26장

리스트 고급 연산

26장 리스트 고급 연산

26.1 리스트 정렬 및 원소 순서 거꾸로 하기

26.1 리스트 정렬 및 원소 순서 거꾸로 하기

- 리스트들은 연산에 따라 직접(새로운 객체가 생성되지 않고) 상태가 바뀐다.

- L.sort() : 리스트 L의 원소를 오름차순(ascending order)(수 타입의 경우) 이나 사전순(lexicographical order)(문자열이나 문자의 경우)으로 정렬한다.
- L.reverse() : 리스트 L의 원소들 순서를 역순으로 만든다. 따라서 첫 번째 원소는 맨 뒤로, 두 번째 원소는 뒤에서 두 번째 등으로 바뀐다.

```
heights = [1.4, 1.3, 1.5, 2, 1.4, 1.5, 1]
heights.reverse() ---- 원래 리스트를 역순으로 만듦
print(heights) ---- 원래 리스트를 역순으로 만들었기 때문에 [1, 1.5, 1.4, 2, 1.5, 1.3, 1.4]를 출력
heights.sort() ---- 리스트를 오름차순으로 정렬
print(heights) ---- 리스트를 오름차순으로 정렬했기 때문에 [1, 1.3, 1.4, 1.4, 1.5, 1.5, 2]를 출력
heights.reverse() ---- 정렬된 리스트를 역순으로 만듦
print(heights) ---- 오름차순으로 정렬된 리스트를 역순으로 만들었기 때문에 [2, 1.5, 1.5, 1.4, 1.4, 1.3, 1]을 출력
```

코드 25-1
코드 정렬하기 및 역순으로 뒤집기

- 셀프 체크 26.1

26.2 리스트의 리스트

26.2 리스트의 리스트

» 리스트를 원소로 하는 리스트

» 다음 코드는 빈 리스트 세 개를 원소로 하는 리스트 L을 만들고, 세 원소에 새로운 원소를 넣는 과정이다.

```
L = [[], [], []] ---- 빈 리스트로 이뤄진 리스트를 만듦
L[0] = [1,2,3] ---- 0번 인덱스에 있는 리스트를 [1, 2, 3]으로 설정했으므로 L의 값은 [[1,2,3], [], []]
L[1].append('t') ---- 중간에 있는 리스트에 't'라는 문자열을 넣었으므로 L의 값은 [[1,2,3], ['t'], []]
L[1].append('o') ---- 중간에 있는 (한 번 내용이 변경된) 리스트에 'o'라는 문자열을 추가했으므로
                      L의 값은 [[1,2,3], ['t', 'o'], []]
L[1][0] = 'd' ----
중간에 있는 (값이 바뀐) 리스트의 0번 인덱스에 있는
문자열을 'd'로 바꿨으므로 L의 값은 [[1,2,3], ['d', 'o'], []]
```

코드 26-2
리스트의 리스트를 만들고 내부 리스트에 원소 채워 넣기

- 첫 번째 인덱스로 리스트의 리스트에서 특정 리스트를 선택하고, 두번째 인덱스로 객체를 선택한다(리스트의 리스트의 리스트와 같이 리스트 층이 하나 늘어날 때마다 인덱스 개수도 하나씩 더 늘어나야 한다).
- 리스트에서 어떤 인덱스 위치에 있는 원소가 리스트라면 인덱스를 적용할 수 있다.
- 그 원소가 또 리스트라면 또 다시 인덱스를 적용할 수 있고, 이를 계속 반복할 수 있다.

26.2 리스트의 리스트

» 리스트의 리스트를 사용하면 틱택토(tic-tac-toe) 게임판을 만들 수 있다. 코드 26-3은 리스트로 게임판을 초기화하는 코드다.

```
x = 'x' ---- 변수 x
o = 'o' ---- 변수 o
empty = '_' ---- 빈칸
board = [[x, empty, o], [empty, x, o], [x, empty, empty]] ---- 각 변수에 값이 정해져 있으므로, board라는 변수에는 세 줄(하위 리스트가 각 줄을 표현)과 세 열(하위 리스트 안의 세 원소가 각 열을 표현)이 있음
```

코드 26-3
틱택토 게임판을 리스트의 리스트로 표현하기

- 이 보드는 다음과 같은 상황을 표현한다.

```
x _ o
_ x o
x _ _
```

» 셀프 체크 26.2

26.3 문자열을 리스트로 변환하기

26.3 문자열을 리스트로 변환하기

» 이메일 주소 정보를 콤마로 구분한 문자열을 분리하기

```
emails = "zebra@zoo.com,red@colors.com,tom.sawyer@book.com,pea@veg.com"
```

» emails라는 이름의 문자열에 대해 split() 연산을 사용한다.

```
emails_list = emails.split(',')
```

» 문자열을 분리하면 리스트가 만들어진다.

```
['zebra@zoo.com', 'red@colors.com', 'tom.sawyer@book.com', 'pea@veg.com']
```

» 셀프 체크 26.3

26.4 리스트 응용

26.4.1 스택

» 스택의 맨 위는 리스트의 맨 뒤에 해당한다.

- 새 원소를 넣을 때는 리스트의 맨 뒤에 append()를 사용해 추가한다.
- 원소를 꺼낼 때는 리스트의 맨 뒤에서 pop()을 사용해 꺼낸다

» 스택 = FILO(First In Last Out) = LIFO(Last In First Out)

```
stack = [] ---- 빈 리스트
coocked = ['ㅎ', 'ㅎ', 'ㅎ'] ---- 부침개 세 개 부침
stack.extend(coocked) ---- 주방 이모가 부친 부침개를 스택에 (순서대로) 추가
stack.pop() ---┐
stack.pop() ---┘--- 스택(리스트)의 마지막 원소 제거
cooked = ['ㄱ', 'ㄱ'] ---- 새로 부친 부침개 한 무더기
stack.extend(coocked) ---- 스택 끝에 부침개 더미 추가
stack.pop() ---- 스택(리스트)의 마지막 원소 제거
cooked = ['ㅎ', 'ㅎ'] ---- 새로 부친 부침개 한 무더기
stack.extend(coocked) ---- 스택 끝에 부침개 더미 추가
stack.pop() ---┐
stack.pop()  ---┼--- 스택(리스트)의 마지막 원소 제거
stack.pop() ---┘
```

코드 26-4
리스트로 부침개 스택 표현하기

26.4.2 큐

큐(queue)(대기열)를 리스트로 시뮬레이션할 수 있다.

- 큐에 원소를 새로 추가할 때는 리스트의 맨끝에 원소를 넣는다. 하지만 큐에서 원소를 뺄 때는 리스트의 맨 앞에서 원소를 제거한다
- 고객이 계산대로 다가오면 append()를 사용해 리스트의 맨 뒤에 고객을 추가한다.
- 계산이 끝난 고객은 pop(0)으로 리스트의 맨 앞에서 제거한다

```
line = [] ---- 빈 리스트
line.append('오현석') ---- 이제 큐에 한 명이 들어 있음
line.append('이계영') ---- 이제 큐에 두 명이 들어 있음
line.pop(0) ---- 큐의 첫 번째 사람 제거
line.append('이일민') ---┐
line.append('김도남') ---┘--- 새로운 사람을 큐(리스트) 맨 뒤에 추가
line.pop(0) ---┐
line.pop(0)    ├--- 사람을 제거할 때는 큐(리스트) 맨 앞부터 제거
line.pop(0) ---┘
```

코드 26-5
리스트로 슈퍼마켓 계산 대기열 표현하기

셀프 체크 26.4

26.5 요약

26.5 요약

- 리스트에 다른 리스트가 원소로 들어갈 수 있다.
- 리스트의 원소를 정렬하거나 원소 순서를 거꾸로 뒤집을 수 있다.
- 스택이나 큐의 동작을 리스트를 사용해 구현할 수 있다

26.5 요약

» (Q26.1) 도시 이름이 콤마로 구분된 문자열을 받아서 사전 순으로 정렬해 출력하는 프로그램을 작성하라. 다음 도시 목록을 사용하라.

```
cities = "서울,대전,대구,부산,제주"
```

26.5 요약

» (Q26.2) is_permutation이라는 이름의 함수를 작성하라. is_permutation은 두 리스트 L1과 L2를 인자로 받는다. 함수는 L1과 L2가 서로 열(permutation)관계에 있으면 True를 반환하고, 아니면 False를 반환한다. L1과 L2의 원소는 서로 다른 순서로 나열되어 있다고 가정하라. 다음 예를 보라.

- is_permutation([1,2,3], [3,1,2]): True 반환
- is_permutation([1,1,1,2], [1,2,1,1]): True 반환
- is_permutation([1,2,3,1], [1,2,3]): False 반환