

파이썬 프로그래밍

리스트의 활용



한국기술교육대학교
온라인평생교육원

▣ 리스트 정렬하기

1. 리스트의 sort 메소드

- L.sort() 함수는 리스트 L 자체를 변경하며 리턴값을 반환하지 않는다.

```
L = [1, 5, 3, 9, 8, 4, 2]
print L.sort()
print L
```

```
None
[1, 2, 3, 4, 5, 8, 9]
```

- sort 함수 자체는 리턴값을 반환하지 않고 리스트 L 자체를 변경

▣ 리스트 정렬하기

1. 리스트의 sort 메소드

- 파이썬은 디폴트로 `cmp(a, b)` 내장 함수를 이용하여 정렬 방식을 결정한다.
- `cmp(a, b)`
 - if $a < b$: return -1
 - if $a > b$: return 1
 - if $a == b$: return 0.

```
print cmp(1,2)
```

```
print cmp(5,2)
```

```
print cmp('abc', 'abc')
```

```
-1  
1  
0
```

- `cmp`는 기본적으로 2개의 인자를 가지며 2개의 인자의 형은 같음
- 3가지 값을 리턴함 (1, 0, -1)
- $1 < 2 \rightarrow -1$ 리턴
- $5 > 2 \rightarrow 1$ 리턴
- `'abc' == 'abc' \rightarrow 0` 리턴
- `cmp`가 `sort`의 핵심적인 내장함수

▣ 리스트 정렬하기

1. 리스트의 sort 메소드

- 기본 정렬 방식을 변경하려면 `cmp(a, b)`와 같은 비교 함수를 직접 만들어서 `sort()` 함수의 인자로 넣는다.

```
def mycmp(a1, a2): # 대소관계에 따른 순서를 반대로 바꾸었음
    return cmp(a2, a1)
```

```
L = [1, 5, 3, 2, 4, 6]
L.sort(mycmp) # 역순으로 정렬
print L
```

```
[6, 5, 4, 3, 2, 1]
```

- `sort`의 인자로서 `mycmp` 함수를 넣어주면 비교하는 함수로 활용
- `mycmp`는 대소관계에 따른 순서를 반대로 바꿔줌
- $a1 > a2 \rightarrow \text{cmp}$ 함수는 1 리턴, `mycmp` 함수는 -1 리턴

■ 리스트 정렬하기

1. 리스트의 sort 메소드

- 여러 튜플을 요소로 지닌 리스트인 경우, 튜플의 첫번째 값이 아닌 다른 위치에 있는 값을 기준으로 정렬

```
def cmp_1(a1, a2):  
    return cmp(a1[1], a2[1])
```

```
def cmp_2(a1, a2):  
    return cmp(a1[2], a2[2])
```

```
L = [ ('lee', 5, 38), ('kim', 3, 28), ('jung', 10, 36)]  
L.sort()  
print 'sorted by name:', L
```

```
L.sort(cmp_1)  
print 'sorted by experience:', L
```

```
L.sort(cmp_2)  
print 'sorted by age:', L
```

```
sorted by name: [('jung', 10, 36), ('kim', 3, 28), ('lee', 5, 38)]  
sorted by experience: [('kim', 3, 28), ('lee', 5, 38), ('jung', 10, 36)]  
sorted by age: [('kim', 3, 28), ('jung', 10, 36), ('lee', 5, 38)]
```

- 리스트의 원소가 튜플
- 튜플안의 원소가 총 3개 (문자열, 정수, 정수)
- sort하면 각 튜플의 첫 번째 원소를 바탕으로 비교
- 알파벳 순서로 정렬됨
- cmp_1 → 각 튜플의 두 번째 원소를 기준으로 부름
- cmp_2 → 각 세 번째 원소를 cmp의 내장함수로 넣어줌

▣ 리스트 정렬하기

1. 리스트의 sort 메소드

- sort() 함수 인자로 reverse 값을 받을 수 있다.
- 디폴트 reverse 인자값은 False
- reverse 인자값을 True로 주면 역순으로 정렬됨

```
L = [1, 6, 3, 8, 6, 2, 9]
L.sort(reverse = True)
print L
```

```
[9, 8, 6, 6, 3, 2, 1]
```

- reverse 함수는 true 아니면 false 값을 반환하는 변수인자
- reverse 인자의 기본적 디폴트가 false → 오름차순으로 정렬
- reverse이 true 디폴트 → 내림차순으로 정렬

▣ 리스트 정렬하기

1. 리스트의 sort 메소드

- sort() 함수 인자로 key에 함수를 넣어줄 수 있다.
 - key 인자에 함수가 할당되어 있으면 각 리스트 원소에 대해 비교함수 호출 직전에 key 함수를 먼저 호출한다.

```
L = ['123', '34', '56', '2345']
```

```
L.sort()
```

```
print L
```

```
L.sort(key=int)
```

```
print L
```

```
['123', '2345', '34', '56']
```

```
['34', '56', '123', '2345']
```

- L = [문자열]
- L.sort() → 문자열을 가지고 정렬
- 123 중에 문자로 생각하면 1이 가장 작은 문자
- 안의 숫자 형태가 아니라 문자 형태를 기준으로 정렬
- int('123') → 문자열을 정수로 바꿔주는 내장함수
- L.sort(key=int) → int라는 내장함수를 key에 넣음
- sort가 cmp함수 넣을 때, 각 인자마다 int한 결과를 cmp 함수 인자로 넣어줌
- 문자열에 대한 cmp 비교가 아니라 각 숫자 형태를 비교
- 비교하는 순간에 정수로 바꿔 비교 후 다시 문자열로 반환

■ 리스트 정렬하기

2. sorted 내장 함수

- sorted() 내장함수는 L 자체에는 내용 변경 없이 정렬이 되어진 새로운 리스트를 반환한다.

```
L = [1, 6, 3, 8, 6, 2, 9]
newList = sorted(L)
print newList
print L
```

```
[1, 2, 3, 6, 6, 8, 9]
[1, 6, 3, 8, 6, 2, 9]
```

- sorted 함수는 인자에 L을 넣음

```
for ele in sorted(L):
    print ele,
```

```
1 2 3 6 6 8 9
```

- ele 다음에 콤마(,)가 있어서 옆으로 프린트 됨

■ 리스트 정렬하기

2. sorted 내장 함수

- sorted() 함수의 두번째 인자로 cmp 함수 지정 가능

```
def mycmp(a1, a2):    # 대소관계에 따른 순서를 반대로 바꾸었음
    return cmp(a2, a1)
```

```
L = [1, 5, 3, 2, 4, 6]
print sorted(L, mycmp) # 역순으로 정렬
print L
```

```
[1, 2, 3, 4, 5, 6]
[1, 5, 3, 2, 4, 6]
```

- sorted 함수도 두 번째 인자로 cmp 함수 지정 가능

- 인자로 reverse와 key 지정 가능

```
L = [1, 6, 3, 8, 6, 2, 9]
print sorted(L, reverse=True)
```

```
L = ['123', '34', '56', '2345']
print sorted(L, key=int)
```

```
[9, 8, 6, 6, 3, 2, 1]
['34', '56', '123', '2345']
```

- sorted 함수는 reverse와 key 함수를 인자로 사용 가능
- reverse에 true가 들어가면 역순 순서대로 정렬
- key=int 함수 넣어주면 문자열을 정수로 바꿔 정렬
- 기존의 L은 변화가 없음

▣ 리스트 정렬하기

3. L.reverse() 와 reversed() 내장 함수

- L.reverse()도 반환값이 없다.
 - 즉, L 자체를 역순으로 뒤집는다.
 - [주의] 역순 정렬이 아니다.

```
L = [1, 6, 3, 8, 6, 2, 9]
print L.reverse()
print L
```

```
None
[9, 2, 6, 8, 3, 6, 1]]
```

- L.reverse() → 그대로 뒤집어서 거꾸로 바꿔줌 → 반환되는 것 X

```
L = [1, 6, 3, 8, 6, 2, 9]
L.reverse()    # 역순으로 뒤집는다.
for ele in L:
    print ele + 2,

print
L.reverse()    # 다시 원상태로 복귀시킨다.
print L
```

```
11 4 8 10 5 8 3
[1, 6, 3, 8, 6, 2, 9]
```

- 1, 6, 3, 8, 6, 2, 9 뒤집은 각 값에 2를 더한 값 출력
- print = 한 줄 띄우기
- reverse 후 reverse → 원래대로

▣ 리스트 정렬하기

3. L.reverse() 와 reversed() 내장 함수

- reversed() 내장 함수는 리스트 내부의 각 원소들의 순서를 반대로 뒤집은 리스트를 반환함.
 - sorted() 내장 함수와 유사하게 원래 리스트 내용에는 변화 없음

```
L = [1, 6, 3, 8, 6, 2, 9]
print L
for ele in reversed(L):
    print ele + 2,
print
print L
```

```
[1, 6, 3, 8, 6, 2, 9]
11 4 8 10 5 8 3
[1, 6, 3, 8, 6, 2, 9]
```

- reversed(L) → L을 뒤집은 새로운 리스트 반환
- 기존의 L은 변함이 없음 = sort 함수와 같은 성격

파이썬 프로그래밍

리스트의 활용



한국기술교육대학교
온라인평생교육원

■ 리스트 내포

1. 일반적인 리스트 생성법

```
L = []  
for k in range(10):  
    L.append(k*k)  
  
print L
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- `range(10)` = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- $2^2, 3^2, 4^2, \dots, 9^2$

• 위 코딩은 리스트 내포 리터럴 방식을 활용해서 아래와 같이 변경할 수 있다.

```
L = [k * k for k in range(10)]  
print L
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- 리스트 내포 = 리스트 안에 실제 포함되어야 할 원소가 식으로 들어감
- `for` 의 변수가 식으로 들어간 변수와 동일해야함
- 0, 1, 2, 3...이 `k`에 할당될 때마다 식을 진행하여 리스트의 원소로 할당

▣ 리스트 내포

2. 리스트 내포 리터럴

```
[expression for expr1 in sequence1
    for expr2 in sequence2
    ...
    for exprN in sequenceN
    if condition]
```

- expression의 평가 결과 반드시 한 개의 원소가 나와야 한다.
 - 틀린 예: [x, y for x in seq1 for u in seq2]
- 만약 두 개의 이상의 평가 결과가 나오면 튜플 등으로 감싸 주어야 한다.
 - 올바른 예: [(x, y) for x in seq1 for u in seq2]
- 위 리터럴은 다음의 일반적인 for 문의 리터러와 동일

```
l = []
for expr1 in sequence1:
    for expr2 in sequence2:
        ...
        for exprtN in sequenceN:
            if condition:
                l.append(expression)
```

- k*k 또는 K % 10 등등 → expression → 식
- expr1, expr2의 값들이 expression 안에 쓰임
- if condition → if 뒤에 있는 조건 충족 시 expression 안 변수 적용
- 리스트 내포 안 각각의 for 문을 중첩된 포문 형태로 작성
- for 문 앞에 나오는 평가 결과는 반드시 한 개 원소
- 튜플로 묶어주면 하나의 원소로 취급 가능

▣ 리스트 내포

2. 리스트 내포 리터럴

```
L = [k * k for k in range(10) if k % 2] # 홀수의 제곱만 리스트로 형성  
print L
```

```
[1, 9, 25, 49, 81]
```

- $k=0 \rightarrow 0 \% 2 = 0 \rightarrow \text{if } 0 \rightarrow \text{결과가 false로 expression 수행 X}$
- $k=1 \rightarrow 1 \% 2 = 1 \rightarrow \text{if 결과가 true로 expression 수행}$
- $k=2 \rightarrow 2 \% 2 = 0 \rightarrow \text{if } 0 \rightarrow \text{결과가 false로 expression 수행 X}$
- $k=3 \rightarrow 3 \% 2 = 1 \rightarrow \text{if 결과가 true로 expression 수행}$

• 위의 리스트 내포 코드는 아래와 동일

```
L = []  
for k in range(10):  
    if k % 2:  
        L.append(k*k)  
print L
```

```
[1, 9, 25, 49, 81]
```

▣ 리스트 내포

2. 리스트 내포 리터럴

- 20보다 작은 2의 배수와 3의 배수에 대해 그 두 수의 합이 7의 배수인 것들에 대해 그 두 수의 곱을 출력하는 코드

```
L = [(i, j, i*j) for i in range(2, 20, 2) for j in range(3, 20, 3) if (i + j) % 7 == 0]
print L
```

```
[(2, 12, 24), (4, 3, 12), (6, 15, 90), (8, 6, 48), (10, 18, 180), (12, 9, 108),
(16, 12, 192), (18, 3, 54)]
```

- `range(2, 20, 2) = [2, 4, 6, 8, 10, 12, 14, 16, 18]`
- `range(3, 20, 3) = [3, 6, 9, 12, 15, 18]`
- $i \leftarrow$ 20보다 작은 2의 배수, $j \leftarrow$ 20보다 작은 3의 배수
- $2 + 12 = 14 \leftarrow$ 7의 배수
- $4 + 3 = 7 \leftarrow$ 7의 배수
- $18 + 3 = 21 \leftarrow$ 7의 배수

- 두 개의 시퀀스 자료형에 대해 각각의 원소에 대한 쌍을 튜플 형태로 만들면서 리스트에 저장하는 코드

```
seq1 = 'abc'
seq2 = (1, 2, 3)
print [(x, y) for x in seq1 for y in seq2]
```

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

- `seq1` → 문자열, `seq2` → 튜플
- 첫 번째 `for`문 = `seq1`에서 `x`, 두 번째 `for`문 = `seq2`에서 `y`
- `x`가 `a`인 경우, `b`인 경우, `c`인 경우 `y`는 1, 2, 3 반복 → 9쌍

■ 리스트 내포

2. 리스트 내포 리터럴

```
words = 'The quick brown fox jumps over the lazy dog'.split()
stuff = [[w.upper(), w.lower(), len(w)] for w in words]
for i in stuff:
    print i
```

```
['THE', 'the', 3]
['QUICK', 'quick', 5]
['BROWN', 'brown', 5]
['FOX', 'fox', 3]
['JUMPS', 'jumps', 5]
['OVER', 'over', 4]
['THE', 'the', 3]
['LAZY', 'lazy', 4]
['DOG', 'dog', 3]
```

- `split ()` → 공백을 기준으로 문자열 잘라 리스트 만들기
- `stuff`는 리스트 내포 문법 형태
- 리스트 내포 안의 원소는 3개의 원소를 가진 리스트 형태로 하나
- `upper` : 대문자로 변환, `lower`: 소문자로 변환, `len`: 단어의 길이
- `i` 자체가 리스트