (문자열 안에 포함된 작은따옴표, 큰따옴표는 포함됨)
  - 한글 문자열의 길이도 len으로 구하면 됨

```
>>> hello = '안녕하세요'
>>> len(hello)
5
```

- **인덱스 사용하기**
  - 시퀀스 객체의 각 요소는 순서가 정해져 있으며, 이 순서를 인덱스라고 부름
  - 시퀀스 객체에 [ ](대괄호)를 붙이고 [ ] 안에 각 요소의 인덱스를 지정하면 해당 요소에 접근할 수 있음
    - 시퀀스객체[인덱스]

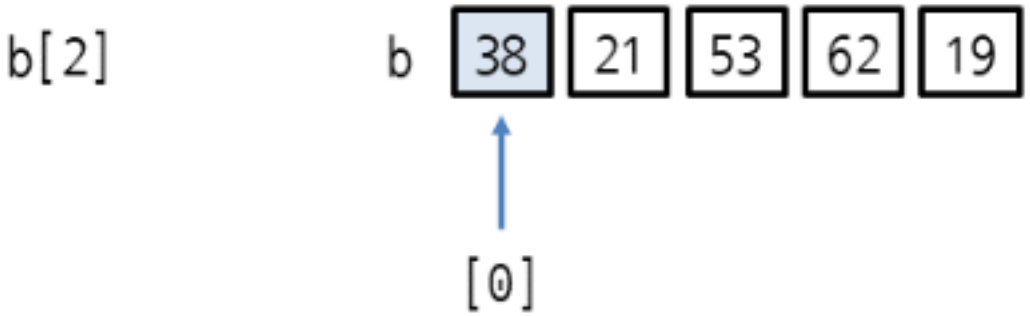
```
>>> a = [38, 21, 53, 62, 19]
>>> a[0]    # 리스트의 첫 번째(인덱스 0) 요소 출력
38
>>> a[2]    # 리스트의 세 번째(인덱스 2) 요소 출력
53
>>> a[4]    # 리스트의 다섯 번째(인덱스 4) 요소 출력
19
```

- **인덱스 사용하기**
  - 인덱스(index, 색인)는 위치 값을 뜻하는데
    국어사전 옆면에 ㄱ, ㄴ, ㄷ으로 표시해 놓은 것과 비슷함
  - 여기서 **주의할 점은 시퀀스 객체의 인덱스는 항상 0부터 시작한다는 점**
    (대다수의 프로그래밍 언어는 인덱스가 0부터 시작함)
  - 리스트 a의 첫 번째 요소는 a[0]이 됨



- **인덱스 사용하기**
  - 튜플, range, 문자열도 [ ]에 인덱스를 지정하면 해당 요소를 가져올 수 있음
  - 튜플 b의 첫 번째(인덱스 0) 요소를 출력

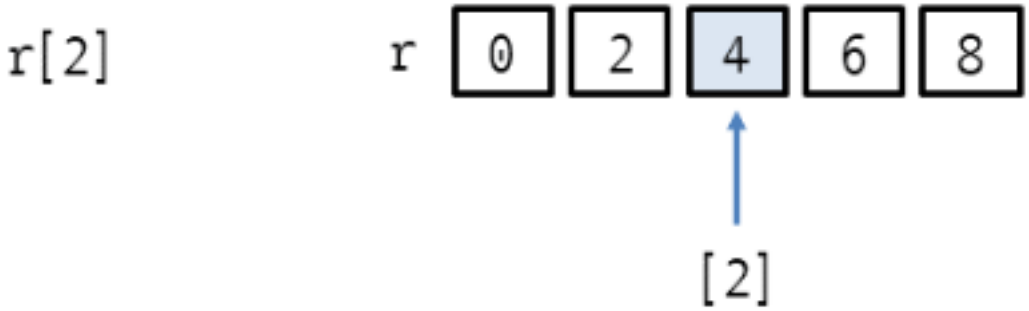
```
>>> b = (38, 21, 53, 62, 19)
>>> b[0]        # 튜플의 첫 번째(인덱스 0) 요소 출력
38
```



인덱스로 튜플의 첫 번째 요소에 접근

- **인덱스 사용하기**
  - range도 인덱스로 접근할 수 있음
  - range의 세 번째(인덱스 2) 요소를 출력

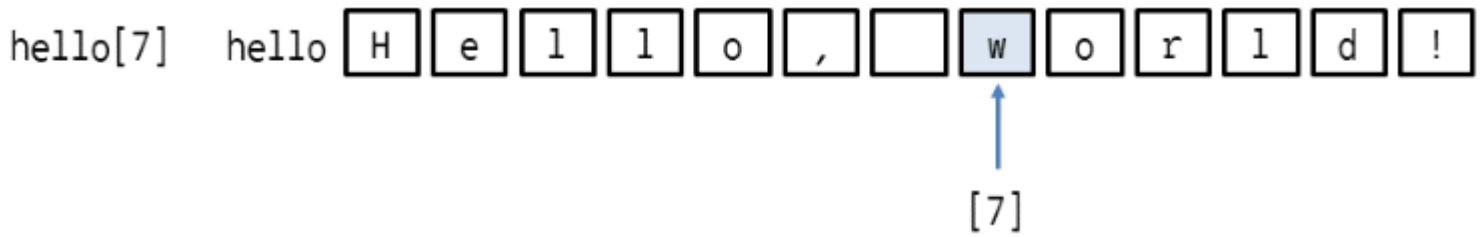
```
>>> r = range(0, 10, 2)
>>> r[2]        # range의 세 번째(인덱스 2) 요소 출력
4
```



인덱스로 range 객체의 세 번째 요소에 접근

- **인덱스 사용하기**
  - 문자열은 요소가 문자이므로 인덱스로 접근하면 문자가 나옴
  - 문자열 hello의 여덟 번째 요소를 출력

```
>>> hello = 'Hello, world!'
>>> hello[7]    # 문자열의 여덟 번째(인덱스 7) 요소 출력
'w'
```

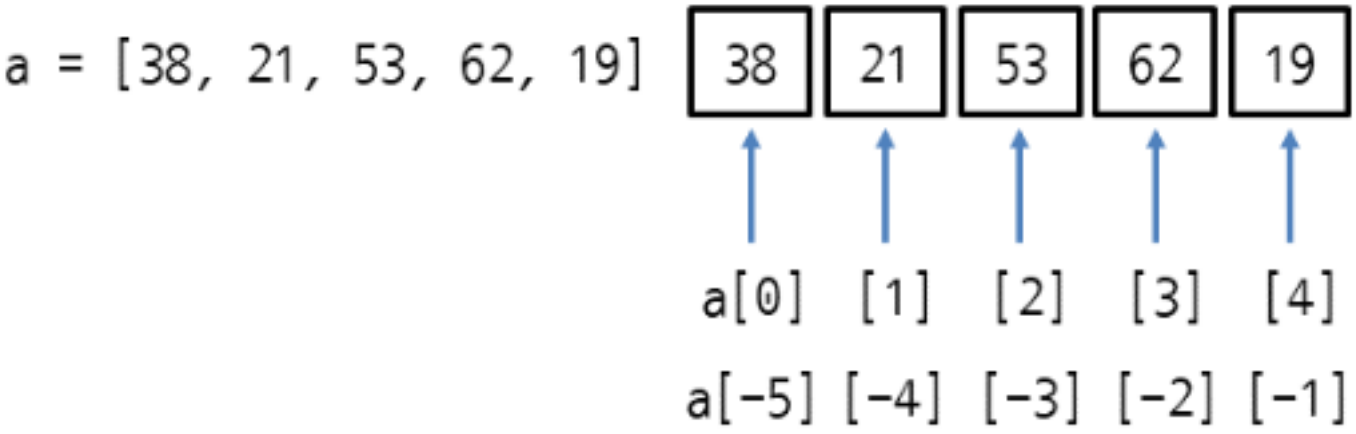


인덱스로 문자열의 요소에 접근

- **음수 인덱스 지정하기**

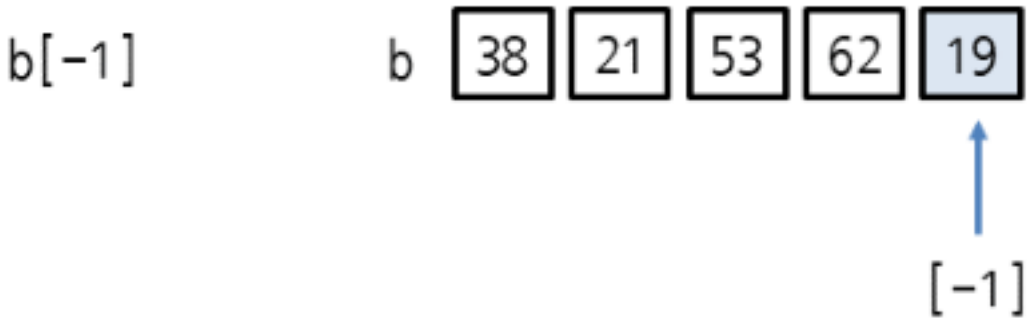
```
>>> a = [38, 21, 53, 62, 19]
>>> a[-1]   # 리스트의 뒤에서 첫 번째(인덱스 -1) 요소 출력
19
>>> a[-5]   # 리스트의 뒤에서 다섯 번째(인덱스 -5) 요소 출력
38
```

  - 시퀀스 객체에 인덱스를 음수로 지정하면 뒤에서부터 요소에 접근하게 됨
  - 즉, -1은 뒤에서 첫 번째, -5는 뒤에서 다섯 번째 요소임
  - a[-1]은 뒤에서 첫 번째 요소인 19, a[-5]는 뒤에서 다섯 번째 요소인 38이 나옴



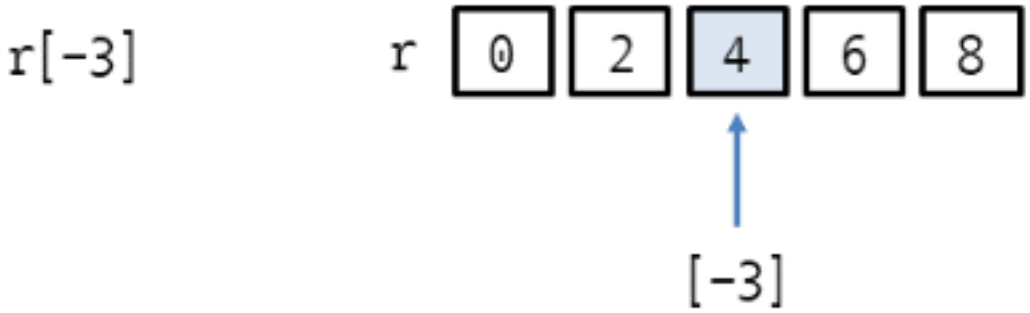
- **음수 인덱스 지정하기**
  - 튜플, range, 문자열도 음수 인덱스를 지정하면 뒤에서부터 요소에 접근함
  - 튜플 b의 뒤에서 첫 번째(인덱스 -1) 요소를 출력

```
>>> b = (38, 21, 53, 62, 19)
>>> b[-1]       # 튜플의 뒤에서 첫 번째(인덱스 -1) 요소 출력
19
```



- **음수 인덱스 지정하기**
  - range도 음수 인덱스로 접근할 수 있음
  - range의 뒤에서 세 번째(인덱스 -3) 요소를 출력

```
>>> r = range(0, 10, 2)
>>> r[-3]        # range의 뒤에서 세 번째(인덱스 -3) 요소 출력
4
```

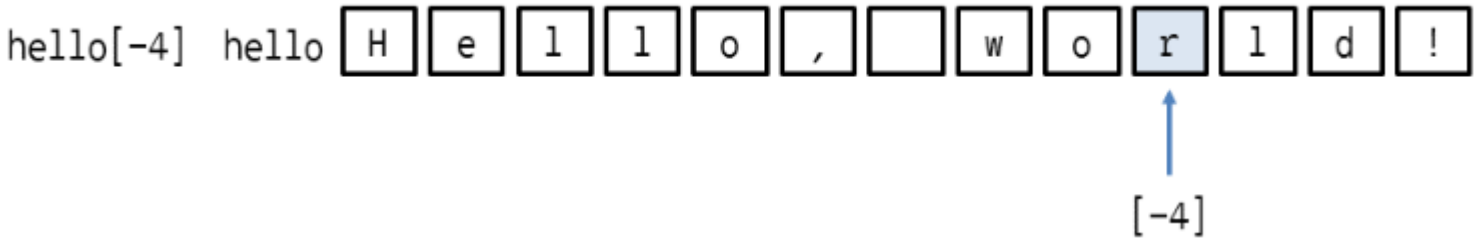


음수 인덱스로 range 객체의 요소에 접근하기

- **음수 인덱스 지정하기**
  - 문자열 hello의 뒤에서 네 번째(인덱스 -4) 요소를 출력

```
>>> hello = 'Hello, world!'
>>> hello[-4]    # 문자열의 뒤에서 네 번째(인덱스 -4) 요소 출력
'r'
```

음수 인덱스로 문자열의 요소에 접근하기



- **인덱스의 범위를 벗어나면?**
  - 실행을 해보면 리스트의 인덱스가 범위를 벗어났다는 IndexError가 발생함
  - 인덱스는 0부터 시작하므로 마지막 요소의 인덱스는 4
  - 마지막 요소의 인덱스는 시퀀스 객체의 요소 개수보다 1 작음
  - 시퀀스 객체를 사용할 때 자주 틀리는 부분이므로 꼭 기억해두자
  - 튜플, range, 문자열도 범위를 벗어난 인덱스를 지정하면 IndexError가 발생함

```
>>> a = [38, 21, 53, 62, 19]
>>> a[5]    # 인덱스 5는 범위를 벗어났으므로 에러
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    a[5]
IndexError: list index out of range
```

- **마지막 요소에 접근하기**
  - 인덱스를 -1로 지정: 뒤에서 첫 번째 요소, 바로 시퀀스 객체의 마지막 요소임
  - 시퀀스 객체의 마지막 요소에 접근하는 다른 방법은 없을지 다음과 같이 len 함수로 리스트의 길이를 구한 뒤 이 길이를 인덱스로 지정해보면 에러가 발생함

```
>>> a = [38, 21, 53, 62, 19]
>>> len(a)    # 리스트의 길이를 구함
5
>>> a[5]      # 리스트의 길이를 인덱스로 지정
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    a[5]
IndexError: list index out of range
```

- **마지막 요소에 접근하기**
  - 리스트 a의 인덱스는 0부터 4

```
>>> a[4]
19
```

  - 그럼 조금 응용해서 인덱스에 len을 조합해보자

```
>>> a[len(a)]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    a[len(a)]
IndexError: list index out of range
```

  - len(a) - 1은 4이므로 마지막 문자가 나옴
  - 마지막 요소 19가 나옴

```
>>> a[len(a) - 1]    # 마지막 요소(인덱스 4) 출력
19
```

- **요소에 값 할당하기**
  - 시퀀스 객체는 [ ]로 요소에 접근한 뒤 =로 값을 할당함
    - 시퀀스객체[인덱스] = 값

```
>>> a = [0, 0, 0, 0, 0]    # 0이 5개 들어있는 리스트
>>> a[0] = 38
>>> a[1] = 21
>>> a[2] = 53
>>> a[3] = 62
>>> a[4] = 19
>>> a
[38, 21, 53, 62, 19]
>>> a[0]
38
>>> a[4]
19
```

  - **튜플의 [ ]에 인덱스를 지정한 뒤 값을 할당하면 에러가 발생함**

```
>>> b = (0, 0, 0, 0, 0)
>>> b[0] = 38
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    b[0] = 38
TypeError: 'tuple' object does not support item assignment
```

- 요소에 값 할당하기
  - range와 문자열도 안에 저장된 요소를 변경할 수 없음

```
>>> r = range(0, 10, 2)
>>> r[0] = 3
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    r[0] = 3
TypeError: 'range' object does not support item assignment
>>> hello = 'Hello, world!'
>>> hello[0] = 'A'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    hello[0] = 'A'
TypeError: 'str' object does not support item assignment
```

  - 시퀀스 자료형 중에서 튜플, range, 문자열은 읽기 전용임

- **del로 시퀀스 객체의 요소 삭제**
  - 요소 삭제는 다음과 같이 del 뒤에 삭제할 요소를 지정해주면 됨
    - del 시퀀스객체[인덱스]
  - 리스트를 만들고 세 번째 요소(인덱스 2)를 삭제

```
>>> a = [38, 21, 53, 62, 19]
>>> del a[2]
>>> a
[38, 21, 62, 19]
```

  - del a[2]와 같이 사용하면 리스트 a의 세 번째 요소(인덱스 2)인 53을 삭제함

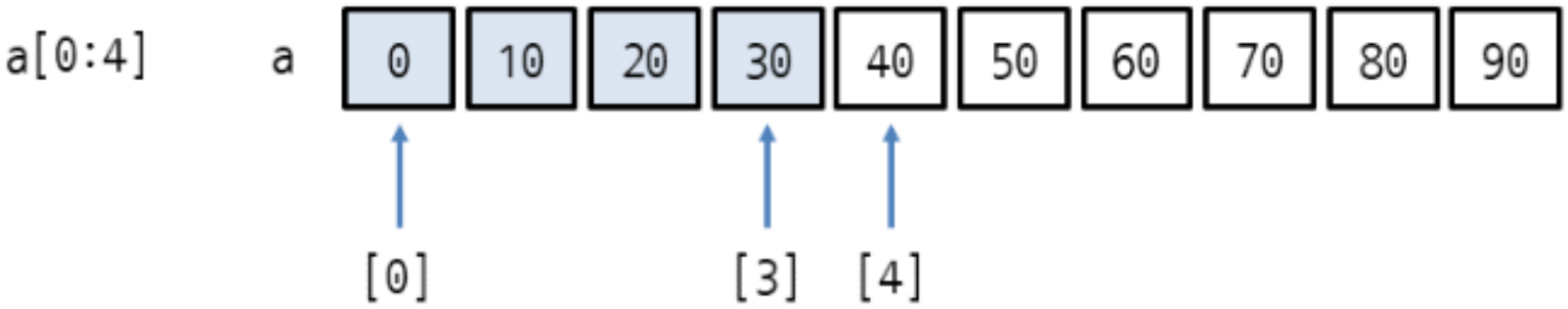
```
>>> b = (38, 21, 53, 62, 19)
>>> del b[2]
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    del b[2]
TypeError: 'tuple' object doesn't support item deletion
```

- **del로 시퀀스 객체의 요소를 삭제해보자**
  - **range와 문자열도 안에 저장된 요소를 삭제할 수 없음**

```
>>> r = range(0, 10, 2)
>>> del r[2]
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    del r[2]
TypeError: 'range' object doesn't support item deletion
>>> hello = 'Hello, world!'
>>> del hello[2]
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    del hello[2]
TypeError: 'str' object doesn't support item deletion
```

- **슬라이스 사용하기**
  - 시퀀스 슬라이스: 시퀀스 객체의 일부를 잘라냄
    - 시퀀스객체[시작인덱스:끝인덱스]
  - 리스트의 일부를 잘라서 새 리스트를 만듦

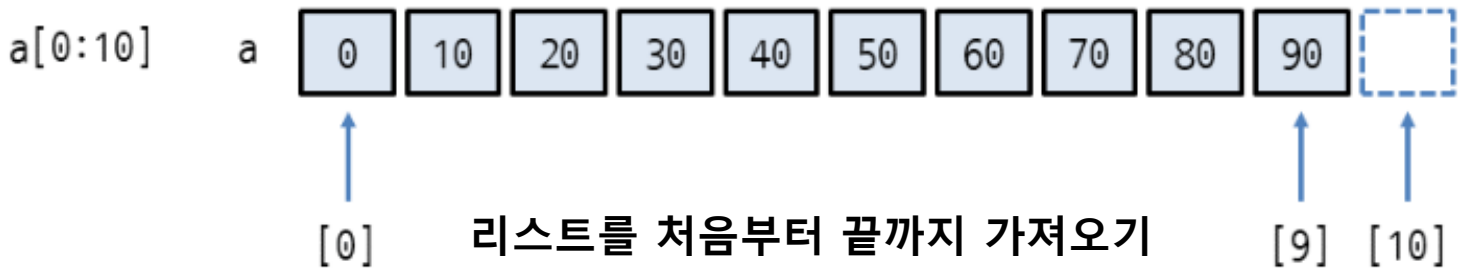
```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[0:4]     # 인덱스 0부터 3까지 잘라서 새 리스트를 만듦
[0, 10, 20, 30]
```



리스트 슬라이스

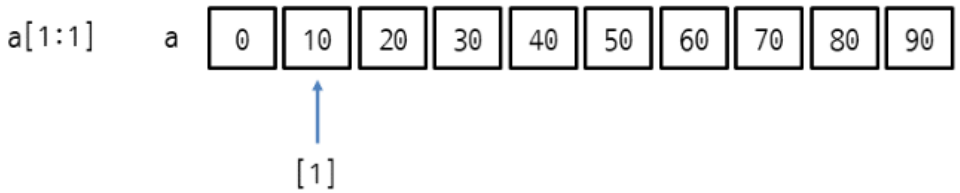
- **슬라이스 사용하기**
  - 예를 들어 요소가 10개 들어있는 리스트를 처음부터 끝까지 가져오려면 [0:9]가 아닌 [0:10]이라야 함
    (**끝 인덱스는 범위를 벗어난 인덱스를 지정할 수 있음**)

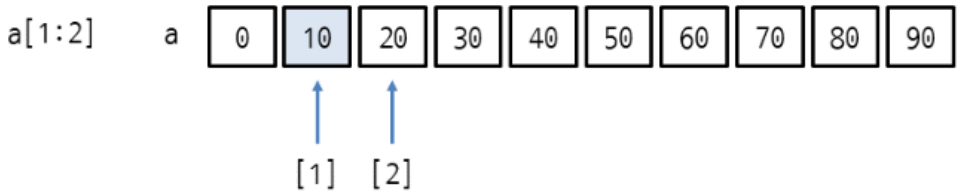
```
>>> a[0:10]    # 인덱스 0부터 9까지 잘라서 새 리스트를 만듦
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```



- **슬라이스 사용하기**
  - a[1:1]처럼 시작 인덱스와 끝 인덱스를 같은 숫자로 지정하면 아무것도 가져오지 않음
  - a[1:2]처럼 끝 인덱스에 1을 더 크게 지정해야 요소 하나를 가져옴
  - **슬라이스를 했을 때 실제로 가져오는 요소는 시작 인덱스부터 끝 인덱스 - 1까지**

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[1:1]    # 인덱스 1부터 0까지 잘라서 새 리스트를 만듦
[]
>>> a[1:2]    # 인덱스 1부터 1까지 잘라서 새 리스트를 만듦
[10]
```

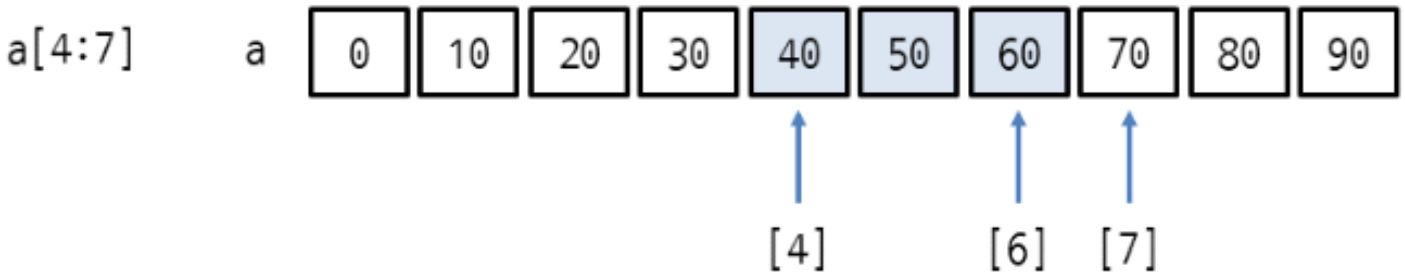




- **리스트의 중간 부분 가져오기**

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[4:7]     # 인덱스 4부터 6까지 요소 3개를 가져옴
[40, 50, 60]
```

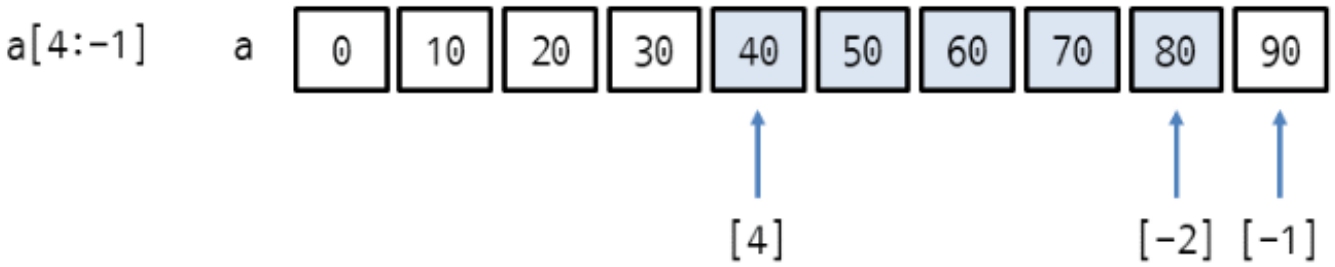
  - a[4:7]은 리스트 a 중간에 있는 인덱스 4부터 6까지 요소 3개를 가져옴



리스트의 중간 부분 가져오기

- **리스트의 중간 부분 가져오기**
  - 슬라이스는 a[4:-1]과 같이 음수를 인덱스로 지정할 수도 있음

```
>>> a[4:-1]    # 인덱스 4부터 -2까지 요소 5개를 가져옴
[40, 50, 60, 70, 80]
```



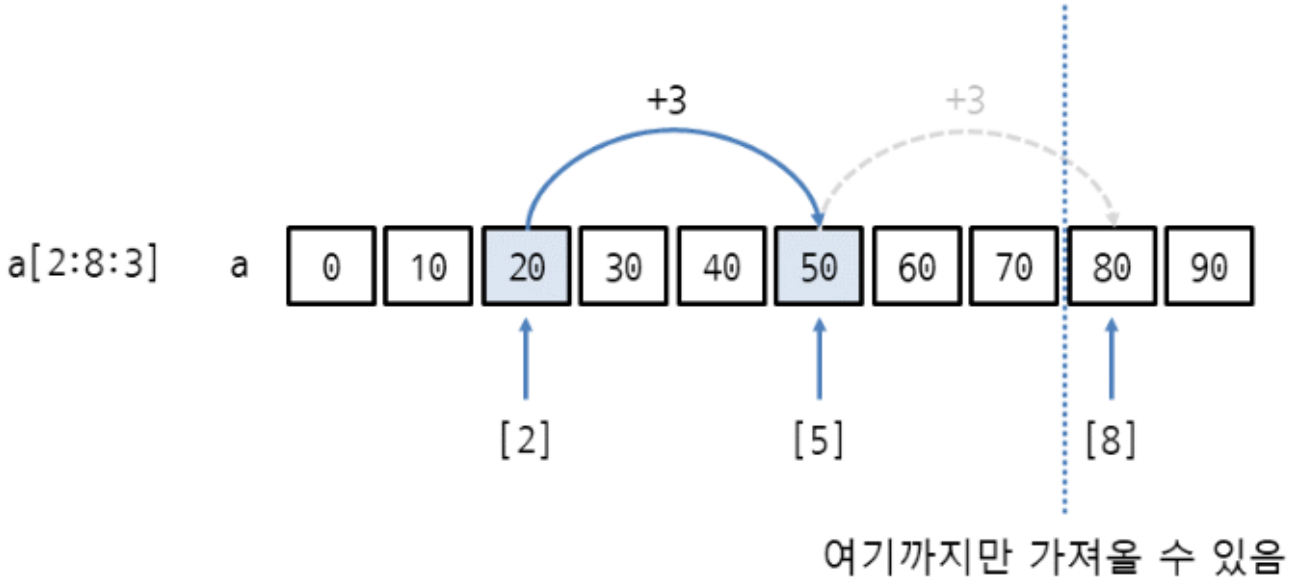
음수 인덱스를 지정하여 리스트의 중간 부분 가져오기

- **인덱스 증가폭 사용하기**
  - **슬라이스는 인덱스의 증가폭을 지정하여 범위 내에서 인덱스를 건너뛰며 요소를 가져올 수 있음**

• 시퀀스객체[시작인덱스:끝인덱스:인덱스증가폭]

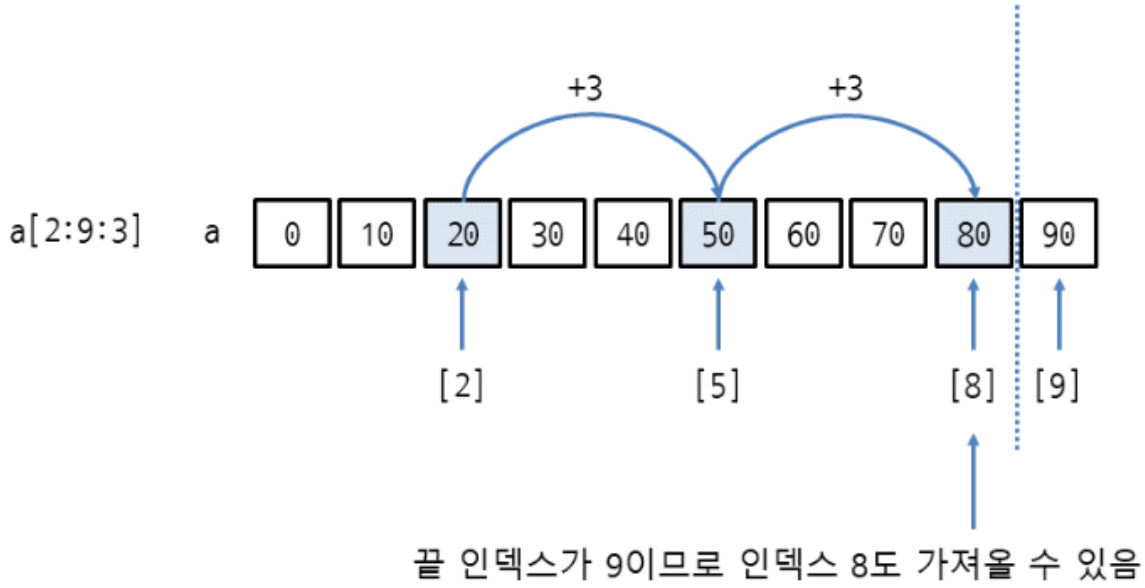
```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[2:8:3]    # 인덱스 2부터 3씩 증가시키면서 인덱스 7까지 가져옴
[20, 50]
```

  - a[2:8:3]을 실행하니 [20, 50]이 나옴



- **인덱스 증가폭 사용하기**
  - **인덱스 증가폭을 지정하더라도 가져오려는 인덱스(끝 인덱스 - 1)를 넘어설 수 없음**
  - 끝 인덱스 - 1과 증가한 인덱스가 일치한다면 해당 요소까지 가져올 수 있음
  - 끝 인덱스를 9로 지정하여 인덱스 8의 80까지 가져옴
  - [20, 50, 80]이 나옴

```
>>> a[2:9:3]    # 인덱스 2부터 3씩 증가시키면서 인덱스 8까지 가져옴
[20, 50, 80]
```



- **인덱스 생략하기**
  - **슬라이스를 사용할 때 시작 인덱스와 끝 인덱스를 생략할 수도 있음**
  - 리스트 a에서 a[:7]과 같이 시작 인덱스를 생략하면
    리스트의 처음부터 끝 인덱스 - 1(인덱스 6)까지 가져옴

• 시퀀스객체[:끝인덱스]

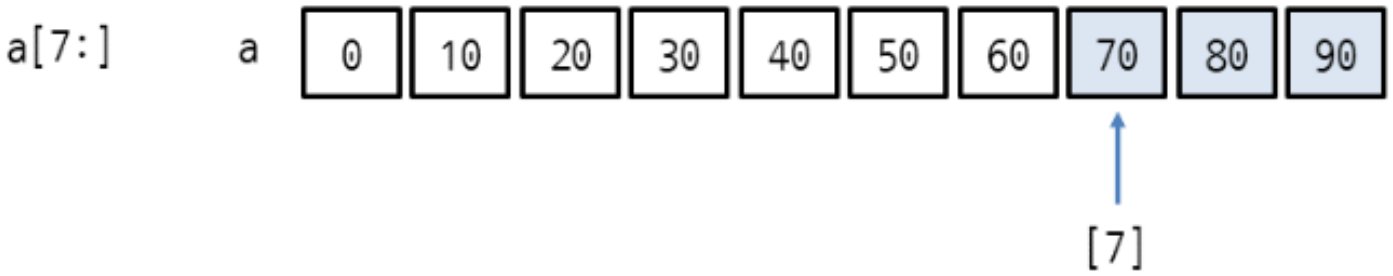
```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[:7]    # 리스트 처음부터 인덱스 6까지 가져옴
[0, 10, 20, 30, 40, 50, 60]
```

시작 인덱스 생략하기

- **인덱스 생략하기**
  - a[7:]같이 끝 인덱스를 생략하면 시작 인덱스(인덱스 7)부터 마지막 요소까지 생성

• 시퀀스객체[시작인덱스:]

```
>>> a[7:]    # 인덱스 7부터 마지막 요소까지 가져옴
[70, 80, 90]
```

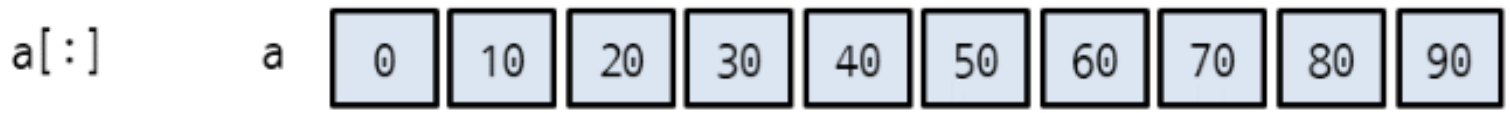


끝 인덱스 생략하기

- **인덱스 생략하기**
  - a[:]와 같이 시작 인덱스와 끝 인덱스를 둘 다 생략하면 리스트 전체를 가져옴

• 시퀀스객체[:]

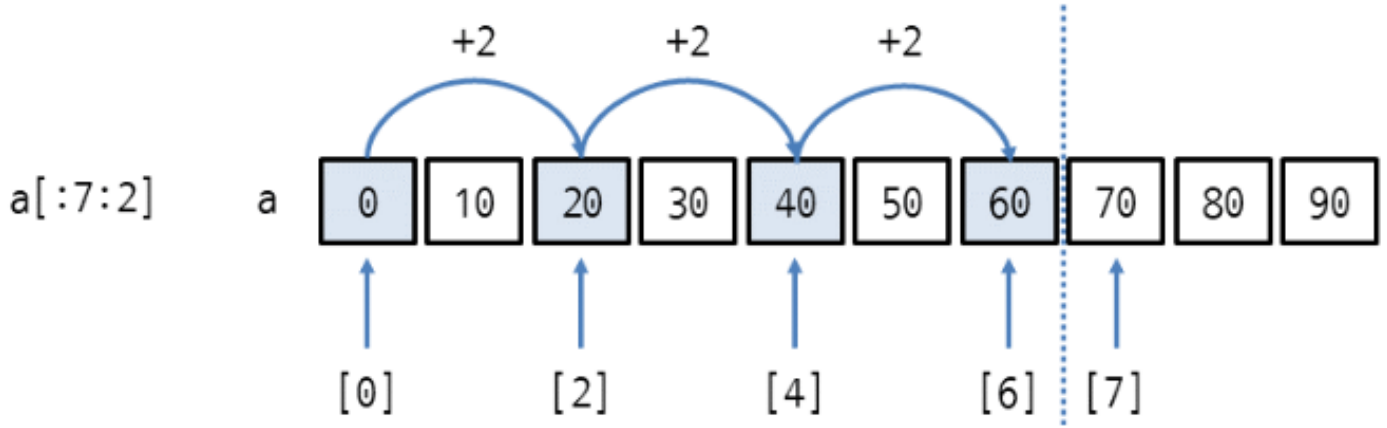
```
>>> a[:]     # 리스트 전체를 가져옴
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```



- **인덱스를 생략하면서 증가폭 사용하기**
  - 리스트 a에서 a[:7:2]와 같이 시작 인덱스를 생략하면서 인덱스 증가폭을 2로 지정하면 리스트의 처음부터 인덱스를 2씩 증가시키면서 끝 인덱스 – 1 (인덱스 6)까지 요소를 가져옴

• 시퀀스객체[:끝인덱스:증가폭]

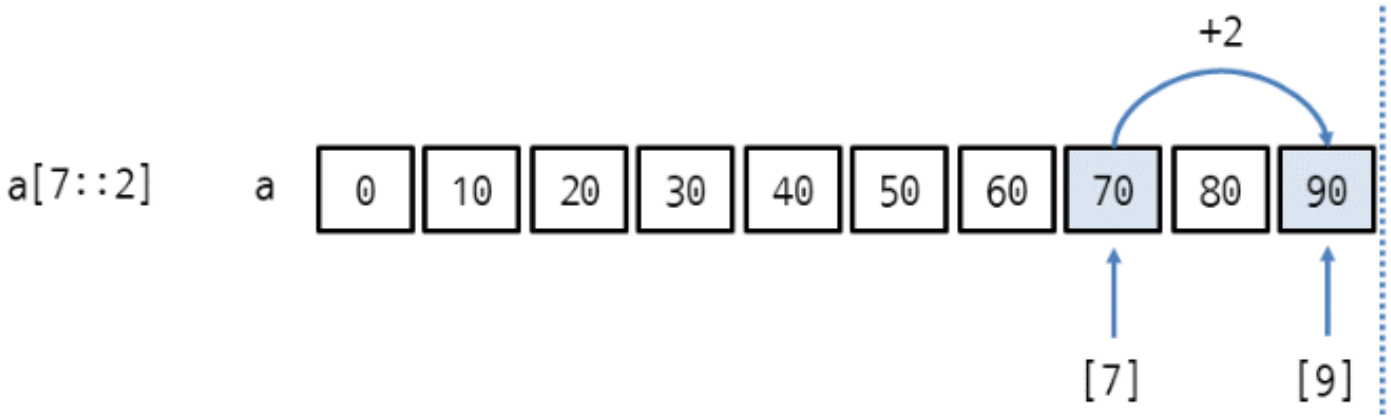
```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[:7:2]    # 리스트의 처음부터 인덱스를 2씩 증가시키면서 인덱스 6까지 가져옴
[0, 20, 40, 60]
```



- **인덱스를 생략하면서 증가폭 사용하기**
  - a[7::2]와 같이 끝 인덱스를 생략하면서 인덱스 증가폭을 2로 지정하면 시작 인덱스(인덱스 7)부터 인덱스를 2씩 증가시키면서 리스트의 마지막 요소까지 가져옴

• 시퀀스객체[시작인덱스::증가폭]

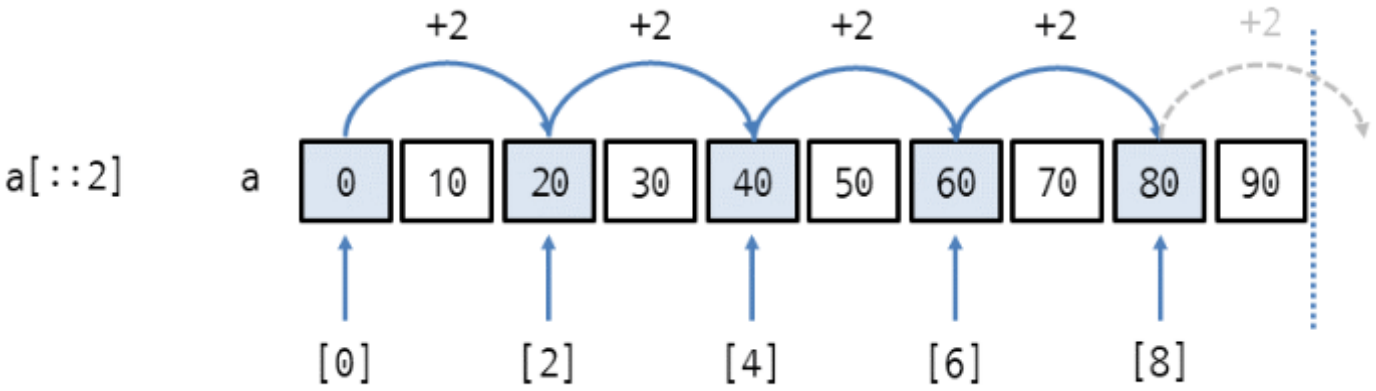
```
>>> a[7::2]    # 인덱스 7부터 2씩 증가시키면서 리스트의 마지막 요소까지 가져옴
[70, 90]
```



- **인덱스를 생략하면서 증가폭 사용하기**
  - a[::2]와 같이 시작 인덱스와 끝 인덱스를 둘다 생략하면서 인덱스 증가폭을 2로 지정하면 리스트 전체에서 인덱스 0부터 2씩 증가하면서 요소를 가져옴

• 시퀀스객체[::증가폭]

```
>>> a[::2]     # 리스트 전체에서 인덱스 0부터 2씩 증가시키면서 요소를 가져옴
[0, 20, 40, 60, 80]
```



- **인덱스를 생략하면서 증가폭 사용하기**
  - a[:7:2]와 a[7::2]는 2씩 증가한 인덱스와 끝 인덱스 - 1이 일치하여 지정된 범위에 맞게 요소를 가져옴
  - a[::2]는 끝 인덱스가 9이므로 인덱스가 2씩 증가하더라도 8까지만 증가할 수 있음
  - 인덱스 0, 2, 4, 6, 8의 요소를 가져옴
  - 만약 시작 인덱스, 끝 인덱스, 인덱스 증가폭을 모두 생략하면 어떻게 될까?

• 시퀀스객체[::]

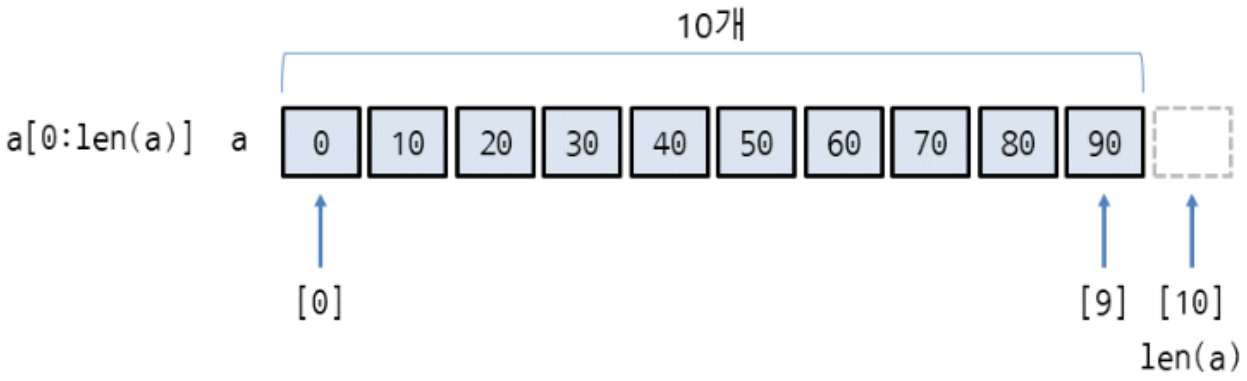
```
>>> a[::]    # 리스트 전체를 가져옴
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

  - 리스트 전체를 가져옴
  - 즉, a[:]와 a[::]는 결과가 같음

- **len 응용하기**
  - len을 응용하여 리스트 전체를 가져와보자

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[0:len(a)]    # 시작 인덱스에 0, 끝 인덱스에 len(a) 지정하여 리스트 전체를 가져옴
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[:len(a)]     # 시작 인덱스 생략, 끝 인덱스에 len(a) 지정하여 리스트 전체를 가져옴
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

  - 리스트 a의 요소는 10개임
  - len(a)는 10이고, a[0:10]과 같음
  - 끝 인덱스는 가져오려는 인덱스보다 1을 더 크게 지정한다고 했으므로 **len(a)에서 1을 빼지 않아야 함**
  - 길이가 10인 리스트는 [0:10]이라야 리스트 전체를 가져옴

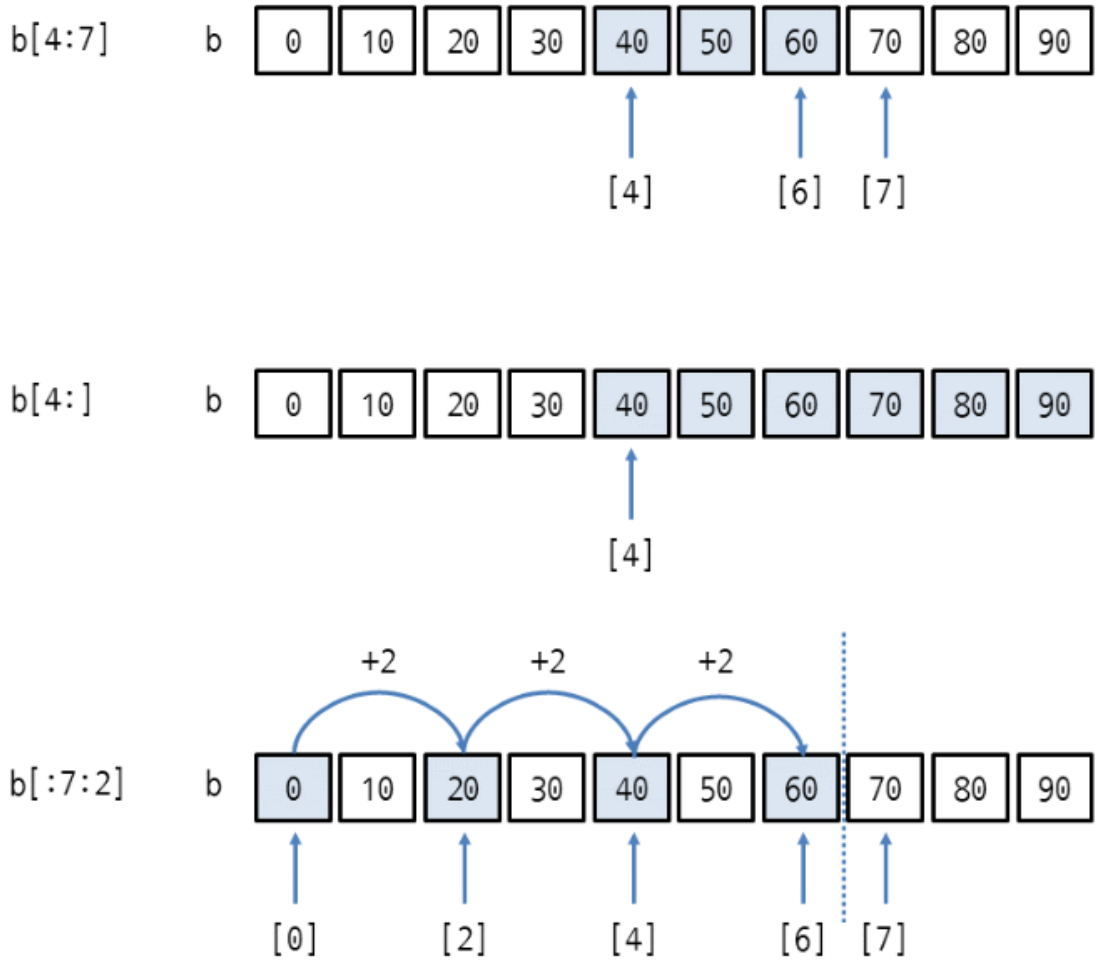


- **튜플에 슬라이스 사용하기**
  - 파이썬에서는 튜플, range, 문자열도 시퀀스 자료형이므로 리스트와 같은 방식으로 슬라이스를 사용할 수 있음
  - 지정된 범위만큼 튜플을 잘라서 새 튜플을 만들자

- 튜플[시작인덱스:끝인덱스]
- 튜플[시작인덱스:끝인덱스:인덱스증가폭]

```
>>> b = (0, 10, 20, 30, 40, 50, 60, 70, 80, 90)
>>> b[4:7]     # 인덱스 4부터 6까지 요소 3개를 가져옴
(40, 50, 60)
>>> b[4:]      # 인덱스 4부터 마지막 요소까지 가져옴
(40, 50, 60, 70, 80, 90)
>>> b[:7:2]    # 튜플의 처음부터 인덱스를 2씩 증가시키면서 인덱스 6까지 가져옴
(0, 20, 40, 60)
```

- **튜플에 슬라이스 사용하기**

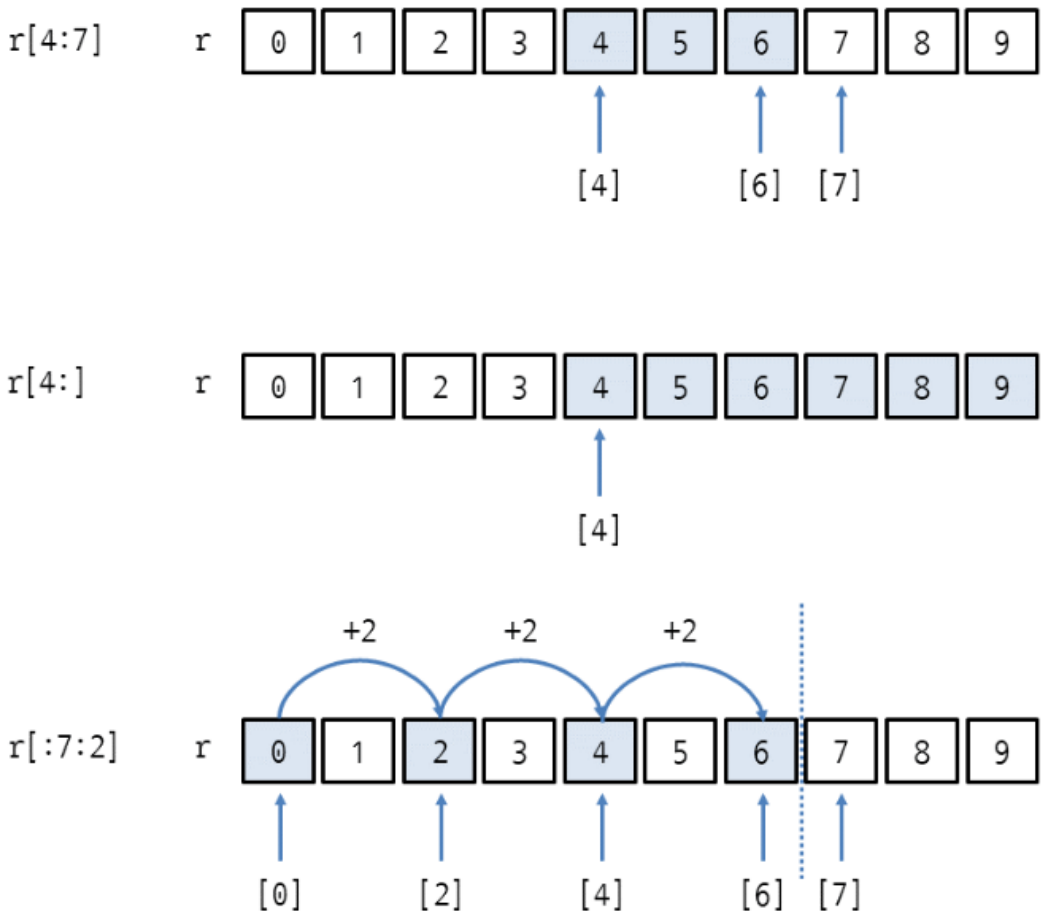


- **range에 슬라이스 사용하기**
  - range에 슬라이스를 사용하면 지정된 범위의 숫자를 생성하는 range 객체를 새로 생성

- range 객체[시작인덱스:끝인덱스]
- range 객체[시작인덱스:끝인덱스:인덱스증가폭]

```
>>> r = range(10)
>>> r
range(0, 10)
>>> r[4:7]     # 인덱스 4부터 6까지 숫자 3개를 생성하는 range 객체를 만듦
range(4, 7)
>>> r[4:]      # 인덱스 4부터 9까지 숫자 6개를 생성하는 range 객체를 만듦
range(4, 10)
>>> r[:7:2]    # 인덱스 0부터 2씩 증가시키면서 인덱스 6까지 숫자 4개를 생성하는 range 객체를 만듦
range(0, 7, 2)
```

- **range에 슬라이스 사용하기**



- **range, 문자열에 슬라이스 사용하기**
  - range는 리스트, 튜플과는 달리 요소가 모두 표시되지 않고 생성 범위만 표시됨
  - 잘라낸 range 객체를 리스트로 만들려면 list에 넣으면 됨

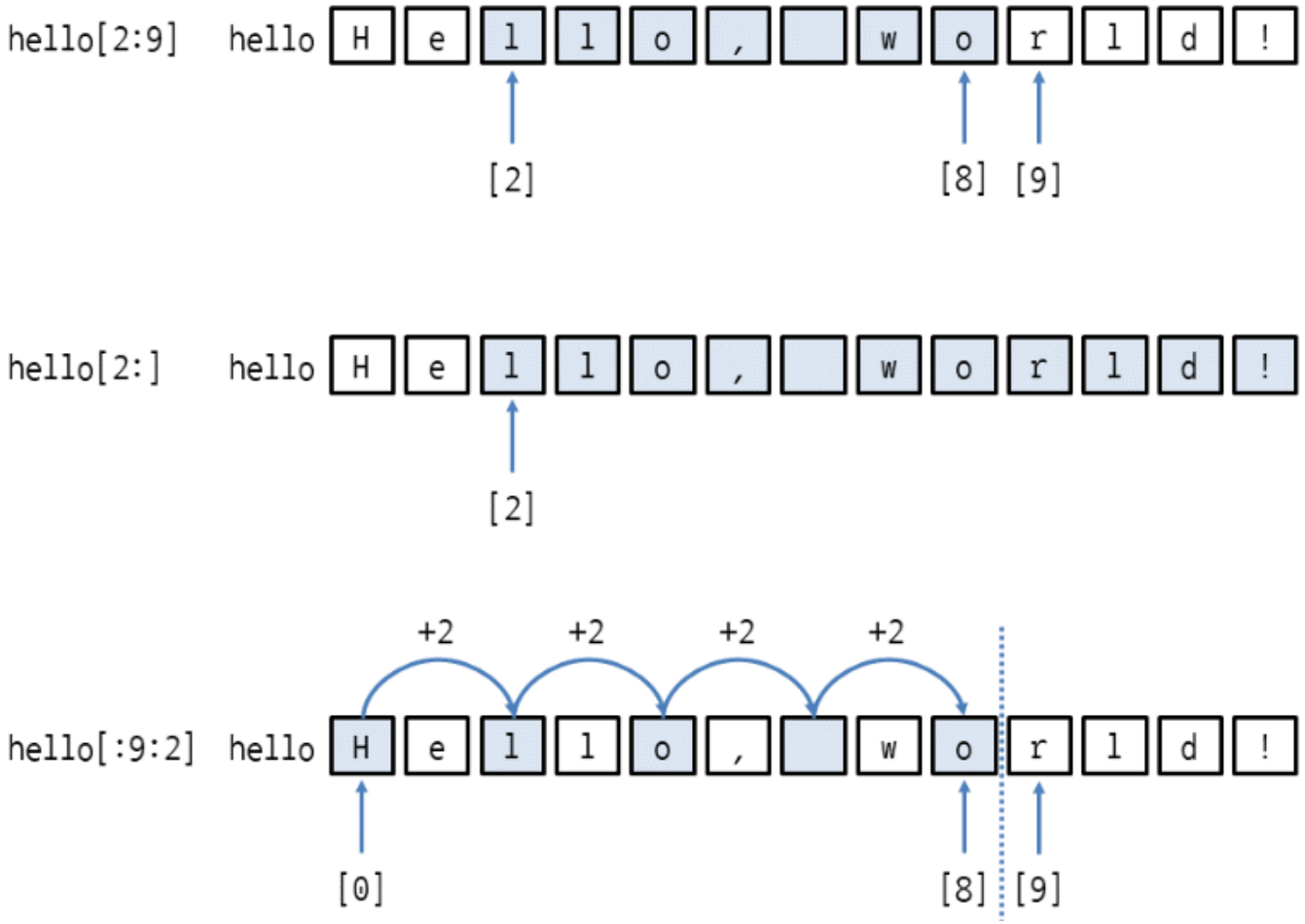
```
>>> list(r[:7:2])
[0, 2, 4, 6]
```

  - 문자열도 시퀀스 자료형이므로 슬라이스를 사용할 수 있음
  - 문자열은 문자 하나가 요소이므로 문자 단위로 잘라서 새 문자열을 생성

- 문자열[시작인덱스:끝인덱스]
- 문자열[시작인덱스:끝인덱스:인덱스증가폭]

```
>>> hello = 'Hello, world!'
>>> hello[2:9]    # 인덱스 2부터 인덱스 8까지 잘라서 문자열을 만듦
'llo, wo'
>>> hello[2:]     # 인덱스 2부터 마지막 요소까지 잘라서 문자열을 만듦
'llo, world!'
>>> hello[:9:2]   # 문자열의 처음부터 인덱스를 2씩 증가시키면서 인덱스 8까지 잘라서 문자열을 만듦
'Hlo o'
```

- 문자열에 슬라이스 사용하기



- **슬라이스에 요소 할당하기**
  - 시퀀스 객체는 슬라이스로 범위를 지정하여 여러 요소에 값을 할당할 수 있음

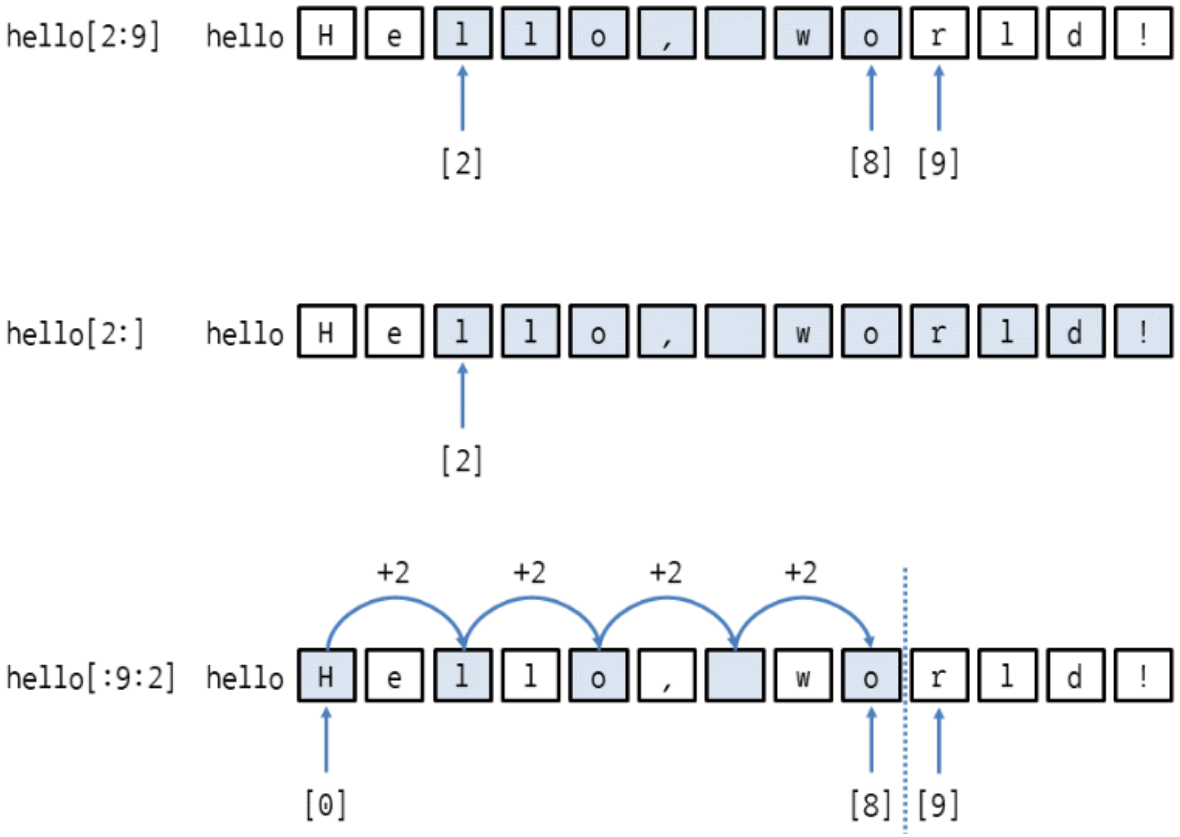
• 시퀀스객체[시작인덱스:끝인덱스] = 시퀀스객체

  - 리스트를 만든 뒤 특정 범위의 요소에 값을 할당해보자

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[2:5] = ['a', 'b', 'c']    # 인덱스 2부터 4까지 값 할당
>>> a
[0, 10, 'a', 'b', 'c', 50, 60, 70, 80, 90]
```

  - 범위를 지정해서 요소를 할당했을 경우에는 원래 있던 리스트가 변경되며 새 리스트는 생성되지 않음

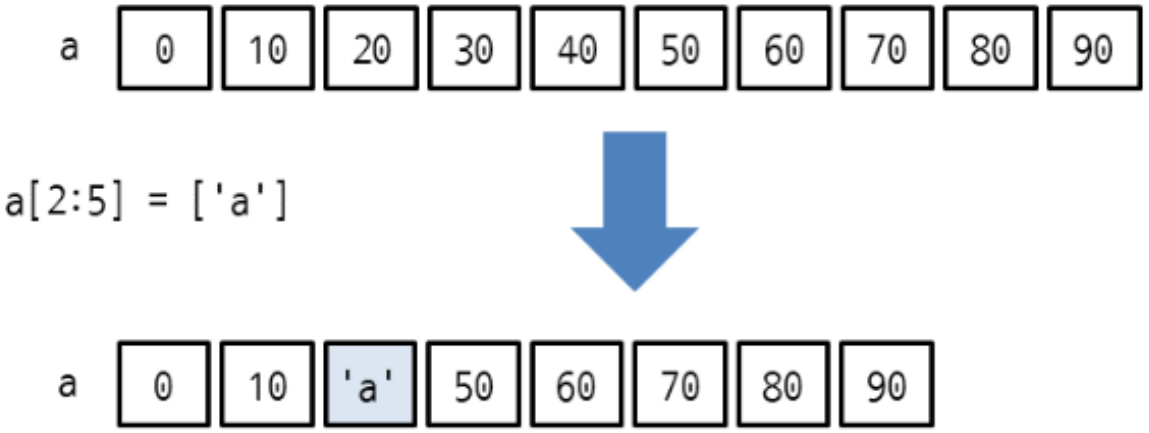
- 슬라이스에 요소 할당하기



특정 범위에 요소 할당하기

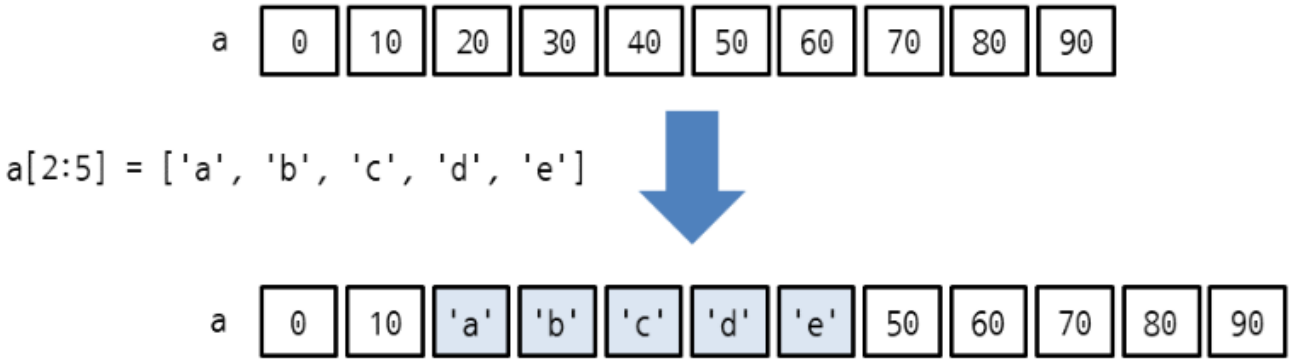
- **슬라이스에 요소 할당하기**
  - a[2:5] = ['a', 'b', 'c']는 슬라이스 범위와 할당할 리스트의 요소 개수를 정확히 맞추었지만, 사실 개수를 맞추지 않아도 상관없음
  - 요소 개수를 맞추지 않아도 알아서 할당됨
  - 할당할 요소 개수가 적으면 그만큼 리스트의 요소 개수도 감소

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[2:5] = ['a']    # 인덱스 2부터 4까지에 값 1개를 할당하며 요소의 개수가 줄어듦
>>> a
[0, 10, 'a', 50, 60, 70, 80, 90]
```



- **슬라이스에 요소 할당하기**
  - 할당할 요소 개수가 많으면 그만큼 리스트의 요소 개수도 늘어남

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[2:5] = ['a', 'b', 'c', 'd', 'e'] # 인덱스 2부터 4까지 값 5개를 할당하여 요소의 개수가 늘어남
>>> a
[0, 10, 'a', 'b', 'c', 'd', 'e', 50, 60, 70, 80, 90]
```

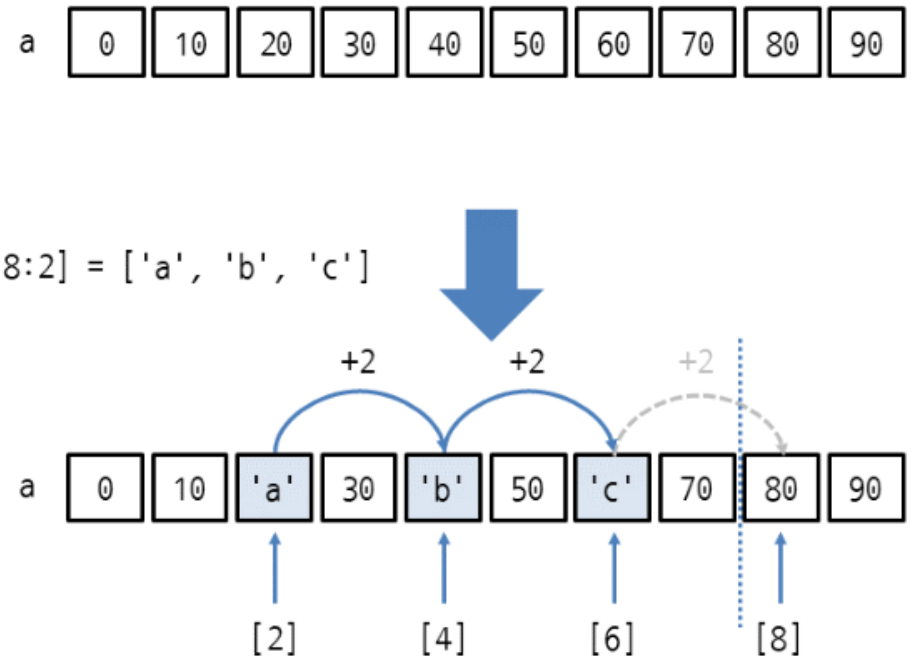


슬라이스 범위와 할당할 요소의 개수가 다를 때

- **슬라이스에 요소 할당하기**
  - 인덱스 증가폭을 지정하여 인덱스를 건너뛰면서 할당해보자
    - 시퀀스객체[시작인덱스:끝인덱스:인덱스증가폭] = 시퀀스객체

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[2:8:2] = ['a', 'b', 'c']    # 인덱스 2부터 2씩 증가시키면서 인덱스 7까지 값 할당
>>> a
[0, 10, 'a', 30, 'b', 50, 'c', 70, 80, 90]
```

  - a[2:8:2] = ['a', 'b', 'c']와 같이 인덱스 2부터 2씩 증가시키면서 7까지 'a', 'b', 'c'를 할당함



- 슬라이스에 요소 할당하기
  - 인덱스 증가폭을 지정했을 때는 슬라이스 범위의 요소 개수와 할당할 요소 개수가 정확히 일치해야 함

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> a[2:8:2] = ['a', 'b']
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    a[2:8:2] = ['a', 'b']
ValueError: attempt to assign sequence of size 2 to extended slice of size 3
```

- 슬라이스에 요소 할당하기
  - 튜플, range, 문자열은 슬라이스 범위를 지정하더라도 요소를 할당할 수 없음

```
>>> b = (0, 10, 20, 30, 40, 50, 60, 70, 80, 90)
>>> b[2:5] = ('a', 'b', 'c')
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    b[2:5] = ('a', 'b', 'c')
TypeError: 'tuple' object does not support item assignment
>>> r = range(10)
>>> r[2:5] = range(0, 3)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    r[2:5] = range(0, 3)
TypeError: 'range' object does not support item assignment
>>> hello = 'Hello, world!'
>>> hello[7:13] = 'Python'
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    hello[7:13] = 'Python'
TypeError: 'str' object does not support item assignment
```

- **del로 슬라이스 삭제하기**
  - 슬라이스 삭제는 다음과 같이 del 뒤에 삭제할 범위를 지정해주면 됨

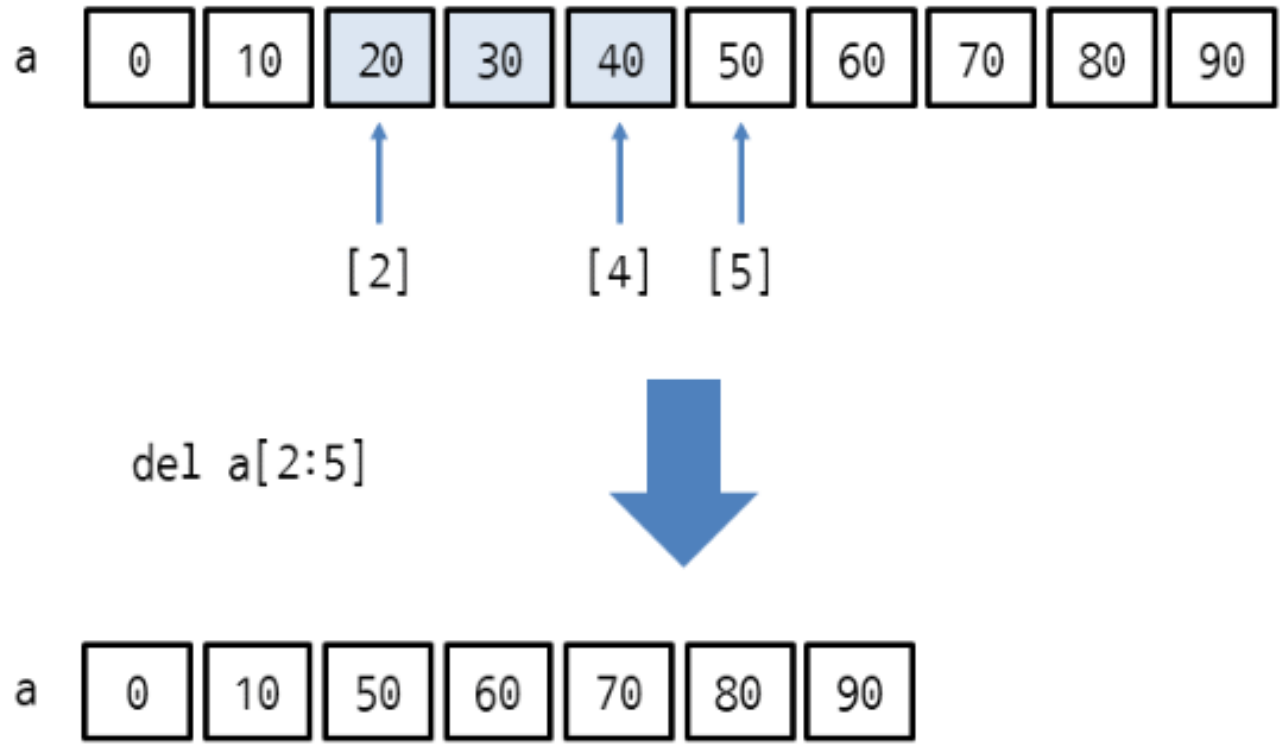
• del 시퀀스객체[시작인덱스:끝인덱스]

  - 리스트의 인덱스 2부터 4까지 요소를 삭제함

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> del a[2:5]    # 인덱스 2부터 4까지 요소를 삭제
>>> a
[0, 10, 50, 60, 70, 80, 90]
```

  - 리스트 a에서 인덱스 2, 3, 4인 요소 20, 30, 40이 삭제됨
  - **del로 요소를 삭제하면 원래 있던 리스트가 변경되며 새 리스트는 생성되지 않음**

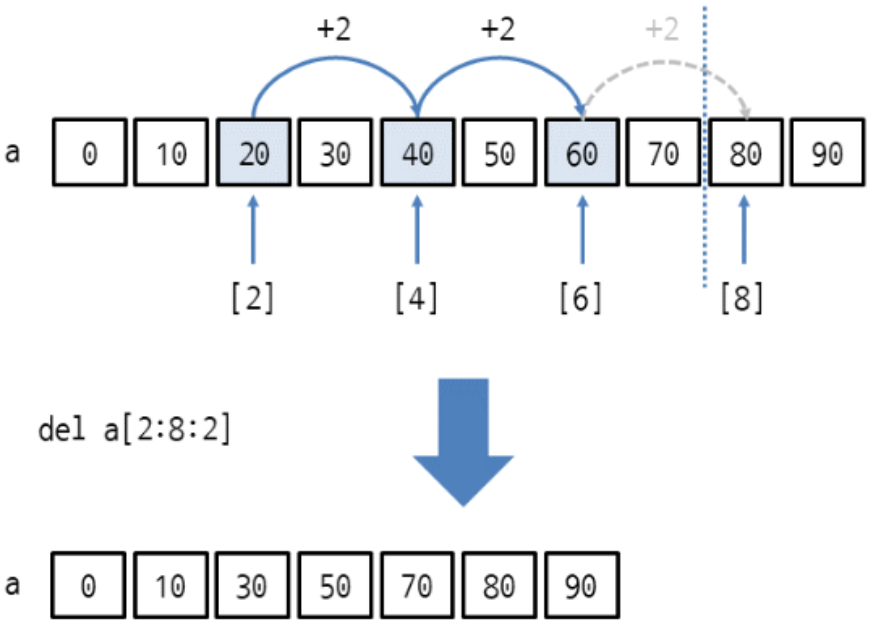
- **del로 슬라이스 삭제하기**



- **del로 슬라이스 삭제하기**
  - 인덱스 2부터 2씩 증가시키면서 인덱스 6까지 삭제함

```
>>> a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> del a[2:8:2]    # 인덱스 2부터 2씩 증가시키면서 인덱스 6까지 삭제
>>> a
[0, 10, 30, 50, 70, 80, 90]
```

  - 인덱스 2, 4, 6인 요소 20, 40, 60이 삭제됨



- **del로 슬라이스 삭제하기**
  - **튜플, range, 문자열은 del로 슬라이스를 삭제할 수 없음**

```
>>> b = (0, 10, 20, 30, 40, 50, 60, 70, 80, 90)
>>> del b[2:5]
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    del b[2:5]
TypeError: 'tuple' object does not support item deletion
>>> r = range(10)
>>> del r[2:5]
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    del r[2:5]
TypeError: 'range' object does not support item deletion
>>> hello = 'Hello, world!'
>>> del hello[2:9]
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    del hello[2:9]
TypeError: 'str' object does not support item deletion
```

  - **시퀀스 자료형의 인덱스가 0부터 시작한다는 점이 가장 중요함**