

【수업 준비】

1. 바탕화면 오른쪽 위 **출석확인** : 자기이름 쓰기

2. 깃허브 접속하기

<https://github.com/swKyungbock>

3. 깃허브 링크 **진단평가** 풀기

4. 깃허브에서 **수업자료** 살펴보기

[중간고사 전까지 배운 내용: RUR-PLE]

[RUR-PLE(러플) 메뉴]



[RUR-PLE 내장함수]

1) move()

2) turn_off()

3) turn_left()

4) pick_beeper()

5) put_beeper()

[RUR-PLE 사용자 정의 함수]

사용자 정의 함수 만드는 형식

```
def 사용자_함수_이름( ) :
```

```
    명령-1
```

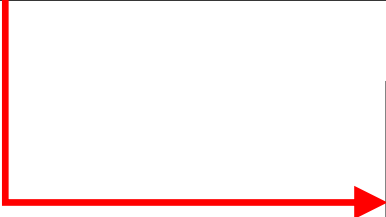
```
    명령-2
```

```
    ....
```

[RUR-PLE repeat 함수]

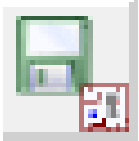
repeat(A,B) : A명령어를 B번 반복하게 하는 함수

```
def turn_right() :  
    turn_left()  
    turn_left()  
    turn_left()
```



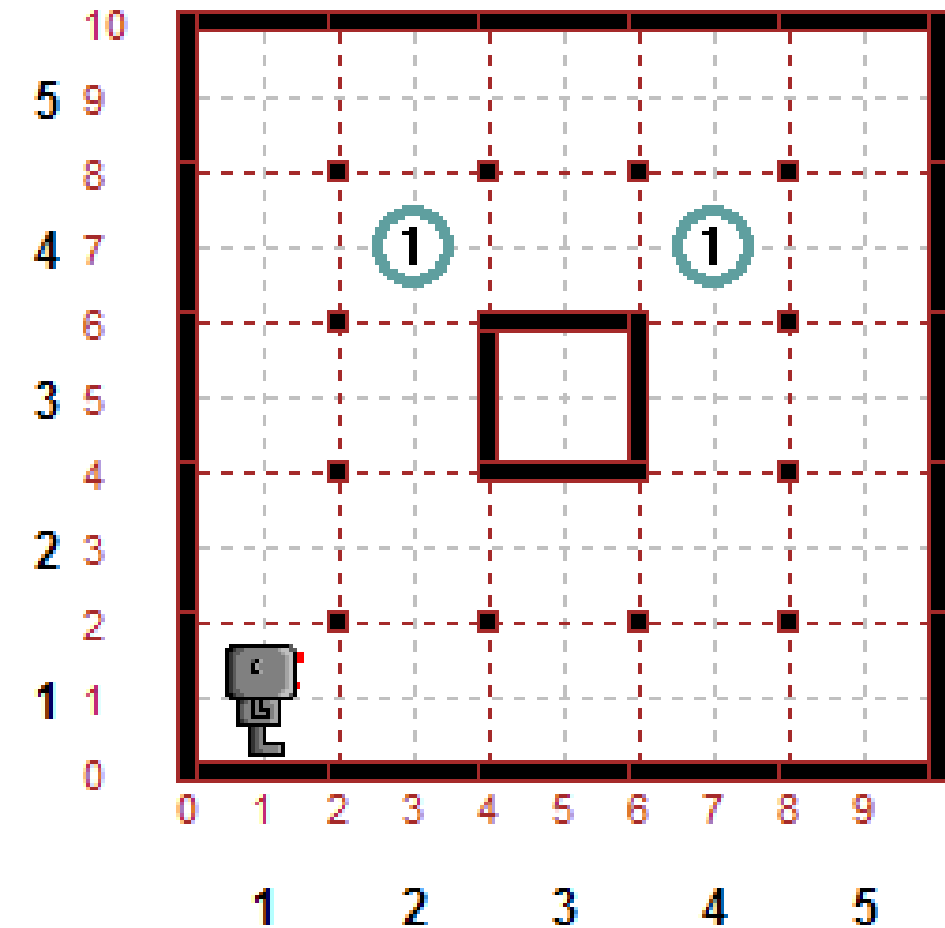
```
def turn_right() :  
    repeat(turn_left, 3)
```

[RUR-PLE 저장하기]

1) 월드 저장하기  파일명.wld

2) 코드 저장하기  파일명 .rur

[RUR-PLE 월드 파일의 정보]



```
avenues = 5
streets = 5
robot = (1, 1, 'E', 0)
walls = [
    (4, 5),
    (5, 6),
    (6, 5),
    (5, 4)
]
beepers = {
    (4, 4): 1,
    (2, 4): 1
}
```

월드의 가로 크기

월드의 세로 크기

로봇정보(x좌표, y좌표, 방향, 비퍼 수)

벽 정보(자료형:리스트)

비퍼 정보(자료형:사전)

【오늘의 수업 목표!】

조건에 따라 움직이는 똑똑한 리보그 만들기

[오늘의 수업내용 : RUR-PLE(러플)]

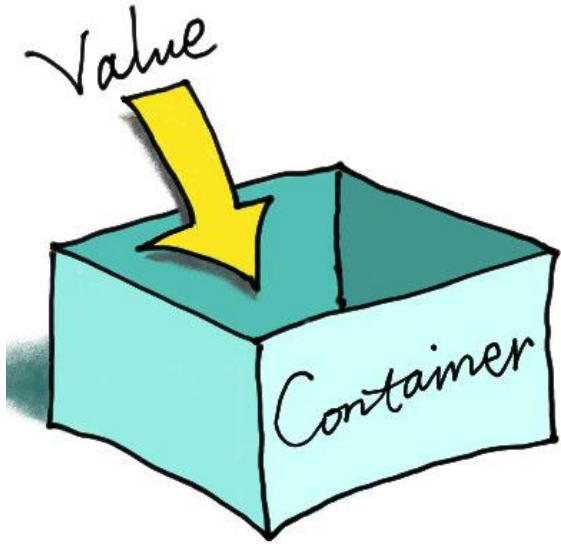
#1. 변수

#2. 센서함수

#3. 조건문

#1. 변수

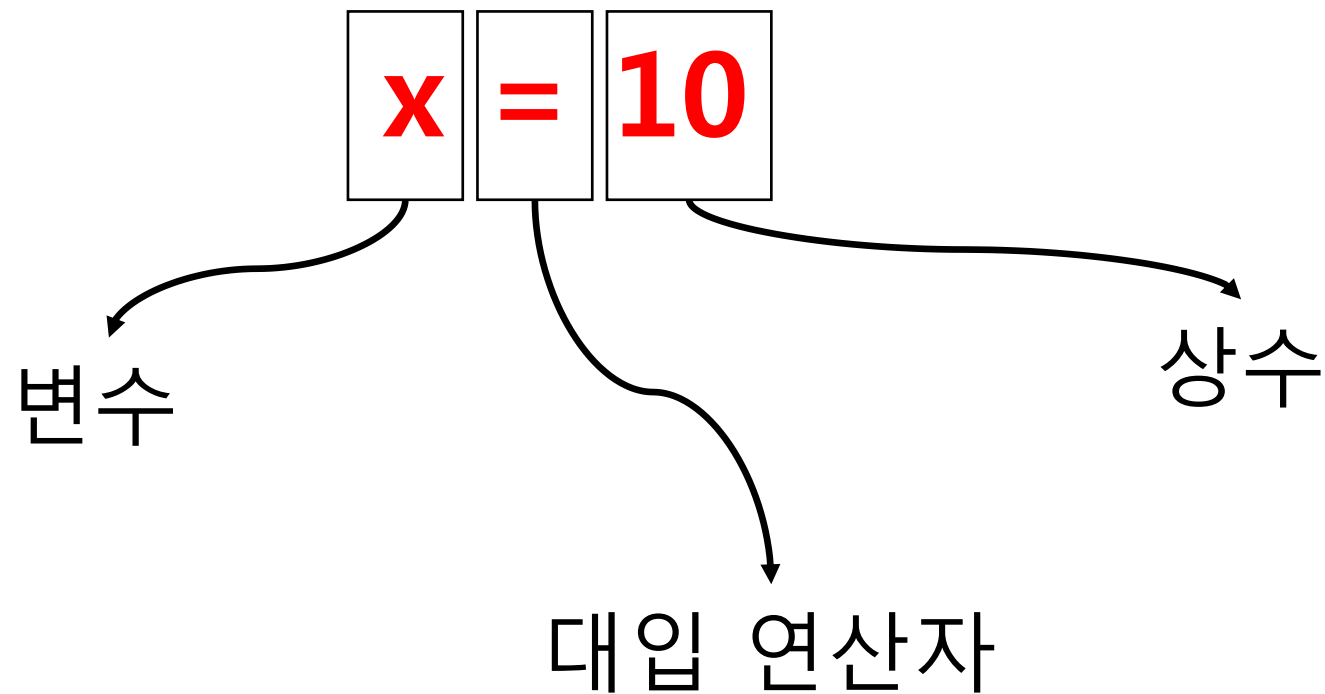
【변수】 : 특정 숫자나 글자 등을 저장할 수 있는 기억 공간



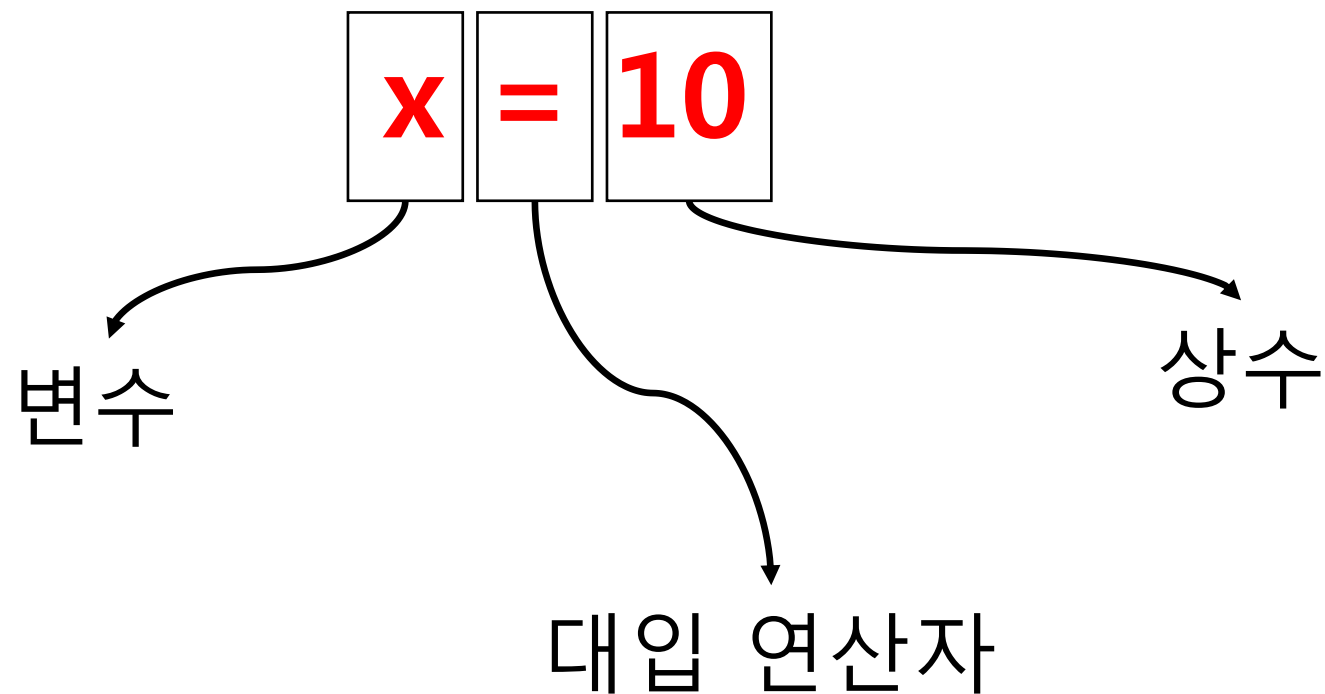
$x = 10$

x 에 10을 넣는다.

【변수】 : 특정 숫자나 글자 등을 저장할 수 있는 기억 공간



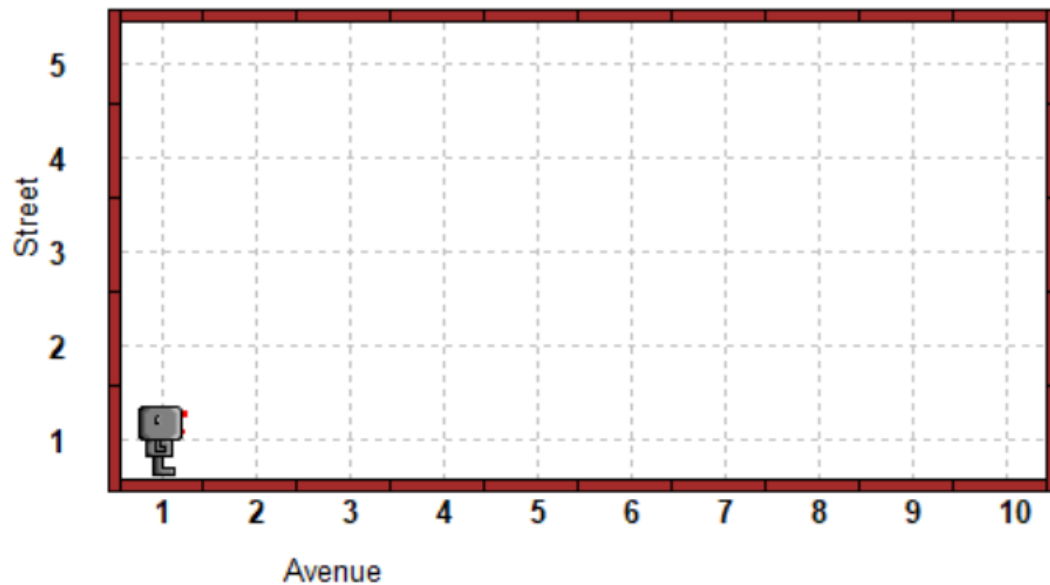
【변수】 : 특정 숫자나 글자 등을 저장할 수 있는 기억 공간



변수 x에 상수 10을 대입한다.

[변수] : 다음 프로그램의 결과를 예상해 보세요!

```
steps=0
def move_and_step_count():
    ... move()
    ... steps=steps+1
#program start
repeat(move_and_step_count, 9)
print(steps)
turn_off()
```



[변수] : 다음 프로그램의 결과를 예상해 보세요!

```
steps=0
```

```
def mov
```

```
... mov
```

```
... ste
```

```
#progra
```

```
repeat
```

```
print(s
```

```
turn_of
```

Execution error



local variable 'steps' referenced before assignment
Unrecognized instruction.

확인

【변수】 : 지역변수 Vs. 전역변수

지역변수(Local Variable)

전역변수(Global Variable)

【변수】 : 지역변수 Vs. 전역변수

지역변수(Local Variable)

함수 또는 특정 범위 내에서만 활용 가능

전역변수(Global Variable)

프로그램 전체에서 활용 가능

[변수] : 지역변수 Vs. 전역변수

```
steps=0
```

```
def move_and_step_count():
```

```
    move()
```

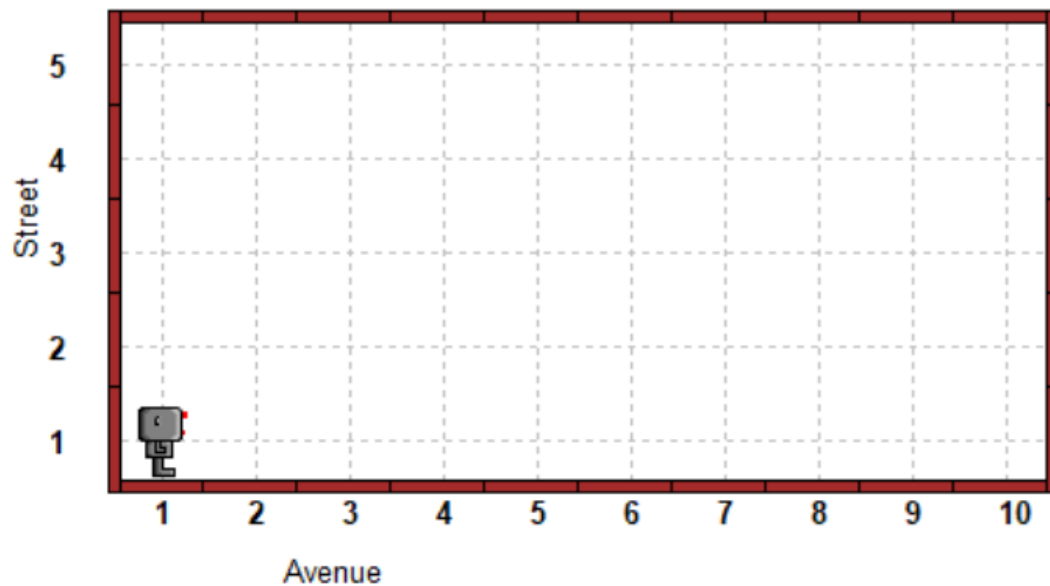
```
    steps=steps+1
```

```
#program start
```

```
repeat(move_and_step_count, 9)
```

```
print(steps)
```

```
turn_off()
```



[변수] : 지역변수 Vs. 전역변수

```
steps=0
```

```
def move_and_step_count():
```

```
    global steps
```

```
    move()
```

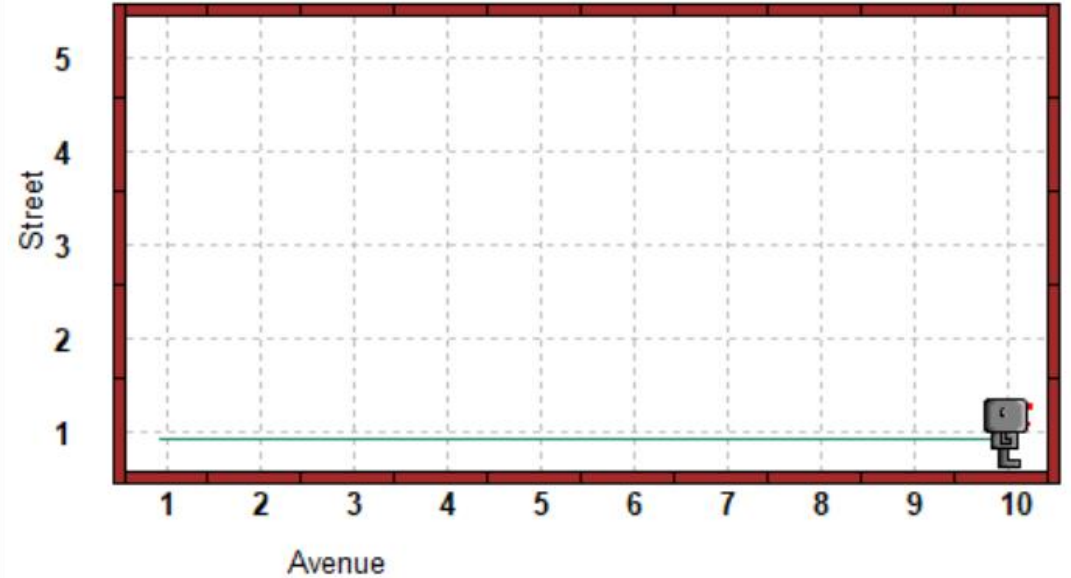
```
    steps=steps+1
```

```
#program start
```

```
repeat(move_and_step_count, 9)
```

```
print(steps)
```

```
turn_off()
```



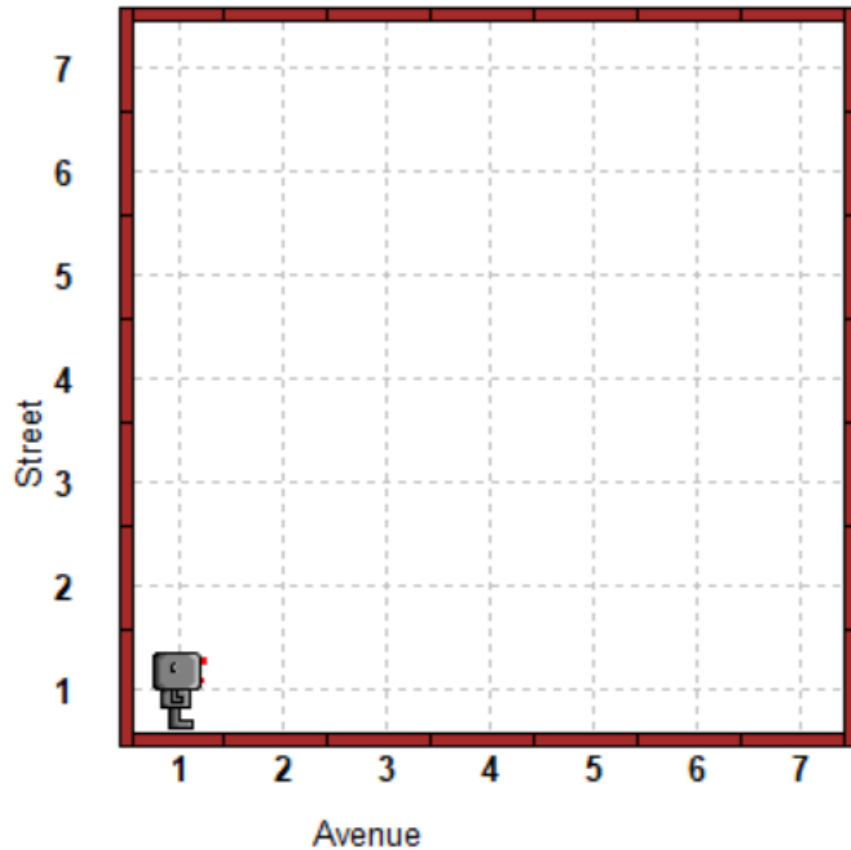
함수 밖에서 선언한 전역 변수를

함수 내에서 사용하고자 하면 **global**과 함께 선언

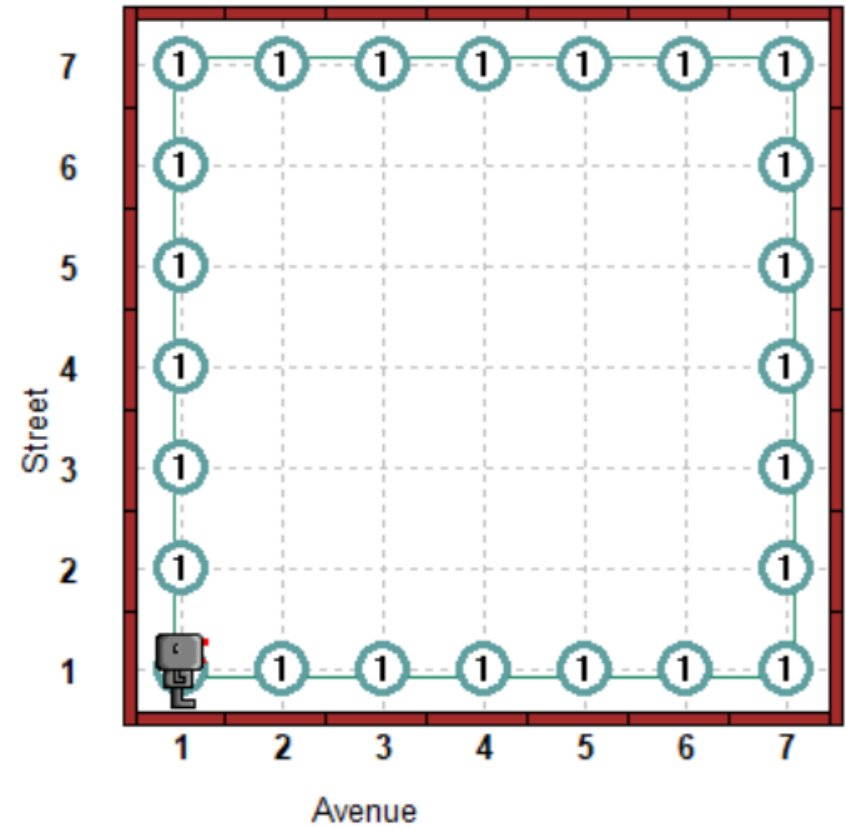
[Ex1_Variable]

리보그가 내려놓은 비퍼의 갯수를 출력해 보세요

<실행 전>

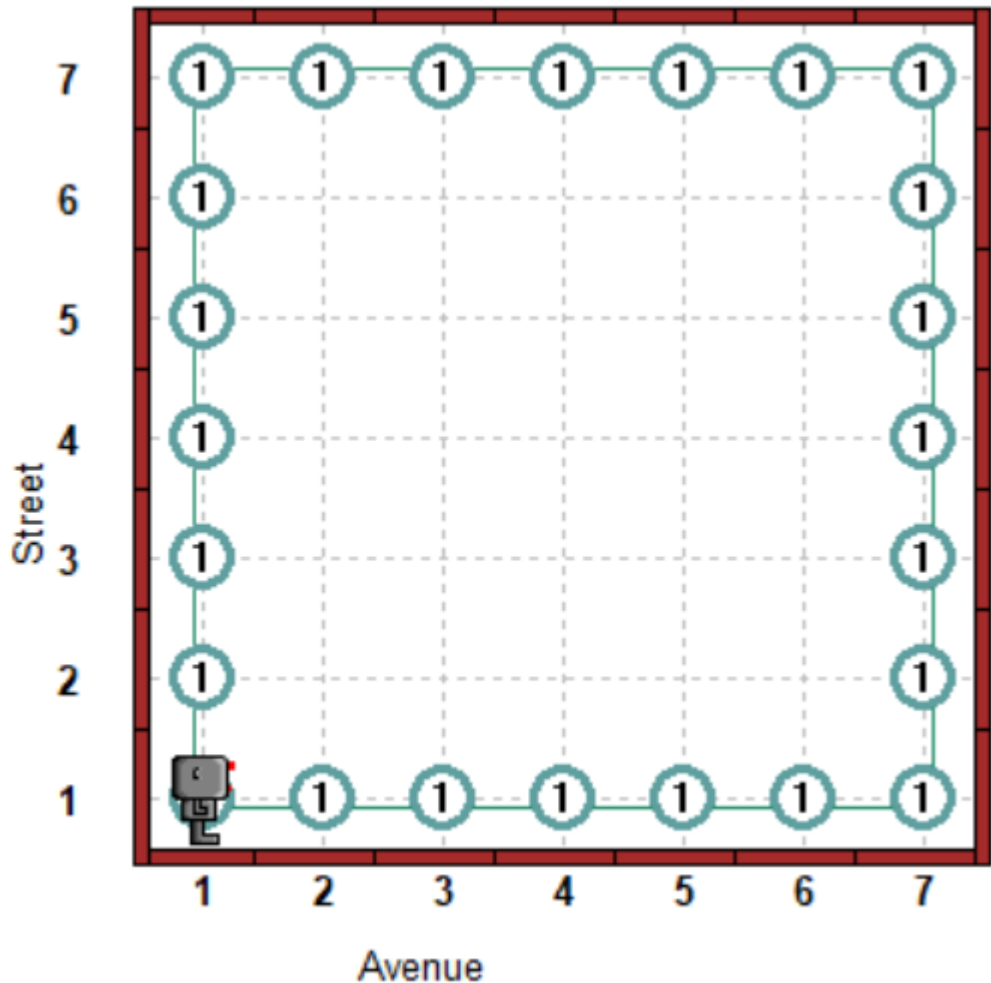


<실행 후>



[Ex1_Variable : Hint 프로그램에 변수를 추가하기]

리보그가 내려놓은 비퍼의 갯수를 출력해 보세요



```
#define functions
def move_and_put():
    move()
    put_beeper()
def walking_around():
    repeat(move_and_put, 6)
    turn_left()
#program start
repeat(walking_around, 4)
turn_off()
```

[Ex1_Variable : 완성]

리보그가 내려놓은 비퍼의 갯수를 출력해 보세요

```
#declare variable
```

```
beeperCnt=0
```

변수 선언과 초기화

```
#define functions
```

```
def move_and_put():
```

```
    ... global beeperCnt
```

전역 변수를 함수 내에서 사용하겠다는 선언

```
    ... move()
```

```
    ... put_beeper()
```

```
    ... beeperCnt=beeperCnt+1
```

변수의 값을 1씩 증가시킴

```
def walking_around():
```

```
    ... repeat(move_and_put, 6)
```

```
    ... turn_left()
```

```
#program start
```

```
repeat(walking_around, 4)
```

```
print(beeperCnt)
```

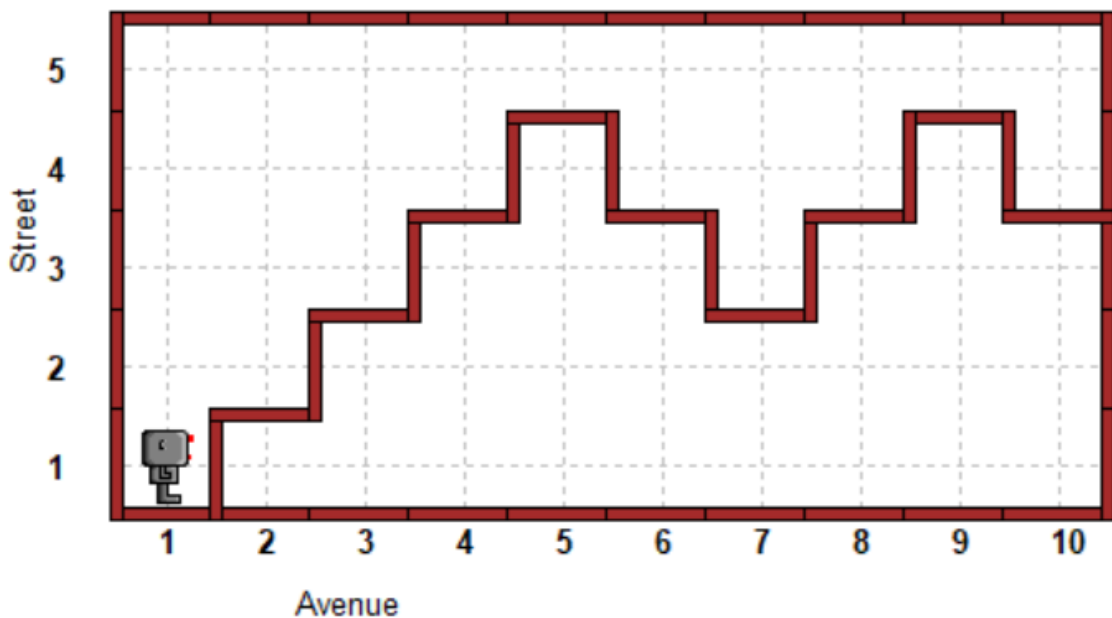
변수의 값을 출력

```
turn_off()
```

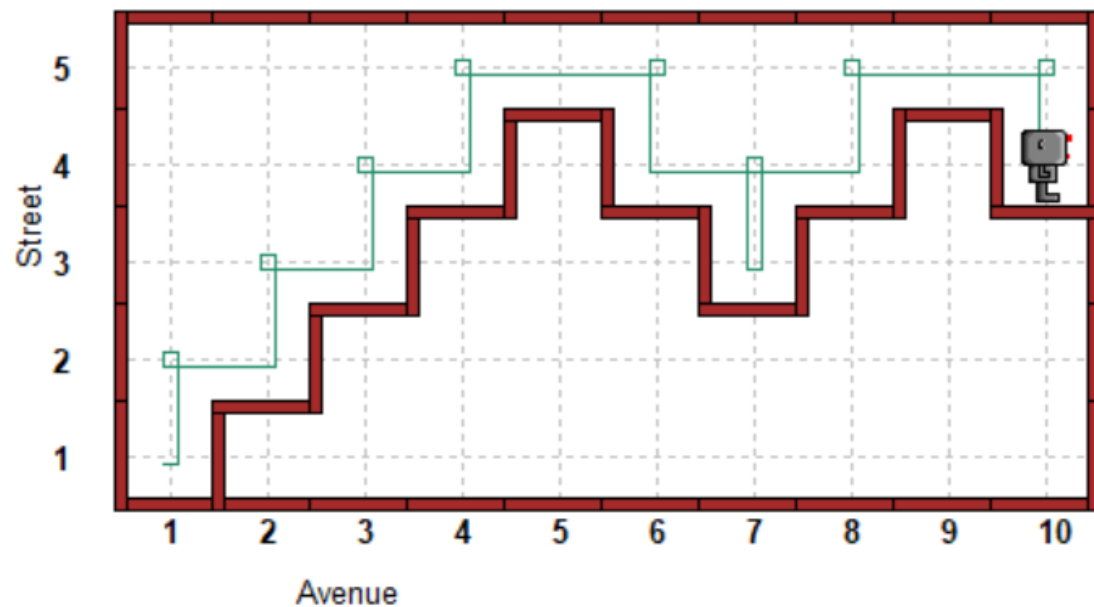
[Ex2_Variable]

리보그가 계단을 올라간 수를 세어보세요.

<실행 전>

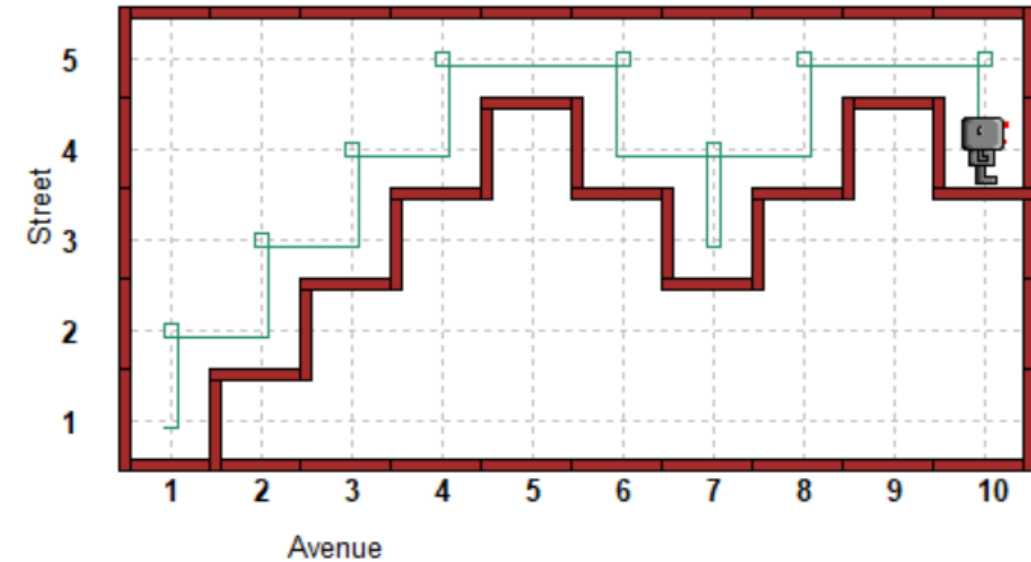


<실행 후>



[Ex2_Variable : Hint1(변수 외 프로그램 완성)]

리보그가 계단을 올라간 수를 세어보세요.



```
#define functions
def up_stair():
    ... global upStep
    ... turn_left()
    ... move()
    ... repeat(turn_left, 3)
    ... move()
def down_stair():
    ... move()
    ... repeat(turn_left, 3)
    ... move()
    ... turn_left()
#program start
repeat(up_stair, 4)
repeat(down_stair, 2)
repeat(up_stair, 2)
down_stair()
turn_off()
```

[Ex2_Variable : 완성(변수추가)]

리보그가 계단을 올라간 수를 세어보세요.

```
#declare variable
```

```
upStep=0
```

```
#define functions
```

```
def up_stair():
```

```
    ... global upStep
```

```
    ... turn_left()
```

```
    ... move()
```

```
    ... repeat(turn_left, 3)
```

```
    ... move()
```

```
    ... upStep=upStep+1
```

```
def down_stair():
```

```
    ... move()
```

```
    ... repeat(turn_left, 3)
```

```
    ... move()
```

```
    ... turn_left()
```

```
#program start
```

```
repeat(up_stair, 4)
```

```
repeat(down_stair, 2)
```

```
repeat(up_stair, 2)
```

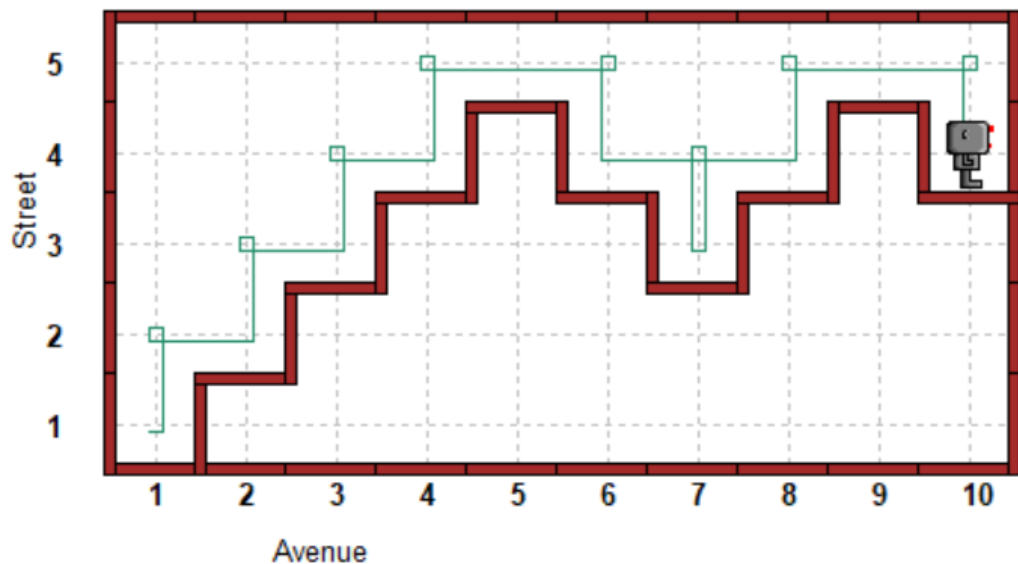
```
down_stair()
```

```
print(upStep)
```

```
turn_off()
```

[Ex2_Variable : 도전!]

리보그가 계단을 올라간 수, 내려간 수,
계단을 오르내린 총 횟수를 모두 세어 출력해 보세요.



```
1 ('upStep', 6)
2 ('downStep', 3)
3 ('totalStep', 9)
```

#2. 센서함수

【러플의 센서 함수】 현재 위치하고 있는 월드의 상태를 알수 있는 함수

```
front_is_clear()
```

```
left_is_clear()
```

```
right_is_clear()
```

```
on_beeper()
```

```
carries_beeper()
```

```
facing_north()
```

【러플의 센서 함수】 현재 위치하고 있는 월드의 상태를 알수 있는 함수

front_is_clear()	^□ D4□Ä Æ□?
------------------	-------------

left_is_clear()	□□ D4□Ä Æ□?
-----------------	-------------

right_is_clear()	\$x□□ D4□Ä Æ□?
------------------	----------------

on_beeper()	□ X□ D ^□?
-------------	-------------

carries_beeper()	□□∅ D □□ ^□?
------------------	----------------

facing_north()	□□∅ □□D □□ ^□?
----------------	-----------------

【러플의 센서 함수】 현재 위치하고 있는 월드의 상태를 알수 있는 함수

front_is_clear()	^□ D4□Ä Æ□?
------------------	-------------

left_is_clear()	□□ D4□Ä Æ□?
-----------------	-------------

right_is_clear()	\$x□□ D4□Ä Æ□?
------------------	----------------

on_beeper()	□ X□ D ^□?
-------------	-------------

carries_beeper()	□□∅ D □□ ^□?
------------------	----------------

facing_north()	□□∅ □□D □□ ^□?
----------------	-----------------

Yes
or
No

【러플의 센서 함수】 현재 위치하고 있는 월드의 상태를 알수 있는 함수

front_is_clear() ^ D4 Å Æ ?

left_is_clear() | D4 Å Æ ?

right_is_clear() \$x D4 Å Æ ?

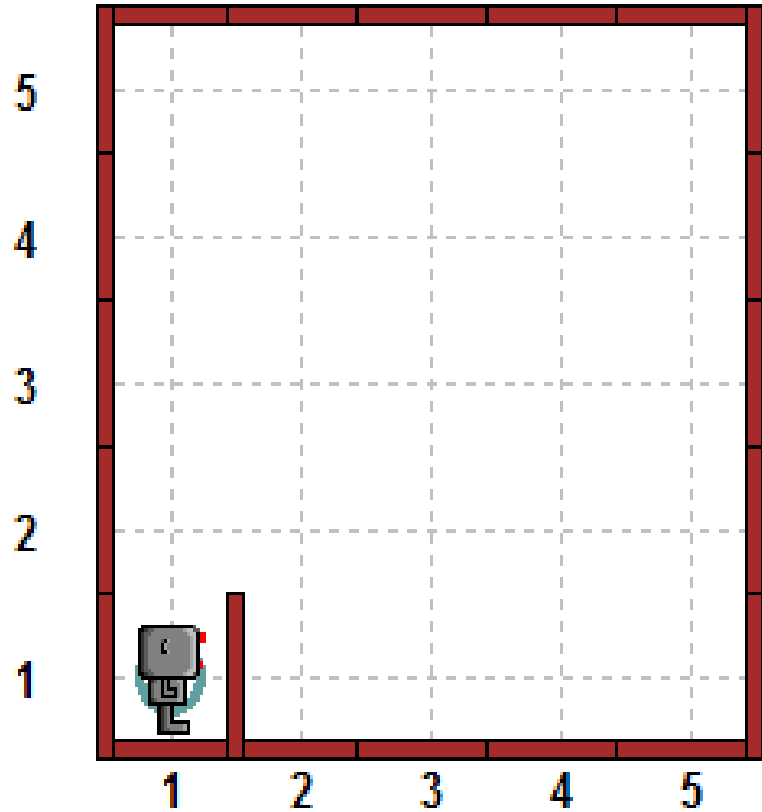
on_beeper() X D | ^ ?

carries_beeper() Ø D || Ø ^ ?

facing_north() Ø Ø D | Ø ^ ?

True
or
False

【러플의 센서 함수】 현재 위치하고 있는 월드의 상태를 알수 있는 함수



```
print front_is_clear()
```

```
print right_is_clear()
```

```
print left_is_clear()
```

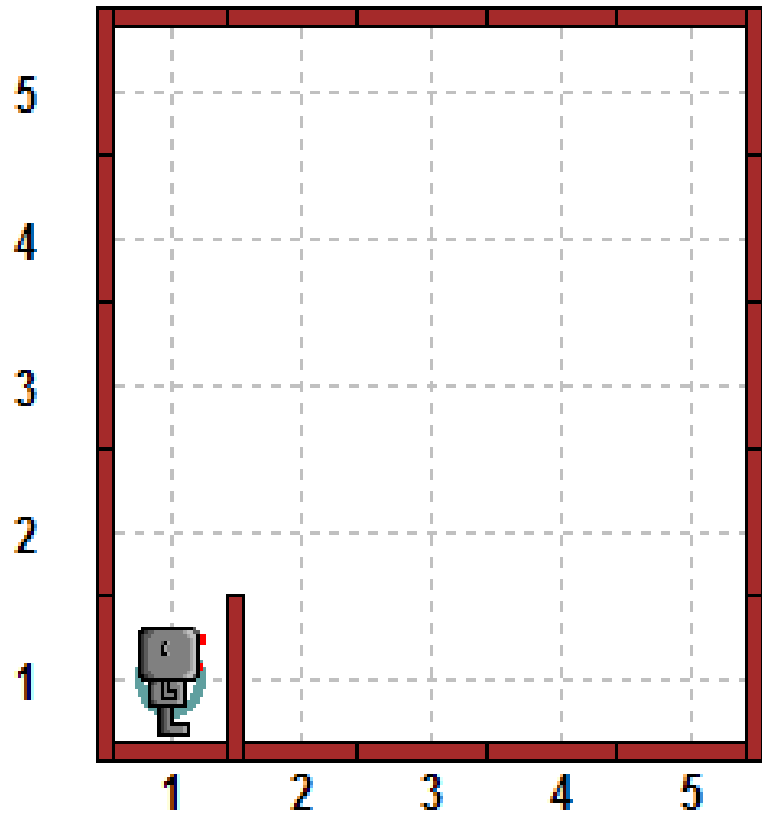
```
print on_beeper()
```

```
print carries_beeper()
```

```
print facing_north()
```

```
turn_off()
```

【러플의 센서 함수】 현재 위치하고 있는 월드의 상태를 알수 있는 함수



`print front_is_clear` False

`print right_is_clear` False

`print left_is_clear()` True

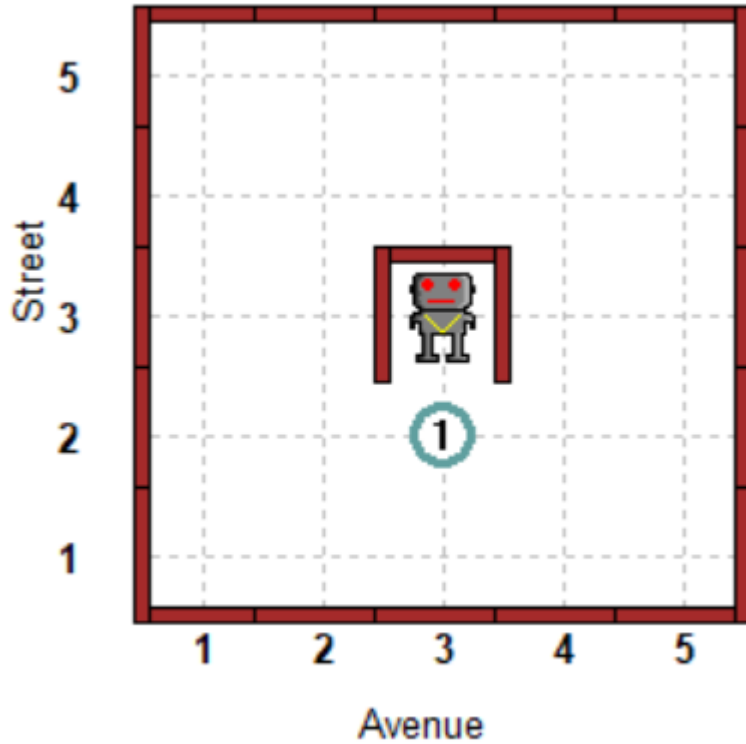
`print on_beeper()` True

`print carries_beeper` True/False

`print facing_north()` False

`turn_off()`

【러플의 센서 함수】 현재 위치하고 있는 월드의 상태를 알수 있는 함수



```
print front_is_clear()
```

```
print right_is_clear()
```

```
print left_is_clear()
```

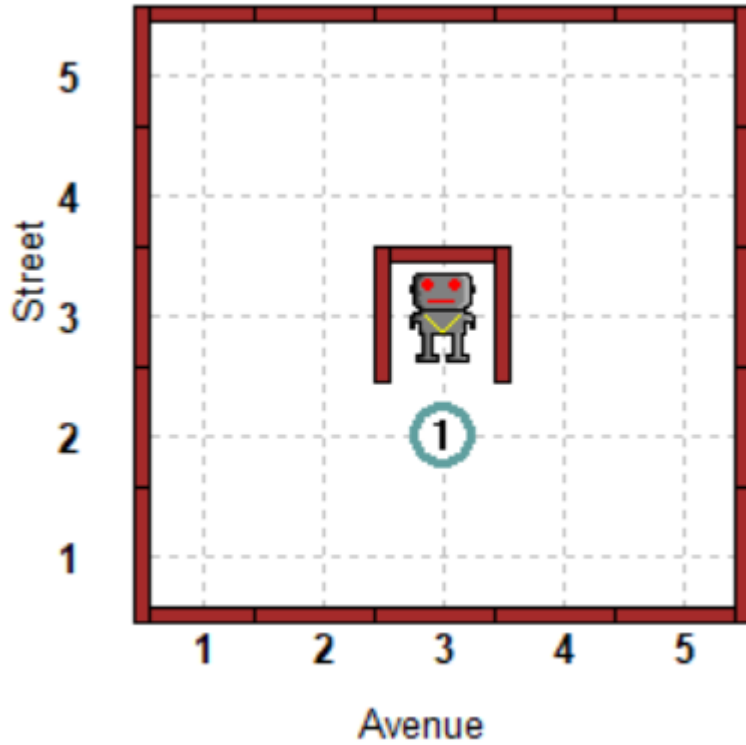
```
print on_beeper()
```

```
print carries_beeper()
```

```
print facing_north()
```

```
turn_off()
```

【러플의 센서 함수】 현재 위치하고 있는 월드의 상태를 알수 있는 함수



```
print front_is_clear()
```

True

```
print right_is_clear()
```

False

```
print left_is_clear()
```

False

```
print on_beeper()
```

False

```
print carries_beeper()
```

True/False

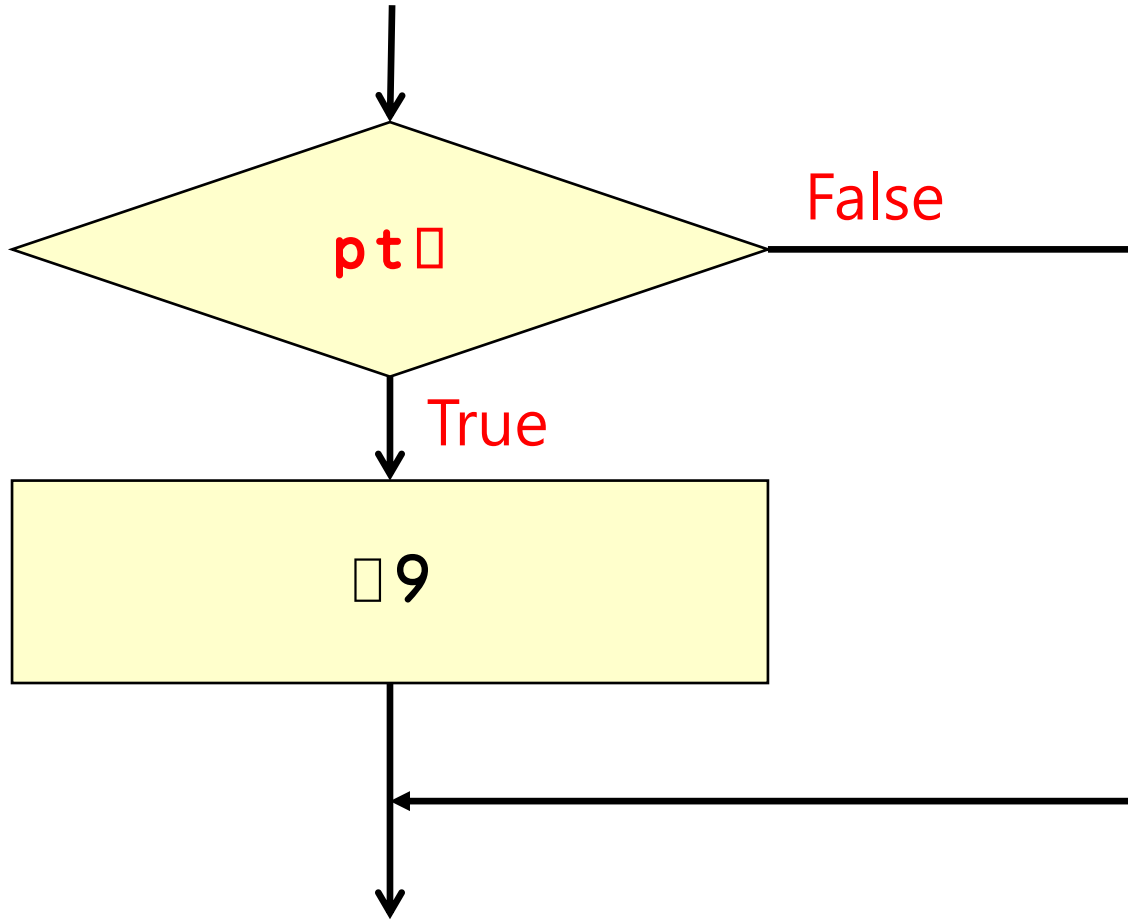
```
print facing_north()
```

False

```
turn_off()
```

#3. 조건문

[조건문] : 조건식이 True(참)일 때만 특정 명령을 실행



[조건문] : 조건식이 **True(참)일 때만 특정 명령을 실행**

조건문의 형식

if (조건식):

명령-1

명령-2

....

[조건문] : 조건식이 **True(참)**일 때만 특정 명령을 실행

조건문의 형식

if (조건식):

① : if예약어와 조건식을 써요!

명령-1

명령-2

② : 공백4칸
들여쓰기의 칸은 꼭 똑같이!

....

[조건문] : 조건식이 True(참)일 때만 특정 명령을 실행

조건문의 형식

if (조건식):

명령-1

명령-2

....

① : if예약어와 조건식을 써요!

<조건식은 반드시!>

True 또는 **False**로

표현되어야만 함!

[조건문] : 조건식이 True(참)일 때만 특정 명령을 실행

조건문의 형식

if (조건식):

명령-1

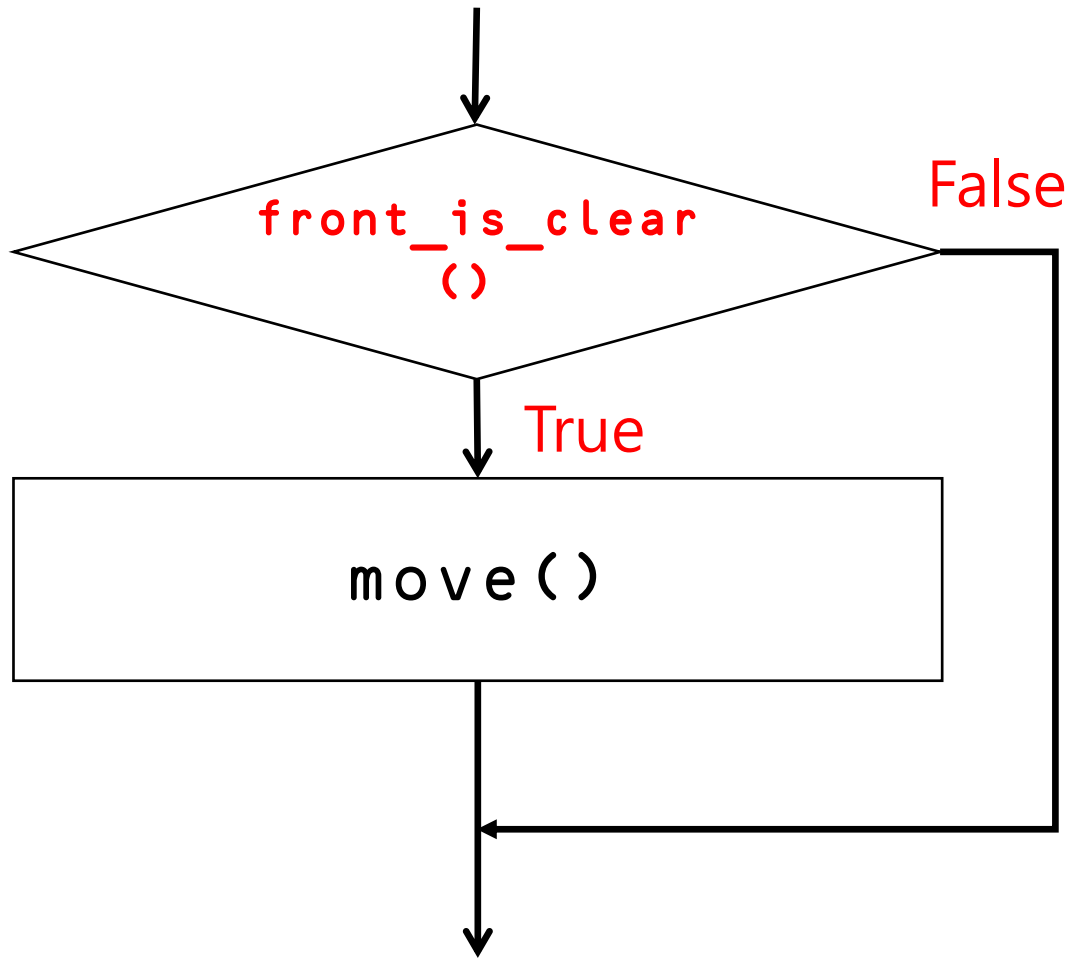
명령-2

....

② : 공백4칸
들여쓰기의 칸은 꼭 똑같이!

조건식이 True일 때만
들여쓰기 된 명령들을 실행

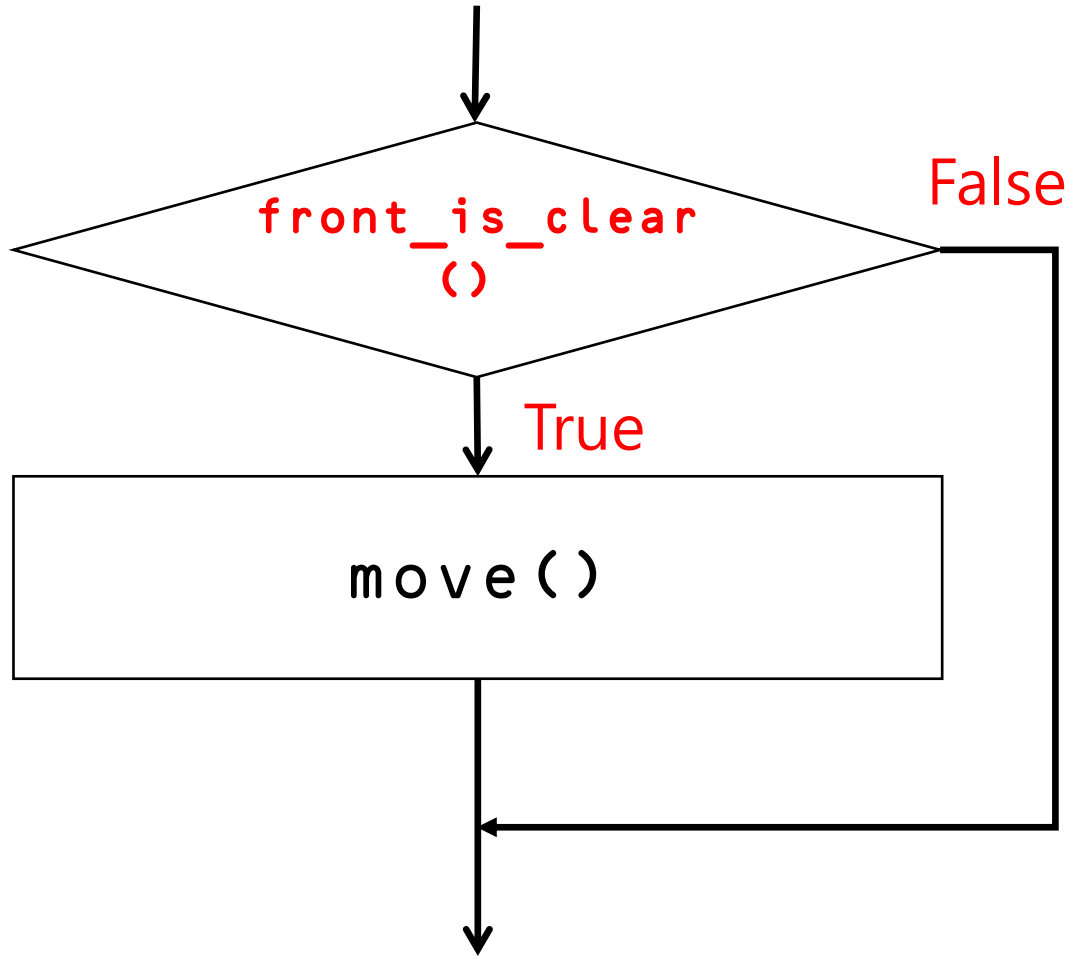
【조건문과 센서함수의 결합】



□ } ^□ D4□ Ä Æ□ t,

^< \ \ xL

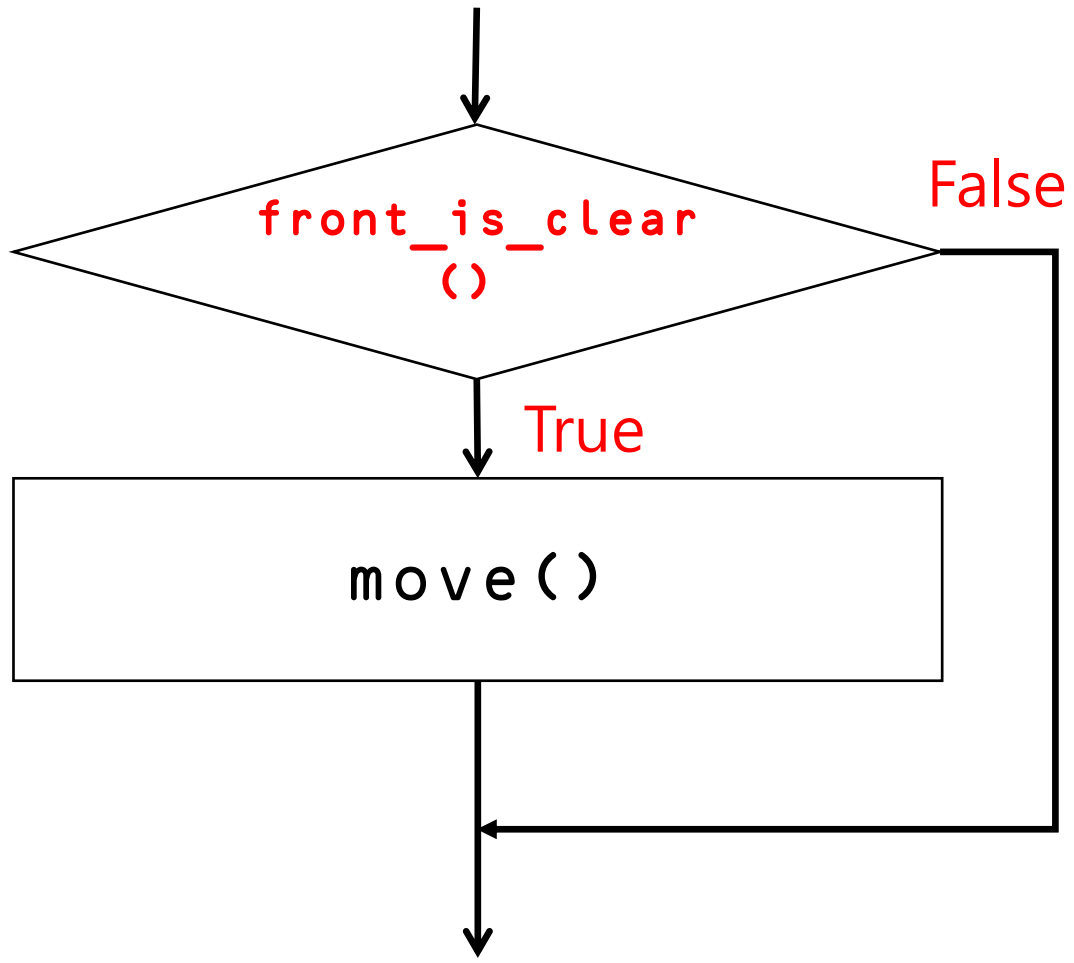
【조건문과 센서함수의 결합】



tl TÜ\

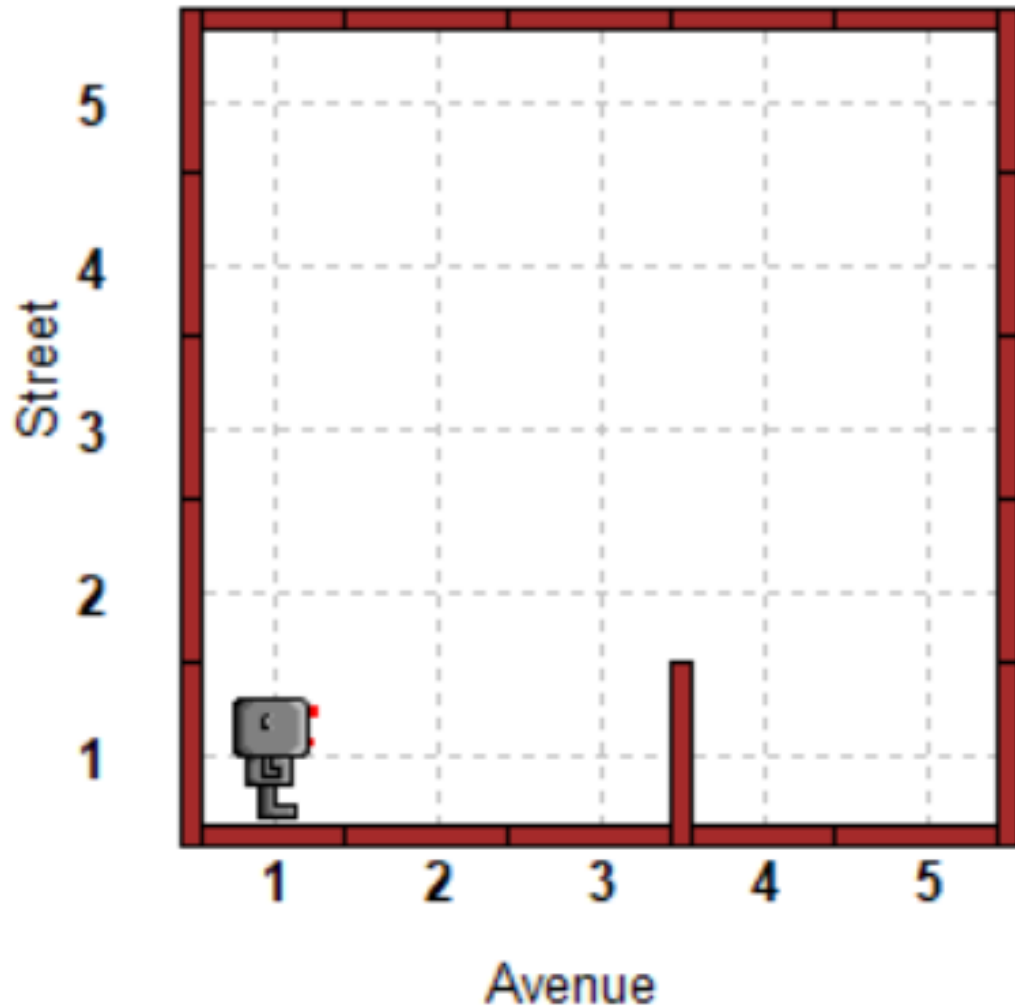
표현하면?

【조건문과 센서함수의 결합】



```
if (front is clear()) :  
    move()
```

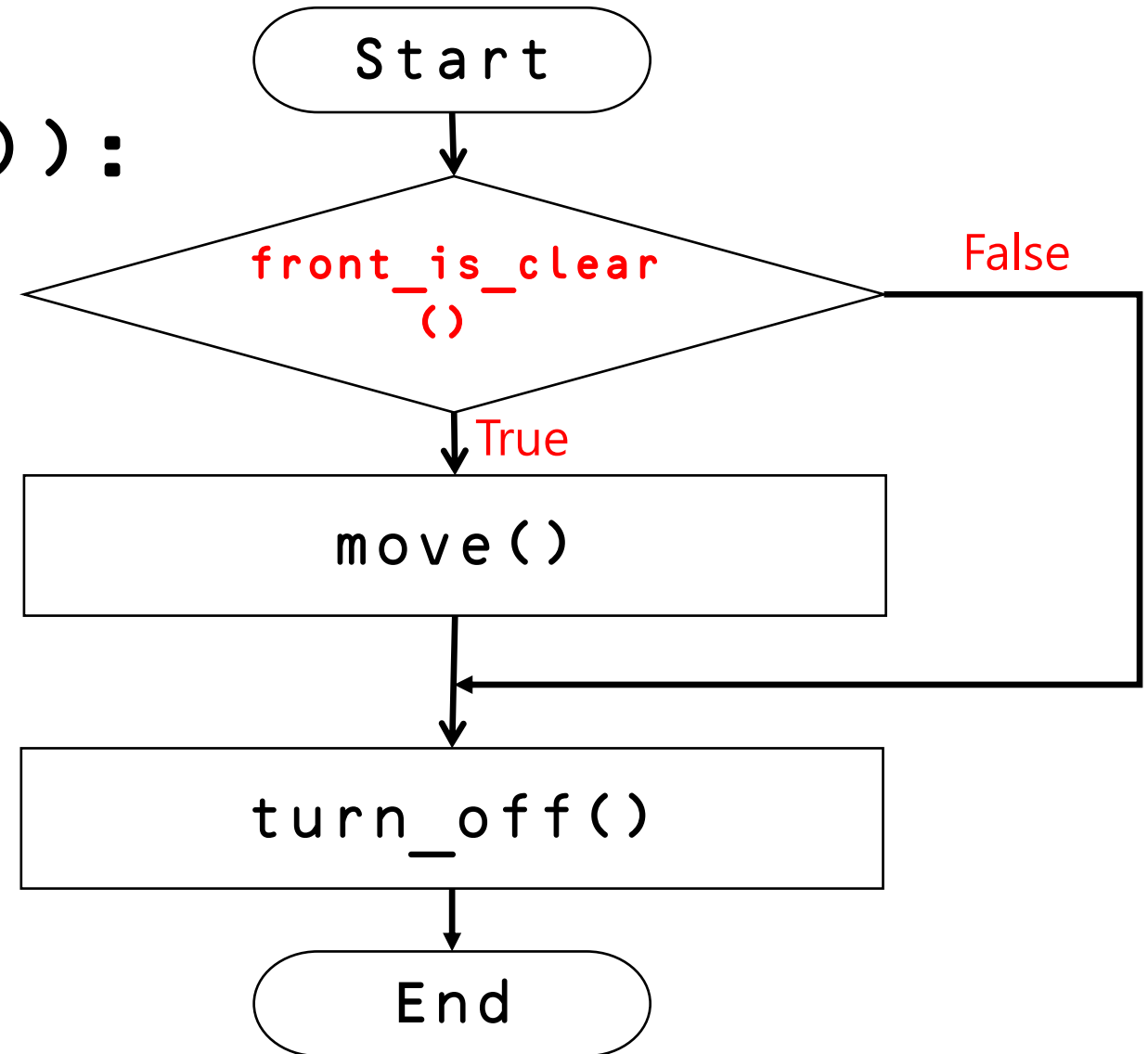
【조건문과 센서함수】 : 앞에 아무것도 없으면 한 걸음 앞으로!



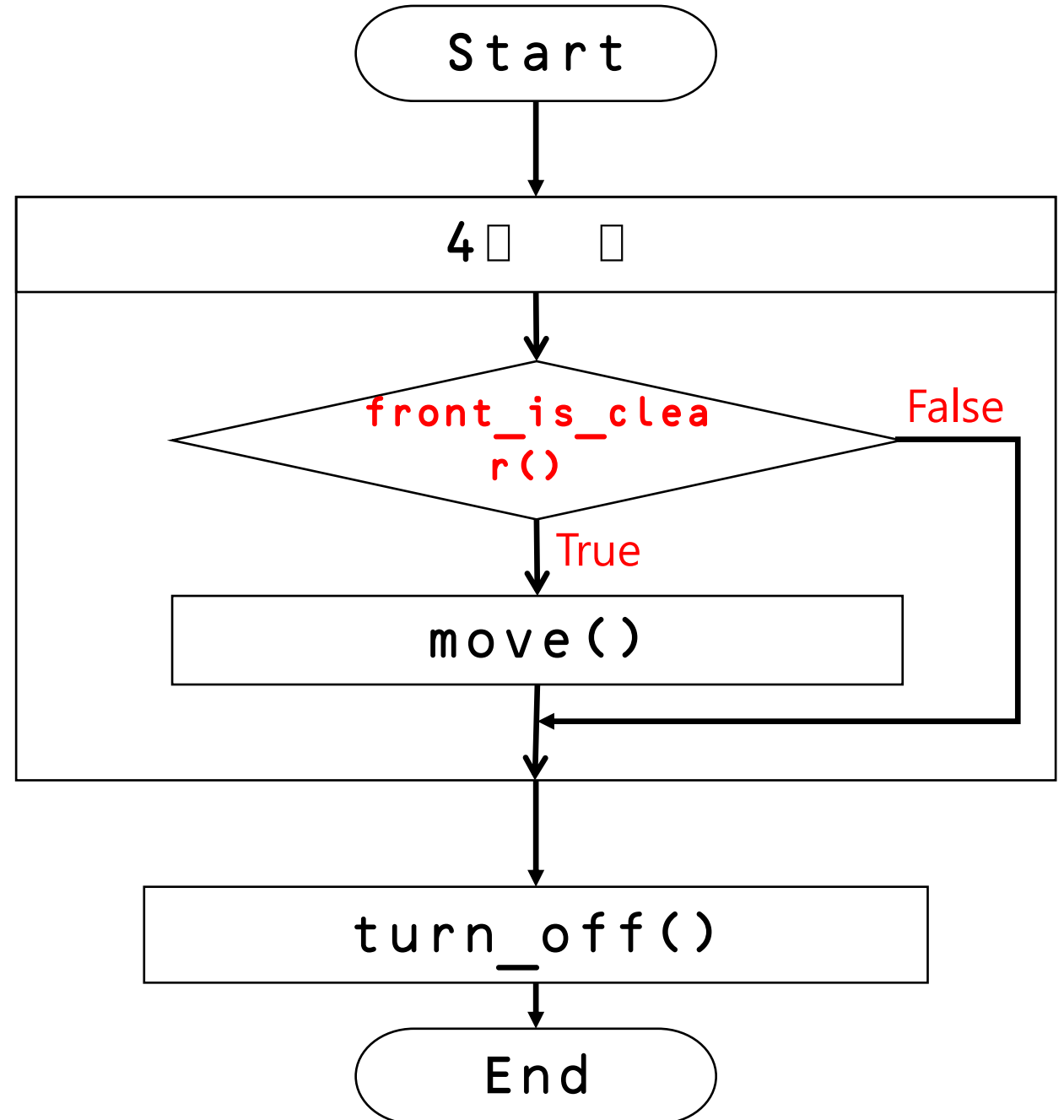
```
if (front_is_clear()):  
    move()  
  
turn_off()
```

【조건문과 센서함수】 : 앞에 아무것도 없으면 한 걸음 앞으로!

```
if (front_is_clear()):  
    move()  
  
turn_off()
```

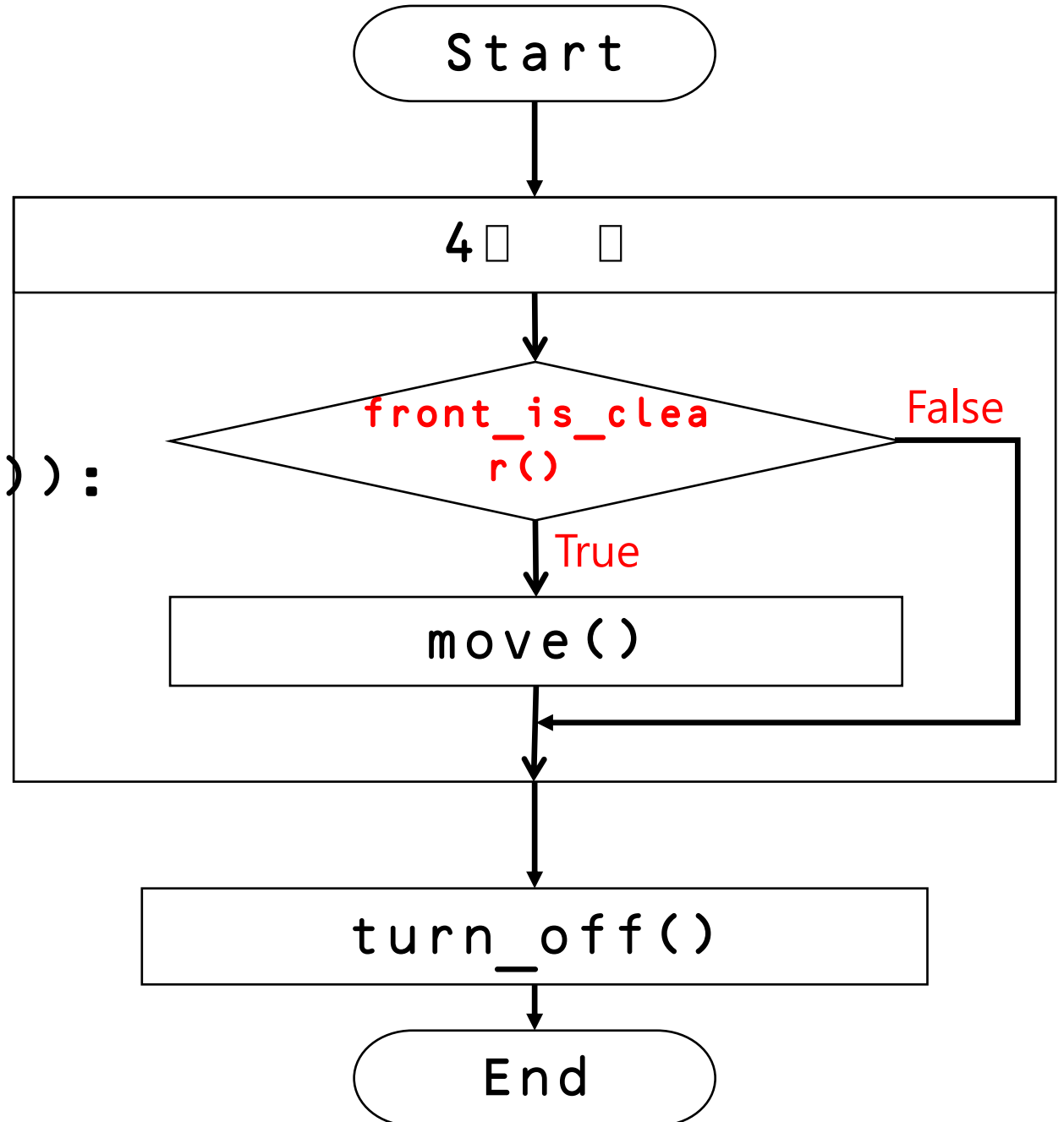


【조건문과 센서함수】



【조건문과 센서함수】

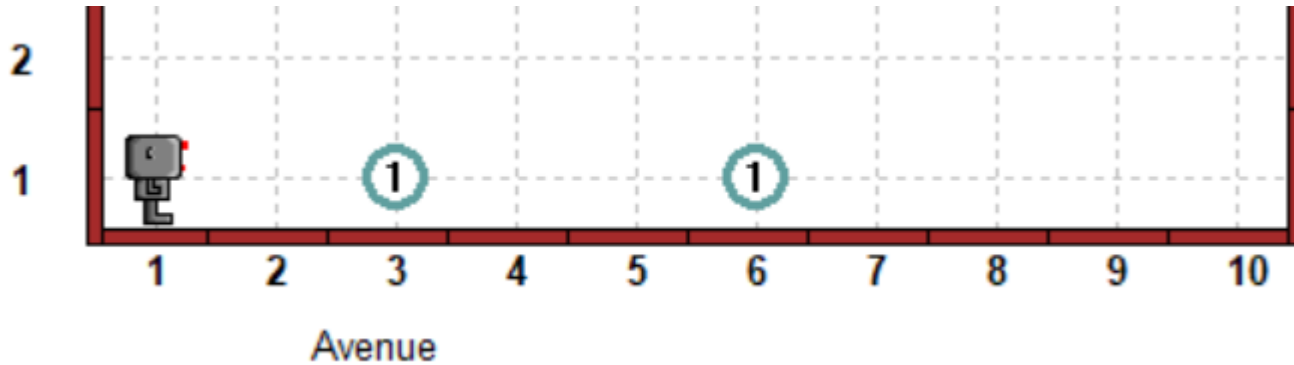
```
#define function  
  
def move_robot():  
    if (front_is_clear()):  
        move()  
  
#program start  
  
repeat(move_robot,4)  
turn_off()
```



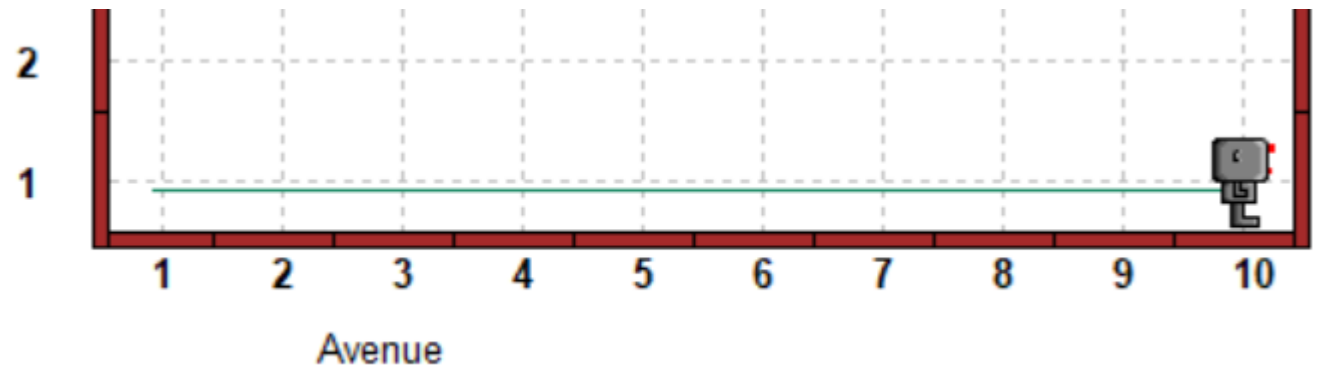
[Ex3_if]

1. 비퍼가 있는지 확인하세요
 2. 비퍼가 있다면 비퍼를 주워요!
 3. 앞으로 한 칸 이동하세요.
- 1~3을 9번 반복해요.

<실행 전>

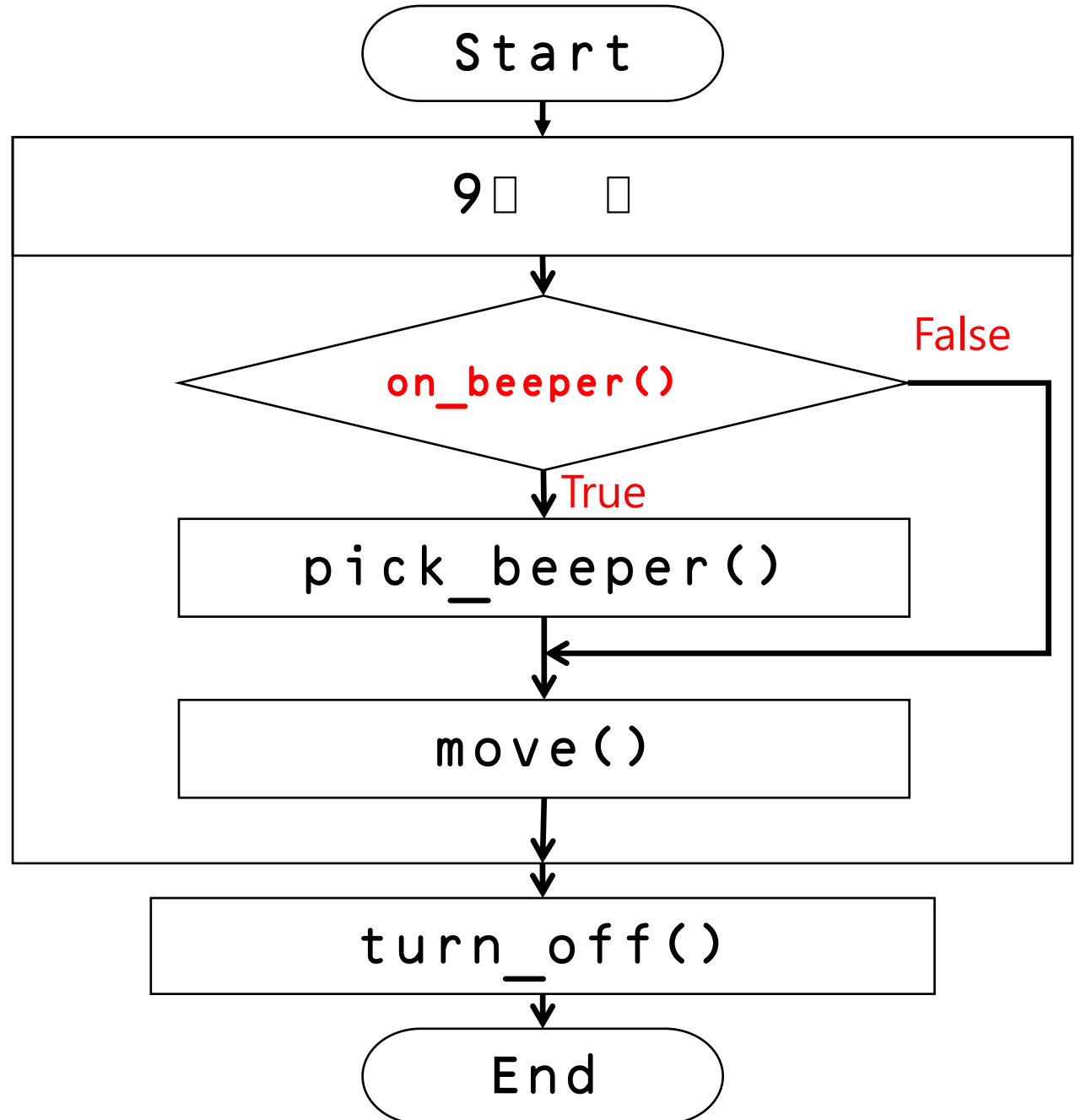


<실행 후>



[Ex3_if-Hint / 순서도]

1. 비퍼가 있는지 확인하세요
 2. 비퍼가 있다면 비퍼를 주워요!
 3. 앞으로 한 칸 이동하세요.
- 1~3을 9번 반복해요.



[Ex3_if : Hint / 코드 완성하기]

```
#define function
def move_and_pick():
    . . . 만약, 비퍼가 있다면 :
    . . . . . 비퍼를 주워요.
    . . . 앞으로 한 걸음 가요

#program start
repeat(move_and_pick, 9)
turn_off()
```

[Ex3_if : Hint]

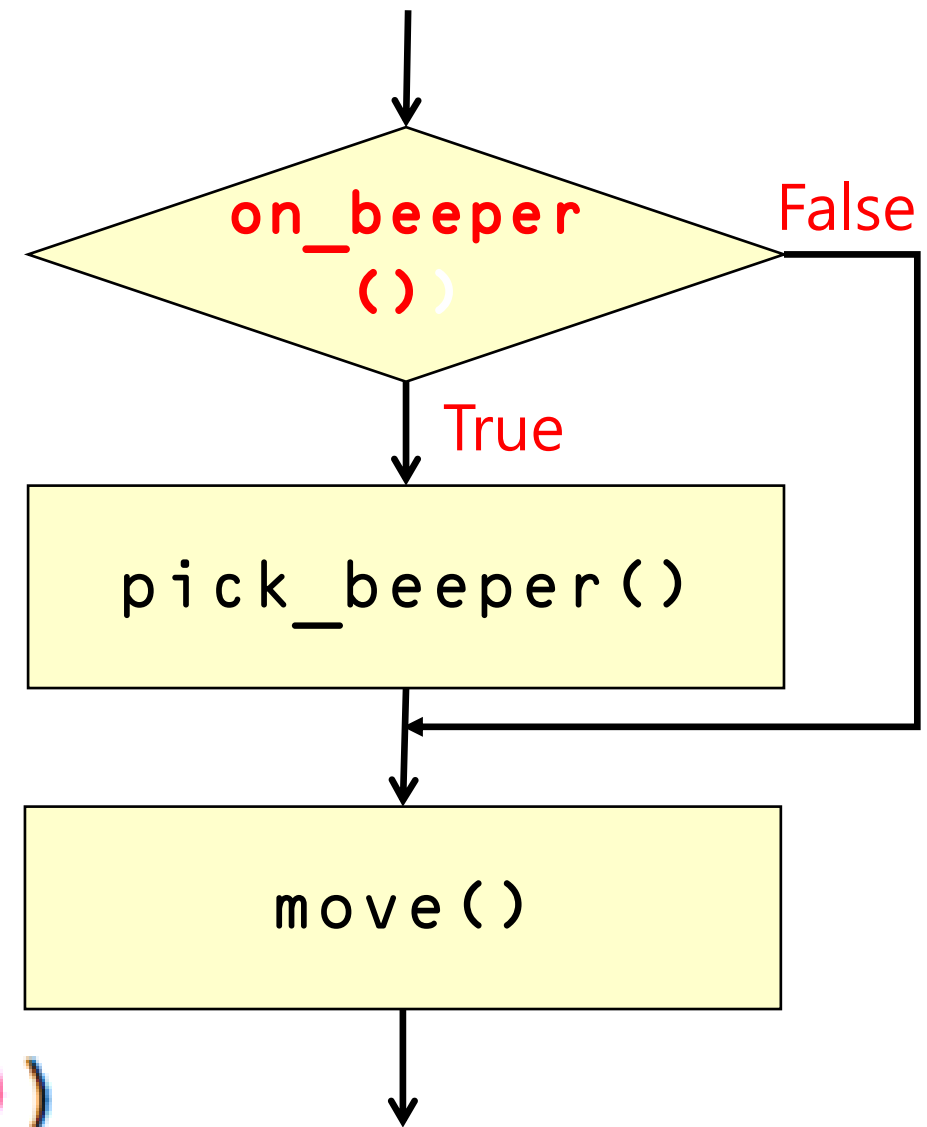
```
#define function  
def move_and_pick():
```

```
    만약, 비퍼가 있다면 :
```

```
    비퍼를 주워요.
```

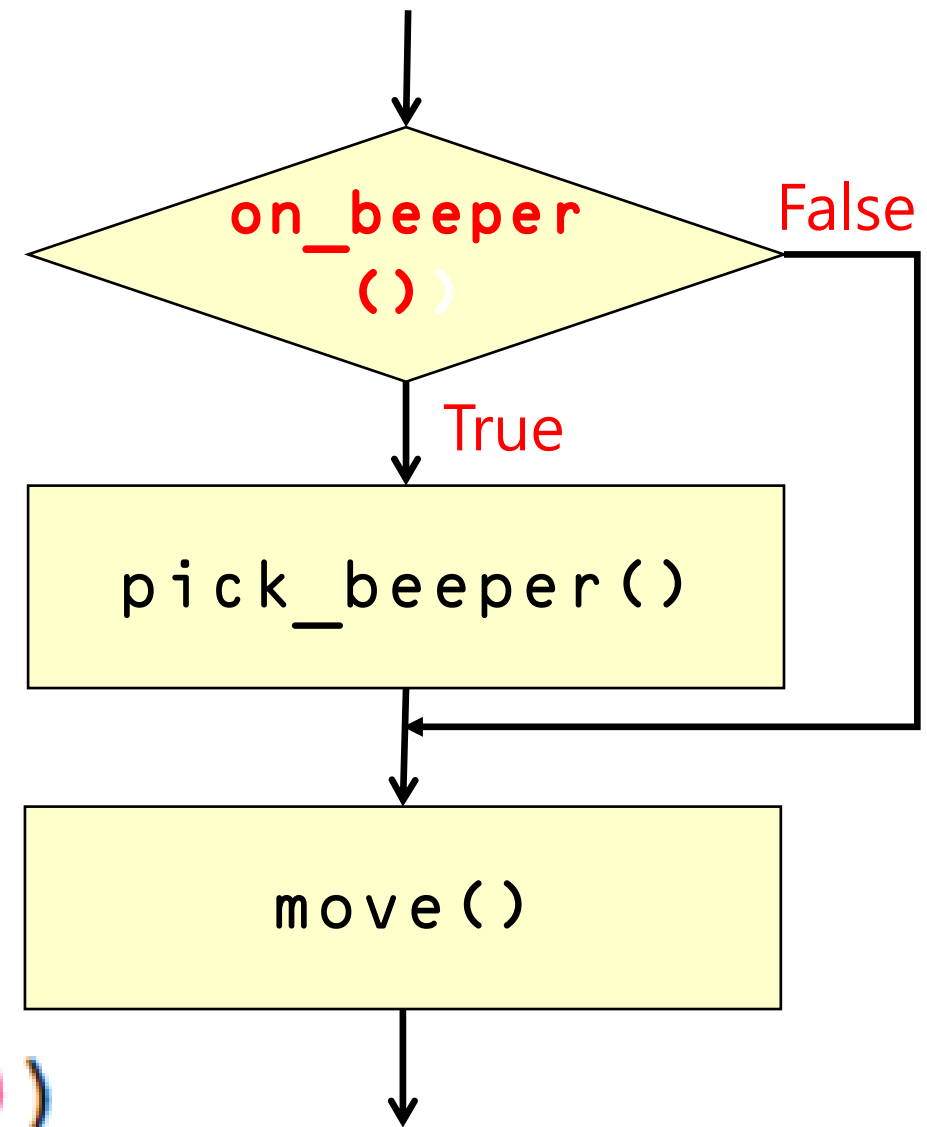
```
    앞으로 한 걸음 가요
```

```
#program start  
repeat(move_and_pick, 9)  
turn_off()
```



[Ex3_if : 완성]

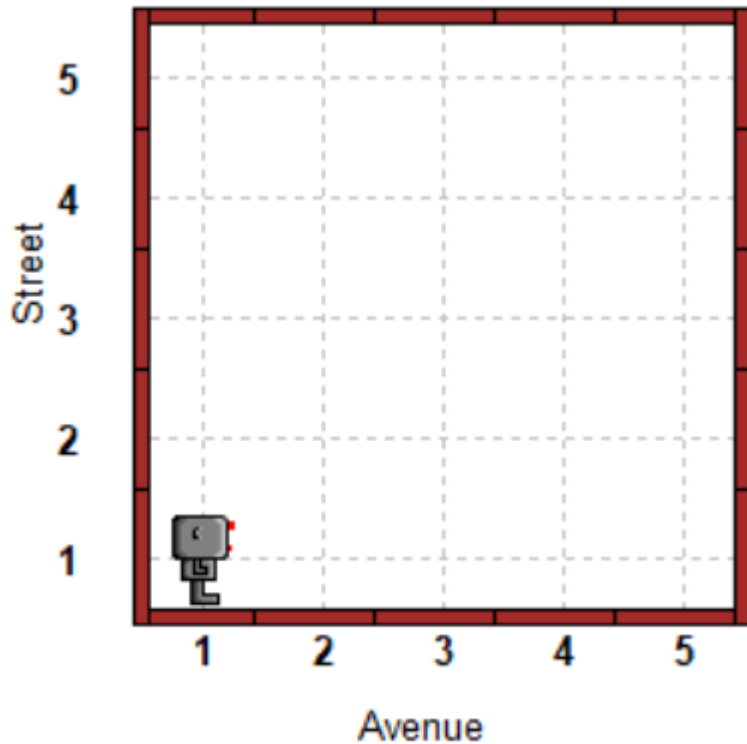
```
#define function
def move_and_pick():
    . . . if (on_beeper()):
    . . .     . . . pick_beeper()
    . . .     move()
#program start
repeat(move_and_pick, 9)
turn_off()
```



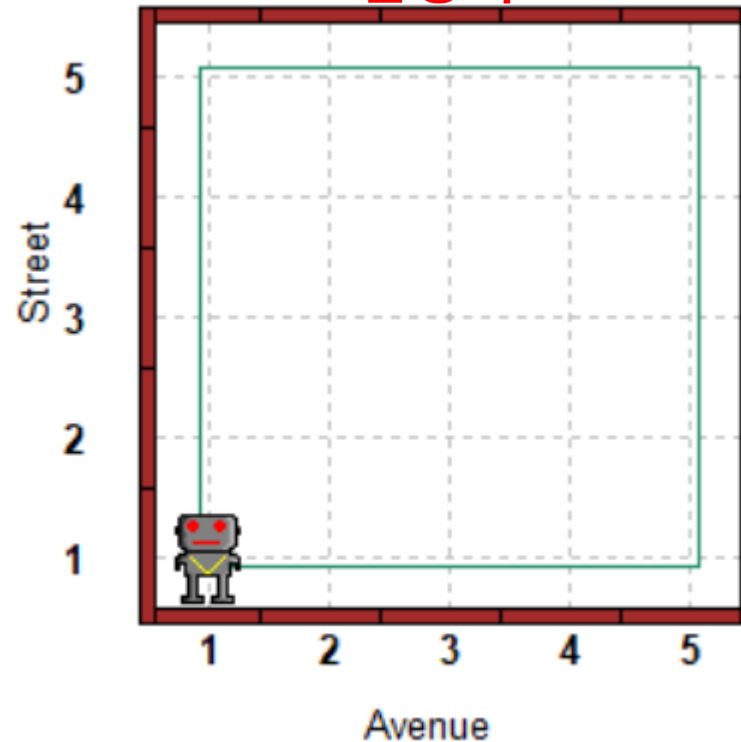
리보그의 산책

- 앞에 벽이 있는지 판단 후,
- 앞으로 한 칸 이동하세요.

<실행 전>



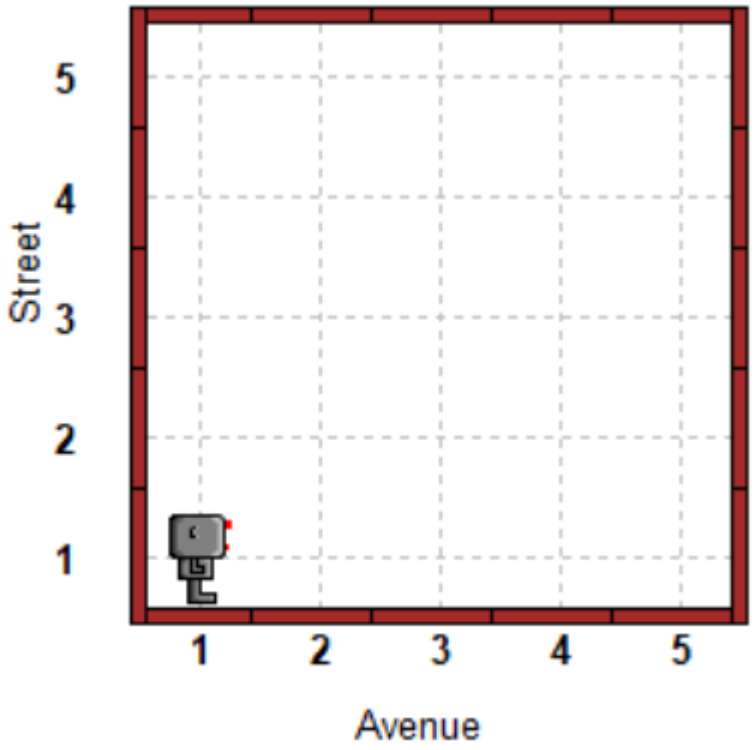
<실행 후>



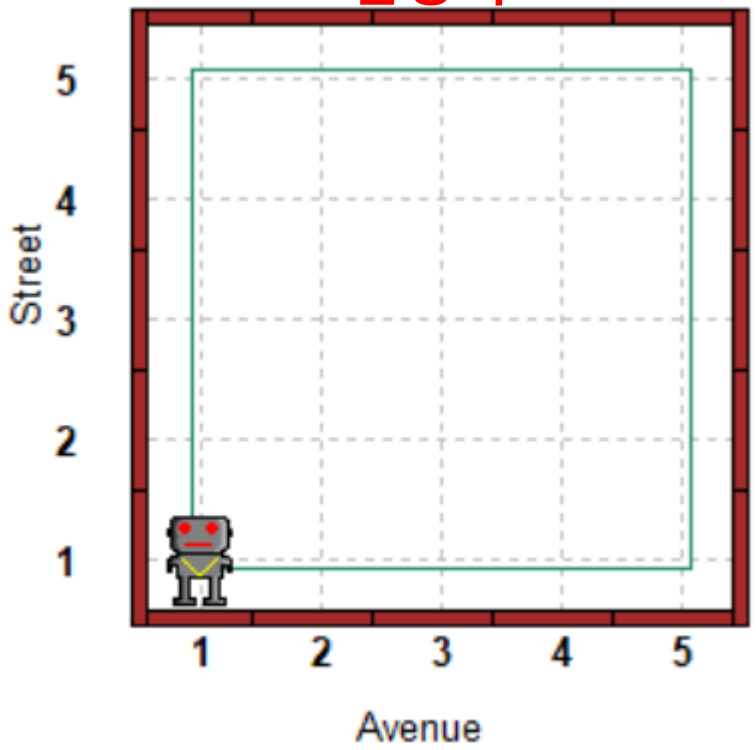
#리보그의 산책 : **not** 예약어

- 앞에 벽이 있는지 판단 후, `□□∅X ^t D` ^□ J<t |□□0`
- 앞으로 한 칸 이동하세요.

<실행 전>



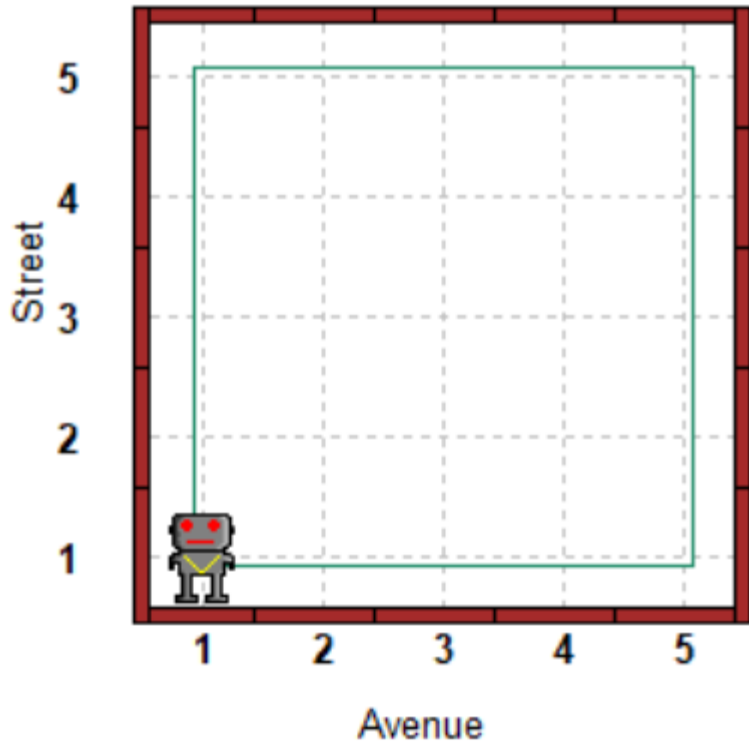
<실행 후>



#리보그의 산책 : **not** 예약어

- 앞에 벽이 있는지 판단 후,
- 앞으로 한 칸 이동하세요.

```
if not X ^ t D ^ J < t | 0
```



```
#define function  
def move_and_turn():
```

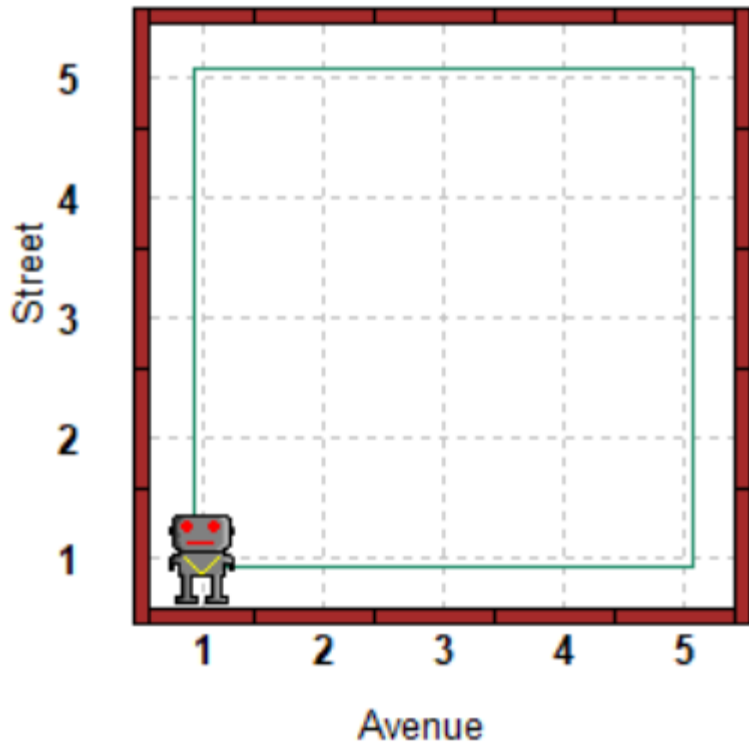
```
    만약, 앞이 비어 있지 않으면 :  
    왼쪽으로 돌아요  
    move()
```

```
#program start  
repeat(move_and_turn, 16)  
turn_off()
```

#리보그의 산책 : **not** 예약어

- 앞에 벽이 있는지 판단 후,
- 앞으로 한 칸 이동하세요.

```
□□∅X ^t D` ^□ J<t |□□0
```



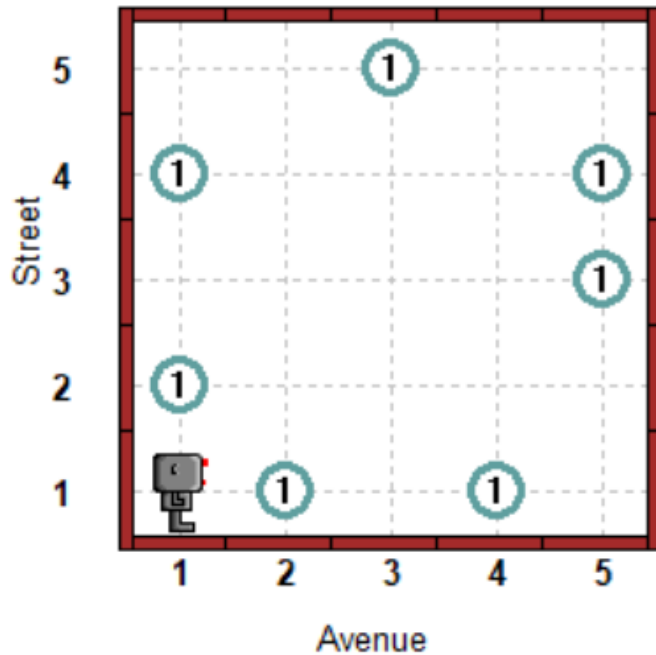
```
#define function
def move_and_turn():
    . . . if not front_is_clear():
    . . .     . . . turn_left()
    . . . move()

#program start
repeat(move_and_turn, 16)
turn_off()
```

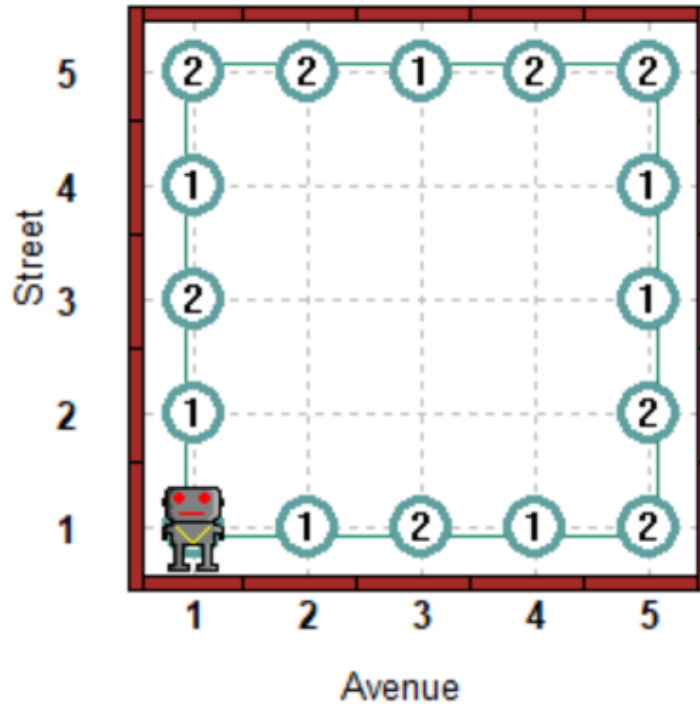
[Ex4_if] 리보그의 산책 : 비퍼를 채워요!

- 비퍼가 놓여있는지 판단한 후, 비퍼가 없는 곳은 비퍼를 2개씩 내려 놓아요
- 새로 놓은 비퍼의 개수를 출력해 주세요!

<실행 전>

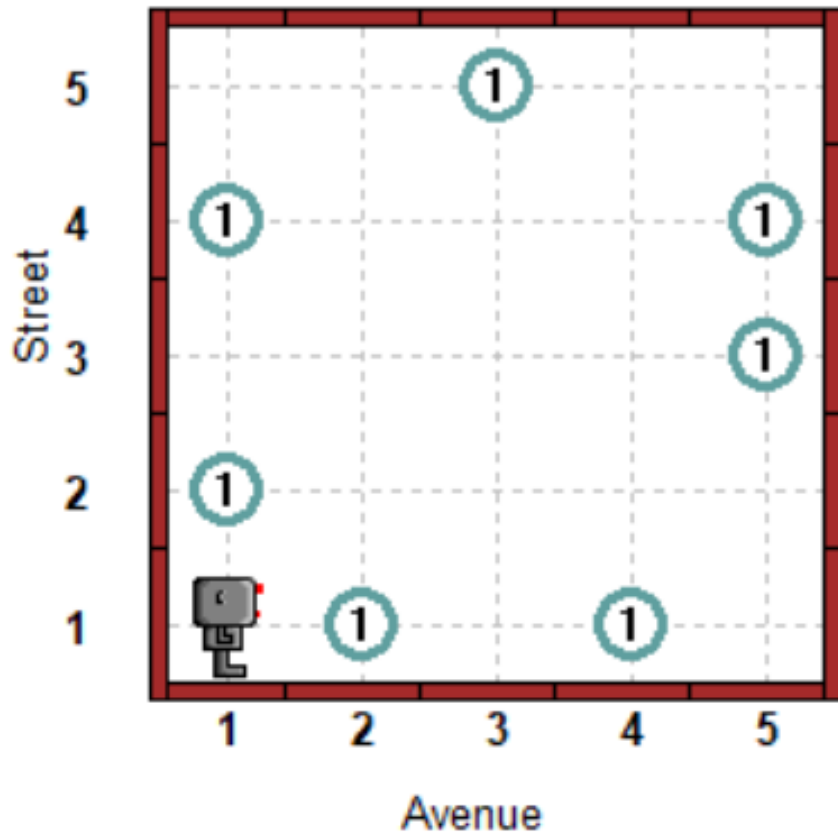


<실행 후>



[Ex4_if : Hint / 구문] 리보그의 산책

- 비퍼가 놓여있는지 판단한 후, 비퍼가 없는 곳은 비퍼를 2개씩 내려 놓아요

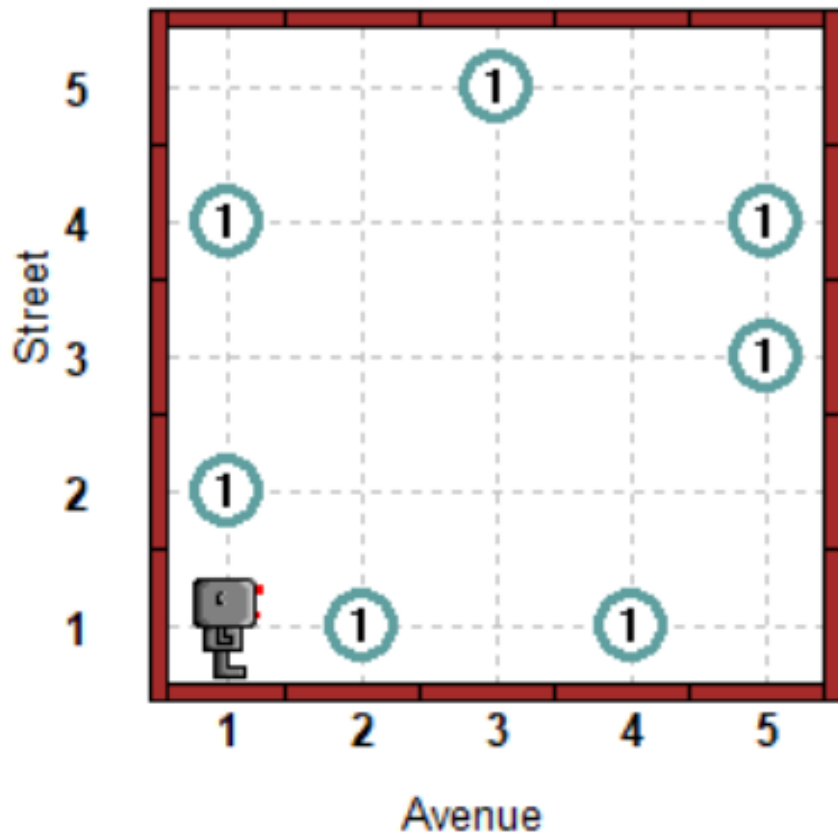


```
#define function
def move_and_turn():
    ... if not front_is_clear():
    ...     turn left()
    ... 만약, 비퍼가 놓여있지 않으면 :
    ...     repeat (put_beeper, 2)
    ... move()
```

```
#program start
repeat (move_and_turn, 16)
turn_off()
```

[Ex4_if : Hint / 구문] 리보그의 산책

- 비퍼가 놓여있는지 판단한 후, 비퍼가 없는 곳은 비퍼를 2개씩 내려 놓아요



```
#define function
def move_and_turn():
    ... if not front_is_clear():
    ...     turn_left()
    ... if not on_beeper():
    ...     repeat (put_beeper, 2)
    ... move()
```

```
#program start
repeat (move_and_turn, 16)
turn_off()
```

[Ex4_if : Hint / 변수설정] 리보그의 산책

- 비퍼를 놓은 개수를 세기

```
#define function
def move_and_turn():
    ... if not front_is_clear():
    ...     turn_left()
    ... if not on_beeper():
    ...     put_beeper()
    ... move()
```

비퍼를 내려놓을 때마다 숫자를 세 주면 된다

```
#program start
repeat(move_and_turn, 16)
turn_off()
```

[Ex4_if : 완성] 리보그의 산책

```
#declare variable
beeperCnt=0
#define function
def move_and_turn():
    ... global beeperCnt
    ... if not front_is_clear():
    ...     turn_left()
    ...     if not on_beeper():
    ...         repeat(put_beeper, 2)
    ...         beeperCnt+=2
    ... move()
```

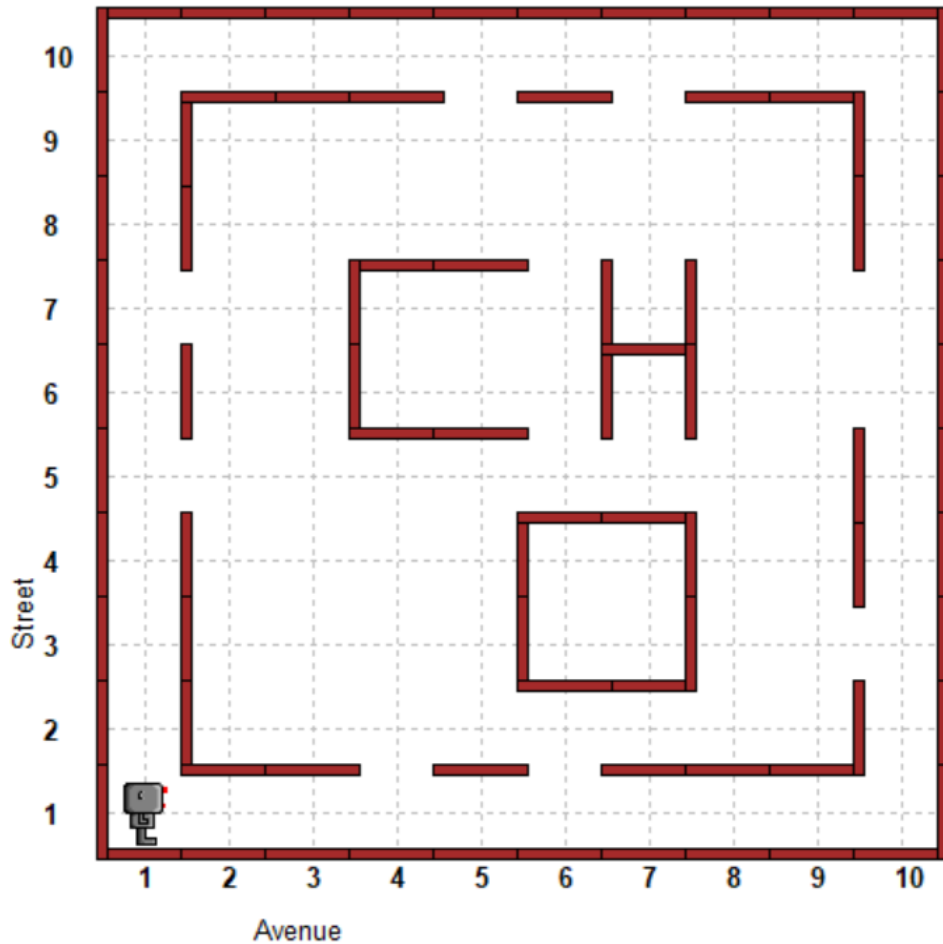
비퍼를 내려놓을 때마다 숫자를 세 주면 된다

```
#program start
repeat(move_and_turn, 16)
print(beeperCnt)
turn_off()
```

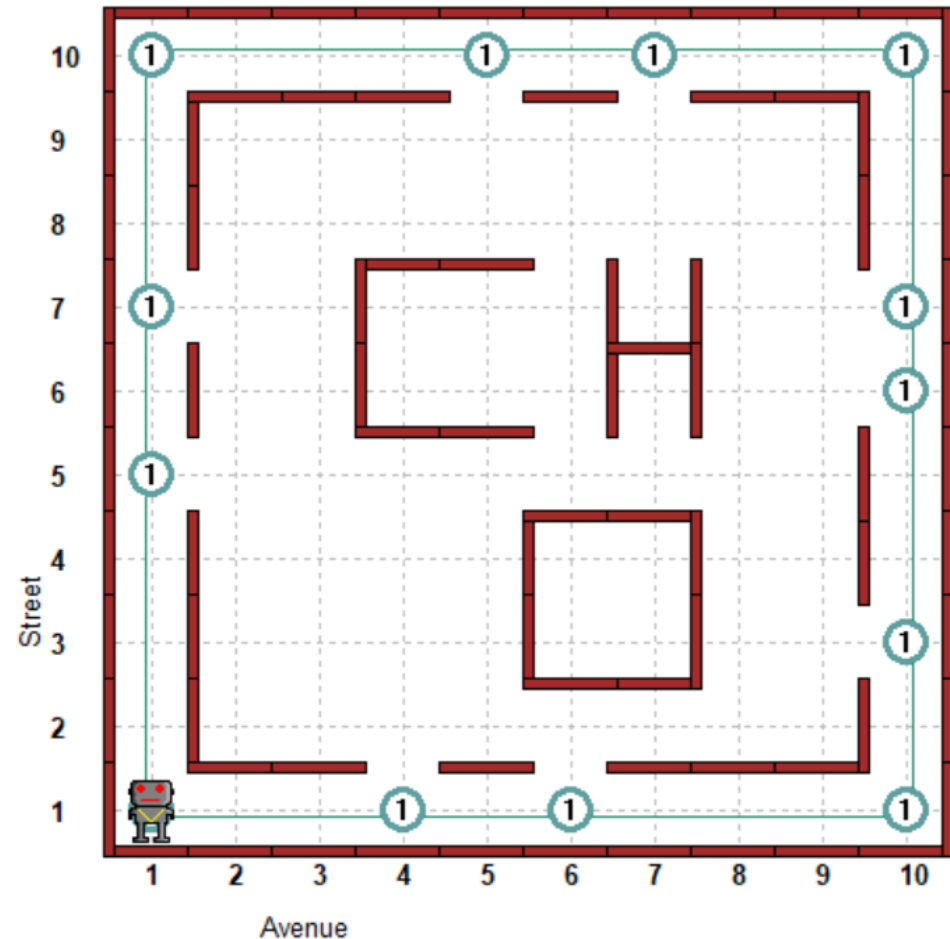
[Ex5_if_flood] : 홍수를 막아라!!

댐 주위를 돌면서 구멍이 있는 부분을 비퍼로 막아주세요!

<실행 전>



<실행 후>



[수업 정리]

1. 구글 드라이브 접속
2. <2019_Python_학번_이름> 폴더에
<오늘날짜_변수, 센서함수, 조건문>폴더를 생성
예) 0507_조건문
3. 오늘 실습한 파일을 모두 업데이트(총 10개 파일)
Ex1~Ex5까지 wld파일, rur파일
4. 수업 피드백 작성 :
<https://forms.gle/6b4PP7hxJM9p4i316>

【다음 시간에는】

**좀 더 다양한 조건들을 해결할 수 있도록
리보그를 업그레이드 시켜주자!**