

# 스크립트 프로그래밍

## 05 루프

2016 2학기 (02분반)

강승우

# 학습 목표

- while 루프를 사용하여 반복적 명령문을 실행하는 프로그램을 작성할 수 있다 (5.2절).
- 루프 설계 전략을 따라 루프를 개발할 수 있다(5.2.1~5.2.3 절).
- 사용자 확인을 통해 루프를 제어할 수 있다(5.2.4 절).
- 감시 값을 이용하여 루프를 제어할 수 있다(5.2.5절).
- 직접 키보드 입력 대신 입력 재지정을 이용하여 파일로부터 많은 양의 데이터를 입력 받고, 출력 재지정을 사용하여 출력 데이터를 파일에 저장할 수 있다(5.2.6절).
- for 루프를 사용하여 계수 제어 루프를 구현할 수 있다(5.3절).
- 중첩 루프를 작성할 수 있다(5.4절).
- 수치적 오류를 최소화하는 기법을 이해할 수 있다(5.5절).
- 다양한 예제를 통해 루프를 이해할 수 있다(5.6, 5.8 절).
- break 문과 continue 문으로 제어하는 프로그램을 구현할 수 있다(5.7절).

# 루프 (반복문)

- 동일한 연산을 반복하여 수행해야 할 경우 코드를 중복하여 쓰지 않고 루프를 이용하여 처리
- 예
  - 하나의 문자열 '프로그래밍은 재미있습니다!'를 100번 출력해야 할 때

100번 {

```
print("프로그래밍은 재미있습니다!")
print("프로그래밍은 재미있습니다!")
print("프로그래밍은 재미있습니다!")
print("프로그래밍은 재미있습니다!")
print("프로그래밍은 재미있습니다!")
...
print("프로그래밍은 재미있습니다!")
print("프로그래밍은 재미있습니다!")
print("프로그래밍은 재미있습니다!")
```

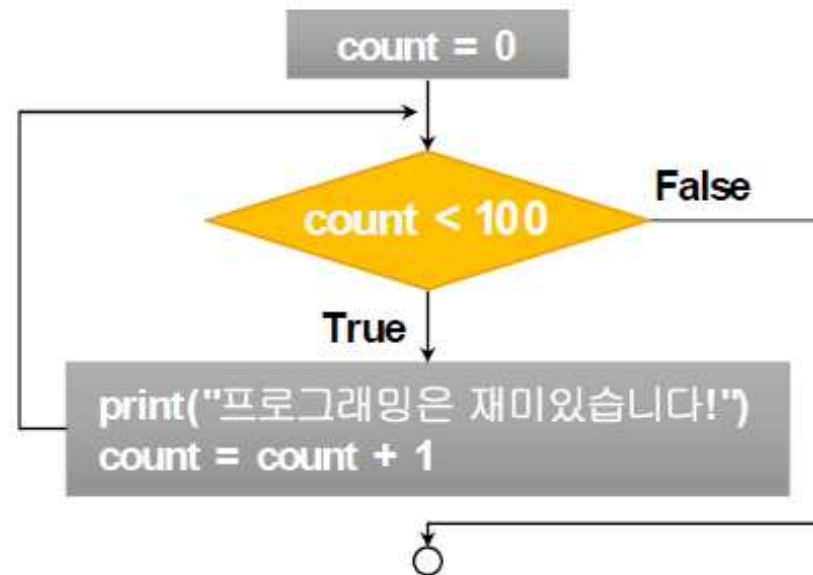
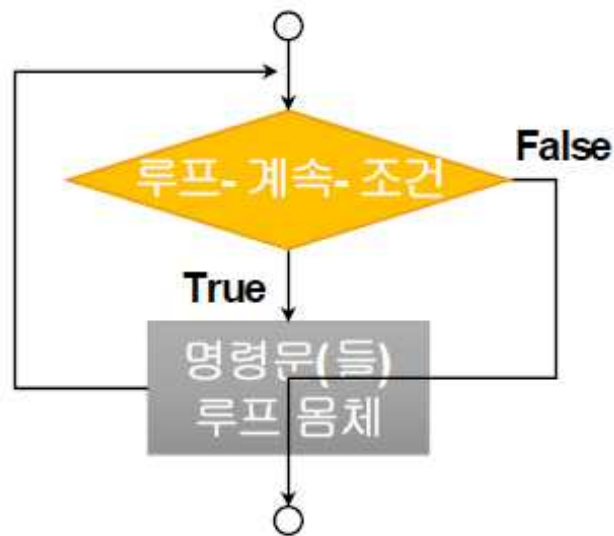
```
count = 0
while count < 100:
    print("프로그래밍은 재미있습니다!")
    count = count + 1
```

# while 루프

## 구문형식

**while** 루프-계속-조건:  
# 루프 몸체  
명령문(들)

```
count = 0
while count < 100:
    print("프로그래밍은 재미있습니다!")
    count = count + 1
```



# while 루프 트레이스

count 초기화

```
count = 0
```

```
while count < 2:
```

```
    print("프로그래밍은 재미있습니다!")
```

```
    count = count + 1
```

# while 루프 트레이스

(count < 2)는 true

```
count = 0
while count < 2:
    print("프로그래밍은 재미있습니다!")
    count = count + 1
```

# while 루프 트레이스

```
count = 0  
while count < 2:
```

```
    print("프로그래밍은 재미있습니다!")
```

```
    count = count + 1
```

프로그래밍은 재미있습니다! 출력

# while 루프 트레이스

```
count = 0
while count < 2:
    print("프로그래밍은 재미있습니다!")
    count = count + 1
```

count를 1 증가  
count는 이제 1



# while 루프 트레이스

count는 1이기 때문에  
(count < 2)는 아직 true

```
count = 0
while count < 2:
    print("프로그래밍은 재미있습니다!")
    count = count + 1
```

# while 루프 트레이스

```
count = 0  
while count < 2:
```

```
    print("프로그래밍은 재미있습니다!")
```

```
    count = count + 1
```

프로그래밍은 재미있습니다! 출력

# while 루프 트레이스

```
count = 0
while count < 2:
    print("프로그래밍은 재미있습니다!")
    count = count + 1
```

count를 1 증가  
count는 이제 2

# while 루프 트레이스

count는 2이기 때문에  
(count < 2)는 이제 false

```
count = 0
while count < 2:
    print("프로그래밍은 재미있습니다!")
    count = count + 1
```

# while 루프 트레이스

```
count = 0
while count < 2:
    print("프로그래밍은 재미있습니다!")
    count = count + 1
```

루프가 종료  
루프 이후의 다음 명령문을  
실행

# 뺨셈 퀴즈

- 코드 4.4 뺨셈 퀴즈
  - 뺨셈 문제에 대한 답을 사용자로부터 입력 받아 정답 여부를 출력
  - 한 번의 입력만 받을 수 있었음
- 정확한 입력을 받을 때까지 새로운 값을 입력 받게 하는 프로그램으로 작성해보자

# 루프 설계 전략

- 단계 1
  - 반복되어야 하는 명령문을 파악한다
- 단계 2
  - 다음과 같이 파악된 명령문을 루프로 묶는다  
while True:  
    명령문
- 단계 3
  - 루프-계속-조건 코드를 작성하고 루프를 제어하기 위한 적절한 명령문을 추가한다  
while 루프-계속-조건:  
    명령문  
    루프를 제어하기 위한 추가적인 명령문

# 사례 연구: 숫자 맞추기

- 컴퓨터가 임의로 생성한 숫자를 맞추는 게임
  - 0 이상 100이하 사이 정수를 랜덤하게 생성
  - 생성된 숫자가 사용자의 입력 숫자와 일치할 때까지 계속 입력 받음
  - 각 사용자 입력에 대해, 생성 숫자보다 큰지, 작은지 응답하여 사용자가 지능적으로 다음 숫자 입력을 할 수 있게 함



# 사례 연구: 숫자 맞추기 – 실행 예제

```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>>
RESTART: D:\WGitRepo\ScriptProgramming_2016-2\Wch5\GuessNumber.py
0과 100 사이의 마법수를 맞춰보세요.
마법수는 무엇일까요?: 50
너무 큼니다.
마법수는 무엇일까요?: 25
너무 큼니다.
마법수는 무엇일까요?: 11
너무 작습니다.
마법수는 무엇일까요?: 19
너무 작습니다.
마법수는 무엇일까요?: 22
너무 큼니다.
마법수는 무엇일까요?: 20
너무 작습니다.
마법수는 무엇일까요?: 21
정답, 마법수는 21 입니다.
>>> |
```

Ln: 41 Col: 4

# 사례 연구: 숫자 맞추기

- 문제 해결을 위한 과정
  - 0-100 사이 정수를 생성
  - 사용자로부터 추측값 입력 받음
  - 생성 숫자와 사용자 추측값을 비교하여 결과 출력

```
import random
# 사용자가 맞춰야하는 마법수를 생성한다.
number = random.randint(1, 100)
print("0과 100 사이의 마법수를 맞춰보세요.")

# 사용자로부터 추측값을 입력받는다.
guess = eval(input("마법수는 무엇일까요?: "))

if guess == number:
    print("정답, 마법수는", number, "입니다.")
elif guess > number:
    print("너무 큼니다.")
else:
    print("너무 작습니다.")
```

# 사례 연구: 숫자 맞추기

- 문제 해결을 위한 과정
  - 반복이 필요한 명령문 파악
  - 루프 적용

```
import random
# 사용자가 맞춰야하는 마법수를 생성한다.
number = random.randint(1, 100)
print("0과 100 사이의 마법수를 맞춰보세요.")

while True:
    # 사용자로부터 추측값을 입력받는다.
    guess = eval(input("마법수는 무엇일까요?: "))

    if guess == number:
        print("정답, 마법수는", number, "입니다.")
    elif guess > number:
        print("너무 큼니다.")
    else:
        print("너무 작습니다.")
```

# 사례 연구: 숫자 맞추기

- 문제 해결을 위한 과정
  - 루프-계속-조건 작성
  - 루프 제어를 위한 추가 명령문

```
import random
# 사용자가 맞춰야하는 마법수를 생성한다.
number = random.randint(1, 100)
print("0과 100 사이의 마법수를 맞춰보세요.")

guess = -1
while guess != number:
    # 사용자로부터 추측값을 입력받는다.
    guess = eval(input("마법수는 무엇일까요?: "))

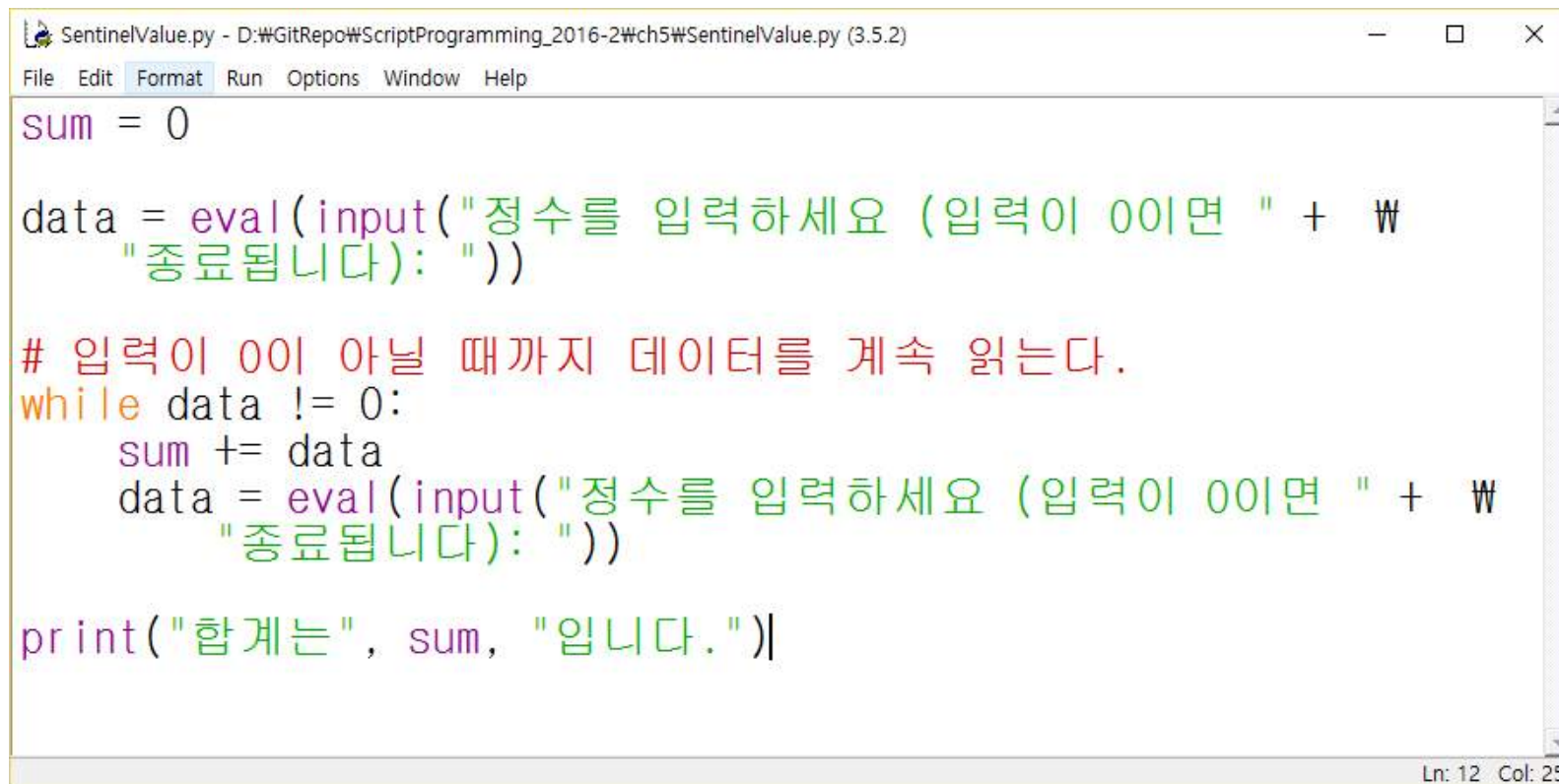
    if guess == number:
        print("정답, 마법수는", number, "입니다.")
    elif guess > number:
        print("너무 큼니다.")
    else:
        print("너무 작습니다.")
```

# 감시값을 사용하여 루프 제어하기

- 루프의 반복 횟수가 사전에 정해지지 않는 경우
  - 감시값을 사용하여 반복 여부를 결정
  - 감시값(Sentinel value): 반복의 끝을 의미하는 입력값
  - 감시값을 사용하는 루프를 감시-제어 루프(sentinel-controlled loop)라고 함

# 감시-제어 루프 예제

- 불특정 개수의 정수를 읽고 합계를 계산하는 프로그램

A screenshot of a Python IDE window titled "SentinelValue.py - D:\GitRepo\ScriptProgramming\_2016-2\ch5\SentinelValue.py (3.5.2)". The window has a menu bar with "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The code editor contains the following Python code:

```
sum = 0

data = eval(input("정수를 입력하세요 (입력이 0이면 " + W
               "종료됩니다): "))

# 입력이 0이 아닐 때까지 데이터를 계속 읽는다.
while data != 0:
    sum += data
    data = eval(input("정수를 입력하세요 (입력이 0이면 " + W
                     "종료됩니다): "))

print("합계는", sum, "입니다.")
```

The status bar at the bottom right shows "Ln: 12 Col: 25".



## 주의

- 루프 제어에서 동등 검사에 부동소수점 값을 사용하면 문제가 발생한다. 부동소수점 값은 근사치이기 때문에 부정확한 계수 값으로 이어질 수 있다. 이번 예제에서는 data는 정수값으로 사용되었다.  $1 + 0.9 + 0.8 + \dots + 0.1$ 을 계산하는 다음의 코드를 생각해 보자.

```
item = 1
sum = 0
while item != 0: # item이 0이 될 것 이라고 보장할 수 없다
    sum += item
    item -= 0.1
print(sum)
```

- 변수 item은 1에서 시작하고 루프 몸체가 실행될 때마다 0.1씩 감소한다. 루프는 item이 0이 될 때 종료되어야 한다. 그러나 부동소수점 연산은 근사치를 사용하기 때문에 item이 정확하게 0이 될 것이라고 보장할 수 없다. 이 루프는 겉으로 문제가 없는 것처럼 보이지만 실제로 무한루프이다.

# for 루프

- 루프 몸체의 실행이 몇 번 반복되어야 하는지 그 횟수를 정확히 알고 있을 때

## 구문형식

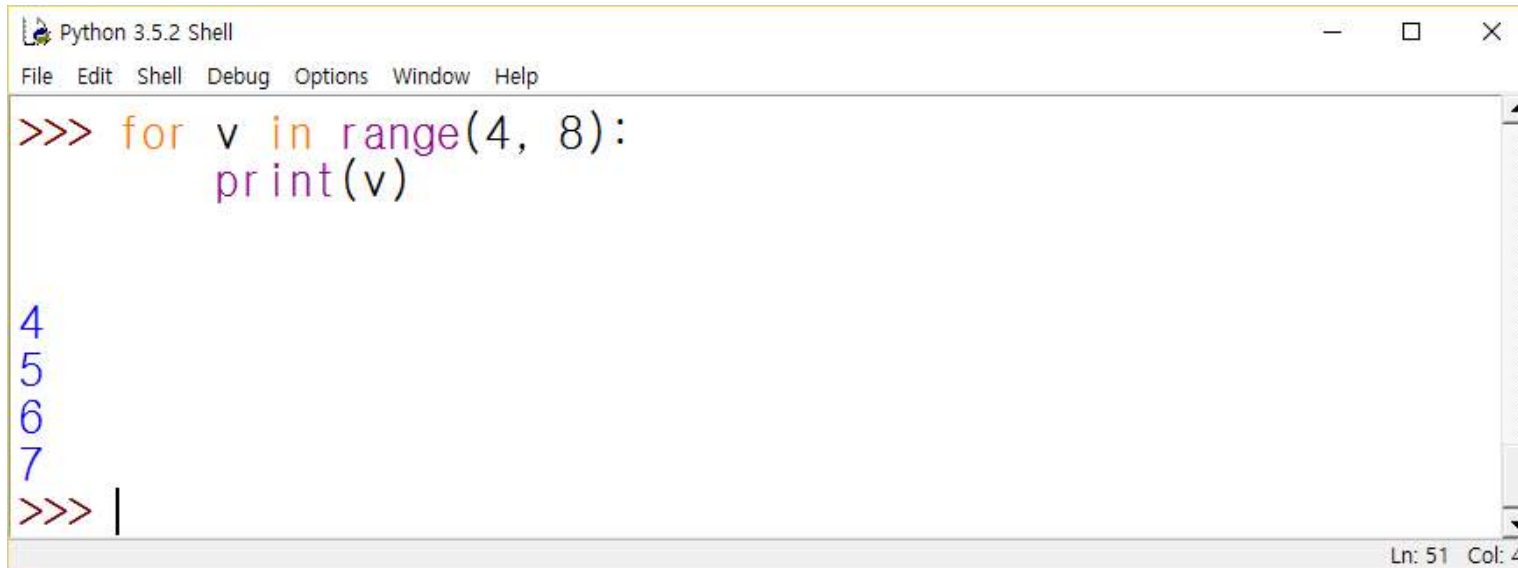
```
for i in range(초깃값, 종료값):  
    # 루프 몸체
```

```
i = initialValue # 루프 제어 변수를 초기화한다.  
while i < endValue:  
    # 루프 몸체  
    ...  
    i++ # 루프 제어 변수를 조정한다.
```

이와 같이 while 루프를 사용하여 동일한 반복 수행을 할 수 있지만 for 루프를 사용하면 더 단순하게 할 수 있음



# for 루프 - range



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> for v in range(4, 8):
        print(v)
4
5
6
7
>>> |
```

Ln: 51 Col: 4

- range(a, b): a, b는 반드시 정수, 연속된 정수 a부터 b-1까지
- range(a): range(0, a)와 동일, 0부터 a-1까지
- range(a, b, step)
  - step이 양수: a부터 step만큼 증가되어 b보다 작은 최대 정수까지
  - step이 음수: a에서 감소하여 b보다 큰 최소 정수까지

# 중첩 루프

- 한 개의 외부 루프(outer loop)와 한 개 이상의 내부 루프(inner loop)로 구성
- 외부 루프가 반복될 때마다 내부 루프는 재진입되고 새롭게 시작됨
- 예제
  - 구구단 표 출력 프로그램: MultiplicationTable.py

# 사례 연구: 최대공약수 찾기

- 문제

- 사용자로부터 두 양수를 입력받고 두 수의 최대공약수를 찾는 프로그램을 작성하시오

- 최대 공약수

- 4와 2의 최대공약수(GCD: Greatest Common Divisor)는 2
- 16과 24의 최대공약수는 8

- 약수?

# 사례 연구: 최대공약수 찾기

- 입력된 두 정수를  $n_1$ 과  $n_2$ 라고 하자. 1이 공약수라는 것은 당연하지만 최대 공약수가 아닐 수도 있다. 따라서  $k$ ( $k$ 는 2, 3, 4 등)가  $n_1$ 과  $n_2$ 의 공약수인지 아닌지를  $n_1$ 과  $n_2$ 보다 커지지 않을 때까지 검사해야 한다.
- 알고리즘
  - gcd라는 이름의 변수에 공약수를 저장
  - 초기에 gcd는 1
  - 새로운 공약수를 찾을 때마다, 이 공약수는 gcd에 저장
  - 2에서 시작하여 최대  $n_1$  또는  $n_2$ 까지 모든 가능한 공약수에 대한 검사가 끝나면 변수 gcd의 값은 최대공약수

# break와 continue 키워드

- 루프 명령문에 추가적인 제어를 하기 위해 사용
- break
  - 루프를 즉시 종료
- continue
  - 현재의 반복은 종료하고 루프 몸체의 맨 끝으로 이동
- 코드를 간략화하고 프로그램을 읽기 쉽게 만들 수 있는 경우 사용

# break

루프부터  
빠져나온다

```
sum = 0
number = 0

while number < 20:
    number += 1
    sum += number
    if sum >= 100:
        break
    print("마지막 숫자는", number, "입니다.")
print("합계는", sum, "입니다.")
```

# continue

반복의 맨  
끝으로 건너  
뛴다

```
sum = 0
number = 0

while (number < 20):
    number += 1
    if (number == 10 or number == 11):
        continue
    sum += number

print("The sum is ", sum)
```

# 사례 연구: 소수 출력하기

- 소수 (prime number)
  - 1보다 큰 정수 중 1과 자기 자신으로만 나누어 떨어지는 수
  - 2, 3, 5는 소수, 4, 6, 8, 9는 소수 아님
- 첫 50개의 소수를 한 행에 10개씩 5개의 행에 출력하는 프로그램
  - 주어진 수(number)가 소수인지 결정
    - 2, 3, 4, 5, 6, .. 에 대하여 number가 나누어지는지 검사
    - 최대  $\text{number}/2$ 의 수까지
    - 만약 나누어지면 소수가 아님
  - 소수의 개수를 셈
  - 각 소수를 출력하고 한 행에 10개씩 숫자가 들어가도록 출력
  - 50개의 소수를 출력했으면 종료