



Python Programming for Beginners

05 반복문

2023학년도 2학기

Suk-Hwan Lee

Computer Engineering
Artificial Intelligence

Creating the Future

Dong-A University

Division of Computer Engineering &
Artificial Intelligence

Section01 기본 for 문

■ for 문의 개념

■ 기본 형식

```
for 변수 in range(시작값, 끝값+1, 증가값) :  
    이 부분을 반복  
        range(3)은 range(0, 3, 1)과 같다
```

■ 예 : range() 함수 사용과 내부적 변경

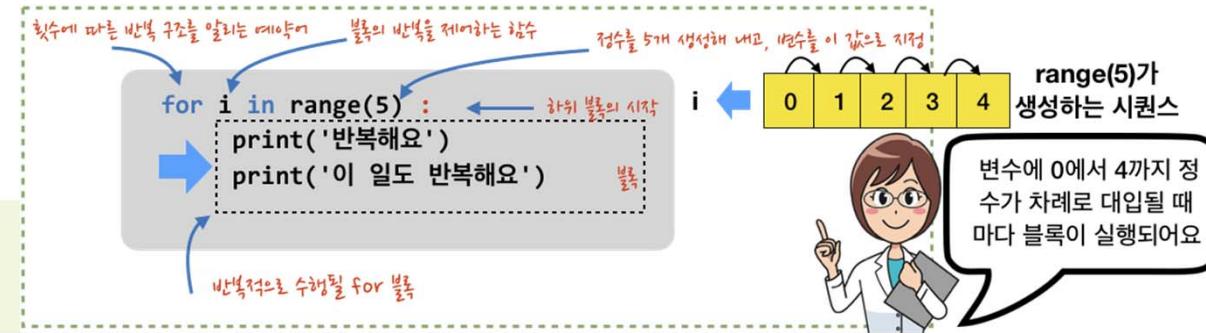
```
for i in range(0, 3, 1) :  
    print("안녕하세요? for 문을 공부 중입니다. ^^")
```

```
for i in [0, 1, 2] :  
    print("안녕하세요? for 문을 공부 중입니다. ^^")
```

■ for-in 다음에는 리스트나 문자열도 올 수 있다

```
for i in [1, 2, 3, 4, 5]:      # in 뒤에 리스트를 넣고 끝에 :을 넣자  
    print("i =", i)           # i 값을 출력해보자
```

```
for i in [1, 2, 3, 4, 5]:      # 1에서 5까지의 리스트  
    print("9 *", i, "=", 9 * i ) # 9*i 값을 출력해보자
```



■ i값 코드 내부 사용

```
for i in range(0, 3, 1) :  
    print("%d : 안녕하세요? for 문을 공부 중입니다. ^^" % i)
```

출력 결과

```
0 : 안녕하세요? for 문을 공부 중입니다. ^^  
1 : 안녕하세요? for 문을 공부 중입니다. ^^  
2 : 안녕하세요? for 문을 공부 중입니다. ^^
```

Tip •_(언더바) : i를 사용하지 않으려면 i 대신 _(언더바) 사용

```
for _ in range(0, 3, 1) :  
    print("안녕하세요? for 문을 공부 중입니다. ^^")
```

Section01 기본 for 문

- 예 : range() 함수의 시작값 2, i값을 1씩 줄여(0이 될 때까지)
print() 함수 3번 실행

```
for i in range(2, -1, -1) :  
    print("%d : 안녕하세요? for 문을 공부 중입니다. ^^" % i)
```

출력 결과

```
2 : 안녕하세요? for 문을 공부 중입니다. ^^  
1 : 안녕하세요? for 문을 공부 중입니다. ^^  
0 : 안녕하세요? for 문을 공부 중입니다. ^^
```

- 예 : 1~5의 숫자들을 차례로 출력

```
for i in range(1, 6, 1) :  
    print("%d " % i, end = " ")
```

출력 결과

```
1 2 3 4 5
```

■ range() 함수

- range(3) 함수는 0, 1, 2까지의 **숫자열 sequence**을 반환한다. 반복할 때마다 변수 i에 이 값을 대입하면서 문장을 반복한다. 즉 첫 번째 반복에서는 i는 0이고 되고 두 번째 반복에서는 1이 된다. 마지막 반복에서 i는 2가 된다.
- 하나 이상의 반복되는 문장이 올 수 있는데, 반복되는 문장은 if 문과 같이 반드시 동일한 간격의 들여쓰기를 해야 한다. 이 때 들여쓰기가 있는 문장들만이 반복된다.

파이썬 쉘에서 range() 함수에 list() 함수를 적용시키면 range() 함수가 생성하는 정수들을 볼 수 있다. 이와 같이 range() 함수는 연속적인 값을 생성하는 일을 한다.

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



잠깐 – range() 함수가 반환하는 값은 range 형이다

파이썬의 range() 함수는 range 형의 자료형을 반환한다. range 형의 자료형은 호출이 발생할 때마다 매번 연속된 값을 생성하여 반환하는 특별한 일을 한다. 따라서 주로 for 문과 함께 사용된다.

Section01 기본 for 문

■ range() 함수

- range() 함수는 여러개의 인자를 이용하여 좀 더 다양한 정수 시퀀스를 생성할 수 있으며, 일반적인 형식은 다음과 같다.
- range(start, stop, step)이라고 호출하면 start에서 시작하여 (stop-1)까지 step 간격으로 정수들이 생성된다.
- 0, 1, 2, 3, 4가 바로 그것이다. 그리고 for 루프는 이 시퀀스의 갯수 만큼 반복 수행한다.
- 여기서 start와 step가 생략될 수 있는데 이 경우 start는 0으로 간주되고 step은 1로 간주된다.
- 하지만 stop 값은 반드시 지정해야만 루프가 수행된다.

```
for i in range(1, 6, 1):
    print(i, end = " ")      # end = " "로 지정하면 줄바꿈을 하지 않고 공백으로 나열한다.
```

```
1 2 3 4 5
```

```
for i in range(10, 0, -1):
    print(i, end = " ")
```

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

☞ 잠깐 – range(n)은 range(0, n, 1)과 같다

range(10) 함수를 사용할 때 가장 혼동하는 부분이 1에서 10까지의 정수가 생성된다고 생각하는 것이다. 반복 횟수로 생각하면 10번 반복은 맞다. 하지만 생성되는 정수는 0부터 9까지이다. 이것은 컴퓨터의 오랜 논쟁거리였다. 지금은 0부터 시작하는 것이 대세가 되었다. 아무튼 range(10)하면 10번 반복되고 생성되는 정수는 0부터 9까지이다. 만약 1부터 10까지의 정수가 필요하면 range(1, 11)로 하면 된다.

생성을 종료하는 값인데, 이 stop은 생성에서 제외된다

생성을 시작하는 값

range(start = 0, stop, step = 1)

한 번에 증가되는 값의 크기



start와 step은 따로
지정하지 않으면 각각
0과 1로 처리되어요.
반드시 지정해야 하는 값은
stop이에요

Section01 기본 for 문

■ for 문을 활용한 합계 구하기

- Code06-02(1).py 1행의 hap 초기화 코드 추가

Code06-02(2).py

```
1 i, hap = 0, 0
2
3 for i in range(1, 11, 1) :
4     hap = hap + i
5
6 print("1에서 10까지의 합계 : %d" % hap)
```

주의 : i 변수와 hap 변수의 값

출력 결과

1에서 10까지의 합계 : 55

- 예 : 500과 1000 사이에 있는 홀수의 합계

Code06-03.py

```
1 i, hap = 0, 0
2
3 for i in range(501, 1001, 2) :
4     hap = hap + i
5
6 print("500과 1000 사이에 있는 홀수의 합계 : %d" % hap)
```

출력 결과

500과 1000 사이에 있는 홀수의 합계 : 187500

SELF STUDY 6-1

Code06-03.py를 0과 100 사이에 있는 7의 배수 합계를 구하도록 수정해 보자.

출력 결과

0과 100 사이에 있는 7의 배수 합계 : 735

Section01 기본 for 문

■ 키보드로 입력한 값까지 합계 구하기

- 키보드로 입력한 수까지의 합계 구하기
 - `input()` 함수로 1부터 사용자가 입력한 수까지 합계 구하는 프로그램

Code06-04.py

```
1 i, hap = 0, 0
2 num = 0
3
4 num = int(input("값을 입력하세요 : "))
5
6 for i in range(1, num + 1, 1) :
7     hap = hap + i
8
9 print("1에서 %d까지의 합계 : %d" % (num, hap))
```

2행 : 사용자가 입력한 값 저장할 num 변수 선언
4행 : `input()` 함수로 사용자가 입력한 숫자를 num에 대입
6행 : `range(1, 입력숫자+1, 1)`을 사용해 1부터 사용자가 입력한 숫자(num)까지 1씩 증가하면서 for 문 반복

출력 결과

값을 입력하세요 : 100
1에서 100까지의 합계 : 5050

9행 : 사용자가 입력한 숫자까지 합계를 구해 사용자가 입력한 숫자와 함께 출력

Section01 기본 for 문

- 예: 시작값과 끝값, 증가값까지 사용자 입력

Code06-05.py

```
1 i, hap = 0, 0
2 num1, num2, num3 = 0, 0, 0
3
4 num1 = int(input("시작값을 입력하세요 : "))
5 num2 = int(input("끝값을 입력하세요 : "))
6 num3 = int(input("증가값을 입력하세요 : "))
7
8 for i in range(num1, num2 + 1, num3) :
9     hap = hap + i
10
11 print("%d에서 %d까지 %d씩 증가시킨 값의 합계 : %d" % (num1, num2, num3, hap))
```

4~6행 : 값 3개 입력
8행 : 입력한 값 사용 range() 지정

출력 결과

```
시작값을 입력하세요 : 2
끝값을 입력하세요 : 300
증가값을 입력하세요 : 3
2에서 300까지 3씩 증가시킨 값의 합계 : 15050
```

Section01 기본 for 문

- 예: 사용자가 입력한 숫자의 단에서 구구단을 출력

Code06-06.py

```
1 i, dan = 0, 0
2
3 dan = int(input("단을 입력하세요 : "))
4
5 for i in range(1, 10, 1) :
6     print("%d X %d = %2d" % (dan, i, dan * i))
```

1행 : 출력하려는 단을 입력받을 변수 선언
3행 : 키보드로 입력
5행 : i는 1에서 9까지 증가
6행 : 구구단의 각 행 출력

출력 결과

단을 입력하세요 : 7

```
7 X 1 = 7
7 X 2 = 14
... 중략 ...
7 X 9 = 63
```

SELF STUDY 6-2

Code06-06.py를 수정해서 입력한 단을 거꾸로 출력하도록 해보자.

출력 결과

단을 입력하세요 : 7

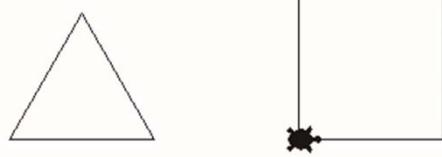
```
9 X 7 = 63
8 X 2 = 56
... 중략 ...
1 X 9 = 7
```

Section01 기본 for 문

LAB⁵⁻² 반복을 사용하여 도형을 그리자

터틀 그래픽에서도 반복을 사용할 수 있다. 정삼각형과 정사각형을 반복을 이용하여 화면에 그려보자. for 문을 사용하면 같은 문장을 반복해서 적을 필요가 없다.

원하는 결과



```
import turtle  
t = turtle.Turtle()  
t.shape("turtle")  
  
# 정삼각형 그리기  
for i in range(3):  
    t.forward(100)  
    t.left(360/3) # 360/3을 통해 120도 왼쪽으로 틀기  
  
# 정사각형을 그리기 위하여 터틀을 이동하기  
t.penup() # 펜 자국이 남지 않도록 펜을 들어서 이동한다  
t.goto(200, 0)  
t.pendown() # 이동을 마치면 펜은 내리도록 한다  
  
# 정사각형 그리기  
for i in range(4):  
    t.forward(100)  
    t.left(360/4) # 360/4를 통해 90도 왼쪽으로 틀기
```

Section01 기본 for 문

LAB⁵⁻⁵ 반복을 이용하여 팩토리얼 계산하기

사용자로부터 임의의 정수 n을 입력받은 뒤에 for 문을 이용하여 팩토리얼을 계산해보자. 팩토리얼 n!은 1부터 n까지의 정수를 모두 곱한 것을 의미한다. 즉, $n! = 1 \times 2 \times 3 \times \dots \times (n-1) \times n$ 이다.

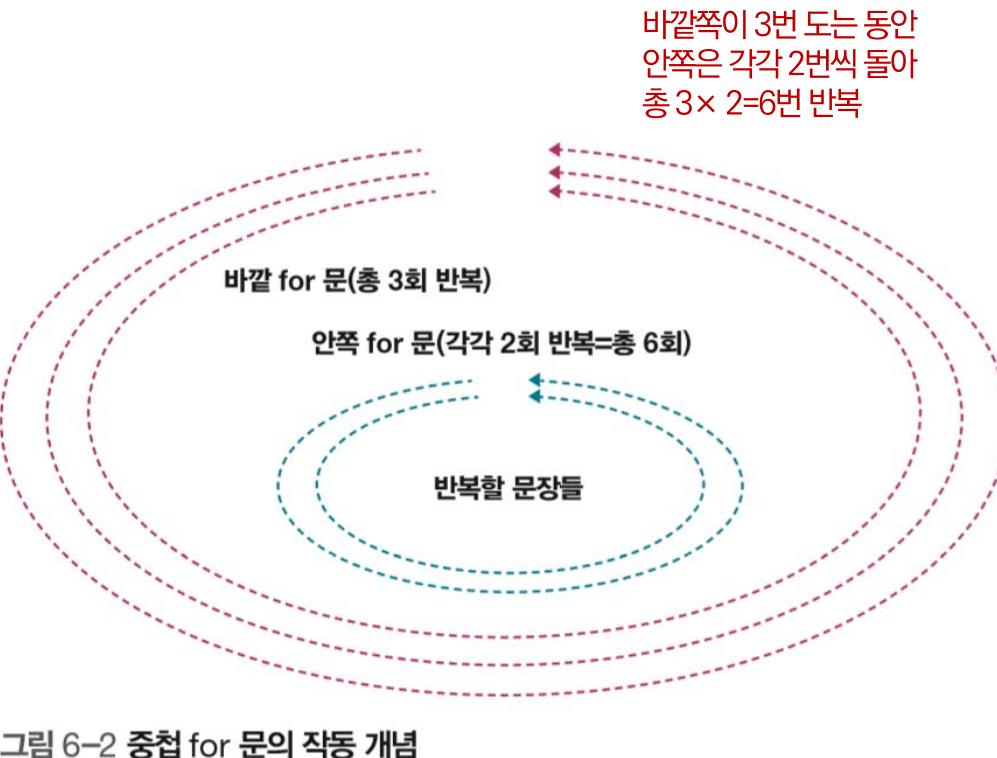
원하는 결과

정수를 입력하시오: 10
10!은 3628800 이다.

```
n = int(input("정수를 입력하시오: "))  
fact = 1  
for i in range(1, n+1):  
    fact = fact * i  
  
print(n, "!은", fact, "이다.")
```

Section02 중첩 for 문

■ 중첩 for 문의 개념



■ 중첩 for 문의 기본 형식

```
for i in range(0, 3, 1) :  
    for k in range(0, 2, 1) :  
        print("파이썬은 꿀잼입니다. ^~ (i값 : %d, k값 : %d)" % (i, k))
```

출력 결과

```
파이썬은 꿀잼입니다. ^~ (i값 : 0, k값 : 0)  
파이썬은 꿀잼입니다. ^~ (i값 : 0, k값 : 1)  
파이썬은 꿀잼입니다. ^~ (i값 : 1, k값 : 0)  
파이썬은 꿀잼입니다. ^~ (i값 : 1, k값 : 1)  
파이썬은 꿀잼입니다. ^~ (i값 : 2, k값 : 0)  
파이썬은 꿀잼입니다. ^~ (i값 : 2, k값 : 1)
```

Section02 중첩 for 문

■ 처리 순서

- 외부 변수인 i는 계속 0, 1, 2로 변경된 후 끝나지만, 내부 변수인 k는 0과 1을 계속 반복

① 외부 for 문 1회 : i에 0을 대입

내부 for 문 1회 : k에 0을 대입 후 print() 함수 수행
내부 for 문 2회 : k에 1을 대입 후 print() 함수 수행

② 외부 for 문 2회 : i에 1을 대입

내부 for 문 1회 : k에 0을 대입 후 print() 함수 수행
내부 for 문 2회 : k에 1을 대입 후 print() 함수 수행

③ 외부 for 문 3회 : i에 2를 대입

내부 for 문 1회 : k에 0을 대입 후 print() 함수 수행
내부 for 문 2회 : k에 1을 대입 후 print() 함수 수행

■ 중첩 for 문에서 i와 k값 변화

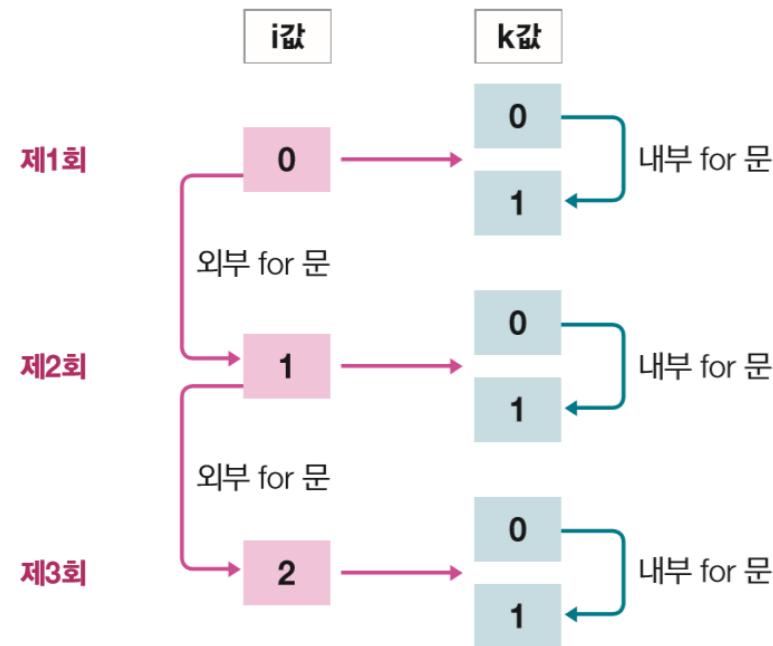


그림 6-3 중첩 for 문에서 i와 k값 변화

Section02 중첩 for 문

■ 중첩 for 문의 활용

- 예 : 중첩 for 문 활용 2단부터 9단까지 구구단 출력



그림 6-4 구구단에서 i와 k 변수 추출

Section02 중첩 for 문

Code06-07.py

```
1 i, k = 0, 0
2
3 for i in range(2, 10, 1) :      3행 : 2단에서 9단까지 반복
4     for k in range(1, 10, 1) :    4행 : 각 단의 뒷자리 숫자 1에서 9까지 반복
5         print("%d X %d = %2d" % (i, k, i * k))
6     print("")                      5행 : 구구단을 형식에 맞추어 출력
                                    6행 : 각 단이 끝나면 한 줄 띄우려고 사용
```

출력 결과

```
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
...
...
9 X 8 = 72
9 X 9 = 81
```

SELF STUDY 6-3

Code06-07.py를 각 단의 제목이 출력되도록 수정해 보자.

출력 결과

```
## 2단 ##
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
```

Section02 중첩 for 문

■ [프로그램 1]의 완성

- 가로 먼저 출력 : 일단 세로 방향으로 한 번 출력하면 다시 위로 올라가서 출력 불가

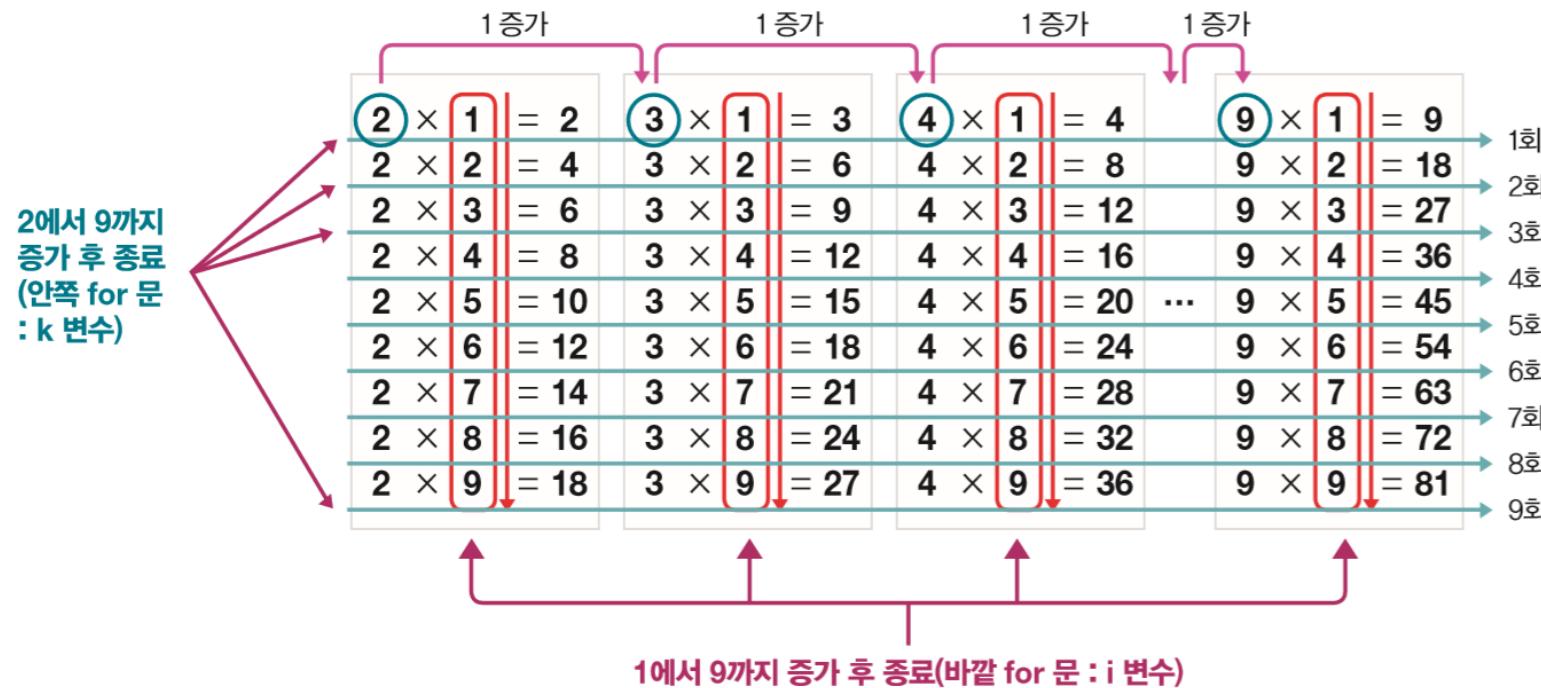


그림 6-5 구구단에서 i와 k 변수 추출(단 가로 먼저 출력)

Section02 중첩 for 문

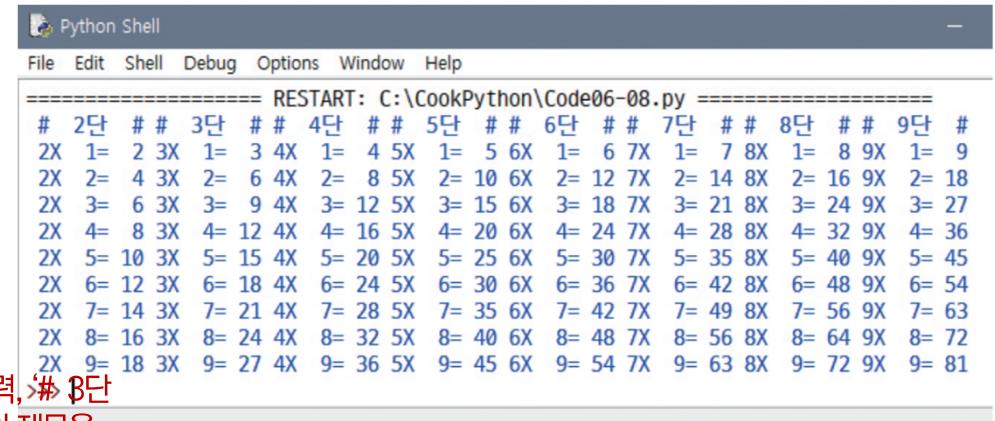
Code06-08.py

```
1 ## 전역 변수 선언 부분 ##
2 i, k, guguLine = 0, 0, ""
3
4 ## 메인 코드 부분 ##
5 for i in range(2, 10) :
6     guguLine = guguLine + ("# %d단 #" % i)
7
8 print(guguLine)
9
10 for i in range(1, 10) :
11     guguLine = ""
12     for k in range(2, 10) :
13         guguLine = guguLine + str("%2dX %2d= %2d" % (k, i, k * i))
14     print(guguLine)
```

2행 : 각 줄에 출력될 문자열 저장하는
guguLine 변수 준비

5~6행 : 맨 뒤의 단 제목 출력, '# 2단 #' 하나 출력, '# 3단 #' 하나 출력 아니라 guguLine에 각 단의 제목을
문자열로 모두 넣은 후 8행에서 한 번에 출력

10~14행 : 중첩 for 문으로 구구단 출력

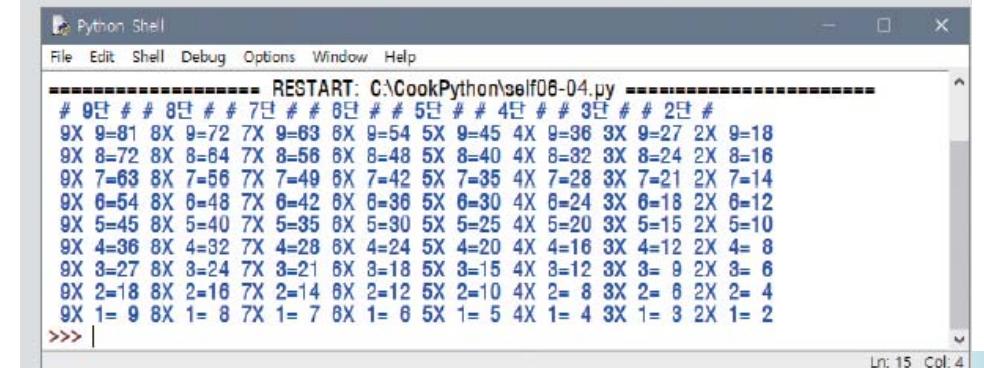


```
Python Shell
File Edit Shell Debug Options Window Help
=====
RESTART: C:\CookPython\Code06-08.py =====
# 2단 # # 3단 # # 4단 # # 5단 # # 6단 # # 7단 # # 8단 # # 9단 #
2X 1= 2 3X 1= 3 4X 1= 4 5X 1= 5 6X 1= 6 7X 1= 7 8X 1= 8 9X 1= 9
2X 2= 4 3X 2= 6 4X 2= 8 5X 2= 10 6X 2= 12 7X 2= 14 8X 2= 16 9X 2= 18
2X 3= 6 3X 3= 9 4X 3= 12 5X 3= 15 6X 3= 18 7X 3= 21 8X 3= 24 9X 3= 27
2X 4= 8 3X 4= 12 4X 4= 16 5X 4= 20 6X 4= 24 7X 4= 28 8X 4= 32 9X 4= 36
2X 5= 10 3X 5= 15 4X 5= 20 5X 5= 25 6X 5= 30 7X 5= 35 8X 5= 40 9X 5= 45
2X 6= 12 3X 6= 18 4X 6= 24 5X 6= 30 6X 6= 36 7X 6= 42 8X 6= 48 9X 6= 54
2X 7= 14 3X 7= 21 4X 7= 28 5X 7= 35 6X 7= 42 7X 7= 49 8X 7= 56 9X 7= 63
2X 8= 16 3X 8= 24 4X 8= 32 5X 8= 40 6X 8= 48 7X 8= 56 8X 8= 64 9X 8= 72
2X 9= 18 3X 9= 27 4X 9= 36 5X 9= 45 6X 9= 54 7X 9= 63 8X 9= 72 9X 9= 81
```

SELF STUDY 6-4

Code06-08.py를 구구단이 거꾸로 출력되도록 수정해 보자.

힌트 range() 함수의 값을 큰 값에서 작은 값으로 변경되도록 해야 한다.



```
Python Shell
File Edit Shell Debug Options Window Help
=====
RESTART: C:\CookPython\self06-04.py =====
# 9단 # # 8단 # # 7단 # # 6단 # # 5단 # # 4단 # # 3단 # # 2단 #
9X 9=81 8X 9=72 7X 9=63 8X 9=54 5X 9=45 4X 9=36 3X 9=27 2X 9=18
9X 8=72 8X 8=64 7X 8=56 8X 8=48 5X 8=40 4X 8=32 3X 8=24 2X 8=16
9X 7=63 8X 7=56 7X 7=49 6X 7=42 5X 7=35 4X 7=28 3X 7=21 2X 7=14
9X 6=54 8X 6=48 7X 6=42 6X 6=36 5X 6=30 4X 6=24 3X 6=18 2X 6=12
9X 5=45 8X 5=40 7X 5=35 6X 5=30 5X 5=25 4X 5=20 3X 5=15 2X 5=10
9X 4=36 8X 4=32 7X 4=28 6X 4=24 5X 4=20 4X 4=16 3X 4=12 2X 4= 8
9X 3=27 8X 3=24 7X 3=21 6X 3=18 5X 3=15 4X 3=12 3X 3= 9 2X 3= 6
9X 2=18 8X 2=16 7X 2=14 6X 2=12 5X 2=10 4X 2= 8 3X 2= 6 2X 2= 4
9X 1= 9 8X 1= 8 7X 1= 7 6X 1= 6 5X 1= 5 4X 1= 4 3X 1= 3 2X 1= 2
>>> |
```

Section03 while 문

■ for 문과 while 문 비교

- for 문의 형식

```
for 변수 in range(시작값, 끝값+1, 증가값)
```

- for 문은 반복할 횟수를 range() 함수에서 결정 후 그 횟수만큼 반복, while 문은 반복 횟수를 결정하기보다는 조건식이 참일 때 반복하는 방식

- for 문과 비슷하게 사용할 수 있는 while 문의 형식

```
변수 = 시작값  
while 변수 < 끝값 :  
    이 부분을 반복  
    변수 = 변수 + 증가값
```

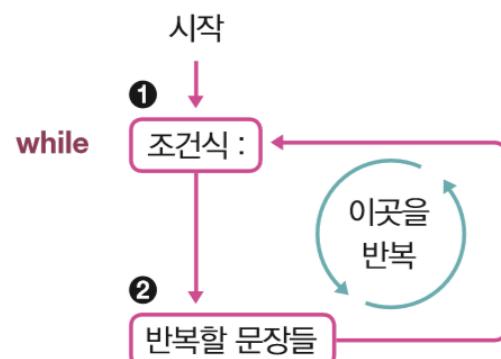


그림 6-6 while 문의 형식과 순서도

- for 문으로 '안녕하세요?~' 문장을 3회 출력하는 코드

```
for i in range(0, 3, 1) :  
    print("%d : 안녕하세요? for 문을 공부 중입니다. ^^" % i)
```

Section03 while 문

■ 문장을 3회 반복하도록 while 문

for 문에서 사용한 변수와 시작값을 i=0으로 while 문 위에 작성
for 문의 끝값 while 문의 조건식인 i<3로 지정
for 문의 증가값 while 문의 마지막에 i=i+1로 작성

```
i = 0
while i < 3 :
    print("%d : 안녕하세요? while 문을 공부 중입니다. ^^" % i)
    i = i + 1
```

출력 결과

```
0 : 안녕하세요? while 문을 공부 중입니다. ^^
1 : 안녕하세요? while 문을 공부 중입니다. ^^
2 : 안녕하세요? while 문을 공부 중입니다. ^^
```

■ 예 : Code06-02(2).py에서 for 문으로 작성한 1에서 10까지의 합계 구하기

Code06-09.py

```
1 i, hap = 0, 0
2
3 i = 1
4 while i < 11 :
5     hap = hap + i
6     i = i + 1
7
8 print("1에서 10까지의 합계 : %d" % hap)
```

3행 : i의 시작값을 1로 지정
4행 : i가 11보다 작으면 참, i가 10일 때까지 5~6행
반복 5행 : hap에 i값(처음에는 1)을 누적
6행 : i를 1 증가

출력 결과

```
1에서 10까지의 합계 : 55
```

SELF STUDY 6-5

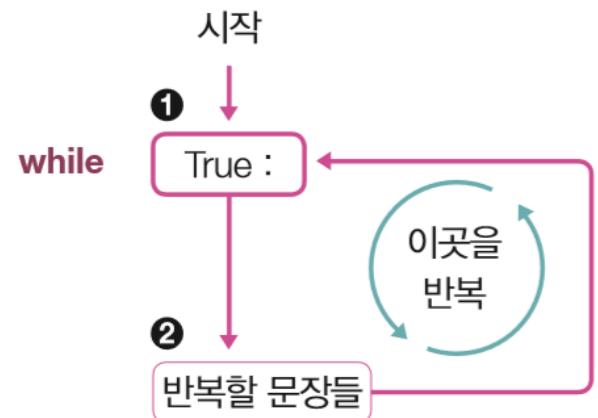
Code06-05.py를 while 문으로 수정해 보자.

힌트 Code06-09.py를 참고한다.

Section03 while 문

■ 무한 루프를 하는 while 문

- 무한 루프 적용 : 'while 조건식 :'에 들어가는 조건식을 True로 지정
- 예 : 무한 루프



```
while True :  
    print("ㅋ", end = " ")
```

출력 결과

ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ~~~ 무한 반복

그림 6-7 while 문을 이용한 무한 루프

Section03 while 문

- 예 : 무한 루프를 사용해 입력한 두 숫자의 합계를 반복해서 계산

Code06-10.py

```
1 hap = 0
2 a, b = 0, 0
3
4 while True :
5     a = int(input("더할 첫 번째 수를 입력하세요 : "))
6     b = int(input("더할 두 번째 수를 입력하세요 : "))
7     hap = a + b
8     print("%d + %d = %d" % (a, b, hap))
```

4행의 무한 반복문 때문에 사용자가 Ctrl + C
를 누를 때까지 5~8행 반복

출력 결과

더할 첫 번째 수를 입력하세요 : 55

더할 두 번째 수를 입력하세요 : 22

55 + 22 = 77

더할 첫 번째 수를 입력하세요 : 77

더할 두 번째 수를 입력하세요 : 128

77 + 128 = 205

더할 첫 번째 수를 입력하세요 :

Section03 while 문

- 예 : 사용자가 Ctrl + C 를 누를 때까지 덧셈, 뺄셈, 곱셈, 나눗셈, 나머지까지 계산

Code06-11.py

```
1 ch = ""  
2 a, b = 0, 0  
3  
4 while True :  
5     a = int(input("계산할 첫 번째 수를 입력하세요 : "))  
6     b = int(input("계산할 두 번째 수를 입력하세요 : "))  
7     ch = input("계산할 연산자를 입력하세요 : ")  
8  
9     if (ch == "+") :  
10        print("%d + %d = %d" % (a, b, a + b))  
11    elif (ch == "-") :  
12        print("%d - %d = %d" % (a, b, a - b))  
13    elif (ch == "*") :  
14        print("%d * %d = %d" % (a, b, a * b))  
15    elif (ch == "/") :  
16        print("%d / %d = %5.2f" % (a, b, a / b))  
17    elif (ch == "%") :  
18        print("%d %% %d = %d" % (a, b, a % b))  
19    elif (ch == "//") :  
20        print("%d // %d = %d" % (a, b, a // b))  
21    elif (ch == "**") :  
22        print("%d ** %d = %d" % (a, b, a ** b))  
23    else :  
24        print("연산자를 잘못 입력했습니다.")
```

5~6행 : 두 숫자를 입력
7행 : 연산자를 입력

출력 결과

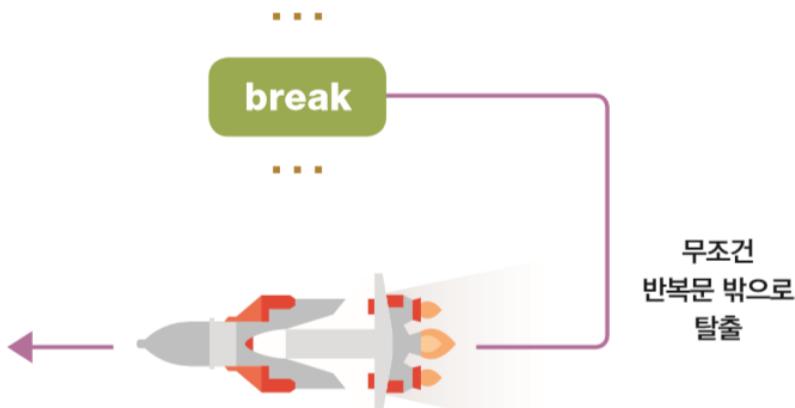
```
계산할 첫 번째 수를 입력하세요 : 22  
계산할 두 번째 수를 입력하세요 : 33  
계산할 연산자를 입력하세요 : *  
22 * 33 = 726  
계산할 첫 번째 수를 입력하세요 : 10  
계산할 두 번째 수를 입력하세요 : 4  
계산할 연산자를 입력하세요 : %  
10 % 4 = 2  
계산할 첫 번째 수를 입력하세요 :
```

Section04 break 문과 continue 문

■ 반복문을 탈출시키는 break 문

- 계속되는 반복을 끝리 적으로 빠져나가는 방법

반복문 for, while



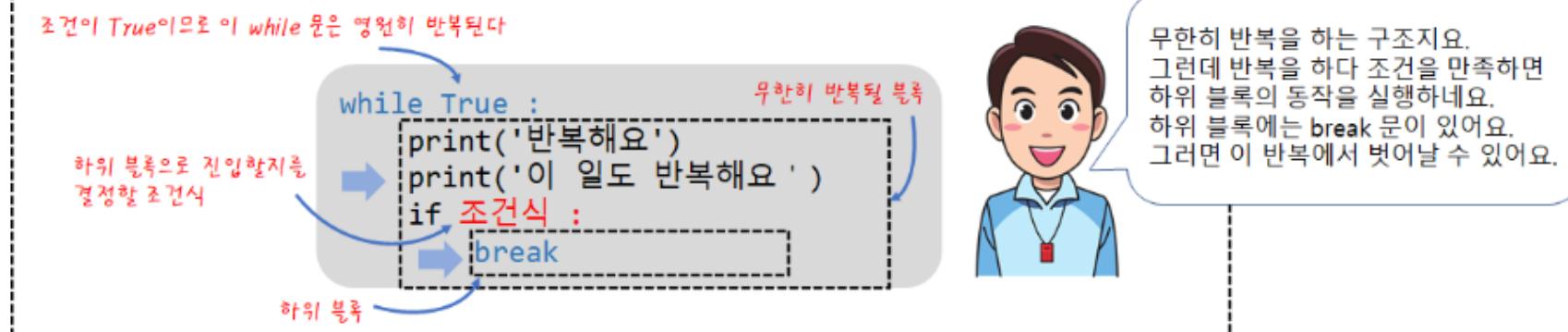
```
for i in range(1, 100):  
    print("for 문을 %d번 실행했습니다." % i)  
    break
```

출력 결과

for 문을 1번 실행했습니다.

```
while True:  
    light = input('신호등 색상을 입력하시오: ')  
    if light == 'green':  
        break  
    print('전진!!')
```

그림 6-8 break 문의 작동



Section04 break 문과 continue 문

- 예 : Code06-10.py를 break 문으로 첫 번째 수에 0이 입력될 때 자동 종료

Code06-12.py

```
1 hap = 0
2 a, b = 0, 0
3
4 while True :
5     a = int(input("더할 첫 번째 수를 입력하세요 : "))
6     if a == 0 :
7         break
8     b = int(input("더할 두 번째 수를 입력하세요 : "))
9     hap = a + b
10    print("%d + %d = %d" % (a, b, hap))
11
12 print("0을 입력해 반복문을 탈출했습니다.")
```

4행 : 무한 반복 하도록 했다
5행 : a값을 입력
6행 : 입력한 a값이 0이면 7행 실행한 후 break 문으로
 while 문을 탈출해 11행으로 건너뜀
11행에는 아무것도 없으므로 자연스럽게 12행 실행

출력 결과

더할 첫 번째 수를 입력하세요 : 55
더할 두 번째 수를 입력하세요 : 22
55 + 22 = 77
더할 첫 번째 수를 입력하세요 : 77
더할 두 번째 수를 입력하세요 : 128
77 + 128 = 205
더할 첫 번째 수를 입력하세요 : 0
0을 입력해 반복문을 탈출했습니다.

SELF STUDY 6-6

'\$'를 입력하면 while 문을 빠져나가도록 Code06-12.py를 수정해 보자.

Section04 break 문과 continue 문

- 예 : 누적 합계(hap)가 1000 이상이 되는 시작 지점 알기

Code06-13.py

```
1 hap, i = 0, 0
2
3 for i in range(1, 101) :      3행 : 값이 1부터 100까지 변경되어 100회 실행하고, hap에 i값을 누적 6행 : hap이 1000
4     hap += i                  보다 크거나 같으면 for 문을 탈출해서 8행으로
5
6     if hap >= 1000 :          9행 : 값을 출력
7         break
8
9 print("1~100의 합계를 최초로 1000이 넘게 하는 숫자 : %d" % i)
```

출력 결과

1~100의 합계를 최초로 1000이 넘게 하는 숫자 : 45

SELF STUDY 6-7

Code06-13.py를 while 문으로 변경해 보자. 출력 결과는 동일하다.

Section04 break 문과 continue 문

■ 반복문으로 다시 돌아가게 하는 continue 문



그림 6-9 continue 문의 작동

- 예: 1~100의 합계를 구하되, 3의 배수 (제외하고) 더하기

Code06-14.py

```
1 hap, i = 0, 0
2
3 for i in range(1, 101) :
4     if i % 3 == 0 :
5         continue
6
7     hap += i
8
9 print("1~100의 합계(3의 배수 제외) : %d" % hap)
```

출력 결과

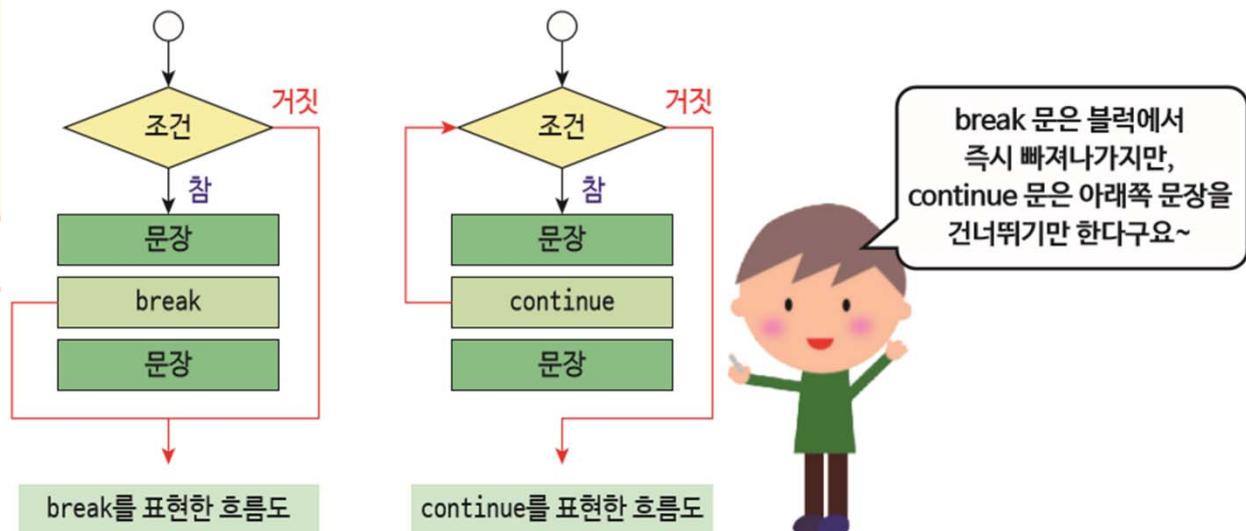
1~100의 합계(3의 배수 제외) : 3367

Section04 break 문과 continue 문

■ 반복문으로 다시 돌아가게 하는 continue 문

```
st = 'I love Python Programming' # 출력을 위한 문자열
for ch in st:
    if ch in ['a','e','i','o','u', 'A','E','I','O','U']:
        continue      # 모음일 경우 아래 출력을 건너뛴다
    print(ch, end='')
```

Iv Python Prgrmmng



잠깐 – continue와 break를 너무 많이 쓰면 코드를 읽기 어려워진다

break와 continue는 프로그램의 제어를 효율적으로 하는데 편리하게 사용할 수 있다. 하지만, break와 continue 문이 너무 많이 사용되는 경우 제어의 흐름에 일관성이 없어 프로그램을 이해하는 것이 어려워진다. 따라서 continue와 break는 필요한 경우에만 제한적으로 사용하는 것이 좋다.

Section04 break 문과 continue 문

■ [프로그램 2]의 완성

- While 문으로 구현

Code06-15.py

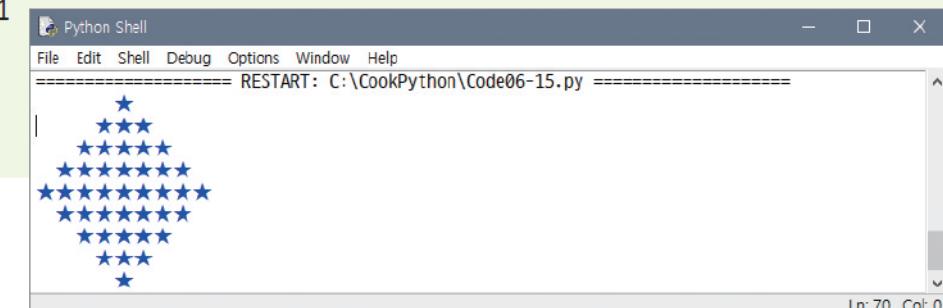
```
1 ## 전역 변수 선언 부분 #
2 i, k = 0, 0
3
4 ## 메인 코드 부분 #
5 i = 0
6 while i < 9 :
7     if i < 5 :
8         k = 0
9         while k < 4 - i :
10            print(' ', end = '')
11            k += 1
12         k = 0
13         while k < i * 2 + 1 :
14            print('\u2605', end = '')
```

```
15             k += 1
16     else :
17         k = 0
18         while k < i - 4 :
19            print(' ', end = '')
20            k += 1
21         k = 0
22         while k < (9 - i) * 2 - 1 :
23            print('\u2605', end = '')
24            k += 1
25     print()
26     i += 1
```

```
print('\u2605')
```

- 별 모양의 글자 출력하는 코드

```
print('\u2605')
```



6~26행 : 9번 반복되어 출력 줄이 9개 표시

다섯 줄은 7~15행이 출력하고, 나머지 네 줄은 16~24행이 출력

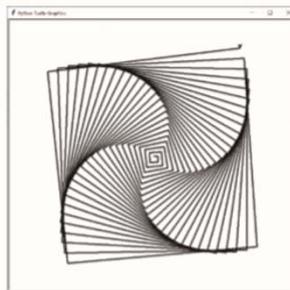
각 줄이 출력될 때 공백과 별의 개수는 9행, 13행, 18행, 22행이 결정

Section04 break 문과 continue 문

LAB^{5~9} 간단한 코드로 멋진 나선형 도형 그려보기

반복문과 터틀 그래픽을 결합하면 상당히 복잡한 형상을 쉽게 그릴 수 있다. 예를 들어서 다음과 같은 그림을 그릴 수 있다. 화면에 사각형을 그리는 것이지만 한번 반복할 때마다 각도가 90도로 회전하는 것이 아니고 89도로 하면 약간씩 회전되는 사각형을 그리는 것이 가능하다. 많은 사각형을 반복해서 그리면 위와 같은 그림이 얻어진다. 이와 같은 패턴을 만드는 그림을 그리는 프로그램을 작성해 보자. 이때 가장 짧은 선분의 길이는 10으로 하고, 가장 긴 것은 500보다 작게 하자.

원하는 결과



```
import turtle  
  
t = turtle.Turtle()  
# 거북이의 속도는 0으로 설정하면 최대가 된다.  
t.speed(0)  
t.width(3)  
  
length = 10 # 초기 선의 길이는 10으로 한다.  
# normal 반복문이다. 선의 길이가 500보다 작으면 반복한다.  
while length < 500:  
    t.forward(length) # length만큼 전진한다.  
    t.right(89) # 89도 오른쪽으로 회전한다.  
    length += 5 # 선의 길이를 5만큼 증가시킨다.
```



도전문제 5.9

각 반복에서 거북이가 회전하는 각도를 약간 다르게 해서 결과가 어떻게 달라지는지 본다.

Section04 break 문과 continue 문

LAB⁵⁻¹¹ 무한 반복문으로 숫자 맞추기 게임

사용자가 답을 제시하면 프로그램은 자신이 저장한 정수와 비교하여 제시된 정수가 더 높은지 낮은지 만을 알려준다. 정수의 범위를 1부터 100까지로 한정하면 최대 7번이면 누구나 알아맞힐 수 있다. 정수의 범위를 1부터 1,000,000까지 확대하더라도 최대 20번이면 맞출 수 있다. 왜 그럴까? 이진 탐색의 원리 때문이다. 중간값과 한 번씩 비교할 때마다 탐색의 범위는 1/2로 대폭 줄어든다. 게임이 끝나면 몇 번 만에 맞추었는지도 함께 출력한다. 다음의 경우 87이 정답이지만 매번 임의로 생성된 수가 정답이 될 수 있음에 유의하자.

원하는 결과

```
1부터 100 사이의 숫자를 맞추시오
숫자를 입력하시오: 50
낮음!
숫자를 입력하시오: 86
낮음!
숫자를 입력하시오: 87
축하합니다. 총 시도횟수 = 3
```



```
import random

tries = 0
guess = 0
answer = random.randint(1, 100)
print("1부터 100 사이의 숫자를 맞추시오")
while guess != answer:
    guess = int(input("숫자를 입력하시오: "))
    tries = tries + 1
    if guess < answer:
        print("낮음!")
    elif guess > answer:
        print("높음!")

print("축하합니다. 총 시도횟수=", tries)
```



도전문제 5.10

시도 횟수를 최대 10번으로 제한하려면 위의 프로그램을 어떻게 변경하여야 하는가?

코드의 오류를 처리하는 방법

자료출처 : 데이터 과학을 위한 파이썬 프로그래밍 (한빛아카데미) – 04_조건문과 반복문 강의교안

Section05 코드의 오류를 처리하는 방법

■ 버그와 디버그

- 버그(bug) : 프로그래밍에서의 오류
- 디버그(debug) : 오류를 수정하는 과정
- 디버깅(debugging) : 코드에서 오류를 만났을 때, 프로그램의 잘못을 찾아내고 고치는 것

■ 오류의 종류와 해결 방법 : 문법적 오류

- 문법적 오류는 코딩했을 때, 인터프리터가 해석을 못 해 코드 자체를 실행시키지 못하는 오류이다. 문법적 오류는 비교적 쉬운 유형의 오류이며, 대표적으로 들여쓰기 오류와 오탈자로 인한 오류가 있다.

Section05 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법 : 문법적 오류

- 들여쓰기 오류(indentation error)

코드 4-22 indentation.py

```
1 x = 2
2 y = 5
3 print(x + y)
```

[1]:

```
x = 2
y = 5
print(x+y)
```

```
File "C:\Users\sky\AppData\Local\Temp\ipykernel_30416\3606719476.p
y", line 2
    y = 5
    ^
IndentationError: unexpected indent
```



여기서 잠깐! 들여쓰기 오류가 발생하는 이유

- 〈Space〉 키나 〈Tab〉 키로 들여쓰기를 하면 들여쓰기 오류가 자주 발생한다. 초기 파일로 코딩할 때는 들여쓰기를 4 스페이스(4 space)로 하는 사람과 〈Tab〉 키로 하는 사람이 각각 따로 있어 함께 코딩하면 많은 문제가 발생하였다. 하지만 최근에는 이러한 문제가 많이 줄었다

Section05 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법 : 문법적 오류

- 오탈자로 인한 오류

코드 4-23 name.py

```
1 pront (x + y)      # Print가 아닌 Pront로 작성
2 korean = "ACE"
3 print(Korean)       # K는 소문자
```

```
pront(x+y)
korean = "ABC"
print(Korean)
```

```
-----
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_30416\1957022412.py in <module>
----> 1 pront(x+y)
      2 korean = "ABC"
      3 print(Korean)

NameError: name 'pront' is not defined
```

■ 오류의 종류와 해결 방법 : 논리적 오류

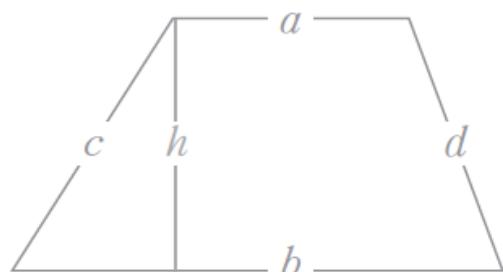
- 프로그램을 작성하다 보면 코드를 제대로 작성했다고 생각했음에도, 원하는 결과가 나오지 않는 경우가 종종 있다.
- 논리적 오류를 해결하는 방법은 다양한데, 당장 쉽게 사용할 수 있는 방법은 확인이 필요한 변수들에 print() 함수를 사용하여 값을 확인하는 것이다.

Section05 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법

- 사다리꼴 넓이를 구하는 [프로그램을 작성하면서, 이 논리적 오류를 해결하는 연습을 해 보자.](#)

$$A = \frac{a + b}{2} h \quad (a: \text{윗변}, b: \text{아랫변}, h: \text{높이})$$



[사다리꼴의 넓이 구하기]

- 사다리꼴의 넓이를 구하는 공식은 ‘{(밑변+윗변)/2} * 높이’이다. 이 수식에 있는 각각의 과정을 하나씩 함수로 만들어 변환하는 연습을 할 것이다. 첫 번째는 두 변수를 더하는 addition() 함수, 두 번째는 두 값을 곱하는 multiplication() 함수, 세 번째는 2로 나누는 divided() 함수이다.

코드 4-24 trapezium_def.py

```
1 def addition(x, y):  
2     return x + y  
3  
4 def multiplication(x, y):  
5     return x * y  
6  
7 def divided_by_2(x):  
8     return x / 2
```

Section05 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법

- ① 함수가 잘 작동하는지 확인하는 방법 : 파일 셀에서 실행하기

```
>>> import trapezium_def as ta          # trapezium_def 파일을 ta라는 이름으로 부름
>>> ta.addition(10, 5)
15
>>> ta.multiplication(10, 5)
50
>>> ta.divided_by_2(50)
25.0
```

jupyter lab

```
def addition(x,y):
    return x+y

if __name__ == '__main__':
    print(addition(10,5))
```

15

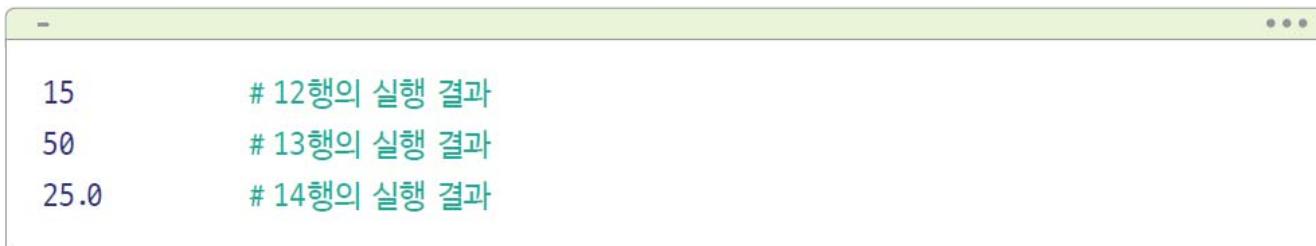
- ② 함수가 잘 작동하는지 확인하는 방법 : 파일에서 체크할 수 있도록 if `_name_ == "__main__"`을 써 주는 구조로, if 문 안에 테스트할 코드를 작성하기

코드 4-25 trapezium_area.py

```
1 def addition(x, y):
2     return x + y
3
4 def multiplication(x, y):
5     return x * y
6
7 def divided_by_2(x):
8     return x / 2
9
10 # 파일 셀에서 호출할 경우 실행되지 않음
11 if __name__ == '__main__':
12     print(addition(10, 5))
13     print(multiplication(10, 5))
14     print(divided_by_2(50))
```

Section05 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법



```
15      # 12행의 실행 결과  
50      # 13행의 실행 결과  
25.0    # 14행의 실행 결과
```

→ if `__name__ == '__main__'`을 넣는 가장 큰 이유는 해당 파일을 파이썬 셀에서 불러올 import 때 함수 안에 들어 있지 않은 코드들이 작동되지 않게 하기 위해서이다. 만약 해당 구문 없이 `print()` 함수로 구문을 작성한다면, 해당 파일을 파이썬 셀에서 호출할 때 그 구문이 화면에 출력되는 것을 확인할 수 있다.

따라서 어떤 것을 테스트하기 위해서는 반드시 `if __name__ == '__main__'` 안에 코드를 넣는 것이 좋다.

모듈과 패키지에서 다를 예정임. !!!

Section05 코드의 오류를 처리하는 방법

■ 오류의 종류와 해결 방법

- 실제 사다리꼴의 넓이 구하기 프로그램을 작성

코드 4-26 trapezium_area_final.py

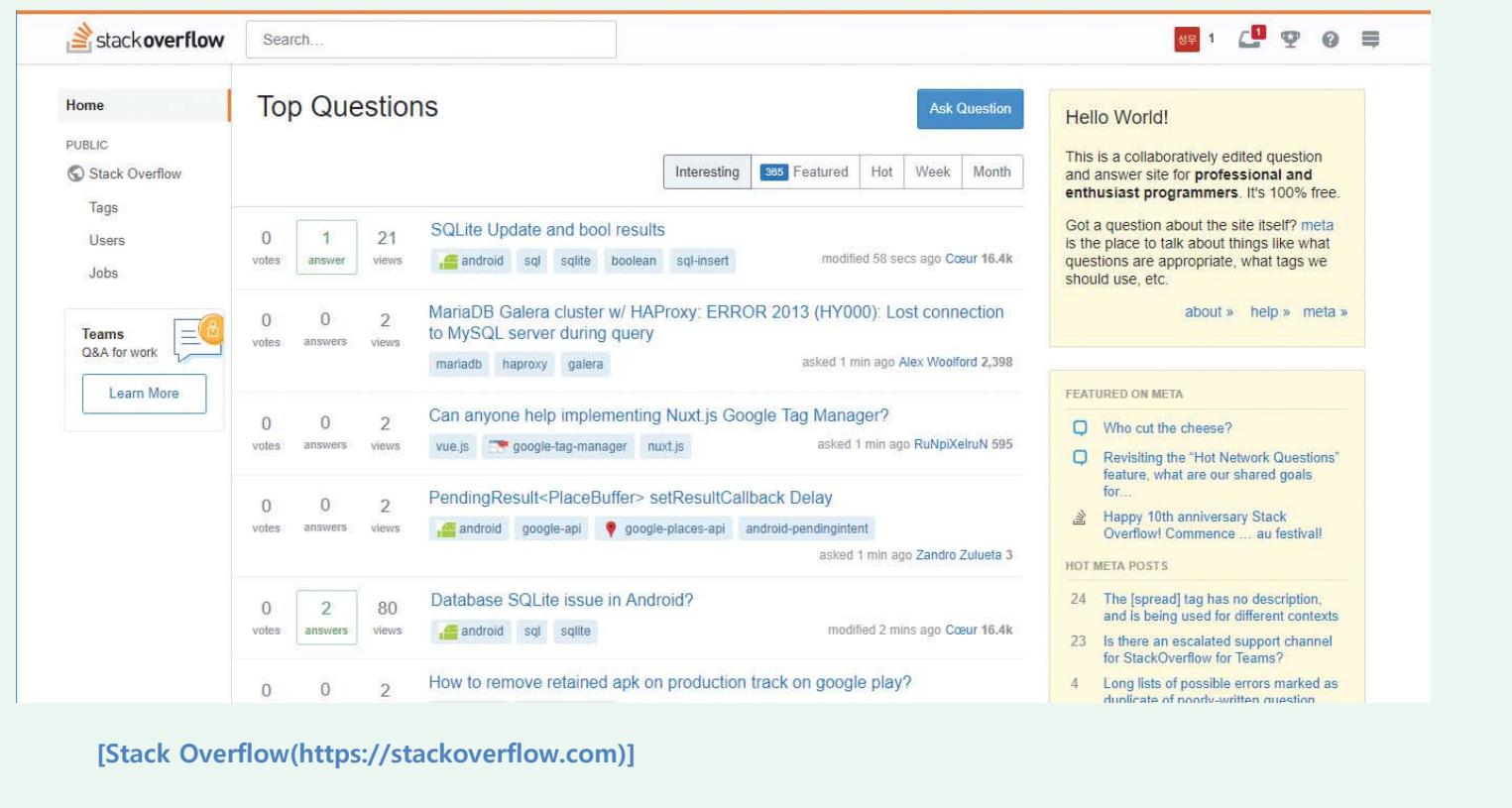
```
1 def addition(x, y):
2     return x + y
3
4 def divided_by_2(x):
5     return x / 2
6
7 def main():
8     base_line = float(input("밑변의 길이는? "))
9     upper_edge = float(input("윗변의 길이는? "))
10    height = float(input("높이는? "))
11
12    print("넓이는:", divided_by_2(addition(base_line, upper_edge) * height))
13
14 if __name__ == '__main__':
15     main()
```



Section05 코드의 오류를 처리하는 방법

여기서 잠깐! 오류를 스스로 해결하기

- 많은 사람이 구글과 프로그래밍계의 지식인인 Stack Overflow를 통해 문제를 해결하고 있다.
Stack Overflow는 전 세계 개발자들이 사용하는 Q&A 사이트이다.



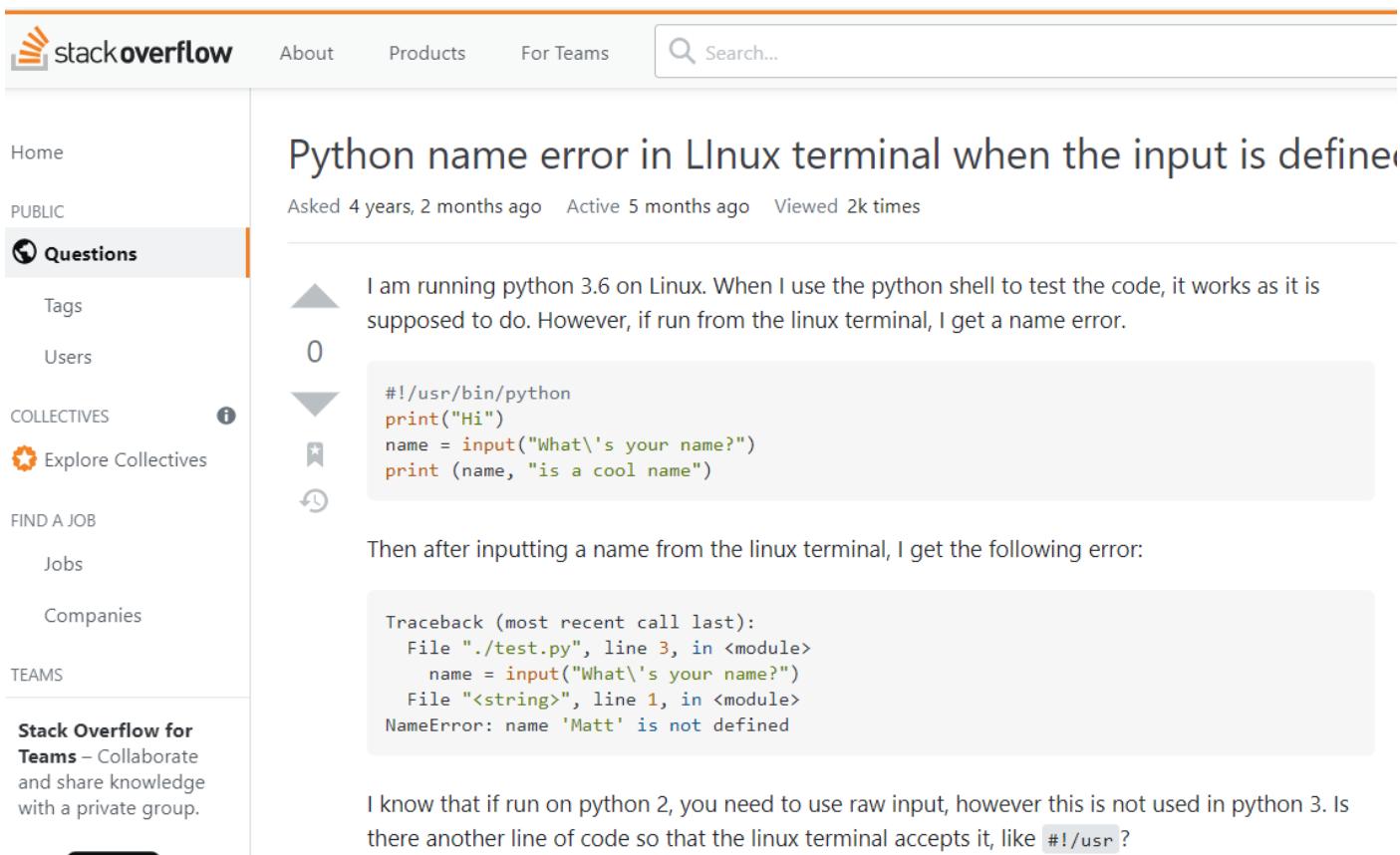
The screenshot shows the Stack Overflow homepage with the following details:

- Left Sidebar:** Home, PUBLIC, Stack Overflow, Tags, Users, Jobs, Teams (Q&A for work), Learn More.
- Header:** stackoverflow, Search bar, Notifications (1), Help, Logout.
- Top Questions:** A list of five questions with their titles, votes, answers, views, tags, and last activity.
 - SQLite Update and bool results (1 answer, 21 views)
 - MariaDB Galera cluster w/ HAProxy: ERROR 2013 (HY000): Lost connection to MySQL server during query (0 answers, 2 views)
 - Can anyone help implementing Nuxt.js Google Tag Manager? (0 answers, 2 views)
 - PendingResult<PlaceBuffer> setResultCallback Delay (0 answers, 2 views)
 - Database SQLite issue in Android? (2 answers, 80 views)
 - How to remove retained apk on production track on google play? (0 answers, 2 views)
- Right Sidebar:**
 - Hello World!**: A collaboratively edited question about the site itself.
 - FEATURED ON META**: Posts from the Stack Overflow Meta site.
 - HOT META POSTS**: Top posts from the Stack Overflow Meta site.

[Stack Overflow(<https://stackoverflow.com>)]

Section05 코드의 오류를 처리하는 방법

여기서  잠깐! 오류를 스스로 해결하기



The screenshot shows a Stack Overflow question page. The left sidebar has navigation links: Home, PUBLIC, Questions (which is selected), Tags, Users, COLLECTIVES, Explore Collectives, FIND A JOB, Jobs, Companies, and TEAMS. A note for Teams says: "Stack Overflow for Teams – Collaborate and share knowledge with a private group." The main content area shows a question titled "Python name error in Llunx terminal when the input is defined". It was asked 4 years, 2 months ago, last active 5 months ago, and viewed 2k times. The question text is: "I am running python 3.6 on Linux. When I use the python shell to test the code, it works as it is supposed to do. However, if run from the linux terminal, I get a name error." Below the question text is a code block:

```
#!/usr/bin/python
print("Hi")
name = input("What's your name?")
print (name, "is a cool name")
```

Underneath the code, it says: "Then after inputting a name from the linux terminal, I get the following error:" followed by a traceback:

```
Traceback (most recent call last):
  File "./test.py", line 3, in <module>
    name = input("What's your name?")
  File "<string>", line 1, in <module>
NameError: name 'Matt' is not defined
```

At the bottom, there is a comment: "I know that if run on python 2, you need to use raw input, however this is not used in python 3. Is there another line of code so that the linux terminal accepts it, like `#!/usr` ?"