

# 기초 Python programming

## 7장 리스트 (list)



## 리스트 ( list )

- 학생 10명의 파이썬 성적 평균을 구하려고 한다면, 우선 학생 각각의 성적을 저장하기 위해서 다음과 같이 각자 서로 다른 변수명을 사용해서 저장해야 함(예 : `st1 = 56`, `st2 = 90`, `st3 = 76`, ..., `st10 = 14` )
- 만약 학생 수가 10000명이라면?
- 좀 더 편리하게 대량의 데이터를 저장/처리할 수 있는 리스트 자료형 이용 : 변수명을 여러 개 사용할 필요 없이, 하나의 변수명으로 여러 데이터를 저장할 수 있음 (예 : `st_score = [56, 90, 76, .....14]` )

# 리스트 ( list )

- 같은 의미의 여러 데이터를 '순서대로' 저장하고 관리해야 할 때 사용
- 순서 있는 데이터(sequence data)를 저장하는 객체
- 어떠한 데이터 형도 리스트의 원소 값이 될 수 있음
- 문자열처럼 인덱싱(indexing)과 슬라이싱(slicing)이 가능
- 형식(syntax)

**리스트명 = [원소1, 원소2, 원소3, ...]**

- 대괄호 [ ]안에 콤마로 구분하여 원소들을 표시
- 리스트의 각 원소는 변수이며, 그 값은 변경 가능
- 리스트 example

```
Ages = [21, 22, 23, 24, 25]
```

```
a = [1, 2.2, 'python']
```

```
b = ["mouse", [8, 4, 6]] # 리스트도 원소가 될 수 있음
```

# 리스트 생성

- 대괄호 [ ] 안에 직접 원소를 지정
  - 원소 없이 []만 있으면 빈 리스트 생성
- list() 함수를 이용하여 생성할 수도 있음
  - 함수의 인수로는 원소를 가진 iterable한 객체 하나를 지정할 수 있음
  - 그 객체의 원소들이 리스트의 원소로 변환
  - 인수가 없으면 빈 리스트 생성

L1 = list()	# 빈 리스트 L1를 생성
L2 = []	# 마찬가지로 빈 리스트 L2를 생성
L3 = list("abcd")	# L3 = ['a', 'b', 'c', 'd']를 생성
L4 = list((1,2,3))	# L4 = [1, 2, 3]를 생성

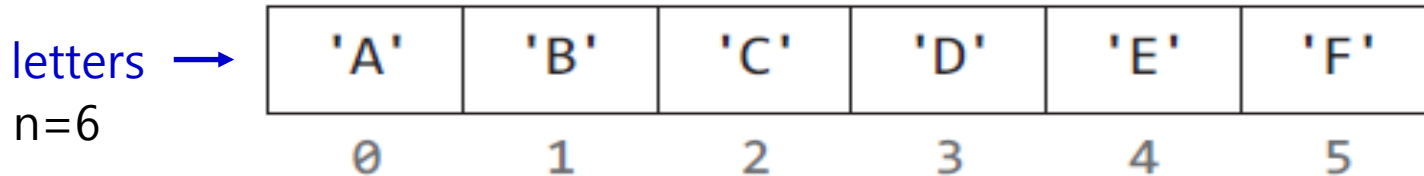
L4 = list(1,2,3)
------------------

Traceback (most recent call last):  
File "<pyshell#4>", line 1, in <module>  
L4 = list(1,2,3)  
TypeError: list expected at most 1 argument, got 3

# 리스트 indexing

- 인덱싱을 이용해 리스트의 각 원소가 가리키고 있는 값에 접근할 수 있음
- indexing 범위(문자열과 동일하게 적용)
  - $n$  : 리스트의 크기, 내장 함수 **len()**으로 값을 구할 수 있음
  - Positive indexing :  $0 \sim n-1$

```
letters = [ 'A', 'B', 'C', 'D', 'E', 'F' ]  
n = len(letters) #size of list, n = 6
```



# 리스트 indexing

- 리스트의 원소가 순서있는 데이터 형인 경우 추가 인덱싱이 가능
- 대괄호 [ ] 안에 첨자(Index) 번호 입력하여 특정 요소의 값을 참조

```
      0  1  2  3
intL = [4, 1, 8, 6]

      0  1      2      3      4
mixed = [1, 2.2, "Sogang", "대학", [8, 4, "Python"]]

NintL = len(intL)
print(NintL, len(mixed))
print(intL[0], mixed[1], mixed[4])
print(mixed[4][0], mixed[4][1])
print(mixed[4][2], mixed[4][2][0])
```

*Diagram annotations:*

- `mixed[4][0]` points to the element `8` in the nested list.
- `mixed[4][2][0]` points to the first character `P` of the string `Python` in the nested list.

*Output:*

```
#4 5
#4 2.2 [8, 4, 'Python']
#8 4
#Python P
```

# 리스트 Slicing

- 문자열과 동일하게 적용
- 형식 (L을 n개의 원소를 갖는 list라고 가정)

**$L[b : e : s]$  #  $b = \text{begin}$ ,  $e = \text{end}$ ,  $s = \text{step}$ 을 의미**

- Slicing Rule ( $b, e \geq 0$  인 경우로 설명)
  - $s > 0$  ( $b < e$  이어야 함. 아니면 빈 리스트 생성)
- $s < 0$  ( $b > e$  이어야 한다. 아니면 빈 리스트 생성)

- index를  $b$ 부터  $e-1$ 까지  $s$ 씩 증가하며 리스트를 참조  
-  $b$ 를 생략하면 0부터,  $e$ 를 생략하면 끝까지,  $s$ 를 생략하면 +1씩

$L[:e:s]$	$L[b::s]$	$L[b:e]$
-----------	-----------	----------

- index를  $b$ 부터  $e+1$ 까지  $|s|$ 씩 감소하며 리스트를 참조  
-  $b$ 를 생략하면 끝에서부터,  $e$ 를 생략하면 첫 번째 원소까지

$L[:e:s]$	$L[b::s]$
-----------	-----------

# 리스트 Slicing

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

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

- 리스트 slicing의 결과는 리스트(원소가 없거나 한 개일 경우도 리스트)
- slicing을 통하여 `a`의 일부 원소 값을 참조

```
b = a[2:5:1]      # a[2:5]와 동일. b=[2,3,4] (리스트 생성)
print(a[:6:2])    # [0, 2, 4]
print(a[1:7:2])   # [1,3,5] a[1],a[3],a[5]을 선택
print(a[0:9:3])   # [0,3,6] a[0],a[3],a[6]을 선택
print(a[3:20:3])  # [3,6] a[3],a[6]을 선택(범위 밖은 무시)
print(a[4:4])     # [] (an empty list)
print(a[4:1:1])   # [] empty list (b > e)
print(a[4:5])     # [4]
print(a[4])       # 4 (원소 참조, slicing 아님)
print(a[::-1])    # [8, 7, 6, 5, 4, 3, 2, 1, 0]
```



# Slicing을 이용한 리스트 원소 제거/교체/추가

- 리스트 `a = [0, 1, 2, 3, 4]`

list a

0	1	2	3	4
0	1	2	3	4

- 원소의 제거 = 의 좌변이 slicing이면, 우변은 iterable 객체가 와야 함

```
a[1:3] = []      # a[1], a[2] 제거
print(a)         # [0,3,4]
a[0] = 1         # a[0] 값을 1로 변경 (slicing 아님)
print(a)         # [1,3,4]
a[1:2] = []      # a[1] 제거
print(a)         # [1,4]
a[1] = []        # a[1]의 값을 empty list [ ]로 변경 (slicing 아님)
print(a)         # [ 1, [ ] ]
del a[:2]        # a[0],a[1] 원소 제거
print(a)         # [ ]
# del a[:] 는 리스트의 모든 원소를 삭제하고 빈 리스트로 만듦
```

# Slicing을 이용한 리스트 원소 제거/교체/추가

- 리스트 a = [0, 1, 2, 3, 4]

list a

0	1	2	3	4
0	1	2	3	4

- 원소의 교체(바꾸기)

= 의 좌변이 slicing이면, 우변은 iterable 객체가 와야 함

```
a[1::2] = [9,9]    # 홀수 인덱스 원소 값을 모두 9로 교체
                  # 개수가 같아야 함 [0,9,2,9,4]
a[1:3] = [1]       # a[1]과 a[2]를 1로 교체
print(a)           # [0, 1, 9, 4]
a[2:3] = [5, 6]    # a[2]를 5, 6으로 교체
print(a)           # [0,1, 5, 6, 4]
a[3:] = [8,9,10]   # a[3] 부터 순서대로 원소를 8,9,10로 교체
print(a)           # [0,1,5,8,9,10]
```

# Slicing을 이용한 리스트 원소 제거/교체/추가

- 리스트 `a = [0, 1, 2, 3, 4]`

list a	0	1	2	3	4
	0	1	2	3	4

- 리스트에서 **하나의 원소를 수정** : 인덱스 사용
- 리스트에서 **연속된 원소들 수정** : 슬라이싱 사용

```
a[1] = ['a','b']    # a[1]의 값을 list ['a','b']로 수정 (slicing 아님)
print(a)           # [0, ['a', 'b'], 2, 3, 4]
a[1] = 1           # a[1]의 값을 1로 수정 (slicing 아님). a= [0,1,2,3,4]
a[1:2] = ['a', 'b', 'c']    # a[1]의 값 1 대신 'a','b','c' 로 대체
print(a)           # [0,'a','b','c', 2, 3, 4]
```

# Slicing을 이용한 리스트 원소 제거/교체/추가

- 리스트 a = [0, 1, 2, 3, 4]

list a

0	1	2	3	4
0	1	2	3	4

- 원소의 추가

= 의 좌변이 slicing이면, 우변은 iterable 객체가 와야 함

a[1:1] = ['a','b']	# index 1에 'a','b'추가 (끼워 넣기)
print(a)	# [0,'a','b',1,2,3,4]
a[1:1] = [ ['A','B'] ]	# index 1에 리스트 ['A','B'] 추가
print(a)	# [0,['A','B'],'a','b',1,2,3,4]
a[5:5]= [5]	# a[5]에 추가
	# [0, ['A','B'], 'a', 'b', 1, 5, 2, 3, 4]
a[len(a):len(a)]= [10]	# 리스트 끝에 새 원소 a[9]를 추가
	# [0, ['A','B'], 'a', 'b', 1, 5, 2, 3, 4,10]

# Slicing example

```
A = []  
A[len(A):len(A)] = ["a", "b"]      #list (iterable 객체)  
print(A)  
A[len(A):len(A)] = "cd"           #string (iterable 객체)  
print(A)  
A[len(A):len(A)] = {"ef", "gh"}    #set (iterable 객체)  
print(A)  
A[len(A):len(A)] = ["ij"]  
print(A)  
A[len(A):len(A)] = ("ab1", "ab2") #tuple (iterable 객체)  
print(A)  
A[len(A):len(A)] = [("ab3", "ab4")]  
print(A)
```

출력

```
['a', 'b']  
['a', 'b', 'c', 'd']  
['a', 'b', 'c', 'd', 'gh', 'ef']  
['a', 'b', 'c', 'd', 'gh', 'ef', 'ij']  
['a', 'b', 'c', 'd', 'gh', 'ef', 'ij', 'ab1', 'ab2']  
['a', 'b', 'c', 'd', 'gh', 'ef', 'ij', 'ab1', 'ab2', ('ab3', 'ab4')]
```

## 내장 함수 id( )

- 모든 객체는 생성될 때 identity 라는 고유번호를 가지게 됨
- id() 함수를 사용하여 객체의 고유번호를 알 수 있음

```
>>> id(50)                # 정수형 객체 50의 고유번호
1706318880
>>> id( " Hello " )      # 문자열 " Hello " 의 고유번호
53861792
>>>
```

- 고유번호는 객체가 생성될 때 정해지며, 컴퓨터 환경에 따라 또는 생성 시간에 따라 다른 번호를 가질 수 있음
- 일단 생성되면 객체가 소멸될 때까지 변하지 않음

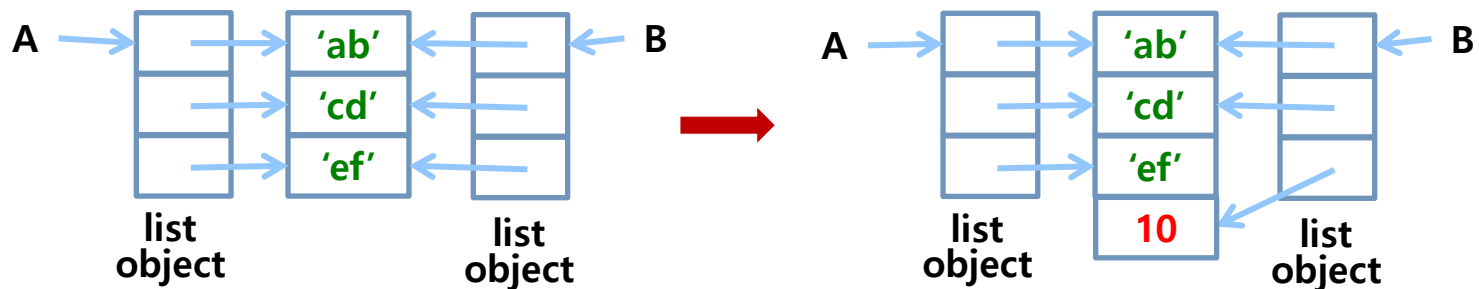
# Slicing을 이용한 리스트 복사

- 리스트 A를 복사하여 새로운 리스트 B를 생성

```
A = ['ab', 'cd', 'ef']  
B = A[:] # B = A.copy( )  
print(id(A), id(B)) # id가 다름(다른 객체)  
B[2] = 10  
print(A, B)
```

출력

```
91272376 86381832  
['ab', 'cd', 'ef'] ['ab', 'cd', 10]
```



# Slicing을 이용한 리스트 복사

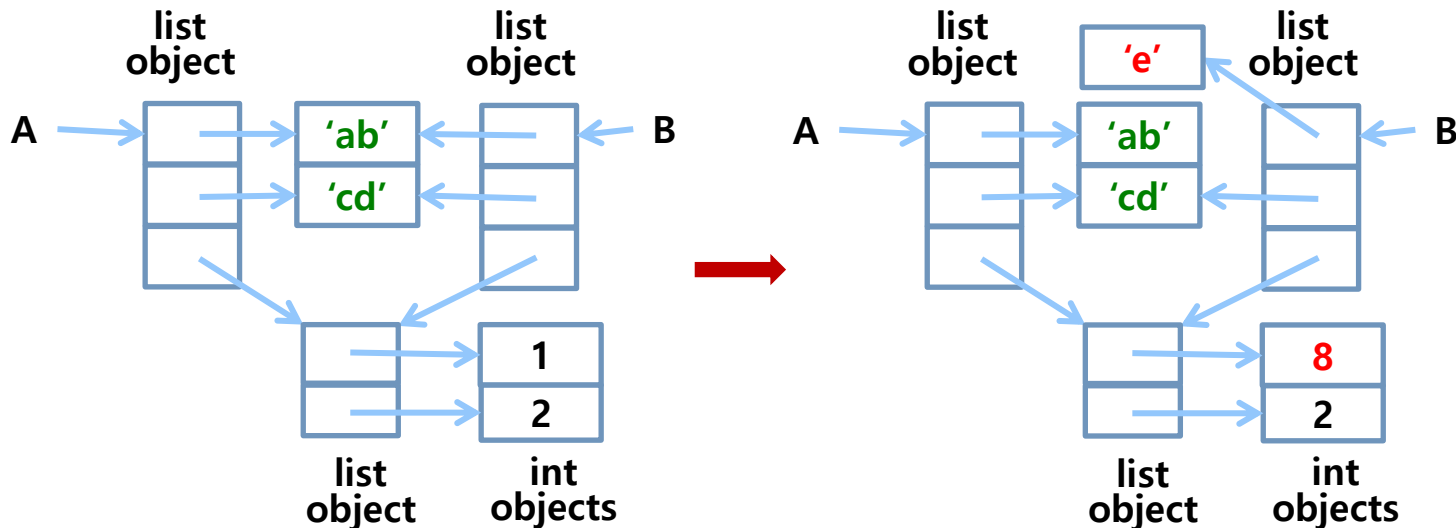
- 리스트 A를 복사하여 새로운 리스트 B를 생성

```
A = ['ab', 'cd', [1,2]]  
B = A[:]                # B = A.copy( )  
print(id(A), id(B))    # id가 다름(다른 객체)  
B[0] = 'e'             # B의 원소 값만 바뀜  
B[2][0] = 8             # A와 B 모두 값이 바뀜(원소가 리스트인 경우:문제 발생)  
print(A, B)
```

copy 모듈의 함수  
deepcopy()를 사용하면  
리스트의 내부 구조를 모두  
복사

출력

```
91362568 100355248  
['ab', 'cd', [8, 2]] ['e', 'cd', [8, 2]]
```





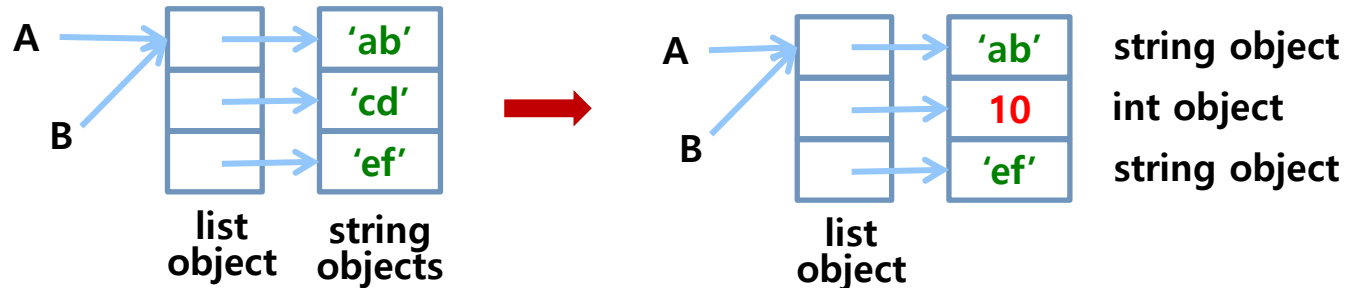
# 리스트 복사와 다른, 참조하기

- 같은 리스트 참조하기 ('=' 이용)  
: 동일한 객체를 다른 이름으로 참조하기

```
A = ['ab', 'cd', 'ef']  
B = A # 같은 object를 참조(객체 복사가 아님)  
print(id(A), id(B)) # id 동일(같은 객체를 가리키기 때문)  
B[1] = 10  
print(A, B) # 같은 데이터 값 출력
```

출력

```
85046256 85046256  
['ab', 10, 'ef'] ['ab', 10, 'ef']
```



## 리스트의 연산

- Concatenation operator + (리스트의 연결 연산)  
: 리스트들을 연결하여 새로운 리스트로 만듦
- Repetition Operator \* (리스트의 반복 연산)  
: **list \* n** 은 list의 item들을 n번 반복하여 새로운 list를 만듦
- Membership operator in, not in (문자열에서와 동일)

```
>>> L = [1,3,5] ; M = [2,4,6]
>>> LM = L + M
>>> print(L, M, LM)      # L과 M은 변하지 않음
[1, 3, 5] [2, 4, 6] [1, 3, 5, 2, 4, 6]
>>> L * 3                 # 리스트 L을 3회 반복한 리스트를 반환
[1, 3, 5, 1, 3, 5, 1, 3, 5]
>>> print( 3 in L )      # 3 이 리스트 L의 원소이므로 True 반환
True
>>> print( 8 not in L )
True
>>>
```

# 리스트 Method

- 메소드 (method) : 데이터 객체에 대해서 정해진 일을 처리하는 함수
- 메소드 호출 형식

**objectName.method()**

- 리스트의 메소드  
: dir() 내장 함수는 객체가 가지고 있는 변수나 함수(메소드)들을 나열

```
>>> dir(list)
['__add__', '__class__', ..... '__subclasshook__', 'append', 'clear',
'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
```

# 리스트 Method

method	Description (x: 리스트, a: 임의의 객체)
<b>x.append(a)</b>	데이터 a를 리스트 x의 끝에 추가
<b>x.extend(L)</b>	리스트 L의 모든 원소를 리스트 x의 마지막에 추가
<b>x.insert(i, a)</b>	a를 리스트 x의 index i에 추가
<b>x.remove(a)</b>	리스트 x에서 원소 값이 데이터 a인 첫 원소 제거 (반환 값 없음)
<b>x.pop( )</b>	x의 <b>마지막 원소</b> 제거 및 <b>반환</b>
<b>x.pop(i)</b>	x[i]를 x에서 제거하고 그 <b>값을 반환</b>
<b>x.clear( )</b>	리스트 x의 모든 원소를 삭제. 빈 리스트가 됨
<b>x.index(a)</b>	리스트 x에서 원소 값이 a인 첫 번째 원소의 <b>index를 반환</b>
<b>x.count(a)</b>	리스트 x에서 a 값과 같은 원소의 <b>개수를 반환</b>
<b>x.sort( )</b>	x의 원소들을 오름차순으로 정렬 내림차순으로 정렬하려면 x.sort(reverse=True) 사용
<b>x.reverse( )</b>	x의 원소들을 역으로 재 배치 (정렬과 다름)
<b>x.copy( )</b>	a shallow copy of the list (y = x[:]와 동일)

## 리스트 Method

```
>>> L = [1, 2, 3, 4]
>>> L.append(5)           # 5를 끝에 추가. L[ len(L):len(L) ]= [5]
>>> print(L)
[1, 2, 3, 4, 5]
>>> L.insert(2,10)        # 10을 인덱스 2에 삽입. L[2:2]= [10]
>>> print(L)
[1, 2, 10, 3, 4, 5]
>>> L.pop()              # 마지막 원소를 제거 및 값 반환
5
>>> print(L)
[1, 2, 10, 3, 4]
>>> L.pop(2)             # 인덱스 2의 원소를 제거 및 값 반환
10
>>> print(L)
[1, 2, 3, 4]
>>> L.remove(3)          # [1, 2, 4], 원소 값 3을 제거
```

## 리스트 Method

```
>>> L = ['M', 'A', 'a', 'm', 'a', 'R', 'r', 'a']
>>> L.index('a')           # 데이터가 여러 개이면 첫번째 인덱스 반환
2
>>> L.count('a')
3
>>> A = [ ]                # 빈 리스트
>>> A.append('red')
>>> A.append('blue')
>>> print(A)
['red', 'blue']
>>> A.clear()              # 리스트의 모든 원소 삭제
>>> print(A)
[ ]
>>> A = ['red', 'blue', 'yellow']
>>> del A[:]               # 모든 원소를 삭제하고 빈 리스트로 만듦
>>> print(A)
[ ]
>>> del A                  # 리스트 A 자체 삭제
>>> print(A)
NameError: name 'A' is not defined
```

## 리스트 Method

```
>>> L = [1, 5, 3]
>>> M = [19, 2]
>>> L.extend(M)    # L에 M( iterable 객체 허용)를 더하여 확장
>>> print(L)
[1, 5, 3, 19, 2]
>>> L.sort()       # 오름차순으로 정렬
>>> print(L)
[1, 2, 3, 5, 19]
>>> L.sort(reverse=True)  # 내림차순으로 정렬
>>> print(L)
[19, 5, 3, 2, 1]
>>> L = [4, 6, 2, 9, 1]
>>> L.reverse()
>>> print(L)
[1, 9, 2, 6, 4]
```

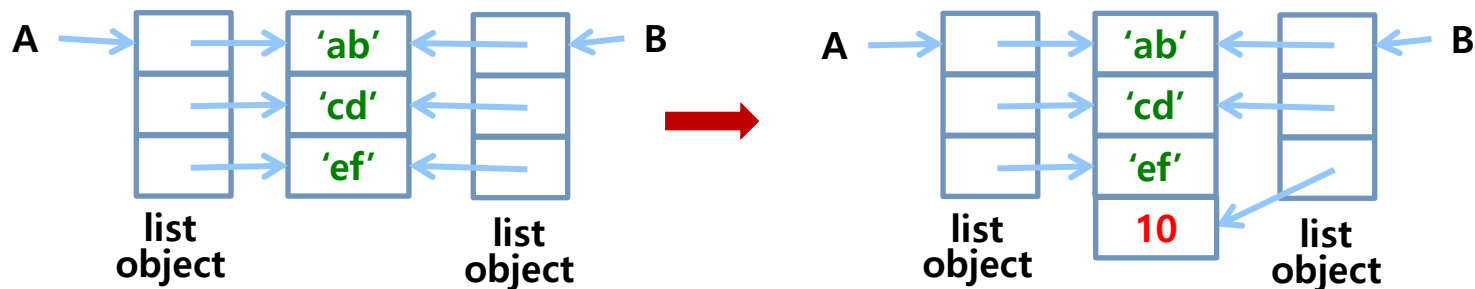
# 리스트 Method

- 리스트 복사하기 – copy() : 복사해서 새로운 리스트 객체를 생성

```
A = ['ab', 'cd', 'ef']  
B = A.copy()    # B = A[:]  
print(id(A), id(B)) # id가 다름(다른 객체)  
B[2] = 10  
print(A, B)
```

출력

```
91272376 86381832  
['ab', 'cd', 'ef'] ['ab', 'cd', 10]
```





## 내장함수 (인자가 리스트 자료형 일 때)

- 문자열, 튜플, 집합, 사전 등도 사용 가능.

function	Description(인수 x : 리스트)
<b>all(x)</b>	x의 모든 원소가 True(i.e. != 0)이거나, 또는 x가 빈 리스트이면 True 반환
<b>any(x)</b>	x에 True인(i.e. != 0인) 원소가 한 개라도 있으면 True, 또는 x가 빈 리스트이면 False 반환
<b>enumerate(x)</b>	x의 모든 원소에 대해 튜플 (index, 원소 값)을 얻을 수 있는 enumerate object를 반환. index = 0, 1, 2, ...
<b>len(x)</b>	x의 원소 개수를 반환
<b>list(y)</b>	Iterable한 y를 리스트로 변환하여 반환
<b>max(x)</b>	x의 원소 중 최대 값 반환(비교 불가이면 TypeError)
<b>min(x)</b>	x의 원소 중 최소 값 반환(비교 불가이면 TypeError)
<b>sorted(x)</b>	x의 원소를 정렬한 리스트 반환. x는 불변이고 오름차순 또는 내림차순으로 정렬( <b>sorted(x, reverse=True)</b> )
<b>sum(x)</b>	x의 원소 합을 반환(더할 수 없으면 TypeError)

# 내장함수 (인자가 리스트 자료형 일 때)

- 예제

```
L = [1, 9, 4]
L = sorted(L, reverse=True)      # L을 내림차순 정렬
print(L)                         # [9, 4, 1]
print(sum(L))                    # 14      L[0]+L[1]+L[2]
L = ["Jan", "Apr", "Sep"]        # 문자열은 사전식 순서
print(min(L), max(L))            # Apr Sep
S = sorted(L)                    # L을 오름차순 정렬한 리스트 반환
print(S)                         # ['Apr', 'Jan', 'Sep']
M = list(enumerate(L))           # L의 튜플 (index, value)들 생성
print(M)                         # [(0, 'Jan'), (1, 'Apr'), (2, 'Sep')]
L = [1,'2',3,'4']
print(max(L),min(L))             # TypeError (비교 불가)
```

```
>>> L = [100, 900, 400, 300]
>>> E = enumerate(L)
>>> E
<enumerate object at 0x0342F308>
>>> type(E)
<class 'enumerate'>
>>> EtoL = list(E)
>>> EtoL
[(0, 100), (1, 900), (2, 400), (3, 300)]
>>> L = ['Apr', 'Jan', 'Sep']
>>> M = list(enumerate(L))
>>> M
[(0, 'Apr'), (1, 'Jan'), (2, 'Sep')]
>>>
```

## 메소드(함수)의 반환값 처리

- 리스트 메소드 `sort()` / 내장 함수 `sorted()`  
: 메소드(함수)는 반환 값없이 정의된 기능을 실행만 하는 경우와 실행 후 반환 값이 있는 경우가 있음

```
L = [1, 9, 4, 2]
M = sorted(L, reverse=True)      # L의 데이터를 내림차순 정렬한 새 리스트를 반환
print(L, M)                     # [1, 9, 4, 2] [9, 4, 2, 1]
L.sort()
print(L)                         # [1, 2, 4, 9]
L = [1, 9, 4, 2]
L1 = sorted(L, reverse=True)     # L을 내림차순 정렬
L2 = L.sort()                   # L의 데이터를 오름차순 정렬, 반환값은 없음
print(L1)                       # [9, 4, 2, 1]
print(L2)                       # None, 메소드 sort()는 반환값이 없기 때문
print(L)                        # [1, 2, 4, 9]
```