8가지 쿨한 파이썬 기능

리스트 컴프리헨션(List Comprehensions)

파이썬에서 제가 가장 좋아하는 기능은 단연 리스트 컴프리헨션입니다. 솔직히 이 기능이 특별히 흥미로운 건 아니고요. 그저 리스트를 편리하게 생성하는 방법일 뿐입니다.

```
# 0부터 9까지의 값을 포함하는 리스트를 생성합니다.
[i for i in range(10)]

# 0부터 9까지의 모든 짝수 값을 포함하는 리스트를 생성합니다.
[i for i range(10) if i % 2 == 0]

# 1부터 10까지의 값을 포함하는 리스트를 생성합니다.
[i + 1 for i in range(10)]

# 0부터 -9까지의 값을 포함하는 리스트를 생성합니다.
[-i for i in range(10)]

# 0부터 9까지 가능한 모든 쌍을 생성합니다.
[(a, b) for a in range(10) for b in range(10)]

# 다른 리스트를 얕게 복사합니다.
my_list = [1, 3, 5, 7, 9]
[item for item in my_list]
```

제너레이터 표현식(Generator Expressions)

리스트 컴프리헨션 구문을 익히면 좋은 점 중 하나는 제너레이터 표현식도 작성할 수 있다는 것입니다. 결국 둘은 매우 유사하며, 제너레이터 표현식은 공간을 절약해 줍니다. 맞습니다! 제너레이터 표현식은 실제로 리스트를 생성하지 않습니다. 대신, 리스트를 실제로 구성하지 않고도 한 번에 하나의 항목을 생성할 수 있는 수단을 제공합니다. 다음을 살펴보세요.

```
# 0부터 9까지의 값을 생성합니다.
(i for i in range(10))

# 0부터 9까지의 값을 생성합니다.
(i for i range(10) if i % 2 == 0)

# 1부터 10까지의 값을 생성합니다.
(i + 1 for i in range(10))

# 0부터 -9까지의 값을 생성합니다.
(-i for i in range(10))

# 0부터 9까지 가능한 모든 쌍을 생성합니다.
((a, b) for a in range(10) for b in range(10))

# 다른 리스트의 얕은 복사본을 생성합니다.
my_list = [1, 3, 5, 7, 9]
(item for item in my_list)
```

슬라이스 할당(Slice Assignment)

리스트의 전체 섹션을 한 번에 바꾸고 싶었던 적이 있나요? 파이썬에는 한 줄로 그렇게 할 수 있는 기능이 있습니다. 바로 슬라이스 할당입니다. 슬라이싱과 마찬가지로 슬라이스 할당을 사용하면 리스트의 특정 영역을 지정할 수 있습니다. 물론 차이점은 슬라이스 할당을 사용하면 해당 영역을 원하는 것으로 교체할 수 있다는 것입니다.

```
my_list = [1, 2, 3]

# 슬라이스 할당으로 리스트에 추가합니다.
my_list[len(my_list):] = [4]

# 슬라이스 할당으로 리스트 앞에 추가합니다.
my_list[:0] = [0]

# 중간 요소를 대체합니다.
midpoint = len(my_list) // 2
my_list[midpoint: midpoint + 1] = [-2]

# 임의의 하위 섹션을 대체합니다.
my_list[:2] = [3, 4, 5]
```

이터러블 언패킹 (Iterable Unpacking, 일명 구조 분해)

리스트의 마지막 항목을 가져오는 것에 대한 제 글을 읽어보셨다면, 이터러블 언패킹이 해결책 중 하나였다는 것을 기억하실 겁니다. 리스트를 마지막 항목과 나머지 모든 것으로 두 조각으로 나눌 수 있다는 아이디어입니다.

```
my_list = [1, 2, 3]
*remainder, last_item = my_list
```

음수 인덱싱(Negative Indexing)

이 목록에 있는 모든 기능 중에서 음수 인덱싱은 아마도 가장 미묘할 것입니다. 결국 많은 최신 프로그래밍 언어에는 어떤 형태의 리스트 인덱싱이 있습니다. 그러나 리스트의 마지막 요소를 그렇게 우아하게 가져오는 방법이 있는 언어는 거의 없습니다.

```
my_list = [1, 2, 3]
last_item = my_list[-1]
```

딕셔너리 컴프리헨션(Dictionary Comprehensions)

이전 목록에서 리스트 컴프리헨션에 대해 언급했습니다. 분명히 이 기능은 너무 좋아서 개발자들이 딕셔너리와 같은 다른 데이터 구조까지 포함하도록 기능을 확장하기로 결정했습니다. 결국 한 줄의 코드로 딕셔너리를 생성할 수 있다면 좋지 않을까요? PEP 274부터 가능합니다.

```
# 숫자-문자 딕셔너리를 생성합니다.
{num: chr(65 + num) for num in range(5)}

# 같은 것을 생성합니다.
nums = [1, 2, 3, 4, 5]
letters = ["A", "B", "C", "D", "E"]
{num: letter for num, letter in zip(nums, letters)}
```

비교 연결(Chaining Comparisons)

많은 최신 프로그래밍 언어에서 값 비교는 간단한 과정입니다. 예를 들어, 자바에서는 두 숫자를 다음과 같이 비교할 수 있습니다.

```
17 > 5
```

f-문자열(f-Strings)

마지막으로, 제가 가장 좋아하는 "새로운" (PEP 498) 파이썬 기능 중 하나인 f-문자열에 대해 알아보겠습니다. 일반적으로 디버깅을 위해 문자열을 만들 때, 우리는 문자열 연결을 통해 게으르게 출력합니다. 만약 우리가 영리하다면, 몇 가지 문자열 형식 지정 기술을 사용할 수도 있습니다. 이제 f-문자열을 통해 두 가지 장점을 모두 얻을 수 있습니다.

```
age = 25
name = 'Jeremy'
print(f'My name is {name}, and I am {age}')
```

특별한 언급(Honorable Mentions)

파이썬으로 작업하는 것을 정말 좋아하는 사람으로서, 이 목록을 짧게 유지하는 것이 힘들었습니다. 결과적으로, 포함되지 않은 몇 가지 추가 기능은 다음과 같습니다.

- For/Else 루프
- 허수(Imaginary numbers)
- `Any()` 및 `All()`
- 여러 값 반환 (튜플)
- 임의로 큰 정수
- 키워드 인수(Keyword arguments)
- 세트(Sets)
- 문자열 연결(Joining strings)
- 문자열 곱하기(Multiplying strings)
- Walrus operator (대입 표현식)
- 문자열 보간(String interpolation)
- 슬라이싱(Slicing)