

# SIT22001 **PROGRAMMING I**

## Lecture 3

Fall 2019

School of Global Entrepreneurship & ICT  
Handong Global University

## Characteristics of Python

### Instruction set

Arithmetic and logical operations

+, -, \*, /, and \*\*

and, or, not

Assignment

Conditionals

Iterations

Input/output

} for defining  
expressions

**No pointers**

**No explicit declarations**

A small grid-like 2D world

Basic actions

move (): moving one grid forward

turn\_left (): turning left by 90°

pick\_beeper(): pick ing up beepers

drop\_beeper(): putting down beepers

set\_trace("blue")

set\_pause(0.3)

**Our own instructions: functions**

Comments

Conditionals

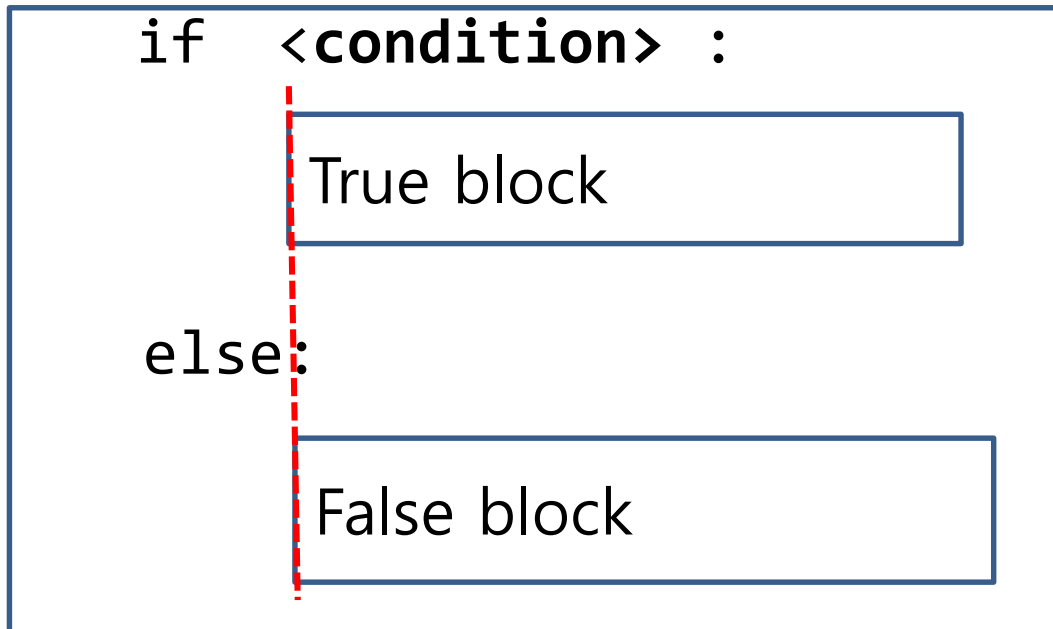
Iterations

for\_loops

while\_loops

Reading assignment:

Chapter 2 of the text book



<**condition**> has a "True" or "False" value, representing true or false, respectively.

If it is true, the **True block** is executed; otherwise, the **False block** is executed.

What will be printed ?

```
if True:  
    print ("SIT22001 is my favorite course.")  
else:  
    print ("Every SIT22001 student will receive an A+.")
```

Now, do you understand?

```
if 5 > 3:  
    print ("SIT22001 is my favorite cours.")  
else:  
    print ("Every SIT22001 student will receive an A+.")
```

Now, what will happen?

```
if False:  
    print ("SIT22001 student will receive an A+.")
```

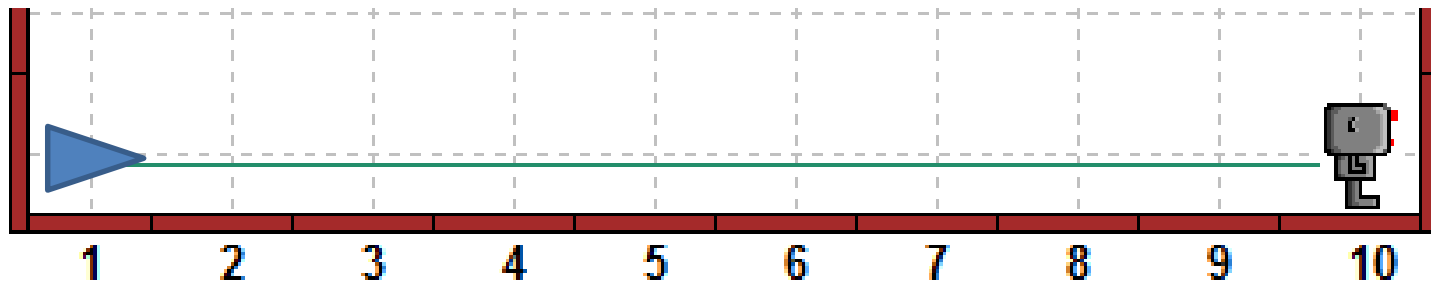
The keyword `not` inverts the sense of the condition: `not True` is False, and `not False` is True.

What is the output?

```
>>>print (not 3 < 5)
```

False

We want Hubo to make 9 steps and pick all **beepers** on the way if there is any. However, we do not know where beepers are. If there is **no beeper** at a grid point, then `hubo.pick_beeper()` causes an error.



How to sense a beeper?

`hubo.on_beeper()`



Move forward 9 steps.  
At each step, **move**  
**and pick up a beeper**  
**if any.**

```
for i in range(9):  
    move_and_pick()
```

**Move and pick a beeper if any.**

Take a step forward.  
Check if there is a beeper.  
If yes, pick it up.

```
def move_and_pick():  
    hubo.move()  
    if hubo.on_beeper():  
        hubo.pick_beeper()
```

No False block!!

---

Let's do the opposite: we want to drop a beeper, but only if there is no beeper at the current location.

```
if not hubo.on_beeper():  
    hubo.drop_beeper()
```

# FOLLOWING THE BOUNDARY

SIT22001 HGU

Hubo tries to follow the boundary of the world: He moves forward if there is no wall; otherwise, turn to the left.

1

```
from cs1robots import *  
create_world(avenues =5, streets = 5)  
hubo = Robot()  
hubo.set_trace("blue")  
Hubo.set_pause(0.2)
```

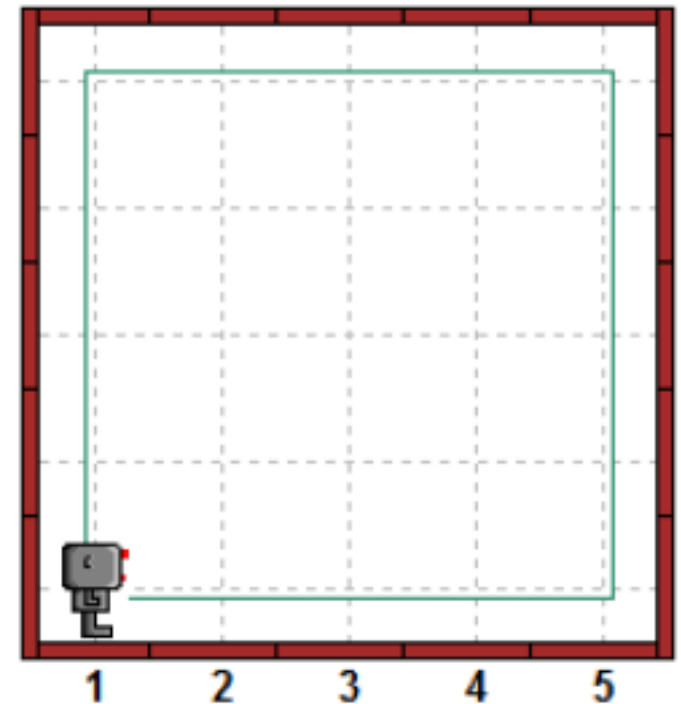
3

```
def move_or_turn():  
    if hubo.front_is_clear():  
        hubo.move()  
    else:  
        hubo.turn left()
```

2

```
for i in range(20):  
    move_or_turn()
```

**Why 20 ?**



Using a for-loop

# DANCING AND MOVING

SIT22001 HGU

```
from cs1robots import *  
create_world(avenues = 5, streets = 5)  
hubo = Robot(beepers=4)  
hubo.set_trace("blue")  
hubo.set_pause(0.2)
```

```
def dance():  
    for i in range(4):  
        hubo.turn_left()
```

```
def move_or_turn():  
    if hubo.front_is_clear():  
        dance()  
        hubo.move()  
    else:  
        hubo.drop_beeper()  
        hubo.turn_left()
```

```
for i in range(18):  
    move_or_turn()
```

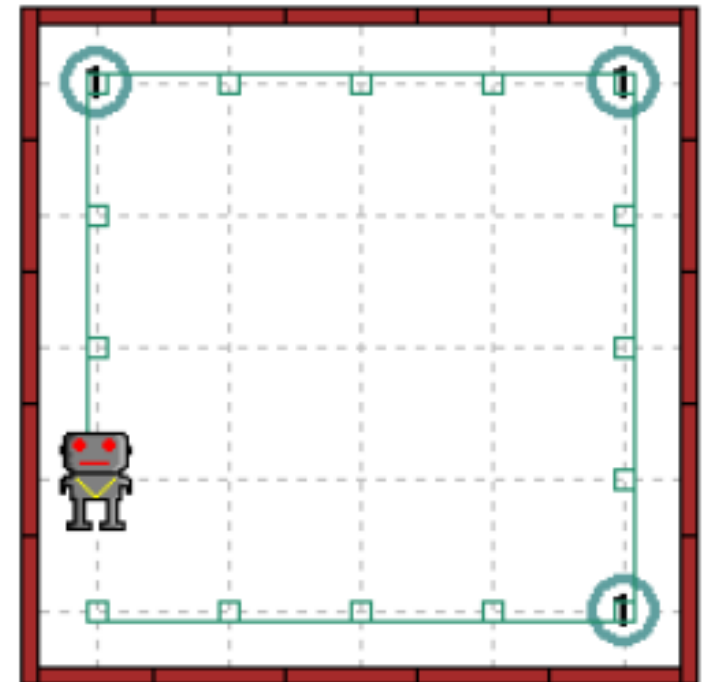
Why 18?

1

4

3

2



`elif` combines `else` and `if` to express many alternatives without complicated indentation.

```
if hubo.on_beeper():
    hubo.pick_beeper()
elif hubo.front_is_clear():
    hubo.move()
elif hubo.left_is_clear():
    hubo.turn_left()
elif hubo.right_is_clear():
    turn_right()
else:
    turn_around()
```

```
if hubo.on_beeper():
    hubo.pick_beeper
else:
    if hubo.front_is_clear():
        hubo.move()
    else:
        if hubo.left_is_clear():
            hubo.turn_left()
        else:
            if -----
```

while <**condition**>:



block

The diagram illustrates the structure of a while loop. It consists of a blue-outlined rectangular box containing the word 'block'. To the left of the box is a vertical red line, and to the right of the box is a vertical blue line, representing the loop's boundaries.

The block is executed as long as <**condition**> is True; otherwise, it is skipped.

A while-loop repeats instructions as long as a condition is true.

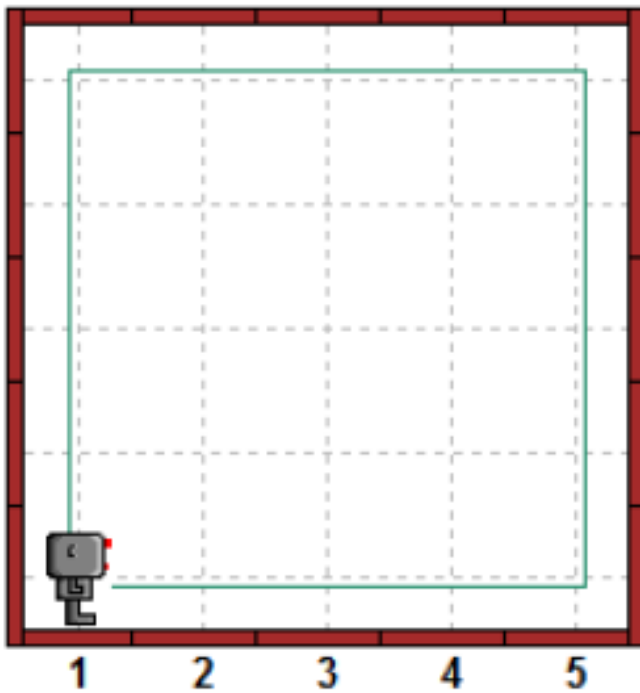
```
while not hubo.on_beeper():  
    hubo.move()
```

Move forward as long  
as there is no beeper

A for-loop repeats some instructions a fixed number of times.

```
for i in range(9):  
    hubo.move()
```

Let's write a program to let the robot **walk around the boundary of the world** until he comes back to the **starting point**



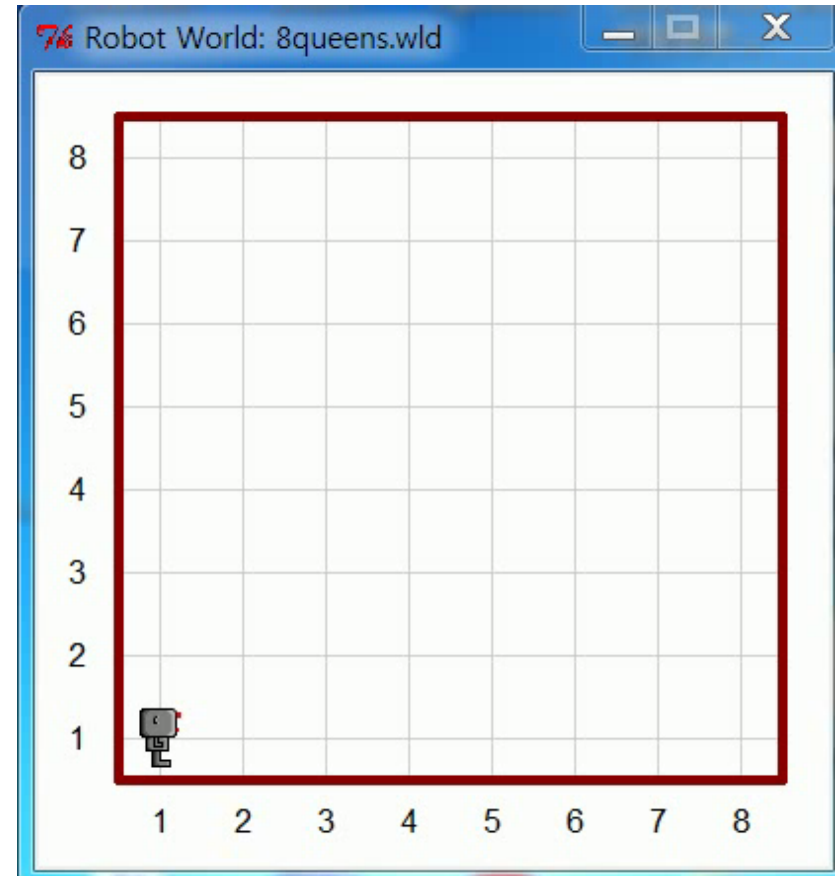
1. Put down a beeper to mark the starting point.
2. Repeat steps 2.1 and 2.2 while no beeper is found:
  - 2.1. If not facing a wall, move forward.
  - 2.2. Otherwise, turn left.
3. **Finish** when we found the beeper.

Facing east



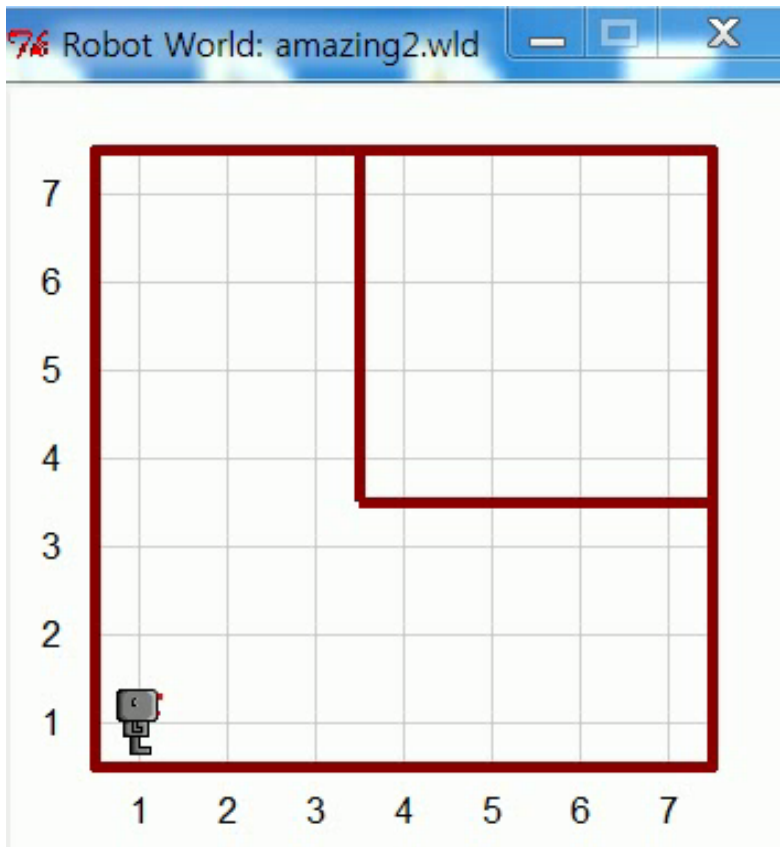
```
from cs1robots import *  
create_world(avenues =8, streets = 8)  
hubo = Robot(beepers=1)  
hubo.set_trace("blue")  
Hubo.set_pause(0.2)
```

```
hubo.drop_beeper()  
hubo.move()    Why this?  
while not hubo.on_beeper():  
    if hubo.front_is_clear():  
        hubo.move()  
    else:  
        hubo.turn_left()  
hubo.turn_left()
```



Does this program always work?  
Well, .....

## How about this case?



amazing2.wld

```
hubo.drop_beeper()  
hubo.move()  
while not hubo.on_beeper():  
    if hubo.front_is_clear():  
        hubo.move()  
    else:  
        hubo.turn_left()  
hubo.turn_left()
```

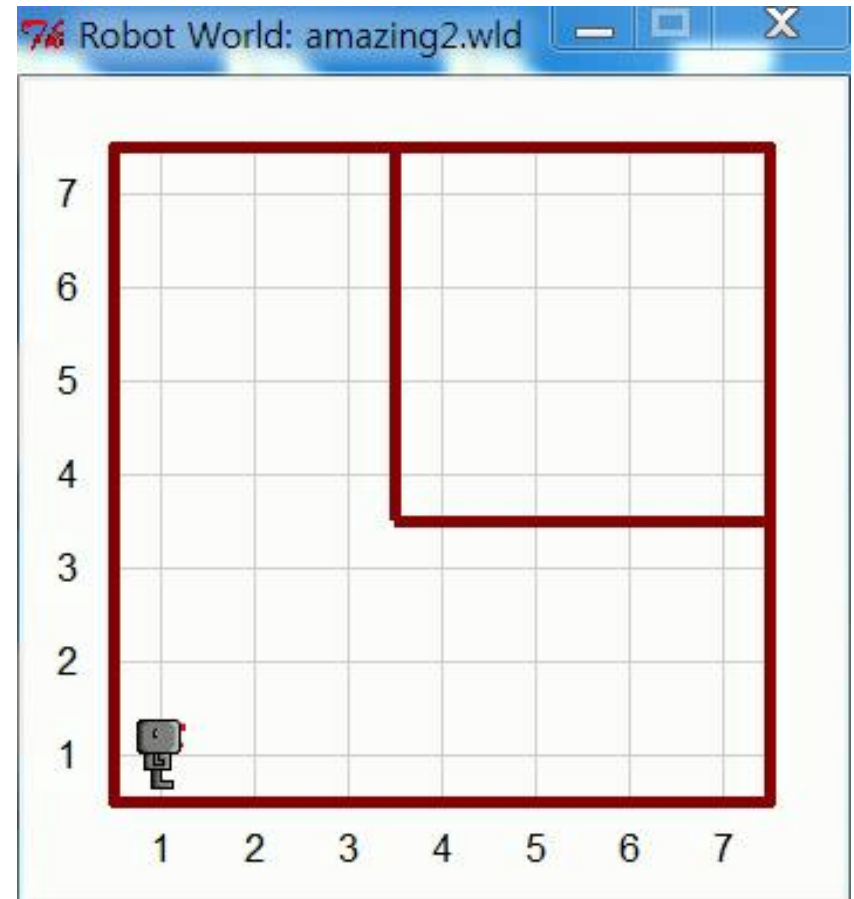
Try the code in the previous page with "amazing2.wld" and see if the previous code works.

Sometimes we need a right turn!

## Does this work?

Well, .....

```
hubo.drop_beeper()
hubo.move()
while not hubo.on_beeper():
    if hubo.right_is_clear():
        turn_right()
    elif hubo.front_is_clear():
        hubo.move()
    else:
        hubo.turn_left()
hubo.turn_left()
```

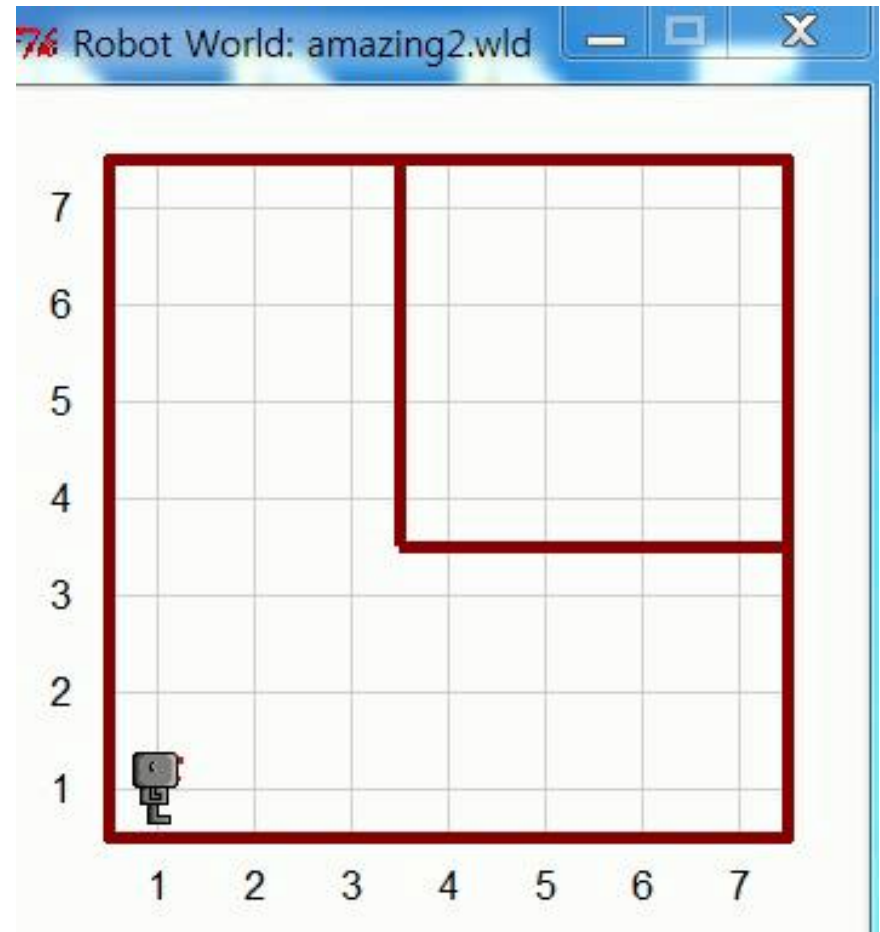


amazing2.wld

Infinite loop!

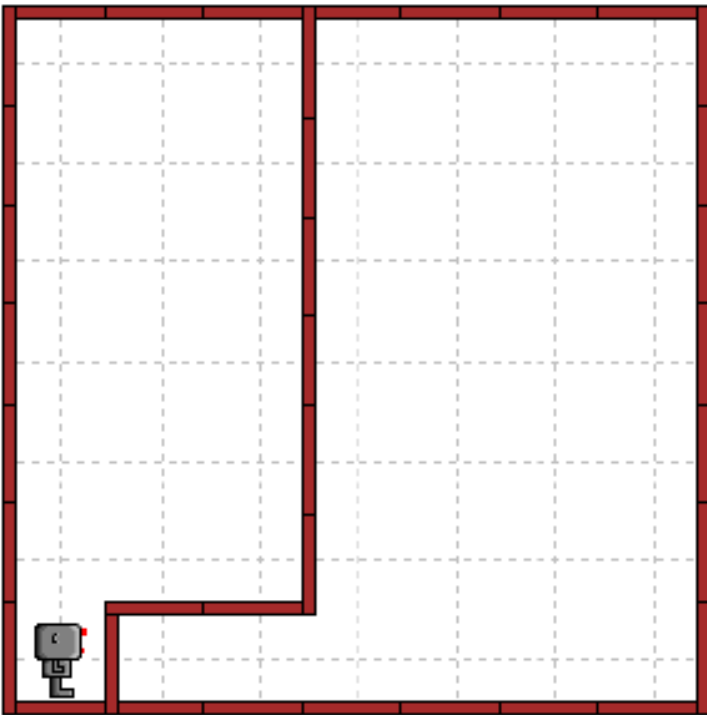
## How about this?

```
hubo.drop_beeper()  
hubo.move()  
while not hubo.on_beeper():  
    if hubo.right_is_clear():  
        turn_right()  
        hubo.move()  
    elif hubo.front_is_clear():  
        hubo.move()  
    else:  
        hubo.turn_left()
```



Does this always work?

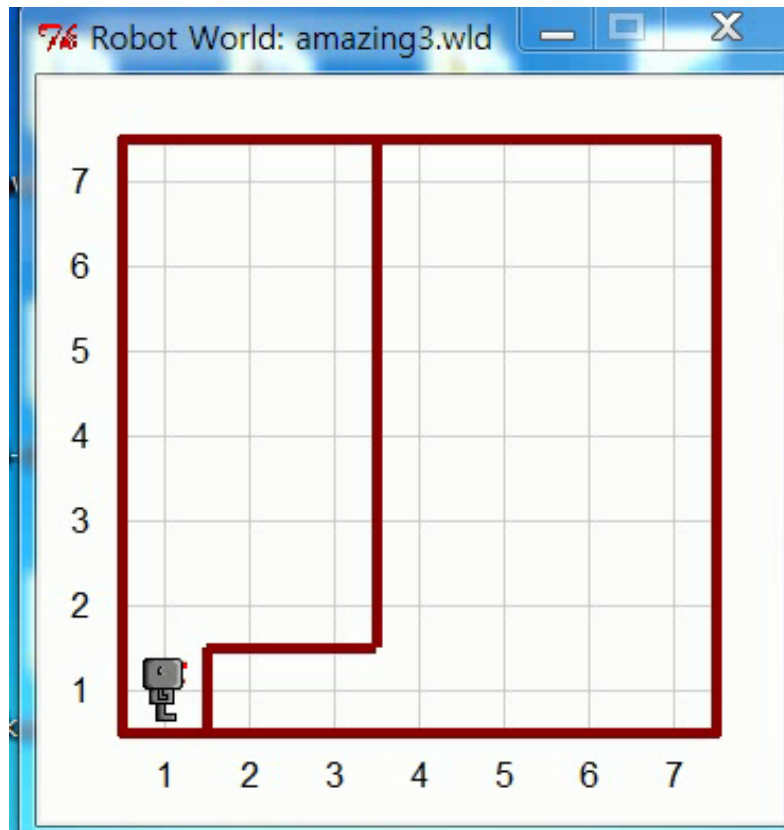
## How about this case?



amazing3.wld

```
hubo.drop_beeper()  
hubo.move()  
while not hubo.on_beeper():  
    if hubo.right_is_clear():  
        turn_right()  
        hubo.move()  
    elif hubo.front_is_clear():  
        hubo.move()  
    else:  
        hubo.turn_left()
```

## Does this work?

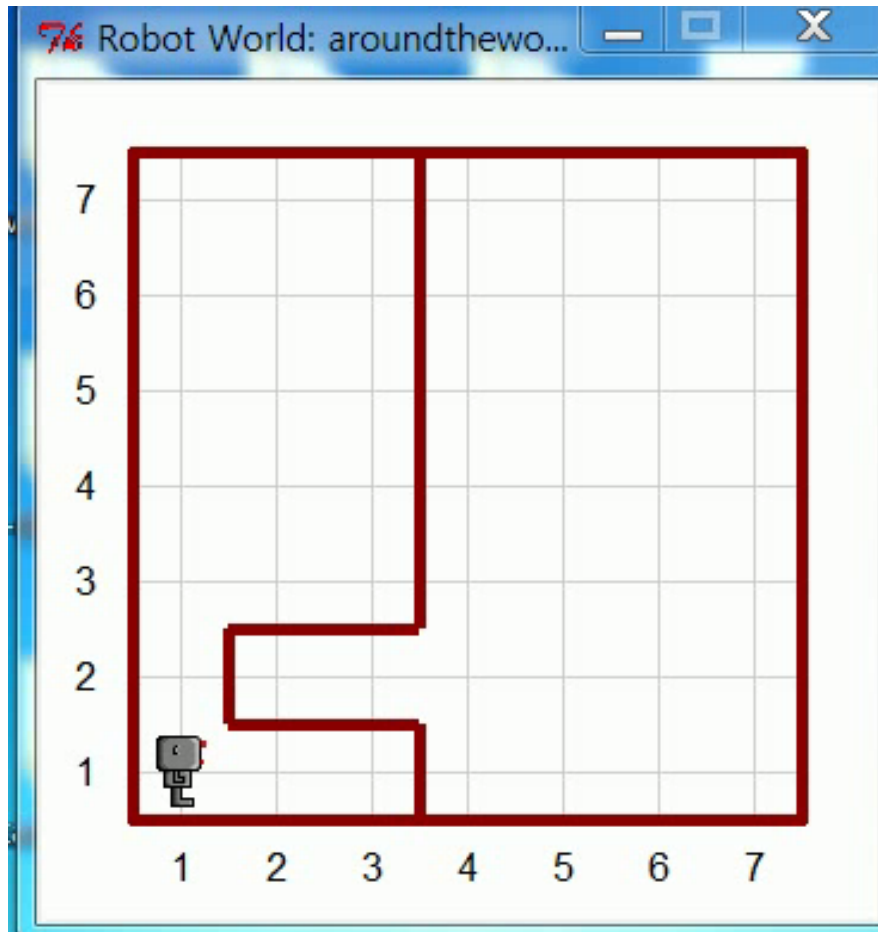


```
hubo.drop_beeper()
while not hubo.front_is_clear():
    hubo.turn_left()
hubo.move()
while not hubo.on_beeper():
    if hubo.right_is_clear():
        turn_right()
        hubo.move()
    elif hubo.front_is_clear():
        hubo.move()
    else:
        hubo.turn_left()
Hubo.turn_left()
```

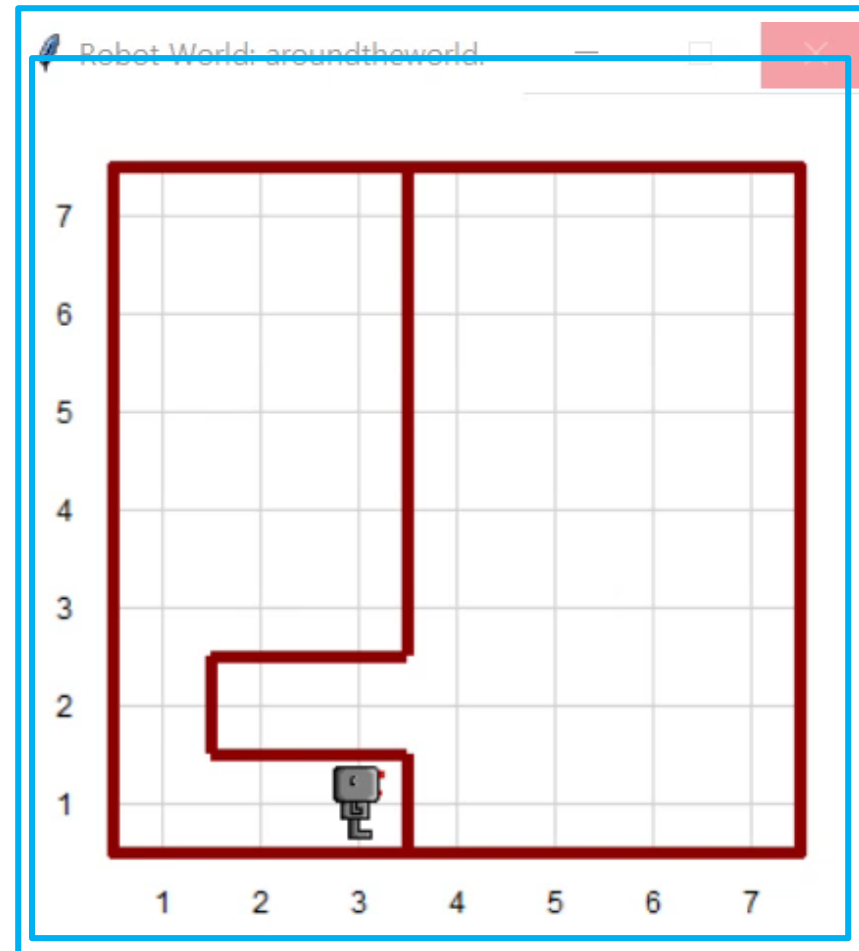
Does this always work?

**Still not perfect !**

Very **sensitive** to the **initial position** of Hubo.



aroundtheworld.wld



aroundtheworld.wld

```
hubo.drop_beeper()
while not hubo.front_is_clear():
    hubo.turn_left()
    hubo.move()
```

← `def mark_and_move():`

```
while not hubo.on_beeper():
    if hubo.right_is_clear():
        turn_right()
        hubo.move()
    elif hubo.front_is_clear():
        hubo.move()
    else:
        hubo.turn_left()
hubo.turn_left()
```

← `def follow_wall():`

```
mark_and_move()
while not hubo.on_beeper():
    follow_wall()
    hubo.turn_left()
```



```
def follow_wall():
    if hubo.right_is_clear():
        turn_right()
        hubo.move()
    elif hubo.front_is_clear():
        hubo.move()
    else:
        hubo.turn_left()

def mark_and_move():
    hubo.drop_beeper()
    while not hubo.front_is_clear():
        hubo.turn_left()
        hubo.move()

mark_and_move()
while not hubo.on_beeper():
    follow_wall()
hubo.turn_left()
```

One of the **secrets** of writing **good, correct, elegant programs** is to write them as if you wrote them for a **human reader**, not a computer. Let's clean up our program.

How ? By adding comments !

```
"""
This program lets the robot go around his world counter-
clockwise, stopping when he comes back to his starting point.
"""
```

```
#Turn right.
def turn_right():
    for i in range(3):
        hubo.turn_left()
#Mark the starting point and move
def mark_and_move():
    hubo.drop_beeper()
    while not hubo.front_is_clear():
        hubo.turn_left()
    hubo.move()
```

(continued)

---

#Follow the wall at each iteration.

```
def follow_wall():  
    if hubo.right_is_clear():  
        # turn right to follow the wall  
        right_turn()  
        hubo.move()  
    elif hubo.front_is_clear():  
        # move forward to follow the wall  
        hubo.move()  
    else:  
        # turn left to follow the wall  
        hubo.turn_left()
```

(continued)

---

```
def main():  
    #mark and move.  
    mark_and_move()  
    #Follow the entire wall.  
    while not hubo.on_beeper():  
        follow_wall()  
    hubo.turn_left()  
  
main()
```

1. Start with a **primitive program** that solves a simple problem.
2. Make **small changes**, one at a time to generalize the program.
3. Make sure that each change **does not invalidate** what you have done before.
4. Add appropriate **comments** (not just repeating what the instruction does).
5. Choose **descriptive names**.