

Python

리스트

Spring 2025



AI융합학과

Seongbok Baik

sbbaik@dju.kr

00 Text Book



교재명	으뜸 파이썬
저자	박동규, 강영민
출판사	생능출판사
발행년	2024.06.14



학습목표

- 리스트 자료형의 필요성과 개념에 대해서 이해한다.
- 리스트의 인덱싱을 비롯한 기본적인 사용법에 대해 이해하고 활용할 수 있다.
- 리스트의 연산에 대해 이해한다.
- 리스트의 다양한 기능을 내장함수를 이용하여 활용할 수 있다.
- 리스트에서 제공하는 여러 메소드들을 활용해 본다.
- 리스트의 슬라이싱을 통해 필요한 데이터를 추출할 수 있다.
- 다양한 슬라이싱 방법을 사용할 수 있다.
- 리스트와 튜플, 문자열 등 다양한 자료형에도 슬라이싱을 적용할 수 있다.

6.1 리스트 자료형의 필요성

- 리스트 변수는 여러 개의 값이나 변수를 한꺼번에 담을 수 있다.
- 데이터가 7개 있기 때문에 7개의 개별적인 변수를 선언하여 이 변수에 값을 담아서 사용해야 할 것이다. 개별 원소를 복사하고 조작하려면 번거롭다.



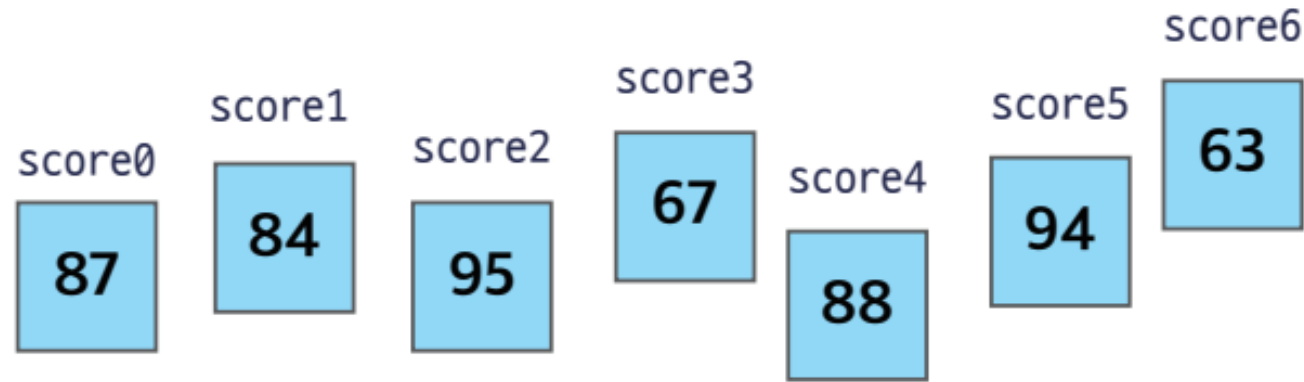
대화창 실습: 여러 개의 변수를 생성하고 사용하기

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

번거로운 작업:
데이터가 10,000개라면,
10,000개의 변수가 필요
하다.

6.1 리스트 자료형의 필요성

- 개별적인 변수의 생성과 값의 할당으로 이루어진 메모리 구조
- 많은 변수 이름이 필요할 것이다.
- 여러 곳의 메모리에 분산되어 저장되므로 연속적으로 읽을 수 없어 비효율적임
- 코딩이 복잡하고 에러의 가능성이 높아짐



[그림 6-1] 개별적인 변수인 score0부터 score6으로 구성된 데이터

6.1 리스트 자료형의 필요성

리스트 **list**

- 개별적인 값을 하나의 변수에 담아서 처리
 - 매번 변수의 이름을 작성하고 관리하는 것보다 편리하고 효율적
 - 한꺼번에 복사하고 조작할 수 있다.
- 항목 **item** 또는 요소 **element**
 - 리스트를 이루는 원소로 심표로 구분된 자료 값



대화창 실습: 수치 값을 가진 리스트 만들기

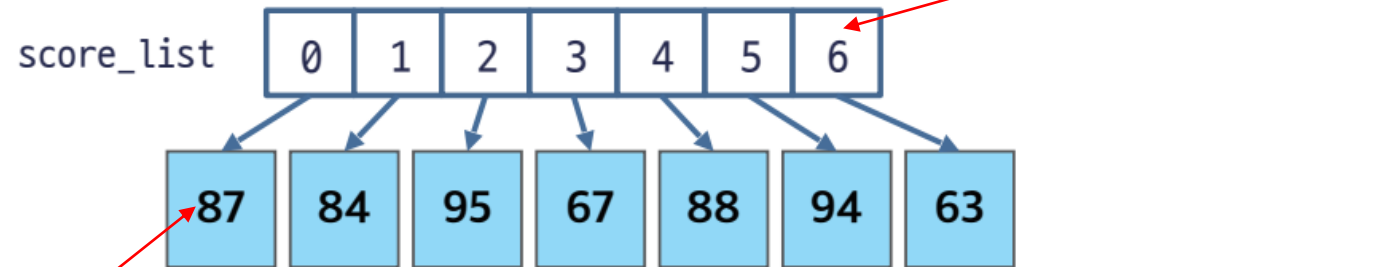
```
>>> score_list = [87, 84, 95, 67, 88, 94, 63]
>>> score_list
[87, 84, 95, 67, 88, 94, 63]
>>> print(score_list[0], score_list[3])
87 67
```

요소의 참조는 리스트 이름과 인덱스를 사용한다

6.1 리스트 자료형의 필요성

리스트 list

- 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



[그림 6-2] 리스트의 인덱스로 여러 개의 수치 자료값을 참조하는 모습

항목, 요소라고 함

6.1 리스트 자료형의 필요성

리스트 list

- 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'] # 문자열 원소 리스트
>>> fruits
['banana', 'apple', 'orange', 'kiwi']
>>> mixed_list = [100, 200, 'apple', 400] # 숫자와 문자열 원소를 가짐
>>> mixed_list
[100, 200, 'apple', 400]
```

파이썬의 리스트는 정수, 실수, 문자열 등 서로 다른 자료형을 가질 수 있다.

6.1 리스트 자료형의 필요성

range()나 문자열을 이용하여 리스트 만들기

- range(1, 10)이라는 함수를 통해 1부터 9까지의 숫자의 **열sequence**을 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']
```

6.1 리스트 자료형의 필요성

range()나 문자열을 이용하여 리스트 만들기



LAB 6-1: 리스트의 생성

1. 1부터 10까지의 숫자 중에서 짝수를 요소로 가지는 `even_list`라는 리스트를 생성하여라(10을 포함). `print()` 함수를 사용하여 이 리스트를 다음과 같이 출력하여라.

```
even_list = [2, 4, 6, 8, 10]
```

2. `range()` 함수를 이용하여 1번의 문제를 다시 풀어보시오.
3. 'Korea', 'China', 'India', 'Nepal'의 네 원소를 가지는 `nations`라는 리스트를 생성하여라. `print()` 함수를 사용하여 이 리스트를 다음과 같이 출력하여라.

```
nations = ['Korea', 'China', 'India', 'Nepal']
```

4. 여러분의 친한 친구 5명의 이름을 원소로 가지는 `friends`라는 리스트를 생성하여라. 그리고 다음과 같이 출력하여라.

```
friends = ['길동', '철수', '은지', '지은', '영민']
```

5. 'XYZ' 문자열을 이용하여 'X', 'Y', 'Z'라는 요소를 가지는 `string`이라는 이름의 리스트를 생성하고 다음과 같이 출력하여라.

```
string = ['X', 'Y', 'Z']
```

6.2 리스트의 인덱스

- 인덱스 **index**
 - 리스트의 항목 값을 가리키는 숫자
 - n 개의 항목을 가진 리스트의 인덱스는 0부터 $n-1$ 까지 증가
- 인덱싱 **indexing**
 - 항목의 인덱스를 이용하여 자료 값에 접근

6.2 리스트의 인덱스

n_list 리스트

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



대화창 실습: 6개의 원소를 가지는 리스트 만들기

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

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

6.2 리스트의 인덱스

n_list 리스트

- 인덱스 값에 최대 인덱스 값보다 더 큰 값을 넣으면 안 됨
- 존재하지 않는 값에 접근하면 IndexError: list index out of range 오류 발생
- 최대 인덱스는 len(n_list)-1 이다



대화창 실습: 리스트 인덱스를 통한 요소의 접근

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

주의 - 리스트 인덱스의 범위

a라는 리스트가 있을 때 이 리스트의 양의 인덱스 범위는 0에서 len(a)-1까지이다. 예를 들어, 6개의 항목을 가지는 리스트는 0에서 5까지의 양의 인덱스 범위를 가진다.

6.2 리스트의 인덱스

음수 인덱스

- 파이썬에서 리스트의 큰 특징
- 음수 인덱스를 사용할 수 있음
- C나 Java와 같은 프로그래밍 언어에서는 지원하지 않음



대화창 실습: 리스트의 음수 인덱스 사용법

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

6.2 리스트의 인덱스

음수 인덱스

- 리스트의 제일 마지막 원소의 항목은 $[-1 + \text{len}(n_list)]$
- 첫 항목은 $[-\text{len}(n_list) + \text{len}(n_list)]=[0]$
- 제일 마지막 원소로부터 -1, -2, -3과 같이 -1씩 감소하면서 인덱싱

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

[그림 6-4] a의 원소와 음수 인덱스

6.2 리스트의 인덱스

음수 인덱스



LAB 6-2: 리스트의 생성과 인덱싱

1. 2부터 10까지의 수 중에서 소수를 원소를 가지는 `prime_list`라는 리스트를 생성하여라. 그리고 이 리스트의 가장 첫 원소를 리스트 인덱싱을 이용하여 다음과 같이 출력하여라.

`prime_list`의 첫 원소 : 2

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

`prime_list`의 마지막 원소 : 7

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

`prime_list`의 마지막 원소 : 7

6.2 리스트의 인덱스

음수 인덱스

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

`nations`의 첫 원소 : Korea

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

`nations`의 마지막 원소 : Malaysia

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

`nations`의 마지막 원소 : Malaysia

6.3 리스트 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]
```

6.3 리스트 항목의 추가와 삭제

리스트 내의 항목을 지우는 세 가지 방법

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

6.3 리스트 항목의 추가와 삭제

del 키워드로 삭제하는 방법

- 지정된 인덱스에 위치한 항목을 삭제함(ex: del n_list[3])
- del 44와 같은 방법으로 삭제할 수 없다



코드 6-1: 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]
```

6.3 리스트 항목의 추가와 삭제

remove 메소드로 삭제하는 방법



코드 6-2: 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

6.3 리스트 항목의 추가와 삭제

remove 메소드로 삭제하는 방법



코드 6-3: 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)
```

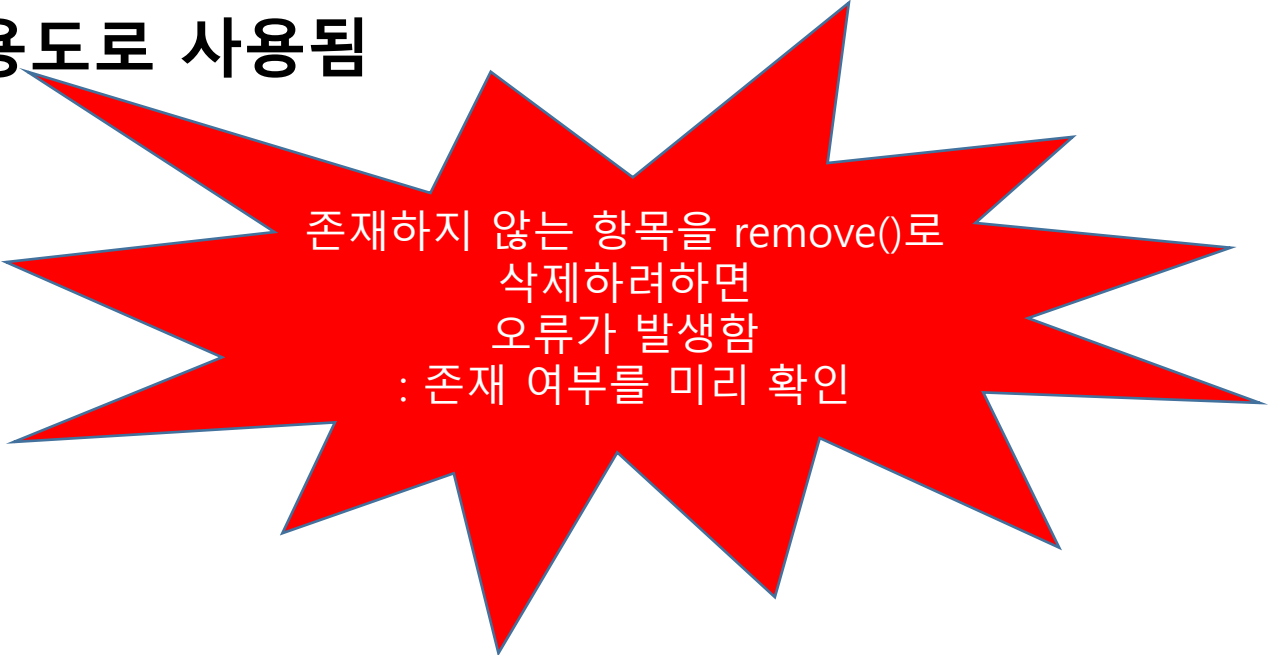
실행결과

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

존재하지 않는 항목을
remove()로 삭제하면
오류가 발생

6.4 멤버 연산자: in, not in

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



존재하지 않는 항목을 remove()로
삭제하려하면
오류가 발생함
: 존재 여부를 미리 확인

6.4 멤버 연산자: in, not in

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
```


6.4 멤버 연산자: in, not in

in 연산자의 사용



코드 6-4: 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
```

6.4 멤버 연산자: in, not in

in 연산을 사용하여 오류 발생을 미리 방지

- 리스트 내부에 지우고자 하는 값이 있는 지 확인



코드 6-5: 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]
```

6.4 멤버 연산자: in, not in



LAB 6-3: 리스트의 삽입과 삭제, in 연산자

1. 1부터 10까지의 수들 중에서 소수 원소를 가지는 `prime_list`라는 리스트를 생성하시오. 그리고 `append()` 메소드를 사용하여 11을 추가하시오. 이 때 추가 전과 추가 후의 결과를 다음과 같이 출력하시오.

소수 목록 : [2, 3, 5, 7]

추가 후 소수 목록 : [2, 3, 5, 7, 11]

2. 1번 문제의 `prime_list`라는 리스트에 있는 3이라는 원소를 `remove()` 메소드를 사용하여 제거하시오. 이 때 삭제 전과 삭제 후의 결과를 다음과 같이 출력하시오.

삭제 전 소수 목록 : [2, 3, 5, 7, 11]

삭제 후 소수 목록 : [2, 5, 7, 11]

6.4 멤버 연산자: in, not in

3. 'Korea', 'China', 'Russia', 'Malaysia'라는 국가 이름을 원소로 가지는 `nations`라는 리스트에 `append()` 메소드를 사용하여 'Nepal'을 추가하시오. 이 때 추가 전과 추가 후의 결과를 다음과 같이 출력하시오.

국가 목록 : ['Korea', 'China', 'Russia', 'Malaysia']

추가 후 국가 목록 : ['Korea', 'China', 'Russia', 'Malaysia', 'Nepal']

4. 3번 문제의 `nations`라는 리스트에 'Japan'과 'Russia'가 있는지 검사하여 출력하시오. 이 때 `in` 연산자를 사용하여 리스트에 항목이 있는가 검사하고 출력 시에는 `if-else` 조건문을 사용하시오.

Japan는(은) 국가 목록에 없습니다.

Russia는(은) 국가 목록에 있습니다.

6.5 리스트에 적용되는 내장함수

- `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

6.5 리스트에 적용되는 내장함수



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) # 리스트 내의 원소의 합을 구한다.
150
```

파이썬은 좋은 내장함수를 많이 가지고 있음.
Min(), max(), sum() 내장함수는 리스트를 인자로
가질 수 있음.

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

6.5 리스트에 적용되는 내장함수

- 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'
```

6.5 리스트에 적용되는 내장함수

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



대화창 실습: 한글 문자열 리스트와 내장함수 min(), max()

```
>>> k_fruits = ['사과', '귤', '포도', '참외']  
>>> min(k_fruits), max(k_fruits) # 한글 사전 순서로 최소값, 최대값을 구함  
( '귤', '포도' )
```


6.5 리스트에 적용되는 내장함수



LAB 6-4: 리스트의 min()과 max(), sum(), len() 함수

1. 1부터 10까지의 수 중에서 소수 원소를 가지는 prime_list라는 리스트를 생성하고 이들 중 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

6.6 리스트의 메소드

- 정렬 **sorting** 메소드인 `sort()`
 - 디폴트로 오름차순 정렬을 수행
 - 리스트 변수 뒤에 마침표(.)와 `sort()`를 적음
- 오름차순 정렬 **ascending order sorting**
 - 값이 증가하는 정렬
- 내림차순 정렬 **descending order sorting**
 - 값이 감소하는 정렬



6.6 리스트의 메소드

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]
```

6.6 리스트의 메소드

[표 6-1] 리스트의 메소드와 하는 일

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

6.6 리스트의 메소드

index() 메소드 실습



대화창 실습: 원소의 인덱스를 반환하는 index() 메소드

```
>>> a_list = ['a', 'b', 'c', 'd', 'e']
>>> a_list.index('a') # 'a' 원소의 인덱스를 반환함
0
>>> a_list.index('b')
1
>>> a_list.index('x') # 존재하지 않는 'x' 원소의 인덱스를 조회 - 오류 발생
...
ValueError: 'x' is not in list
```

- list의 가장 첫 번째에 'a'가 위치, 0번째 인덱스이므로 0을 반환
- 'x'는 리스트에 존재하지 않음

6.6 리스트의 메소드

count() 메소드 실습

- count('a') 메소드를 호출하면 'a' 원소 개수 3 반환
- 마찬가지로 count('b')는 2 반환



대화창 실습: 리스트의 count() 메소드

```
>>> b_list = ['a', 'b', 'c', 'a', 'b', 'a']
>>> b_list.count('a')    # 'a'가 몇 개 있는지 그 개수를 반환
3
>>> b_list.count('b')    # 'b'의 개수를 반환
2
```

6.6 리스트의 메소드

extend() 메소드 실습

```
>>> list1 = ['a', 'b', 'c']
>>> list1 = [1, 2, 3]
>>> list1.extend(list2) # list1에 list2의 개별 원소를 추가함
>>> list1
['a', 'b', 'c', 1, 2, 3]
>>> list1.extend('d')
>>> list1
['a', 'b', 'c', 1, 2, 3, 'd']
```

- extend()는 리스트 뒤에 리스트나 항목을 추가하는 메소드
- 인자로 'd'와 같은 개별적인 원소 또한 삽입 가능

6.6 리스트의 메소드

append() 함수와 extend() 함수의 차이점

주의 - append() 메소드와 extend() 메소드의 차이점

만일 아래의 list1에 append()를 시도하게 되면 우리가 흔히 생각하는 [11, 22, 33, 44, 55, 66]의 결과가 나오지 않는다. list1.append([55, 66]) 메소드를 호출하면 [55, 66]이라는 리스트 항목을 [11, 22, 33, 44] 다음에 추가하여 list1은 [11, 22, 33, 44, [55, 66]]이라는 항목을 가지게 된다. 따라서 list1의 원소의 수는 5개가 된다.

```
>>> list1 = [11, 22, 33, 44]
>>> list1.append([55, 66]) # list1에 [55, 66] 리스트를 추가함
>>> list1
[11, 22, 33, 44, [55, 66]]
```

len(list1)로 항목의 수를 조사하면 **5**가 출력

만일 리스트 내에 새로운 리스트의 항목을 추가하고자 할 경우에는 extend() 메소드를 사용할 수 있다. 다음과 같은 프로그램을 작성하여 수행해 보자. extend() 메소드로 [55, 66]을 추가할 경우, list2의 원소 수는 6개가 된다.

```
>>> list2 = [11, 22, 33, 44]
>>> list2.extend([55, 66]) # list2에 개별 원소 55, 66을 추가함
>>> list2
[11, 22, 33, 44, 55, 66]
```

len(list2)로 항목의 수를 조사하면 **6**이 출력

len() 함수를 사용하여 각각의 크기를 살펴보면 그 차이점을 쉽게 알 수 있다.

```
>>> len(list1), len(list2) # list1과 list2의 원소의 수
(5, 6)
```


6.6 리스트의 메소드

insert() 메소드 실습



대화창 실습: 리스트의 insert() 메소드

```
>>> list1 = ['a', 'c', 'd']  
>>> list1.insert(1, 'b')    # list1의 [1] 인덱스 위치에 'b'를 삽입  
>>> list1  
['a', 'b', 'c', 'd']
```

- 이미 생성된 리스트의 특정 위치에 새로운 원소를 삽입

6.6 리스트의 메소드

• pop() 메소드 실습



대화창 실습: 리스트의 pop() 메소드

```
>>> list1 = ['a', 'b', 'c', 'd']
>>> list1.pop() # list1의 마지막 원소 'd'를 삭제하고 반환
'd'
>>> list1
['a', 'b', 'c']
>>> list1 = ['a', 'b', 'c', 'd']
>>> list1.pop(1) # list1의 두 번째 원소 'b'를 삭제하고 반환
'b'
>>> list1
['a', 'c', 'd']
```

- pop()에 인자가 없을 경우, 리스트의 제일 마지막 항목을 삭제한 후 이 항목을 반환
- pop()과 달리 remove()는 삭제만 수행할 뿐 삭제한 항목을 반환하지 않음

6.6 리스트의 메소드

reserve() 메소드를 이용한 재배열

- 리스트의 원소를 역순으로 **재배열**
- 내림차순 정렬을 하는 것은 아님



대화창 실습: 리스트의 reverse() 메소드

```
>>> list1 = [10, 20, 30, 40]
>>> list1.reverse() # 리스트의 원소를 역순으로 다시 배열함
>>> list1
[40, 30, 20, 10]
```

6.6 리스트의 메소드



LAB 6-5: 리스트 메소드의 응용

1. a 리스트의 원소 값이 [1, 2, 3]이고 b 리스트의 원소 값이 [10, 20, 30]이다. a 리스트에 append(b) 메소드와 extend(b) 메소드를 각각 호출할 때 어떤 결과가 나타날지 예측하시오. 그리고 실제로 수행한 후의 결과를 각각 적으시오.

```
>>> a = [1, 2, 3]
>>> b = [10, 20, 30]
>>> a.append(b)
>>> a
(1) _____
```

```
>>> a = [1, 2, 3]
>>> b = [10, 20, 30]
>>> a.extend(b)
>>> a
(2) _____
```

6.6 리스트의 메소드

2. 1부터 10까지의 정수 값을 원소로 가지는 `nlist`를 생성하여 다음과 같이 출력하여라.

```
nlist = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

3. `insert()` 메소드를 사용하여 `nlist`의 제일 앞에 0을 삽입하여 다음과 같이 출력하여라.

```
nlist = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

4. 위의 3번 문제를 통해 만들어진 `nlist`를 역순으로 다시 배열하여 다음과 같이 출력하여라
(`reverse()` 메소드 사용할 것).

```
마지막 원소 = 0
```

```
nlist = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

6.7 리스트와 연산

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

6.7 리스트와 연산



코드 6-6: 두 리스트를 합치는 연산자 +를 이용하여 그 결과를 저장함
`list_plus_ex.py`

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

list3 = list1 + list2  # 리스트끼리의 덧셈 연산은 가능하다.
print(list3)
```

실행결과

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

- list1과 list2를 더한 결과를 list3에 할당해줌

6.7 리스트와 연산

리스트의 곱셈 연산



코드 6-7: 리스트에 대한 정수 곱셈 연산의 결과

`list_mult_ex.py`

```
list1 = [1, 2, 3, 4]
print('list1 * 2 =', list1 * 2)
print('list1 * 3 =', list1 * 3)
```

실행결과

```
list1 * 2 = [1, 2, 3, 4, 1, 2, 3, 4]
list1 * 3 = [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```


6.7 리스트와 연산

리스트의 곱셈 연산

- 다음의 경우 오류 반환



대화창 실습: 문법 오류를 유발하는 두 리스트의 * 연산

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

6.7 리스트와 연산

리스트의 비교 연산

- == 연산자
 - 두 리스트의 항목 값이 모두 같은지를 검사



대화창 실습: == 연산자를 이용한 리스트의 비교

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = [1, 2, 3, 4]
>>> list1 == list2    # 두 리스트의 내용이 같은지 비교한다.
True
>>> list3 = [4, 1, 2, 3]
>>> list1 == list3    # 두 리스트의 내용이 같은지 비교한다.
False
```

6.7 리스트와 연산

리스트의 크기 비교

- $>$, $>=$, $<$, $<=$ 연산자 사용
- 문자열의 경우 사전적 순서 **lexicographic order**를 비교



대화창 실습: $>$, $<$ 연산자를 이용한 리스트의 비교

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = [2, 3, 3, 4]
>>> list1 > list2
False
>>> list1 < list2
True
```

6.7 리스트와 연산



LAB 6-6: 리스트 연산

1. 사용자로부터 정수 n 을 입력 받아 [1, 2, 3]이라는 원소를 가지는 리스트를 다음과 같이 지정된 정수 n 만큼 반복 출력하여라.

반복할 정수를 입력하시오 : 3

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

반복할 정수를 입력하시오 : 4

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

6.8 리스트의 내용 갱신을 위한 방법

- for 문을 통해 리스트의 항목을 하나하나씩 나열 또는 연산 가능



코드 6-8: 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]
```

6.8 리스트의 내용 갱신을 위한 방법

모든 원소에 10을 곱하여 보기



코드 6-9: 리스트의 축약 표현을 사용하여 10을 곱하는 기능

`list_element_ex2.py`

고급 기능으로 11장에서 상세히 다룹니다

```
list1 = [10, 20, 30, 40, 50]
list1 = [n * 10 for n in list1] # 리스트 축약 표현을 사용
print(list1)
```

실행결과

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

6.8 리스트의 내용 갱신을 위한 방법

모든 원소에 10을 곱하여 보기

- 이전과 동일한 결과가 나옴



코드 6-10: 람다 함수와 map을 이용하여 리스트 요소들을 조작하기

list_element_ex5.py

```
list1 = [10, 20, 30, 40, 50]
list1 = list(map(lambda x: x*10, list1))
print(list1)
```

람다함수는 11장에서 상세히 다룹니다

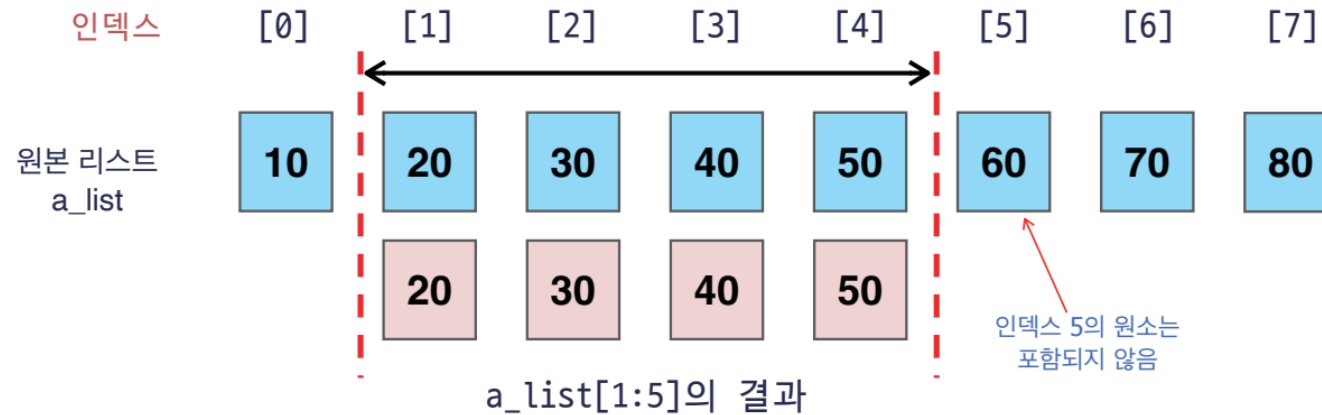
실행결과

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

6.9 리스트의 슬라이싱

슬라이싱 slicing

- 리스트 내의 항목을 특정한 구간별로 선택하여 잘라내는 기능
- 구간을 명시하기 위해 **리스트_이름[start : end]** 문법 사용
- end-1까지(end 미만)의 항목을 새 리스트에 삽입



[그림 6-5] 시작 인덱스와 끝 인덱스를 이용한 리스트 슬라이싱

6.9 리스트의 슬라이싱



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]
```

주의 : 슬라이싱의 범위

슬라이싱을 할 때 콜론(:) 뒤에 명시한 마지막 인덱스는 슬라이싱 리스트에 포함하지 않는다는 것을 항상 기억하자.

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

6.9 리스트의 슬라이싱

- 파이썬의 슬라이싱 기능은 음수 인덱스와 음수 스텝 값을 지원

[표 6-2] 슬라이싱 문법과 하는 일

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

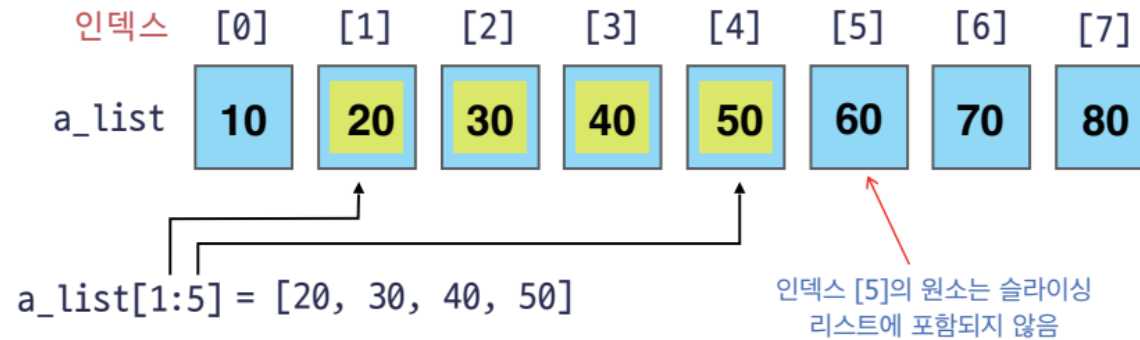
end 인덱스는 포함하지 않음

6.9 리스트의 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]
```



[그림 6-6] 시작 인덱스 1, 마지막 인덱스 5를 명시한 리스트의 슬라이싱

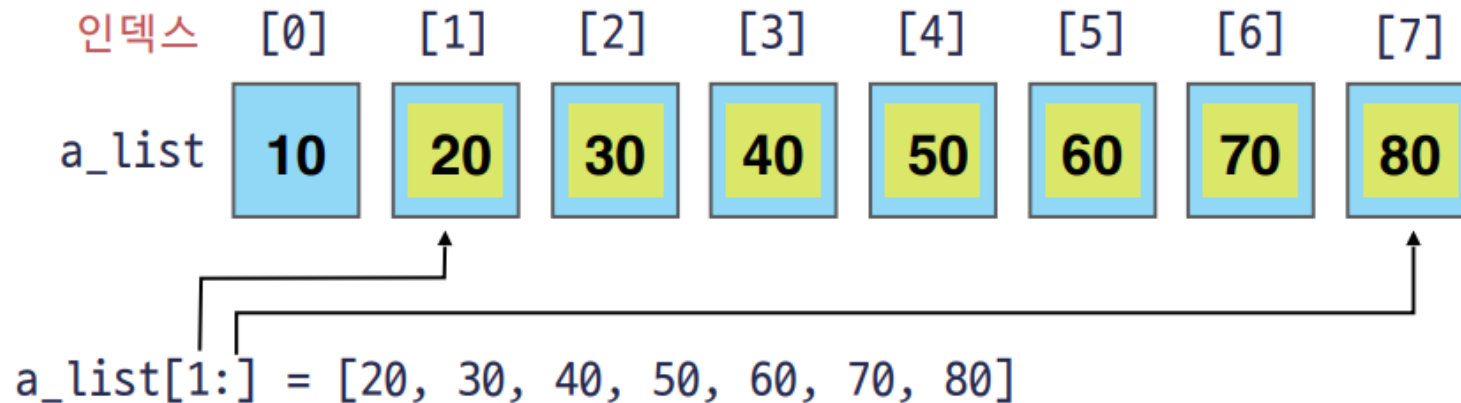
```
>>> a_list[1:5]
[20, 30, 40, 50]
```

6.9 리스트의 슬라이싱

마지막 슬라이싱 인덱스를 생략하는 경우

- `a_list[1:]`은 두 번째 항목부터 리스트의 마지막 항목까지 가져옴

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



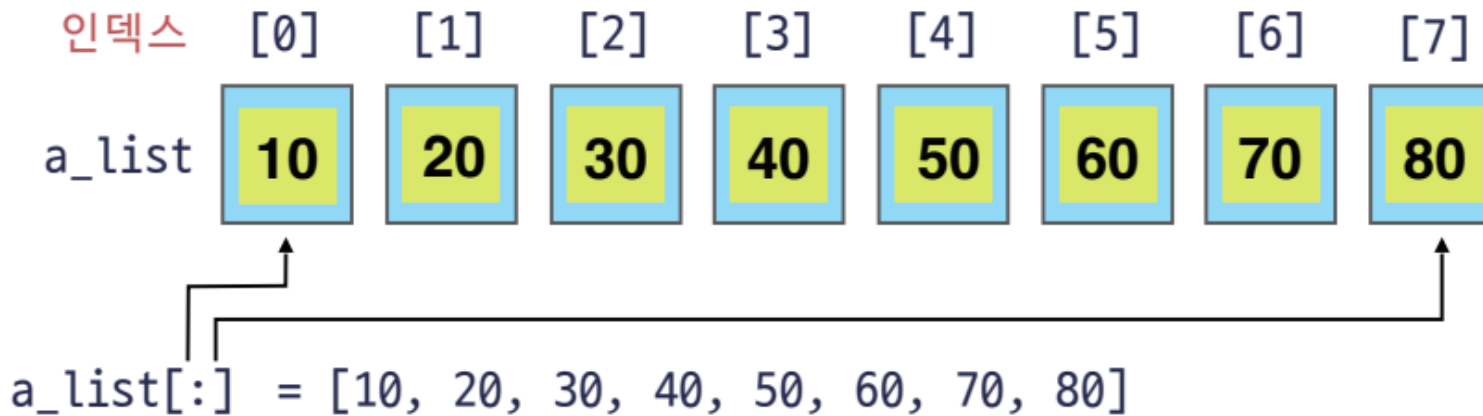
[그림 6-7] 마지막 슬라이싱 인덱스를 생략한 결과

6.9 리스트의 슬라이싱

시작 인덱스와 마지막 인덱스를 모두 생략

- 리스트의 **모든 항목**을 다 가져옴

```
>>> a_list[:]    # 전체 리스트를 슬라이싱
[10, 20, 30, 40, 50, 60, 70, 80]
```



[그림 6-8] 전체 슬라이싱 인덱스를 생략한 리스트의 슬라이싱

6.9 리스트의 슬라이싱

음수 인덱스를 사용한 슬라이싱

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

인덱스	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
a_list	10	20	30	40	50	60	70	80
음수 index	[-8]	[-7]	[-6]	[-5]	[-4]	[-3]	[-2]	[-1]

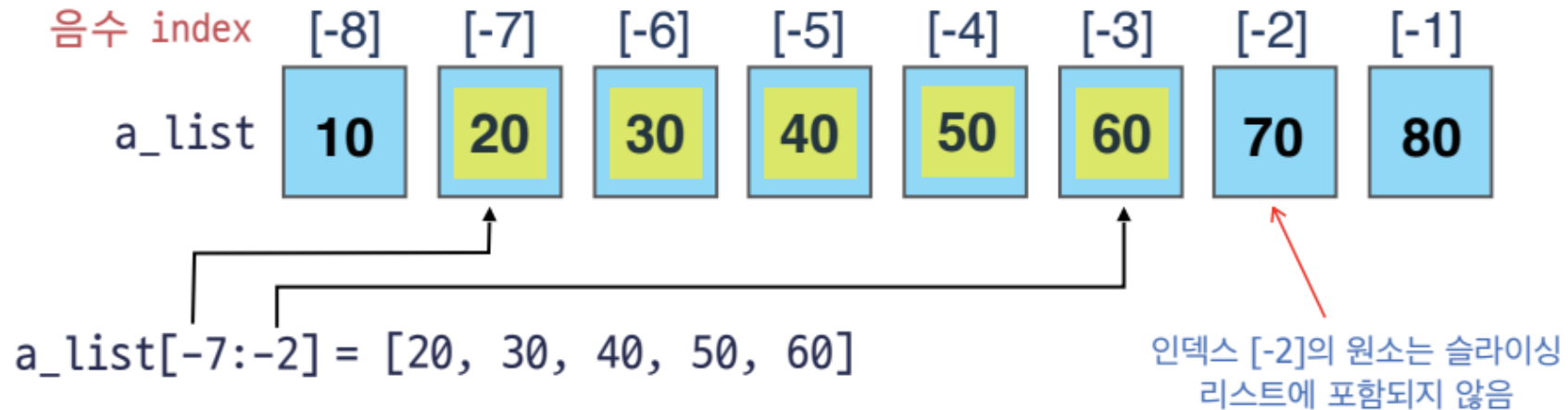
[그림 6-9] a_list의 인덱스와 음수 인덱스

6.9 리스트의 슬라이싱

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]
```



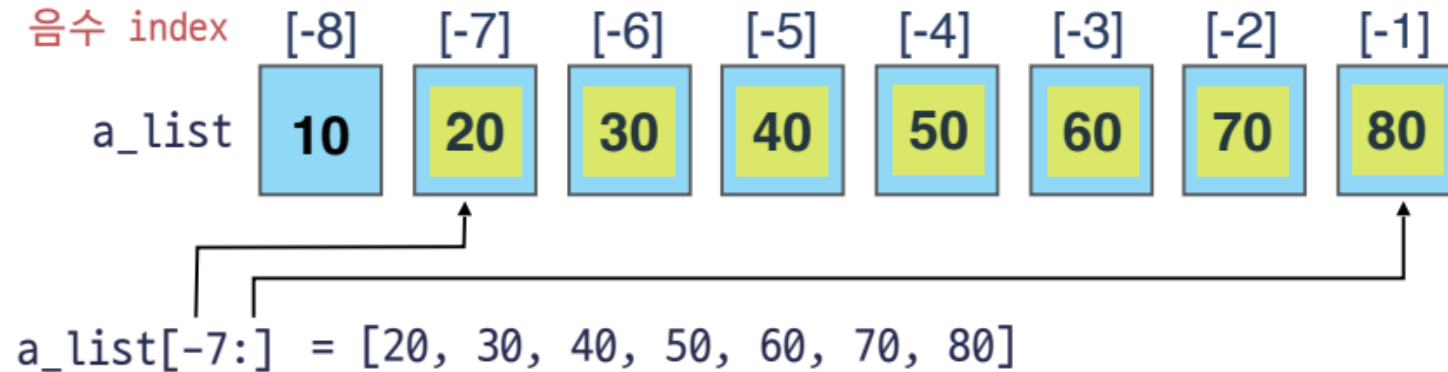
[그림 6-10] 음수 인덱스를 사용한 리스트 슬라이싱

6.9 리스트의 슬라이싱

음수 인덱스를 사용한 슬라이싱

- 마지막 인덱스를 생략할 경우 리스트의 마지막 항목까지 가져옴

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



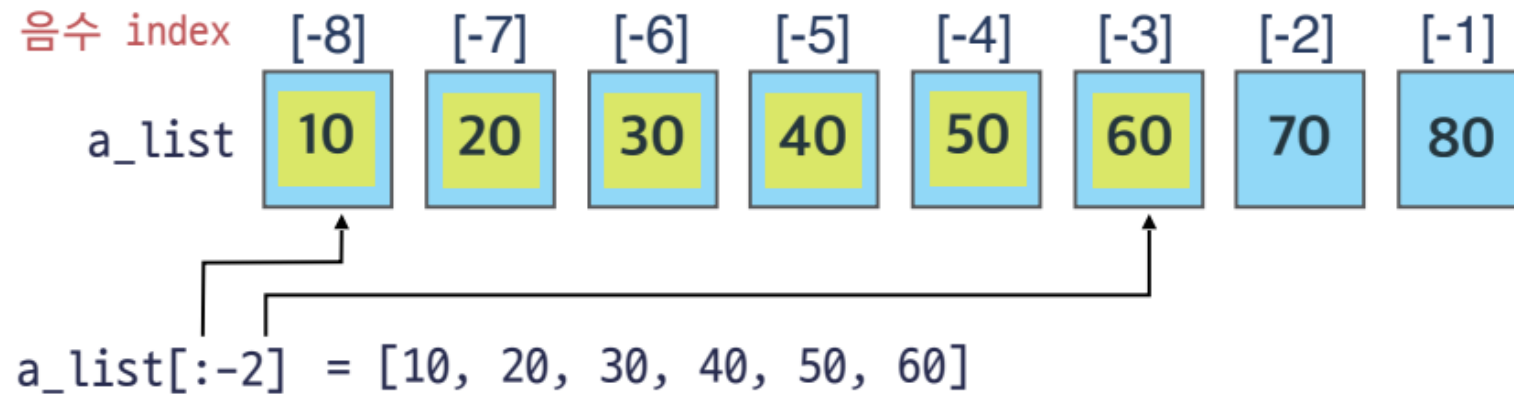
[그림 6-11] 음수 인덱스 사용 시의 리스트 슬라이싱(마지막 인덱스 생략)

6.9 리스트의 슬라이싱

음수 인덱스를 사용한 슬라이싱

- 첫 번째 인덱스를 생략하여 슬라이싱할 경우 처음부터 가져옴
 - $(-2-1)=-3$ 인덱스까지의 항목 값을 가져옴

```
>>> a_list[:-2]
[10, 20, 30, 40, 50, 60]
```

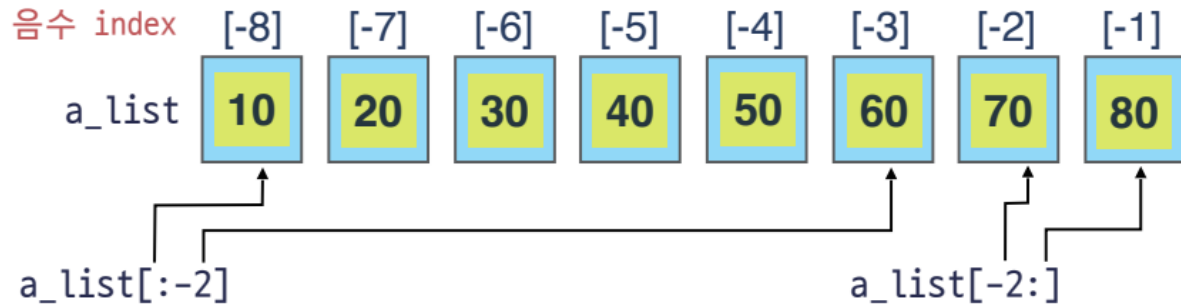


[그림 6-12] 음수 인덱스 사용 시의 리스트 슬라이싱(시작 인덱스 생략)

6.9 리스트의 슬라이싱

음수 인덱스를 사용한 슬라이싱

- `a_list[-2:]`를 통해 슬라이싱
 - 맨 뒤에서부터 2개의 인덱스를 슬라이싱한다.



```
a_list[:-2] = [10, 20, 30, 40, 50, 60]
```

```
a_list[-2:] = [70, 80]
```

```
a_list[:-2] + a_list[-2:] = [10, 20, 30, 40, 50, 60, 70, 80]
```

```
a_list[-2:] + a_list[:-2] = [70, 80, 10, 20, 30, 40, 50, 60]
```

```
>>> a_list[:-2] + a_list[-2:]
[10, 20, 30, 40, 50, 60, 70, 80]

>>> a_list[-2:] + a_list[:-2]
[70, 80, 10, 20, 30, 40, 50, 60]
```

[그림 6-14] 슬라이싱과 덧셈 연산

6.9 리스트의 슬라이싱

슬라이싱의 덧셈 결과

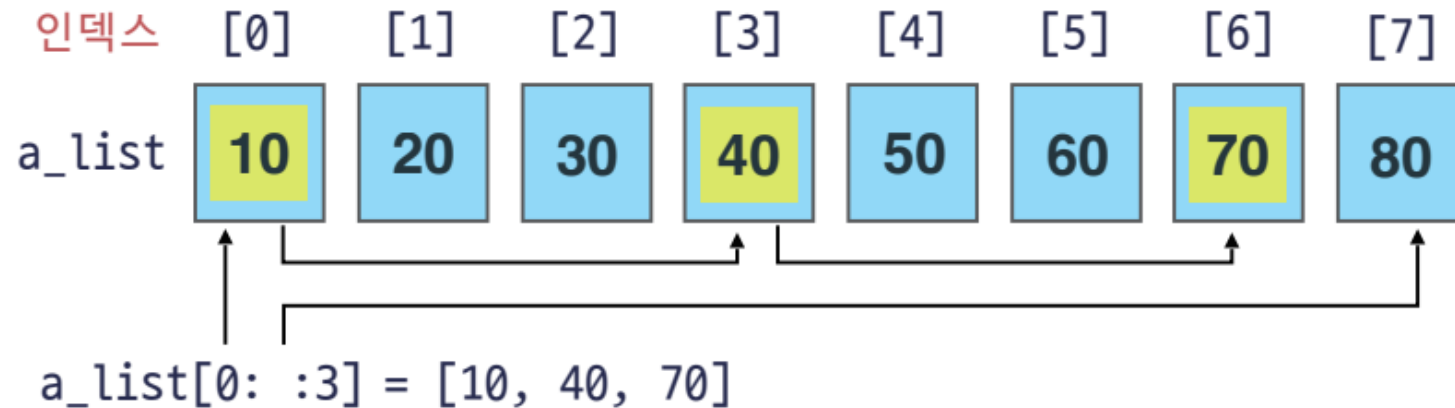
- `a_list[:-2] + a_list[-2:]`은 `a_list[:]`와 같다.
- `a_list[-2:] + a_list[:-2]`의 결과는 `a_list[:]`와 같지 않다.

6.9 리스트의 슬라이싱

슬라이싱에서 사용하는 스텝

- 특정 구간의 원소들을 일정한 간격(스텝)만큼 건너뛰며 가져오는 역할

```
>>> a_list[0::3]
[10, 40, 70]
```



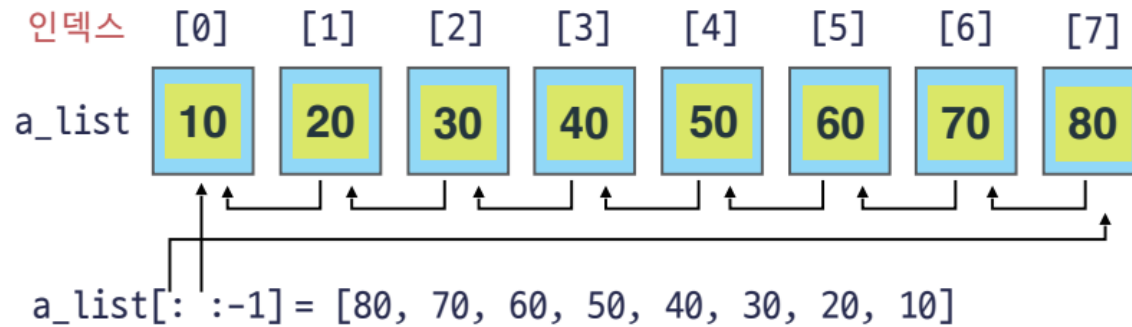
[그림 6-15] 스텝을 이용한 슬라이싱(스텝 값이 3인 경우)

6.9 리스트의 슬라이싱

슬라이싱에서 사용하는 스텝

- 음수의 스텝 값을 줄 경우 전체 구간의 뒤에서부터 앞으로 나아가며 슬라이싱
- `a_list[::-1]`의 예시

```
>>> a_list[::-1]    # 뒤에서부터 앞으로 원소를 읽는다.
[80, 70, 60, 50, 40, 30, 20, 10]
```



[그림 6-16] 음수 스텝 값을 사용한 리스트 슬라이싱

6.9 리스트의 슬라이싱



LAB 6-7: 리스트의 슬라이싱

1. range(15) 함수를 사용하여 다음과 같은 리스트를 생성하여라.

```
n_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

2. 문제 1번의 n_list로부터 슬라이싱을 수행하여 다음과 같은 리스트를 생성하여라.

```
s_list1 = [0, 1, 2, 3, 4]
```

```
s_list2 = [5, 6, 7, 8, 9, 10]
```

```
s_list3 = [11, 12, 13, 14]
```

```
s_list4 = [2, 4, 6, 8, 10]
```

```
s_list5 = [10, 9, 8, 7, 6]
```

```
s_list6 = [10, 8, 6, 4, 2]
```

Leistung ist nicht alles / Keinen Studierenden zurücklassen

