## for 반복문 기초

**for** 문을 배우기 전에 **while** 반복문을 써서 리스트의 모든 원소를 출력해보겠습니다.

```
# 빅뱅 멤버들
big_bang = ["지드래곤", "태양", "탑", "대성", "승리"]

i = 0
while i < len(big_bang):
    print(big_bang[i])
    i = i + 1
```

```
지드래곤
태양
탑
대성
승리
```

위 프로그램에서는 몇 가지 불필요한 점들이 있습니다. 첫째로, **i** 라는 변수는 인덱싱을 위한 용도일 뿐, 그 이외에는 아무런 쓸모가 없습니다. 둘째로, **len(big_bang)** 도 **i** 와의 비교 이외에는 별도의 쓰임이 없습니다.

**for** 문을 쓰면 이런 불필요한 점들 없이, 깔끔하고 직관적이게 코드를 짤 수 있습니다.

```
# 빅뱅 멤버들
big_bang = ["지드래곤", "태양", "탑", "대성", "승리"]

for member in big_bang:
    print(member)
```

```
지드래곤
태양
탑
대성
승리
```

위의 코드에서 **member** 는 **for** 문의 수행부분에서만 쓰이고 사라지는 **local** 변수입니다. 수행부분으로 처음 들어갈 때는 **member** 가 **big_bang** 리스트의 **0** 번 인덱스 요소 **"지드래곤"** 을 갖게 됩니다. 그 다음 들어갈 때는 **member** 가 **1** 번 인덱스의 요소 **"태양"**, 그 다음은 **2** 번 인덱스 요소 **"탑"**, **3** 번 인덱스 요소 **"대성"**, 그리고 마지막으로 **4** 번 인덱스 요소 **"승리"** 를 갖게 됩니다. 결과적으로 **"지드래곤"**, **"태양"**, **"탑"**, **"대성"**, **"승리"** 가 출력됩니다.

여기서 쓴 **member** 는 제가 임의로 정한 이름으로, **x** 나 **i** 등 어떤 (허용된) 이름을 붙이더라도 문제 없이 프로그램이 실행됩니다. 하지만 늘 강조하듯이 **member** 같은 의미 있는 이름을 주는 것이 좋습니다.

지금까지 **while** 문과 **for** 문을 사용하여 빅뱅 맴버들을 출력해보았습니다. 이처럼 **while** 반복문으로 만들 수 있는 프로그램은 **for** 반복문으로도 만들 수 있고, 반대로 **for** 반복문으로 만들 수 있는 프로그램은 모두 **while** 문으로도 만들 수 있습니다. 따라서 이 파트를 배운다고 해서 새로운 문제 해결 능력이 길러지는 것은 아닙니다. 하지만 **for** 문을 잘 활용하면, 코드를 훨씬 읽고 쓰기 쉽게 만들 수 있습니다. 이와 같이 프로그래밍 언어에서 동일한 기능을 깔끔하게 만들어 놓은 것을 'syntactic sugar'(꿀)라고 부릅니다.

또 하나의 예시를 볼까요? 다음은 **[1, 3, 5, 7, 9]** 의 각 원소의 제곱을 출력해주는 프로그램입니다.

```
for num in [1, 3, 5, 7, 9]:
    print(num * num)
```

```
1
9
25
49
81
```

이 경우 **while** 문 보다 **for** 문이 훨씬 더 짧고 깔끔하고 직관적이죠? 이처럼 **for** 문은 리스트를 다루는 데에 최적화되어 있습니다.

위 두 개의 코드를 일반화하면, **for** 문의 기본 구조는 다음과 같습니다:

```
for 변수 in 리스트/range/문자열:
    <첫번째 실행할 줄>
    <두번째 실행할 줄>
    ...
```

## range 함수

**for** 문을 사용하여 **1** 부터 **10** 까지 출력하려면, 어떻게 해야할까요?

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

```
1
2
3
4
5
6
7
8
9
10
```

그런데 **1** 부터 **100** 까지 출력하려면, 어떻게 해야할까요? 이를 위해 **1** 부터 **100** 까지 적혀있는 리스트를 만드는 것은 바보같은 일이겠죠?

**range** 함수를 쓰면 이 문제를 간단하게 해결할 수 있습니다.

## 파라미터가 2개 있는 range 함수

**range(n, m)** 은 **n** 부터 **m - 1** 까지의 수들을 의미합니다.

```
for i in range(n, m):
    print(i)
```

**for** 문에서 **range(1, 11)** 이라는 코드를 쓰면, **1** 부터 **10** 까지의 수이기 때문에, **[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]** 이라는 리스트를 쓰는 것과 동일한 효과를 얻게 됩니다.

```
for i in range(1, 11):
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

## 파라미터가 1개 있는 range 함수

**range(m)** 은 **0** 부터 **m - 1** 까지의 수들을 의미합니다.

```
for i in range(m):
    print(i)
```

**for** 문에서 **range(10)** 이라는 코드를 쓰면, **0** 부터 **9** 까지의 수이기 때문에, **[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]** 라는 리스트를 쓰는 것과 동일한 효과를 얻게 됩니다.

```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

### 파라미터가 3개 있는 range 함수

**range(n, m, s)** 은 **n** 부터 **m - 1** 까지의 수 중 간격이 **s** 인 수들을 의미합니다.

```
for i in range(n, m, s):
    print(i)
```

**for** 문에서 **range(3, 17, 3)** 이라는 코드를 쓰면, **3** 부터 **16** 까지의 수 중 간격이 **3** 인 수이기 때문에, **[3, 6, 9, 12, 15]** 를 쓰는 것과 동일한 효과를 얻게 됩니다.

```
for i in range(3, 17, 3):
    print(i)
```

```
3
6
9
12
15
```

**for** 문에서 **range(3, 16, 3)** 이라는 코드를 쓰면, **3** 부터 **15** 까지의 수 중 간격이 **3** 인 수이기 때문에, **[3, 6, 9, 12, 15]** 라는 리스트를 쓰는 것과 동일한 효과를 얻게 됩니다.

```
for i in range(3, 16, 3):
    print(i)
```

```
3
6
9
12
15
```

**for** 문에서 **range(3, 15, 3)** 이라는 코드를 쓰면, **3** 부터 **14** 까지의 수 중 간격이 **3** 인 수이기 때문에, **[3, 6, 9, 12]** 라는 리스트를 쓰는 것과 동일한 효과를 얻게 됩니다. 이전에 리스트에 포함되어 있었던 **15** 는 **3** 부터 **14** 안의 범위에 들어있지 않기 때문에 빠져 있습니다.

```
for i in range(3, 15, 3):
    print(i)
```

```
3
6
9
12
```

**range** 함수의 장점은 다음과 같습니다:

1. 간편하고 깔끔합니다. 굳이 리스트를 만들지 않아도, 동일한 효과를 낼 수 있습니다.
2. 메모리가 효율적입니다. **1** 부터 **100** 까지의 리스트를 쓰면 파이썬에서 그만큼의 공간을 마련해야 하는 반면, **range** 함수를 쓰면 **1** 의 값을 쓰고 버리고 **2** 의 값을 쓰고 버리고 하는 식으로 메모리 공간을 아낄 수 있습니다.

☑ 수업을 완료하셨으면 체크해주세요.

수강생 Q&A 보기 (/questions?assignment_id=107&sort_by=popular)

질문하기 (/questions/new?assignment_id=107&op1=%ED%94%84%EB%A1%9C%EA%B7%)

이전 강의 range 함수 (/assignments/106)

다음 강의 range 연습 (/assignments/108)