파이썬 프로그래밍
강의 노트 #07

반복문

반복문을 사용하지 않는 입력과 합계

- 정수 세 개를 입력받고 합을 구해서 화면에 출력하는 프로그램 작성

```
# 방법 1
n1 = int(input("정수 한 개를 입력하세요: "))
n2 = int(input("정수 한 개를 입력하세요: "))
n3 = int(input("정수 한 개를 입력하세요: "))
sum = n1 + n2 + n3
print(sum)
```

반복문을 사용하지 않는 입력과 합계

```
# 방법 2
sum = 0
n1 = int(input("정수 한 개를 입력하세요: "))
sum += n1
n2 = int(input("정수 한 개를 입력하세요: "))
sum += n2
n3 = int(input("정수 한 개를 입력하세요: "))
sum += n3
print(sum)
```

반복문을 사용하지 않는 입력과 합계

```
# 방법 3
sum = 0
n = int(input("정수 한 개를 입력하세요: ")) # (1)
sum += n                                    # (2)
n = int(input("정수 한 개를 입력하세요: "))
sum += n
n = int(input("정수 한 개를 입력하세요: "))
sum += n
print(sum)
```

반복문을 사용하지 않는 입력과 합계

```
# 방법 4
def getSum(s):
    n = int(input("정수 한 개를 입력하세요: "))
    s += n
    return s

sum = 0
sum = getSum(sum)
sum = getSum(sum)
sum = getSum(sum)
print(sum)
```

반복문 (Loop Statements)

- 프로그램을 작성하고 실행시키는 이유 중 한 가지는 사람이 하기 싫은 단순하고 반복적인 작업을 시키기 위해서임
- 반복문을 사용하려면 종료 조건을 이해해야 함
  - 종료조건이란 반복을 종료시키는 조건
  - 주로 True 또는 False로 결과가 나타나는 조건식이 많이 사용됨
- 반복문이 필요한 예
  - 버스 정류장에서 7016 버스가 있으면 탄다

```
if "7016버스가 정류장에 있으면":
    버스타기()
```

반복문 (Loop Statements)

- 만약 버스가 없으면 기다린다

```
if "7016버스가 정류장에 있으면":
    버스타기()
if "7016버스가 정류장에 없으면":
    기다리기()
```

- 버스를 타려면, 두 과정을 반복해야 함

```
if "7016버스가 정류장에 있으면":
    버스타기()
if "7016버스가 정류장에 없으면":
    기다리기()
if "7016버스가 정류장에 있으면":
    버스타기()
if "7016버스가 정류장에 없으면":
    기다리기()
... # 생략 표시는 두 개의 if문이 반복됨을 보임
```

반복문 (Loop Statements)

- 종료 조건이 만족될 때까지 반복하라

```
반복 조건이 만족될 때까지:
    기다림
if "7016버스가 정류장에 있으면":
    버스타기()
```

- 종료 조건
  - 버스가 오면

```
while "7016 버스가 정류장에 없으면":
    기다림
if "7016버스가 정류장에 있으면":
    버스타기()
```

while 문

```
while 조건표현식:
    코드 블록
```

- while 문
  - if문은 조건표현식이 만족되면 한 번만 실행되지만, while 문은 조건이 만족되지 않을 때까지 계속 반복됨 (조건 반복형)
  - 조건표현식의 결과가 False가 되면(혹은 0, 빈 문자열 등) 코드블록이 실행되지 않고 다음 실행 구문으로 이동
  - 조건에 따른 반복 작업에 최적화 되어 있음

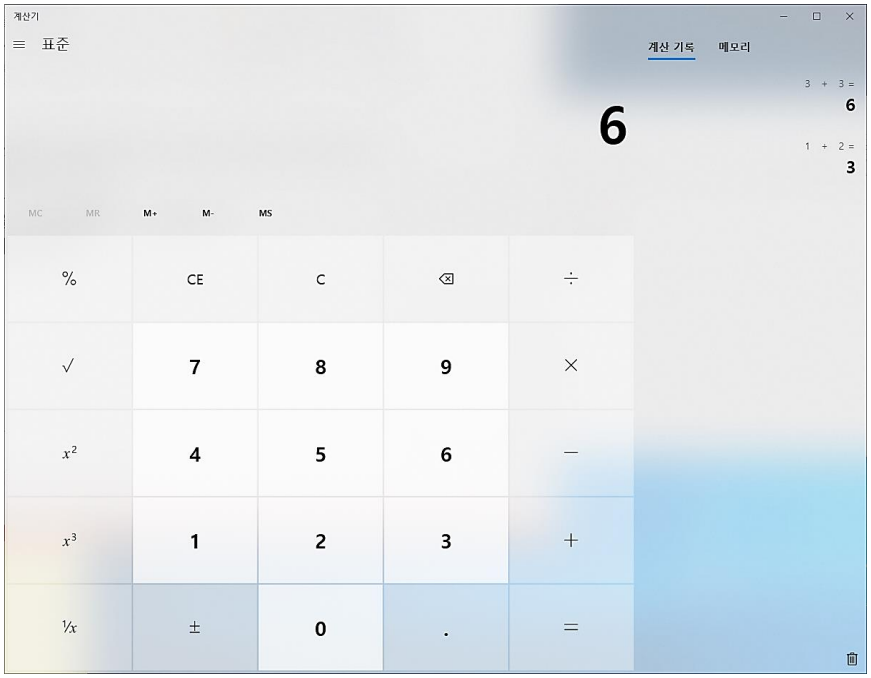
while 문

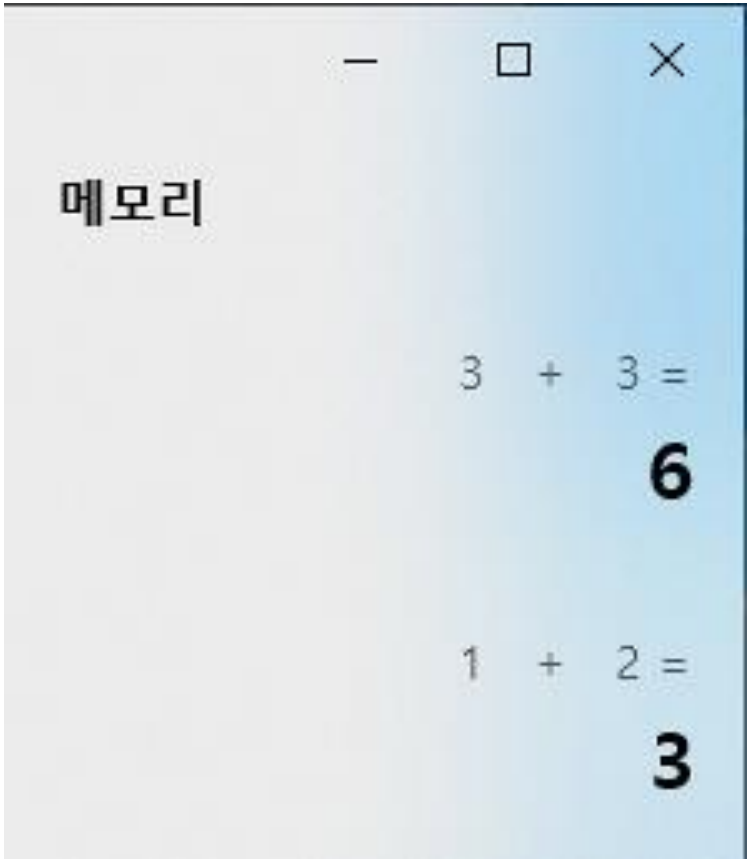
- while문의 예

```
>>> n = 1
>>> while n < 4:
...     print(n)
...     n += 1
...
1
2
3
```

while 문

- 1~10까지의 합계 구하기
  - 1 + 2 + 3 + … + 9 + 10
  - 계산 방법을 다른 사람에게 설명한다면?
  - 계산기를 사용한다면?





while 문

- 코드로 표현하면?

```
sum = 0
sum = 0 + 1
sum = 0 + 1 + 2
sum = 0 + 1 + 2 + 3
…
sum = 0 + 1 + 2 + 3 + … + 9
sum = 0 + 1 + 2 + 3 + … + 9 + 10
```

```
sum = 0
sum += 1
sum += 2
sum += 3
…
sum += 9
sum += 10
```

while 문

```
sum = 0
n = 1   # 1
sum += n
n += 1  # 2
sum += n
n += 1  # 3
…
n += 1  # 10
sum += n
print(sum)
```

n <= 10

```
sum = 0
n = 1
if n <= 10:
    sum += n
    n += 1
if n <= 10:
    sum += n
    n += 1
…
print(sum)
```

n <= 10

while문

- 반복문으로 작성

```
sum = 0
n = 1
while n <= 10:
    sum += n
    n += 1
print(sum)
```

실습문제 1

- 문제
  - 사용자로부터 숫자 5개를 입력받고, 가장 큰 값을 찾아서 반환하는 함수를 작성하고, 이 함수를 활용해서 가장 큰 입력값을 화면에 출력하는 프로그램을 구현
- 요구사항
  - 사용자가 입력하는 숫자 5개는 모두 0보다 큰 양수로 가정
  - 자료의 정렬 알고리즘이 적용된 함수를 사용하지 않음

실습문제 1

```
print("5개 정수를 입력하면 가장 큰 값을 화면에 출력합니다")
maxValue = 0
count = 1
while count < 6:
    n = int(input("0보다 큰 정수 한 개를 입력하세요: "))
    if n > maxValue:
        maxValue = n
    count += 1
print("최대값 = ", maxValue)
```

실습문제 2

- 문제
  - 사용자로부터 1~100사이의 정수를 입력 받고 화면에 출력하는 프로그램을 작성
  - 1~100 범위 밖의 정수가 입력되면 다시 입력받기
- 요구사항
  - 사용자가 입력하는 숫자는 음수와 양수를 포함하는 정수로 제한
  - 1~100사이의 정수는 1, 2, …, 99, 100을 나타냄(100을 포함)

실습문제 2

```
n = int(input("1~100 사이의 정수를 한 개 입력하세요:"))
while n < 1 or n > 100: # (1)
    n = int(input("1~100 사이의 정수를 한 개
입력하세요: ")) # (2)

print(n)
```

for 문

- for문은 주로 정해진 횟수만큼 반복하거나 제한된 개수의 요소들을 순회하면서 처리할 때 주로 사용하는 반복문
  - 자주 활용되는 반복문

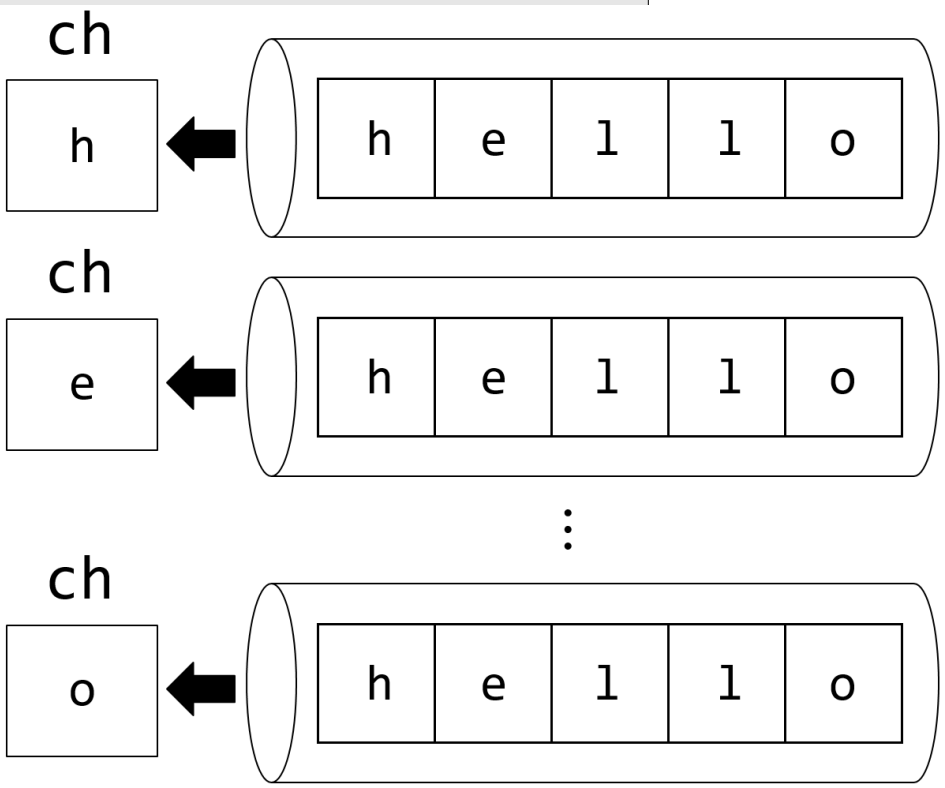
```
for 변수 in 순서가_있는_객체:
    코드_블록
```

  - 순서가 있는 객체
    - 문자열, range 객체, 리스트(list), 튜플(tuple) 등
  - 순서가_있는_객체의 각 요소들은 변수에 치환되고 해당 변수를 이용해서 코드_블록을 실행함
  - 반복 횟수는 순서가_있는_객체의 크기 혹은 아이템의 개수

for 문

- for문과 문자열을 이용해서 글자들을 한 줄에 한 개씩 화면에 출력

```
>>> s = "hello"
>>> for ch in s: # for ch in "hello":
...     print(ch)
...
h
e
l
l
o
```



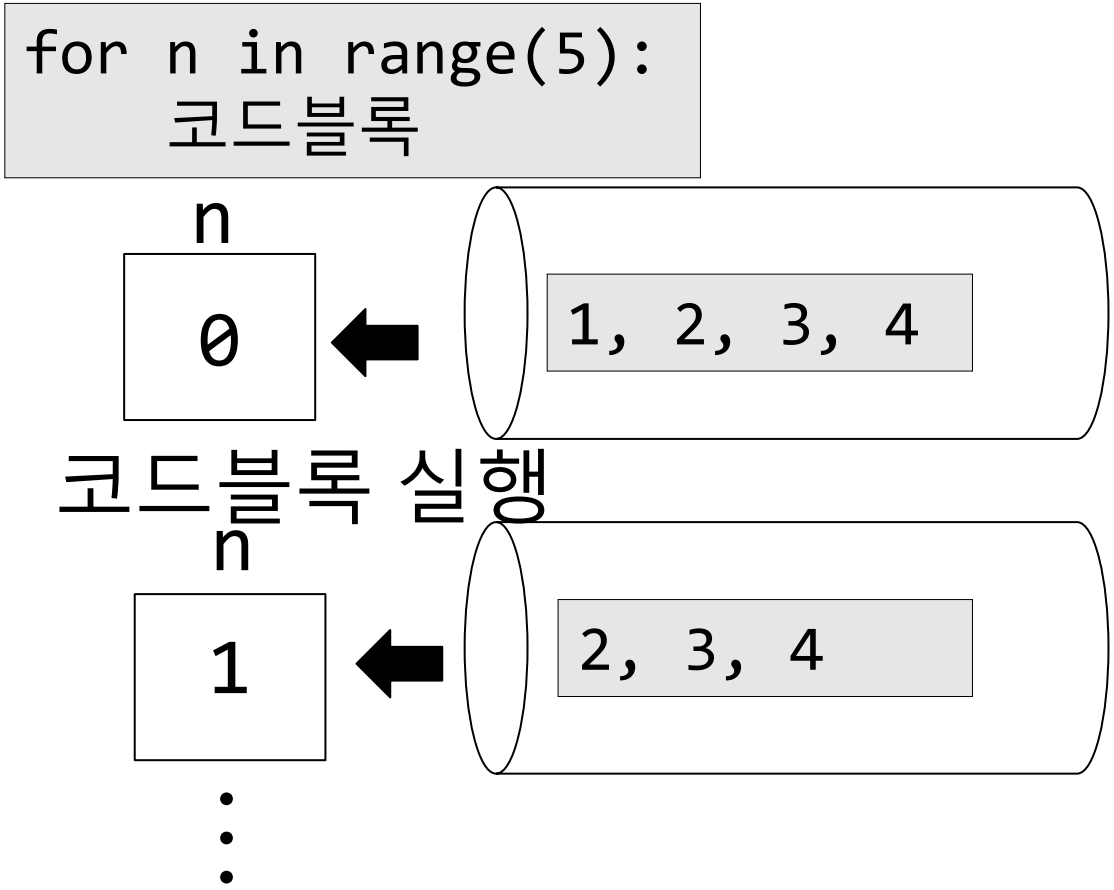
for 문과 range()

- for 반복문
  - 일정 정수 범위 또는 패턴에 대해서 수행되는 경우가 많은데 (예: 특정 횟수만큼 혹은 특정 주기를 반복), 이럴 때에는 range 함수를 이용

```
for 변수 in range(범위):
    코드블록
```

  - range()는 range 객체를 반환

for 문과 range()



for 문과 range()

- range()의 범위는 다양하게 정의 가능함

```
range(c)   # range(0, 1, 2, …, c – 1)
range(c, d) # range(c, c + 1, c + 2, …, d – 1)
range(c, d, e) # range(c, c + e, c + 2 * e,
                 # …, c + n * e (c < d and e >
0이면, c + n * e는 d보다 작은 가장 큰 정수, c > d
and e < 0이면, c + n * e는 d보다 큰 가장 작은
정수))
```

- 예)

```
a = range(-5, -2)     # a = range(-5, -4, -3)
a = range(-5, 3, 2)   # a = range(-5, -3, -1, 1)
a = range(7, 1, -3)   # a = range(7, 4)
a = range(7, -3, -2) # a = range(7, 5, 3, 1, -1)
```

for 문과 range()

- 횟수를 세는 range 객체

```
>>> for i in range(3):
...     print("hello")
...
hello
hello
hello
```

for 문과 range()

- 1~10까지의 합 구하기

```
>>> sum = 0
>>> for n in range(1, 11): # n을 1~10까지 변경
...     sum += n
...
>>> print(sum)
55
```

실습문제 3

- 문제
  - 정수의 약수들을 화면에 출력하는 프로그램을 작성
  - 12와 16의 약수들을 출력할 것
- 요구사항
  - 정수를 한 개 인자로 전달받고, 약수를 화면에 모두 출력하는 함수를 작성

실습문제 3

- 최종 코드

```
def printDivisors(n):
    for i in range(1, n + 1):
        if n % i == 0:
            print(i)

print("12의 약수들")
printDivisors(12)
print("\n16의 약수들")
printDivisors(16)
```

실습문제 4

- 문제
  - 동전을 던져서 앞/뒷면이 나오는 횟수를 세고, ½ 확률에 수렴하는지 확인하는 프로그램을 작성
  - 컴퓨터에서 동전을 던질 수는 없으므로, random.randint() 함수를 이용해서 두 개 숫자 중 한 개를 무작위로 생성하여 동전의 앞/뒷면을 대신할 것
  - 100, 1000, 10000회 던져서 앞/뒷면이 나오는 횟수를 각각 출력
- 요구사항
  - 정해진 횟수만큼 동전을 던지고, 앞/뒷면이 나오는 횟수를 출력하는 함수를 구현
  - 동전을 던지는 횟수는 함수에 입력으로 전달
  - 앞/뒷면이 나오는 확률을 구해서 각각 출력

실습문제 4

- 최종 코드

```
import random
def flipCoin(num):
    countFront = 0 # 앞면이 나오는 횟수
    countBack = 0 # 뒷면이 나오는 횟수
    for i in range(num): # num번 반복
        if random.randint(0, 1) == 0:
            countFront += 1
        else:
            countBack += 1
    print(num, "번 동전을 던짐")
    print("앞면이 나올 확률:", countFront / num)
    print("뒷면이 나올 확률:", countBack / num)
```

실습문제 4

```
flipCoin(100)
flipCoin(1000)
flipCoin(10000)
```

중첩 반복문

- 중첩 반복문
  - 반복문의 코드블록에 다른 반복문이 있는 경우

```
for 변수1 in 순서가_있는_객체1:
    for 변수2 in 순서가_있는_객체2:
        코드_블록
while 조건식1:
    while 조건식2:
        코드_블록:
for 변수1 in 순서가_있는_객체1:
    while 조건식1:
        코드_블록
while 조건식1:
    for 변수1 in 순서가_있는_객체1:
        코드_블록
```

실습문제 5

- 문제
  - 다음 표를 출력하는 프로그램 작성

1 * n	2 * n	3 * n	4 * n	…	8 * n
1	2	3	4	…	8
2	4	6	8	…	16
3	6	9	12	…	24
4	8	12	16	…	32
5	10	15	20	…	40
…	…	…	…	…	…
10	20	30	40	…	80

- 요구사항
  - 반복문 사용하고 같은 행에서 셀(cell)은 탭 문자로 분리

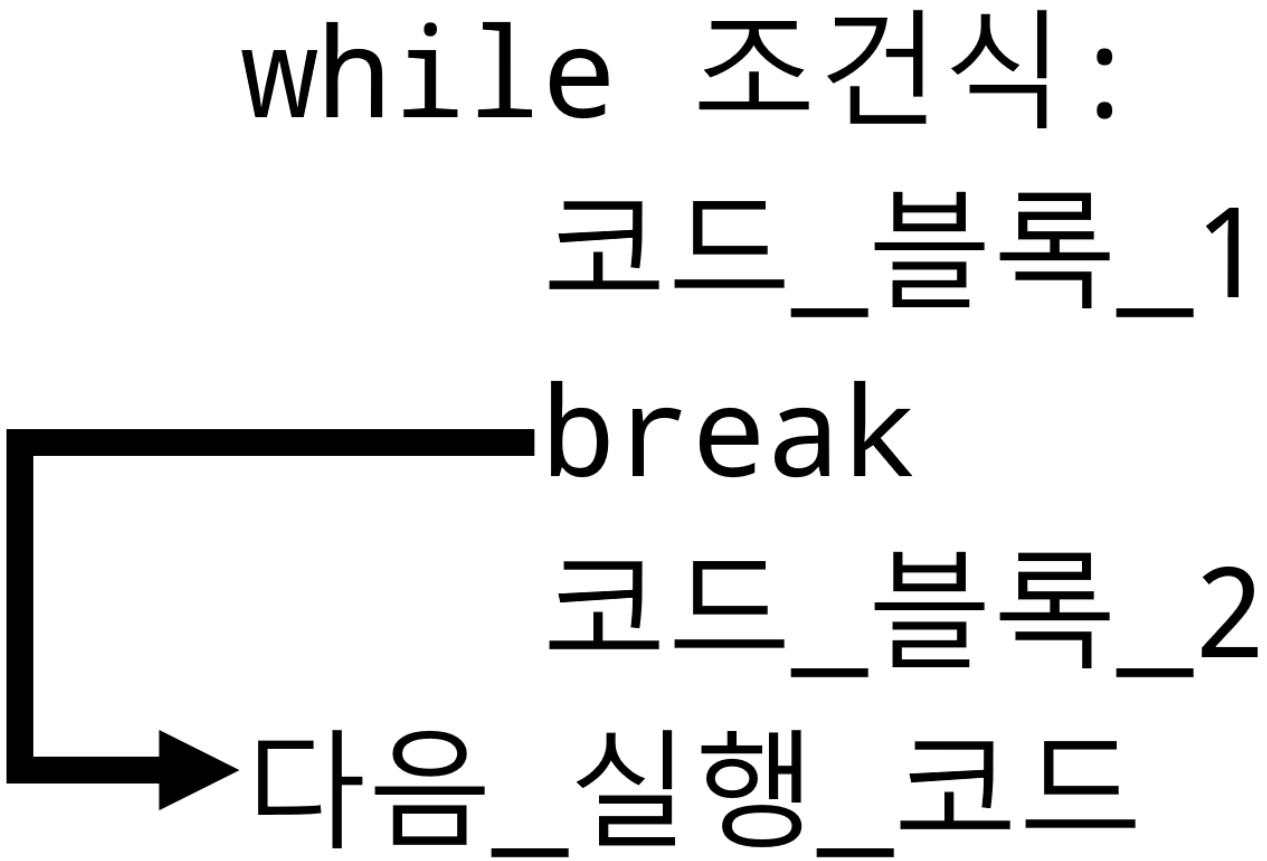
실습문제 5

- 최종 코드

```
for i in range(1, 9):
    if i < 8:
        print(f"{i} * n", end = '\t')
    else:
        print(f"{i} * n")
for n in range(1, 11):
    for i in range(1, 9):
        if i < 8:
            print(i * n, end = '\t')
        else:
            print(i * n)
```

break 문

- break문
  - 반복문(while 또는 for문)의 실행을 중단하고 반복문을 빠져나옴



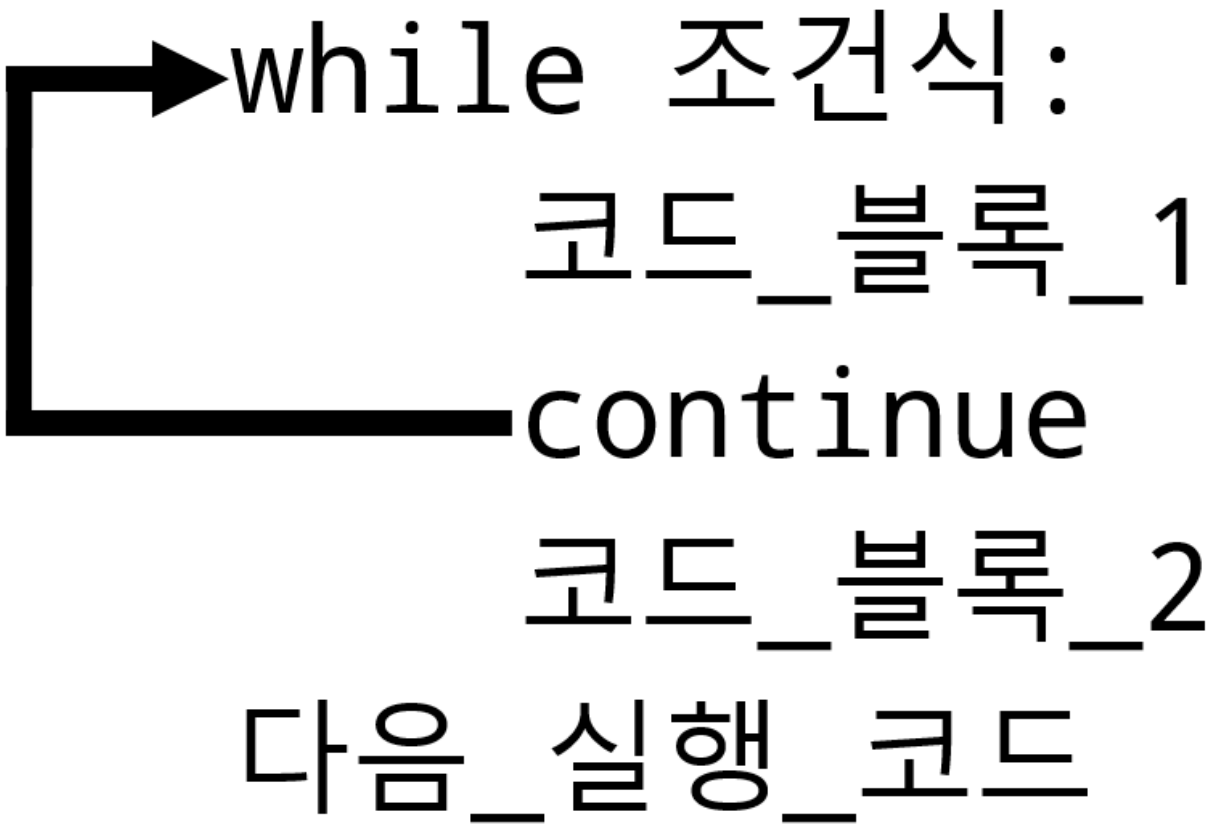
break 문

- 예제 코드
  - 문자열 s에서 첫 번째 숫자의 위치를 출력하는 프로그램

```
s = "what are the 10 best-selling products in this
shopping mall?"
idx = 0
while idx < len(s):
    if s[idx].isdigit():
        print(idx)
        break
    idx += 1
```

continue문

- 반복문에서 사용되는 continue문은 남은 코드의 실행을 중단하고 반복문의 처음으로 되돌아감



continue문

- 예제 코드
  - 사용자로부터 10개의 양의 정수를 입력받고 합을 구하는 프로그램 작성

```
count = 0
sum = 0
while count < 10:
    n = int(input("양의 정수를 입력하세요: "))
    if n <= 0: # 0 또는 음수이면
        continue # 반복문의 처음으로 이동
    # CODE_BLOCK_1
    sum += n
    count += 1
    print("count =", count)
print("합 =", sum)
```