5 리스트/딕셔너리/세트 축약

가변자료형인 리스트, 딕셔너리, 세트는 이미 존재하는 순회형 자료를 기반으로 새로운 리스트, 딕셔너리, 세트를 생성할 수 있다. 이를 각각 리스트 축약, 딕셔너리 축약, 세트 축약이라 한다. 이 책에서 사용하는 '축약'이란 단어는 원래 영어로는 'comprehension'으로 표현된다. 논리학에서 comprehension이란 특정 사물(object)이 가지고 있는 속성, 특징 또는 성질 등을 통해 그 사물이 존재하는 의도(ontology)를 '이해(comprehension)'하는 것을 뜻한다. 또한 수학의 집합 이론에서 특정 집합이 포함하는 원소들을 만족하는 성질 또는 조건을 설명하는 표기법을 'set-builder notation' 또는 'set comprehension'이라고 하는데, 여기서 유래가 되어 일부 프로그래밍 언어에서 특정 조건을 만족하는 객체를 생성할 때 사용하는 문법적 구조를 'comprehension'이라 부른다. 이는 일부 프로그래밍 언어에서만 사용 가능한 문법적 구조로, 파이썬이 이런 기능을 제공하고 있다. 이 책에서는 원래 파이썬에서 사용하는 list/dictionary/set comprehension의 기능적 역할을 잘 전달하기 위해 'comprehension'을 '축약' 또는 '축약 기법'으로 번역한다.

리스트 축약

리스트 축약(list comprehension)이란 순회형 자료의 객체들을 활용해서 리스트를 생성하는 기법 중 하나로, 아주 간결한 문법적 구조를 제공하는 장점이 있다. 파이썬에서 리스트 축약을 작성하는 일반적인 형식은 다음과 같다.

```
[표현식 for 변수 in 순회형 <if 불린-표현식>]
```

특징은 다음과 같다.

- 순환문을 사용하여 리스트의 객체를 생성하는 기법이다.
- 대괄호([ ])로 묶은 표현식과 for문을 통해 간단하게 리스트를 생성할 수 있으며, 필요에 따라 조건문을 넣을 수도 있다.
- 표현식은 변수 혹은 변수의 객체를 생성하는 식이 온다.
- 선택 사항[3]인 조건문을 통해 조건을 충족하는 객체만 선별하는 것이 가능하다.

리스트 축약은 크게 두 종류로 구분해서 살펴볼 수 있는데, 조건문이 없는 리스트 축약 형식과 조건문이 있는 리스트 축약 형식으로 구분할 수 있다. 먼저 조건문이 없는 리스트 축약 형식과 for문을 이용한 일반적인 리스트 생성 방식과 비교해보자.

다음 예는 for문을 사용해서 1부터 9까지의 정수를 담고 있는 리스트를 생성하는 코드이다.

```
>>> int_list = []
>>> for i in range(1, 10):
...     int_list.append(i)
... else:
...     print(int_list)
...
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

이번에는 리스트 축약 기법을 사용해서 작성해보자. 다음 예는 위 코드와 같은 결과를 가져오지만 코드가 훨씬 간결하다는 것을 알 수 있다.

```
>>> [i for i in range(1, 10)]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

이번에는 for문을 사용해서 1부터 9까지의 정수를 문자열로 변환한 후 리스트에 추가하는 코드를 작성해보자.

3 이 책에서 선택 사항을 표시하는 부호로 원래 대괄호([])를 사용하는데, 여기서는 리스트 축약에 사용하는 대괄호와 구분하기 위해 꺽쇠 괄호(〈〉)를 사용한다.

```
>>> str_list = []
>>> for i in range(1, 10):
...     str_list.append(str(i))          # 정수를 문자열로 변환해서 리스트에 추가한다.
... else:
...     print(str_list)
...
['1', '2', '3', '4', '5', '6', '7', '8', '9']
```

정수가 아닌 문자열 '1'부터 '9'까지를 객체로 가지는 리스트를 만들었다. 같은 결과가 나오는 코드를 리스트 축약으로 작성해서 실행해보자.

```
>>> [str(i) for i in range(1, 10)]
['1', '2', '3', '4', '5', '6', '7', '8', '9']
```

마찬가지로 훨씬 간결한 코드로 같은 결과를 가져왔다. 위의 두 예에서 일반 for문으로 리스트를 생성하면 다음과 같은 형식을 가진다는 것을 알 수 있다.

```
tmp = []
for 변수 in 순회형:
    tmp.append(표현식)
```

여기서 표현식은 변수이거나 변수의 객체를 생성하는 식이다. 위의 예에서는 정수형을 문자열로 변환하는 str(i)이 표현식이다.

리스트 축약으로 같은 내용을 표현하면 다음과 같다.

```
[표현식 for 변수 in 순회형]
```

이처럼 리스트 축약을 사용하면 일반 for문을 사용한 것과 비교했을 때 문법적으로 매우 간결할 뿐만 아니라 tmp 변수를 생성하지 않아도 된다.

이번에는 조건문을 사용해서 1부터 9까지의 정수 중 홀수만 문자열로 변환해서 리스트를 생성해보자. 다음 예는 일반 for문을 사용해서 실행한 결과를 보여준다.

```
>>> str_odd_list = []
>>> for i in range(1, 10):
...     if i % 2 == 1:
...         str_odd_list.append(str(i))
... else:
...     print(str_odd_list)
...
['1', '3', '5', '7', '9']
```

다음 예는 조건문이 있는 위 코드를 리스트 축약 형식으로 작성해서 실행한 결과를 보여준다.

```
>>> [str(i) for i in range(1, 10) if i % 2 == 1]
['1', '3', '5', '7', '9']
```

이처럼 리스트 축약을 사용해서 리스트를 생성할 때 코드가 훨씬 간결하다. 다음 코드는 리스트 축약 기법으로 1부터 9 사이 홀수의 제곱을 담고 있는 리스트를 만들어 실행한 결과를 보여준다.

```
>>> [pow(i, 2) for i in range(10) if i % 2 == 1]
[1, 9, 25, 49, 81]
```

조건문을 사용해서 리스트를 생성할 때 일반 for문을 사용하면 다음과 같은 형식을 가진다.

```
tmp = []
for 변수 in 순회형:
    if 불린-표현식:
        tmp.append(표현식)
```

이처럼 조건문이 주어질 경우에는 순회형의 객체 중 if문에 있는 불린-표현식 값이 '참(True)'에 해당하는 객체만 리스트에 추가한다. 여기서 표현식은 변수이거나 변수의 객체를 생성하는 식이다. 앞의 예에서는 정수 중 홀수만을 리스트에 추가하는 식이다.

리스트 축약으로 같은 내용을 표현하면 다음과 같다.

```
[표현식 for 변수 in 순회형 if 불린-표현식]
```

실습 A5

윤년 계산

- 리스트 축약 기법을 사용해서 윤년을 계산해보자.
- 윤년 계산 규칙[4]은 다음과 같다.
  - 그 해의 숫자가 4로 나누어 떨어지는 해는 윤년이라 한다.
    예) 1988년, 1992년, 1996년, 2004년, 2008년, 2012년, …
  - 그 해의 숫자가 4로 나누어지지만 100으로도 나누어지면 평년이라 한다.
    예) 1900년, 2100년, 2200년, 2300년, 2500년, …
  - 그 해의 숫자가 100으로 나누어지지만 400으로도 나누어지면 윤년이라 한다.
    예) 1600년, 2000년, 2400년, …
- 2000년에서 2500년 사이(둘다 포함)의 윤년을 리스트로 생성해보자.

답

```
>>> [year for year in range(2000, 2501)
...          if year % 4 == 0 and year % 100 != 0 or year % 400 == 0]
[2000, 2004, 2008, 2012, 2016, 2020, 2024, 2028, 2032, 2036, 2040, 2044,
2048, 2052, 2056, 2060, 2064, 2068, 2072, 2076, 2080, 2084, 2088, 2092,
2096, 2104, 2108, 2112, 2116, 2120, 2124, 2128, 2132, 2136, 2140, 2144,
2148, 2152, 2156, 2160, 2164, 2168, 2172, 2176, 2180, 2184, 2188, 2192,
2196, 2204, 2208, 2212, 2216, 2220, 2224, 2228, 2232, 2236, 2240, 2244,
2248, 2252, 2256, 2260, 2264, 2268, 2272, 2276, 2280, 2284, 2288, 2292,
2296, 2304, 2308, 2312, 2316, 2320, 2324, 2328, 2332, 2336, 2340, 2344,
2348, 2352, 2356, 2360, 2364, 2368, 2372, 2376, 2380, 2384, 2388, 2392,
2396, 2400, 2404, 2408, 2412, 2416, 2420, 2424, 2428, 2432, 2436, 2440,
2444, 2448, 2452, 2456, 2460, 2464, 2468, 2472, 2476, 2480, 2484, 2488,
2492, 2496]
```

4 https://ko.wikipedia.org/wiki/윤년

축약 기법도 중첩으로 사용할 수 있다. 중첩 리스트 축약을 작성하는 일반적인 형식은 다음과 같다.

```
[표현식 for 변수-1 in 순회형 <if 불린-표현식-1>
          for 변수-2 in 순회형 <if 불린-표현식-2>
              for 변수-3 in 순회형 <if 불린-표현식-3>
                  ...
                      for 변수-N in 순회형 <if 불린-표현식-N>]
```

중첩되는 축약으로는 리스트 축약 외에도 곧이어 다루게 될 딕셔너리 축약이나 세트 축약 등 다른 축약형이 올 수 있다.

몇 가지 예를 통해 중첩 리스트 축약을 작성하는 방법을 알아보자. 다음 예는 표현식에 문자열 결합을 사용해서 성적 조합을 리스트로 생성한 결과를 보여준다.

```
>>> [letter + sign for letter in 'ABCD' for sign in '+0-']
['A+', 'A0', 'A-', 'B+', 'B0', 'B-', 'C+', 'C0', 'C-', 'D+', 'D0', 'D-']
```

다음 코드는 2단부터 9단까지의 구구단 계산식을 문자열로 담고 있는 리스트를 생성한 결과를 보여준다.

```
>>> [f'{x} x {y} = {x * y}' for x in range(2, 10)
...                             for y in range(1, 10)]
['2 x 1 = 2', '2 x 2 = 4', '2 x 3 = 6', '2 x 4 = 8', '2 x 5 = 10', '2 x 6 = 12', '2
x 7 = 14', '2 x 8 = 16', '2 x 9 = 18', '3 x 1 = 3', '3 x 2 = 6', '3 x 3 = 9', '3 x
4 = 12', '3 x 5 = 15', '3 x 6 = 18', '3 x 7 = 21', '3 x 8 = 24', '3 x 9 = 27', '4 x
1 = 4', '4 x 2 = 8', '4 x 3 = 12', '4 x 4 = 16', '4 x 5 = 20', '4 x 6 = 24', '4 x 7
= 28', '4 x 8 = 32', '4 x 9 = 36', '5 x 1 = 5', '5 x 2 = 10', '5 x 3 = 15', '5 x 4
= 20', '5 x 5 = 25', '5 x 6 = 30', '5 x 7 = 35', '5 x 8 = 40', '5 x 9 = 45', '6 x 1
= 6', '6 x 2 = 12', '6 x 3 = 18', '6 x 4 = 24', '6 x 5 = 30', '6 x 6 = 36', '6 x 7
= 42', '6 x 8 = 48', '6 x 9 = 54', '7 x 1 = 7', '7 x 2 = 14', '7 x 3 = 21', '7 x 4
= 28', '7 x 5 = 35', '7 x 6 = 42', '7 x 7 = 49', '7 x 8 = 56', '7 x 9 = 63', '8 x 1
= 8', '8 x 2 = 16', '8 x 3 = 24', '8 x 4 = 32', '8 x 5 = 40', '8 x 6 = 48', '8 x 7
= 56', '8 x 8 = 64', '8 x 9 = 72', '9 x 1 = 9', '9 x 2 = 18', '9 x 3 = 27', '9 x 4
= 36', '9 x 5 = 45', '9 x 6 = 54', '9 x 7 = 63', '9 x 8 = 72', '9 x 9 = 81']
```

중첩 리스트 축약을 사용할 경우 나중에 오는 for문은 앞에서 온 for문의 결과를 사용할 수 있다. 다음 예의 첫 번째 for문의 range()는 1부터 정수를 차례대로 생성한다. 두 번째 for문의 range()에서는 첫 번째 range()의 시작 값보다 1이 큰 수부터 정수를 차례대로 생성한다. 세 번째 for문의 range()에서는 두 번째 range()의 시작 값보다 1이 큰 수부터 시작해서 폭을 두 번째 for문의 range()의 시작 값만큼 건너 뛰면서 정수를 생성한다. 이때, 세 번째 for문의 range()에 의해 생성되는 값이 가장 큰 값을 갖게 되며 9까지 값을 갖게 된다. 즉, 다음과 같은 리스트를 생성하게 된다.

```
>>> [(x, y, z) for x in range(1, 10)
...       for y in range(x + 1, 10)
...           for z in range(y + 1, 10, x + 1)]
[(1, 2, 3), (1, 2, 5), (1, 2, 7), (1, 2, 9), (1, 3, 4), (1, 3, 6), (1, 3, 8),
(1, 4, 5), (1, 4, 7), (1, 4, 9), (1, 5, 6), (1, 5, 8), (1, 6, 7), (1, 6, 9),
(1, 7, 8), (1, 8, 9), (2, 3, 4), (2, 3, 7), (2, 4, 5), (2, 4, 8), (2, 5, 6),
(2, 5, 9), (2, 6, 7), (2, 7, 8), (2, 8, 9), (3, 4, 5), (3, 4, 9), (3, 5, 6),
(3, 6, 7), (3, 7, 8), (3, 8, 9), (4, 5, 6), (4, 6, 7), (4, 7, 8), (4, 8, 9),
(5, 6, 7), (5, 7, 8), (5, 8, 9), (6, 7, 8), (6, 8, 9), (7, 8, 9)]
```

딕셔너리 축약

딕셔너리 축약(dictionary comprehension)도 리스트 축약처럼 간결한 구문으로 딕셔너리를 생성하는 방법이다. 리스트 축약 형식과 비슷하지만 대괄호 대신 중괄호를 사용하며 키와 매핑값을 처리하는 점이 다르다. 딕셔너리 축약을 작성하는 일반적인 형식은 다음과 같다.

{키-표현식:매핑값-표현식 for 변수 in 순회형 [if 불린-표현식]}

특징은 다음과 같다.

- 순환문을 사용하여 딕셔너리의 객체를 생성하는 기법이다.
- 중괄호({})로 묶은 표현식과 for문을 통해 간단하게 딕셔너리를 생성할 수 있으며, 필요에 따라 조건문을 넣을 수도 있다.
- 키-표현식과 매핑값-표현식은 변수 혹은 변수의 객체를 생성하는 식이 온다.
- 변수는 하나의 변수일 수도 있고 복합 변수일 수도 있다. 복합 변수인 경우 보통 (키, 매

핑값) 형식의 튜플을 사용한다.

- 선택 사항인 조건문을 통해 조건을 충족하는 객체만 선별하는 것이 가능하다.

다음 예는 정수 -9부터 9까지를 키로, -9부터 9까지 정수의 제곱을 매핑값으로 하는 딕셔너리를 생성한 결과를 보여준다.

```
>>> {i: pow(i, 2) for i in range(-9, 10)}
{-9: 81, -8: 64, -7: 49, -6: 36, -5: 25, -4: 16, -3: 9, -2: 4, -1: 1, 0: 0,
1: 1,2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

다음 예는 간단한 한영사전을 만들어서 변수 koreng에 할당한 후 딕셔너리 축약 기법을 사용해서 영한사전으로 바꾸고, 영어 알파벳 내림차순 순으로 출력한 결과를 보여준다.

```
>>> koreng = dict(사과='apple', 블루베리='blueberry', 딸기='strawberry',
...               키위='kiwi', 바나나='banana', 포도='grape', 자두='plum')
>>> koreng
{'사과': 'apple', '블루베리': 'blueberry', '딸기': 'strawberry', '키위':
'kiwi', '바나나': 'banana', '포도': 'grape', '자두': 'plum'}
>>> engkor = {v: k for k, v in koreng.items()}                    # 딕셔너리 축약
>>> sorted(engkor.items())
[('apple', '사과'), ('banana', '바나나'), ('blueberry', '블루베리'), ('grape',
'포도'), ('kiwi', '키위'), ('plum', '자두'), ('strawberry', '딸기')]
```

리스트 축약처럼 딕셔너리 축약도 중첩으로 사용할 수 있다. 중첩 딕셔너리 축약을 작성하는 일반적인 형식은 다음과 같다.

```
{키-표현식:매핑값-표현식
    for 변수-1 in 순회형 [if 불린-표현식-1]
        for 변수-2 in 순회형 [if 불린-표현식-2]
            for 변수-3 in 순회형 [if 불린-표현식-3]
                ...
                    for 변수-N in 순회형 [if 불린-표현식-N]}
```

중첩되는 축약으로는 딕셔너리 축약 외에도 리스트 축약이나 세트 축약 등 다른 축약형이 올 수 있다.

세트 축약

세트 축약(dictionary comprehension)이란 앞서 설명한 리스트 축약이나 딕셔너리 축약처럼 간결한 구문으로 세트를 생성하는 방법이다. 딕셔너리 축약 형식과 비슷하지만 표현식이 키와 매핑값 쌍으로 되어 있지 않다는 점이 다르다. 세트 축약을 작성하는 일반적인 형식은 다음과 같다.

```
{표현식 for 변수 in 순회형 [if 불린-표현식]}
```

특징은 다음과 같다.

- 순환문을 사용하여 세트의 객체를 생성하는 기법이다.
- 중괄호({ })로 묶은 표현식과 for문을 통해 간단하게 세트를 생성할 수 있으며, 필요에 따라 조건문을 넣을 수도 있다.
- 표현식은 변수 혹은 변수의 객체를 생성하는 식이 온다.
- 선택 사항인 조건문을 통해 조건을 충족하는 객체만 선별하는 것이 가능하다.

몇 가지 예를 통해 세트 축약을 사용하는 방법을 알아보자. 다음 예는 세트 축약 방식으로 -3에서 3까지 정수의 제곱을 세트로 생성한 결과이다.

```
>>> {i * i for i in (-3, -2, -1, 0, 1, 2, 3)}                    # 세트 축약
{0, 9, 4, 1}
```

세트 자료형이기 때문에 출력한 결과가 순서가 없을 뿐만 아니라 중복된 값은 포함하고 있지 않다. 만약 리스트 축약을 사용했다면 다음과 같은 결과가 나올 것이다.

```
>>> [i * i for i in (-3, -2, -1, 0, 1, 2, 3)]                    # 리스트 축약
[9, 4, 1, 0, 1, 4, 9]
```

다음 코드는 세트 축약 방식으로 range() 클래스를 사용해서 -9에서 9까지 정수를 생성하고 각 정수의 절댓값을 세트로 생성한 결과를 보여준다.

```
>>> {abs(i) for i in range(-9, 10)}                    # 세트 축약
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

만약 리스트 축약 형식을 사용해서 리스트를 생성했다면 다음과 같은 결과가 나올 것이다.

```
>>> [abs(i) for i in range(-9, 10)]                    # 리스트 축약
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

마지막으로 다음 코드는 현재 폴더에 있는 파일 중 확장명이 '.py' 또는 '.pyw'인 파일만 출력하는 코드를 세트 축약을 사용해서 실행한 결과를 보여준다.

```
>>> import os
>>> {file for file in os.listdir() if file.lower().endswith(('.py', '.pyw'))}
{'hello.py', 'my_first_python.py', 'my_first_gui_python.pyw'}
```

세트 축약도 다른 축약형처럼 중첩이 가능하다. 마찬가지로 중첩되는 축약형으로는 세트 외에도 리스트나 딕셔너리 축약이 올 수 있다. 중첩 세트 축약을 작성하는 일반적인 형식은 다음과 같다.

```
{표현식 for 변수-1 in 순회형 [if 불린-표현식-1]
          for 변수-2 in 순회형 [if 불린-표현식-2]
              for 변수-3 in 순회형 [if 불린-표현식-3]
                  ...
                      for 변수-N in 순회형 [if 불린-표현식-N]}
```