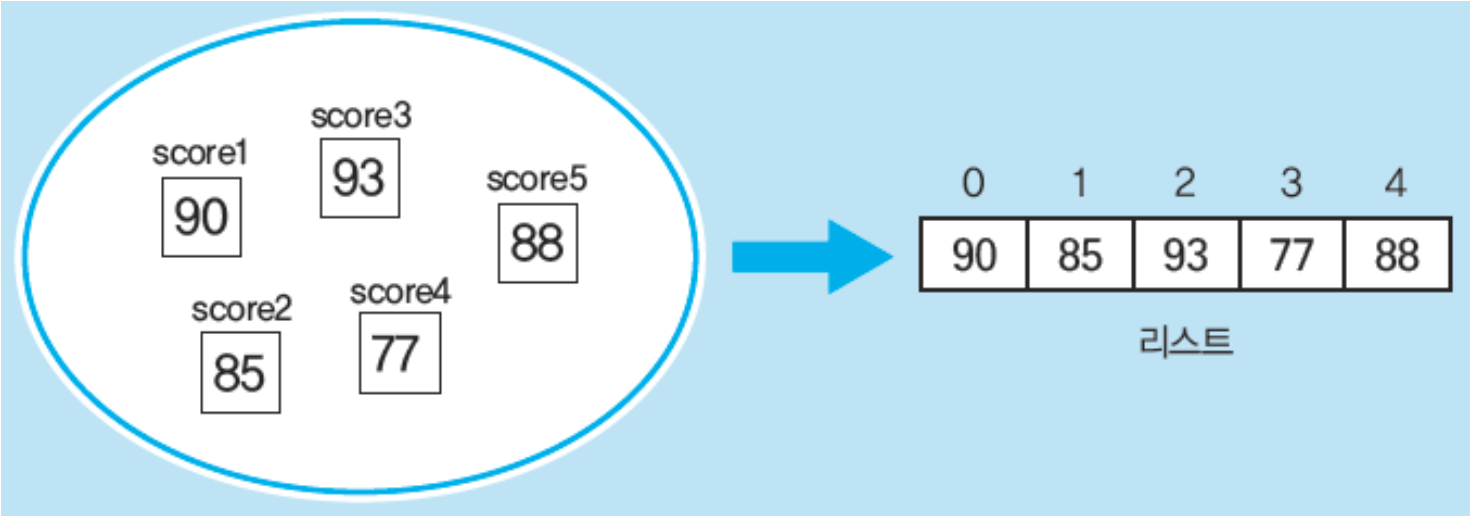
9. 리스트 자료형

1. 리스트 만들기
2. 리스트 인덱싱(indexing), 슬라이싱(slicing)
3. 리스트는 mutable 객체입니다
4. 리스트에 +, *, in, not in, del 연산자 사용하기
5. 리스트에 함수 적용하기 - len(), max(), min(), sum(), sorted(), reversed()
6. 리스트 메소드
7. 리스트 안에 리스트 구조
8. 리스트를 이용한 언패킹 (Unpacking)
9. 리스트 안에 for 반복문 사용하기 (List Comprehension)
10. 정리

1. 리스트 만들기

◆ 리스트

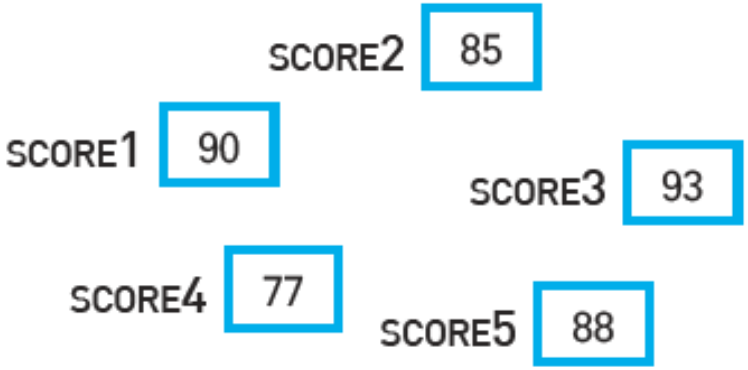
- 데이터들을 모아서 관리함
- 인덱스를 갖는 시퀀스 자료형
- 리스트에는 어떤 자료형의 데이터도 저장할 수 있음



1. 리스트 만들기

- 다섯 명의 성적 평균 구하는 프로그램을 작성하려면…
  - 다섯 개의 성적을 객체로 만들어야 함.

```
score1 = 90
score2 = 85
score3 = 93
score4 = 77
score5 = 88
```



  - 총점은 다음과 같이 구해야 함.

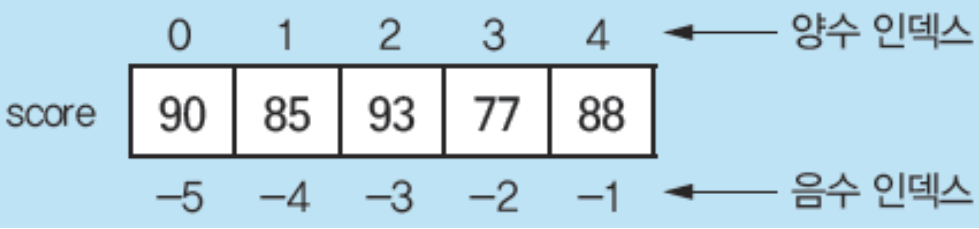
```
total = score1 + score2 + score3 + score4 + score5
```

  - 만약에 다루어야 하는 데이터 개수가 많아지면 이런 식으로 처리하기 어려움.

1. 리스트 만들기

◆ 리스트를 이용하면 많은 데이터를 처리하기 수월해짐.

score = [90, 85, 93, 77, 88]



◆ 빈 리스트 만들기

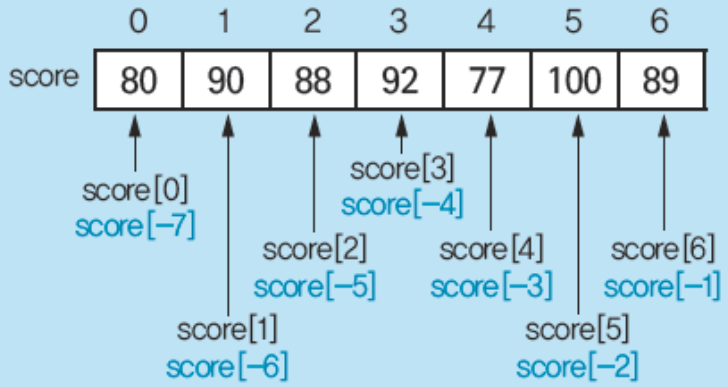
빈 대괄호 [ ] 이용하기	list() 함수 이용하기

>>> A = [ ]
>>> A
[ ]
	>>> B = list()
	>>> B
	[ ]

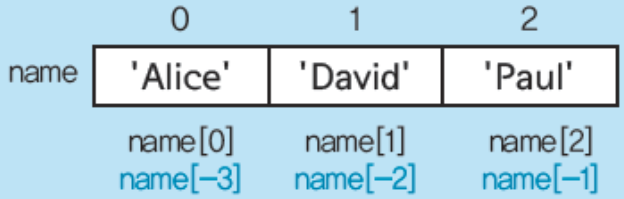
1. 리스트 만들기

- 리스트에는 파이썬이 제공하는 어떤 자료형도 저장 가능함.

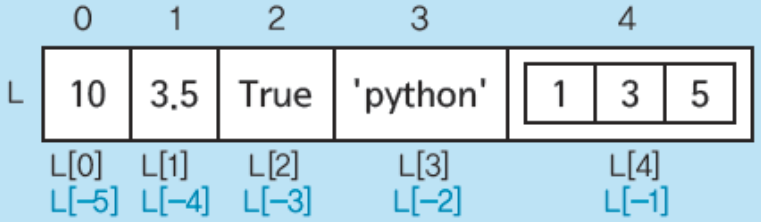
score = [80, 90, 88, 92, 77, 100, 89]



name = ['Alice', 'David', 'Paul']



L = [10, 3.5, True, 'python', [1,3,5]]



1. 리스트 만들기

◆ 다른 자료형의 데이터를 리스트로 변환하기

리스트 객체 = list ( )

iterable 자료형을 넣어야 합니다.

- 컨테이너 자료형 – 문자열, 튜플, 집합, 사전
- 내장 함수 – range(), reversed(), enumerate(), filter(), map(), zip()

리스트 ← 문자열	〉〉〉 name = 'Snow White' 〉〉〉 L1 = list(name) # 문자 하나씩 떼어서 저장함 〉〉〉 print(L1) ['S', 'n', 'o', 'w', ' ', 'W', 'h', 'i', 't', 'e']
리스트 ← 튜플	〉〉〉 odd_data = (1, 3, 5, 7, 9) 〉〉〉 L2 = list(odd_data) 〉〉〉 print(L2) [1, 3, 5, 7, 9]

1. 리스트 만들기

◆ 다른 자료형의 데이터를 리스트로 변환하기

리스트 ← 집합	>>> score = {90, 88, 75, 93, 85} # 집합은 순서 개념이 없어요.
	>>> L3 = list(score)
	>>> print(L3)
	[75, 85, 88, 90, 93]

리스트 ← 사전	>>> area_code = {'서울':'02', '경기':'031', '인천':'032'}
	>>> L4 = list(area_code) # '키'만 리스트에 저장합니다.
	>>> print(L4)
	['서울', '경기', '인천']

리스트 ← range()	>>> L5 = list(range(1, 11, 2))
	>>> print(L5)
	[1, 3, 5, 7, 9]

리스트 ← reversed()	>>> L6 = list(reversed([5, 7, 9]))
	>>> print(L6)
	[9, 7, 5]

1. 리스트 만들기

◆ 두 리스트가 같은지 비교하기

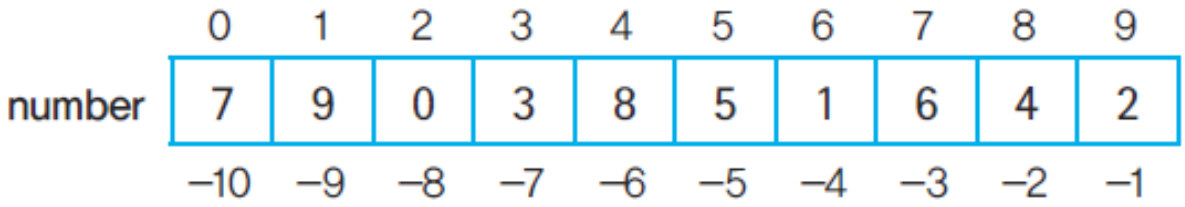
- 같은 위치에 같은 데이터가 있어야 두 리스트는 같음

```
〉〉〉 A = [1, 3, 5]          # A, B는 둘 다 1, 3, 5를 하나씩 갖지만 각 위치가 다릅니다.
〉〉〉 B = [3, 1, 5]
〉〉〉 C = [1, 1, 3, 5]       # C는 1이 두 개입니다.
〉〉〉 D = [1, 3, 5]
〉〉〉 A == B, A == C, A == D    # 같은 위치에 같은 데이터가 있고 개수 같아야 합니다.
(False, False, True)
```

2. 리스트 인덱싱(indexing), 슬라이싱(slicing)

◆ 리스트 인덱싱(indexing)은 문자열과 같음

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



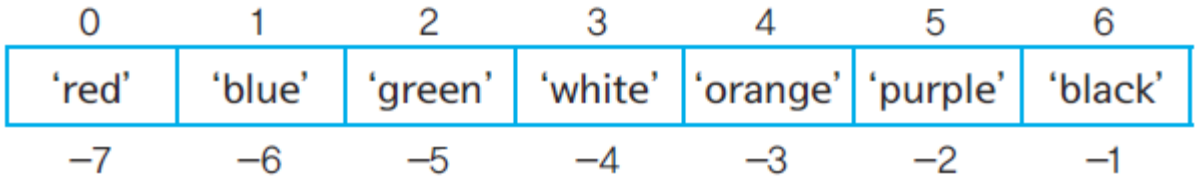

```
〉〉〉 number[2:6]
[0, 3, 8, 5]
〉〉〉 number[9:4]
[ ]
〉〉〉 number[-7:-5]
[3, 8]
〉〉〉 number[:5]      # 처음부터 인덱스 5 전까지
[7, 9, 0, 3, 8]
〉〉〉 number[5:]      # 인덱스 5부터 끝까지
[5, 1, 6, 4, 2]
```

```
〉〉〉 number[3:8:2]
[3, 5, 6]
〉〉〉 number[-8:-2:3]
[0, 5]
〉〉〉 number[:8:3]
[7, 3, 1]
〉〉〉 number[:8:-3]
[2]
〉〉〉 number[::-2]
[2, 6, 5, 3, 9]
```

2. 리스트 인덱싱(indexing), 슬라이싱(slicing)

◆ 문자열들로 구성된 리스트

color = [‘red’, ‘blue’, ‘green’, ‘white’, ‘orange’, ‘purple’, ‘black’]




```
〉〉〉 color[2:5]
['green', 'white', 'orange']
〉〉〉 color[:3]
['red', 'blue', 'green']
〉〉〉 color[4:]
['orange', 'purple', 'black']
〉〉〉 color[-2:-5]
[ ]
〉〉〉 color[-5:-2]
['green', 'white', 'orange']
〉〉〉 color[:-1]
['red', 'blue', 'green', 'white',
'orange', 'purple']
```

```
〉〉〉 color[1:6:1]
['blue', 'green', 'white', 'orange', 'purple']
〉〉〉 color[-2:-6:-1]
['purple', 'orange', 'white', 'green']
〉〉〉 color[:5:2]
['red', 'green', 'orange']
〉〉〉 color[:5:-2]
['black']
〉〉〉 color[::3]
['red', 'white', 'black']
〉〉〉 color[::-1]
['black', 'purple', 'orange', 'white', 'green',
'blue', 'red']
```

3. 리스트는 mutable 객체입니다

- 인덱싱을 이용하여 리스트 수정하기

```
〉〉〉 score = [80, 90, 95, 87, 75]
〉〉〉 id(score)
32129672
〉〉〉 score[3] = 89        # score[3]에 있는 87을 89로 수정합니다.
〉〉〉 print(score)         # 리스트 score의 내용이 수정되었어요.
[80, 90, 95, 89, 75]
〉〉〉 id(score)             # id가 그대로임을 알 수 있죠.
32129672
```

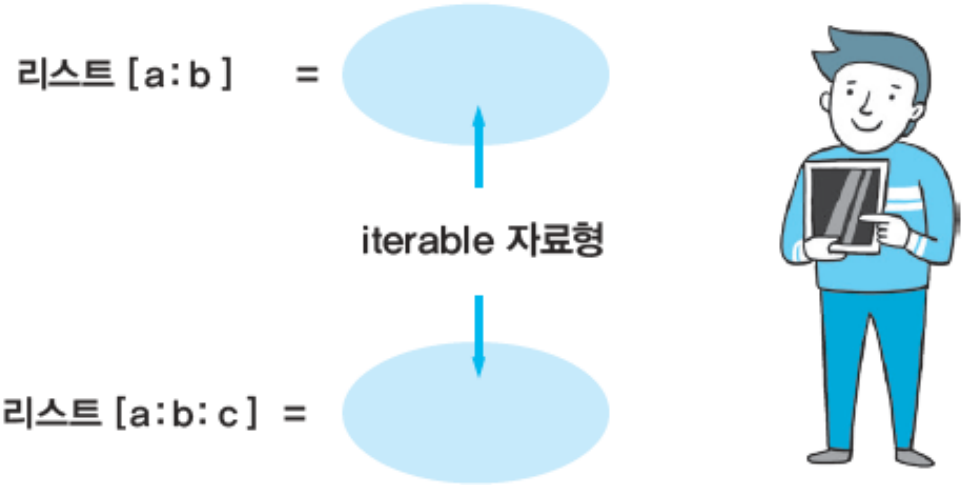
```
〉〉〉 color = ['red', 'blue', 'green', 'white', 'orange', 'purple', 'black']
〉〉〉 color[1] = 'brown'                # color[1]에 다른 문자열로 대체
〉〉〉 color
['red', 'brown', 'green', 'white', 'orange', 'purple', 'black']
```

```
〉〉〉 color = ['red', 'blue', 'green', 'white', 'orange', 'purple', 'black']
〉〉〉 color[1] = ['gray', 'yellow']    # color[1]에 리스트로 대체
〉〉〉 color
['red', ['gray', 'yellow'], 'green', 'white', 'orange', 'purple', 'black']
```

```
〉〉〉 color = ['red', 'blue', 'green', 'white', 'orange', 'purple', 'black']
〉〉〉 color[4] = 100                    # color[4]에 정수로 대체
〉〉〉 color
['red', 'blue', 'green', 'white', 100, 'purple', 'black']
```

3. 리스트는 mutable 객체입니다

- 슬라이싱을 이용하여 리스트 수정하기 - iterable 자료형을 대입해야 함




```
>>> color = ['red', 'blue', 'green', 'white', 'orange', 'purple', 'black']
>>> color[2:5] = ['brown', 'gray']
>>> color
['red', 'blue', 'brown', 'gray', 'purple', 'black']
```

```
>>> color = ['red', 'blue', 'green', 'white', 'orange', 'purple', 'black']
>>> color[2:5] = 'brown'
>>> color                          # 문자열에 있는 문자가 하나씩 분리되어 저장됨
['red', 'blue', 'b', 'r', 'o', 'w', 'n', 'purple', 'black']
```

3. 리스트는 mutable 객체입니다

```
>>> color = ['red', 'blue', 'green', 'white', 'orange', 'purple', 'black']
>>> color[1:5] = (100, 77, 50)    # 튜플을 슬라이싱 된 범위에 넣기
>>> color
['red', 100, 77, 50, 'purple', 'black']
```

```
>>> color = ['red', 'blue', 'green', 'white', 'orange', 'purple', 'black']
>>> color[4:6] = {1, 2, 3}          # 집합을 슬라이싱 된 범위에 넣기
>>> color
['red', 'blue', 'green', 'white', 1, 2, 3, 'black']
```

```
>>> color = ['red', 'blue', 'green', 'white', 'orange', 'purple', 'black']
>>> color[1:5] = {'NY':'New York', 'CA':'California'}      # 사전을 슬라이싱 된 범위에 넣기
>>> color          # '키'만 저장함
['red', 'NY', 'CA', 'purple', 'black']
```

```
>>> color = ['red', 'blue', 'green', 'white', 'orange', 'purple', 'black']
>>> color[3:5] = 100                    # 정수 하나로 슬라이싱 부분을 대체하려면 에러 발생합니다.
... ...
color[3:5] = 100
TypeError: can only assign an iterable        # iterable 자료형을 넣어야 한다는 에러입니다.
```

```
>>> score = [80, 90, 88, 92, 77, 75, 83, 65, 72, 86]
>>> score[2:9:3] = [100, 99, 98]   # score[2]에 100, score[5]에 99, score[8]에 98 대체하기
>>> print(score)
[80, 90, 100, 92, 77, 99, 83, 65, 98, 86]
```

4. 리스트에 +, *, in, not in, del 연산자 사용하기

- + : 두 리스트 연결하기 (+= 기호도 사용 가능함)

```
〉〉〉 L = [1, 3, 5, 7, 9]; M = [2, 4, 6, 8, 10]
〉〉〉 L + M                  # 리스트 L과 리스트 M을 연결합니다.
[1, 3, 5, 7, 9, 2, 4, 6, 8, 10]
〉〉〉 L, M                   # 리스트 L과 M은 변하지 않았습니다.
([1, 3, 5, 7, 9], [2, 4, 6, 8, 10])
〉〉〉 K = L + M              # K는 L과 M을 결합한 새 리스트를 갖게 됩니다.
〉〉〉 K
[1, 3, 5, 7, 9, 2, 4, 6, 8, 10]
```

```
〉〉〉 L = [1, 3, 5]
〉〉〉 M = [10, 20]
〉〉〉 L += M                 # L에 M의 원소가 추가됩니다. M은 그대로입니다.
〉〉〉 L, M
([1, 3, 5, 10, 20], [10, 20])
〉〉〉 M *= 3                 # M을 3번 반복합니다.
〉〉〉 M
[10, 20, 10, 20, 10, 20]
```

4. 리스트에 +, *, in, not in, del 연산자 사용하기

- in / not in : 리스트에 원소가 존재하는지를 확인하는 하는 연산자

```
>>> L = [2, 4, 6, 8]
>>> 6 in L              # 6은 리스트에 L에 있습니다.
True
>>> 7 not in L          # 7은 리스트 L에 없기 때문에 not in이 True입니다.
True
```

- del : 리스트 객체를 통째로 삭제하는 연산자

```
>>> data = [10, 20, 30, 40]
>>> del data[2]         # 인덱스를 이용하여 데이터 한 개를 삭제합니다.
>>> print(data)
[10, 20, 40]
>>> data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> del data[3:7]       # 슬라이스를 이용하여 데이터 여러 개를 삭제합니다.
>>> print(data)         # 3, 4, 5, 6이 삭제 되었습니다.
[0, 1, 2, 7, 8, 9]
>>> L = [1,2,3]
>>> del L               # 리스트 L을 통째로 삭제합니다.
>>> L                   # 삭제 후에 사용하려면 NameError가 발생합니다.
NameError: name 'L' is not defined
```

5. 리스트에 함수 적용하기

- len(), max(), min(), sum(), sorted(), reversed()

len(L)	리스트 L의 원소의 개수를 반환합니다.
max(L)	리스트 L의 원소 중에서 가장 큰 수를 반환합니다.
min(L)	리스트 L의 원소 중에서 가장 작을 수를 반환합니다.
sum(L)	리스트 L의 원소의 합을 구하여 반환합니다.
sorted(L)	리스트 L을 오름차순으로 정렬한 새 리스트를 만들어서 반환합니다. L은 바뀌지 않습니다.
reversed(L)	리스트 L을 역순으로 바꾸어 줍니다. 반환값에 list() 함수를 적용해야 역순으로 바뀐 리스트가 나옵니다.

5. 리스트에 함수 적용하기

```
>>> L = [3, 4, 6, 1, 2, 7, 5]
>>> len(L)      # 원소의 개수
7
>>> max(L)      # 가장 큰 원소
7
>>> min(L)      # 가장 작은 원소
1
>>> sum(L)      # 원소들의 합
28
```

```
>>> L = [3, 4, 6, 1, 2, 7, 5]
>>> S = sorted(L)  # 정렬한 리스트를 반환합니다.
>>> print(S)
[1, 2, 3, 4, 5, 6, 7]
>>> print(L)          # 원래 리스트는 변하지 않습니다.
[3, 4, 6, 1, 2, 7, 5]
```

```
>>> L = [3, 4, 6, 1, 2, 7, 5]
>>> R = reversed(L)
>>> print(R)                  # reversed( ) 함수의 반환값은 다음과 같이 출력됩니다.
<list_reverseiterator object at 0x021F4A50>
>>> V = list(R)
>>> print(V)
[5, 7, 2, 1, 6, 4, 3]
```

5. 리스트에 함수 적용하기

CODE 48 정수들이 저장된 리스트에서 인덱스 1 이후부터 끝까지의 모든 데이터 합을 구해서 인덱스 0에 저장하는 코드

data = [0, 3, 8, 10, 22, 55, 9, 1, 25, 30] data[0] = sum(data[1:]) print(data)	[결과] [163, 3, 8, 10, 22, 55, 9, 1, 25, 30]

CODE 49 리스트 L에 저장된 정수들이 앞에서부터 봐도 거꾸로 봐도 같은 숫자들로 구성되었는지 판단하는 코드

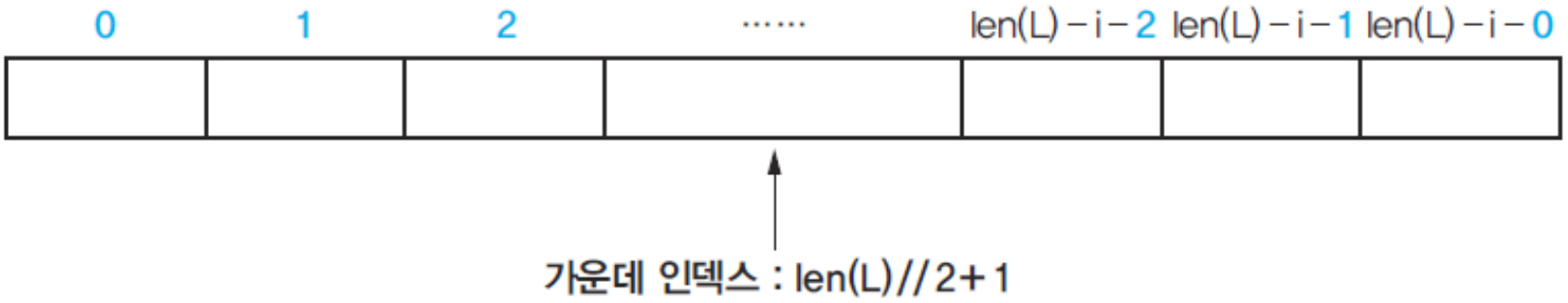
코드 1	import sys # sys.exit(0)를 위해서 필요해요.
	L = [1, 2, 3, 2, 1]
	for i in range(len(L)//2+1): # i가 인덱스 0부터 리스트 중간까지 갑니다.
	  if L[i] != L[len(L)-1-i]: # 리스트 앞에서부터, 그리고 뒤로부터 같은 위치에
	  print(False) # 있는 숫자가 같은지 판단합니다.
	  sys.exit(0)
	print(True)

5. 리스트에 함수 적용하기

CODE 49 (계속)

코드 2	L = [1, 2, 3, 2, 1] LT = list(reversed(L)) # L을 역순으로 만든 후에 리스트로 변환합니다. print(L == LT) # L과 LT가 똑같으면 True, 아니라면 False를 출력합니다.
코드 3	L = [1, 2, 3, 2, 1] print(L == L[::-1]) # L[::-1]은 리스트 L을 거꾸로 만들어 줍니다.

[코드 1]



6. 리스트 메소드

◆ 리스트에 데이터 추가하기 - append(x)

- 인수 x에는 어떤 자료형도 넣을 수 있음
- 리스트 맨 끝에 데이터 x를 추가하고 반환값은 없음 (None)

```
>>> L = [6, 8, 2, 9]
>>> y = L.append(7)
>>> print(L)
[6, 8, 2, 9, 7]
>>> print(y)
None
```

```
>>> L = ['red', 'blue']
>>> y = L.append('green')
>>> print(L)
['red', 'blue', 'green']
>>> print(y)
None
```

```
>>> M = [ ]
>>> M.append(3)
>>> M.append(9)
>>> M.append(7)
>>> print(M)
[3, 9, 7]
```

```
>>> L = [1,2,3]
>>> L.append([4,5])        # 리스트 [4,5]가 하나의 원소로 L에 맨 뒤에 추가됩니다.
>>> print(L)
[1, 2, 3, [4, 5]]
>>> len(L)
4
```

6. 리스트 메소드

CODE 50 다섯 명의 성적을 input() 함수로 입력받아서 리스트에 저장하고 그 중에서 가장 큰 성적을 출력하는 코드

```
score = [ ]
for i in range(5):          # 5회 루프를 수행합니다.
    x = int(input('성적을 입력하세요 : '))      # 성적 입력받기
    score.append(x)     # 입력받은 성적을 score 리스트에 추가
print( )                        # 한 줄띄기
print('최고 성적 :', max(score))     # 가장 좋은 성적 출력
```

[결과]

```
성적을 입력하세요 : 80
성적을 입력하세요 : 90
성적을 입력하세요 : 77
성적을 입력하세요 : 94
성적을 입력하세요 : 85
최고 성적 : 94
```

다음과 같이 한 줄로 적을 수도 있음

```
score.append(int(input('성적을 입력하세요 : ')))
```

6. 리스트 메소드

CODE 51 루프를 돌리면서 이름을 입력받아 리스트에 추가하는 코드. 'none'을 입력하면 루프를 끝내고 리스트를 출력한다.

```
names = [ ]
while True:          # 언제까지 루프가 수행될지 알 수 없으므로 무한 루프로 시작합니다.
    name = input('Enter name : ')
    if name == 'none':          #'none'이 입력되면 루프를 끝냅니다.
        break
    names.append(name)
print(names)
```

[결과]

```
Enter name : Alice
Enter name : Paul
Enter name : Tom
Enter name : none
['Alice', 'Paul', 'Tom']
```

[리스트 변화]

```
name [ ]
name ['Alice']
name ['Alice', 'Paul']
name ['Alice', 'Paul', 'Tom']
```

6. 리스트 메소드

CODE 51 (계속)

append() 메소드 대신에 += 기호를 사용할 수도 있음

```
names = [ ]                          # 빈 리스트를 만들고 시작합니다.
while True:
    name = input('Enter name : ')
    if name == 'none':               #‘none’이 입력되면 루프를 끝냅니다.
        break
    names += [name]                  # [name]을 리스트 names에 연결합니다.
print(names)
```

6. 리스트 메소드

◆ 리스트에 데이터 추가하기 - insert(i, x)

- 인덱스 i에 데이터 x를 삽입하는 메소드

```
>>> L = ['white', 'black', 'blue', 'red']
>>> L.insert(2, 'green')
>>> print(L)
['white', 'black', 'green', 'blue', 'red']
>>> M = [1,3,5,7]
>>> M.insert(2, [2,4,6])    # 리스트 insert 하기
>>> M
[1, 3, [2, 4, 6], 5, 7]
>>> M.insert(0, 100)        # 정수 한 개 insert 하기
>>> M
[100, 1, 3, [2, 4, 6], 5, 7]
```

6. 리스트 메소드

◆ 리스트와 iterable 자료형 연결하기 - extend()

iterable 자료형	예제

문자열	>>> L = [1,3,5]
	>>> L.extend('python') # 문자열은 문자 하나하나가 원소가 됩니다.
	>>> L
	[1, 3, 5, 'p', 'y', 't', 'h', 'o', 'n']

리스트	>>> L = [1,3,5]
	>>> L.extend([10,20])
	>>> L
	[1, 3, 5, 10, 20]

튜플	>>> L = [1,3,5]
	>>> L.extend((7,8,9))
	>>> L
	[1, 3, 5, 7, 8, 9]

집합	>>> L = [1,3,5]
	>>> L.extend({4, 2, 8}) # 집합은 순서 개념이 없어요.
	>>> L
	[1, 3, 5, 8, 2, 4]

6. 리스트 메소드

◆ 리스트와 iterable 자료형 연결하기 - extend()

iterable 자료형	예제

사전	>>> L = [1,3,5]
	>>> L.extend({'one':1, 'two':2, 'three':3})
	>>> L # 사전의 키만 연결합니다.
	[1, 3, 5, 'one', 'two', 'three']

range()	>>> L = [1,3,5]
	>>> L.extend(range(7, 10, 2))
	>>> L
	[1, 3, 5, 7, 9]

reversed()	>>> L = [1,3,5]
	>>> L.extend(reversed(L))
	>>> L
	[1, 3, 5, 5, 3, 1]

6. 리스트 메소드

◆ 리스트에서 데이터 삭제하기 - pop()

- pop() 메소드는 리스트에 있는 하나의 데이터를 삭제하고, 삭제한 데이터를 반환함 (임의의 데이터를 삭제함).

인수	없음	리스트.pop() – 리스트에 있는 마지막 데이터를 삭제하고 반환합니다. 빈 리스트에 pop() 메소드를 적용하면 IndexError가 발생합니다.
	1개	리스트.pop(i) – 리스트에서 인덱스 i에 있는 데이터를 삭제하고 그 데이터를 반환합니다. i에 없는 인덱스를 넣으면 IndexError가 발생합니다.
반환값	리스트에서 삭제한 데이터를 반환합니다.	

6. 리스트 메소드

◆ 리스트에서 데이터 삭제하기 - pop()

```
>>> fruits = ['apple', 'banana', 'melon', 'berry', 'kiwi']
>>> result = fruits.pop( )
>>> print(result)
kiwi
>>> print(fruits)              # 'kiwi'가 삭제되었습니다.
['apple', 'banana', 'melon', 'berry']
>>> result2 = fruits.pop(2)
>>> print(result2)
melon
```

```
>>> M = [ ]                   # 빈 리스트에 pop( ) 메소드를 적용하면 IndexError가 발생합니다.
>>> M.pop( )
... ...
IndexError: pop from empty list
```

```
>>> colors = ['red', 'blue', 'white', 'black', 'brown']
>>> colors.pop(5)         # 없는 인덱스를 인수에 넣으면 IndexError가 발생합니다.
... ...
IndexError: pop index out of range
```

6. 리스트 메소드

◆ 리스트에서 데이터 삭제하기 - remove(x)

- 인수 x에는 삭제하고자 하는 데이터를 넣어야 함.
- 반환값은 없고, 만약에 리스트에 x가 여러 개 있으면 맨 앞에 있는 원소만 삭제함.
- 없는 원소를 삭제하면, 'ValueError'가 발생함.

```
〉〉〉 color = ['red', 'blue', 'white', 'black']
〉〉〉 result = color.remove('white')
〉〉〉 print(result)              # 원소 'white'를 삭제하고 반환하는 값은 없습니다.
None
〉〉〉 print(color)
['red', 'blue', 'black']
〉〉〉 color.remove('green')   # 없는 원소를 삭제하면 ValueError가 발생합니다.
… …
ValueError: list.remove(x): x not in list
```

6. 리스트 메소드

CODE 52 리스트 data에서 짝수를 찾아서 삭제하는 코드. (문제가 있는 코드임)

```
data = [4, 7, 8, 1, 2, 5]
for i in range(len(data)):
    if data[i] % 2 == 0:
        data.remove(data[i])
print(data)
```

[결과]

```
Traceback (most recent call last):
  File "C:/Users/...../test.py", line 4, in <module>
    if data[i] % 2 == 0:
IndexError: list index out of range
```

위 코드의 문제 확인해 보기

```
data = [4, 7, 8, 1, 2, 5]
for i in range(len(data)):
    print(data)  # 추가하였음
    if data[i] % 2 == 0:
        data.remove(data[i])
print(data)
```

[결과]

```
[4, 7, 8, 1, 2, 5]
[7, 8, 1, 2, 5]
[7, 1, 2, 5]
[7, 1, 5]
Traceback (most recent call last):
  File "C:/Users/...../test.py", line 5, in <module>
    if data[i] % 2 == 0:
IndexError: list index out of range
```

6. 리스트 메소드

CODE 52 (계속) 쉬운 해결 방법으로 새로운 리스트 만들어서 홀수만을 추가할 수 있음

```
data = [4, 7, 8, 1, 2, 5]
result = [ ]                          # 홀수만을 저장할 새로운 리스트를 만듭니다.

for i in range(len(data)):
    if data[i] % 2 != 0:
        result.append(data[i])

data = result                         # 홀수만 저장한 리스트를 data에 할당합니다.
del result                            # result는 삭제합니다.
print(data)
```

6. 리스트 메소드

- 리스트에서 데이터 삭제하기 - clear()
  - 리스트에 있는 모든 데이터를 삭제하고 빈 리스트로 만들어 줌

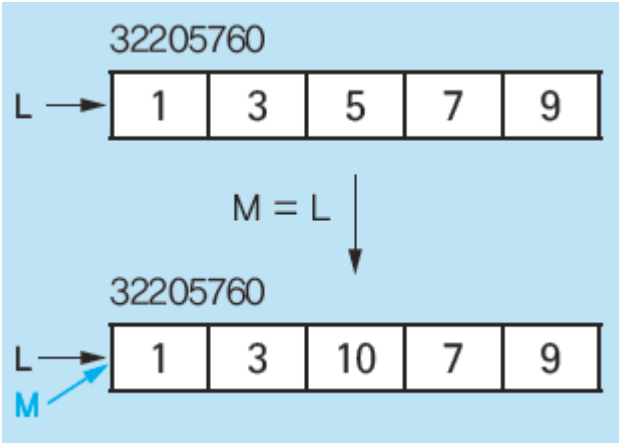
```
>>> L = [1, 3, 5, 7, 9]
>>> L.clear( )
>>> L
[ ]
```

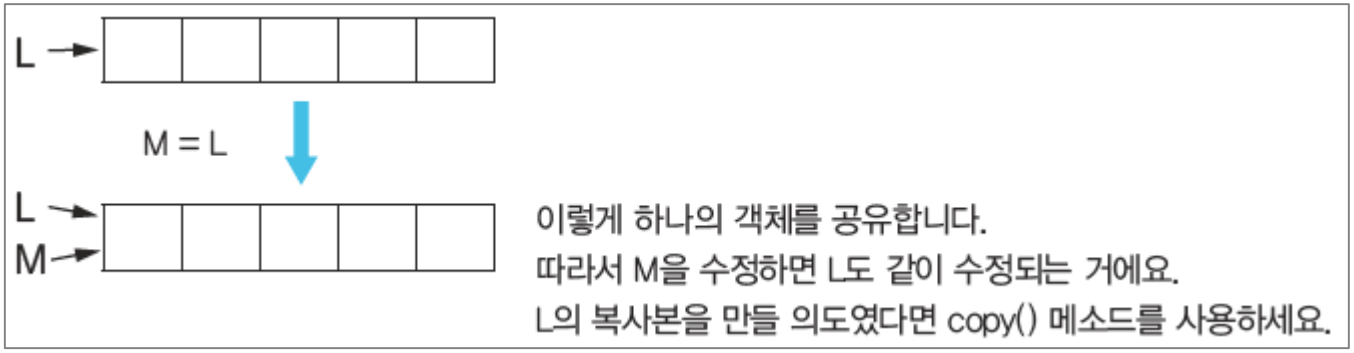
6. 리스트 메소드

◆ 리스트 복사하기 - copy()

▪ 리스트에 리스트 대입하기

```
〉〉〉 L = [1, 3, 5, 7, 9]
〉〉〉 id(L)
32205760
〉〉〉 M = L
〉〉〉 id(M)          # 리스트 L과 M은 같은 id를 갖습니다.
32205760
〉〉〉 M[2] = 10      # M[2]의 값을 10으로 수정합니다.
〉〉〉 print(L)
[1, 3, 10, 7, 9]
〉〉〉 print(M)
[1, 3, 10, 7, 9]
```

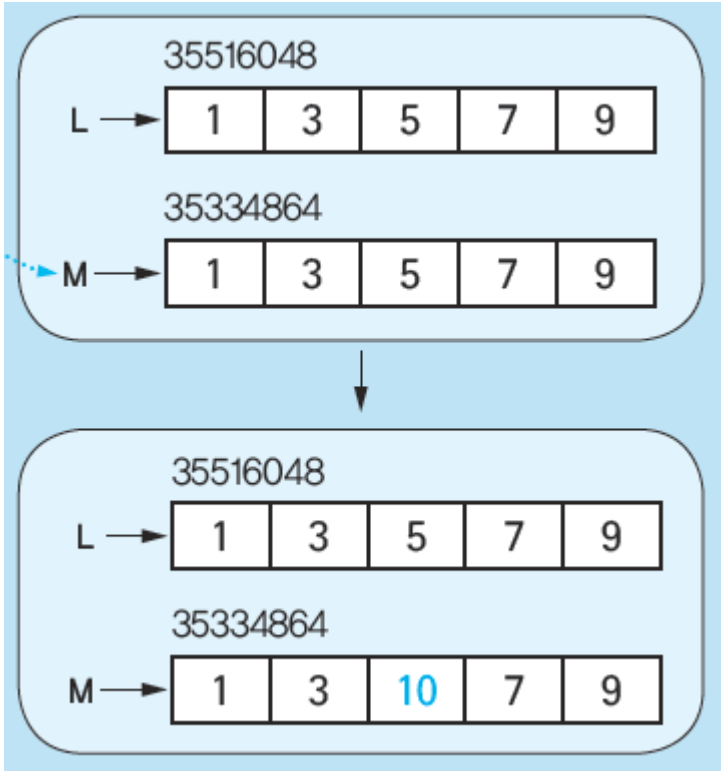




6. 리스트 메소드

◆ 리스트 복사하기 - copy()

```
〉〉〉 L = [1,3,5,7,9]
〉〉〉 M = L.copy( )
〉〉〉 id(L), id(M)     # 다른 id를 갖습니다.
(35516048, 35334864)
〉〉〉 print(L)
[1, 3, 5, 7, 9]
〉〉〉 print(M)
[1, 3, 5, 7, 9]
〉〉〉 M[2] = 10        # 리스트 M을 수정합니다.
〉〉〉 print(L)
[1, 3, 5, 7, 9]
〉〉〉 print(M)
[1, 3, 10, 7, 9]
```



6. 리스트 메소드

◆ 리스트 복사하기 - [:], [::]으로 복사하기

```
>>> A = [1,2,3]
>>> B = A[:]     # 독립된 개체 B를 만듭니다.
>>> id(A), id(B)
(35529352, 35604400)
>>> B[1] = 100
>>> A
[1, 2, 3]
>>> B
[1, 100, 3]
```

```
>>> C = [5,6,7]
>>> D = C[::]    # 독립된 개체 D를 만듭니다.
>>> id(C), id(D)
(35529312, 35604920)
>>> C[2] = 500
>>> print(C)
[5, 6, 500]
>>> print(D)
[5, 6, 7]
```

6. 리스트 메소드

- 리스트에 있는 데이터 개수 세기 - count(x)
  - 리스트에 데이터 x가 몇 개인지를 반환함.
  - 반드시 인수로 한 개의 데이터를 넣어야 하고, x가 없는 데이터인 경우에는 0을 반환함.

```
>>> L = [3, 5, 4, 1, 2, 3, 2, 2, 5]
>>> L.count(2)
3
>>> L.count(0)
0
>>> L.count(3)
2
```

6. 리스트 메소드

◆ 리스트에 있는 데이터 위치 찾기 - index()

인수	1개	리스트.index(찾고자 하는 데이터)
	2개	리스트.index(찾고자 하는 데이터, a) 리스트[a:]에서 찾고자 하는 데이터가 있으면 그 데이터의 인덱스를 반환합니다.
	3개	리스트.index(찾고자 하는 데이터, a, b) 리스트[a:b]에 찾고자 하는 데이터가 있으면 그 데이터의 인덱스를 반환합니다.
반환값	리스트에서 찾고자 하는 데이터의 인덱스를 반환합니다.	

6. 리스트 메소드

L = [89, 84, 90, 77, 95, 90, 65, 100, 90, 84]

인수 1개인 경우

```
>>> L.index(77)
3
>>> L.index(90)    # 여러 개인 경우 가장 앞의 인덱스를 반환합니다.
2
>>> L.index(99)    # 없는 데이터는 ValueError가 발생합니다.
.....
ValueError: 99 is not in list
```

90은 인덱스 2, 5, 8 세 곳에 있습니다.

99는 L에 없습니다.

인수 2개인 경우

```
>>> L.index(90, 4)    # L[4:]에서 90이 처음으로 나오는 인덱스
5
>>> L.index(90, 6)    # L[6:]에서 90이 처음으로 나오는 인덱스
8
>>> L.index(77, 6)    # L[6:]에 77이 없으므로 ValueError 발생함.
.....
ValueError: 77 is not in list
```

인수 3개인 경우

```
>>> L.index(77, 2, 7)    # L[2:7]에서 77의 위치
3
>>> L.index(90, 3, 7)    # L[3:7]에서 90의 위치
5
```

6. 리스트 메소드

- 리스트 역순으로 만들기 - reverse()

```
>>> L = [1,3,5,7,9]
>>> result = L.reverse( )      # 리스트 L이 역순으로 바뀝니다.
>>> print(L)
[9, 7, 5, 3, 1]
>>> print(result) # 반환값은 없습니다. 따라서 그냥 L.reverse( )라고 씁니다.
None
```

6. 리스트 메소드

◆ 리스트 정렬하기 - sort()

- 인수는 없거나, 1개 또는 2개까지 넣을 수 있음.
- 반환값은 없음.

[인수없이 사용하는 경우]

```
>>> L = [5, 10, 2, 7, 4, 2, 3]
>>> L.sort( )
>>> L
[2, 2, 3, 4, 5, 7, 10]
>>> M = ['python', 'java', 'c++', 'javascript']
>>> M.sort( )          # 문자열은 아스키코드 값을 비교해서 정렬합니다.
>>> M
['c++', 'java', 'javascript', 'python']
```

6. 리스트 메소드

◆ 리스트 정렬하기 - sort()

[인수가 있는 경우]

- 내림차순 정렬은 인수에 'reverse=True'를 넣어야 함.
- 정렬 기준을 바꾸려면 'key=함수명' 형태를 사용함. 함수를 적용한 결과에 따라 정렬함.

6. 리스트 메소드

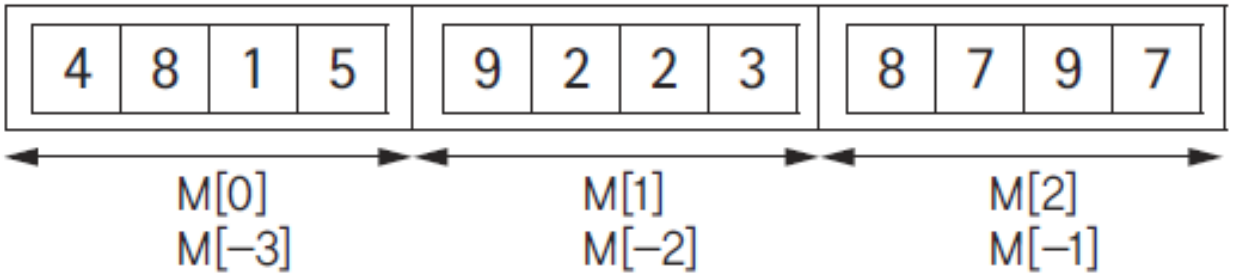
```
>>> N = [4, 10, -5, 0, -8, 1, -9]
>>> N.sort( )                    # 양수, 0, 음수가 섞여 있는 리스트를 오름차순으로 정렬합니다.
>>> N
[-9, -8, -5, 0, 1, 4, 10]
>>> M = [4, 10, -5, 0, -8, 1, -9]
>>> M.sort(key=abs)    # 리스트 M의 각 원소에 abs( ) 함수를 적용한 결과에 따라 정렬합니다.
>>> M
[0, 1, 4, -5, -8, -9, 10]
>>> city = ['Seoul', 'LA', 'New York', 'Berlin']
>>> city.sort(key=len)           # city의 각 원소에 len 함수를 적용한 결과로 정렬합니다.
>>> city
['LA', 'Seoul', 'Berlin', 'New York']
```

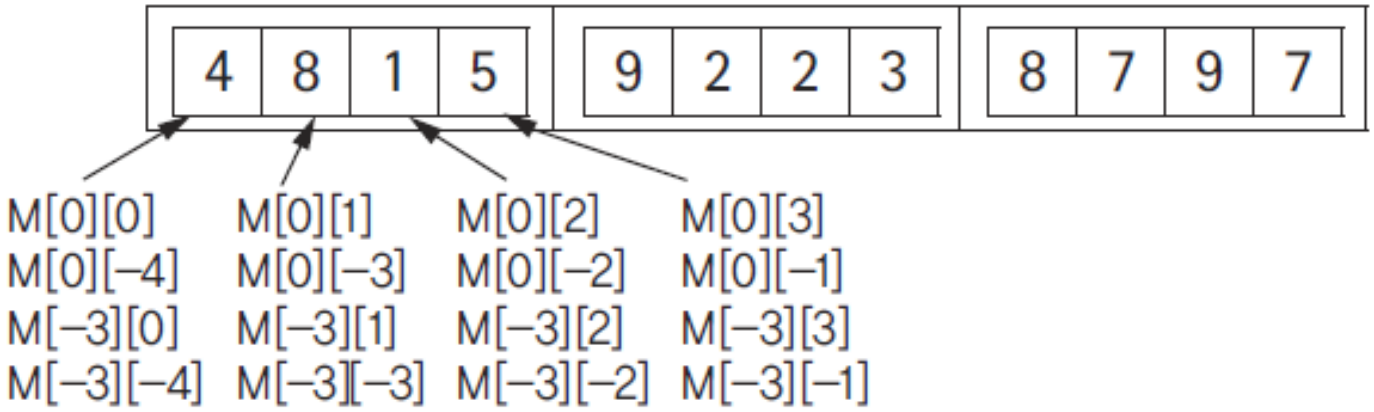
key 인수와 reverse 인수를 같이 사용할 수도 있음

```
>>> city = ['Seoul', 'LA', 'New York', 'Berlin']
>>> city.sort(key=len, reverse=True)
>>> city
['New York', 'Berlin', 'Seoul', 'LA']
```

7. 리스트 안에 리스트 구조

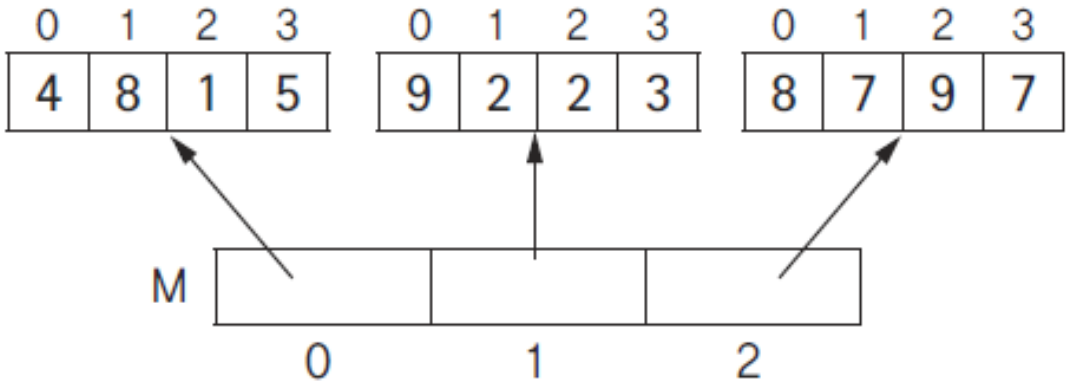
M = [[4,8,1,5], [9,2,2,3], [8,7,9,7]]





7. 리스트 안에 리스트 구조

- 실제 리스트 안에 리스트 구조는 다음과 같이 생성됨.



실제로는 M[0]에 [4,8,1,5]가 어디 있는지 참조하는 값이 저장됩니다.
M[1]에는 [9,2,2,3]의 참조값, M[2]에는 [8,7,9,7]의 참조값이 저장됩니다.

```
〉〉〉 M = [[4,8,1,5], [9,2,2,3], [8,7,9,7]]
〉〉〉 print(M[0])
[4, 8, 1, 5]
〉〉〉 print(M[0] [3])
5
〉〉〉 print(M[-3] [0], M[-3] [1], M[-3] [2], M[-3] [3])
4 8 1 5
```

7. 리스트 안에 리스트 구조

- 리스트 안에 리스트 구조는 행렬과 같음.

```
>>> M = [[4,8,1,5], [9,2,2,3], [8,7,9,7]]
>>> M.sort(key=sum)
>>> M
[[9, 2, 2, 3], [4, 8, 1, 5], [8, 7, 9, 7]]
>>> M.sort(key=sum, reverse=True)
>>> M
[[8, 7, 9, 7], [4, 8, 1, 5], [9, 2, 2, 3]]
```

4	8	1	5
9	2	2	3
8	7	9	7

2차원 행렬

4	8	1	5	→ sum(M[0]) = 18
9	2	2	3	→ sum(M[1]) = 16
8	7	9	7	→ sum(M[2]) = 31

7. 리스트 안에 리스트 구조

CODE 54 배열 M에는 세 반의 학생들 성적이 저장되어 있음. 각 반의 평균을 구하여 리스트에 저장하고 소수점 둘째 자리까지 출력하는 코드.

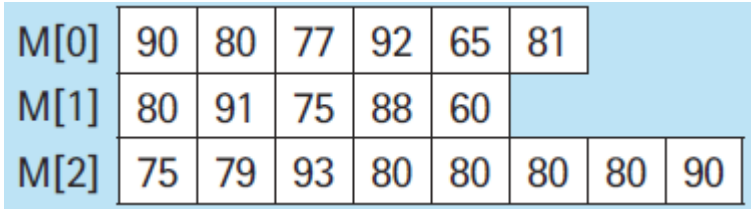
```
M = [[90, 80, 77, 92, 65, 81],
     [80, 91, 75, 88, 60],
     [75, 79, 93, 80, 80, 80, 80, 90]]

average = []            # 각 반의 평균을 구하여 average 리스트에 저장
for i in range(len(M)):
    average.append(sum(M[i])/len(M[i])) # 원소의 합을 길이로 나누어 평균 구하기

for i in range(len(average)):                    # 리스트 average 출력
    print('{:5.2f}'.format(average[i]))
```

[결과]

```
80.83
78.80
82.12
```



7. 리스트 안에 리스트 구조

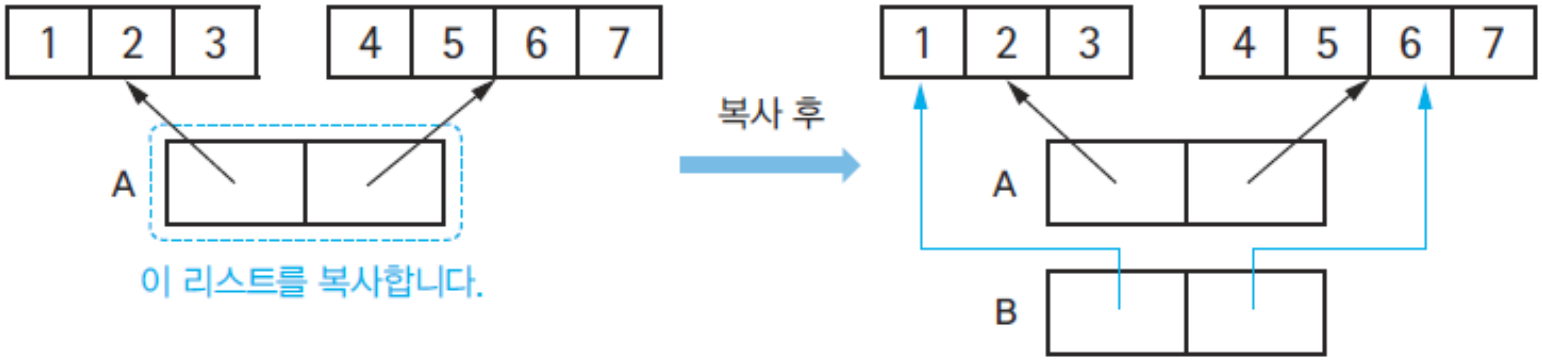
◆ 얕은 복사와 깊은 복사

- 아래 코드에서 복사가 제대로 되지 않는 이유는?

```
>>> A = [[1,2,3], [4,5,6,7]]
>>> B = A.copy( )        # 리스트 A의 복사본 B를 만듭니다.
>>> B[1] [2] = 100       # 리스트 B의 내용을 수정합니다.
>>> print(A)             # 리스트 A도 같이 수정되었습니다. 무슨 문제일까요?
[[1, 2, 3], [4, 5, 100, 7]]
>>> print(B)
[[1, 2, 3], [4, 5, 100, 7]]
```

7. 리스트 안에 리스트 구조

◆ 얕은 복사 (shallow copy)

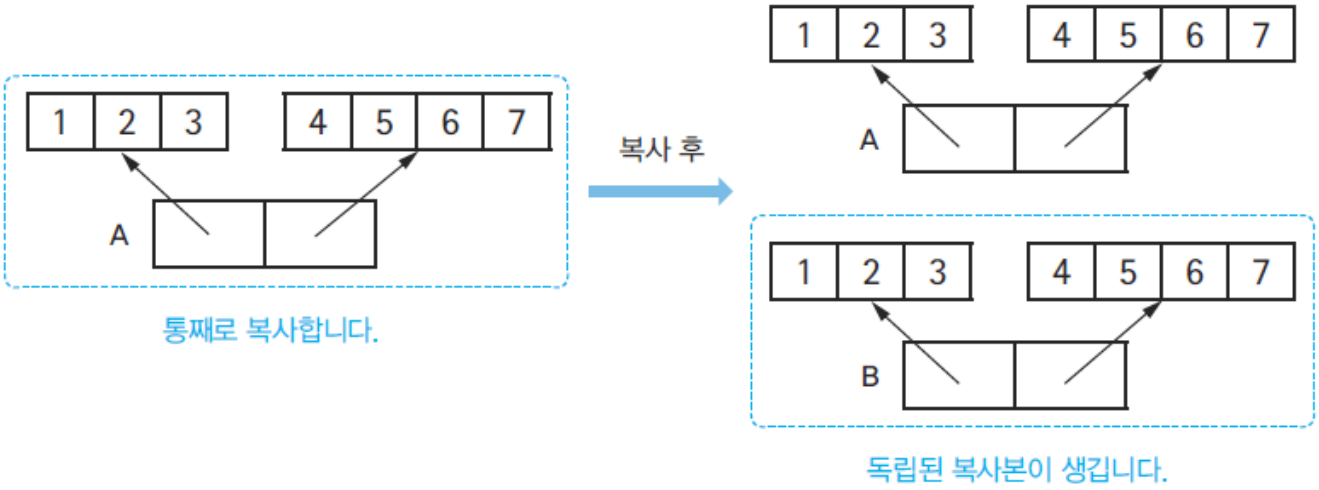


- 참조값을 갖고 있는 배열만을 복사함. 이러한 복사를 '얕은 복사'라고 함.
- 이 문제를 해결하기 위해서는 '깊은 복사'를 해야 함.
  - 깊은 복사는 copy 모듈에 있는 deepcopy() 함수를 사용해야 함.

7. 리스트 안에 리스트 구조

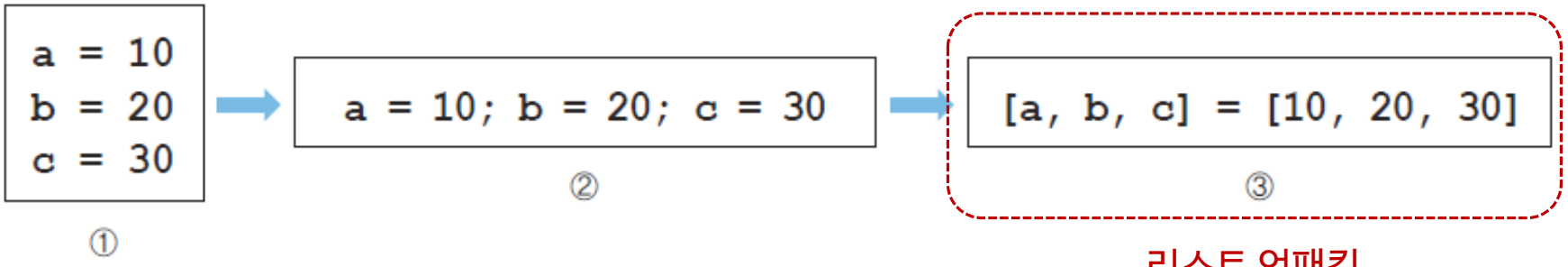
◆ 깊은 복사 (deep copy)

```
>>> import copy
>>> A = [[1,2,3], [4,5,6,7]]
>>> B = copy.deepcopy(A)      # 깊은 복사를 수행함.
>>> B[1][2] = 100
>>> print(A)
[[1, 2, 3], [4, 5, 6, 7]]     # A는 그대로임.
>>> print(B)
[[1, 2, 3], [4, 5, 100, 7]]   # 복사본 B는 수정됨.
```



8. 리스트를 이용한 언패킹 (Unpacking)

◆ 리스트 언패킹



- 기본적으로 '=' 양변에 개수가 같아야 함.
- 개수가 다르면 ValueError가 발생함.

```
〉〉〉 [a, b] = [10, 20, 30]      # '=' 양변에 개수가 맞지 않으면 에러가 발생합니다.
Traceback (most recent call last):
  File "〈pyshell#229〉", line 1, in 〈module〉
    [a, b] = [10, 20, 30]
ValueError: too many values to unpack (expected 2)
```

9. 리스트 안에 for 반복문 사용하기 (List Comprehension)

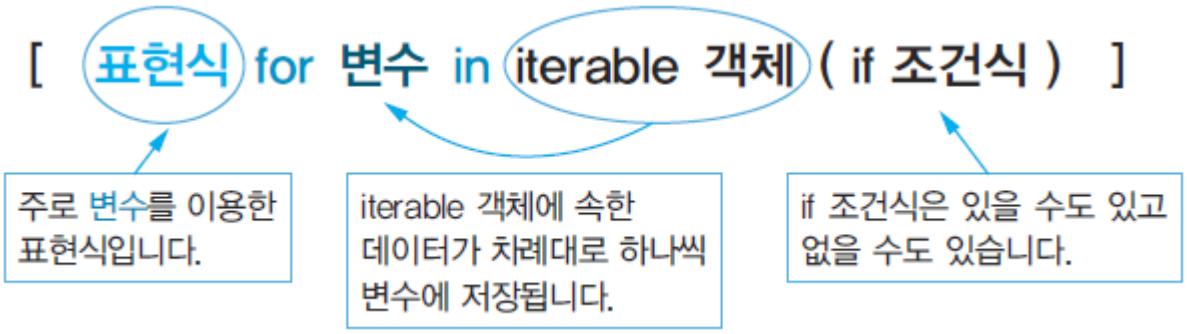
```
A = [ ]                          # 빈 리스트를 만듭니다.
for x in range(1,11):            # x는 1부터 10까지 변합니다.
    A.append(x*x)                # x²을 A에 추가합니다.
print(A)
```

[결과]

[1, 4, 9, 6, 25, 36, 49, 64, 81, 100]

```
〉〉〉 A = [x*x for x in range(1,11)]      # 이런 표현을 list comprehension 이라고 합니다.
〉〉〉 print(A)
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- List comprehension의 일반적인 형태



9. 리스트 안에 for 반복문 사용하기(List Comprehension)

집합의 조건제시법 표현	List Comprehension
A = { x^2 | 1 ≦ x ≦ 10, x는 정수}	A = [x*x for x in range(1, 11)]
B = { 2^i | 0 ≦ i ≦ 10, i는 정수}	B = [2 ** i for i in range(11)]
C = { n | n ∈ A 그리고 n는 짝수 }	C = [n for n in A if n % 2 == 0]

- 중첩된 반복문을 사용할 수도 있음

```
L = [ ]
for x in range(1,4):
    for y in range(3,7):
        L.append(x*y)
print(L)
```

```
L= [x * y for x in range(1,4) for y in range(3,7)]
print(L)
```

9. 리스트 안에 for 반복문 사용하기(List Comprehension)

CODE 55 정수가 저장된 리스트 L에서 홀수만 찾아서 새로운 리스트 M에 저장하는 list comprehension 코드.

```
L = [4, 7, 8, 1, 2, 5]                  # L에는 정수가 여러 개 저장되어 있습니다.
M = [x for x in L if x%2 != 0]          # L에서 x%2 != 0인 데이터만 M에 추가합니다.
print(M)
```

[결과]
[7, 1, 5]

CODE 56 리스트 L에 있는 모든 정수들을 양수로 바꾸어 새로운 리스트 M에 저장하는 list comprehension 코드.

```
L = [3, -1, -7, 5, 10, -11, 14, 2, -8, -5]
M = [abs(n) for n in L]                 # L의 원소에 abs( ) 함수를 적용하여 M에 추가합니다.
print(M)
```

[결과]
[3, 1, 7, 5, 10, 11, 14, 2, 8, 5]

9. 리스트 안에 for 반복문 사용하기(List Comprehension)

CODE 57 영어 단어들이 저장된 리스트 words에서 가장 길이가 긴 단어를 출력하는 list comprehension 코드.

```
words = ['hello', 'python', 'beautiful', 'bookshelf', 'programming']
m = max([len(x) for x in words]) # words에 있는 단어에 len( ) 함수를 적용합니다.
print(m)
```

[결과]
11

CODE 58 다양한 문자열들이 저장된 리스트 data에서 숫자로만 구성된 문자열들을 찾아서 정수로 바꾸어 새로운 리스트 result에 저장하는 list comprehension 코드.

```
data = ['a123', '500t', '135', 'a2b5', '123!', '100', '120*', '150']
result = [int(x) for x in data if x.isdigit( )]
print(result)
```

[결과]
[135, 100, 150]

10. 정리

- 리스트 자료형과 객체에 대해 학습함.
- 리스트는 어떤 자료형도 저장할 수 있고, 중복된 데이터를 넣을 수도 있는 시퀀스 자료형임.
- mutable 자료형이라서 저장된 데이터를 언제든지 수정할 수 있음.
- 다양한 메소드를 사용할 수 있어서 코딩에 많이 사용되는 자료형임.