Robot Software Education Institute
로봇 SW 교육원

로봇활용 소프트웨어 교육

파이썬 #2

광운대학교 로봇학부
박광현

이번 강의에서 실습하는 내용들은
반 학기 분량의 내용입니다.
라인 트레이싱도 단계를 나누어서 천천히 학습해야 하는데
본 강의의 특성상 압축하여 하루에 진행하였습니다.

일정

데이터를 다루자	13:00 ~ 14:00
라인 트레이서	14:10 ~ 15:10
컵 따라 돌기	15:20 ~ 16:20
복합 라인 트레이서	16:30 ~ 17:30

데이터를 다루자

덧셈기

```
sum = 0
while True:
      number = int(raw_input('Number: '))
      sum += number
      print sum
```

정수가 아닌 걸 입력하면?

덧셈기

```
sum = 0
while True:
      number = float(raw_input('Number: '))
      sum += number
      print sum
```

실수가 아닌 걸 입력하면?

덧셈기

```
sum = 0
while True:
      number = complex(raw_input(‘Number: '))
      sum += number
      print sum
```

원의 면적을 구해 볼까요?

```
while True:
      radius = float(raw_input('Radius: '))
      area = ?????
      print area
```

원의 면적을 구해 볼까요?

```
while True:
      radius = float(raw_input('Radius: '))
      area = 3.13 * radius * radius
      print area
```

원의 면적을 구해 볼까요?

```
import math
while True:
      radius = float(raw_input('Radius: '))
      area = math.pi * radius ** 2
      print area
```

많은 데이터를 다루자: 리스트

```
empty = []
numbers = [1, 2, 3]
texts = [‘hello’, ‘hamster’]
mixed = [1, ‘hello’, 2, ‘hamster’]
embedded = [1, 2, [1, 2]]
```

```
sentence = 'hello hamster.'
sentence[6:13]
sentence[6:-1]
sentence[6:]
sentence[:5]
```

```
numbers = [1, 2, 3, 4, 5, 6]
```

[4, 5]만 뽑아 내기

이 예제들의 핵심은
이미 알고 있는 사실들(지난 시간에 배운 문자 연산들)을 단서로 하여
유추하고 결과 확인을 통해 검증하여
스스로 규칙을 찾아내는 것입니다.

```
sentence = 'hello hamster.'
sentence[6:13]
sentence[6:-1]
sentence[6:]
sentence[:5]
```

```
numbers = [1, 2, 3, 4, 5, 6]
```

[4, 5]만 뽑아 내기

```
print numbers[3:5]
```

```
sentence = 'hello hamster.'
sentence[6:13]
sentence[6:-1]
sentence[6:]
sentence[:5]
```

```
numbers = [1, 2, 3, 4, 5, 6]
```

[4, 5]만 뽑아 내기

```
print numbers[3:5]

print numbers[3:-1]
```

```
sentence = 'hello hamster.'
sentence[6:13]
sentence[6:-1]
sentence[6:]
sentence[:5]
```

```
numbers = [1, 2, 3, 4, 5, 6]
```

[4, 5]만 뽑아 내기

```
print numbers[3:5]

print numbers[3:-1]
```

[4, 5, 6]까지

```
sentence = 'hello hamster.'
sentence[6:13]
sentence[6:-1]
sentence[6:]
sentence[:5]
```

```
numbers = [1, 2, 3, 4, 5, 6]
```

[4, 5]만 뽑아 내기

```
print numbers[3:5]

print numbers[3:-1]

print numbers[3:]
```

[4, 5, 6]까지

```
sentence = 'hello hamster.'
sentence[6:13]
sentence[6:-1]
sentence[6:]
sentence[:5]
```

```
numbers = [1, 2, 3, 4, 5, 6]
```

[4, 5]만 뽑아 내기

```
print numbers[3:5]

print numbers[3:-1]

print numbers[3:]
```

[4, 5, 6]까지

[1, 2, 3] 뽑아 내기

```
sentence = 'hello hamster.'
sentence[6:13]
sentence[6:-1]
sentence[6:]
sentence[:5]
```

```
numbers = [1, 2, 3, 4, 5, 6]
```

[4, 5]만 뽑아 내기

```
print numbers[3:5]

print numbers[3:-1]

print numbers[3:]

print numbers[:3]
```

[4, 5, 6]까지

[1, 2, 3] 뽑아 내기

```
test = [1, [2, 'hello', 'hamster'], [3, 4, 5]]
```

['hello', 'hamster'] 빼와 내기

```
test = [1, [2, 'hello', 'hamster'], [3, 4, 5]]
```

['hello', 'hamster'] 뺌아 내기

```
print test[1][1:]
```

```
test = [1, [2, 'hello', 'hamster'], [3, 4, 5]]
```

['hello', 'hamster'] 뺄아 내기

```
print test[1][1:]
```

[3, 4] 뺄아 내기

```
test = [1, [2, 'hello', 'hamster'], [3, 4, 5]]
```

['hello', 'hamster'] 뺄아 내기

```
print test[1][1:]
```

[3, 4] 뺄아 내기

```
print test[-1][:2]
```

```
print 'hello' + 'hamster'
print '-'*30
```

다양한 것을 지난 시간에 배운 문자 연산을 단서로 하여 유추하는 예제들입니다.

```
[1, 2, 3]
['hello', 'hamster'
```

```
[1, 2, 3, 'hello', 'hamster']
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
numbers = [1, 2, 3]  →  [1, 5, 3]

numbers[1] = 5
print numbers
```

일단 하나의 예제를 설명해 주고
이를 단서로 하여
다른 예제들의 답을 유추하고
스스로 만든 규칙을 확인합니다.

유추하고 검증을 통해 확인하면서 규칙을 만들어 나가는 과정이 중요한데
SW교육의 장점이 여기에 있습니다.
이러한 탐구 과정은 과학 실험으로도 학습할 수 있지만
과학 실험은 확인하는데 시간과 노력이 많이 들어갑니다.
이에 비해 SW는 즉각적인 피드백이 가능하므로 쉽게 학습할 수 있습니다.

```
numbers = [1, 2, 3]  →  [1, 5, 3]

numbers[1] = 5
print numbers
```

```
numbers = [1, 2, 3]  →  [1, 2, 4]
```

이제 유추해 봅시다.

```
numbers = [1, 2, 3]  →  [1, 5, 3]

numbers[1] = 5
print numbers


numbers = [1, 2, 3]  →  [1, 2, 4]

numbers[-1] = 4
print numbers
```

```
numbers = [1, 2, 3]  →  [10, 11, 3]

numbers[0:2] = [10, 11]
print numbers

numbers[1] = [10, 11]

numbers[1:2] = [10, 11]


numbers = [1, 2, 3]  →  [3]

numbers = [1, 2, 3]  →  [2, 3]
```

```
numbers = [1, 2, 3]
```

파이썬에서 제공하는 함수들

```
numbers.append(4)

del numbers[1]

del numbers[:2]

len(numbers)
```

미션 #1

성적 처리

```
park = [1, 2, 3]
kim = [ 4, 5, 6]
lee = [7, 8, 9]
choi = [10, 11, 12]
```

국어 영어 수학

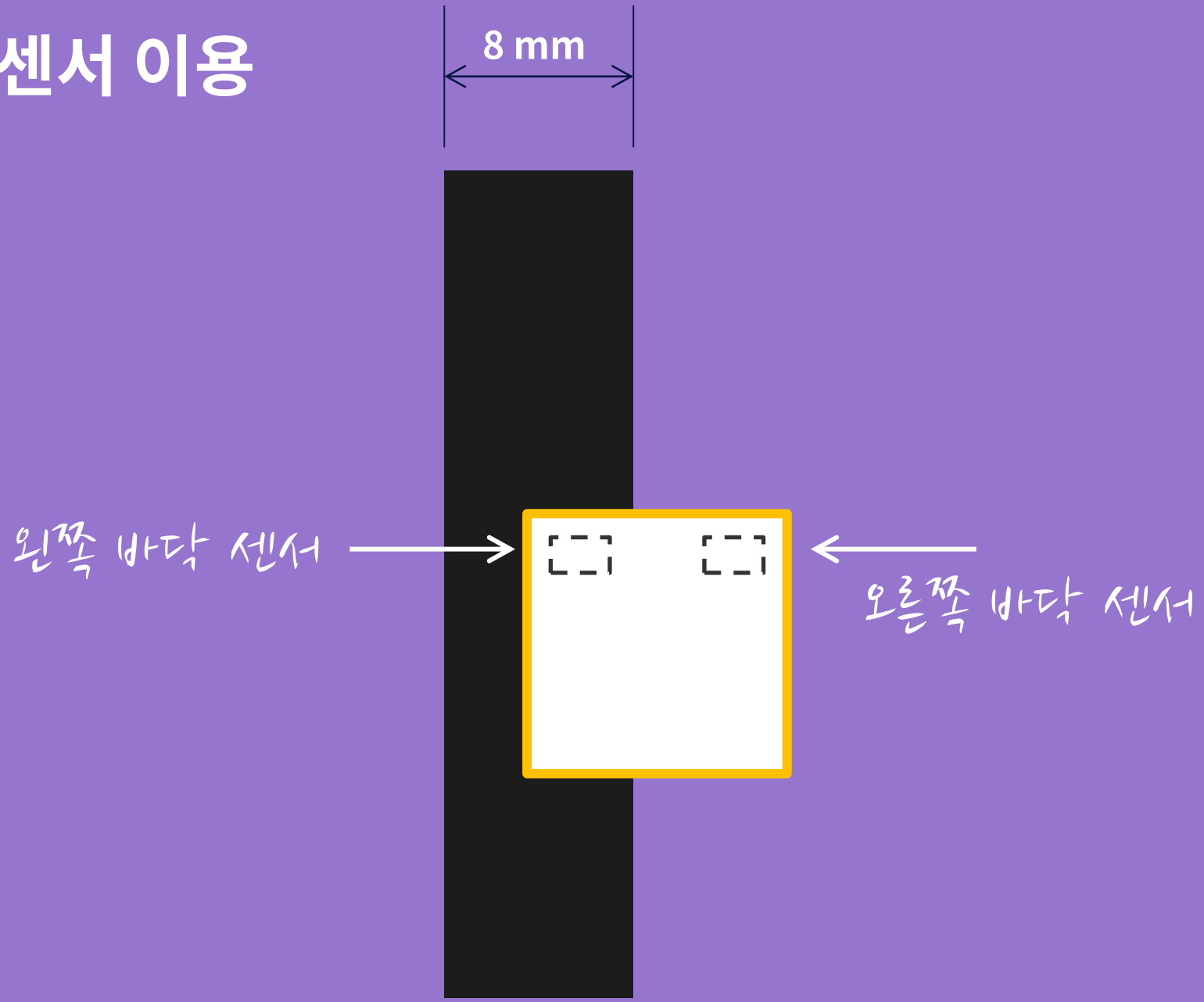
각 과목의 평균을
리스트로 출력

코드를 한 줄로 짤 수도 있다능: 이건 그냥 참고.
코드를 짧게 짠다고 무조건 좋은 것은 아님

```
print map(lambda x: x/4.0, map(sum, zip(park, kim, lee, choi)))
```

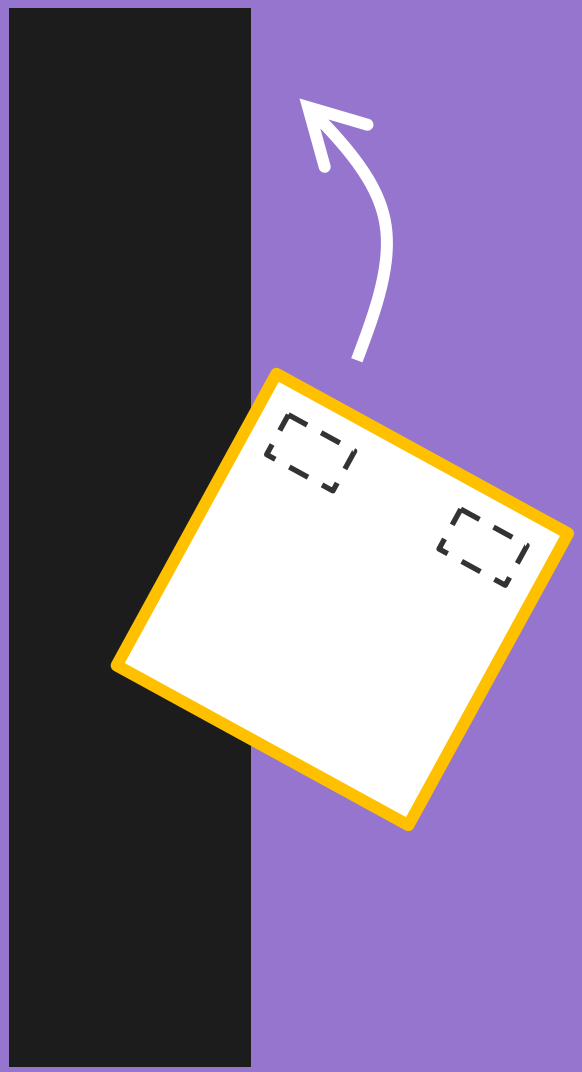
라인 트레이서

왼쪽 센서 이용

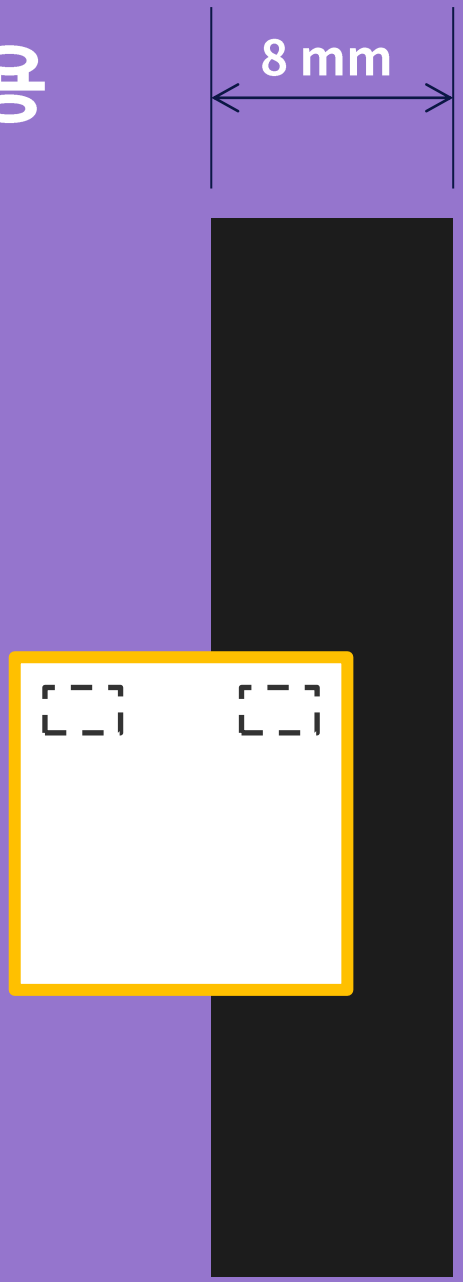


왼쪽 센서 이용

검은 색 위로 들어가면 오른쪽으로 회전
하얀색 영역으로 나가면 왼쪽으로 회전

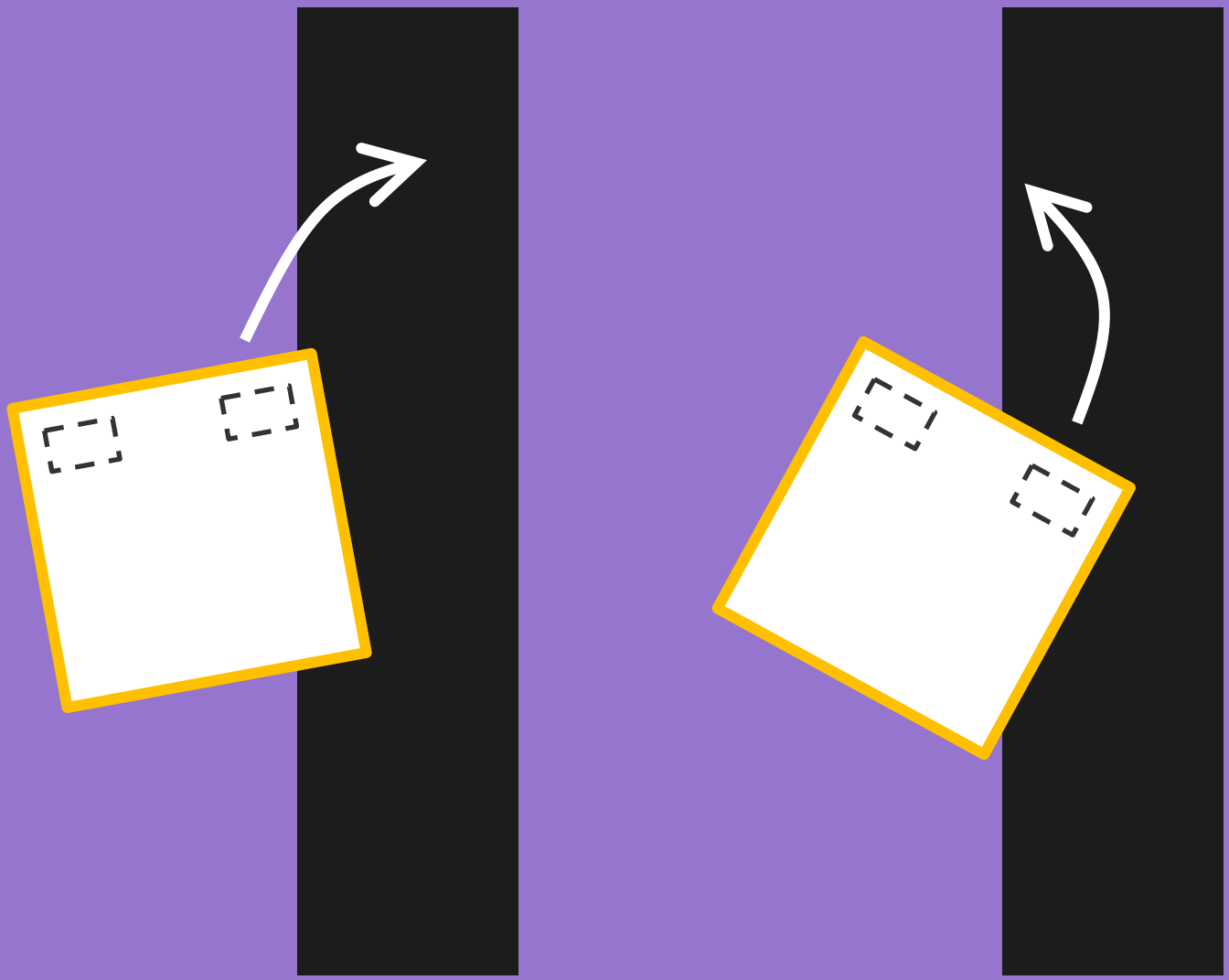


오른쪽 센서 이용

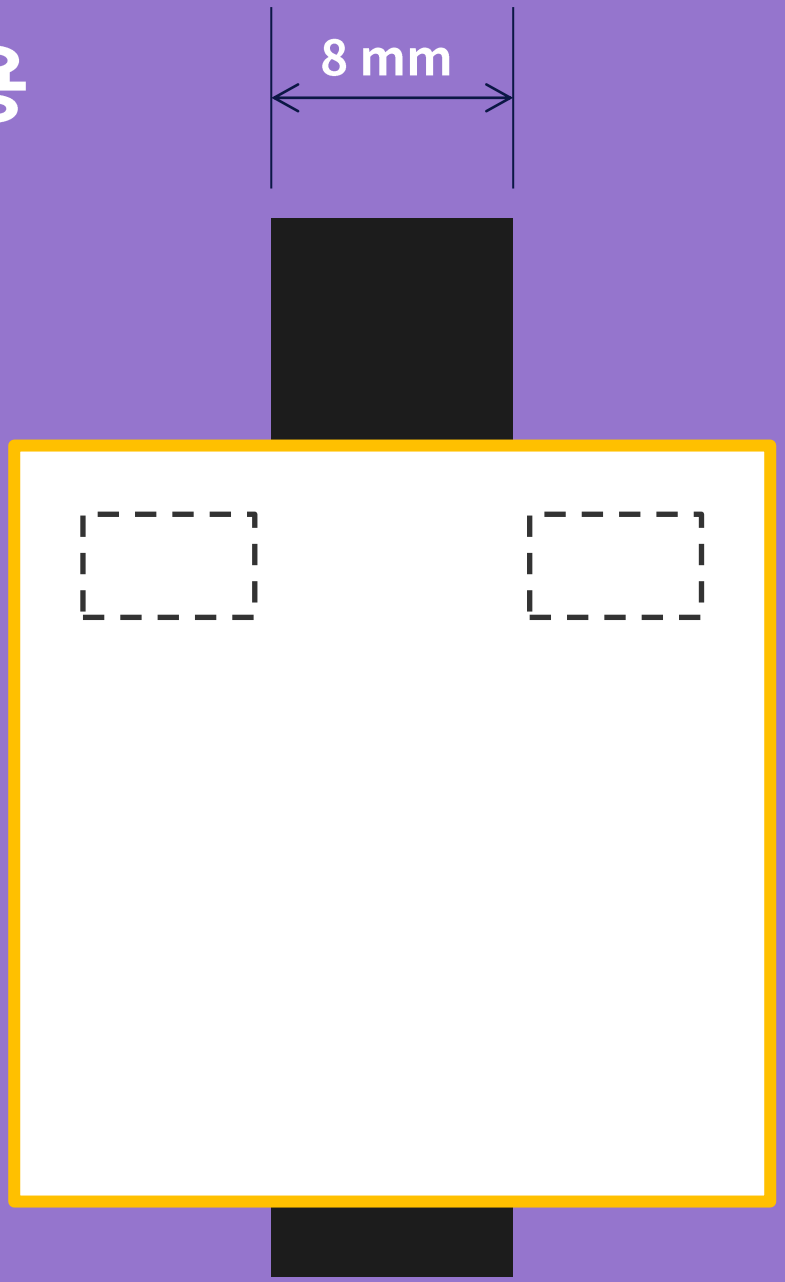


오른쪽 센서 이용

검은 색 위로 들어가면 왼쪽으로 회전
하얀색 영역으로 나가면 오른쪽으로 회전

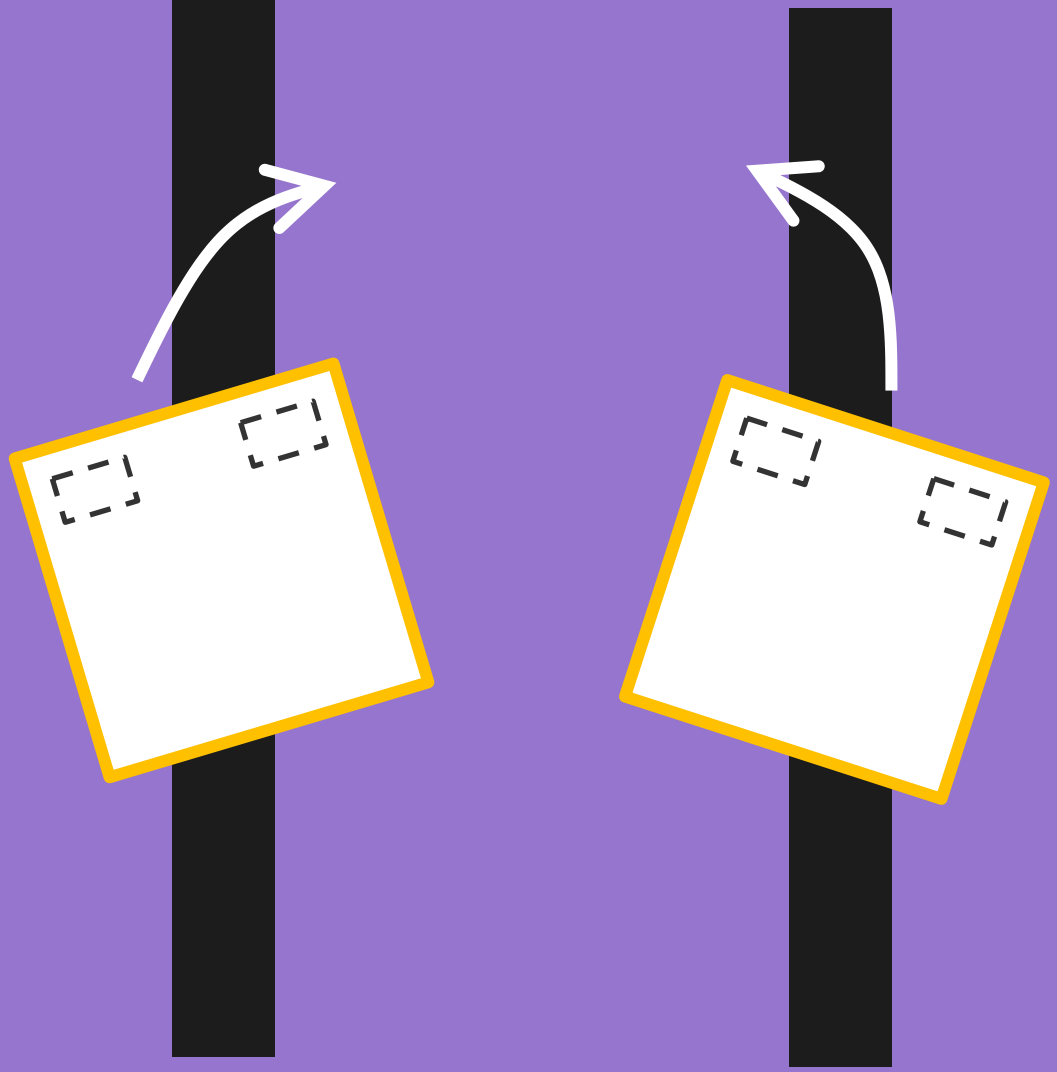


양쪽 센서 이용



양쪽 센서 이용

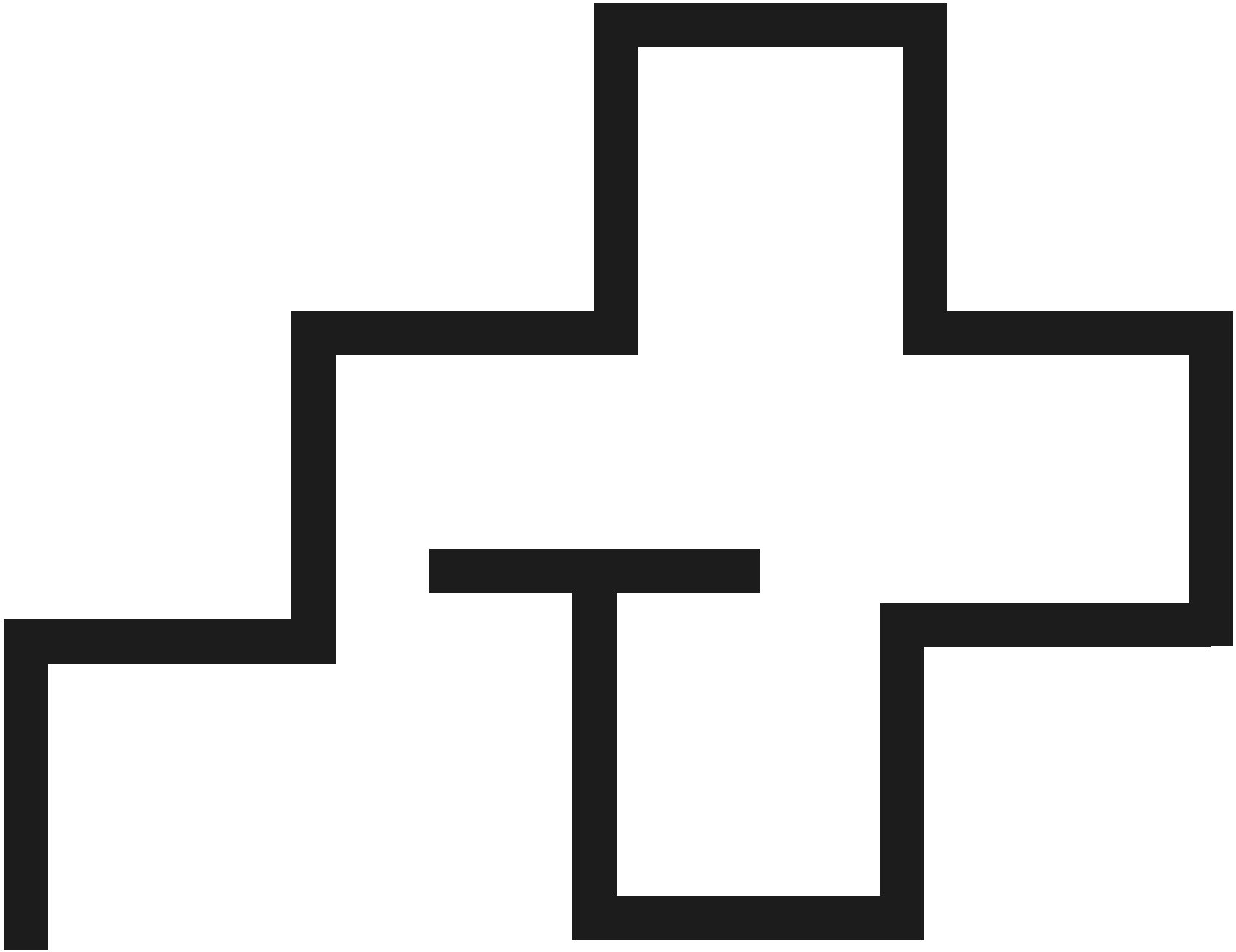
오른쪽 바닥 센서가 검은색을 감지하면 오른쪽으로 회전, 왼쪽 바닥 센서가 검은색을 감지하면 왼쪽으로 회전



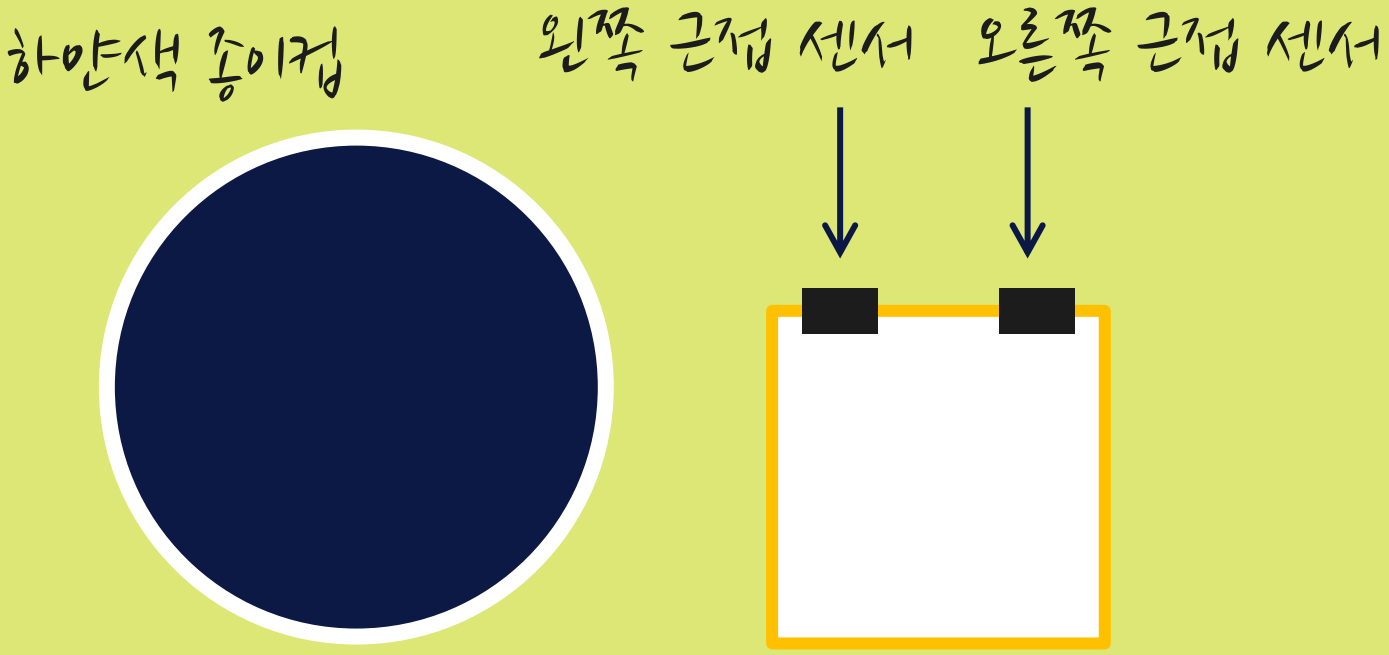
힌트

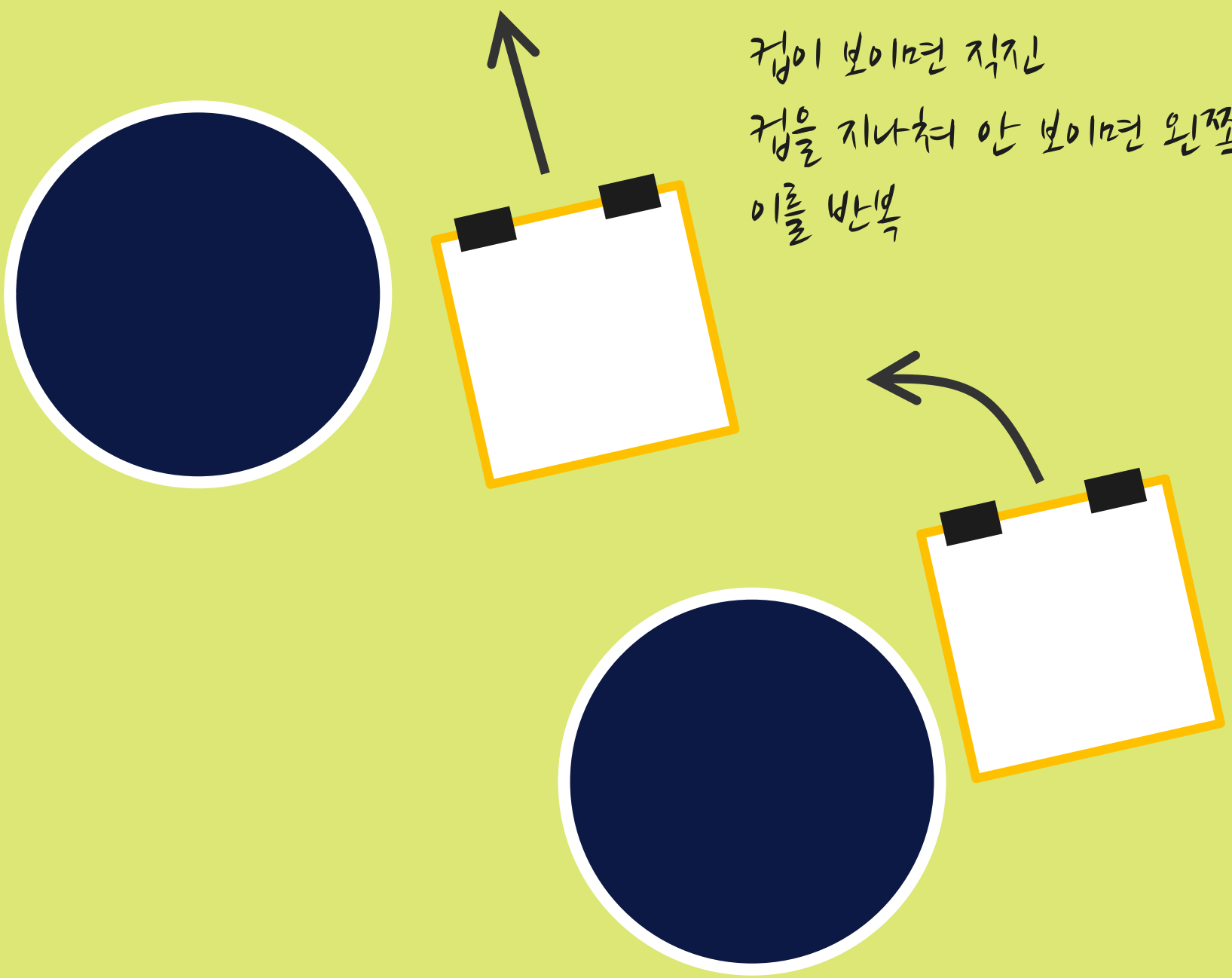
똑바로 전진하기 위해서는
양쪽 바닥 센서 값이 차이를 이용한
비례 제어가 필요

```
from roboid.hamster import Hamster

robot = Hamster()
robot.wait(100) 처음엔 센서 값이 이이므로 0.1초 정도 쉰다.

leftFloor = robot.read(Hamster.LEFT_FLOOR)
rightFloor = robot.read(Hamster.RIGHT_FLOOR)
diff = leftFloor - rightFloor

robot.write(Hamster.LEFT_WHEEL, a + diff * b)
robot.write(Hamster.RIGHT_WHEEL, a - diff * b)
```



컵 따라 돌기


하얀색 종이컵
왼쪽 근접 센서
오른쪽 근접 센서


컵이 보이면 직진
컵을 지나쳐 안 보이면 왼쪽으로 회전
이를 반복

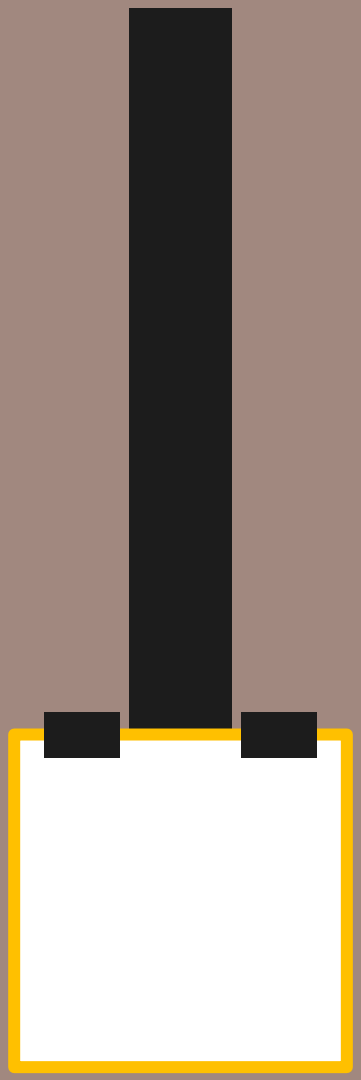
복합 라인 트레이서

문제 분해

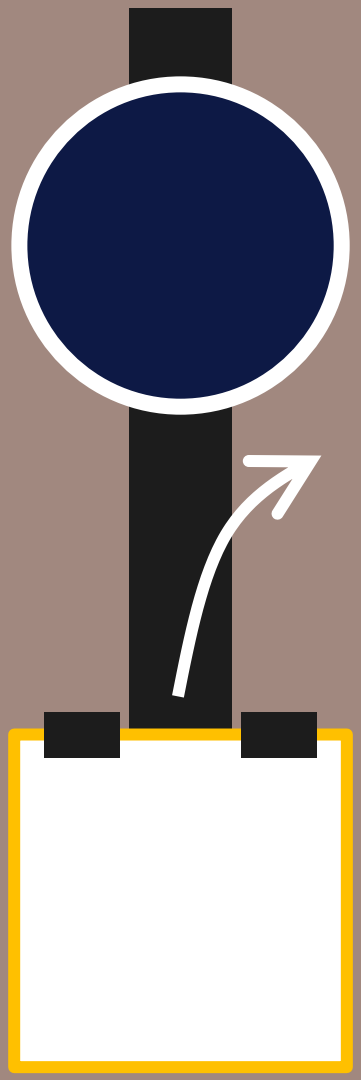
라인 트레이싱과 컵 따라 돌기가
결합된 문제를 통해
'문제 분해'의 개념을 학습합니다.

각 단계는 하나의 상태가 되며
상태 천이, FSM(Finite State Machine)
등의 학습으로 넘어갈 수도 있습니다.
(심화 문제)

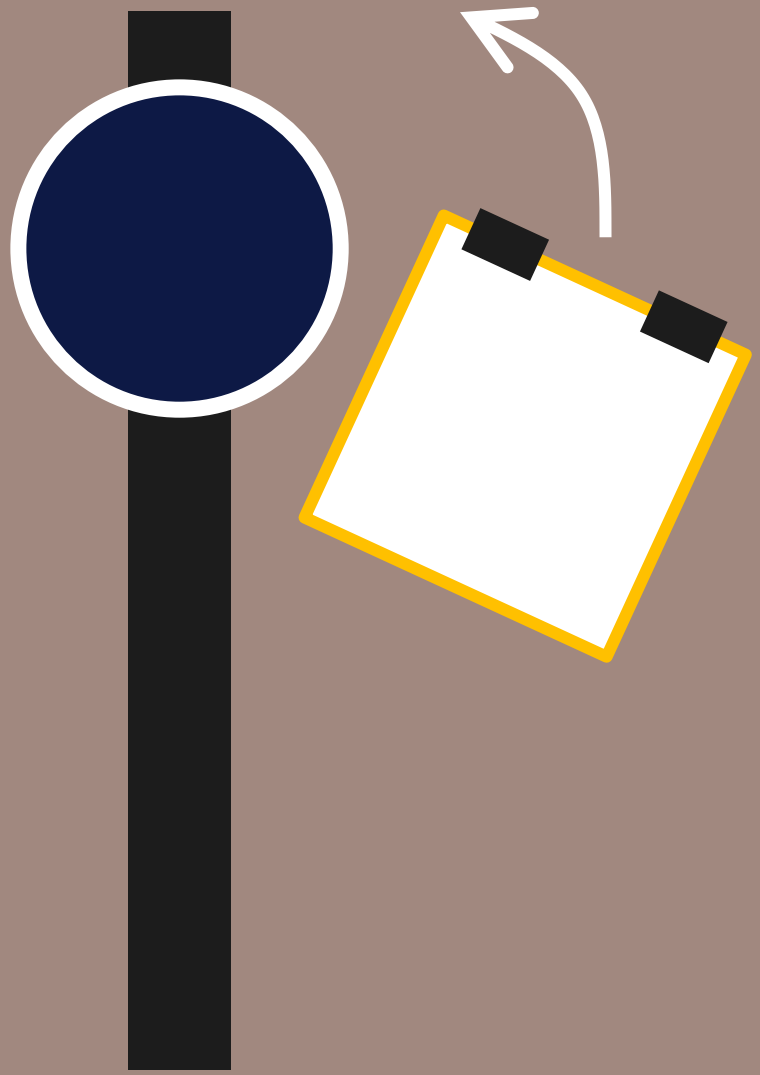
1) 라인 트레이싱



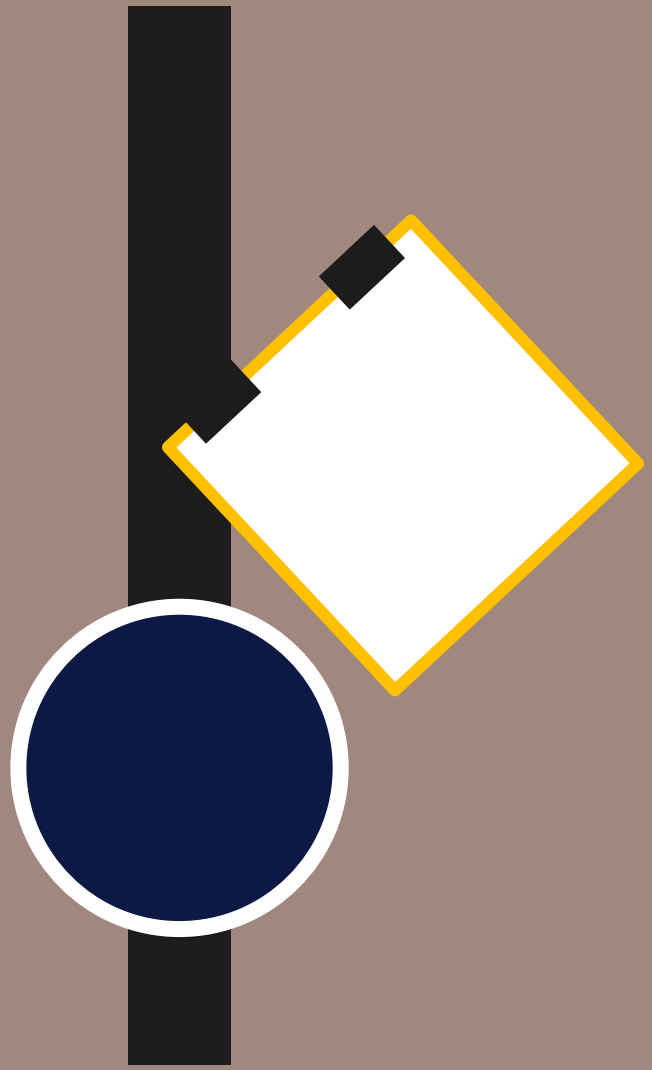
2) 장애물 감지 및 회피



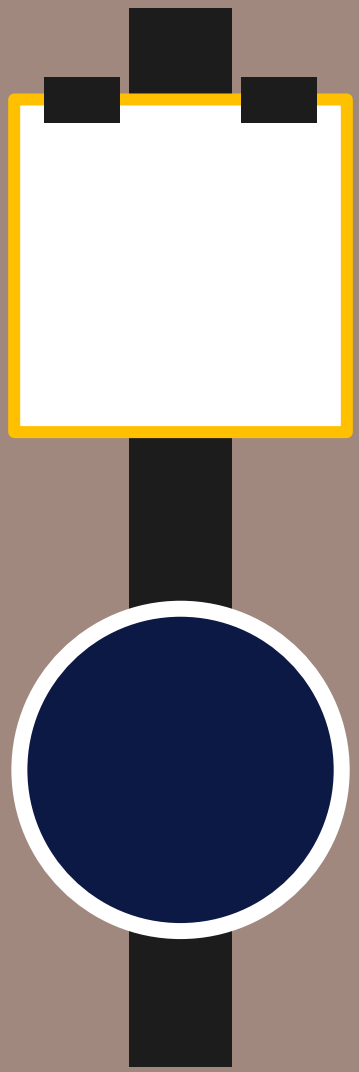
3) 컵 따라 돌기



4) 라인 찾기



5) 복귀



추후 일정

파이썬 #3 8월 22일

대회의실

정리

다음 문제: 미로 탈출

메이즈 러너

수고하셨습니다 !

akaii@kw.ac.kr