

루프와 반복문

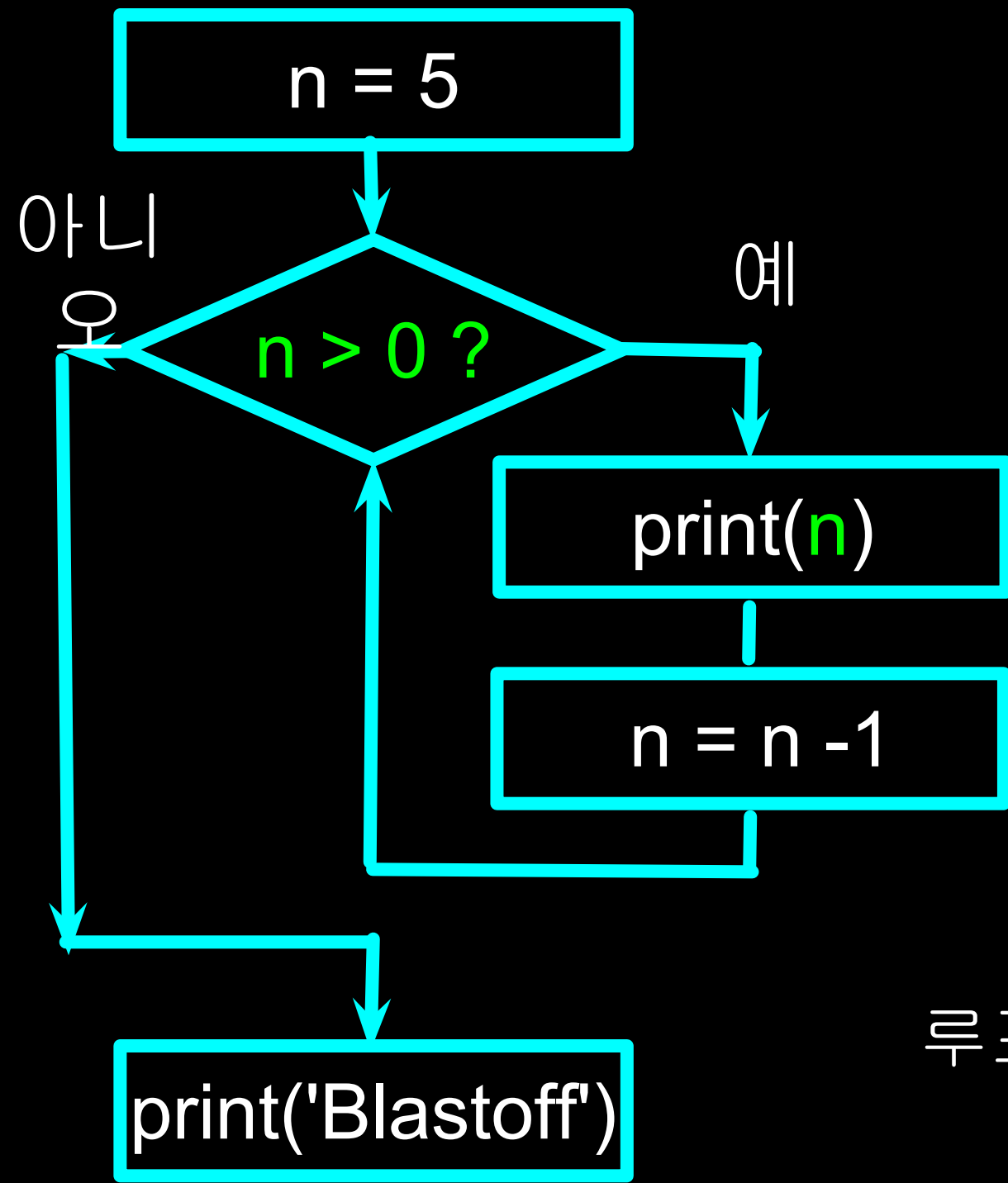
제5장



Python for Everybody
www.py4e.com



반복 단계



프로그램:

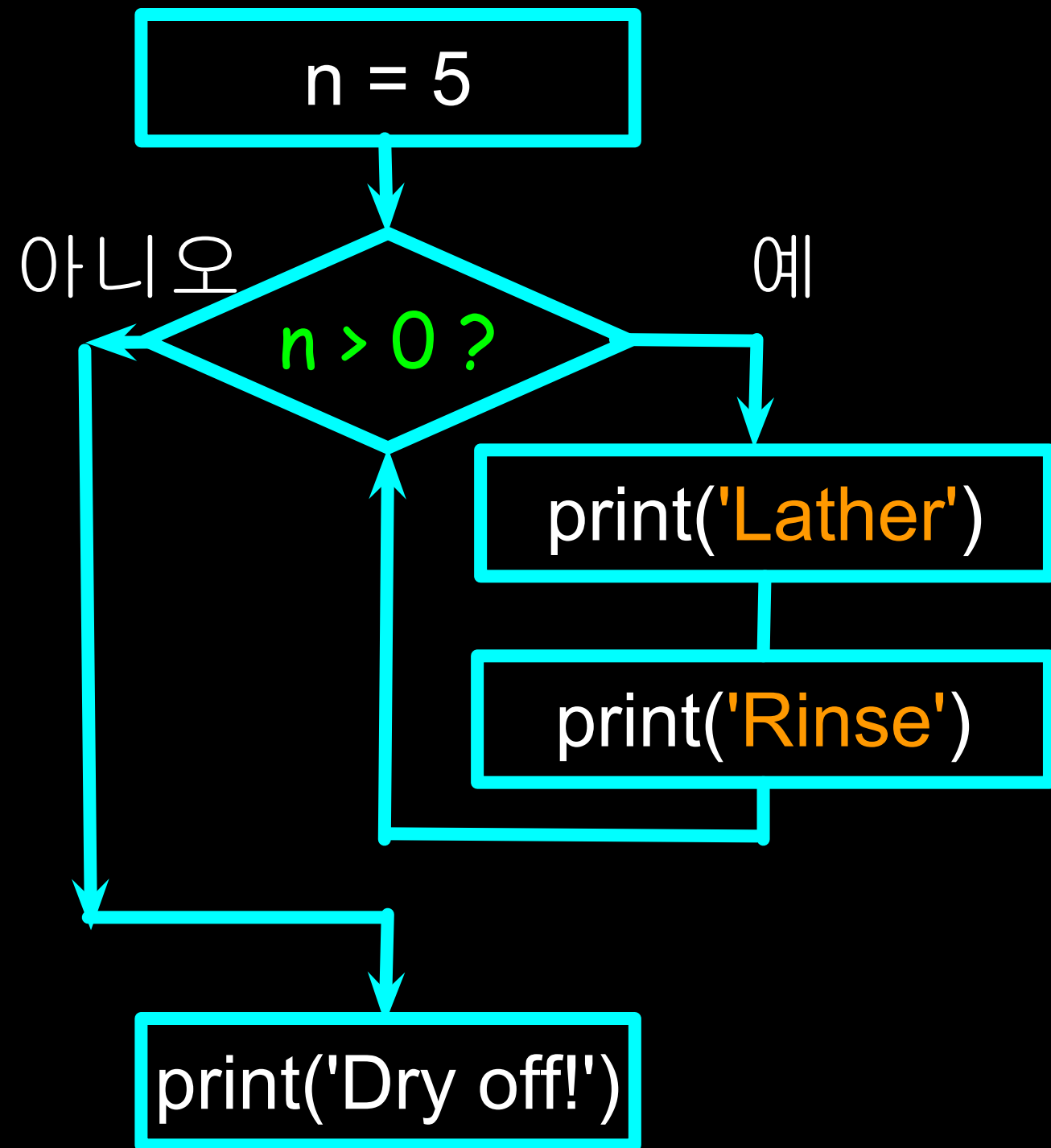
```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print('Blastoff!')
print(n)
```

출력:

5
4
3
2
1
Blastoff!
0

루프 (반복 단계) 는 각 루프 마다 변하는 반복 변수를 가지고 있음

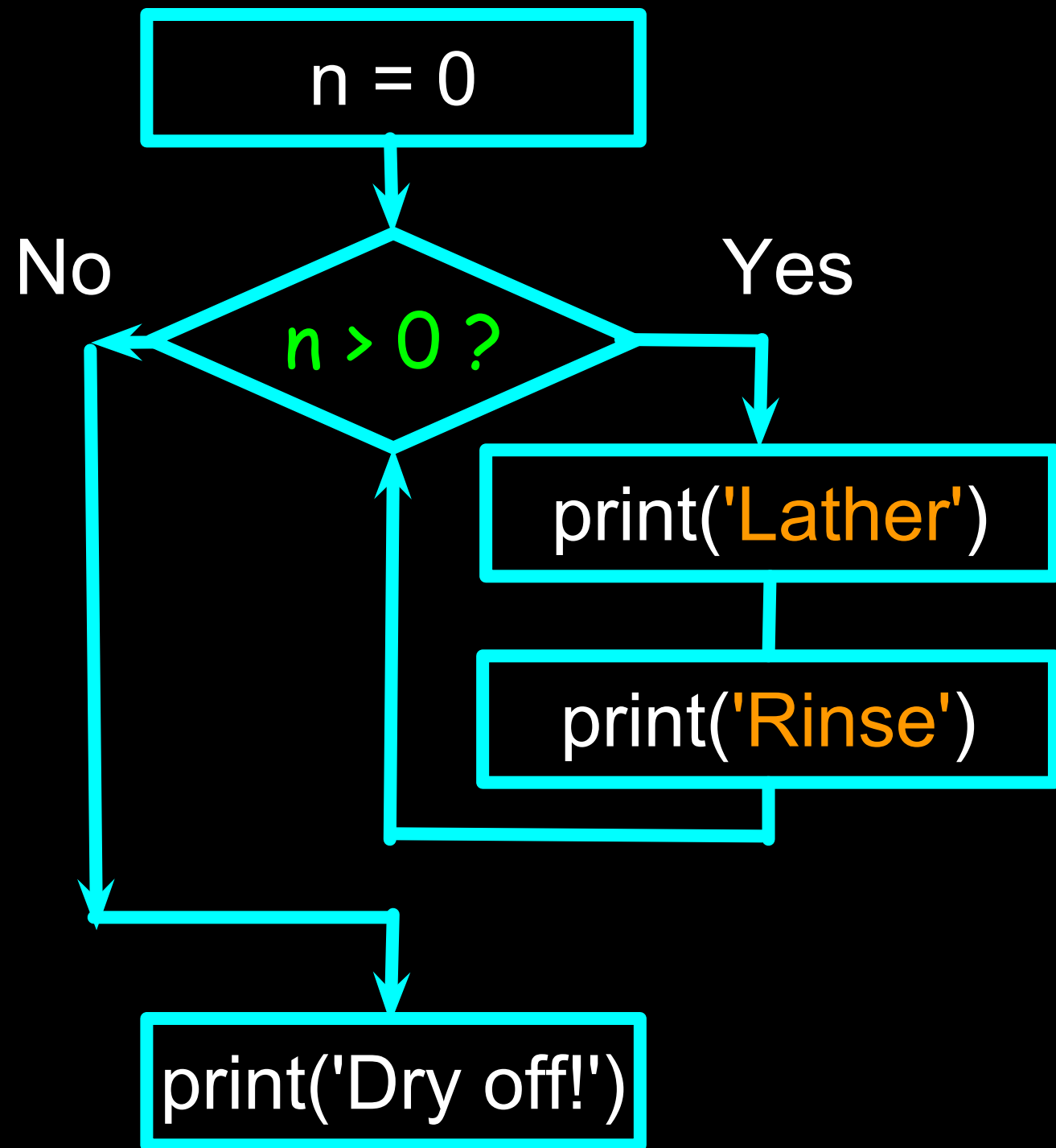
종종 반복 변수는 연속된 숫자를 차례대로 받음



무한 루프

```
n = 5
while n > 0 :
    print('Lather')
    print('Rinse')
print('Dry off!')
```

이 루프의 문제?



또 다른 루프

```
n = 0
while n > 0 :
    print('Lather')
    print('Rinse')
print('Dry off!')
```

이 루프의 역할은?

루프에서 빠져나오기

- **break** 구문은 현재 루프를 끝내고 루프 다음에 있는 구문으로 바로 건너뛴다
- 루프 본문 어디에서든 일어날 수 있는 루프 테스트와 같음


```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> finished
finished
> done
Done!
```

루프에서 빠져나오기

- **break** 구문은 현재 루프를 끝내고 루프 다음에 있는 구문으로 바로 건너뛴다
- 루프 본문 어디에서든 일어날 수 있는 루프 테스트와 같음

```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

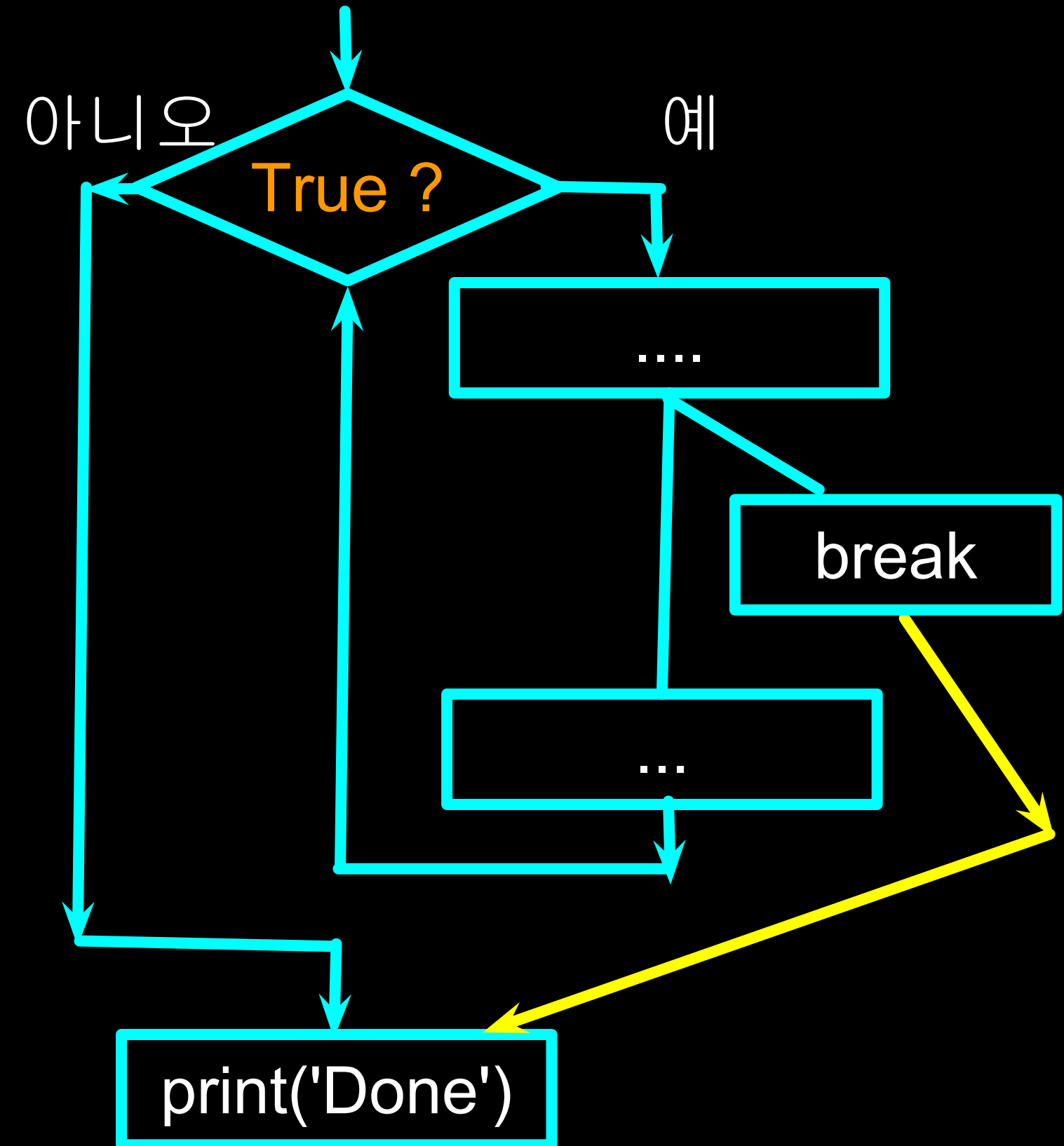


```
> hello there
hello there
> finished
finished
> done
Done!
```

```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```



[http://en.wikipedia.org/wiki/Transporter_\(Star_Trek\)](http://en.wikipedia.org/wiki/Transporter_(Star_Trek))



continue로 반복문 끝내기

continue구문은 현재 반복을 끝내고 루프의 시작으로 점프해서 다음 반복을 실행

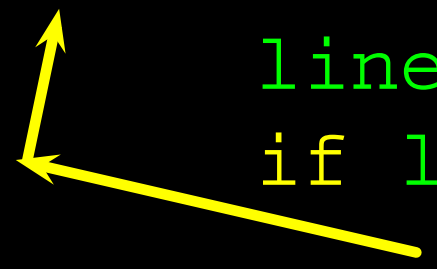
```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```


continue로 반복문 끝내기

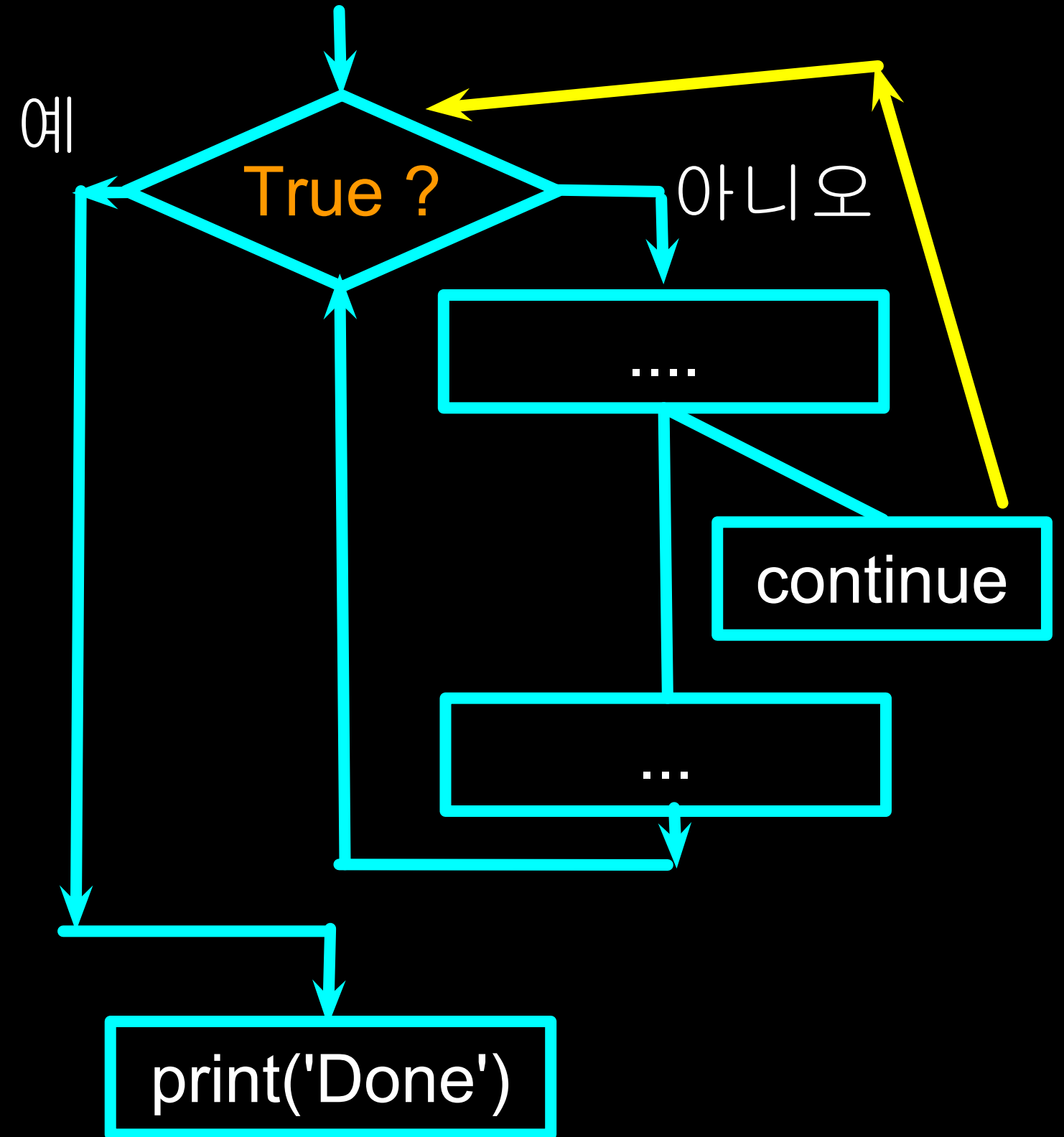
continue구문은 현재 반복을 끝내고 루프의 시작으로 점프해서 다음 반복을 실행

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```



```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```

```
while True:
    line = raw_input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```



불확정 루프

- `while` 루프는 조건문이 거짓이 되기 전까지 계속 실행 되기 때문에 “불확정 루프”라고 불림
- 지금까지 본 루프는 종료 가능한지 아니면 “무한 루프”인지 검토하기 쉬웠음
- 그러나 가끔은 루프가 종료할 수 있는지 확인하기 어려울 수 있음

유한 루프

집합의 원소에 대해 반복하는 경우

유한 루프

- 우리는 자주 어떤 항목의 **리스트**, 예를 들어 **파일의 줄**을 데이터로 받음 - 다른 말로 어떤 것의 **유한 집합**
- 파이썬 **for** 구조를 이용해서 집합의 각 항목에 대해서 반복문을 실행하는 루프를 만들 수 있음
- 정확히 특정 횟수 만큼 실행되므로 이 루프를 “**유한 루프**” 라고 함
- “**유한 루프는 집합의 원소를 통해 반복**”

간단한 유한 루프

```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff!')
```

5

4

3

2

1

Blastoff!

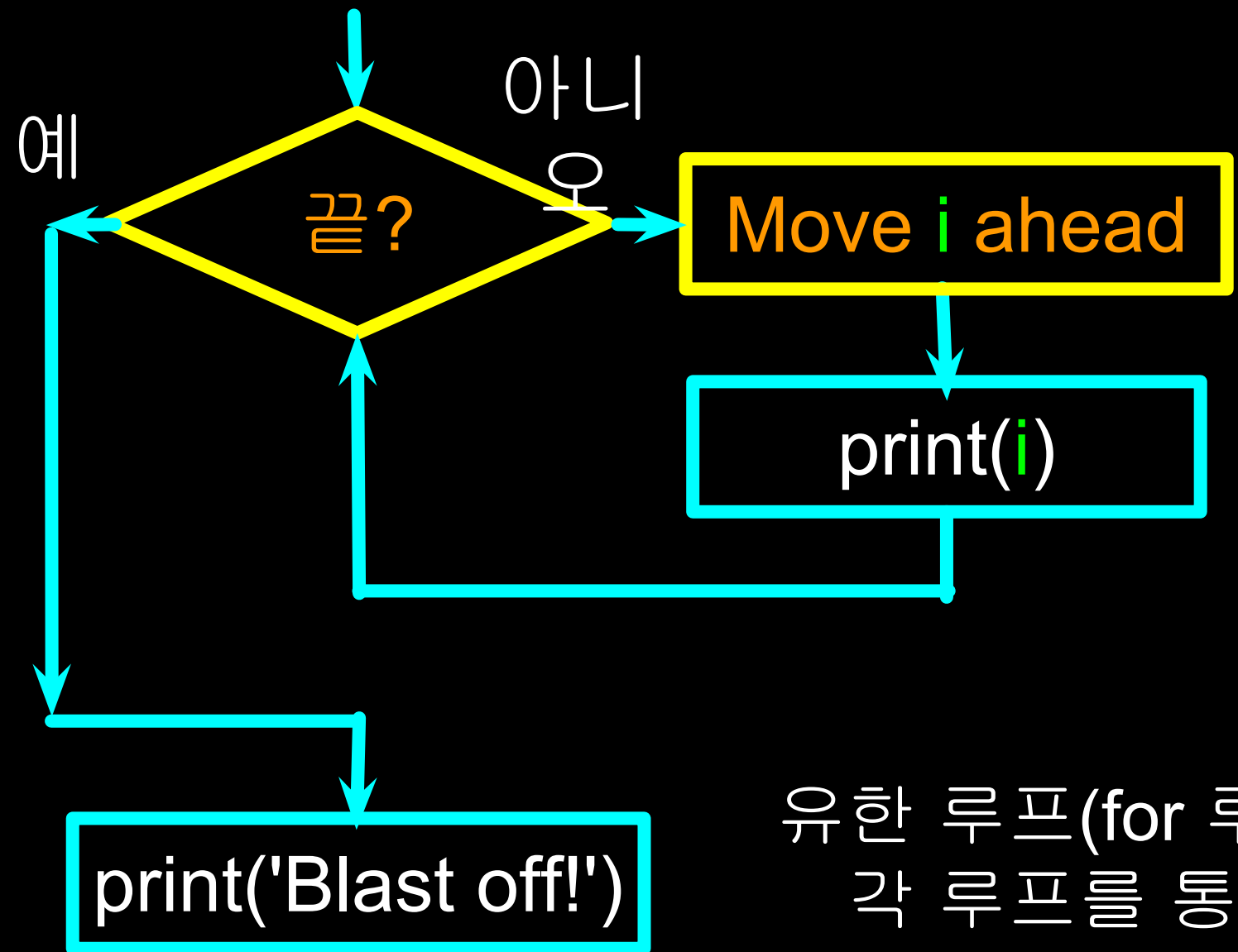
문자열을 이용한 유한 루프

```
friends = ['Joseph', 'Glenn', 'Sally']  
for friend in friends :  
    print('Happy New Year:', friend)  
print('Done!')
```

Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally

Done!

간단한 유한 루프



```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff!')
```

5
4
3
2
1
Blastoff!

유한 루프(for 루프)는 명시된 **반복 변수**를 가지고 있으며
각 루프를 통과할 때 마다 값이 변함. 이 **반복 변수**는
시퀀스나 집합의 원소를 따라 이동하며 값이 변함

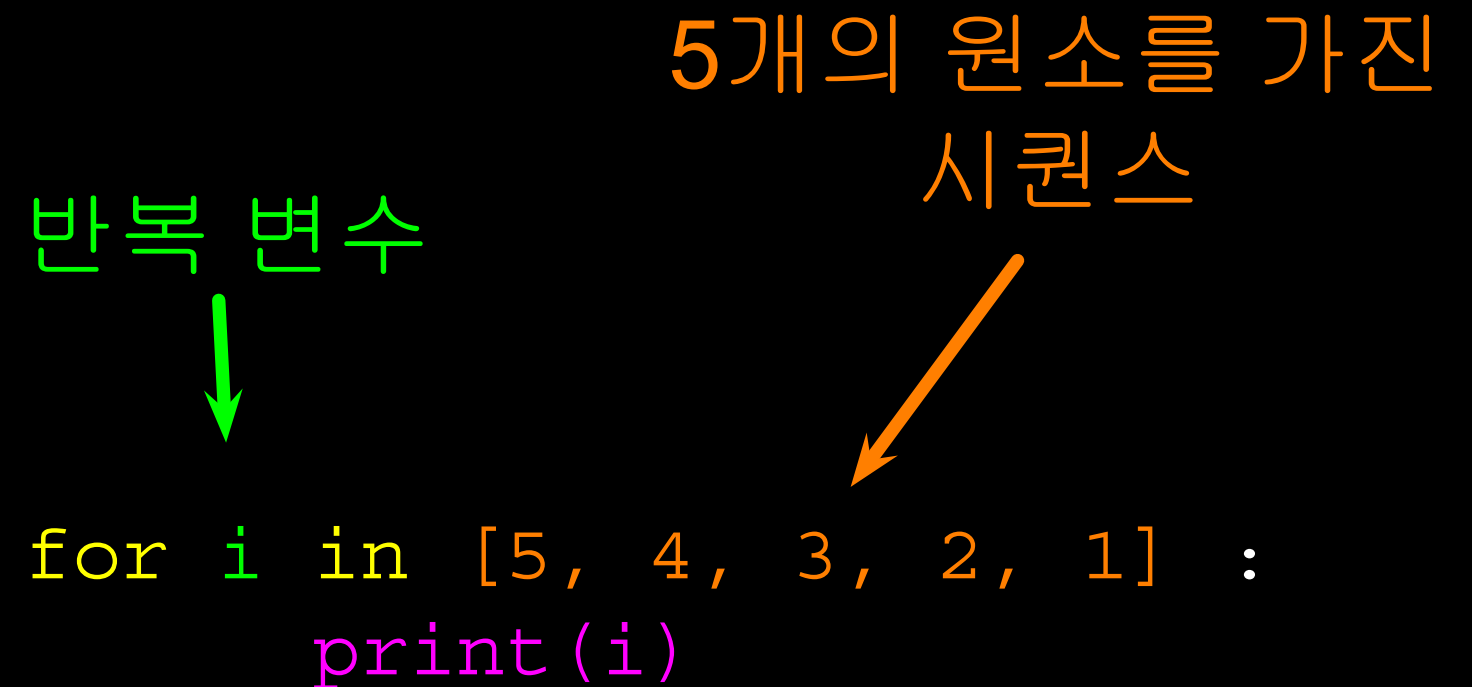
in 파헤치기

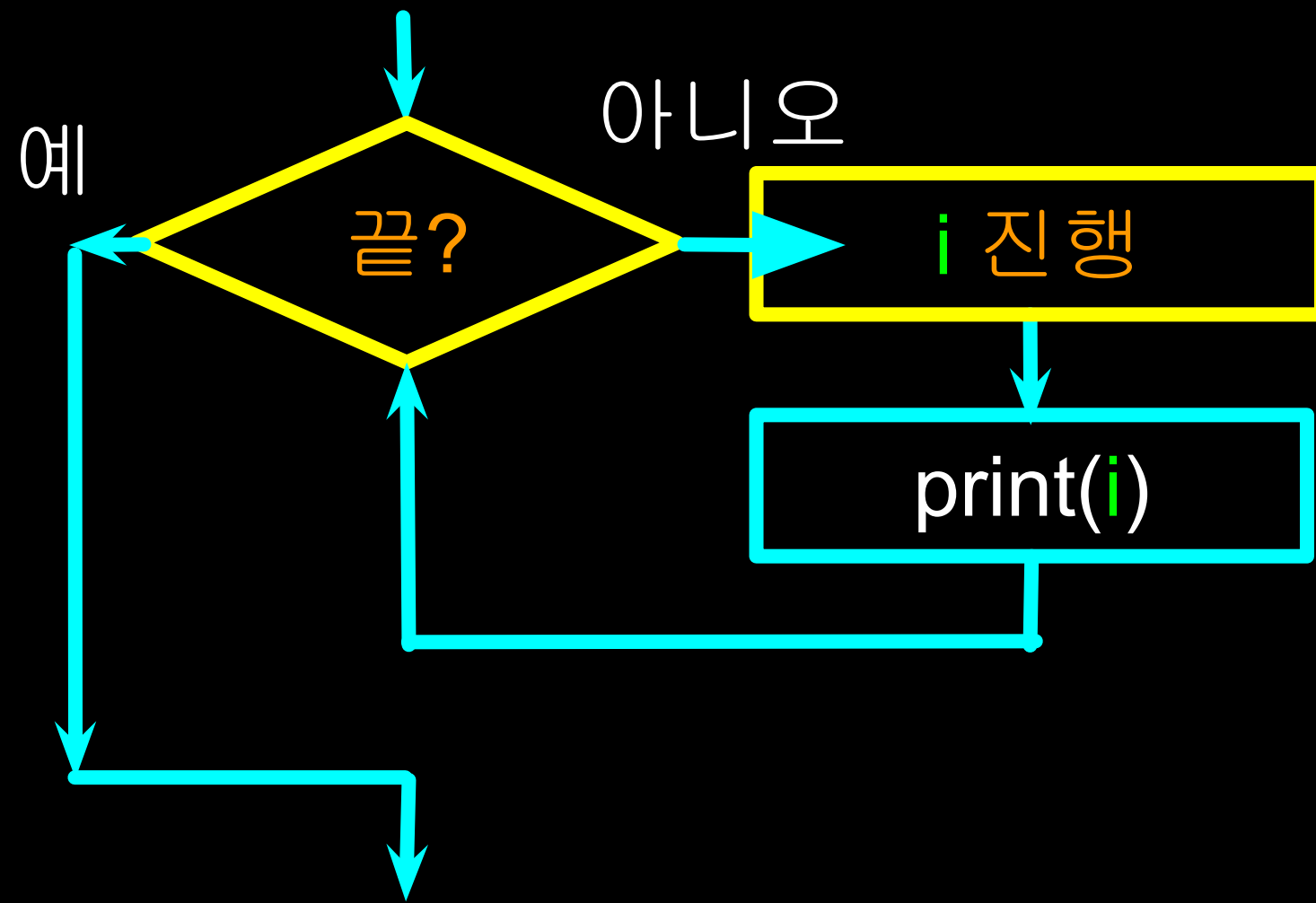
- 반복 변수는 시퀀스(순서가 있는 집합)를 통해 “반복”
- 코드의 루프 블록(본문)은 시퀀스 안의 각 값에 대해 한번씩 실행
- The 반복 변수는 시퀀스 안의 모든 값을 가지고 실행

5개의 원소를 가진
시퀀스

반복 변수

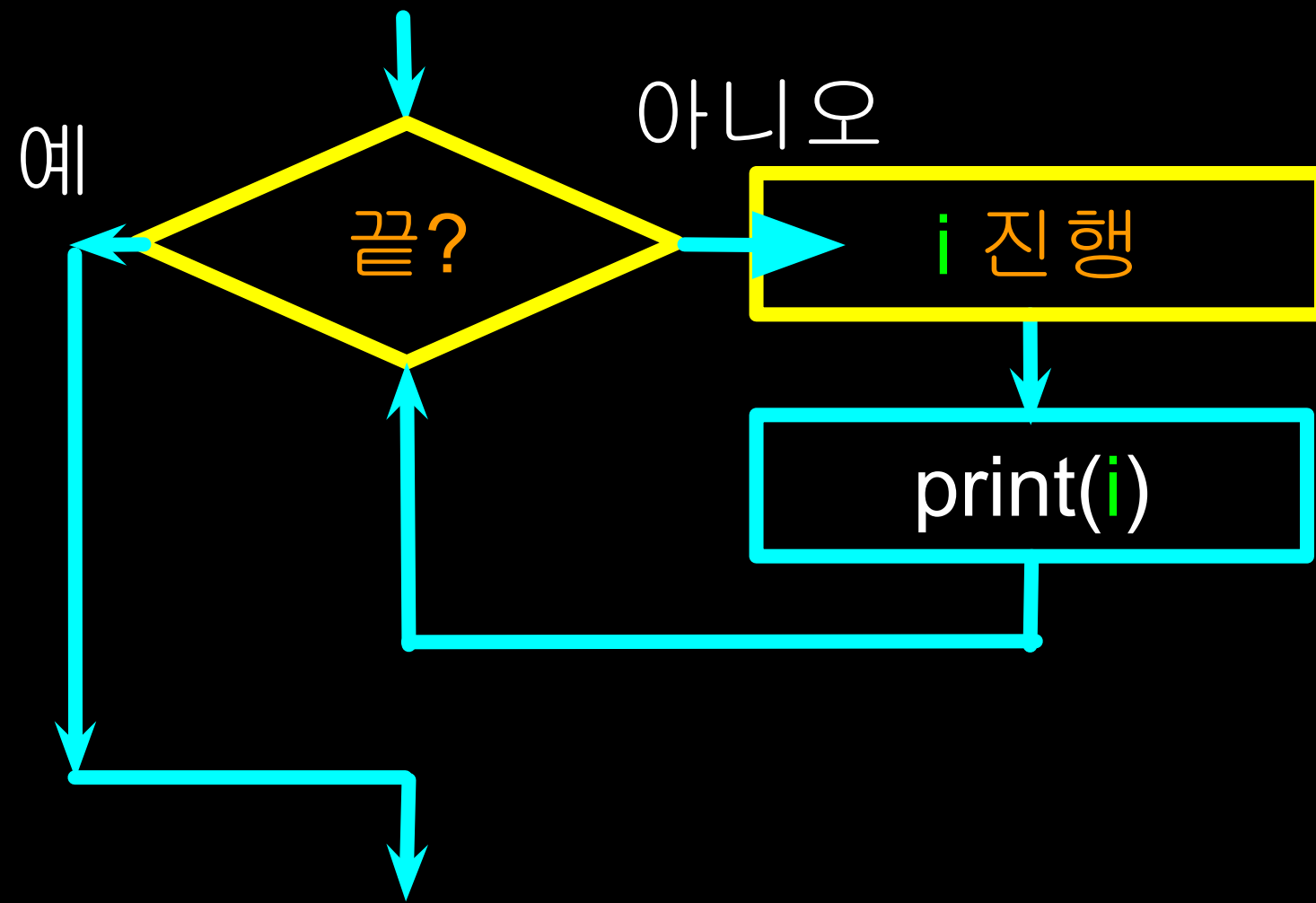
```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```



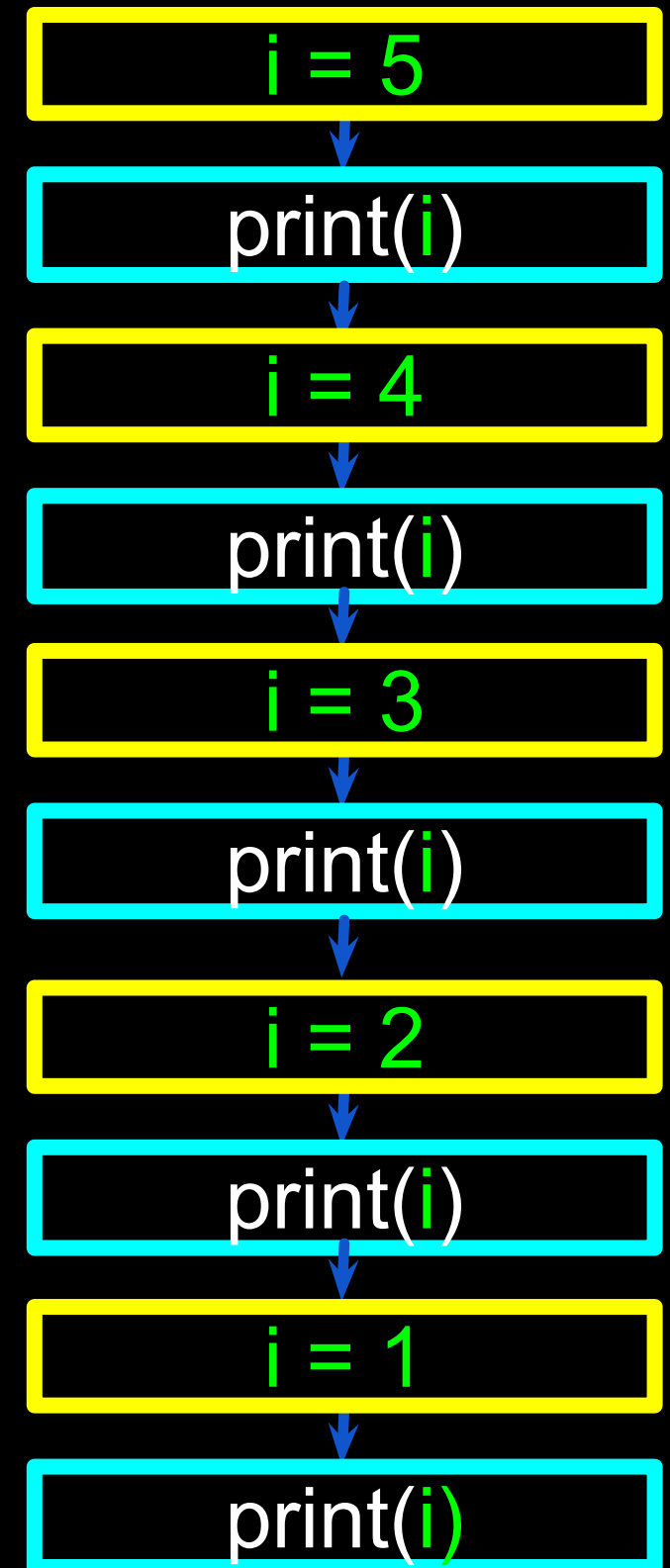


```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

- 반복 변수는 시퀀스(순서가 있는 집합)를 통해 “반복”
- 코드의 루프 블록 (본문)은 시퀀스 안의 각 값에 대해 한번씩 실행
- 반복 변수는 시퀀스 안의 모든 값을 가지고 실행



```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```



루프 패턴: 우리가 루프에서 하는 것

노트: 예제는 간단하지만,
패턴은 모든 루프에 대해 적용 가능

“똑똑한” 루프 만들기

원소를 하나씩 한 번에 보는
코드를 짜는 것이 어렵다면
전체의 루프의 작동방식에 대해
알아야 함

변수를 초기 값으로 설정

for thing in data:

각 원소에 대해
독립적으로 탐색하거나
무언가를 하고 변수
값을 업데이트

변수 값 확인

집합을 이용한 루프

```
print('Before')
for thing in [9, 41, 12, 3, 74, 15] :
    print(thing)
print('After')
```

\$ python basicloop.py

Before

9

41

12

3

74

15

After

무엇이 최대값인가?

무엇이 최대값인가?

3

무엇이 최대값인가?

41

무엇이 최대값인가?

12

무엇이 최대값인가?

9

무엇이 최대값인가?

74

무엇이 최대값인가?

15

무엇이 최대값인가?

무엇이 최대값인가?

3 41 12 9 74 15

무엇이 최대값인가?

largest_so_far

-1

무엇이 최대값인가?

3

largest_so_far

3

무엇이 최대값인가?

41

largest_so_far

41

무엇이 최대값인가?

12

largest_so_far

41

무엇이 최대값인가?

9

largest_so_far

41

무엇이 최대값인가?

74

largest_so_far

74

무엇이 최대값인가?

15

74

무엇이 최대값인가?

3 41 12 9 74 15

74

무엇이 최대값인가?

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('After', largest_so_far)
```

\$ python largest.py

Before -1

9 9

41 41

41 12

41 3

74 74

74 15

After 74

우리가 확인한 숫자 중 최대값을 저장하는 변수를 생성.
현재 보고 있는 숫자가 더 크면, 그 숫자가 새로운 확인한 숫자 중 최대값.

다른 루프 패턴들...

루프에서 개수 구하기

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + 1
    print(zork, thing)
print('After', zork)
```

\$ python countloop.py

Before 0

1 9

2 41

3 12

4 3

5 74

6 15

After 6

루프를 몇 번 실행했는지 **횟수**를 구하기 위해서
0에서 시작하는 카운팅 변수를 도입하고 루프를 실행할 때 **마다 1을 더함**

루프에서 합계 구하기

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + thing
    print(zork, thing)
print('After', zork)
```

\$ python countloop.py

Before 0

9 9

50 41

62 12

65 3

139 74

154 15

After 154

루프에서 만난 값을 모두 더하기 위해,
0에서 시작하는 합계 변수를 도입하고 루프를 실행할 때 마다 값을 더함

루프에서 평균 구하기

```
count = 0
sum = 0
print('Before', count, sum)
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print(count, sum, value)
print('After', count, sum, sum / count)
```

```
$ python averageloop.py
```

```
Before 0 0
```

```
1 9 9
```

```
2 50 41
```

```
3 62 12
```

```
4 65 3
```

```
5 139 74
```

```
6 154 15
```

```
After 6 154 25.666
```

평균은 개수 구하기 패턴과 합계 패턴을 결합해서 구할 수
있음.

루프가 끝나면 합계를 개수로 나눔.

루프에서 필터링 하기

```
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
        print('Large number',value)
print('After')
```

\$ python search1.py

Before

Large number 41

Large number 74

After

if 구문을 루프에서 사용해서
찾고자 하는 값을 발견하거나 필터링

불리언 변수 이용해서 탐색하기

```
found = False
print('Before', found)
for value in [9, 41, 12, 3, 74, 15] :
    if value == 3 :
        found = True
    print(found, value)
print('After', found)
```

```
$ python search1.py
```

```
Before False
```

```
False 9
```

```
False 41
```

```
False 12
```

```
True 3
```

```
True 74
```

```
True 15
```

```
After True
```

단순히 탐색해서 어떤 값이 존재하는지 알고 싶다면, **False**값으로 시작하는 변수를 도입해서 찾고자 하는 값을 찾는 순간 값을 **True**로 바꿉니다

최솟값 찾기

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('After', largest_so_far)
```

\$ python largest.py

Before -1

9 9

41 41

41 12

41 3

74 74

74 15

After 74

이 코드를 어떻게 바꾸면 최솟값을 찾을 수 있나?

최솟값 찾기

```
smallest_so_far = -1
print('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('After', smallest_so_far)
```

변수명을 **smallest_so_far**로 바꾸고 연산자 **>**를 연산자 **<**로 치환

최솟값 찾기

```
smallest_so_far = -1
print('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('After', smallest_so_far)
```

\$ python smallbad.py

Before -1

-1 9

-1 41

-1 12

-1 3

-1 74

-1 15

After -1

변수명을 **smallest_so_far**로 바꾸고 연산자 >를 연산자 <로 바꾸었습니다

최솟값 찾기

```
smallest = None
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('After', smallest)
```

\$ python smallest.py

Before

9 9

9 41

9 12

3 3

3 74

3 15

After 3

여전히 예비 **최소값**을 저장하는 변수가 존재. 루프의 첫 번째 실행에서 **smallest**의 값은 **None**이므로, 리스트 첫 번째 **값**을 **smallest**에 저장.

is 와 is not 연산자

```
smallest = None
print('Before')
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)

print('After', smallest)
```

- 파이썬은 논리 표현식에 사용할 수 있는 **is** 연산자를 가지고 있음
- “양변은 같은 값이다”를 의미
- 비슷하지만 **==** 보다 강력
- **is not** 역시 논리 연산자임

요약

- while 루프 (불확정)
- 무한 루프
- break 구문
- continue 구문
- None 상수와 변수
- for 루프 (유한)
- 반복 변수
- 루프 패턴
- 최대값과 최소값 구하기



Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

Contributor:

- Seung-June Lee (plusjune@gmail.com)
- Connect Foundation

Translator:

- Hakyong Kim
- Jeungmin Oh (tangza@gmail.com)