→ 스트라이드와 슬라이스를 한 식에 사용하지 말라

스트라이드를 통해 시퀀스를 슬라이싱하면서 매 n번째 원소만 가져올 수 있지만 (예: x[1:5:2]) 각 콜론에 3개의 수가 들어가면서 코드읽기가 너무 높아져 이해하기가 어려워진다. 따라서 슬라이드 (ex: x[1:5]) 와 스트라이드(x[::2]) 는 따로 쓸 것을 권한다.

→ 슬라이싱 보다 나머지를 모두 잡아내는 언패킹을 사용하라 — 별표 식(starred expression)을 사용.

- 언패킹 대입에 별표식을 사용하면 언패킹 패턴에서 대입되지 않는 모든 부분을 리스트에 잡아낼 수 있다.

```
car_ages = [0, 9, 4, 8, 7, 20, 19, 1, 6, 15]
car_ages_descending = sorted(car_ages, reverse=True)

oldest = car_ages_descending[0]
second_oldest = car_ages_descending[1]
others = car_ages_descending[2:]
print(oldest, second_oldest, others)
```

```
20 19 [15, 9, 8, 7, 6, 4, 1, 0]
```

VS

```
oldest, second_oldest, *others = car_ages_descending
print(oldest, second_oldest, others)
```

```
20 19 [15, 9, 8, 7, 6, 4, 1, 0]
```

별표식(starred expression)을 사용.
→ 위 코드와 달리 더 짧고, 더 읽기 쉽고, 여러 줄에 인덱스 계산해서 어디서 오류가 발생할 여지도 없음.

- 별표식을 다른 위치에 쓸수도 있다.
- 별표 식이 포함된 언패킹 대입을 처리하려면, 적어도 필수인 부분이 하나는 있어야 함.

  ex) *others = car_ages_descending (x)
- 한 수준의 언패킹 패턴에 별표식을 2개 이상 쓸 수 없다.

  ex) first, *middle, *second_middle, last = [1,2,3,4] (x)

- 별표 식은 항상 list 인스턴스를 가리고, 언패킹하는 시퀀스에 남는 원소가 없으면 별표식 부분은 빈 리스트가 된다.
- 별표 식은 항상 리스트를 만들어내기에 이터레이터를 별표 식으로 언패킹하면 컴퓨터 메모리를 모두 다 사용해서 프로그램이 멈출 수 있다. → so, 결과 데이터가 모두 메모리에 들어갈 수 있다고 확신할 때만 이터레이터를 모두 잡아내는 언패킹을 사용하라.

→ 복잡한 기준을 사용해 정렬할 때는 key 파라미터를 사용하라.

- sort 메서드의 key 파라미터를 이용하면, 비교에 사용할 객체를 반환하는 도우미 함수를 제공할 수 있다.
- "-" 기호를 통해 부호를 바꿀 수 있는 타입이 정렬 기준인 경우, 정렬 순서를 반대로 바꿀 수 있다.
- 부호를 바꿀 수 없는 타입의 경우 여러 정렬 기준을 조합하려면 각 정렬 기준마다 reverse 매개변수로 정렬 순서를 지정한 뒤 sort 메서드를 여러번 사용해야 한다.

  리스트 타입의 sort 메서드는 key 함수가 반환하는 값이 서로 같은 경우 리스트에 들어 있던 순서대로 반환하기 때문.

  (이때 정렬 기준의 우선순위가 점점 높아지는 순서로 sort 함수호출해야 함)

```
class Tool:
    def __init__(self, name, weight):
        self.name = name
        self.weight = weight

    def __repr__(self):
        return f'Tool({self.name!r}, {self.weight})'

power_tools = [
    Tool('드릴', 4),
    Tool('원형 톱', 5),
    Tool('착암기', 40),
    Tool('연마기', 4),
]

power_tools.sort(key=lambda x: (-x.weight, x.name))
print(power_tools)
```

```
[Tool('착암기', 40), Tool('원형 톱', 5), Tool('드릴', 4), Tool('연마기', 4)]
```

⇒ 같은 결과를 만듦.

```
power_tools.sort(key=lambda x: x.name)   # name 기준 오름차순
power_tools.sort(key=lambda x: x.weight, reverse=True) # weight 기준 내림차순
print(power_tools)
```

```
[Tool('착암기', 40), Tool('원형 톱', 5), Tool('드릴', 4), Tool('연마기', 4)]
```