으뜸 파이썬

8강 리스트 자료형

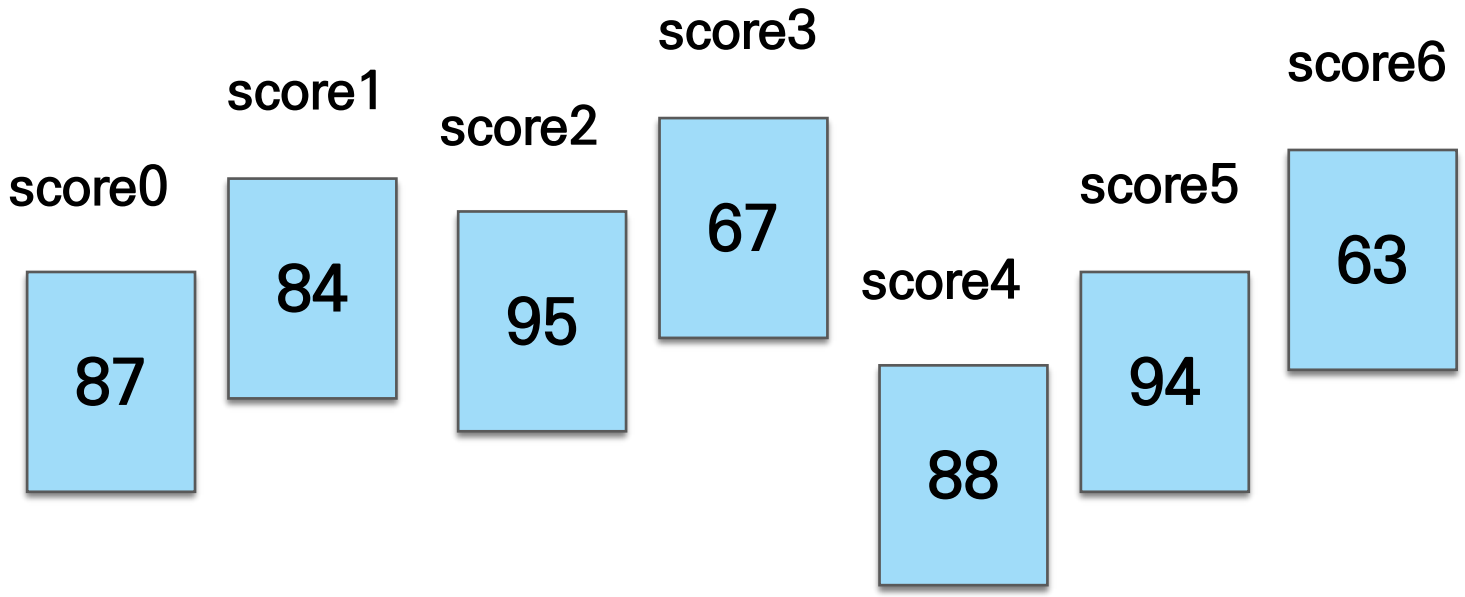
리스트 자료형의 필요성

- 아래와 같이 유사한 특성을 가진 변수가 여러개 필요한 경우를 생각해 보자.
- 데이터가 7개 있기 때문에 7개의 개별적인 변수를 선언하여 이 변수에 값을 담아서 사용해야 할 것이다. 개별 원소를 복사하고 조작하려면 많은 코딩이 필요하다.

대화창 실습 : 여러 개의 변수를 생성하고 사용하기

```
>>> score0 = 87
>>> score1 = 84
>>> score2 = 95
>>> score3 = 67
>>> score4 = 88
>>> score5 = 94
>>> score6 = 63
>>> print(score0, score3)
87 67
```

- 개별적인 변수의 생성과 값의 할당으로 이루어진 메모리 구조
- 많은 변수 이름이 필요할 것이다
- 변수의 합과 평균등 연산을 할 경우 코딩이 복잡하고 에러의 가능성이 높다



리스트list

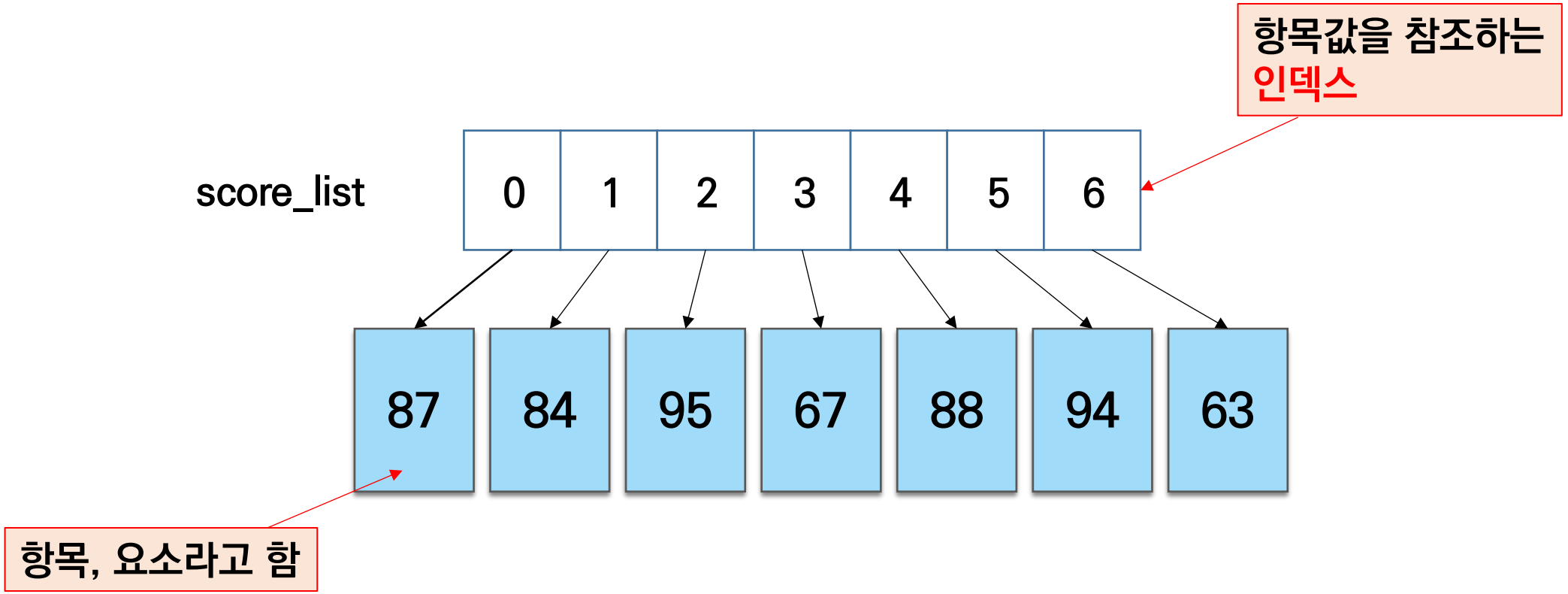
- 개별적인 값을 하나의 변수에 담아서 처리
  - 매번 변수의 이름을 작성하고 관리하는 것보다 편리하고 효율적
  - 한꺼번에 복사하고 조작할 수 있다.
- 항목item 또는 요소element
  - 리스트를 이루는 원소로 쉼표로 구분된 자료 값

대화창 실습 : 수치 값을 가진 리스트 만들기

```
>>> score_list = [87, 84, 95, 67, 88, 94, 63]
>>> score_list
[87, 84, 95, 67, 88, 94, 63]
>>> print(score_list[0], score_list[3])
87 67
```

요소의 참조는 리스트 이름과 인덱스를 사용한다

- 연속적인 자료값들은 score_list라는 변수와 인덱스를 통해서 참조하는 것이 가능
- 리스트는 대괄호 [ ]내에 쉼표를 이용하여 값을 구분
- '세 번째 변수'와 같이 위치를 지정해서 원하는 값을 불러오는 것도 가능



- fruits 리스트에 'banana', 'apple', 'orange', 'kiwi'의 4개 문자열 값을 한꺼번에 저장하고 출력
- mixed_list에 100, 200, 400과 같은 정수 값과 ‘apple'이라는 문자열 값을 동시에 리스트에 저장하고 출력

대화창 실습 : 문자열과 복합 자료값을 가진 리스트 만들기

```
>>> fruits = ['banana', 'apple', 'orange', 'kiwi']   # 문자열을 가지는 리스트
>>> fruits
['banana', 'apple', 'orange', 'kiwi']
>>> mixed_list = [100, 200, 'apple', 400]
>>> mixed_list
[100, 200, 'apple', 400]
```

파이썬의 리스트는 정수, 실수, 문자열등 서로 다른 자료형을 가질 수 있다

range()나 문자열을 이용하여 리스트 만들기

- range(1, 10)이라는 함수를 통해 1부터 9까지의 숫자의 열sequence을 얻은 후 이 열을 원소로 가지는 리스트를 list() 함수를 통해 생성

대화창 실습 : 다양한 방법으로 리스트 만들기

```
>>> list1 = list()                 # 빈 리스트 생성하기 1
>>> list2 = []                     # 빈 리스트 생성하기 2
>>> list3 = list((1, 2, 3))        # 튜플로부터 리스트 생성
>>> list3
[1, 2, 3]
>>> list4 = list(range(1, 10))     # range() 함수로부터 리스트 생성
>>> list4
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list5 = list('ABCDEF')         # 문자열로부터 리스트 생성
>>> list5
['A', 'B', 'C', 'D', 'E', 'F']
```

리스트의 인덱스

- 리스트list
  - 여러개의 항목을 포함할 수 있는 파이썬 자료구조로 내부 항목을 교체할 수 있다.
  - 또한 저장된 항목들 중 필요한 값을 추출하는 기능을 지원한다
- 인덱스index
  - 리스트의 항목 값을 가리키는 숫자
  - n개의 항목을 가진 리스트의 인덱스는 0부터 n-1 까지 증가
- 인덱싱indexing
  - 항목의 인덱스를 이용하여 자료값에 접근

n_list 리스트

- 리스트에서 항목 위치는 제일 첫 항목의 인덱스가 0이 되며 마지막 항목은 n-1

대화창 실습 : 6개의 원소를 가지는 리스트 만들기

```
>>> n_list = [11, 22, 33, 44, 55, 66]
>>> n_list
[11, 22, 33, 44, 55, 66]
>>> len(n_list)  # 리스트의 요소의 개수를 구하는 함수
6
>>> n_list[0]    # 리스트의 첫 번째 항목의 인덱스는 0이다.
11
>>> n_list[1]    # 리스트의 두 번째 항목의 인덱스는 1이다.
22
```

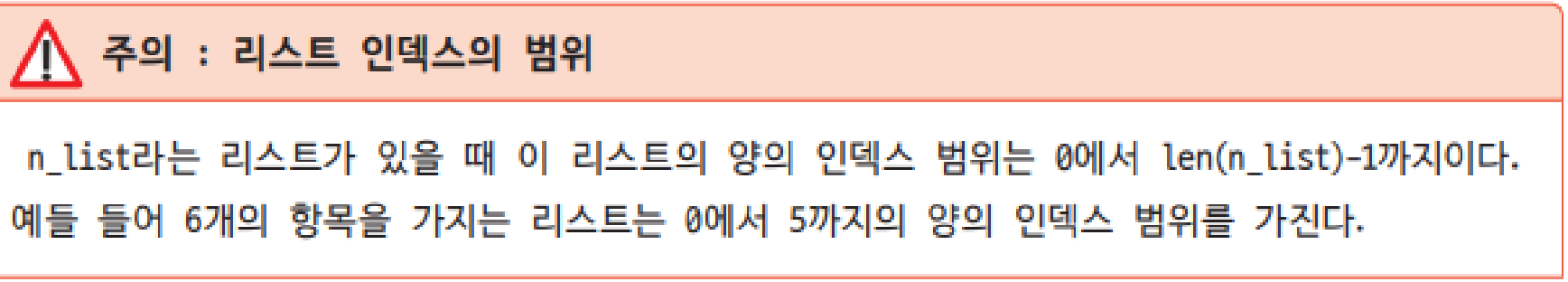
index	[0]	[1]	[2]	[3]	[4]	[5]
n_list =	11	22	33	44	55	66

n_list[0] = 11
n_list[1] = 22
n_list[2] = 33
n_list[3] = 44
n_list[4] = 55
n_list[5] = 66

- 인덱스 값에 최대 인덱스 값보다 더 큰 값을 넣으면 안 됨
- 최대 인덱스는 len(n_list)-1

대화창 실습 : 리스트 인덱스를 통한 요소의 접근

```
>>> n_list = [11, 22, 33, 44, 55, 66]    # 6개의 요소를 가진 리스트
>>> n_list[5]                            # 리스트의 여섯 번째 요소 값
66
>>> n_list[6]                            # 리스트의 일곱 번째 요소 값은 존재하지 않음
...
IndexError: list index out of range
```



음수 인덱스

대화창 실습 : 리스트의 음수 인덱스 사용법

```
>>> n_list = [11, 22, 33, 44, 55, 66]
>>> n_list[-1]       # 리스트의 마지막 요소 값
66
>>> n_list[-2]
55
>>> n_list[-3]
44
```

- 리스트의 제일 마지막 원소의 항목은 [-1 + len(n_list)]
- 첫 항목은 [-len(n_list) + len(n_list)]=[0]
- 파이썬 리스트는 음수 인덱스를 이용하여 원소들을 참조 가능
- 제일 마지막 원소로부터 -1, -2, -3과 같이 -1씩 감소하면서 인덱싱

음수 index	[-6]	[-5]	[-4]	[-3]	[-2]	[-1]
n_list =	11	22	33	44	55	66

```
n_list[-1] = 66
n_list[-2] = 55
n_list[-3] = 44
n_list[-4] = 33
n_list[-5] = 23
n_list[-6] = 11
```

[그림 5-4] n_list의 원소와 음수 인덱스

LAB 5-2 : 리스트의 생성과 인덱싱

1. 2부터 10까지의 수중에서 소수를 원소를 가지는 prime_list라는 리스트를 생성하여라. 그리고 이 리스트의 가장 첫 원소를 리스트 인덱싱을 이용하여 다음과 같이 출력하여라.

```
prime_list의 첫 원소 : 2
```

2. 문제 1.의 prime_list의 가장 마지막 원소를 양수 인덱스를 사용하여 출력하여라.

```
prime_list의 마지막 원소 : 7
```

3. 문제 1.의 prime_list의 가장 마지막 원소를 음수 인덱스를 사용하여 출력하여라.

```
prime_list의 마지막 원소 : 7
```

4. 'Korea', 'China', 'Russia', 'Malaysia'를 원소로 가지는 nations라는 리스트를 생성하여라. 그리고 이 리스트의 가장 첫 원소를 print() 함수를 이용하여 출력하여라.

```
nations의 첫 원소 : Korea
```

5. 4번에서 생성한 nations 리스트의 가장 마지막 원소를 음수 인덱스를 사용하여 출력하여라.

```
nations의 마지막 원소 : Malaysia
```

6. 4번에서 생성한 nations 리스트의 가장 마지막 원소를 len(nations)-1 인덱스를 사용하여 출력하여라.

```
nations의 마지막 원소 : Malaysia
```

리스트의 항목의 추가와 삭제

- 기존의 리스트에 원하는 항목을 추가 또는 삭제 가능
- append() 메소드
  - 이미 존재하는 리스트의 끝에 새로운 항목을 삽입하는 기능

대화창 실습 : 리스트의 append() 메소드를 사용한 항목의 추가

```
>>> a_list = ['a', 'b', 'c', 'd', 'e']
>>> a_list.append('f')          # 'f' 항목 추가
>>> a_list
['a', 'b', 'c', 'd', 'e', 'f']
>>> n_list = [10, 20, 30, 40]
>>> n_list.append(50)           # 50 항목 추가
>>> n_list
[10, 20, 30, 40, 50]
```

리스트내의 항목을 지우는 방법

- 파이썬의 키워드 del 사용(명령어임)
- 리스트 클래스에 있는 remove() 라는 메소드 사용
- pop() 메소드를 사용
  - 리스트의 특정 위치에 있는 항목을 삭제함과 동시에 이 항목을 반환

del 키워드로 삭제하는 방법

코드 5-1 : del 명령어를 통한 리스트의 항목 삭제

list_del_ex.py

```
n_list = [11, 22, 33, 44, 55, 66]
print(n_list)        # 전체 항목 출력


del n_list[3]        # 네 번째 항목(44) 삭제
print(n_list)
```

실행결과

```
[11, 22, 33, 44, 55, 66]
[11, 22, 33, 55, 66]
```

- 지정된 인덱스에 위치한 항목을 삭제
- del 44와 같은 방법으로 삭제할 수 없다

remove 메소드로 삭제하는 방법

코드 5-2 : remove() 메소드를 이용한 리스트의 항목 삭제

list_remove_ex1.py

```
n_list = [11, 22, 33, 44, 55, 66]
print(n_list)


n_list.remove(44)          # 44라는 값을 가진 항목 삭제
print(n_list)
```

실행결과

```
[11, 22, 33, 44, 55, 66]
[11, 22, 33, 55, 66]
```

- list가 가진 메소드로 특정한 값을 리스트의 항목에서 삭제
- .remove(44) 와 같은 방법을 사용할 수 있다

remove 메소드의 문제점

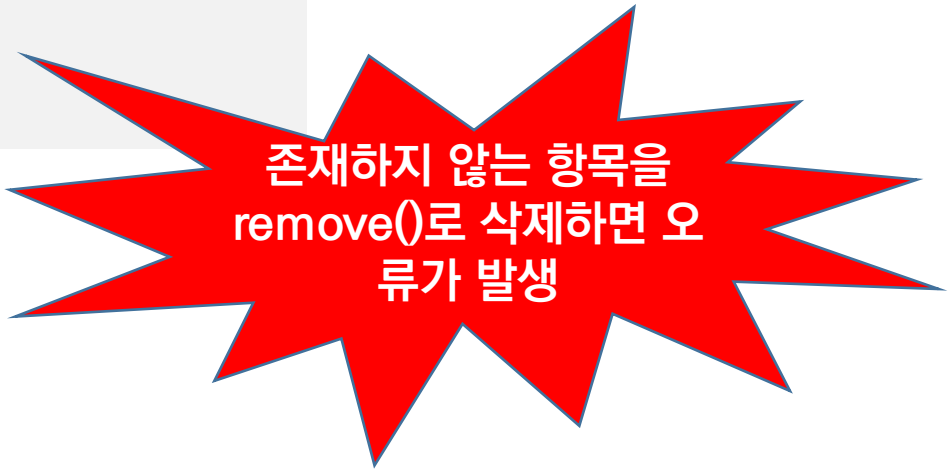
코드 5-3 : 리스트 내부에 존재하지 않는 항목을 삭제하는 경우

list_remove_ex2.py

```
n_list = [11, 22, 33, 44, 55, 66]
print(n_list)


n_list.remove(88)
print(n_list)
```

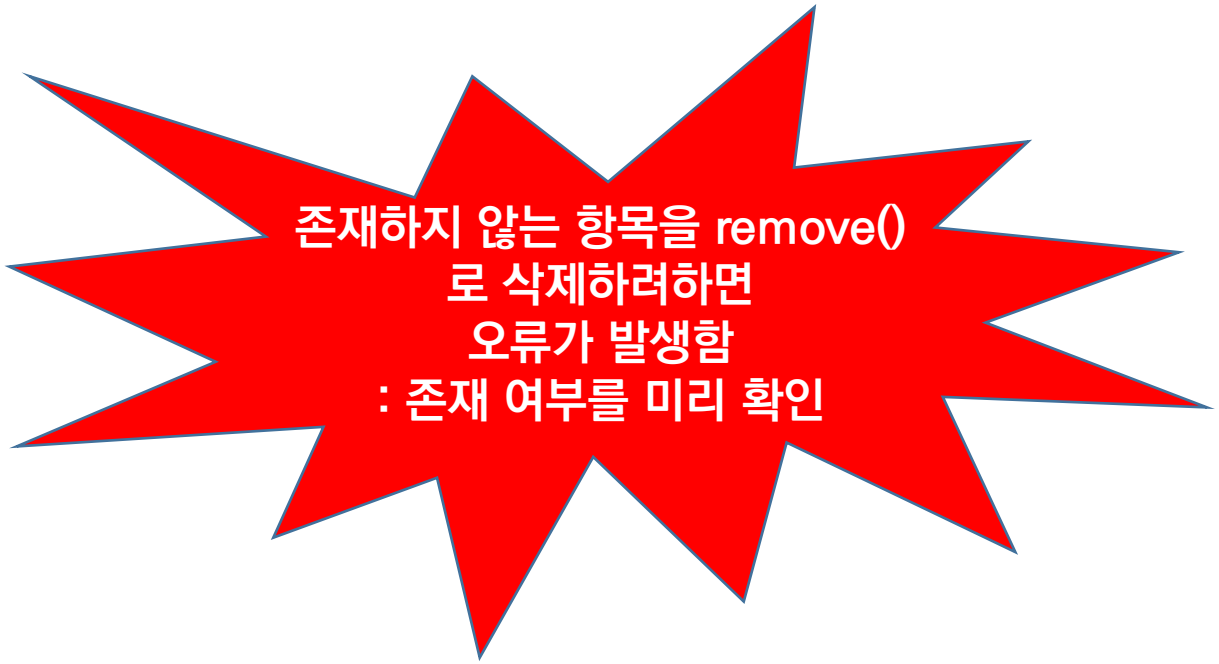
실행결과

```
...
ValueError: list.remove(x): x not in list
```



멤버 연산자: in, not in

- 멤버 연산자는 참(True) 혹은 거짓(False)를 반환하는 연산자
- 특정한 원소가 문자열, 리스트, 튜플과 같은 자료구조의 내부에 포함되어 있는가를 검사하는 용도로 사용됨



in 연산으로 존재여부를 확인하고 True인 경우에만 삭제할 경우는 오류가 생기지 않음

대화창 실습 : 멤버 연산자 in과 리스트

```
>>> a_list = [10, 20, 30, 40]
>>> 10 in a_list                # 리스트의 요소로 10이 있으므로 참
True
>>> 50 in a_list                # 리스트의 요소로 50이 없으므로 거짓
False
>>> 10 not in a_list
False
>>> 50 not in a_list
True
```

in 연산자의 사용

코드 5-4 : 리스트 내부에 값이 존재하는가를 확인하는 기능

value_in_list.py

```
n_list = [11, 22, 33, 44, 55, 66]


print(88 in n_list)        # 88은 n_list에 없음
print(55 in n_list)        # 55는 n_list에 있음
```

실행결과

```
False
True
```

in 연산을 사용하여 오류 발생을 미리 방지

- 리스트 내부에 지우고자 하는 값이 있는 지 확인

코드 5-5 : in 연산자를 이용한 안전한 원소 삭제

list_remove_ex3.py

```
n_list = [11, 22, 33, 44, 55, 66]
if (55 in n_list) :       # 리스트의 요소로 55가 있을 경우
    n_list.remove(55)     # 리스트에서 55를 삭제함
if (88 in n_list) :       # 리스트의 요소로 88이 있을 경우
    n_list.remove(88)     # 리스트에서 88을 삭제함
print(n_list)
```

실행결과

```
[11, 22, 33, 44, 66]
```

리스트에 적용되는 내장함수

- 4장의 내장함수에서 다루었음
- min(), max(), sum()과 같은 파이썬 내장함수의 인자로 리스트를 넘겨주면 각각 리스트 안의 최솟값, 최댓값, 합 연산 가능
- len() 함수는 리스트 내 항목의 개수를 반환

대화창 실습 : 리스트와 내장함수 min(), max(), sum()

```
>>> list1 = [20, 10, 40, 50, 30]
>>> min(list1)   # 리스트의 원소들 중 가장 작은 원소를 구한다.
10
>>> max(list1)   # 리스트의 원소들 중 가장 큰 원소를 구한다.
50
>>> sum(list1)   # 리스트내의 원소의 합을 구한다.
150
```

- list1은 5개의 항목을 가짐
- len(list1)은 리스트 내 항목 개수 반환

파이썬은 좋은 내장함수를 많이 가지고 있음
min, max, sum 내장함수는 리스트를 인자로 가질 수 있음

- **문자열이 들어있는 리스트 변수에도 min(), max() 함수 동작**
- **sum() 함수는 문자열에 대해서는 동작하지 않음**

대화창 실습 : 문자열 리스트와 내장함수 min(), max()

```
>>> fruits = ['banana', 'orange', 'apple', 'kiwi']
>>> min(fruits)      # 영어사전 순서로 가장 앞에 있는 단어를 반환
'apple'
>>> max(fruits)      # 영어사전 순서로 가장 뒤에 있는 단어를 반환
'orange'
```

- **min()과 max()는 한글 문자열을 요소로 가지는 리스트에도 동작**

대화창 실습 : 한글 문자열 리스트와 내장함수 min(), max()

```
>>> k_fruits = ['사과', '오렌지', '포도', '바나나']
>>> min(k_fruits)
'바나나'
>>> max(k_fruits)
'포도'
```

LAB 5-4 : 리스트의 min()과 max(), sum(), len() 함수

1. 1부터 10까지의 수중에서 소수 원소를 가지는 prime_list라는 리스트를 생성하고 이들중 최솟값, 최댓값을 다음과 같이 출력하시오. min()과 max(), sum(), len() 내장함수를 사용하여 다음과 같이 출력하시오.

```
1에서 10까지의 소수 : [2, 3, 5, 7]
최솟값 : 2
최댓값 : 7
합계 : 17
평균 : 4.25
```

2. 'Korea', 'China', 'Russia', 'Malaysia' 원소를 가지는 nations라는 리스트가 있다. 이들 나라 중에서 사전 순서로 가장 먼저 나오는 나라와 가장 뒤에 나오는 나라를 다음과 같이 출력하시오.

```
국가 목록 : ['Korea', 'China', 'Russia', 'Malaysia']
사전에 가장 먼저 나오는 나라 : China
사전에 가장 뒤에 나오는 나라 : Russia
```

리스트의 메소드

- 정렬sorting 메소드인 sort()
  - 디폴트로 오름차순 정렬을 수행
  - 리스트 변수 뒤에 마침표(.)와 sort()를 적음

- 오름차순 정렬ascending order sorting
  - 값이 증가하는 정렬

- 내림차순 정렬descending order sorting
  - 값이 감소하는 정렬

sort() 메소드 실습

대화창 실습 : 리스트와 sort() 메소드

```
>>> list1 = [20, 10, 40, 50, 30]
>>> list1.sort()                    # list1의 원소를 오름차순 정렬
>>> list1
[10, 20, 30, 40, 50]
>>> list1.sort(reverse = True)      # list1의 원소를 내림차순 정렬
>>> list1
[50, 40, 30, 20, 10]
```

리스트가 제공하는 메소드

[표 5-1] 리스트의 메소드와 하는 일

메소드	하는 일
index(x)	원소 x를 이용하여 위치를 찾는 기능을 한다.
append(x)	원소 x를 리스트의 끝에 추가한다.
count(x)	리스트 내에서 object 원소의 개수를 반환한다.
extend([x1, x2])	[x1, x2] 리스트를 리스트에 삽입한다.
insert(index, x)	원하는 index 위치에 x를 추가한다.
remove(x)	x 원소를 리스트에서 삭제한다.
pop(index)	index 위치의 원소를 삭제한 후 반환한다. 이때 index는 생략될 수 있으며 이 경우 리스트의 마지막 원소를 삭제하고 이를 반환한다.
sort()	값을 오름차순 순서대로 정렬한다. reverse 인자의 값이 True 이면 내림차순으로 정렬한다.
reverse()	리스트를 원래 원소들의 역순으로 만들어 준다.

리스트와 연산

- 리스트 사이에는 더하기 연산자 사용 가능
- 빼기 연산자나 곱하기, 나누기 연산자는 사용 불가

코드 5-6 : 두 리스트를 합치는 연산자 +

list_plus_ex1.py

```
list1 = [11, 22, 33, 44]
list2 = [55, 66]


print(list1)
print(list1 + list2)
```

실행결과

```
[11, 22, 33, 44]
[11, 22, 33, 44, 55, 66]
```

- 리스트 list1의 항목에 list2 항목을 추가
- 결과를 다른 리스트에 할당해야 list1의 상태변화가 있음

코드 5-7 : 두 리스트를 합치는 연산자 +를 이용하여 그 결과를 저장함

list_plus_ex2.py

```
list1 = [11, 22, 33, 44]
list2 = [55, 66]
list3 = list1 + list2
print(list3)
```

실행결과

```
[11, 22, 33, 44, 55, 66]
```

- **list1과 list2를 더한 결과를 list3에 할당해 줌**

리스트의 곱셈 연산

대화창 실습 : * 연산자를 이용한 반복 리스트

```
>>> list1 = [1, 2, 3, 4]
>>> list1 * 2
[1, 2, 3, 4, 1, 2, 3, 4]
>>> list2 = list1 * 3        # list1 * 3의 결과를 list2에 할당
>>> list2
[1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

• 다음의 경우 오류 반환

대화창 실습 : 문법오류를 유발하는 두 리스트의 * 연산(-, / 연산도 사용불가)

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = [10, 20, 30]
>>> list1 * list2
...
TypeError: can't multiply sequence by non-int of type 'list'
```

리스트의 내용 갱신을 위한 방법

- for 문을 통해 리스트의 항목을 하나하나씩 나열 또는 연산 가능

코드 5-8 : 리스트 요소들을 하나하나 방문하며 10을 곱하는 기능

list_element_ex1.py

```
list1 = [10, 20, 30, 40, 50]
i = 0
for n in list1:
    list1[i] = n * 10
    i = i + 1

print(list1)
```

실행결과

```
[100, 200, 300, 400, 500]
```

모든 원소에 10을 곱하여 보기

코드 5-9 : 리스트의 축약 표현을 사용하여 10을 곱하는 기능

list_element_ex2.py

```
list1 = [10, 20, 30, 40, 50]
list1 = [n * 10 for n in list1]
print(list1)
```

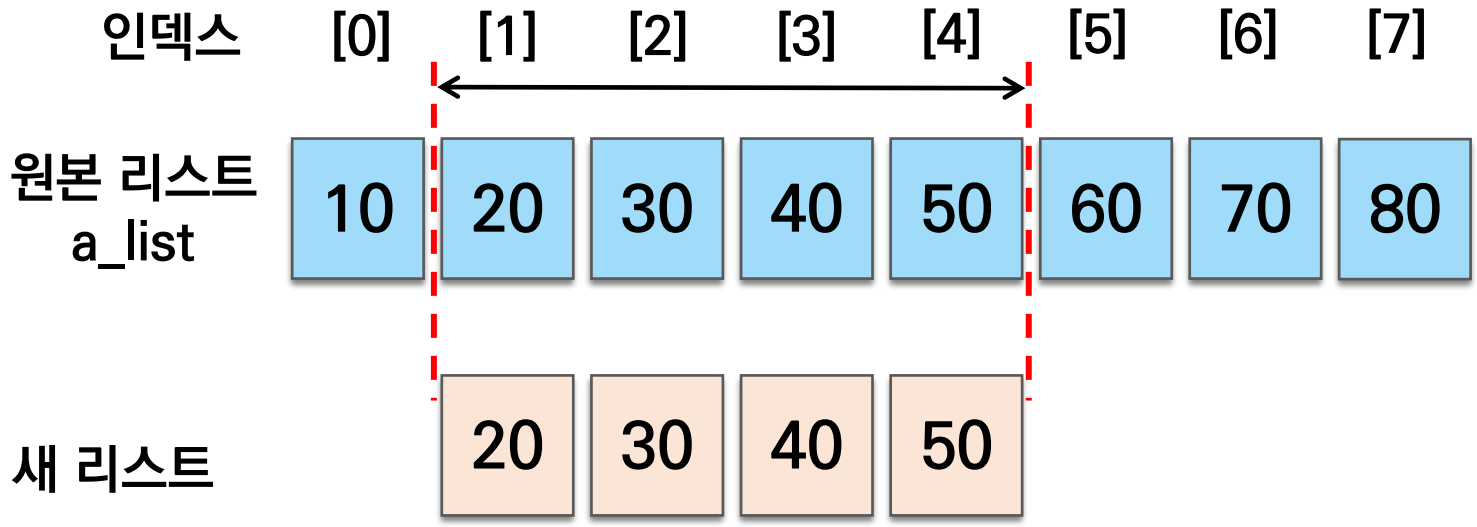
고급 기능으로 따로 학습이 필요

실행결과

```
[100, 200, 300, 400, 500]
```

리스트의 슬라이싱

- 슬라이싱slicing
  - 리스트내의 항목을 특정한 구간별로 선택하여 잘라내는 기능
  - 구간을 명시하기 위해 리스트_이름[start : end] 문법 사용
  - end-1까지의 항목을 새 리스트에 삽입

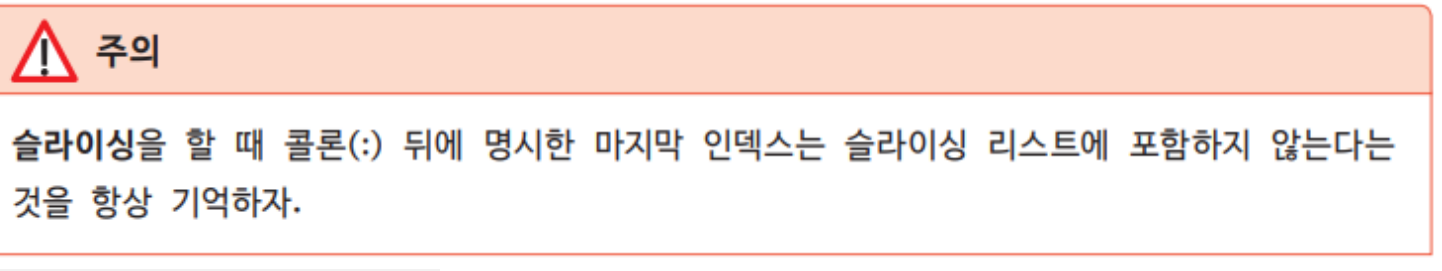


a_list[1:5]의 결과

대화창 실습 : 리스트와 슬라이싱

```
>>> a_list = [10, 20, 30, 40, 50, 60, 70, 80]
>>> a_list[1:5]
[20, 30, 40, 50]
>>> a_list[0:1]
[10]
>>> a_list[0:2]
[10, 20]
>>> a_list[0:5]
[10, 20, 30, 40, 50]
>>> a_list[1:]
[20, 30, 40, 50, 60, 70, 80]
>>> a_list[:5]
[10, 20, 30, 40, 50]
```

- 콜론(:)을 사용하여 가져올 항목의 범위를 지정
- 시작 인덱스와 끝 인덱스는 생략가능



- **파이썬의 슬라이싱 기능은 음수 인덱스와 음수 스텝 값을 지원**

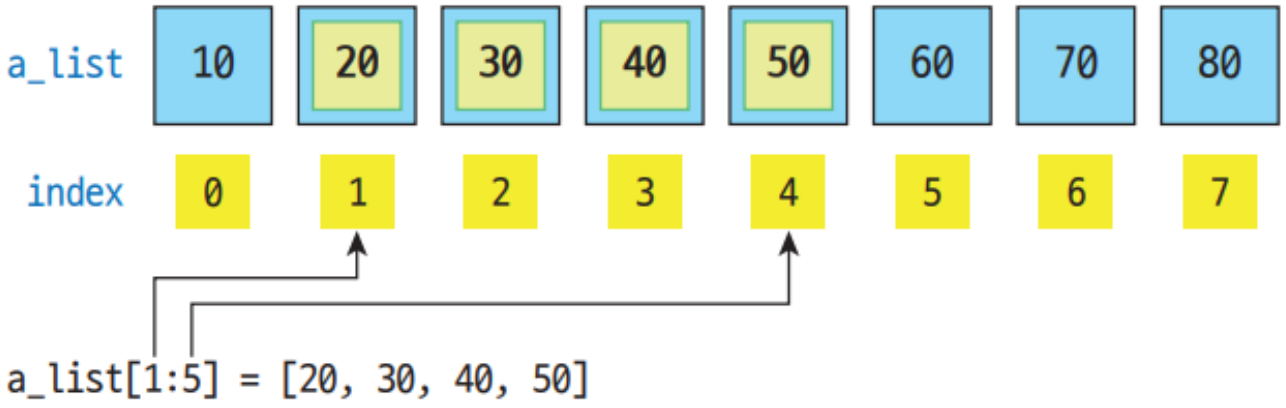
[표 5-2] 슬라이싱 문법과 하는 일

문법	하는 일
a_list[start:end]	start부터 (end-1)까지의 항목들을 슬라이싱(end 인덱스의 항목은 포함하지 않음)
a_list[start:]	start부터 리스트의 끝까지, 즉 뒷부분 모두를 슬라이싱
a_list[:end]	처음부터 end-1번째 인덱스 항목을 슬라이싱
a_list[:]	전체를 슬라이싱
a_list[start:end:step]	start부터 end-1까지를 step만큼 건너뛰며 슬라이싱
a_list[-2:]	뒤에서부터 두 개의 항목을 슬라이싱
a_list[:-2]	처음부터 끝의 두 개를 제외한 모든 항목을 슬라이싱
a_list[::-1]	모든 항목을 **역순으로 가져옴**
a_list[1::-1]	처음의 두 개 **항목을 역순으로 슬라이싱**

시작 인덱스와 마지막 인덱스가 명시된 리스트 슬라이싱

- a_list[1]부터 a_list[5-1]까지 항목을 가져온다.

```
>>> a_list = [10, 20, 30, 40, 50, 60, 70, 80]
>>> a_list[1:5]
[20, 30, 40, 50]
```

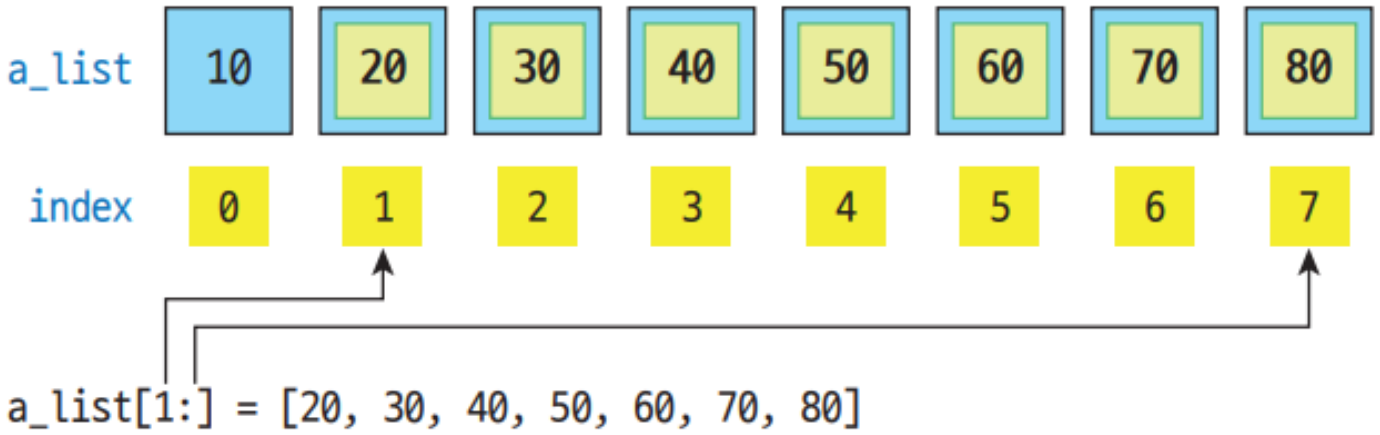


[그림 5-6] 시작 인덱스와 마지막 인덱스가 명시된 리스트 슬라이싱

마지막 슬라이싱 인덱스를 생략하는 경우

- 리스트의 마지막 항목까지 모두 가져옴

```
>>> a_list[1:]
[20, 30, 40, 50, 60, 70, 80]
```

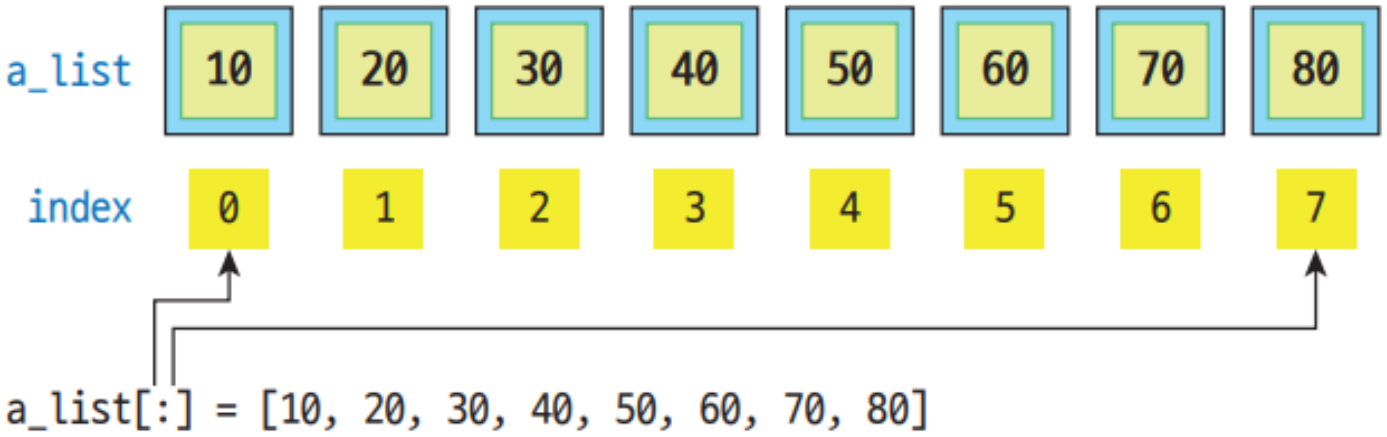


[그림 5-7] 마지막 슬라이싱 인덱스를 생략한 결과

시작 인덱스와 마지막 인덱스를 모두 생략

- 리스트의 **모든 항목**을 다 가져옴

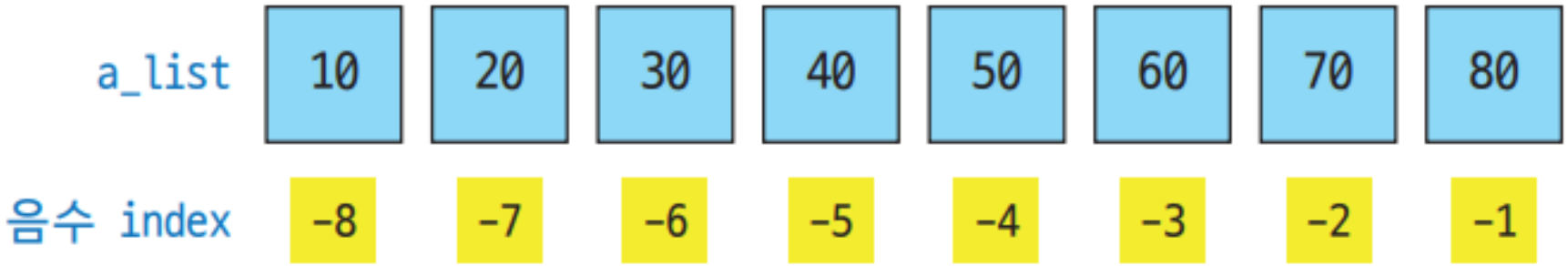
```
>>> a_list[:]
[10, 20, 30, 40, 50, 60, 70, 80]
```



[그림 5-8] 전체 슬라이싱 인덱스를 생략한 리스트의 슬라이싱

음수 인덱스를 사용한 슬라이싱

- 가장 끝 원소의 인덱스가 –1이 되며 그 앞의 원소가 –2, –3, …과 같이 부여됨

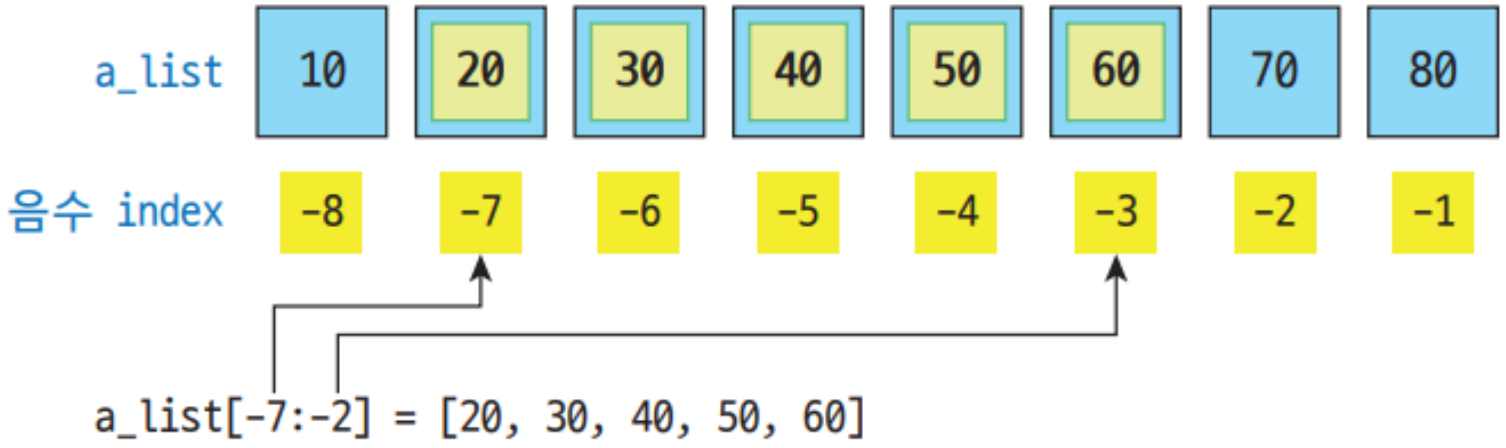


[그림 5-9] a_list와 음수 인덱스

음수 인덱스를 사용한 슬라이싱

- a_list[-7:-2]를 설정한다면 [그림 5-10]과 같이 [20, 30, 40, 50, 60] 항목을 포함하게 된다.

```
>>> a_list[-7:-2]
[20, 30, 40, 50, 60]
```

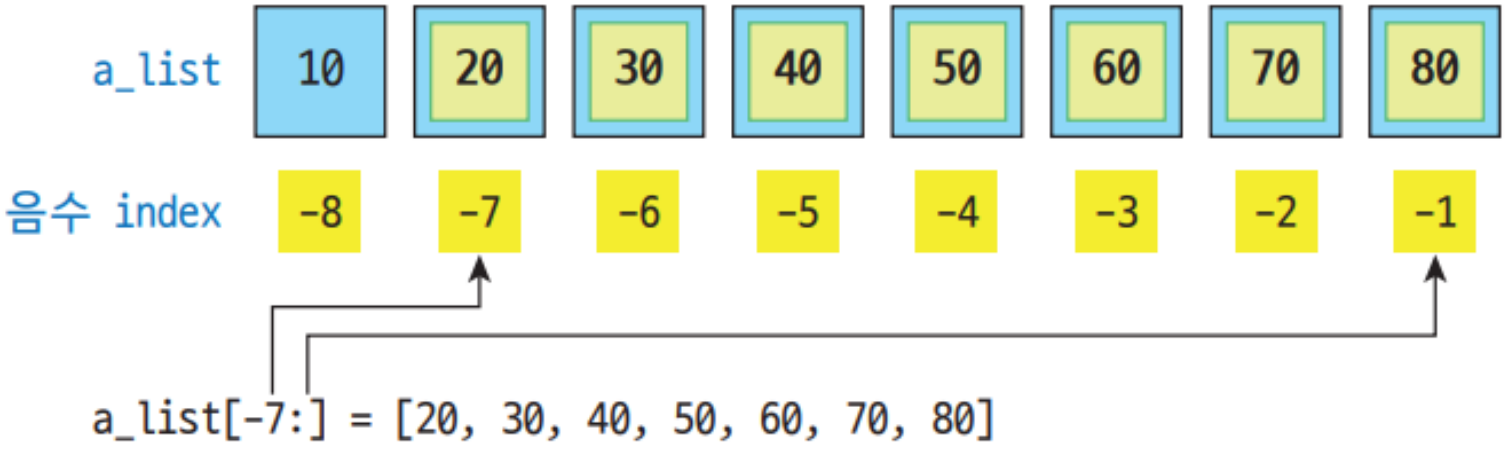


[그림 5-10] 음수 인덱스를 사용한 리스트 슬라이싱

음수 인덱스를 사용한 슬라이싱

- 마지막 인덱스를 생략할 경우 리스트의 마지막 항목까지 가져옴

```
>>> a_list[-7:]
[20, 30, 40, 50, 60, 70, 80]
```



[그림 5-11] 음수 인덱스 사용시의 리스트 슬라이싱(마지막 인덱스 생략)

Questions?

GUI 프로젝트

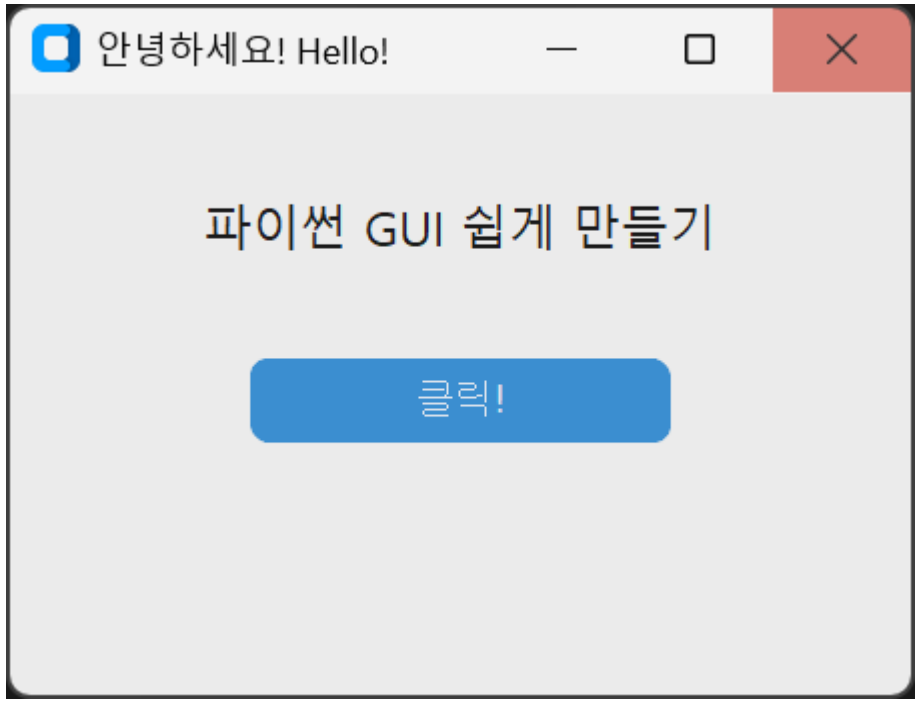
- Tkinter: built-in

```
import tkinter as tk


window = tk.Tk()
window.title("My First GUI")
label = tk.Label(window, text="Hello,
World!")
label.pack()
window.mainloop()
```

GUI 프로젝트

- Custom Tkinter: better look-and-feel
    - Pip install customtkinter

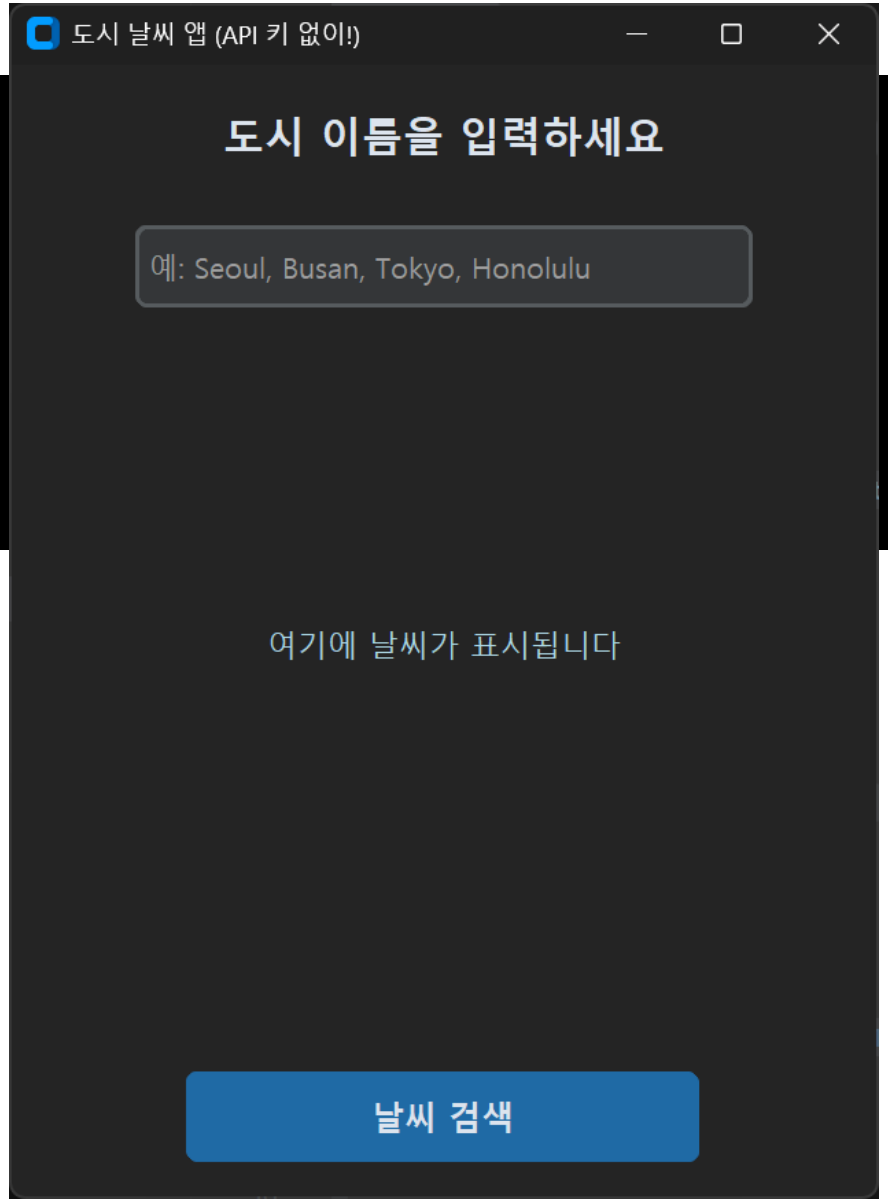
```python
import customtkinter as ctk


def change_label_text():
    label.configure(text="멋지네요! 😊")


app = ctk.CTk()
app.geometry("300x200")
app.title("안녕하세요! Hello!")


label = ctk.CTkLabel(app, text="파이썬 GUI 쉽게 만들기",
font=("맑은 고딕", 16))
label.pack(pady=30)


button = ctk.CTkButton(app, text="클릭!",
command=change_label_text)
button.pack()


app.mainloop()
```



```python
import customtkinter as ctk
import requests


# ===== GUI 설정 =====
ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")


app = ctk.CTk()
app.geometry("420x550")
app.title("도시 날씨 앱")

# 제목
title = ctk.CTkLabel(app, text="도시 이름을 입력하세요", font=("맑은 고딕", 20, "bold"))



# 입력창
city_entry = ctk.CTkEntry(
    app,
    placeholder_text="예: Seoul, Busan, Tokyo, Honolulu",
    width=300,
    height=40,
    font=("맑은 고딕", 14)
)
```

```python
# 결과 라벨
result_label = ctk.CTkLabel(
    app,
    text="여기에 날씨가 표시됩니다",
    font=("맑은 고딕", 15),
    wraplength=380,
    justify="center",
    text_color="lightblue"
)


# 로딩 메시지
loading_label = ctk.CTkLabel(app, text="", font=("맑은 고딕", 12), text_color="yellow")

# 검색 버튼
search_button = ctk.CTkButton(
    app,
    text="날씨 검색",
    width=250,
    height=45,
    font=("맑은 고딕", 16, "bold"),
    command=None
)
```

```python
title.pack(pady=20)
city_entry.pack(pady=10)
result_label.pack(pady=25, expand=True)
loading_label.pack(pady=5)
search_button.pack(pady=15)


# ===== 앱 실행 =====
app.mainloop()
```



```python
# ===== 날씨 가져오는 함수 (wttr.in 사용) =====
def get_weather():
    city = city_entry.get().strip()
    if not city:
        result_label.configure(text="도시 이름을 입력해주세요!", text_color="red")
        return

    loading_label.configure(text="날씨 불러오는 중...")
    app.update()

    # wttr.in 사용: 간단한 텍스트 형식
    url = f"https://wttr.in/{city}?format=%l:+%c+%t+%w+%p+%m"
    # %l: 위치, %c: 날씨 아이콘, %t: 온도, %w: 바람, %p: 강수량, %m: 달
    response = requests.get(url, timeout=10)
    if response.status_code == 200:
        weather_text = response.text.strip()
        # 예쁘게 포맷팅
        result_label.configure(
            text=f"🌍 {weather_text}",
            text_color="white",
            font=("맑은 고딕", 16)
        )
    else:
        result_label.configure(text="도시를 찾을 수 없어요", text_color="red")

    loading_label.configure(text="날씨 부르기 완료")
```

```python
# 검색 버튼
search_button = ctk.CTkButton(
    app,
    text="날씨 검색",
    width=250,
    height=45,
    font=("맑은 고딕", 16, "bold"),
    command=get_weather
)
```