리스트 메소드 정리

메소드	설명
append()	요소를 리스트의 끝에 추가한다.
extend()	리스트의 모든 요소를 다른 리스트에 추가한다.
insert()	지정된 위치에 항목을 삽입한다.
remove()	리스트에서 항목을 삭제한다.
pop()	지정된 위치에서 요소를 삭제하여 반환한다.
clear()	리스트로부터 모든 항목을 삭제한다.
index()	일치되는 항목의 인덱스를 반환한다.
count()	인수로 전달된 항목의 개수를 반환한다.
sort()	오름차순으로 리스트 안의 항목을 정렬한다.
reverse()	리스트 안의 항목의 순서를 반대로 한다.
copy()	리스트의 복사본을 반환한다.

리스트에서 사용할 수 있는 내장 함수

함수	설명
round()	주어진 자리수대로 반올림한 값을 반환한다.
reduce()	특정한 함수를 리스트 안의 모든 요소에 적용하여 결과값을 저장하고 최종 합계값만을 반환한다.
sum()	리스트 안의 숫자들을 모두 더한다.
ord()	유니코드 문자의 코드값을 반환한다.
cmp()	첫 번째 리스트가 두 번째 보다 크면 1을 반환한다.
max()	리스트의 최대값을 반환한다.
min()	리스트의 최소값을 반환한다.
all()	리스트의 모든 요소가 참이면 참을 반환한다.
any()	리스트 안의 한 요소라도 참이면 참을 반환한다.
len()	리스트의 길이를 반환한다.
enumerate()	리스트의 요소들을 하나씩 반환하는 객체를 생성한다.
accumulate()	특정한 함수를 리스트의 요소에 적용한 결과를 저장하는 리스트를 반환한다.
filter()	리스트의 각 요소가 참인지 아닌지를 검사한다.
map()	특정한 함수를 리스트의 각 요소에 적용하고 결과를 담은 리스트를 반환한다.

내장 함수 예

```
numbers =[10,20,30,40,50]

print("합=",sum(numbers))          # 항목의 합계를 계산한다.
print("최대값=",max(numbers))      # 가장 큰 항목을 반환한다.
print("최소값=",min(numbers))      # 가장 작은 항목을 반환한다
```

```
합= 150
최대값= 50
최소값= 10
```

내장 함수

성적 처리 프로그램

- 학생들의 성적을 사용자로부터 입력받아서 리스트에 저장한다. 성적의 평균을 구하고 최대점수, 최소점수, 80점 이상 성적을 받은 학생의 숫자를 계산하여 출력해보자.

```
성적을 입력하시요: 10
성적을 입력하시요: 20
성적을 입력하시요: 60
성적을 입력하시요: 70
성적을 입력하시요: 80
성적 평균= 48.0
최대점수= 80
최소점수= 10
80점 이상= 1
```

Lab: 리스트에서 2번째로 큰 수 찾기

- 정수들이 저장된 리스트에서 두 번째로 큰 수를 찾아보자.

list1 = [1, 2, 3, 4, 15, 99]

두 번째로 큰 수= 15

콘테스트 평가

- 심판들의 점수가 리스트에 저장되어 있다고 가정하고 최소값과 최대값을 리스트에서 제거하는 프로그램을 작성해보자.

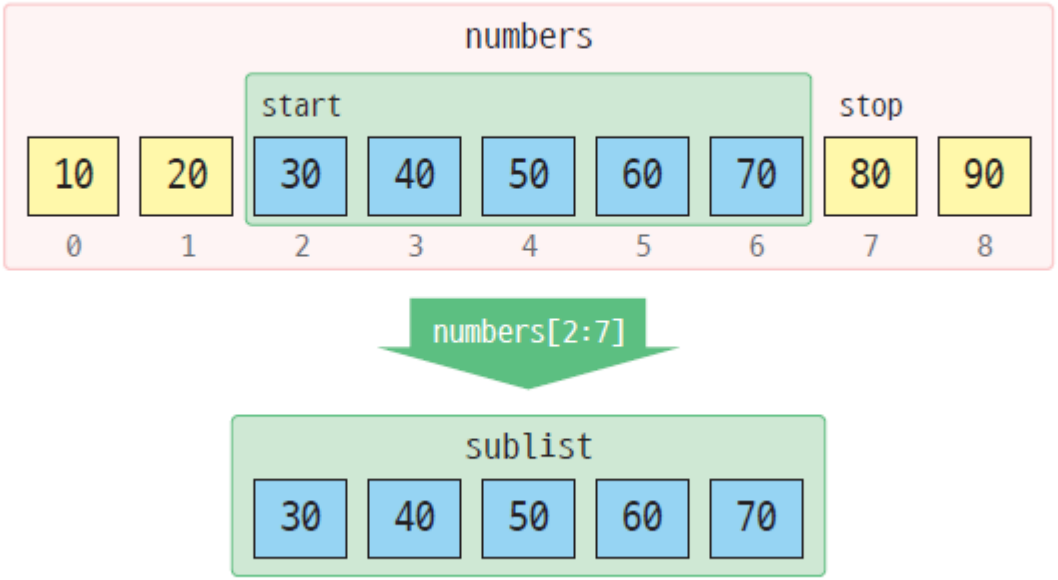
```
제거전 [10.0, 9.0, 8.3, 7.1, 3.0, 9.0]
제거후 [9.0, 8.3, 7.1, 9.0]
```

슬라이싱

Syntax: 슬라이싱 #1

형식 리스트[start : stop]

예
```
numbers = [ 10, 20, 30, 40, 50, 60, 70, 80, 90 ]
sublist = numbers[2:7]
```

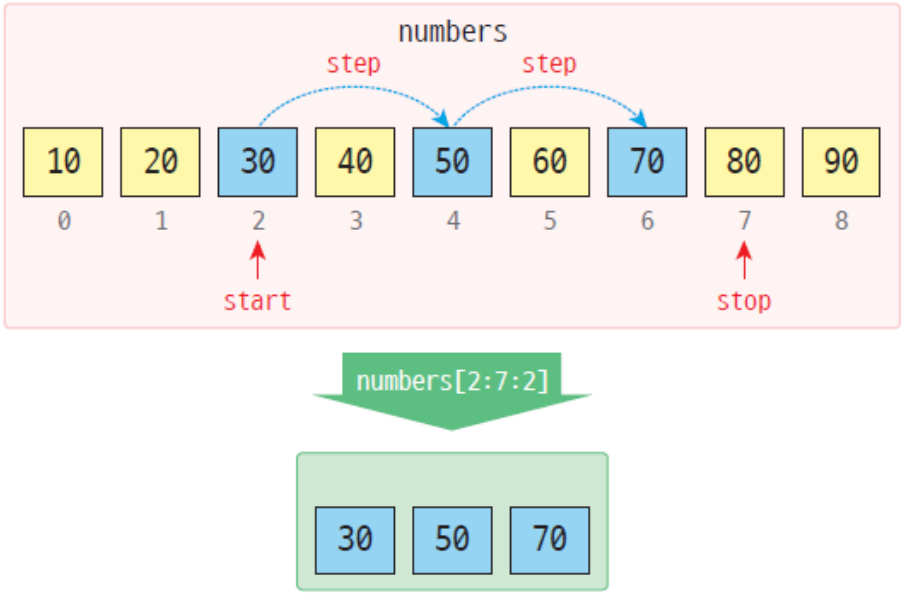


고급 슬라이싱

Syntax: 슬라이싱 #2

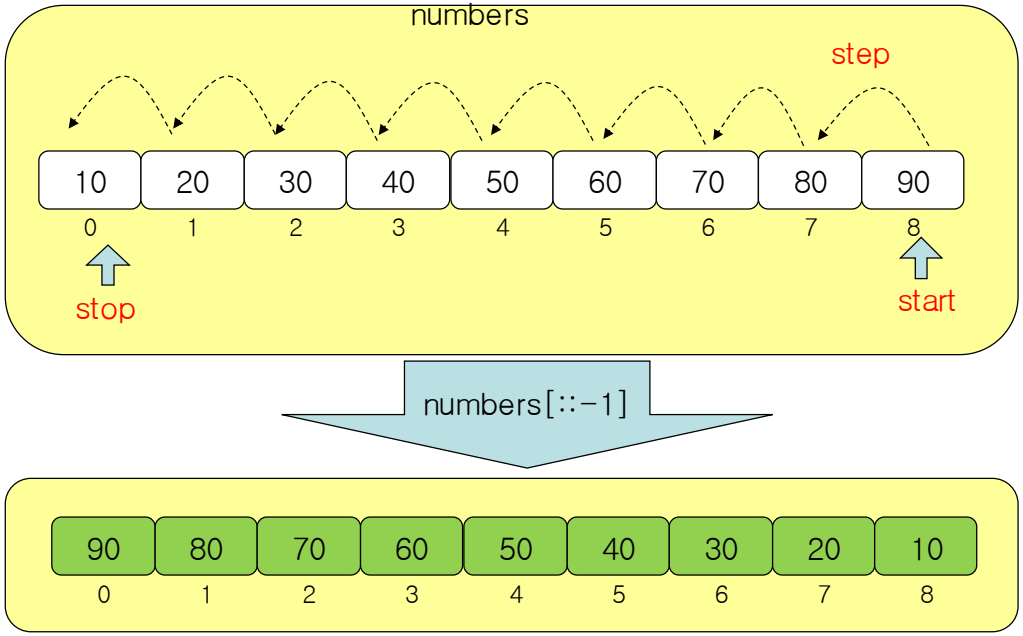
형식 리스트[start : stop : step]

예
```
numbers = [ 10, 20, 30, 40, 50, 60, 70, 80, 90 ]
sublist = numbers[2:7:2]
```



리스트를 역순으로 만드는 방법

```
>>> numbers = [ 10, 20, 30, 40, 50, 60, 70, 80, 90 ]
>>> numbers[:: -1]
[90, 80, 70, 60, 50, 40, 30, 20, 10]
```



리스트 변경

```
>>> lst = [1, 2, 3, 4, 5, 6, 7, 8]
>>> lst[0:3] = ['white', 'blue', 'red']
>>> lst
['white', 'blue', 'red', 4, 5, 6, 7, 8]
```

리스트 일부 변경

```
>>> lst = [1, 2, 3, 4, 5, 6, 7, 8]
>>> lst[::2] = [99, 99, 99, 99]
>>> lst
[99, 2, 99, 4, 99, 6, 99, 8]
```

99를 중간에 추가한다.

```
>>> lst = [1, 2, 3, 4, 5, 6, 7, 8]
>>> lst[:] = [ ]
>>> lst
[]
```

리스트의 모든 요소를 삭제한다.

리스트 변경

```
numbers = list(range(0, 10))          # 0에서 시작하여 9까지를 저장하는 리스트
print(numbers)
del numbers[-1]                       # 마지막 항목을 삭제한다.
print(numbers)
```

리스트의 특정 요소 삭제

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

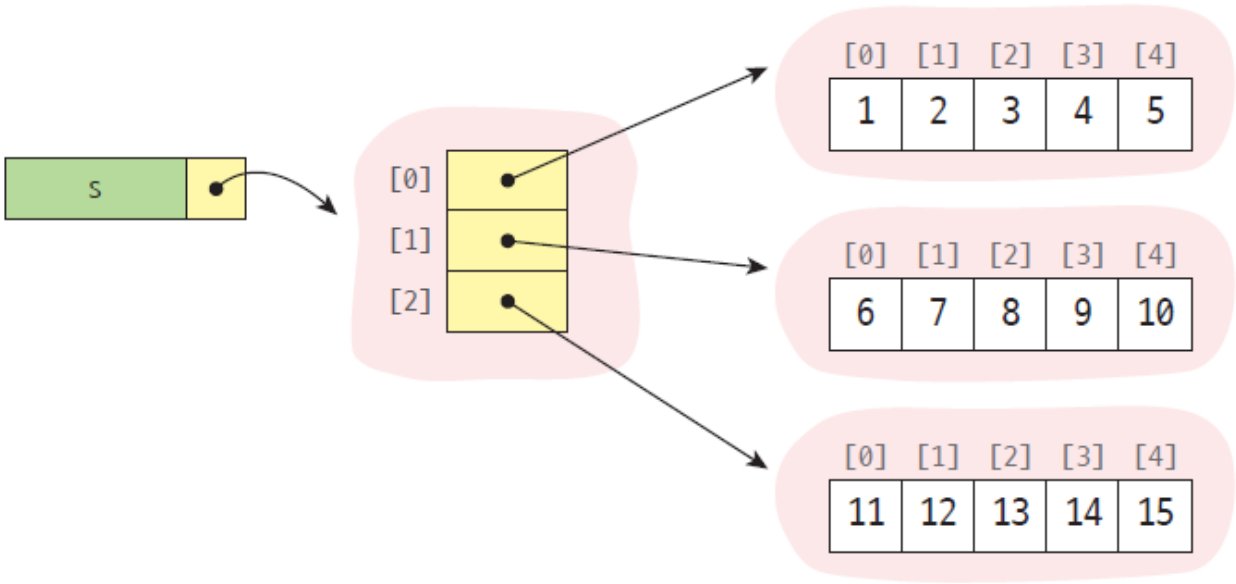
2차원 리스트

- 2차원 리스트 == 2차원 테이블


```
# 2차원 리스트를 생성한다.
s = [
        [ 1, 2, 3, 4, 5 ] ,
        [ 6, 7, 8, 9, 10 ],
        [11, 12, 13, 14, 15 ]
]
print(s)
```

2차원 리스트의 구현

- 리스트의 리스트로 구현된다.



2차원 리스트 요소 접근

```
s = [       [ 1, 2, 3, 4, 5 ] ,
            [ 6, 7, 8, 9, 10 ],
            [11, 12, 13, 14, 15 ] ]

# 행과 열의 개수를 구한다.
rows = len(s)
cols = len(s[0])

for r in range(rows):
            for c in range(cols):
                        print(s[r][c], end=",")
            print()
```

```
1,2,3,4,5,
6,7,8,9,10,
11,12,13,14,15,
```

Lab: 전치 행렬 계산

- 행렬의 전치 연산을 파이썬으로 구현해보자. 인공 지능을 하려면 행렬에 대하여 잘 알아야 한다. 중첩 된 **for** 루프를 이용하면 행렬의 전치를 계산할 수 있다.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

원래 행렬= [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
전치 행렬= [[1, 4, 7], [2, 5, 8], [3, 6, 9]]

Solution:

```
transposed = []
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print("원래 행렬=", matrix)
# 열의 개수만큼 반복한다.
for i in range(len(matrix[0])):
    transposed_row = []
    for row in matrix:                          # 행렬의 각 행에 대하여 반복
        transposed_row.append(row[i])# i번째 요소를 row에 추가한다.
    transposed.append(transposed_row)

print("전치 행렬=", transposed)
```