

# 딕셔너리와 반복문

# 목차

- 시작하기 전에
- 범위
- for 반복문: 범위와 함께 사용하기
- for 반복문 : 리스트와 범위 조합하기
- for 반복문: 반대로 반복하기
- while 반복문
- while 반복문 : for 반복문처럼 사용하기
- while 반복문: 상태를 기반으로 반복하기
- while 반복문 : 시간을 기반으로 반복하기
- while 반복문: break 키워드/continue 키워드
- 키워드로 정리하는 핵심 포인트
- 확인문제

**[핵심 키워드]** : 범위, while 반복문, break 키워드, continue 키워드

**[핵심 포인트]**

특정 횟수 / 특정 시간만큼, 그리고 어떤 조건이 될 때까지 반복하는 등의 경우에 대해 알아본다.

- 범위 (range)
  - 특정 횟수만큼 반복해서 돌리고 싶을 때 for 반복문과 조합하여 사용

- 매개변수에 숫자를 한 개 넣는 방법
  - 0부터 A-1까지의 정수로 범위 만듦

`range(A)` → A는 숫자

- 매개변수에 숫자를 두 개 넣는 방법
  - A부터 B-1까지의 정수로 범위 만듦

`range(A, B)` → A와 B는 숫자

- 매개변수에 숫자를 세 개 넣는 방법
  - A부터 B-1까지의 정수로 범위 만듦 앞뒤의 숫자가 C만큼의 차이 가짐

`range(A, B, C)` → A, B, C는 숫자

- 예시
  - 매개변수에 숫자 한 개 넣은 범위

```
>>> a = range(5)
```

```
>>> a  
range(0, 5)
```

```
>>> list(range(10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- 매개변수에 숫자 두 개 넣은 범위

```
>>> list(range(0, 5)) → 0부터 (5-1)까지의 정수로 범위를 만듭니다.
```

```
[0, 1, 2, 3, 4]
```

```
>>> list(range(5, 10)) → 5부터 (10-1)까지의 정수로 범위를 만듭니다.
```

```
[5, 6, 7, 8, 9]
```

- 매개변수에 숫자 세 개 넣은 범위

```
>>> list(range(0, 10, 2)) → 0부터 2씩 증가하면서 (10-1)까지의 정수로 범위를 만듭니다.
```

```
[0, 2, 4, 6, 8]
```

```
>>> list(range(0, 10, 3)) → 0부터 3씩 증가하면서 (10-1)까지의 정수로 범위를 만듭니다.
```

```
[0, 3, 6, 9]
```

- 범위 만들 때 매개변수 내부에 수식 사용하는 경우
  - 코드 특정 부분의 강조

```
>>> a = range(0, 10 + 1)
>>> list(a)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

- 예시 - 나누기 연산자 사용

```
>>> n = 10
>>> a = range(0, n / 2) → 매개변수로 나눗셈을 사용한 경우 오류가 발생합니다.
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
TypeError: 'float' object cannot be interpreted as an integer
```

- TypeError 발생



- 정수 나누기 연산자

```
>>> a = range(0, int(n / 2)) → 실수를 정수로 바꾸는 방법보다
```

```
>>> list(a)
```

```
[0, 1, 2, 3, 4]
```

```
>>> a = range(0, n // 2) → 정수 나누기 연산자를 많이 사용합니다!
```

```
>>> list(a)
```

```
[0, 1, 2, 3, 4]
```

# for 반복문: 범위와 함께 사용하기

- for 반복문과 범위의 조합

for 숫자 변수 in 범위:

코드

```
01  # for 반복문과 범위를 함께 조합해서 사용합니다.
02  for i in range(5):
03      print(str(i) + "= 반복 변수")
04  print()
05
06  for i in range(5, 10):
07      print(str(i) + "= 반복 변수")
08  print()
09
10  for i in range(0, 10, 3):
11      print(str(i) + "= 반복 변수")
12  print()
```

실행결과

```
0 = 반복 변수
1 = 반복 변수
2 = 반복 변수
3 = 반복 변수
4 = 반복 변수

5 = 반복 변수
6 = 반복 변수
7 = 반복 변수
8 = 반복 변수
9 = 반복 변수

0 = 반복 변수
3 = 반복 변수
6 = 반복 변수
9 = 반복 변수
```

# for 반복문 : 리스트와 범위 조합하기

- 몇 번 반복인지를 알아야 하는 경우

```
# 리스트를 선언합니다.
array = [273, 32, 103, 57, 52]
# 리스트에 반복문을 적용합니다.
for element in array:
    # 출력합니다.
    print(element)
```

현재 무엇을 출력하고 있는지 보다, 몇 번째 출력인지를 알아야 하는 경우가 있습니다.

```
01 # 리스트를 선언합니다.
02 array = [273, 32, 103, 57, 52]
03
04 # 리스트에 반복문을 적용합니다.
05 for i in range(len(array)):
06     # 출력합니다.
07     print("{}번째 반복: {}".format(i, array[i]))
```

**실행결과**

0번째 반복:	273
1번째 반복:	32
2번째 반복:	103
3번째 반복:	57
4번째 반복:	52

- 역반복문

- 큰 숫자에서 작은 숫자로 반복문 적용
- `range()` 함수의 매개변수 세 개 사용하는 방법

```
01  # 역반복문
02  for i in range(4, 0 - 1, -1):
03      # 출력합니다.
04      print("현재 반복 변수: {}".format(i))
```

실행결과

현재 반복 변수: 4  
현재 반복 변수: 3  
현재 반복 변수: 2  
현재 반복 변수: 1  
현재 반복 변수: 0

# for 반복문: 반대로 반복하기

- reversed() 함수 사용하는 방법

```
01  # 역반복문
02  for i in reversed(range(5)):
03      # 출력합니다.
04      print("현재 반복 변수: {}".format(i))
```

실행결과

현재 반복 변수: 4
현재 반복 변수: 3
현재 반복 변수: 2
현재 반복 변수: 1
현재 반복 변수: 0

- while 반복문

- 리스트 또는 딕셔너리 내부의 요소를 특정 횟수만큼 반복

```
while 불 표현식:  
    문장
```

```
01  # while 반복문을 사용합니다.  
02  while True:  
03      # "."을 출력합니다.  
04      # 기본적으로 end가 "\n"이라 줄바꿈이 일어나는데  
05      # 빈 문자열 ""로 바꿔서 줄바꿈이 일어나지 않게 합니다.  
06      print(".", end="")
```

실행결과

```
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....
```

# while 반복문 : for 반복문처럼 사용하기

```
01  # 반복 변수를 기반으로 반복하기
02  i = 0
03  while i < 10:
04      print("{}번째 반복입니다.".format(i))
05      i += 1
```

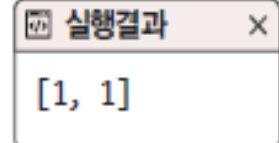
실행결과

0번째 반복입니다.  
1번째 반복입니다.  
2번째 반복입니다.  
3번째 반복입니다.  
4번째 반복입니다.  
5번째 반복입니다.  
6번째 반복입니다.  
7번째 반복입니다.  
8번째 반복입니다.  
9번째 반복입니다.

# while 반복문: 상태를 기반으로 반복하기

- 리스트 내부에서 해당하는 값을 여러 개 제거
  - while 반복문의 조건을 '리스트 내부에 요소가 있는 동안'으로 지정

```
01  # 변수를 선언합니다.  
02  list_test = [1, 2, 1, 2]  
03  value = 2  
04  
05  # list_test 내부에 value가 있다면 반복  
06  while value in list_test:  
07      list_test.remove(value)  
08  
09  # 출력합니다.  
10  print(list_test)
```



실행결과

[1, 1]



# while 반복문 : 시간을 기반으로 반복하기

- 예시 - 유닉스 타임 구하기
  - 시간 관련된 기능 가져오기

```
>>> import time
```

```
>>> time.time()  
1557241486.6654928
```

# while 반복문 : 시간을 기반으로 반복하기

- 유닉스 타임과 while 반복문을 조합
  - 5초 동안 반복하기

```
01  # 시간과 관련된 기능을 가져옵니다.  
02  import time  
03  
04  # 변수를 선언합니다.  
05  number = 0  
06  
07  # 5초 동안 반복합니다.  
08  target_tick = time.time() + 5  
09  while time.time() < target_tick:  
10      number += 1  
11  
12  # 출력합니다.  
13  print("5초 동안 {}번 반복했습니다.".format(number))
```

실행결과

5초 동안 14223967번 반복했습니다.

# while 반복문: break 키워드/continue 키워드

- break 키워드
  - 반복문 벗어날 때 사용하는 키워드

```
01  # 변수를 선언합니다.
02  i = 0
03
04  # 무한 반복합니다.
05  while True:
06      # 몇 번째 반복인지 출력합니다.
07      print("{}번째 반복문입니다.".format(i))
08      i = i + 1
09      # 반복을 종료합니다.
10      input_text = input("> 종료하시겠습니까?(y): ")
11      if input_text in ["y", "Y"]:
12          print("반복을 종료합니다.")
13          break
```

**실행결과**

0번째 반복문입니다  
> 종료하시겠습니까?(y/n): n

1번째 반복문입니다  
> 종료하시겠습니까?(y/n): n

2번째 반복문입니다  
> 종료하시겠습니까?(y/n): n

3번째 반복문입니다  
> 종료하시겠습니까?(y/n): n

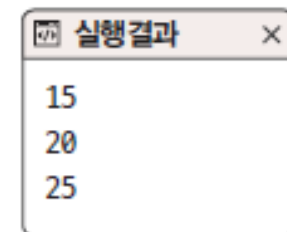
4번째 반복문입니다  
> 종료하시겠습니까?(y/n): y

반복을 종료합니다.

- continue 키워드

- 현재 반복을 생략하고 다음 반복으로 넘어감

```
01  # 변수를 선언합니다.  
02  numbers = [5, 15, 6, 20, 7, 25]  
03  
04  # 반복을 돌립니다.  
05  for number in numbers:  
06      # number가 10보다 작으면 다음 반복으로 넘어갑니다.  
07      if number < 10:  
08          continue  
09      # 출력합니다.  
10      print(number)
```



실행결과
15
20
25

- if else 구문 사용도 가능한 경우이나, continue 키워드 사용하면 이후 처리의 들여쓰기를 하나 줄일 수 있음

## continue 키워드를 사용하지 않은 경우

```
# 반복을 돌립니다.  
for number in numbers:  
    # 반복 대상을 한정합니다.  
    if number >= 10:  
        # 문장  
        # 문장  
        # 문장  
        # 문장  
        # 문장
```

## continue 키워드를 사용한 경우

```
# 반복을 돌립니다.  
for number in numbers:  
    # 반복 대상에서 제외해버립니다.  
    if number < 10:  
        continue  
    # 문장  
    # 문장  
    # 문장  
    # 문장  
    # 문장
```

- **범위** : 정수의 범위 나타내는 값으로, range() 함수로 생성
- **while 반복문** : 조건식을 기반으로 특정 코드를 반복해서 실행할 때 사용하는 구문
- **break 키워드** : 반복문을 벗어날 때 사용하는 구문
- **continue 키워드** : 반복문의 현재 반복을 생략할 때 사용하는 구문

- 다음 표를 채워 보세요.  
코드가 여러 개 나올 수 있는 경우 가장 간단한 형태를 넣어 주세요.

코드	나타내는 값
<code>range(5)</code>	<code>[0, 1, 2, 3, 4]</code>
<code>range(4, 6)</code>	
<code>range(7, 0, -1)</code>	
<code>range(3, 8)</code>	<code>[3, 4, 5, 6, 7]</code>
	<code>[3, 6, 9]</code>

- 빈칸을 채워 키와 값으로 이루어진 각 리스트를 조합해 하나의 딕셔너리를 만들어 보세요.

```
# 숫자는 무작위로 입력해도 상관없습니다.  
key_list = ["name", "hp", "mp", "level"]  
value_list = ["기사", 200, 30, 5]  
character = {}
```

```
# 최종 출력  
print(character)
```

실행결과

```
{'name': '기사', 'hp': 200, 'mp': 30, 'level': 5}
```



- 1부터 숫자를 하나씩 증가시키면서 더하는 경우를 생각해 봅시다. 몇을 더할 때 1000을 넘는지 구해 보세요. 그리고 그때의 값도 출력해보세요. 다음은 1000이 넘는 경우를 구한 예입니다.

1, 1 + 2 = 3, 1 + 2 + 3 = 6, 1 + 2 + 3 + 4 = 10...

```
limit = 10000
```

```
i = 1
```

```
# sum은 파이썬 내부에서 사용하는 식별자이므로 sum_value라는 변수 이름을 사용합니다.
```

```
print("{}를 더할 때 {}을 넘으며 그때의 값은 {}입니다.".format(i, limit, sum_value))
```

실행결과

142를 더할 때 10000을 넘으며 그때의 값은 10011입니다.