# CS101 - Conditionals and `while` Loops

## Lecture 2

School of Computing
KAIST

# Roadmap

Last week we learned

- Functions and for loops

# Roadmap

Last week we learned

- Functions and for loops

This week we will learn

- Conditionals
- `if` statements
- `while` loops

# Conditionals

So far, our programs performed exactly the same steps every time the program is run.

# Conditionals

So far, our programs performed exactly the same steps every time the program is run.
Often, what the robot does must depend on the environment:

```
if it rains:
  listen_to_cs101_lecture()
else:
  eat_strawberries_in_the_sun()
```

# Conditionals

So far, our programs performed exactly the same steps every time the program is run.
Often, what the robot does must depend on the environment:

```
if it rains:
  listen_to_cs101_lecture()
else:
  eat_strawberries_in_the_sun()
```

# Conditionals

So far, our programs performed exactly the same steps every time the program is run.
Often, what the robot does must depend on the environment:

```
if it rains:
  listen_to_cs101_lecture()
else:
  eat_strawberries_in_the_sun()
```

**condition**

**if condition is true, do this**

**if condition is false, do that**

# Conditionals

So far, our programs performed exactly the same steps every time the program is run.
Often, what the robot does must depend on the environment:

```
if it rains:
  listen_to_cs101_lecture()
else:
  eat_strawberries_in_the_sun()
```

**condition**

**if condition is true, do this**

**if condition is false, do that**

A **condition** is something that is either `True` or `False`.

# Silly examples

```
if True:
  print("CS101 is my favorite course")
```

# Silly examples

```
if True:
  print("CS101 is my favorite course")

if False:
  print("Every CS101 student will receive an A+")
```

# Silly examples

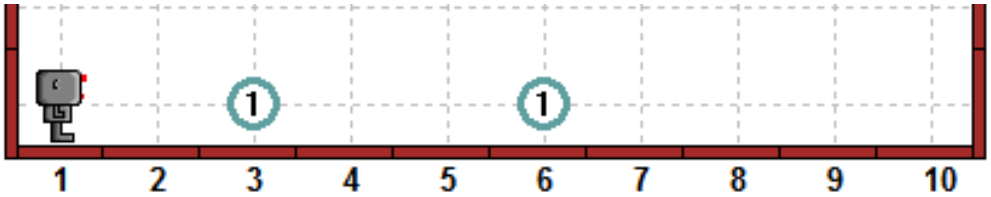
```
if True:
  print("CS101 is my favorite course")

if False:
  print("Every CS101 student will receive an A+")

if 3 < 5:
  print("3 is less than 5")
else:
  print("3 is larger than 5")
```

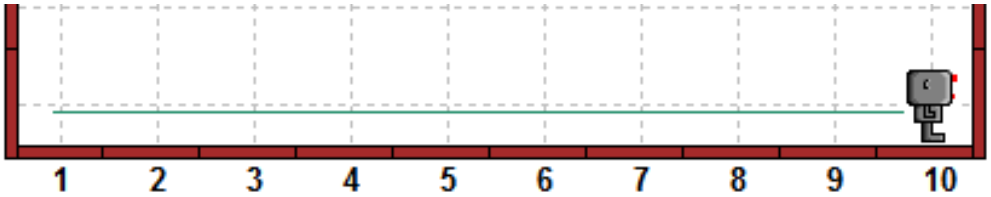
# Sensing beepers

We want the robot to make 9 steps and pick up all beepers on the way.



# Sensing beepers

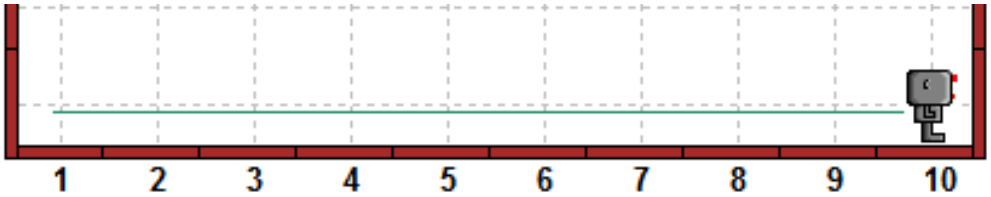
We want the robot to make 9 steps and pick up all beepers on the way.




`hubo.pick_beeper()` causes an error if there is no beeper.

# Sensing beepers

We want the robot to make 9 steps and pick up all beepers on the way.



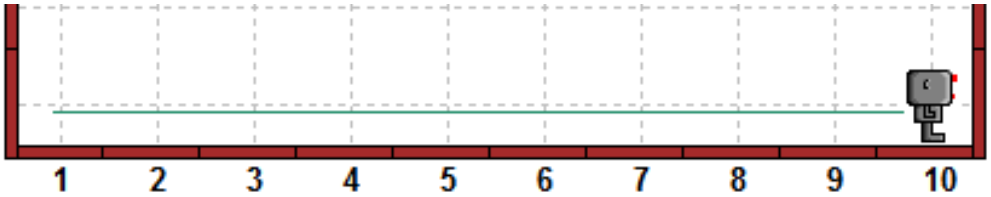

`hubo.pick_beeper()` causes an error if there is no beeper.

Repeat the following 9 times:

- Take a step forward
- Check if there is a beeper
- Pick the beeper up if yes

# Sensing beepers

We want the robot to make 9 steps and pick up all beepers on the way.




`hubo.pick_beeper()` causes an error if there is no beeper.

Repeat the following 9 times:

- Take a step forward
- Check if there is a beeper
- Pick the beeper up if yes

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

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

# not True is False

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

# not True is False

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()
```

# not True is False

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()
```

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

# not True is False

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()
```

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)
```

# What else?

Let's try to follow the boundary of the world: We move forward if there is no wall, otherwise turn to the left.

# What else?

Let's try to follow the boundary of the world: We move forward if there is no wall, otherwise turn to the left.

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

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

# With singing and dancing . . .

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

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

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

# With singing and dancing . . .

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

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

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

Note the indentation!

# With singing and dancing . . .

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

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

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

What happens now?

# With singing and dancing . . .

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

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

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

. . . and now?

# Many choices!

```
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 hubo.right_is_clear():
        turn_right()
      else:
        turn_around()
```

Problem) This code is hard to read and understand!

# Many choices!

```
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()
```

# Many choices!

```
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()
```

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

# while-loops

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

# while-loops

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

A **`while`**-loop repeats instructions as long as some condition is true.

# while-loops

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

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

Go forward until we reach a beeper:

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

# Around the world in 80 days

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

# Around the world in 80 days

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

1. Put down a beeper to mark starting point
2. Move forward until facing wall
3. Turn left
4. Repeat steps 2 and 3 until we find the beeper
5. Finish when we found the beeper

# Around the world in 80 days

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

1. Put down a beeper to mark starting point
2. Move forward until facing wall
3. Turn left
4. Repeat steps 2 and 3 until we find the beeper
5. Finish when we found the beeper

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

# Around the world in 80 days

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

1. Put down a beeper to mark starting point
2. Move forward until facing wall
3. Turn left
4. Repeat steps 2 and 3 until we find the beeper
5. Finish when we found the beeper

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

**Doesn't work**

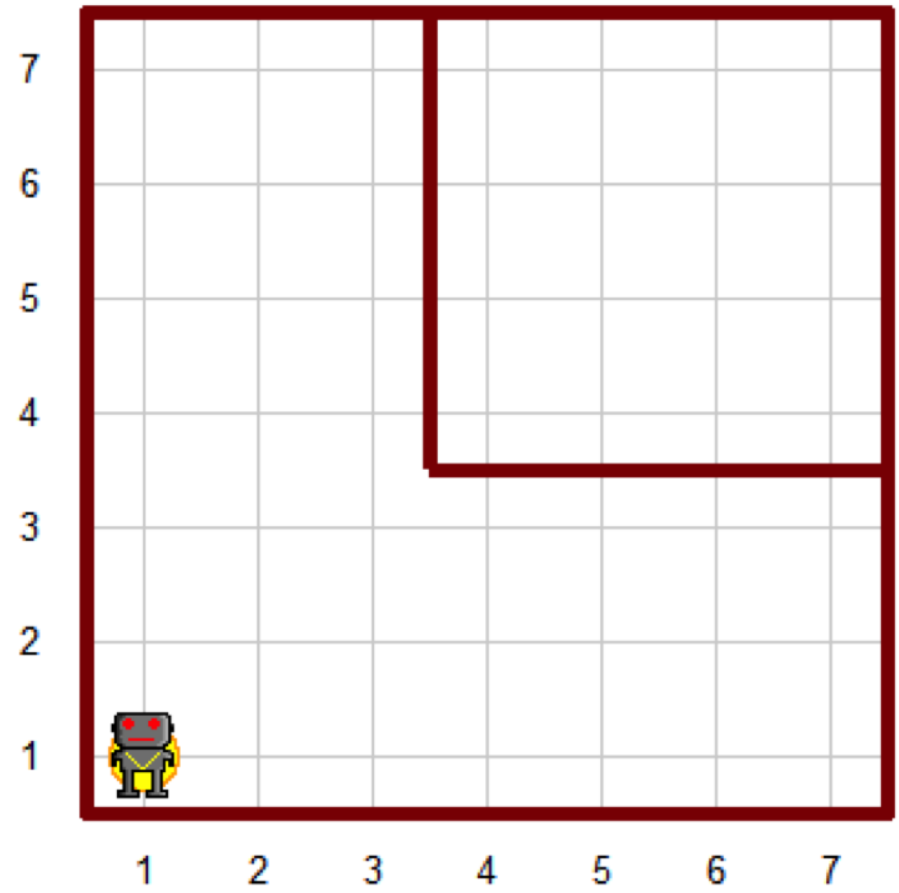
# Around the world in 80 days

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

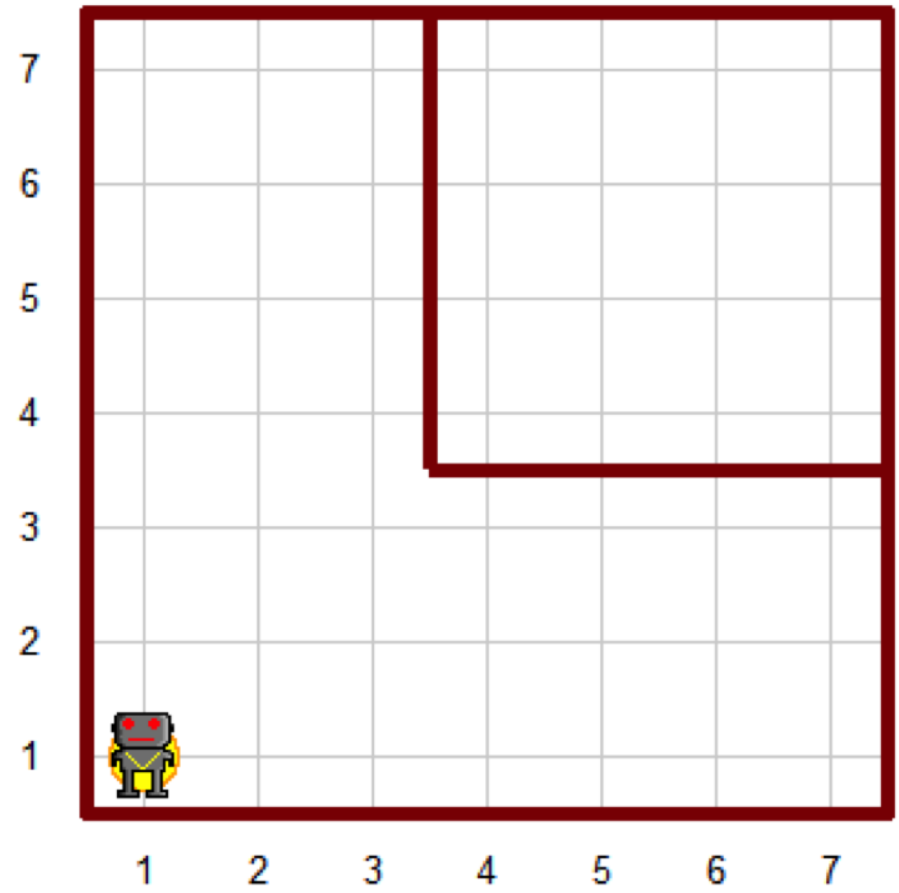
1. Put down a beeper to mark starting point
2. Move forward until facing wall
3. Turn left
4. Repeat steps 2 and 3 until we find the beeper
5. Finish when we found the beeper

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

# What if the world looks like below?



# What if the world looks like below?




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

# Sometimes we need right turns

```
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()
```

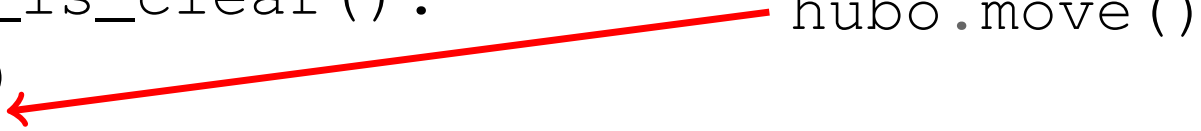
# Sometimes we need right turns

```
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()
```

This can go into an **infinite loop**!

# Sometimes we need right turns

```
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()
```



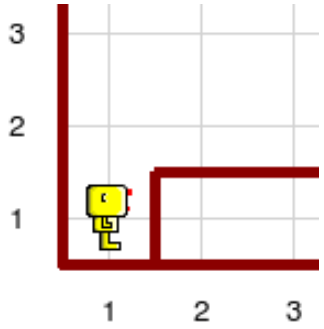

This can go into an **infinite loop**!

# Sometimes we need right turns

```
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.move()

This can go into an **infinite loop**!



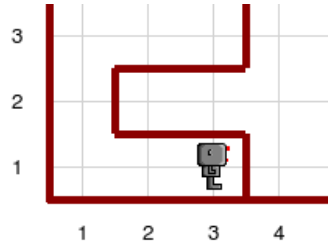

Still does not work when there is a wall in front of the starting position!

# Getting out of the starting position

```
hubo.drop_beeper()
if 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()
```

# Getting out of the starting position

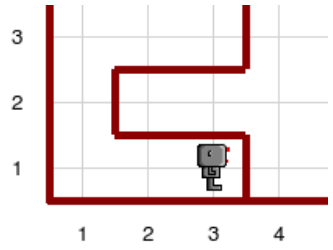
```
hubo.drop_beeper()
if not hubo.front_is_clear():
  hubo.turn_left()
hubo.move()
while not hubo.on_beeper():
  if hubo.right_is_clear():
    turn_right()
    hubo.move()
  elif hubo.front_is_clear():
    hubo.move()
  else:
    hubo.turn_left()
```




Still has a problem if not starting at position (1,1).

# Getting out of the starting position

```
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()
```




Still has a problem if not starting at position (1,1).

# Write code for humans

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:

```
# This program lets the robot go around his world
# counter clockwise, stopping when he returns
# to the starting point.
from cs1robots import *
load_world()
hubo = Robot(beepers=1)

def turn_right():
  for i in range(3):
    hubo.turn_left()

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

```
def follow_right_wall():
  if hubo.right_is_clear():
    # Keep to the right
    turn_right()
    hubo.move()
  elif hubo.front_is_clear():
    # move following the right wall
    hubo.move()
  else:
    # follow the wall
    hubo.turn_left()

# end of definitions, begin solution

mark_starting_point_and_move()

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

# Stepwise refinement

Steps to follow when writing a program:

- Start simple
- Introduce small changes, one at a time
- **Make sure** that each change does not invalidate the work you have done before
- Add appropriate comments (not just repeating what the instruction does)
- Choose descriptive names