Effective Python Study

Chapter 07-09

박진성

목차

목차

list comprehension (리스트 함축 표현식)

- 한 리스트에서 다른 리스트를 만들어내는 간결한 문법
- [] 문자 안에 문법을 작성

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squares = [x**2 for x in a]
```

map

- 계산식에 필요한 lambda 함수를 생성해야한다.

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squares = map(lambda x: x ** 2, a)
```

- 직관적이지 않다.
- list를 사용할 목적으로 map를 사용하면 map 인스턴스를 만드는 비용이 추가된다.

list comprehension

- 표현식에 조건을 추가할 수 있다.

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_squares = [x**2 for x in a if x % 2 == 0]
```

map and filter

- map은 filter와 연계해서 사용한다.

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_squares = map(lambda x: x*2, filter(lambda: x: x%2 == 0, a))
```

- 역시 읽기가 어렵다.

list comprehension

- Dictionary와 set에서도 사용 가능하다.

```
chile_ranks = {'ghost': 1, 'habanero': 2, 'cayenne': 3}
rank_dict = {rank: name for name, rank in chile_ranks.items()}
chile_len_set = {len(name) for name in chile_ranks}
```

결론

- list comprehension은 lambda가 필요 없다.
- map, filter 를 사용하는 것보다 명확하다.
- dictionary, set 도 지원한다.

목차

list comprehension

- 다중 루프를 지원한다.
- 표현식은 왼쪽에서 오른쪽 순서로 실행된다.

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
flat = [x for row in matrix for x in row]
```

- 레이아웃을 두 레벨로 중복해서 구성할 수 있다.

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
squared = [[x**2 for x in row] for row in matrix]
```

list comprehension의 가독성..

- 이전 표현식을 다른 루프에 넣는다면..?

```
my_lists = [
    [[1, 2, 3], [4, 5, 6]],
    # ...
]
flat = [x for sublist1 in my_lists
        for sublist2 in sublist1
        for x in sublist2]
```

- 가독성이 떨어진다.

list comprehension의 가독성..

- 그렇다면 일반 for 문은??

```
my_lists = [
    [[1, 2, 3], [4, 5, 6]],
    # ...
]
flat = []
for sublist1 in my_lists:
    for sublist2 in sublist1:
        flat.extend(sublist2)
```

- 가독성이 더 좋아졌다.

list comprehension 다중 조건문

- 다중 if 조건문을 지원한다.
- 여러 조건이 있는 경우는 암시적으로 and 표현식이 된다.

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
b = [x for x in a if x > 4 if x % 2 == 0]
c = [x for x in a if x > 4 and x % 2 == 0]
```

list comprehension 다중 조건문 가독성..

- 다중 조건문에서의 가독성을 살펴보자

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
filtered = [[x for x in row if x % 3 == 0]
            for row in matrix if sum(row) >= 10]
```

- 역시 가독성이 많이 떨어진다...

결론

- list comprehension은 다중 루프와 다중 조건을 지원한다.
- 표현식이 두 개가 넘어가는 경우는 가독성이 떨어진다.

목차

list comprehension의 메모리 문제

- 입력 시퀀스에 있는 각 아이템별로 새 리스트를 생성한다.

```
value = [len(x) for x in open('./tmp/my_file.txt')]
print(value)
```

- 메모리 할당이 불가피하다.
- 만약 입력 값이 매우 크다면 메모리 고갈 현상이 발생할 수 있다.

generator expression (제네레이터 표현식)

- () 문자 사이에 문법을 넣어 사용

```
it = (len(x) for x in open('./tmp/my_file.txt'))
```

- 실행될 때 출력 시퀀스를 메모리에 로딩하지 않는다.
- iterator를 통해 한 번에 한 아이템만 출력한다.

```
print(next(it))
print(next(it))
print(next(it))
```

generator expression

- generator 을 통해 얻은 iterator를 다른 generator에서도 사용할 수 있다.

```
it = (len(x) for x in open('./tmp/my_file.txt'))
roots = ((x, x**0.5) for x in it)
```

- for에 iterator를 사용하면 내부적으로 StopIteration 예외가 발생할 때까지 next를 계속 호출

결론

- list comprehension은 입력 값마다 메모리를 할당한다.
  - 메모리 고갈이 발생할 수 있다.

- generator expression은 한 번에 한 출력만 한다.
  - 때문에 메모리 문제를 피할 수 있다.

- generator expression을 다른 generator expression에 사용 가능하다.
- generator expression은 서로 연결되어 있을 때 매우 빠르다.

Q&A