파이썬으로 배우는 햄스터와 인공지능 예제

ROBOMATION
MAKE THINGS ALIVE

본 교재는 광운대 로봇SW교육원이 운영하는 햄스터스쿨의
교육자료(https://hamster.school/ko/tutorial/)를 기반으로 작성되었습니다.

목차

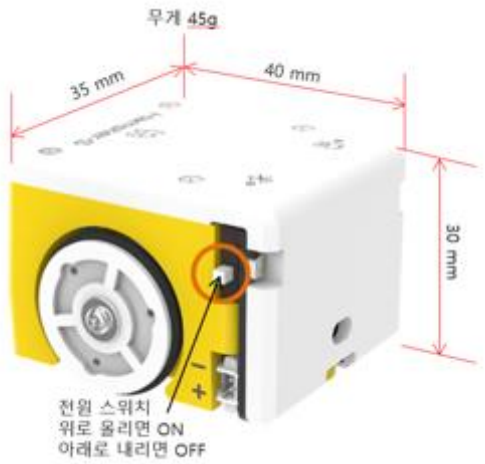
1차시

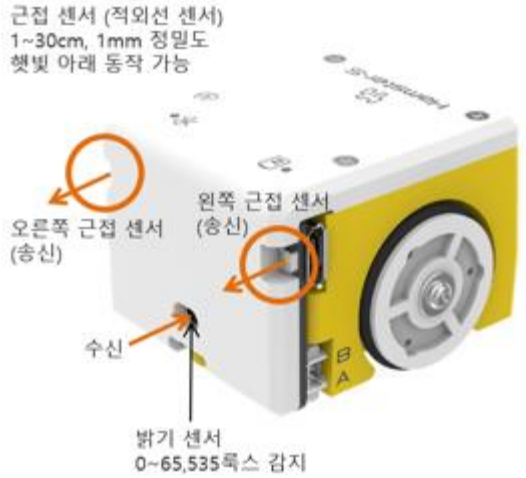
수업 준비

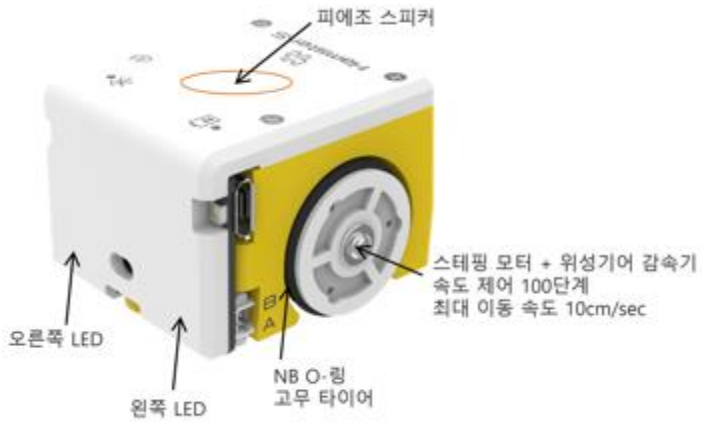
01

하드웨어 이해

- 햄스터S는 소프트웨어 교육용 로봇입니다. 다음 그림과 같이 다양한 장치를 포함하고 있습니다.







내장 리튬 배터리 3.7V, 200mA
충전 약 60분
연속 동작 평균 1시간
대기 최대 12시간

하드웨어 이해

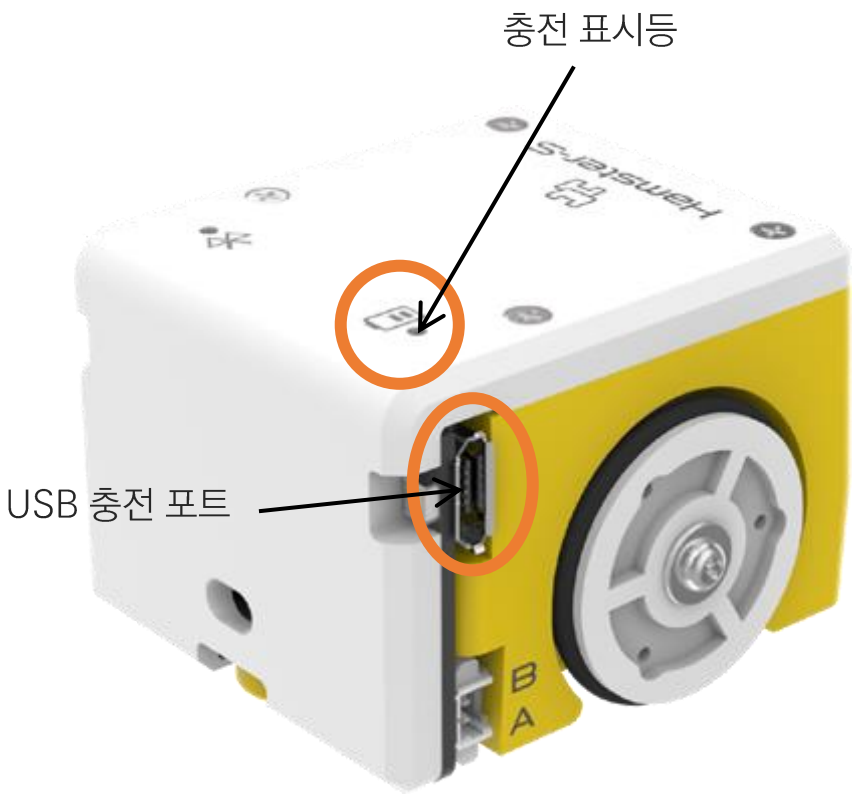
- 다음 그림과 같이 외부 확장 포트를 지니고 있습니다.

01 하드웨어 이해

- 전원을 켜기 위해서는 전원 스위치를 위로하여 ON 위치에 가도록 합니다.
- 전원을 끄기 위해서는 전원 스위치를 아래로 하여 OFF 위치에 가도록 합니다.

01

하드웨어 이해



- 햄스터S 로봇은 스마트 폰용 충전기를 사용하여 충전할 수 있습니다. 마이크로 USB 단자를 햄스터S 로봇의 충전 포트에 연결하면 됩니다.
- USB 케이블을 사용하여 충전할 수도 있습니다. USB 케이블의 마이크로 USB 단자를 햄스터S 로봇의 충전 포트에 연결하고, 반대쪽을 컴퓨터의 USB 포트에 연결하면 됩니다.
- 충전 중에는 충전 표시등이 빨간색으로 표시되고, 충전이 완료되면 충전 표시등이 꺼집니다.
- 완전히 충전하면 약 1시간 정도 사용할 수 있습니다.
- 전원을 끄고 충전하는 것이 더 좋습니다.

내장 리튬 배터리 3.7V, 200mA
충전 약 60분
연속 동작 평균 1시간
대기 최대 12시간

01 하드웨어 이해

- **파란색으로 천천히 깜박임**

  블루투스 연결을 기다리는 상태입니다.

  햄스터S 로봇의 전원을 켜면 블루투스 연결 표시등이 파란색으로 천천히 깜박입니다.

- **파란색으로 계속 켜져 있음**

  블루투스가 연결된 상태입니다.

  컴퓨터 또는 스마트 폰과 블루투스로 연결되면 연결 표시등이 계속 켜져 있습니다.

- **파란색으로 빠르게 깜박임**

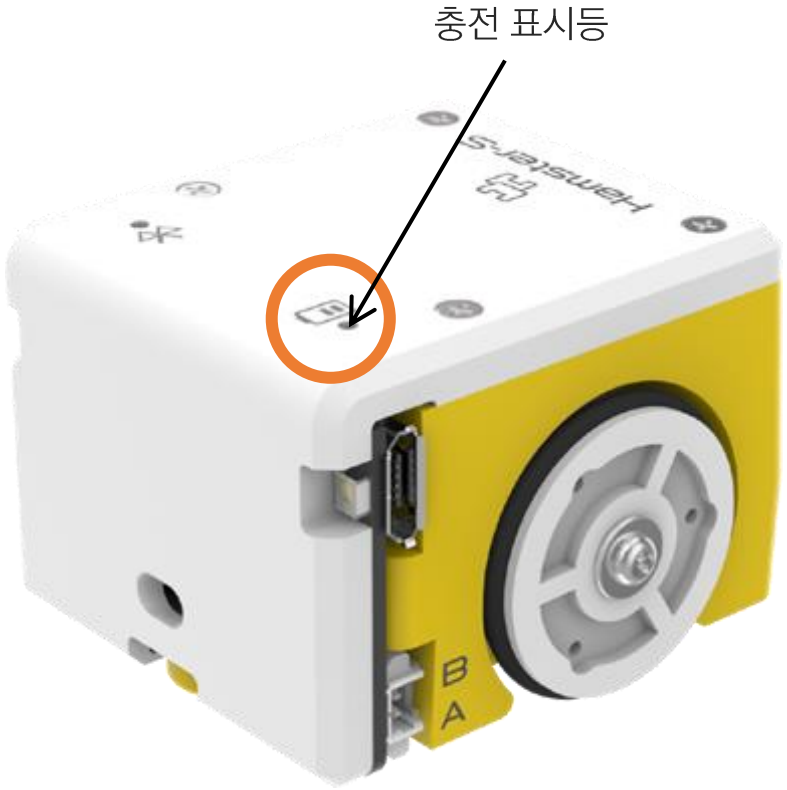
  데이터를 받고 있는 상태입니다.

  컴퓨터 또는 스마트 폰으로부터 데이터를 받고 있는 동안에는 블루투스 연결 표시등이 파란색으로 빠르게 깜박입니다.

- **표시등이 꺼짐**

  로봇의 전원이 꺼진 상태입니다.

01 하드웨어 이해



- **빨간색으로 켜져 있음**

  충전 중입니다.

- **표시등이 꺼짐**

  충전이 완료되었습니다.

- **빨간색으로 깜박임**

  로봇의 배터리가 거의 남아있지 않은 상태입니다.

  햄스터S 로봇의 배터리가 거의 남아있지 않으면 충전 표시등이 빨간색으로 깜박입니다. 이 경우에는 반드시 충전해 주세요.

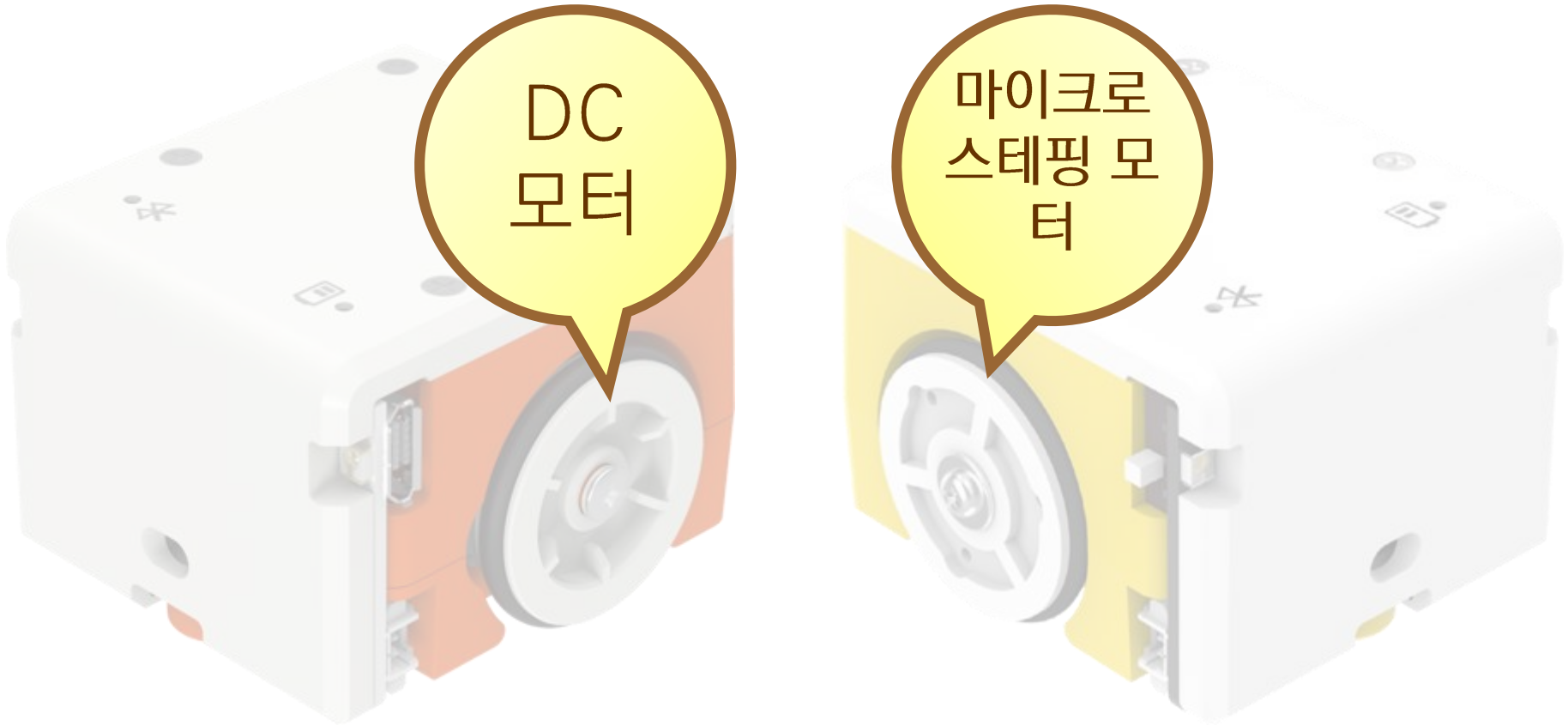
01

하드웨어 이해

- 햄스터와 햄스터S의 주요 차이점입니다.

HAMSTER

HAMSTER-S



하드웨어 이해

- 햄스터와 햄스터S의 주요 차이점입니다.

HAMSTER

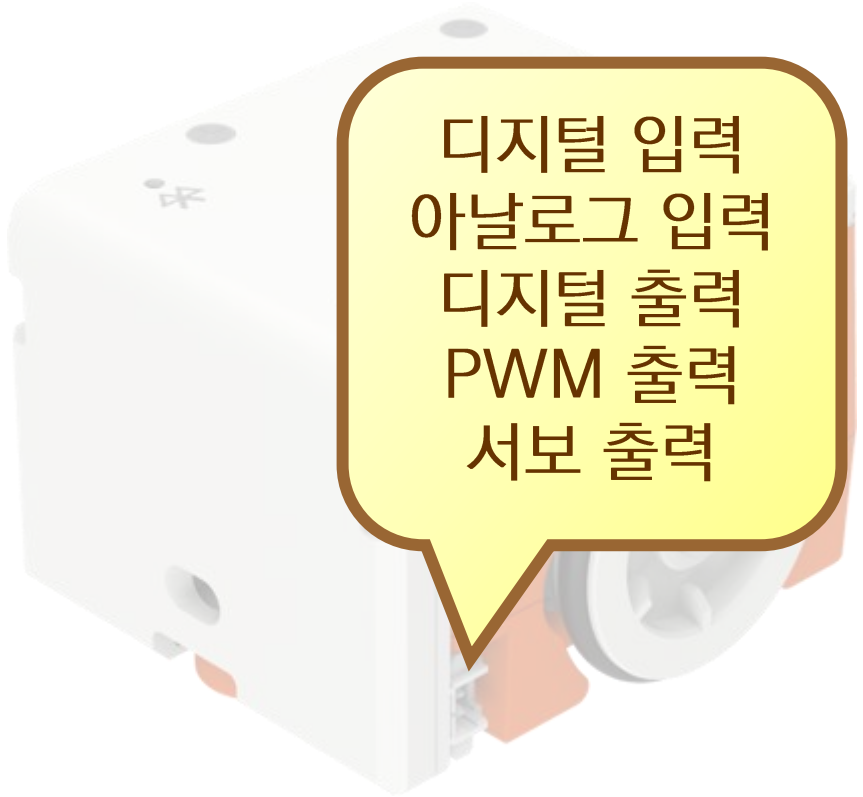
HAMSTER-S

RGB 256 x 256 x 256=16,777,216가지 색상

01 하드웨어 이해

- 햄스터와 햄스터S의 주요 차이점입니다.

HAMSTER

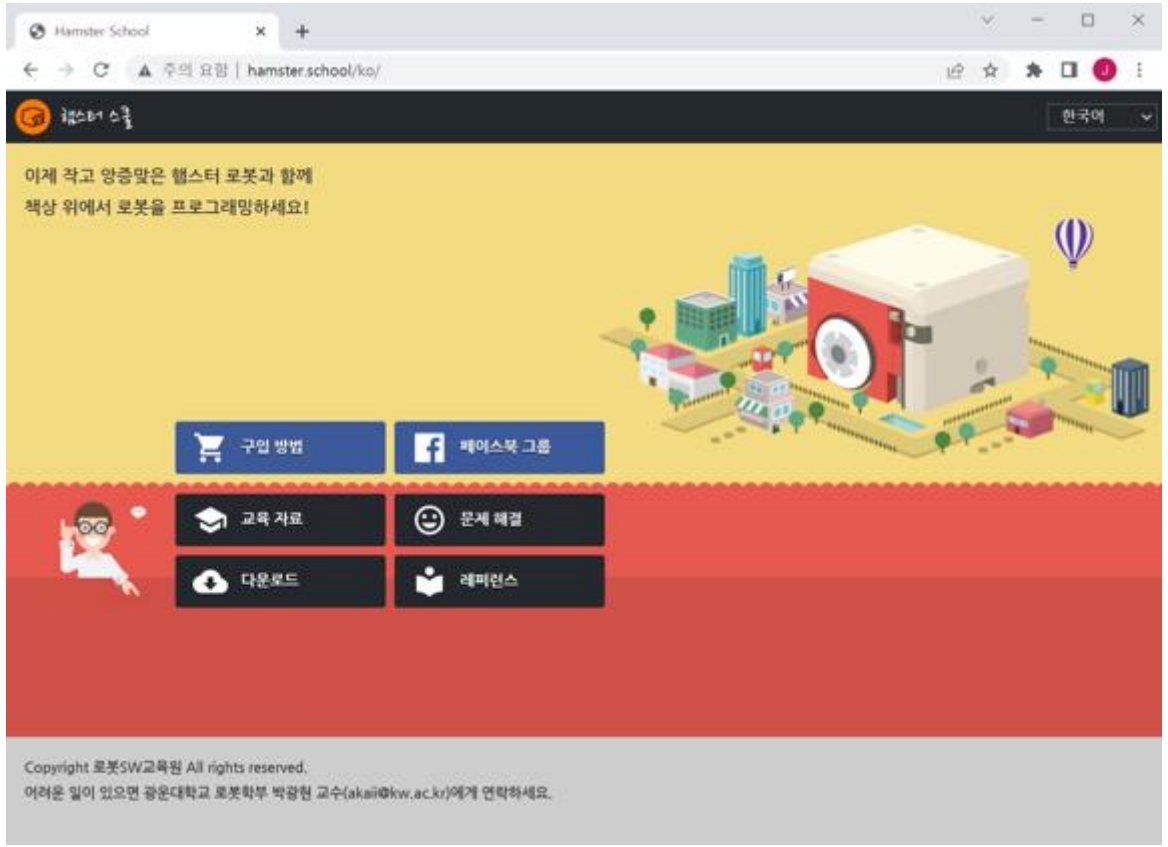


HAMSTER-S

02

디바이스 드라이버 설치

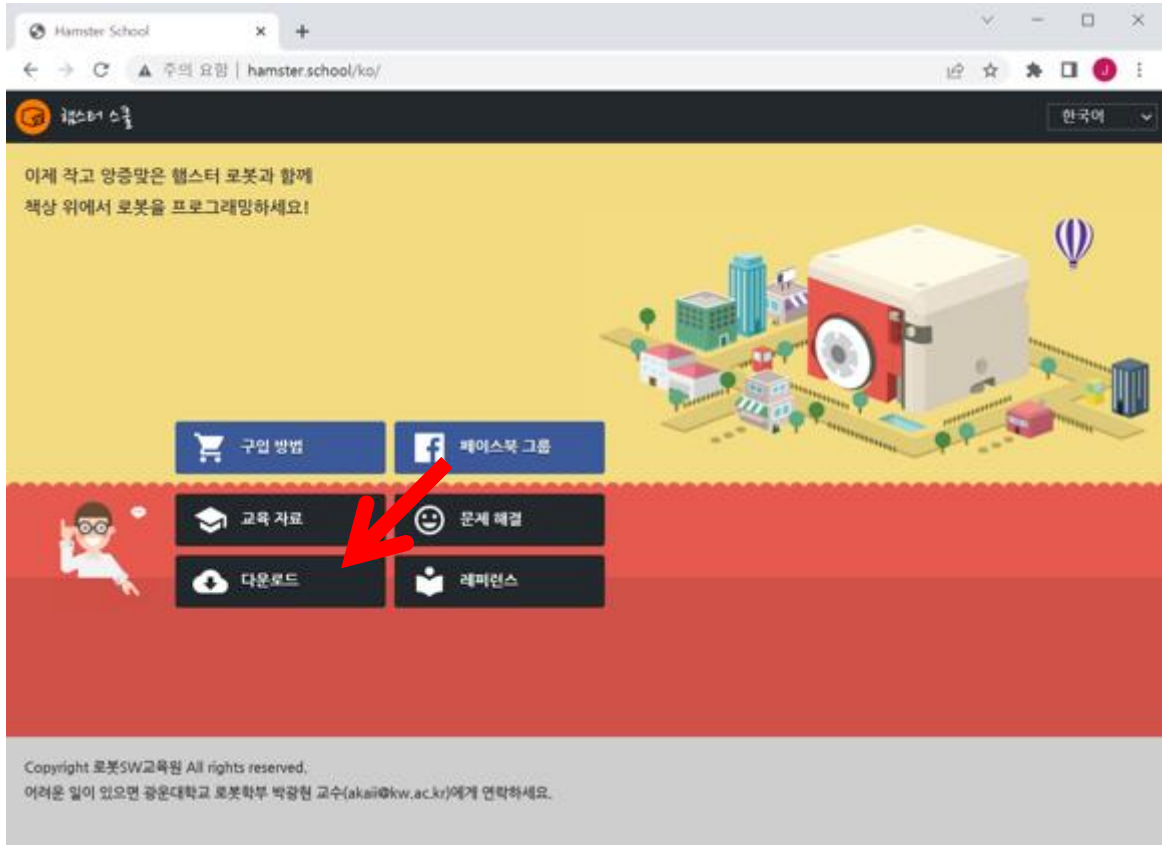
- hamster.school 접속



02

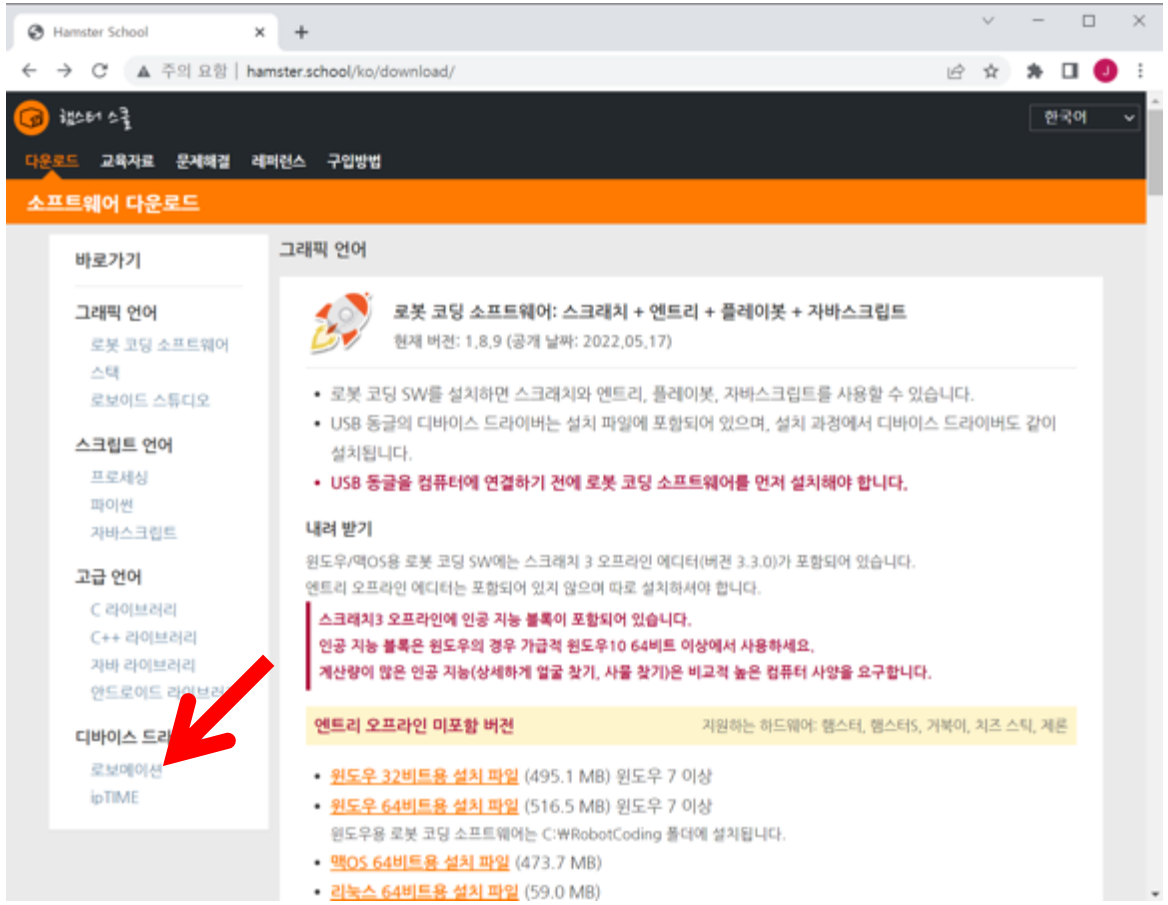
디바이스 드라이버 설치

- **다운로드** 클릭



02 디바이스 드라이버 설치

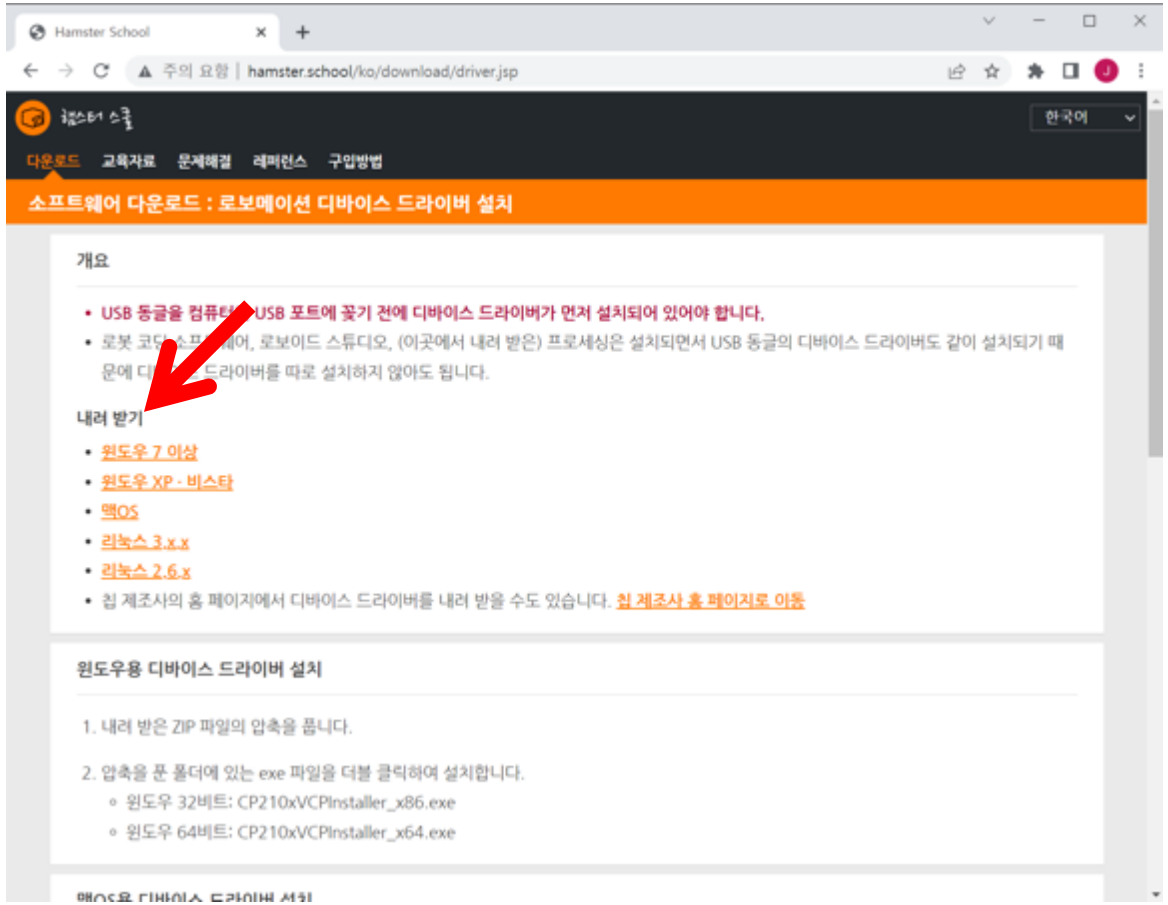
- **디바이스 드라이버 –〉 [로보메이션]** 클릭



02

디바이스 드라이버 설치

- OS에 맞는 설치 파일 다운로드



02

디바이스 드라이버 설치

- **윈도우용 디바이스 드라이버 설치**

1. ZIP 파일 압축 풀기

2. exe 파일 실행
   - 윈도우 32비트: CP210xVCPInstaller_x86.exe
   - 윈도우 64비트: CP210xVCPInstaller_x64.exe

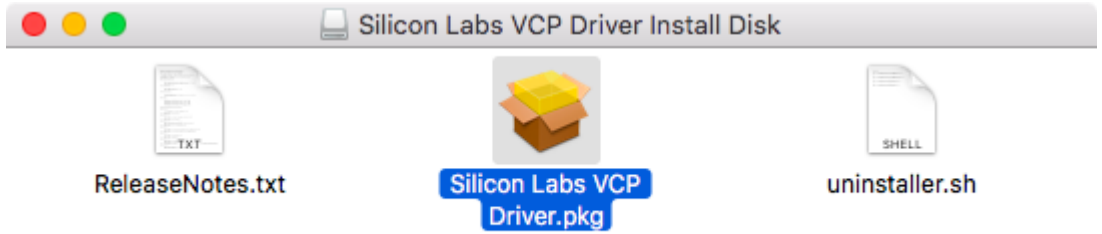
02

디바이스 드라이버 설치

- **맥 OS용 디바이스 드라이버 설치**

1 SiLabsUSBDriverDisk.dmg 디스크 이미지 파일 열기

2 Silicon Labs VCP Driver.pkg 파일 설치



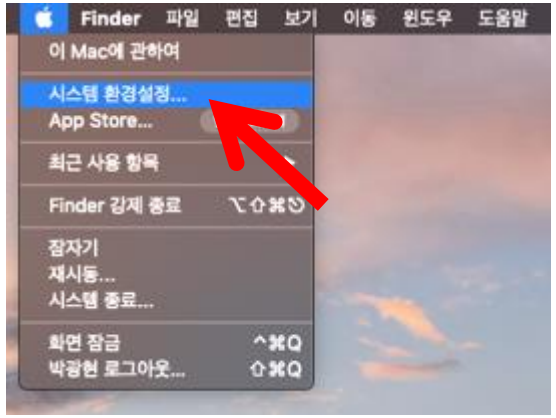
맥OS 버전 10.13(하이 시에라)부터는 사용자 허용 필요

http://hamster.school/ko/help/osx_device_driver.jsp 참고

02

디바이스 드라이버 설치

- 맥 OS용 디바이스 드라이버 설치



02

디바이스 드라이버 설치

- **리눅스용 디바이스 드라이버 설치**

1 tar.gz 파일 압축 풀기 (리눅스 버전에 따라 달리 압축 풀기)

```
tar xvfzp device-driver-linux.3.x.x.tar.gz
```

```
tar xvfzp device-driver-linux.2.6.x.tar.gz
```

2 drivers 폴더가 생성되었는지 확인

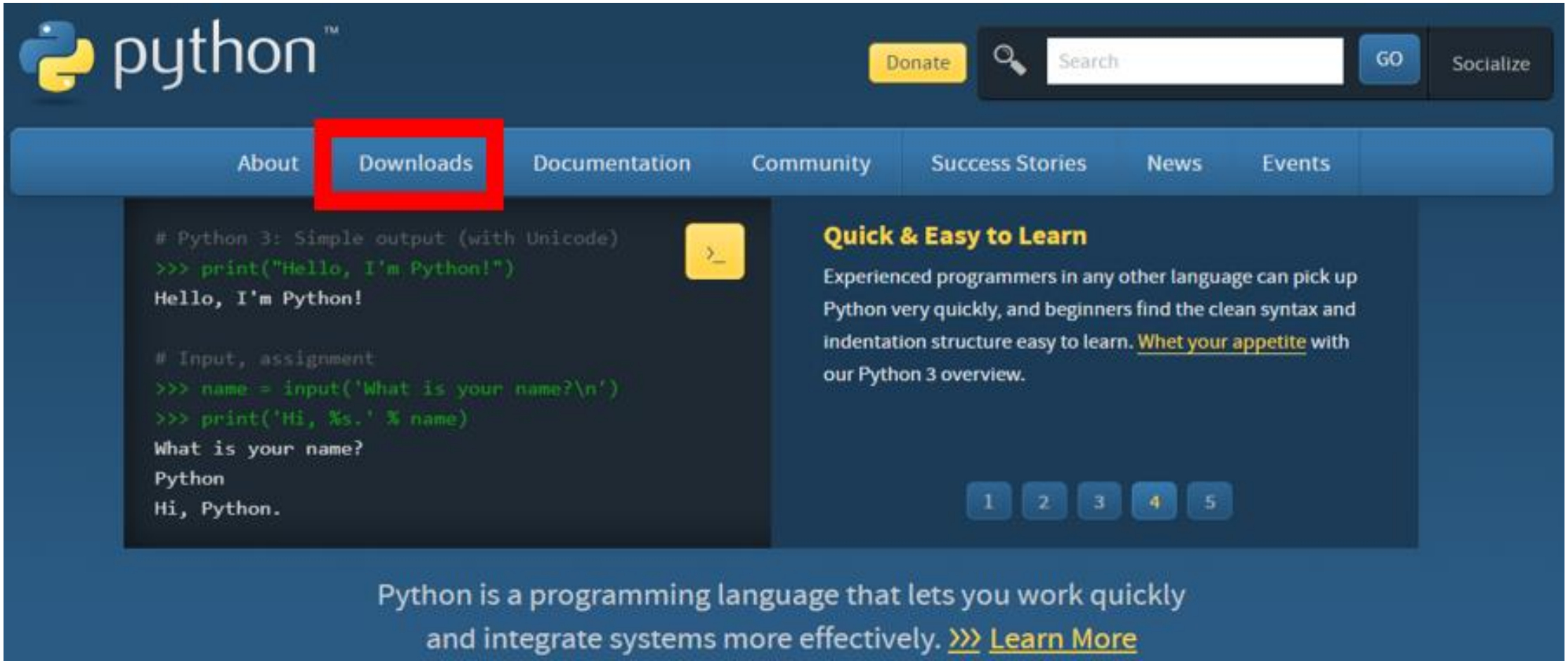
3 터미널에서 root 계정 로그인 & drivers 폴더로 이동

4 쉘 스크립트 실행

```
./setup.sh
```

03 파이썬 설치

- https://www.python.org 접속하고 아래 그림과 같이 Downloads 버튼을 클릭합니다.



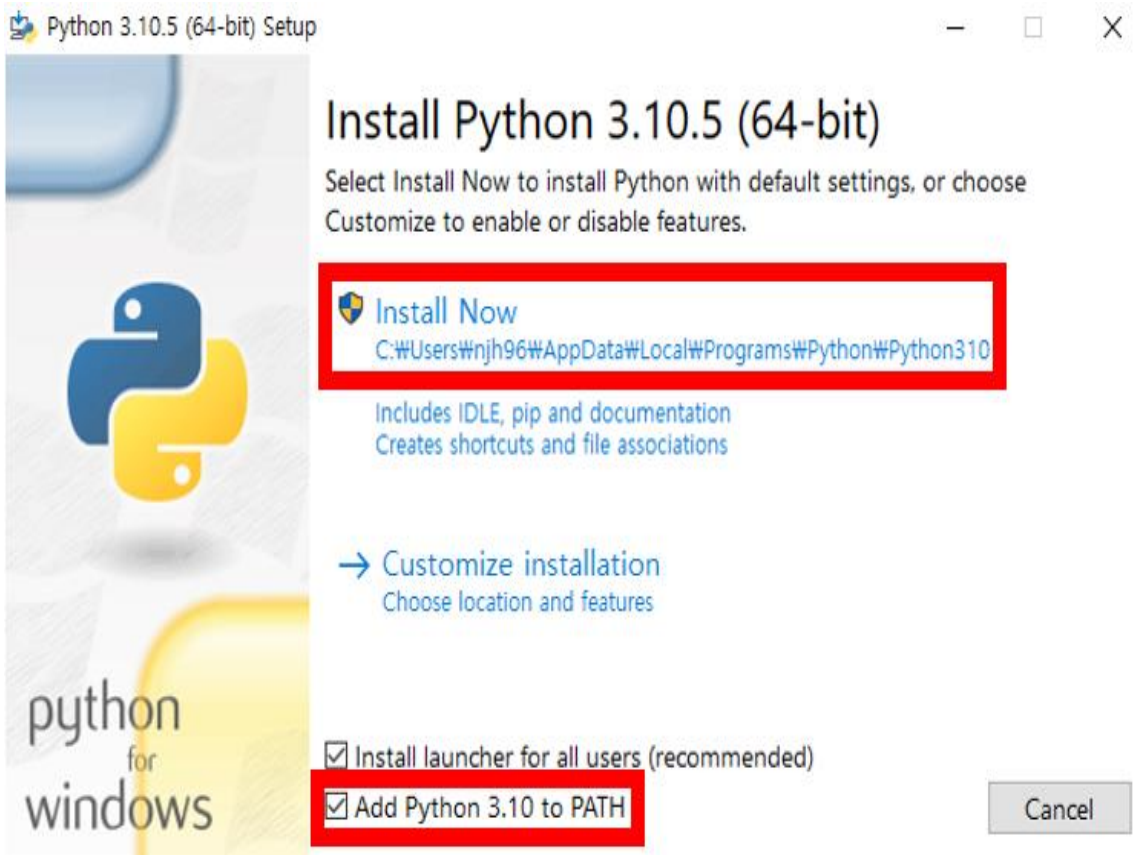
03 파이썬 설치

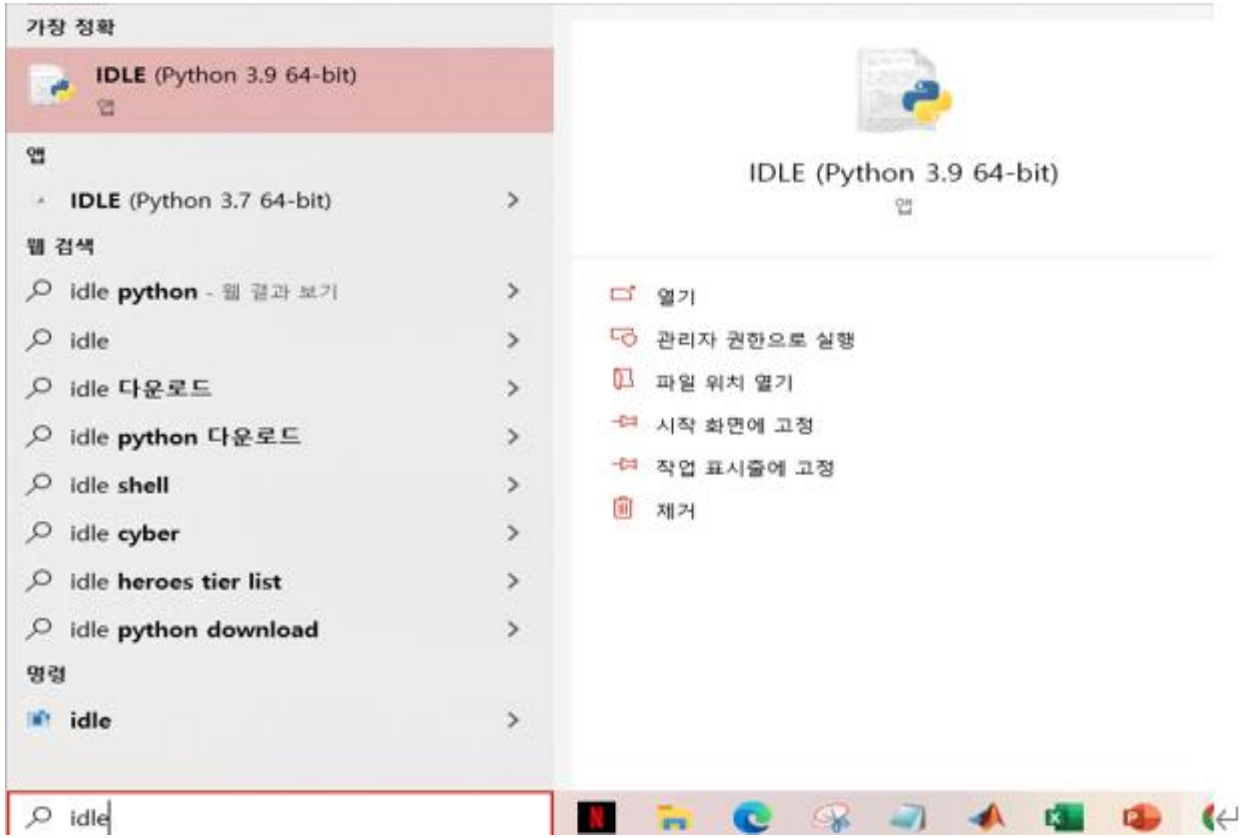
- Download Python 3.10.5 클릭하여 설치합니다.

03

파이썬 설치

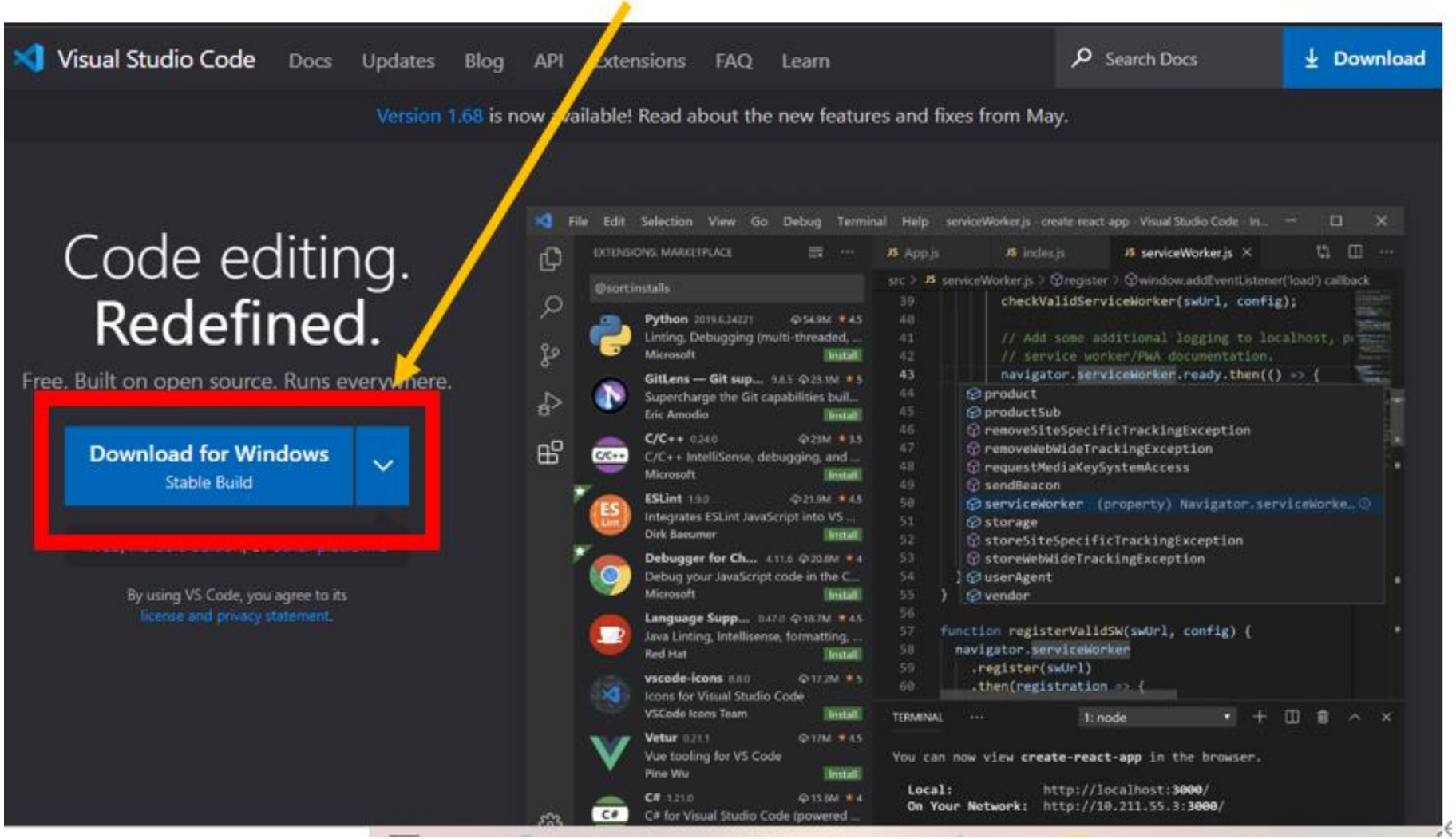
- Add Python 3.10 to PATH를 체크 후 Install Now를 클릭합니다.
- 검색창에 “IDLE” 입력 후 설치되었는지 확인할 수 있습니다.





03 VS Code 설치

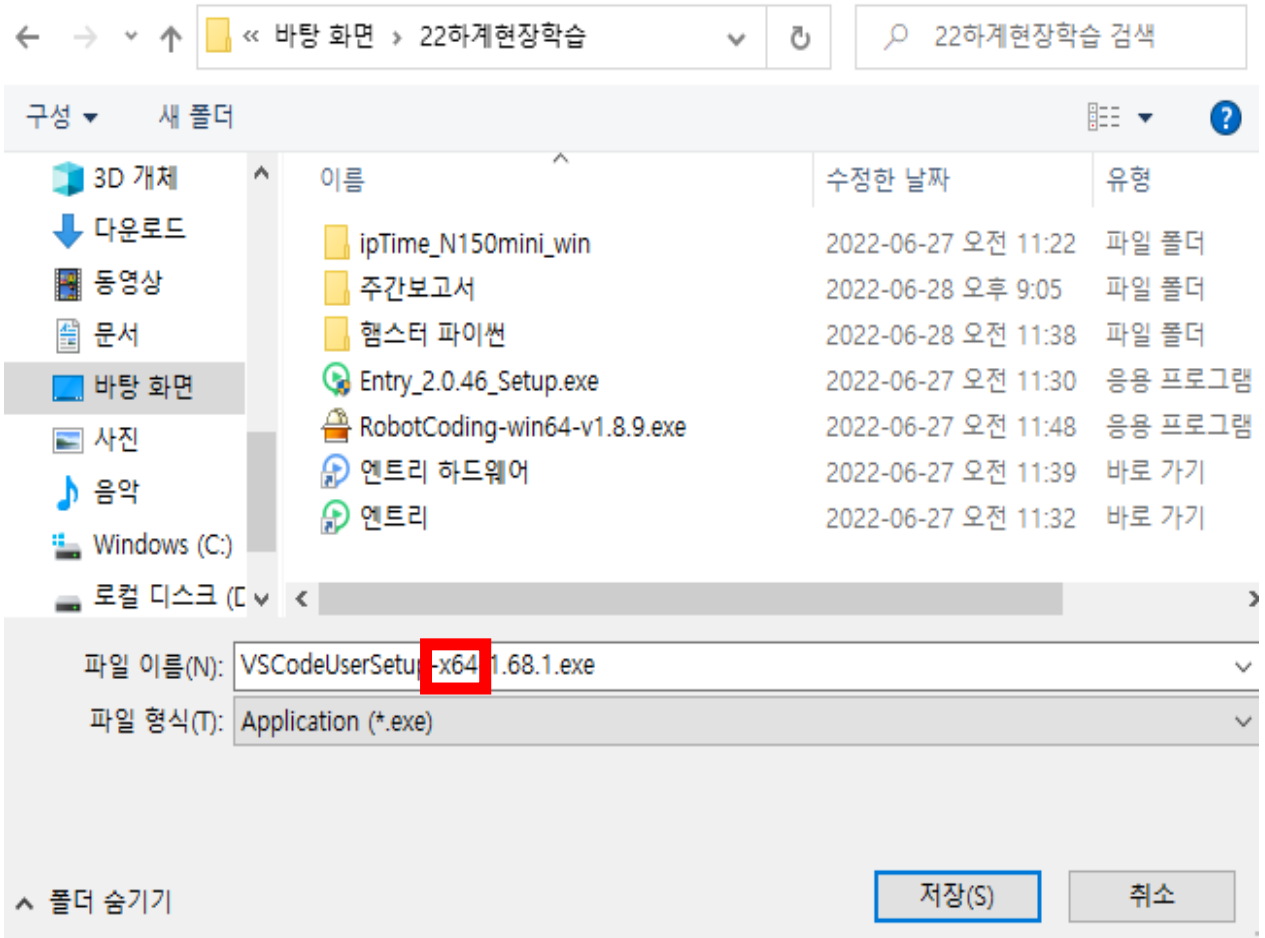
- https://code.visualstudio.com/ 접속하고 Download for windows 클릭합니다.

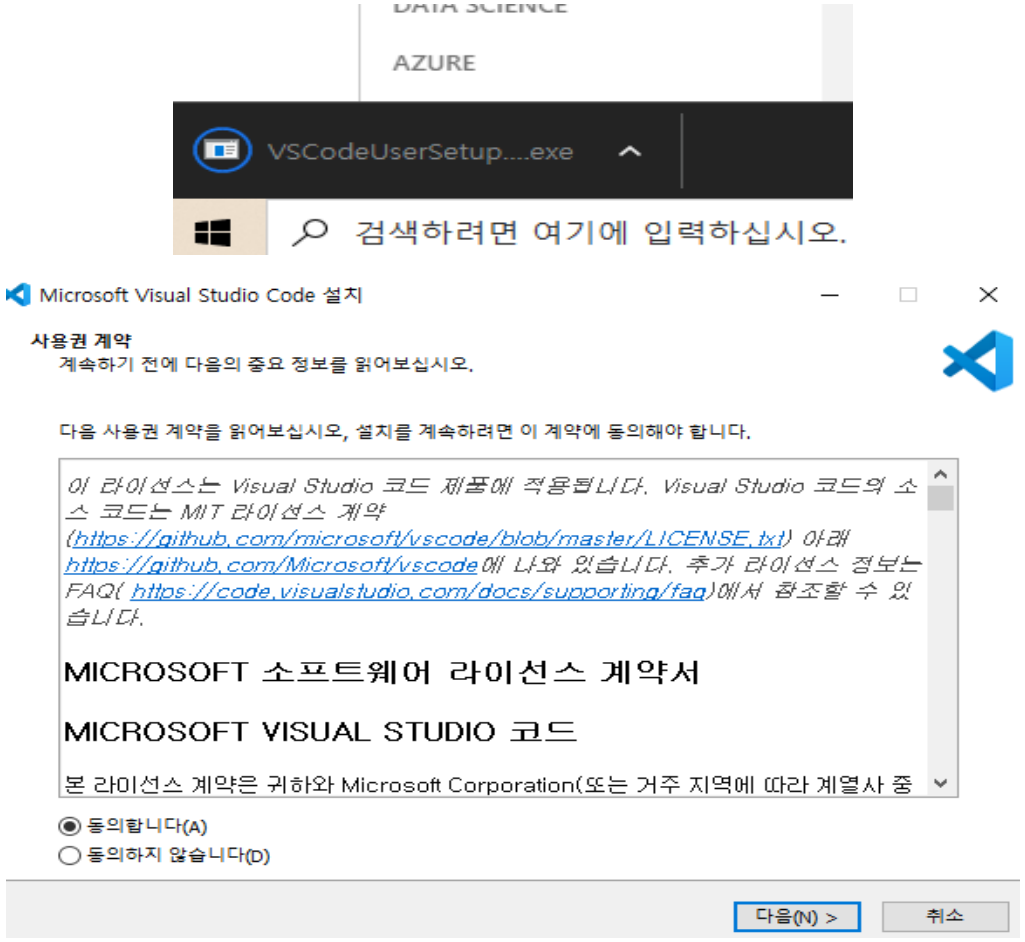


03 VS Code 설치

1) 본인 PC의 운영체제가 맞는지 확인 후, “저장”을 클릭합니다.

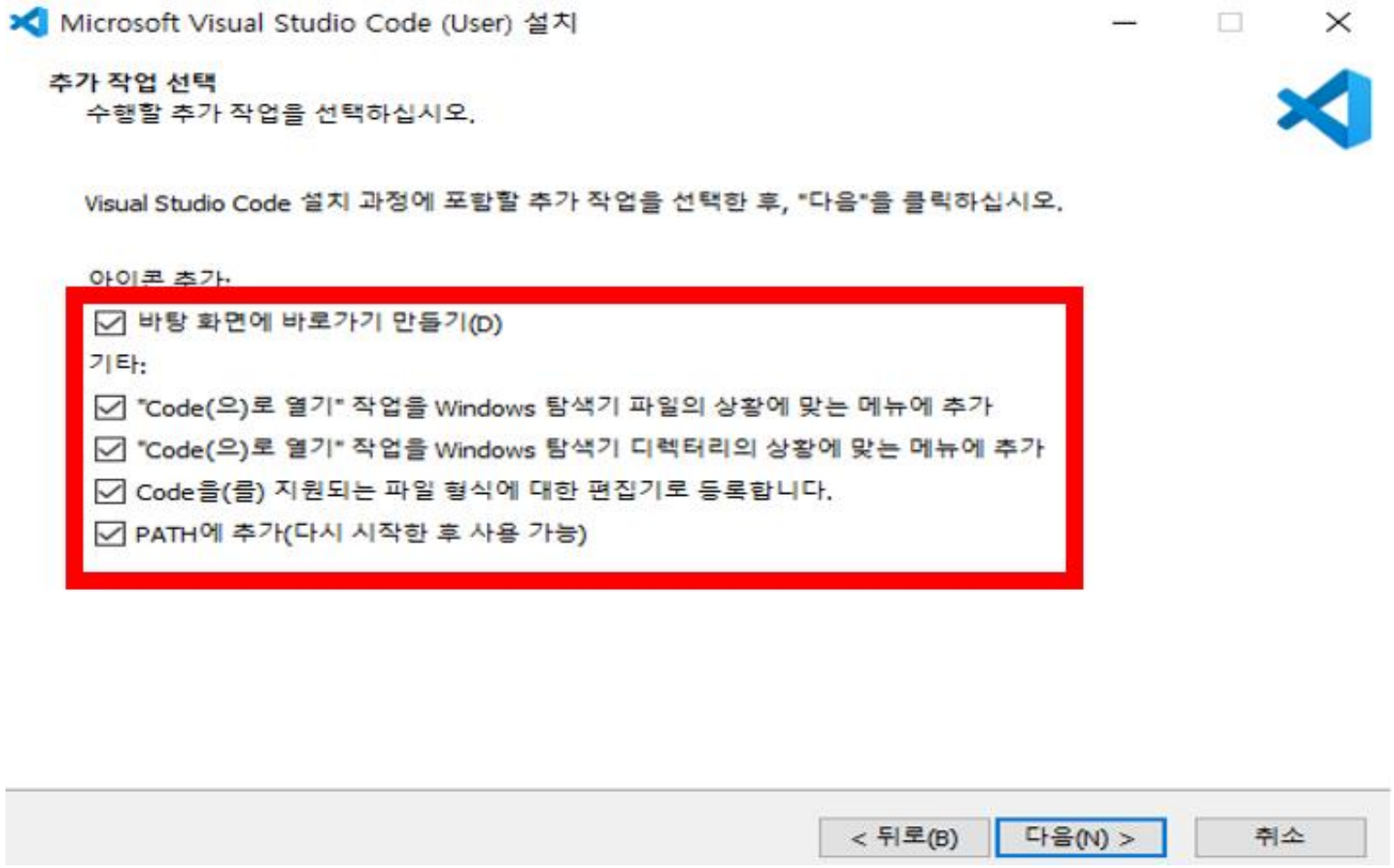
2) 설치된 파일을 실행하고 “동의합니다” 선택 후 “다음”을 클릭합니다.





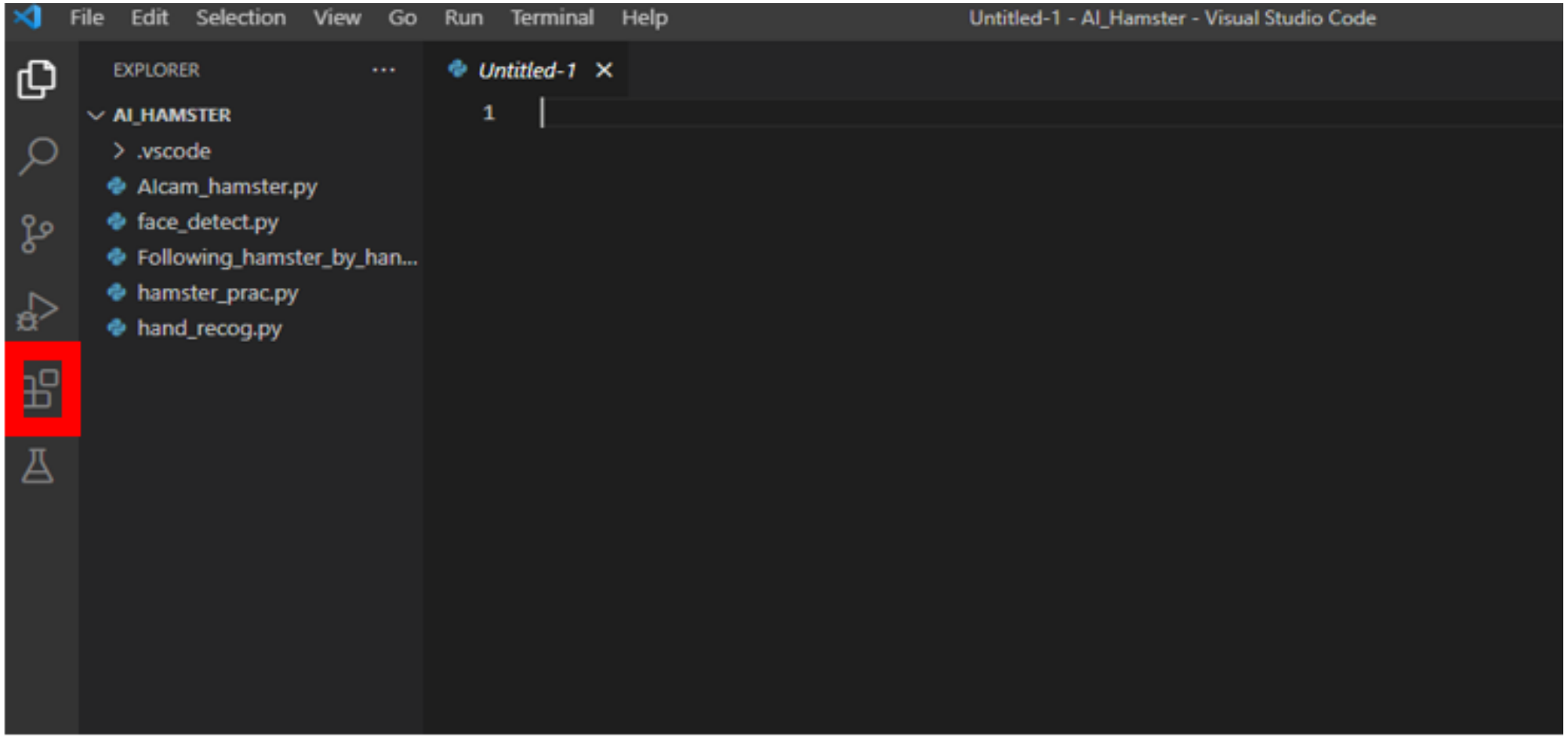
03 VS Code 설치

- 본인이 원하는 것만 체크한 후 “다음”을 클릭합니다.



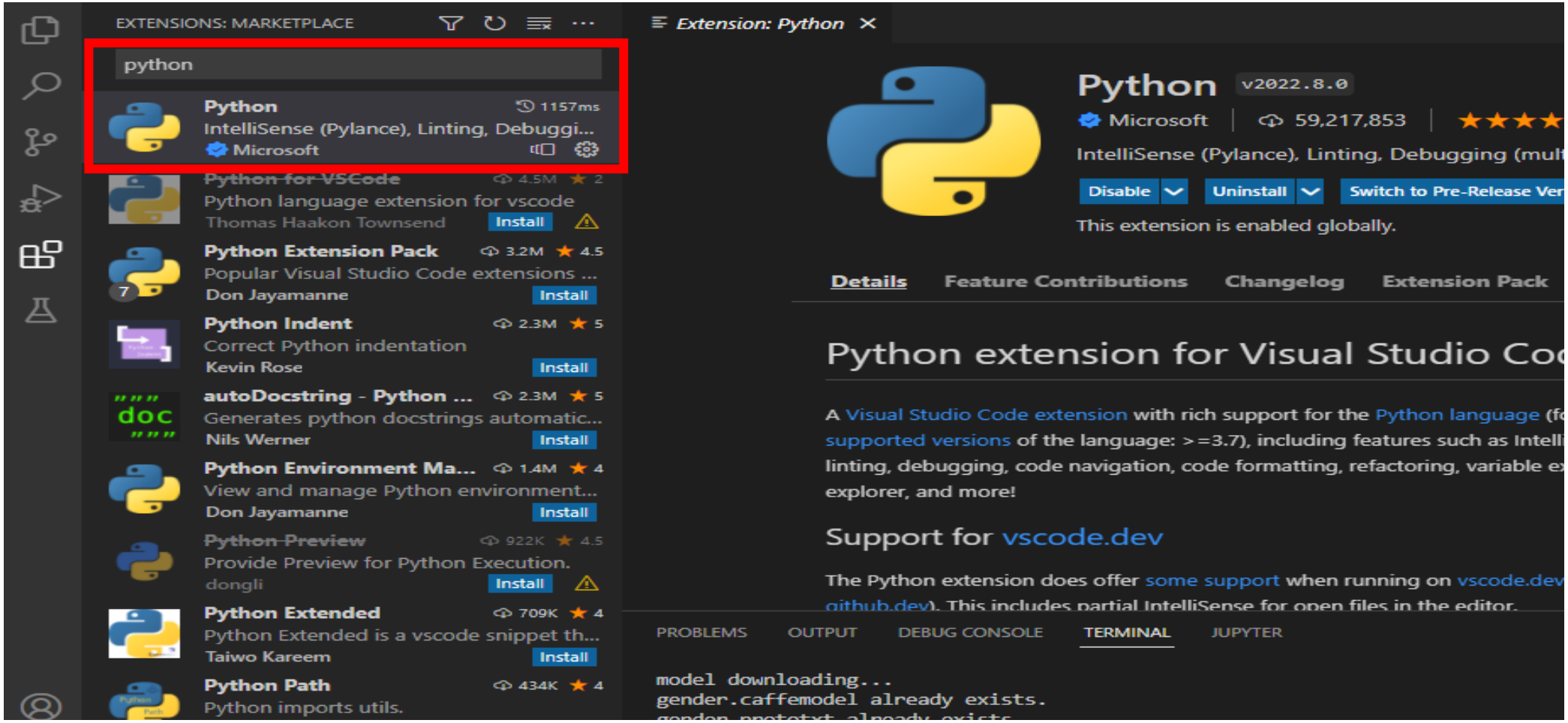
03 VS Code 설치

- VS Code 실행 후 빨간 박스를 클릭합니다.



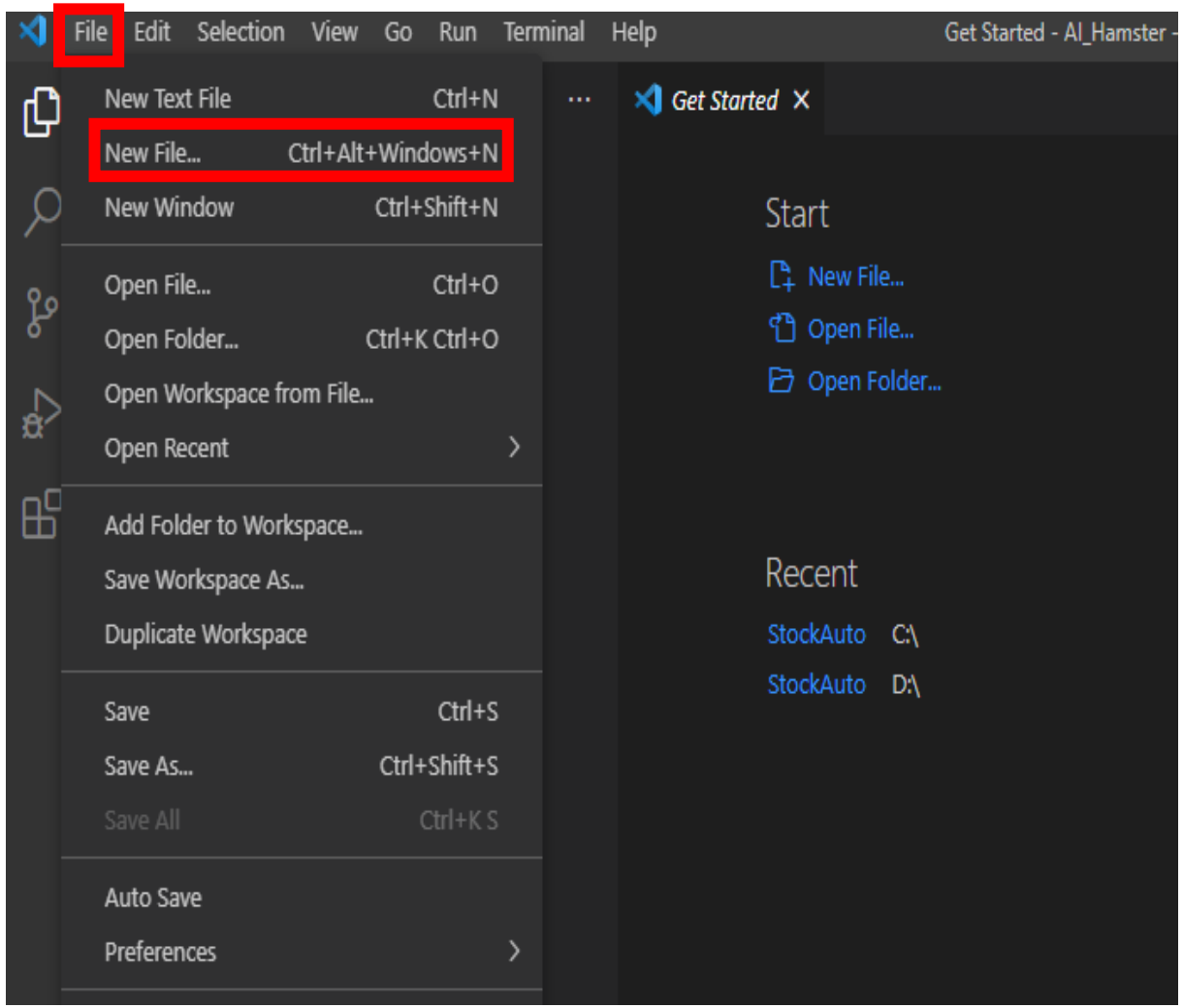
03 VS Code 설치

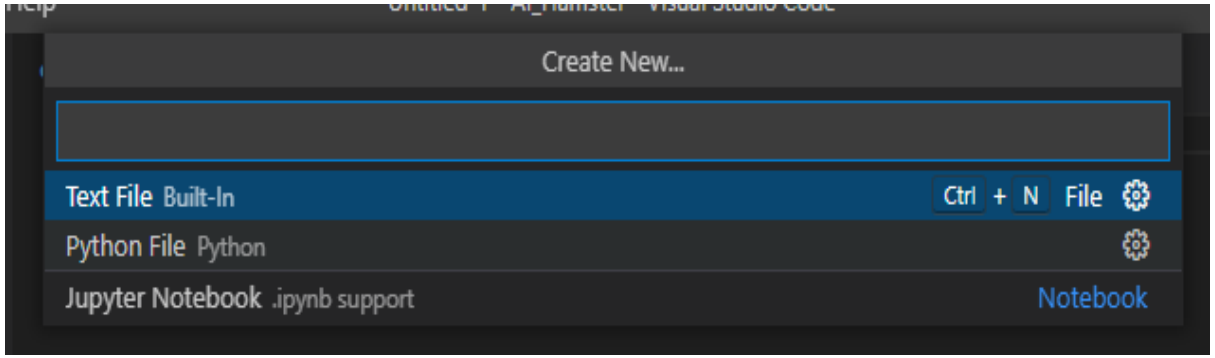
- Python입력 후, 가장 상단에 있는 “Python”을 설치합니다.(Install 버튼)



03 VS Code 설치

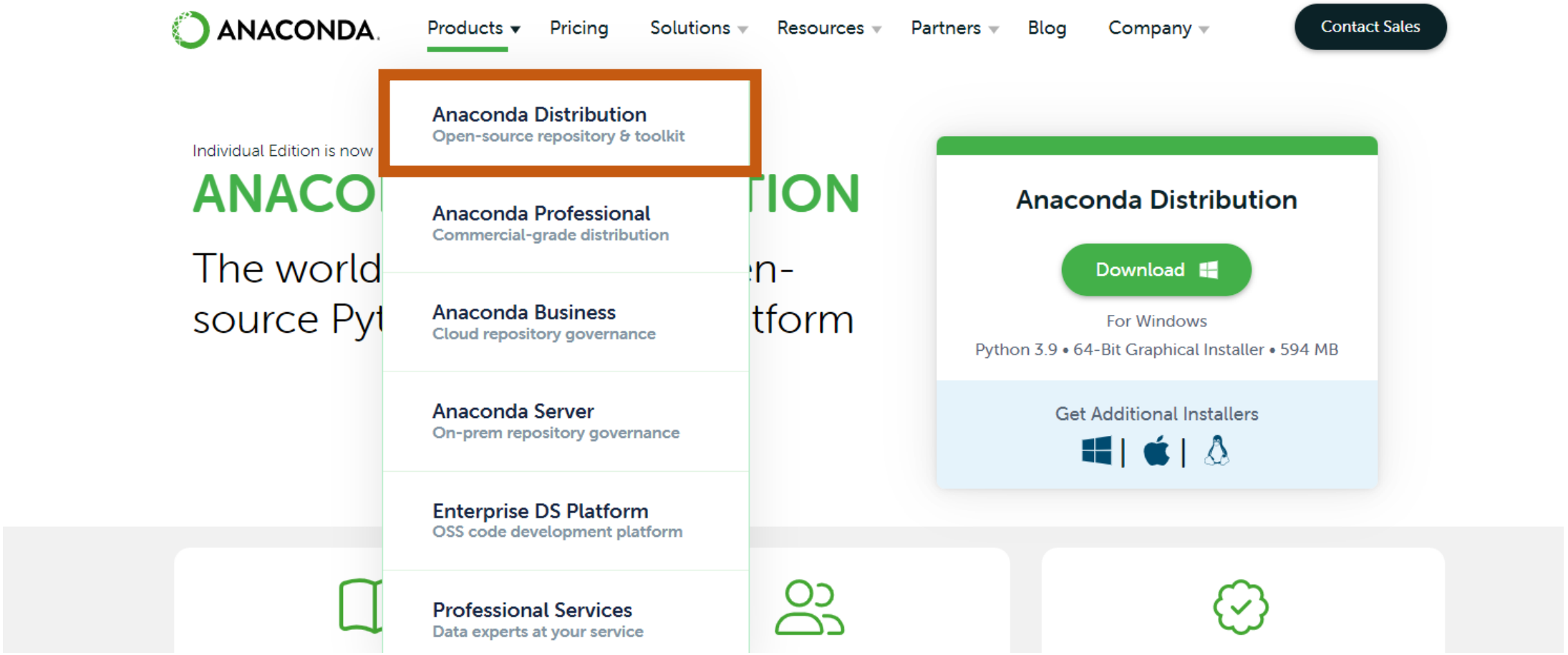
- File에서 New File을 클릭 후, “Text File” 또는 “Python File”을 선택합니다.





03 아나콘다 설치

- https://www.anaconda.com을 접속하고 Products에서 Anaconda Distribution을 클릭합니다.

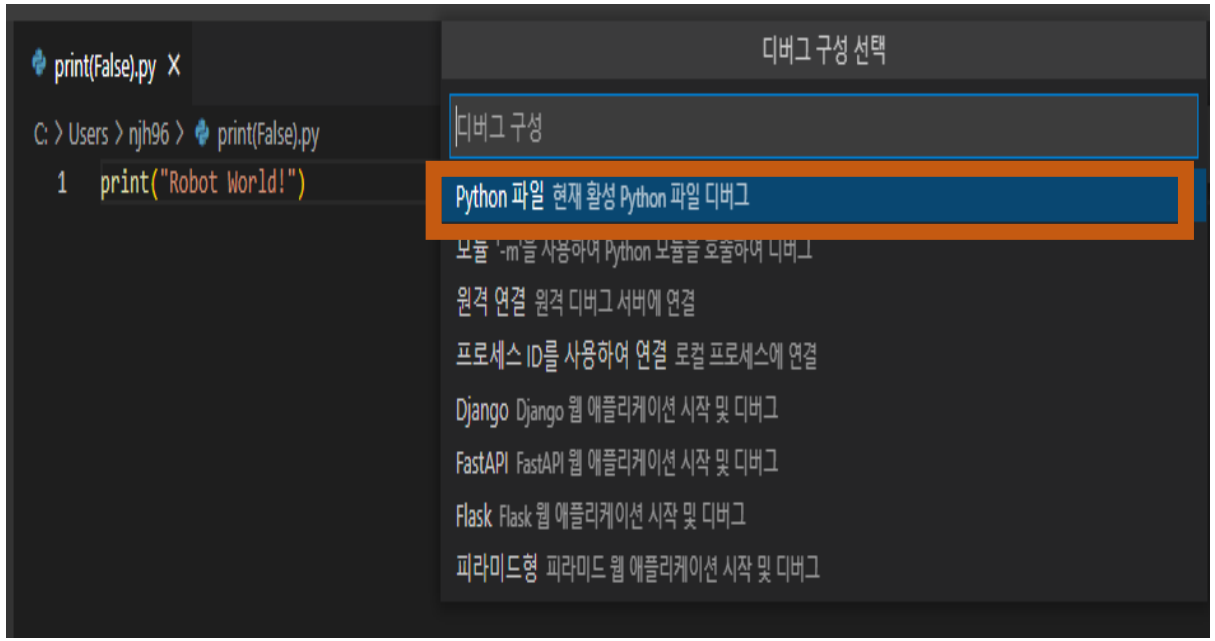


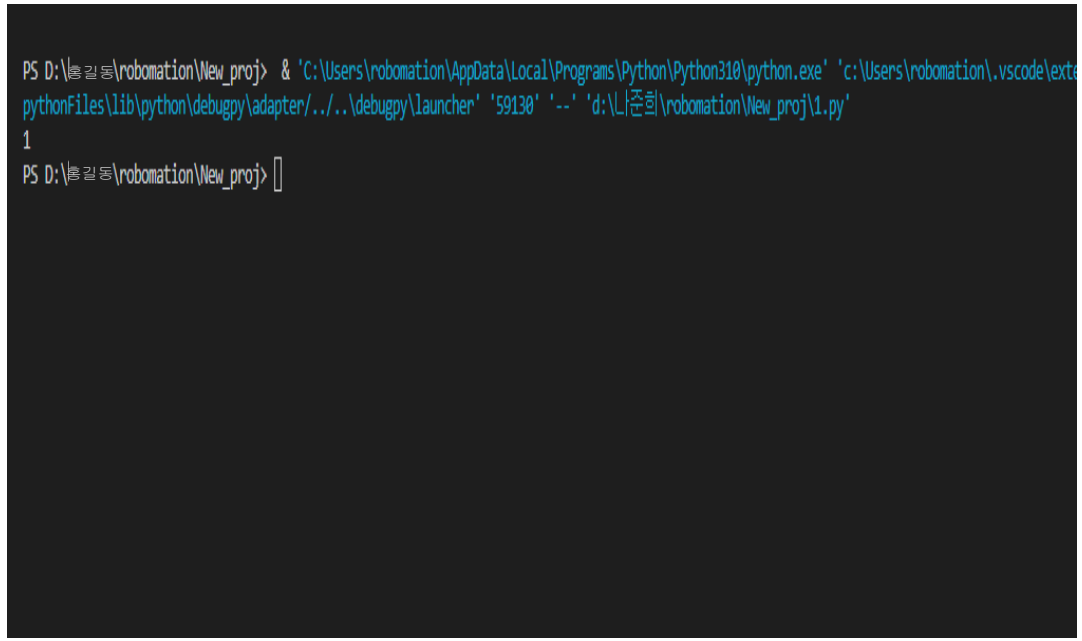
03 아나콘다 설치

- 해당 창에서 제일 아래로 내려가 본인 PC의 운영체제에 맞는 Installer를 클릭하여 설치합니다.

04 라이브러리 설치

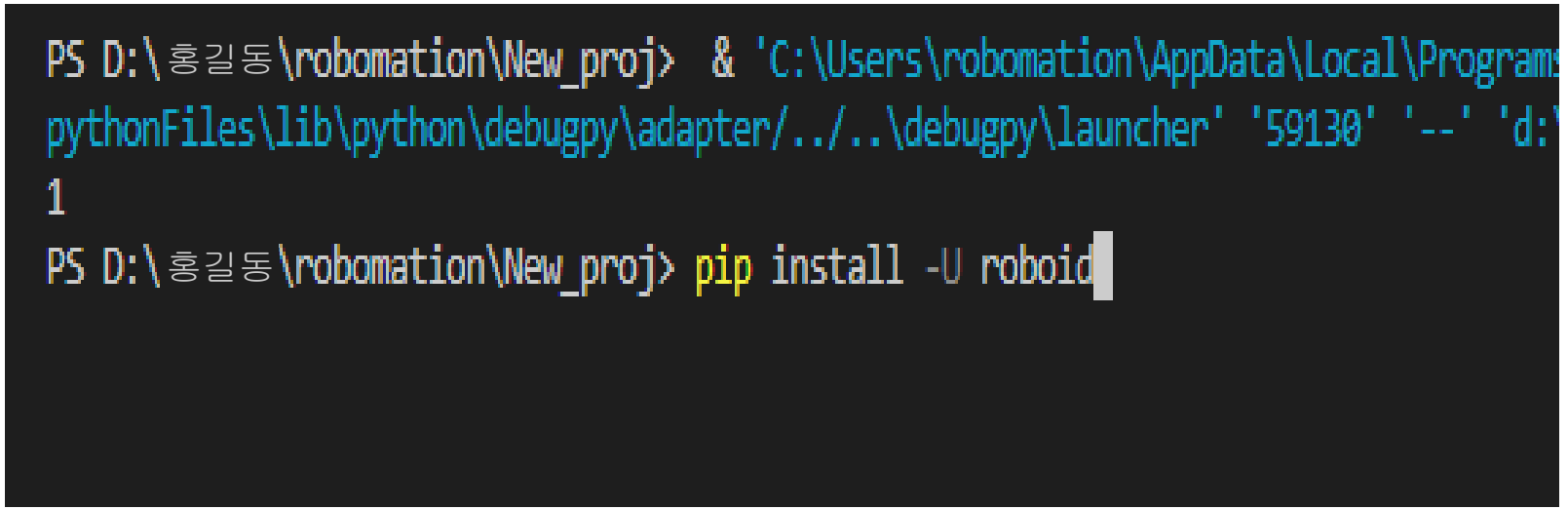
- VS code에서 New File을 열어줍니다. 예제 코드를 작성 후 F5를 눌러 실행시킨 다음 디버그 구성에서 Python파일을 선택합니다.
- 실행 결과, 아래에 오른쪽 그림과 같이 Terminal 창에 결과가 출력됩니다.





04 라이브러리 설치

- 터미널 창에 “pip install –U roboid”를 입력하여 roboid 라이브러리를 설치합니다.



05

로봇과 컴퓨터 연결

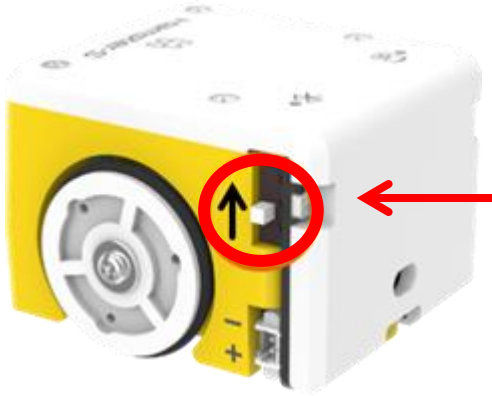
1 “USB to BLE Dongles” 을 컴퓨터에 연결

블루투스 연결 표시등

- Dongles을 컴퓨터에 연결 시 파란색 불빛으로 천천히 깜박입니다.
  표시등이 파란색이 아니라면,
  -〉 접촉 불량: 동글을 살짝 눌러 접촉면을 넓게 해봅니다.
  -〉 포트 문제: PC의 포트 문제일수도 있으니 다른 포트에 연결해봅니다.
  -〉 Dongles 교체

05 로봇과 컴퓨터 연결

2 햄스터 로봇의 전원 스위치를 올린 후, 동글과 가까이 두기



전원 켜기

- 햄스터 전원을 키고 동글과 가까이 두면 삐 소리가 나고 햄스터 로봇과 동글이의 불빛이 계속 파란색으로 켜지거나, 빠르게 깜빡입니다.

  -〉 소리가 나지 않는다면

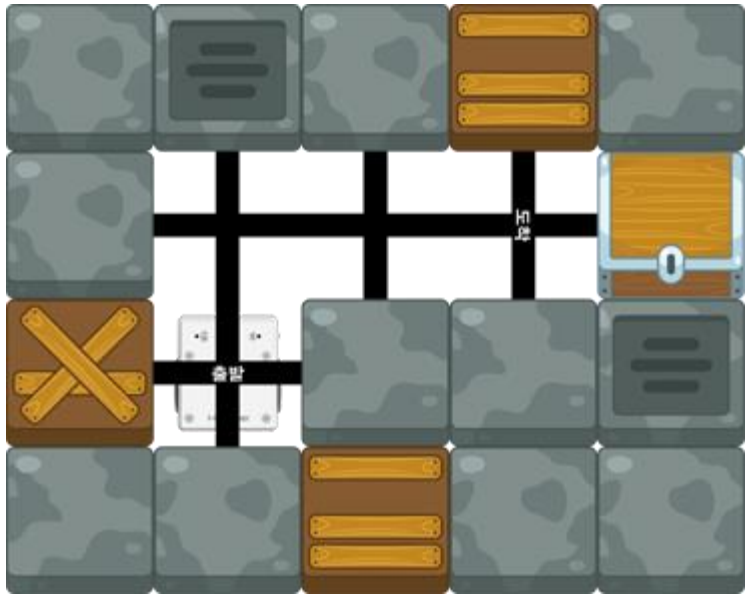
  - 햄스터의 배터리가 충분한지 확인(배터리가 없으면 충전)

2차시

말판 이동하기1
이동하고 회전하기

01 동굴 탐험 (순차)

- 햄스터 로봇이 길을 떠나 동굴 속을 탐험하려고 합니다. 동굴 속에는 햄스터가 좋아하는 해바라기 씨가 가득한 보물 상자가 있습니다. 아래 그림에서 햄스터 로봇이 보물 상자 앞까지 이동하려면 어떻게 움직여야 할지 생각해 봅시다. 도착 지점으로 이동한 후에는 보물 상자를 열 수 있도록 햄스터 로봇이 보물 상자 방향을 바라보아야 합니다.



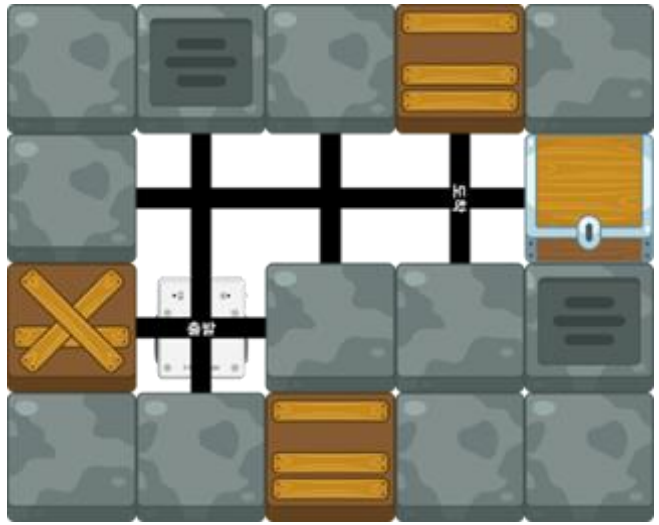
- 말판 내려 받기(PDF) : http://hamster.school/repository/tutorial/class/move_on_board_pdf_ko.zip

01 동굴 탐험 (순차)

- 다음 메소드들을 사용하면 말판 위에서 이동하거나 회전할 수 있습니다.

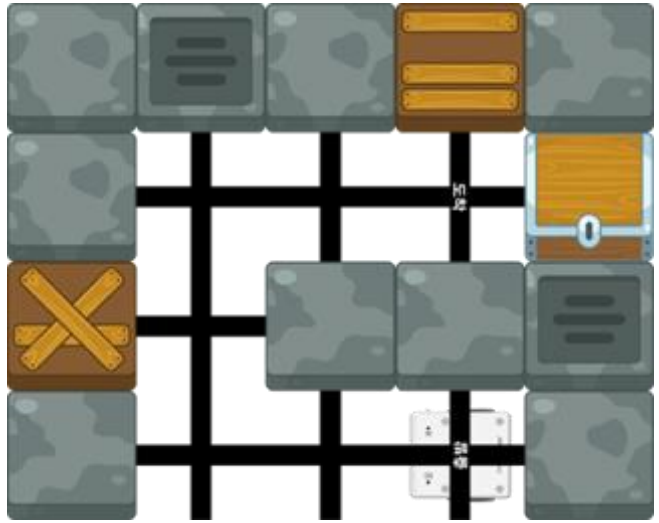
```
from roboid import *

hamster = HamsterS()

hamster.board_forward() # 검은색 격자로 구성된 말판 위에서 한 칸 앞으로 이동합니다.

hamster.board_left() # 검은색 격자로 구성된 말판 위에서 왼쪽 방향으로 제자리에서 90도 회전합니다.

hamster.board_right() # 검은색 격자로 구성된 말판 위에서 오른쪽 방향으로 제자리에서 90도 회전합니다..
```

01 동굴 탐험 (순차)



- 햄스터 로봇을 말판의 출발 위치에 방향을 맞추어 올려놓고 보물 상자 앞까지 이동시켜 봅시다.

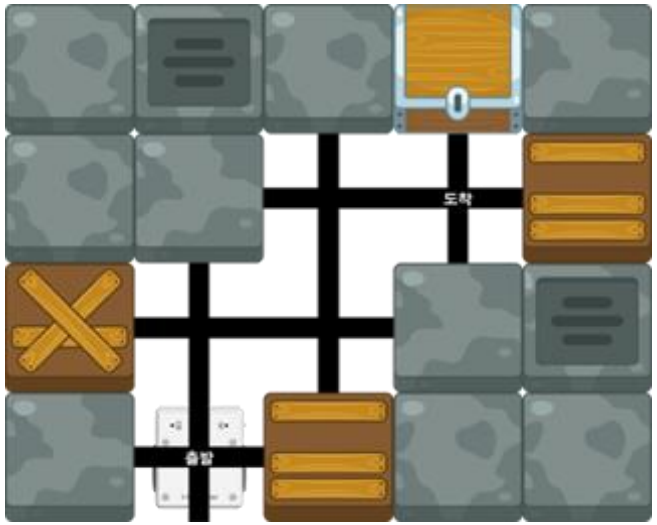
```
from roboid import *

hamster = HamsterS()

hamster.board_forward()
hamster.board_right()
hamster.board_forward()
hamster.board_forward()
```



- 두 번째 동굴도 탐험해 봅시다.

```
from roboid import *

hamster = HamsterS()

hamster.board_forward()
hamster.board_forward()
hamster.board_right()
hamster.board_forward()
hamster.board_forward()
hamster.board_right()
hamster.board_forward()
hamster.board_forward()
```

01 동굴 탐험 (순차)



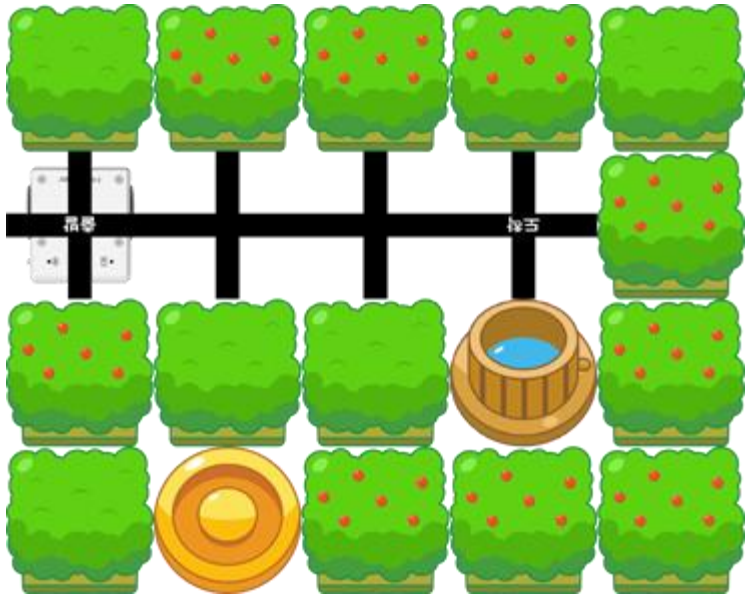
- 세 번째 동굴도 탐험해 봅시다.

```
from roboid import *

hamster = HamsterS()

hamster.board_forward()
hamster.board_right()
hamster.board_forward()
hamster.board_left()
hamster.board_forward()
hamster.board_right()
hamster.board_forward()
hamster.board_left()
```

02

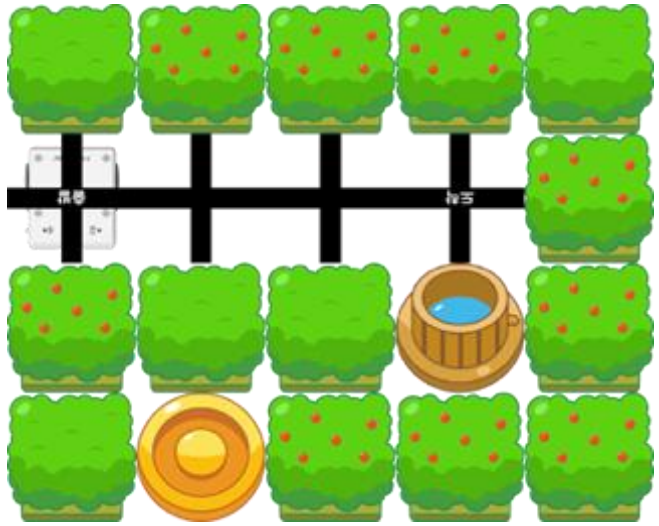
우물 찾기 (횟수 반복)

- 햄스터 로봇이 목이 마릅니다. 숲 속 우물을 향해 달려가야 합니다. 아래 그림에서 햄스터 로봇이 우물 앞까지 이동하려면 어떻게 움직여야 할지 생각해 봅시다. 도착 지점으로 이동한 후에는 햄스터 로봇이 물을 마실 수 있도록 우물 방향을 바라보아야 합니다.



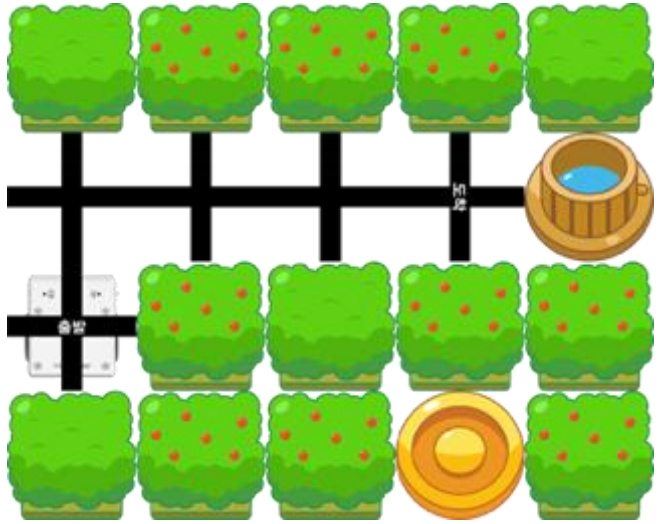
02

우물 찾기 (횟수 반복)



- 햄스터 로봇을 말판의 출발 위치에 방향을 맞추어 올려놓고 우물 앞까지 이동시켜 봅시다.

```
from roboid import *

hamster = HamsterS()

hamster.board_left()
for i in range(3):
    hamster.board_forward()
hamster.board_right()
```

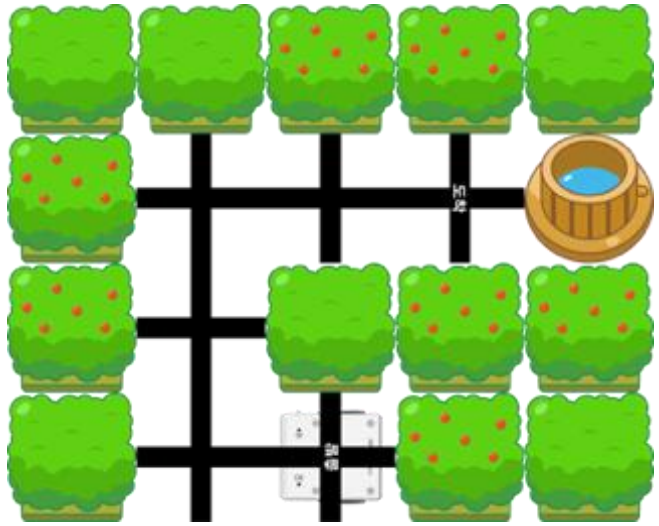


- 두 번째 우물도 찾아 봅시다.

```
from roboid import *

hamster = HamsterS()

hamster.board_forward()
hamster.board_right()
for i in range(3):
    hamster.board_forward()
```

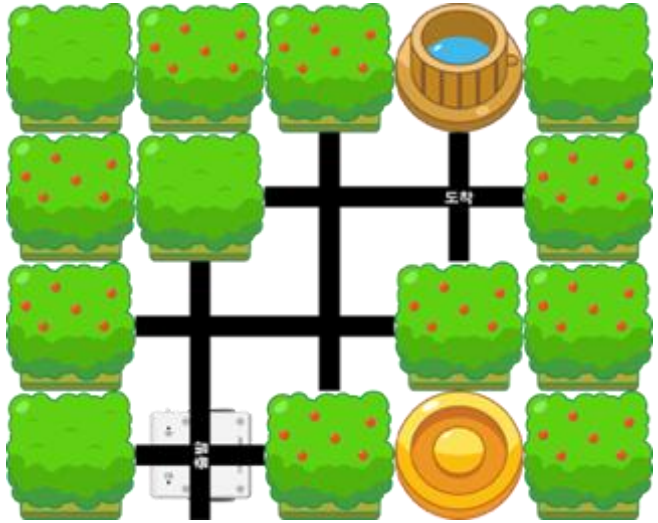
02

우물 찾기 (횟수 반복)



- 세 번째 우물도 찾아 봅시다.

```
from roboid import *

hamster = HamsterS()

for i in range(2):
    hamster.board_forward()
    hamster.board_right()
    hamster.board_forward()
hamster.board_forward()
```

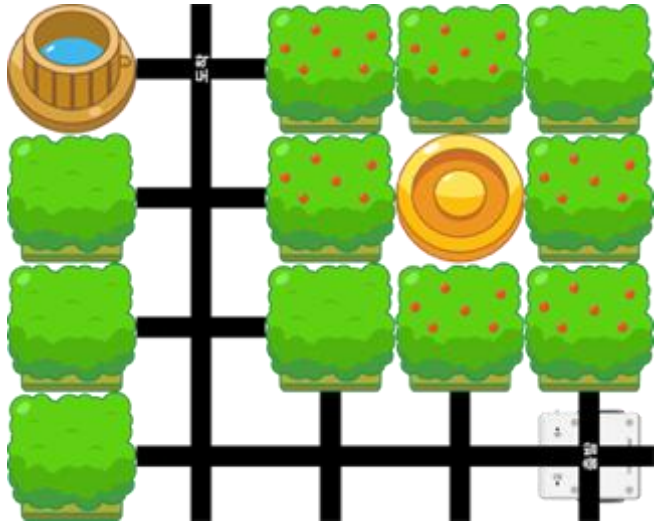


- 네 번째 우물도 찾아 봅시다.

```
from roboid import *

hamster = HamsterS()

hamster.board_right()
for i in range(2):
    hamster.board_forward()
    hamster.board_right()
    hamster.board_forward()
    hamster.board_left()
```

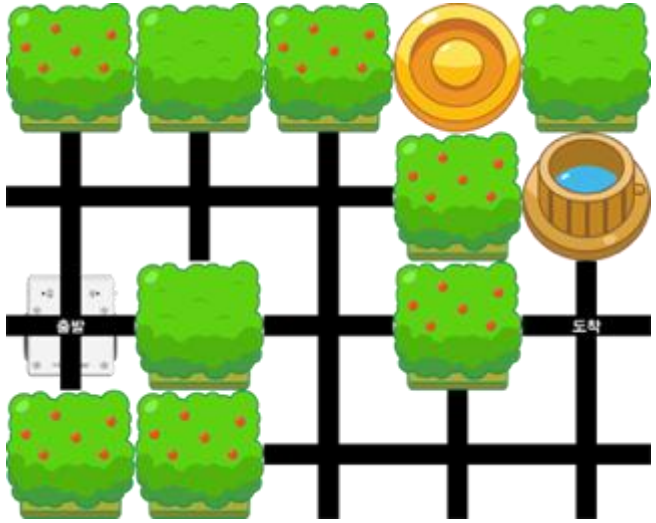
02

우물 찾기 (횟수 반복)



- 다섯 번째 우물도 찾아 봅시다.

```
from roboid import *

hamster = HamsterS()

for i in range(3):
    hamster.board_forward()
hamster.board_right()
for i in range(3):
    hamster.board_forward()
hamster.board_left()
```

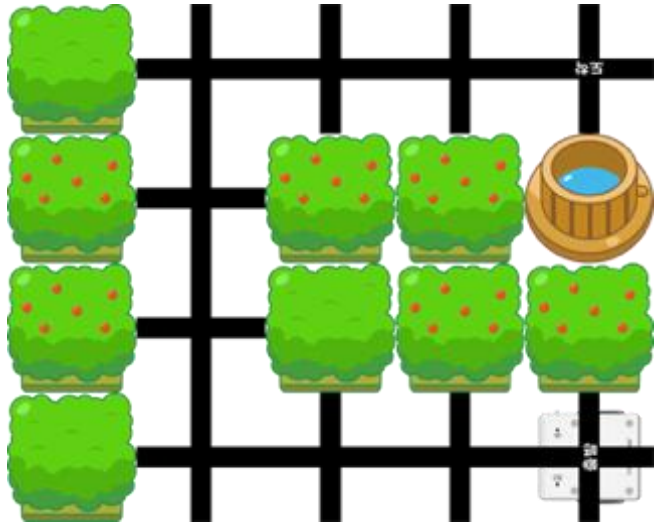


- 여섯 번째 우물도 찾아 봅시다.

```
from roboid import *

hamster = HamsterS()

for i in range(2):
    hamster.board_forward()
    hamster.board_right()
    hamster.board_forward()

for i in range(2):
    hamster.board_forward()
    hamster.board_left()
    hamster.board_forward()
```

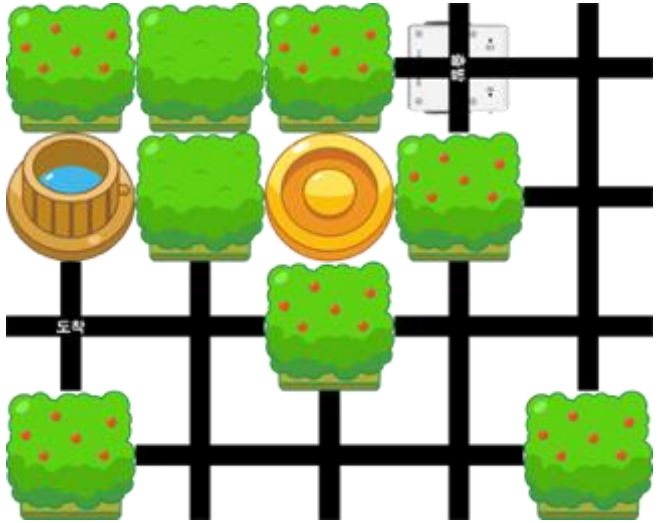
02

우물 찾기 (횟수 반복)



- 일곱 번째 우물도 찾아 봅시다.

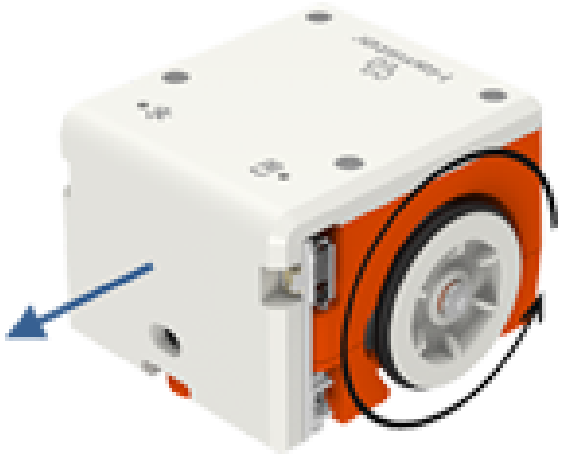
```
from roboid import *

hamster = HamsterS()

for i in range(3):
    for j in range(3):
        hamster.board_forward()
    hamster.board_right()
```



- 여덟 번째 우물도 찾아 봅시다.

```
from roboid import *

hamster = HamsterS()

for i in range(2):
    for j in range(2):
        hamster.board_forward()
        hamster.board_right()
        hamster.board_forward()
    hamster.board_left()
hamster.board_forward()
hamster.board_right()
```

03 앞 뒤 움직이기



- 바퀴의 속도는 -100부터 100까지의 값을 가지는데 최대 속도에 대한 % 값입니다. 양수 값이면 바퀴가 앞으로, 음수 값이면 뒤로 회전하고, 0이면 멈춥니다. 햄스터 로봇이 앞으로 이동하게 하려면 왼쪽 바퀴와 오른쪽 바퀴의 속도를 같은 양수 값으로 하면 됩니다.

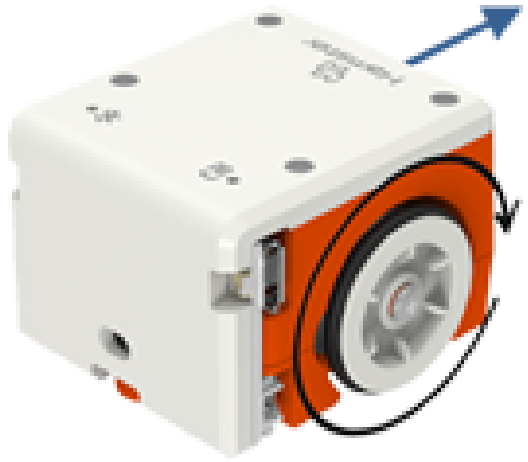
```
from roboid import *

hamster = HamsterS()

while True:
    hamster.wheels(30, 30)  # 왼쪽 바퀴와 오른쪽 바퀴의 속도를 30으로 설정한다.
```

- 값을 하나만 입력하면 양쪽 바퀴의 속도를 같은 값으로 설정합니다.

```
hamster.wheels(30)  # 양쪽 바퀴의 속도를 30으로 설정한다.
```

- 바퀴의 속도를 0부터 100까지 변경하면서 로봇이 이동하는 속도가 어떻게 달라지는지 관찰해 봅시다.

03 앞 뒤 움직이기



- 햄스터 로봇이 뒤로 이동하게 하려면 왼쪽 바퀴와 오른쪽 바퀴의 속도를 같은 음수 값으로 하면 됩니다.

```
from roboid import *

hamster = HamsterS()

while True:
    hamster.wheels(-30, -30)  # 왼쪽 바퀴와 오른쪽 바퀴의 속도를 -30으로 설정한다.
```

- 값을 하나만 입력하면 양쪽 바퀴의 속도를 같은 값으로 설정합니다.

```
hamster.wheels(-30)  # 양쪽 바퀴의 속도를 -30으로 설정한다.
```

- 바퀴의 속도를 0부터 -100까지 변경하면서 로봇이 이동하는 속도가 어떻게 달라지는지 관찰해 봅시다.

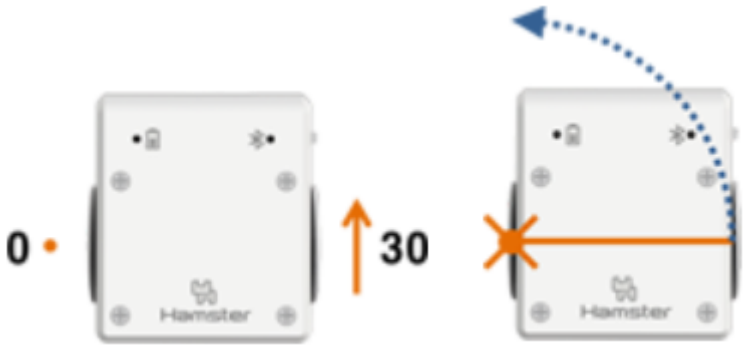
04 제자리 돌기

- 왼쪽 바퀴와 오른쪽 바퀴의 속도를 크기는 같게 하고 부호만 반대로 하면 제자리에서 회전합니다.
  로봇이 회전하는 중심점은 왼쪽 바퀴와 오른쪽 바퀴의 축을 연결하는 직선의 중심입니다.

```
from roboid import *

hamster = HamsterS()

while True:
    hamster.wheels(-30, 30) # 왼쪽 바퀴의 속도를 -30, 오른쪽 바퀴의 속도를 30으로 설정한다.
```

- 바퀴의 속도를 변경하면서 로봇이 회전하는 속도가 어떻게 달라지는지 관찰해 봅시다.

05 한쪽 바퀴를 축으로 회전하기 (피봇 턴)



- 왼쪽 바퀴를 정지하고 오른쪽 바퀴에만 속도 값을 주면 왼쪽 바퀴를 중심으로 회전합니다.

```
from roboid import *

hamster = HamsterS()

while True:
    hamster.wheels(0, 30)  # 왼쪽 바퀴의 속도를 0, 오른쪽 바퀴의 속도를 30으로 설정한다.
```

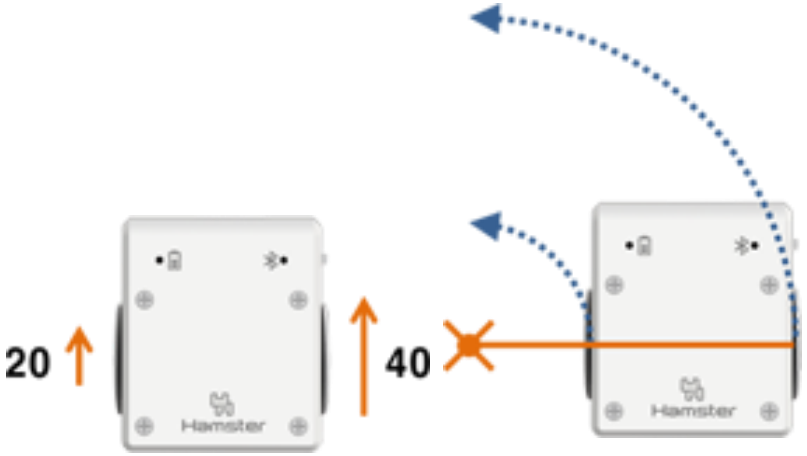
- 오른쪽 바퀴를 중심으로 회전하게 해보고, 시계 방향, 반시계 방향 등 다양한 회전을 만들어 봅시다.
- 왼쪽 바퀴 또는 오른쪽 바퀴, 하나의 값만 설정하기 위해서는 left_wheel() 또는 right_wheel() 메소드를 사용하여도 됩니다.

```
hamster.left_wheel(30)  # 왼쪽 바퀴의 속도를 30으로 설정한다.
```

```
hamster.right_wheel(30)  # 오른쪽 바퀴의 속도를 30으로 설정한다.
```

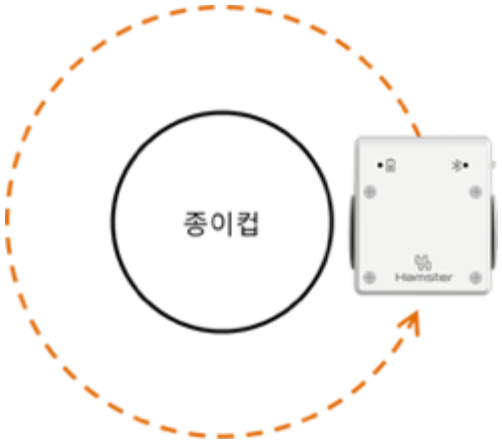
06

둥글게 회전하기 (라운드 턴)



- 왼쪽 바퀴와 오른쪽 바퀴의 속도 크기를 다르게 하면 원을 그리면서 회전합니다. 두 바퀴 간의 속도 차이가 작을수록 큰 원을 그리면서 회전합니다.

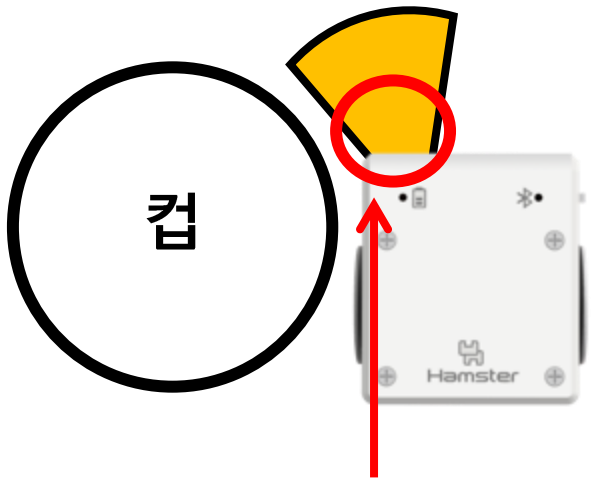
```
from roboid import *

hamster = HamsterS()

while True:
    hamster.wheels(20, 40)  # 왼쪽 바퀴의 속도를 20, 오른쪽 바퀴의 속도를 40으로 설정한다.
```



- 왼쪽 바퀴와 오른쪽 바퀴의 속도를 변경하면서 로봇이 회전하는 원의 크기가 어떻게 달라지는지 관찰해 봅시다.
- 종이컵의 바깥을 돌 수 있도록 왼쪽 바퀴와 오른쪽 바퀴의 속도를 조정해 봅시다.

07

종이컵 두고 돌기 (Beginner)



왼쪽 근접 센서

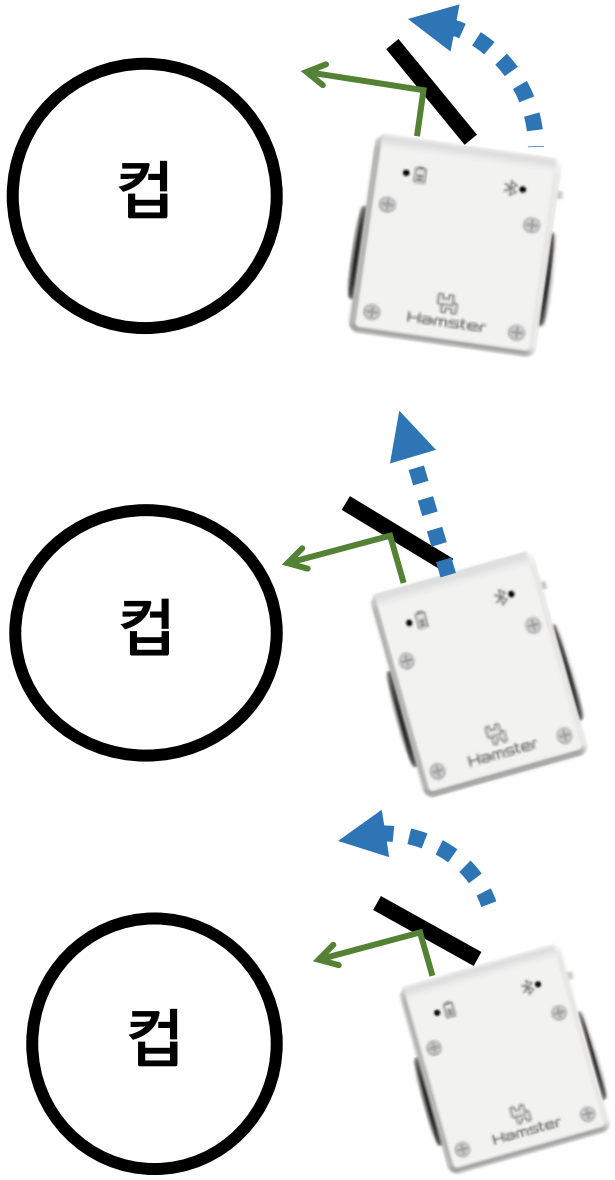
- 왼쪽 근접 센서의 값에 따라 양쪽 바퀴의 속도가 바뀌며 회전합니다. 왼쪽 센서 값이 10보다 커지면 왼쪽 바퀴가 오른쪽 바퀴보다 빨리 돌아 컵에 부딪히지 않게 주행하고, 왼쪽 센서 값이 10보다 작아지면 오른쪽 바퀴가 왼쪽보다 빨리 돌아 컵 쪽으로 주행합니다.

```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.left_proximity() >10: # 왼쪽 근접 센서의 값이 10보다 커지면
        hamster.wheels(30,0) # 왼쪽 바퀴의 속도를 30, 오른쪽 바퀴의 속도를 0으로 설정한다.
    else: # 왼쪽 근접 센서의 값이 10보다 작으면
        hamster.wheels(0,30) # 왼쪽 바퀴의 속도를 0, 오른쪽 바퀴의 속도를 30으로 설정한다.
    wait(10)
```

- 왼쪽 바퀴와 오른쪽 바퀴의 속도를 변경하면서 로봇이 회전하는 원의 크기가 어떻게 달라지는지 관찰해 봅시다.
- 종이컵의 바깥을 돌 수 있도록 왼쪽 바퀴와 오른쪽 바퀴의 속도를 조정해 봅시다.

07

종이컵 두고 돌기 (Intermediate)

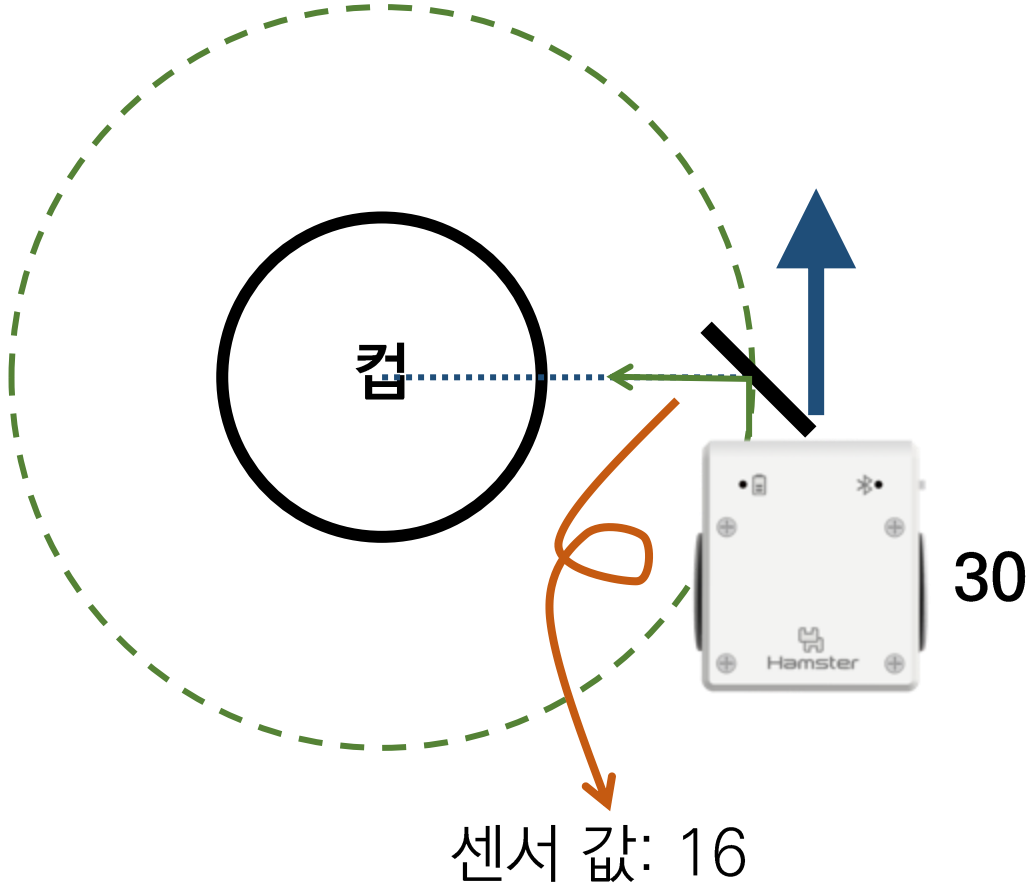


- 왼쪽 근접 센서의 값에 따라 양쪽 바퀴의 속도가 바뀌며 회전합니다. 왼쪽 센서 값이 10보다 커지면 양쪽 바퀴가 같은 속도로 컵에 부딪히지 않게 주행하고, 왼쪽 센서 값이 10보다 작아지면 오른쪽 바퀴가 왼쪽보다 빨리 돌아 컵 쪽으로 주행합니다.

```
from roboid import *

hamster = HamsterS()

while True:
  if hamster.left_proximity() >10: # 왼쪽 근접 센서의 값이 10보다 커지면
      hamster.wheels(30,30) # 왼쪽 바퀴의 속도를 30, 오른쪽 바퀴의 속도를 30으로 설정한다.
  else: # 왼쪽 근접 센서의 값이 10보다 작아지면
      hamster.wheels(0,30) # 왼쪽 바퀴의 속도를 0, 오른쪽 바퀴의 속도를 30으로 설정한다.
  wait(10)
```

07

종이컵 두고 돌기 (Advanced)



- 왼쪽 근접 센서의 값으로 양쪽 바퀴의 속도가 바뀌며 회전합니다. 오른쪽 바퀴는 항상 30의 속도이고 왼쪽 바퀴의 속도는 왼쪽 센서의 값으로 설정하여 주행합니다.

```
from roboid import *

hamster = HamsterS()

while True:
    hamster.wheels(hamster.left_proximity()*(30/16),30)
 # 왼쪽 바퀴의 속도는 왼쪽 센서의 값(16)과 기존 바퀴 속도(30)을 센서(16)로 나눈 값을 곱하여 설정한다.
   오른쪽 바퀴의 속도를 30으로 설정한다.
    wait(10)
```

3차시

LED켜고 소리내기

01 햄스터 버전에 따른 LED 설정 (필독)

- 햄스터 로봇의 LED는 아래 표에 나온 7가지 색상을 표현합니다.
- leds() 메소드를 사용하여 색을 지정합니다.
- 햄스터의 LED를 끄려면 Hamster.LED_OFF 또는 0을 입력합니다.

LED 색상	숫자	설명
Hamster.LED_OFF	0	LED를 끈다.
Hamster.LED_BLUE	1	LED를 파란색으로 켠다. (R: 0, G: 0, B: 255)
Hamster.LED_GREEN	2	LED를 초록색으로 켠다. (R: 0, G: 255, B: 0)
Hamster.LED_CYAN	3	LED를 하늘색으로 켠다. (R: 0, G: 255, B: 255)
Hamster.LED_RED	4	LED를 빨간색으로 켠다. (R: 255, G: 0, B: 0)
Hamster.LED_MAGENTA	5	LED를 보라색으로 켠다. (R: 255, G: 0, B: 255)
Hamster.LED_YELLOW	6	LED를 노란색으로 켠다. (R: 255, G: 255, B: 0)
Hamster.LED_WHITE	7	LED를 하얀색으로 켠다. (R: 255, G: 255, B: 255)

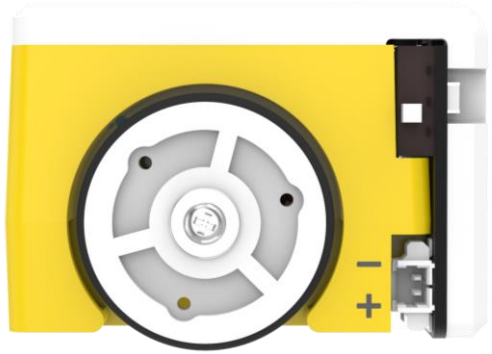
- 햄스터 예시

```
hamster= Hamster() #기본 햄스터 버전 이용

hamster.leds(hamster.LED_BLUE)  # 양쪽 LED를 파란색으로 설정한다.
hamster.leds(1) # 양쪽 LED를 파란색으로 설정한다.
```

- 햄스터 S 예시
- 햄스터의 LED_BLUE 대신 COLOR_NAME_BLUE를 사용합니다.
- 햄스터S는 풀컬러 LED를 사용하므로 R, G, B을 직접 입력하여 더욱 다양한 색을 표현할 수 있습니다.
- 햄스터S의 LED를 끄려면 HamsterS.COLR_NAME_OFF 또는 0을 입력합니다.

```
hamster= HamsterS() #햄스터 S 버전 이용

hamster.leds(hamster.COLOR_NAME_BLUE) # 양쪽 LED를 파란색으로 설정한다.
hamster.leds("blue")  # 양쪽 LED를 파란색으로 설정한다.
hamster.leds(0,0,255,0,0,255) # 양쪽 LED를 파란색으로 설정한다
hamster.leds("blue","off") # 왼쪽 LED만 파란색으로 켜고, 오른쪽은 끈다.
hamster.leds("blue","green") #왼쪽 LED 파란색, 오른쪽 LED 초록색으로 설정
```

01 LED켜고 앞으로 이동하기



- 햄스터 로봇이 양쪽 LED를 파란색으로 켜고 앞으로 이동하게 해봅시다.

```
from roboid import *

hamster = Hamster()

hamster.leds(hamster.LED_BLUE, hamster.LED_BLUE) # 양쪽 LED를 파란색으로 설정한다.
hamster.wheels(30, 30) # 왼쪽 바퀴와 오른쪽 바퀴의 속도를 30으로 설정한다.
wait(100)
```

- 햄스터S 로봇이 양쪽 LED를 파란색으로 켜고 앞으로 이동하게 해봅시다.

```
Hamster = HamsterS()

hamster.leds(hamster.COLOR_NAME_BLUE, hamster.COLOR_NAME_BLUE) # 양쪽 LED를 파란색으로 설정한다.
hamster.wheels(30, 30) # 왼쪽 바퀴와 오른쪽 바퀴의 속도를 30으로 설정한다.
wait(100)
```

- 색상을 하나만 입력하면 양쪽 LED를 같은 색상으로 설정합니다.

```
hamster.leds(hamster.LED_BLUE) # 양쪽 LED를 파란색으로 설정한다.
wait(100)
```

- 양쪽 LED를 다양한 색상으로 켜고 앞으로 이동하게 해봅시다.

02 LED 켜고 제자리 돌기

- 회전하는 방향으로 한쪽 LED만 초록색으로 켜고 제자리에서 돌게 해봅시다.

```
from roboid import *

hamster = Hamster()

hamster.leds(hamster.LED_GREEN, hamster.LED_OFF) # 왼쪽 LED를 초록색으로 켜고 오른쪽 LED를 끈다.

hamster.wheels(-30, 30) # 제자리에서 왼쪽으로 돈다.
wait(100)
```

- 햄스터는 색상에 해당하는 숫자를 입력해도 됩니다.

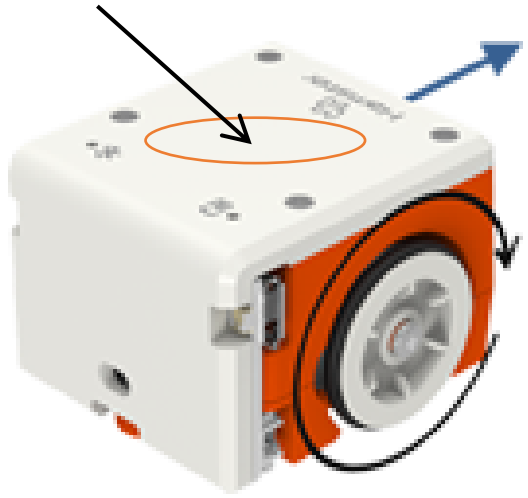
```
hamster.leds(3, 0) # 왼쪽 LED를 하늘색으로 설정한다.
```

- 왼쪽 LED또는 오른쪽 LED, 하나의 값만 설정하기 위해서는 left_led() 또는 right_led()메소드를 사용하여도 됩니다.

```
hamster.left_led(hamster.LED_CYAN) # 왼쪽 LED를 하늘색으로 설정한다.
hamster.leds(-30, 30) # 제자리에서 왼쪽으로 돈다.
wait(100)
```

03 소리 내며 뒤로 이동하기

피에조 스피커



- 햄스터 로봇이 소리를 내면서 뒤로 이동하게 해봅시다.
  버저 소리의 음 높이를 주파수[Hz]로 입력하면 됩니다.

```
from roboid import *

hamster = HamsterS()

while True:
    hamster.buzzer(1000)  # 버저 소리의 음 높이를 1000 Hz로 설정한다.
    hamster.wheels(-30)  # 뒤로 이동한다.
    wait(1000)
```

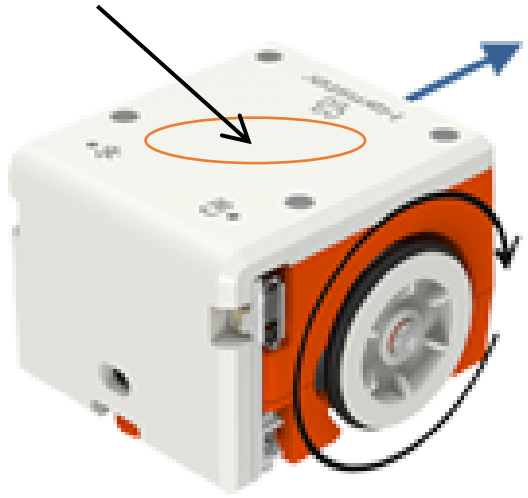
- 버저 소리의 음 높이는 소수점 둘째 자리까지 입력할 수 있으며, 버저 소리를 끄기 위해서는 0을 입력하면 됩니다.

```
hamster.buzzer(261.63)  # 버저 소리의 음 높이를 261.63 Hz로 설정한다.
hamster.wheels(-30)  # 뒤로 이동한다.
```

- 다양한 음 높이로 소리를 내면서 뒤로 이동하게 해봅시다.

04 청력 테스트

피에조 스피커



- 햄스터 로봇의 버저 소리는 0 ~ 167772.15 Hz까지 입력할 수 있는데, 나이에 따라 사람이 들을 수 있는 최고 음의 높이가 다릅니다. 버저 소리의 음 높이를 다양하게 변경하면서 자신이 얼마나 높은 주파수의 음까지 들을 수 있는지 측정해 봅시다.

```
from roboid import *

hamster = HamsterS()

hz = 0

while True:
    for _ in range(10):
        hamster.buzzer(hz)
        hz += 500 # 버저 소리의 음 높이를 500씩 높여준다.
        wait(200)
    hz = 0
```

- 1000 ~ 5000 사이의 값을 넣어보며 확인해 봅시다.

- 숫자를 매번 직접 넣지 않고 위 코드처럼 반복문을 통해 확인할 수도 있습니다.

05 정확한 음정 내기

- buzzer() 메소드를 사용한 경우에는 연속적인 주파수의 음을 낼 수 있지만 소수점 둘째 자리까지밖에 입력할 수 없다는 한계가 있습니다. 좀 더 정확한 음을 내보도록 합시다. 오차 0.01% 이하의 정확한 음정을 낼 수 있습니다. 소리를 끄기 위해서는 Hamster.NOTE_OFF 또는 0을 입력하면 됩니다.

```
from roboid import *

hamster = HamsterS()

hamster.note(hamster.NOTE_C_4) # 4옥타브 도 음을 소리낸다.
```

- 4옥타브 도 음에 해당하는 숫자를 입력해도 됩니다.

```
hamster.note(40)  # 4옥타브 도 음을 소리낸다..
```

- 무엇이 똑같을까 노래를 만들어 봅시다.

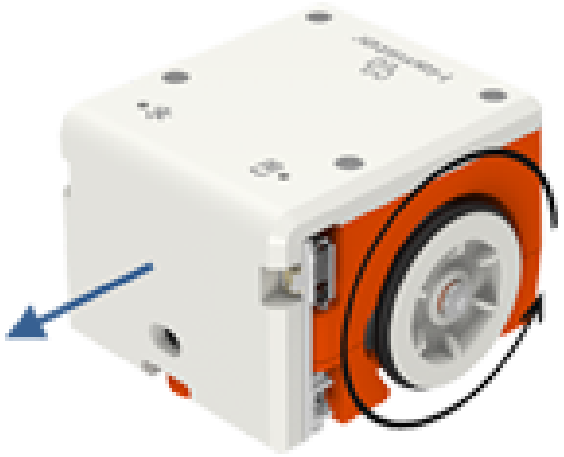
```
def play_music():
    for _ in range(2):
        hamster.note("C4",0.5)
        hamster.note("E4",0.5)
        hamster.note("G4",0.5)
    for _ in range(3):
        hamster.note("A4",0.5)
    hamster.note("G4",1)
    hamster.note(0,0.5)

play_music()
```

4차시

순서대로 반복하여 명령하기

01

일정 시간 동안 움직이기



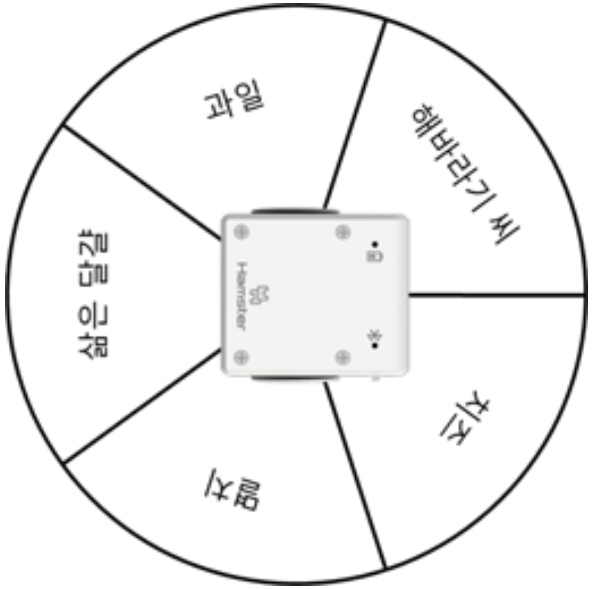
- 햄스터 로봇이 1초 동안 앞으로 이동한 후 정지하게 해봅시다. wait() 함수를 사용하면 코드의 다음 줄로 넘어가지 않고 일정 시간(msec) 동안 기다릴 수 있습니다.

```
from roboid import *

hamster = HamsterS()

hamster.wheels(30)  # 앞으로 이동한다.
wait(1000)  # 1초 기다린다.
hamster.stop()
```

- 1초 동안 후진하거나 회전하기 등 다양한 동작을 만들어 봅시다.

02

햄스터 룰렛



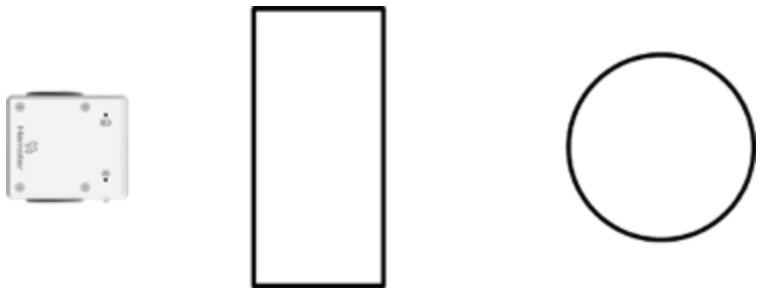
- 햄스터 로봇이 무작위 시간 동안 제자리에서 돌게 하여 룰렛 게임을 만들어 봅시다.
- 파이썬의 random 모듈에 있는 randint(low, high) 함수는 low부터 high까지의 무작위 정수를 반환합니다. random 모듈을 사용하기 위해서는 import random이 필요합니다.
  1초부터 10초까지의 무작위 시간 동안 제자리에서 돌게 해봅시다.

```
import random
from roboid import *

hamster = HamsterS()

hamster.wheels(-30, 30) # 제자리에서 왼쪽으로 돈다.
wait(random.randint(1000, 10000)) # 1초부터 10초까지의 무작위 시간 동안 기다린다.
hamster.stop()
```

- 햄스터 로봇이 제자리에서 도는 속도를 10부터 100까지의 무작위 속도로 해봅시다.

- 룰렛 게임판 내려 받기(PDF) :
  http://hamster.school/repository/tutorial/class/roulette_pdf_ko.zip

03 순서대로 명령하기 (순차)



- 햄스터 로봇이 1초 동안 앞으로 이동한 후 1초 동안 뒤로 이동하게 해봅시다.

```
from roboid import *

hamster = HamsterS()

hamster.wheels(30)  # 1초 동안 앞으로 이동한다.
wait(1000)

hamster.wheels(-30)
wait(1000)

hamster.stop()
```

- 햄스터 로봇이 1초 동안 앞으로 이동한 후 1초 동안 제자리에서 왼쪽 또는 오른쪽으로 돌게 해봅시다.

- 왼쪽 그림과 같이 A4 용지 위에 사각형과 원을 그립니다. 햄스터 로봇이 장애물(사각형)을 피해 오른쪽 원 안으로 들어가야 합니다.
  왼쪽 바퀴와 오른쪽 바퀴의 속도 및 시간을 설정하고, 여러 명령을 순서대로 수행하여 목표 지점으로 이동할 수 있도록 해봅시다.

- 목표 지점으로 갔다가 출발 지점으로 다시 되돌아 오도록 해봅시다.

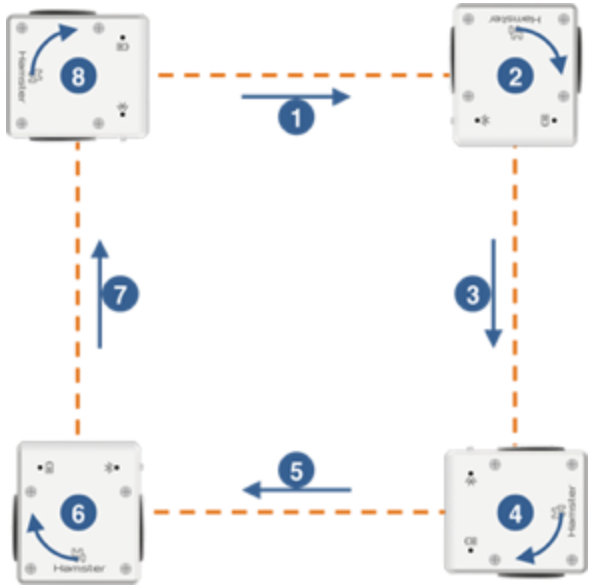
04 노래 연주하기

- 간단한 노래를 연주해 봅시다.

```
from roboid import *

hamster = HamsterS()

hamster.tempo(60) # 연주의 빠르기를 60 BPM(분당 박자 수)으로 한다. (초기 값: 60)
hamster.note(hamster.NOTE_C_4, 0.5) # 4옥타브 도 음을 0.5 박자 소리낸다.
hamster.note(hamster.NOTE_E_4, 0.5) # 4옥타브 미 음을 0.5 박자 소리낸다.
hamster.note(hamster.NOTE_G_4, 0.5) # 4옥타브 솔 음을 0.5 박자 소리낸다.
hamster.note(hamster.NOTE_C_4, 0.5) # 4옥타브 도 음을 0.5 박자 소리낸다.
hamster.note(hamster.NOTE_E_4, 0.5) # 4옥타브 미 음을 0.5 박자 소리낸다.
hamster.note(hamster.NOTE_G_4, 0.5) # 4옥타브 솔 음을 0.5 박자 소리낸다.
hamster.note(hamster.NOTE_A_4, 0.5) # 4옥타브 라 음을 0.5 박자 소리낸다.
hamster.note(hamster.NOTE_A_4, 0.5) # 4옥타브 라 음을 0.5 박자 소리낸다.
hamster.note(hamster.NOTE_A_4, 0.5) # 4옥타브 라 음을 0.5 박자 소리낸다.
hamster.note(hamster.NOTE_G_4, 1) # 4옥타브 솔 음을 1 박자 소리낸다.
```

- 음악 시간에 배운 다른 노래도 연주해 봅시다.

05 사각형 모양으로 이동하기

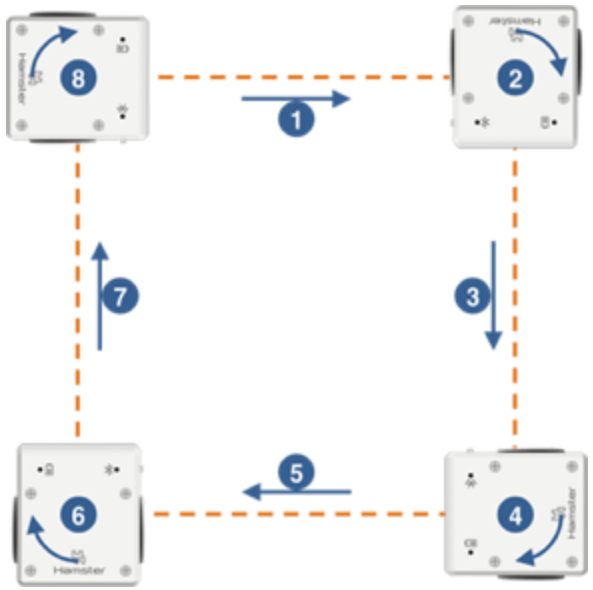


- 햄스터 로봇이 사각형 모양으로 이동하게 해봅시다. 사각형 모양으로 이동하기 위해서는 90도로 회전해야 하는데, 실험을 통해 몇 초 동안 회전하면 90도가 되는지 알아보도록 합시다. 950이라는 숫자는 실험 환경에 따라 달라질 수 있습니다.

```
from roboid import *

hamster = HamsterS()

hamster.wheels(30)
wait(1000)

hamster.wheels(30, -30)
wait(950)

hamster.wheels(30)
wait(1000)

hamster.wheels(30, -30)
wait(950)

hamster.wheels(30)
wait(1000)

hamster.wheels(30, -30)
wait(950)

hamster.wheels(30)
wait(1000)

hamster.wheels(30, -30)
wait(950)

hamster.stop()
wait(100)
```

05

사각형 모양으로 이동하기



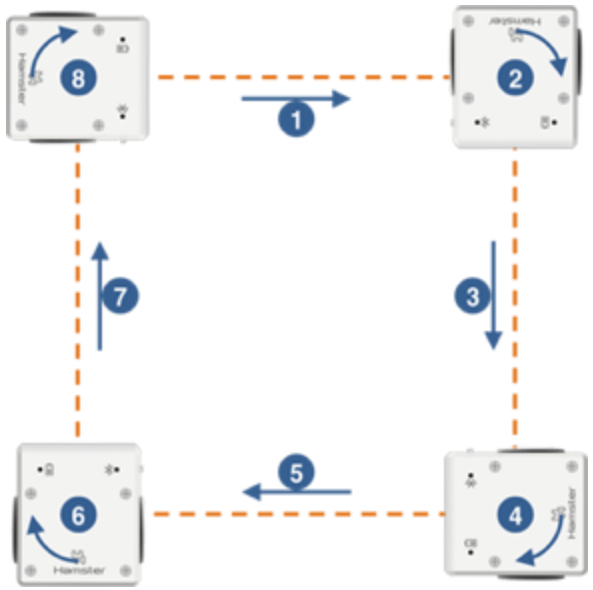
- 코드를 작성한 후 실행해 보면 햄스터 로봇이 정확하게 사각형 모양으로 이동할 수도 있고, 약간 찌그러진 사각형 모양으로 이동할 수도 있습니다.
- 바닥의 재질에 따라서는 완전히 이상한 모양으로 이동할 수도 있습니다.

  사람이 눈을 감고 사각형 모양으로 걸어가는 것과 같습니다. 센서를 사용하지 않기 때문입니다. 눈을 감고 사각형 모양으로 걸어가 보면 얼마나 어려운 일인지 알 수 있을 것입니다.
- 로봇에게 센서는 사람의 눈과 같습니다. 따라서 센서를 사용하지 않고 이동하는 것에는 한계가 있을 수밖에 없습니다. 햄스터 로봇이 센서를 사용하여 이동하는 방법은 앞으로 하나씩 배우게 될 것입니다.

- 직사각형, 삼각형, 다이아몬드 등 다양한 모양으로 이동하게 해봅시다.

06

횟수 반복하기



- 앞서 작성한 코드처럼 같은 내용을 반복하여 작성하는 것은 비효율적입니다.
  반복문을 사용하여 간략하게 작성해 봅시다.

```
from roboid import *

hamster = HamsterS()

for i in range(4):
    hamster.wheels(30)
    wait(1000)

    hamster.wheels(30, -30)
    wait(950)

hamster.stop()
wait(100)
```

- 반복문을 사용하여 직사각형, 삼각형, 다이아몬드 등 다양한 모양으로 이동하게 해봅시다.

07 서서히 출발하기

- 로봇이 이동할 때 서서히 속도를 높이거나 내리는 것은 중요합니다. 지금까지는 왼쪽 바퀴와 오른쪽 바퀴의 속도가 변하지 않았지만 속도를 서서히 높이기 위해서는 속도 값을 계속 다르게 주어야 합니다. 속도를 0에서 50까지 1씩 증가시키도록 합시다.

```
from roboid import *

hamster = HamsterS()

for speed in range(51): # 속도를 1씩 증가시킨다.
  hamster.wheels(speed)
  wait(20) # 20msec마다 속도를 증가시킨다
```

- 속도를 50에서 시작하여 0까지 1씩 감소시켜 정지하게 해봅시다. 속도가 0이 되면 더 이상 감소시키지 않아야 합니다.

```
from roboid import *

hamster = HamsterS()

for speed in reversed(range(51)): # 속도를 1씩 감소시킨다.
  hamster.wheels(speed)
  wait(20) # 20msec마다 속도를 감소시킨다
```

08 계속 반복하기

- 햄스터 로봇이 1초 동안 앞으로 이동한 후 1초 동안 뒤로 이동하는 것을 계속 반복하게 해봅시다.
- 다음의 코드는 while문 내부의 명령을 계속 반복하기 때문에 코드를 계속 수행하고 있습니다.
- 이러한 경우에는 파이썬 쉘의 Shell 〉 Restart Shell 메뉴를 선택하거나 윈도우를 닫아서 실행을 종료할 수도 있지만, 키보드의 콘트롤(Ctrl) 키와 알파벳 C 키를 동시에 눌러 코드가 수행되는 것을 중지할 수도 있습니다

```
from roboid import *

hamster = HamsterS()

While True:
    hamster.wheels(30)  # 1초 동안 앞으로 이동한다.
    wait(1000)

    hamster.wheels(-30)  # 1초 동안 뒤로 이동한다.
    wait(1000)
```

09

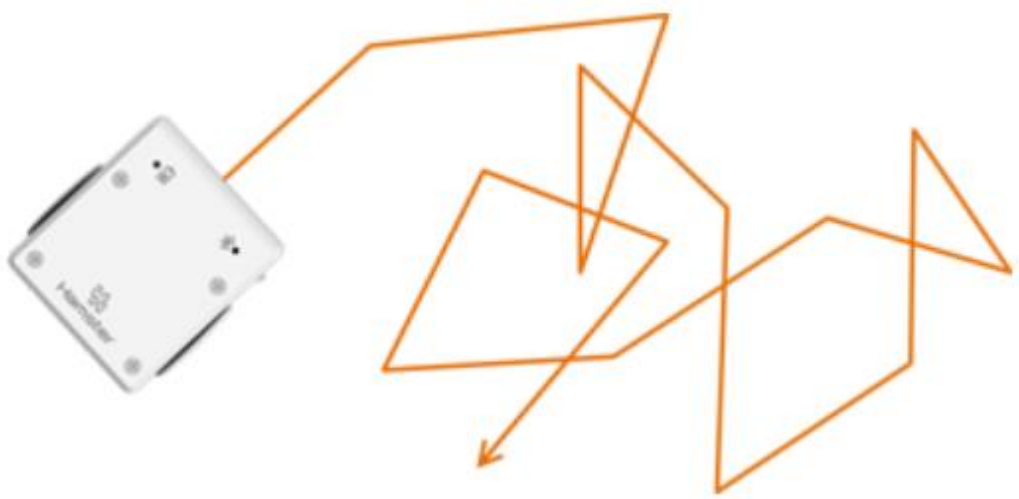
LED 계속 깜빡이기

- 자동차의 비상등처럼 햄스터 로봇의 양쪽 LED를 계속 깜박이게 해봅시다. 양쪽 LED를 0.2초 동안 노란색으로 켜고 0.2초 동안 끄는 것을 계속 반복하면 됩니다.

```
from roboid import *

hamster = Hamster()

While True:
 hamster.leds(hamster.LED_YELLOW)
 # 양쪽 LED를 노란색으로 켠다.
 Wait(200) # 0.2초 기다린다.

 hamster.leds(0)  # 양쪽 LED를 끈다.
 Wait(200) # 0.2초 기다린다.
```

- 햄스터 로봇의 버저 소리를 일정 시간 간격으로 삐삐삐~ 계속 반복하여 내게 해봅시다.

10 예술가 로봇



- 왼쪽 그림과 같이 무작위로 움직이는 것을 랜덤 워크(Random Walk)라 합니다.

```
from roboid import *
import random

hamster = HamsterS()
while True:
 # 양쪽 바퀴의 속도를 -50부터 50까지 무작위로 설정한다.
Hamster.wheels(random.randint(-50,50), random.randint(-50,50))

 # 0.5초부터 1초까지의 무작위 시간 동안 기다린다.
wait(random.randint(500,1000))
```

- 햄스터 로봇에 다양한 색상의 펜을 테이프 등으로 부착하고 A4 용지 위에 올려 놓은 후, 작성한 코드를 실행하면 멋진 추상화 그림을 그리는 예술가 로봇이 됩니다.

- 양쪽 바퀴 속도의 범위 또는 기다리는 시간의 범위를 변경하면서 로봇이 그림을 그리는 형태가 어떻게 달라지는지 관찰해 봅시다.

11 사이렌 소리 내기

- 햄스터 로봇의 버저를 사용하여 사이렌 소리를 만들어 봅시다. 버저 소리의 음 높이를 500부터 700까지 10씩 증가시키는 것을 계속 반복하면 됩니다.

```
from roboid import *

hamster = HamsterS()

while True:
  for hz in range(500,701,10): # 500부터 700까지 반복한다.
    hamster.buzzer(hz)
    wait(20)
```

- 사이렌 소리와 함께 양쪽 LED도 빨간색으로 계속 깜빡이게 해봅시다.

12 LED 깜빡이다 멈추기

- 자동차가 좌회전할 때처럼 햄스터 로봇이 왼쪽으로 회전하는 동안 LED를 깜빡이게 해봅시다.

```
from roboid import *

hamster = Hamster()

for i in range(5):                          # 5번 반복하면 2초가 된다.
  hamster.wheels(10,40)                     # 왼쪽으로 둥글게 회전한다.
  hamster.left_led(hamster.LED_YELLOW)      # 왼쪽 LED를 노란색으로 켠다.
  wait(200)                                 # 0.2초 기다린다.

  hamster.left_led(0)                       # 왼쪽 LED를 끈다.
  wait(200)                                 # 0.2초 기다린다.

hamster.stop()                              # 정지한다.
```

- 햄스터 로봇이 2초 뒤로 이동하는 동안 버저 소리를 일정 시간 간격으로 삐삐삐~ 계속 반복하여 내게 해봅시다.

5차시

근접센서 사용하기
말판 이동하기2

01

장애물 감지하기

- 햄스터 로봇의 전방에 있는 근접 센서는 적외선을 방출하는 IR-LED와 적외선을 감지하는 광 트랜지스터로 이루어져 있습니다.
- 광 트랜지스터는 IR-LED가 방출하는 적외선이 전방의 물체에 반사되어 들어오는 광량을 측정합니다.
- 장애물이 가까이 있으면 반사된 광량이 많아서 측정되는 값이 증가하고, 장애물이 멀면 반사된 광량이 적어서 측정되는 값이 감소합니다.
- 장애물이 없으면 반사된 빛이 없어 0의 값을 가집니다.
- 햄스터 로봇의 근접 센서는 IR-LED가 로봇의 앞면 좌우에 하나씩 설치되어 있으며, 중앙 아래에 광 트랜지스터가 있어서 좌우의 IR-LED에 대한 광량을 번갈아 측정합니다.
- 전방의 1cm 이상, 30cm 이하의 거리에 있는 물체나 장애물을 감지할 수 있으며, 0부터 255까지의 값을 가집니다.

01

장애물 감지하기

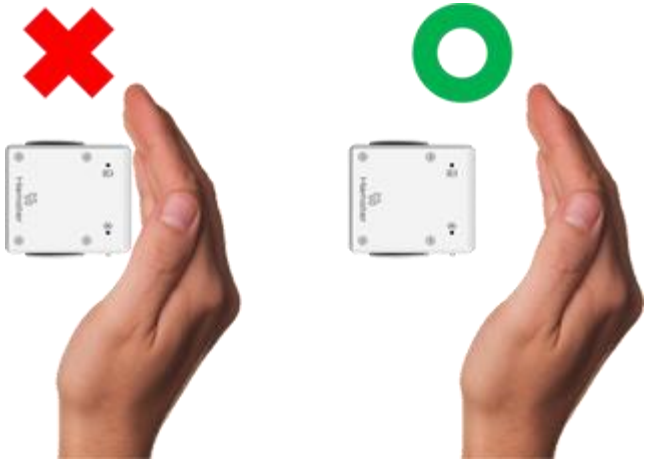
- 코드를 작성하기 전에 우선 장애물과의 거리에 따라 근접 센서의 값이 어떻게 달라지는지 관찰해 봅시다. 파이썬의 print 구문(2.7.x 버전) 또는 print() 함수(3.x.x 버전)를 사용하면 문자열을 출력할 수 있습니다.

```
from roboid import *

hamster = HamsterS()

while True:
    print(hamster.left_proximity(), hamster.right_proximity())  # Python 3
    # print hamster.left_proximity() hamster.right_proximity()  # Python 2.7.x
    wait(20)# 너무 빨리 반복하지 않도록 한다.
```

- while문 안에서 wait(20)과 같이 일정 시간 동안 기다리지 않으면 while문이 너무 과도하게 빨리 수행되기 때문에 다른 일, 즉 컴퓨터가 햄스터 로봇과 통신하는 일에 영향을 줄 수도 있습니다.
- 따라서 너무 빨리 무언가를 반복하는 경우에는 wait(20)과 같이 일정 시간 동안 기다려서 너무 빨리 반복되지 않도록 합시다.
- 컴퓨터와 햄스터 로봇이 약 20msec 주기로 통신하기 때문에 센서의 값도 약 20msec마다 한 번씩 전달됩니다. 이주기에 맞추어 20msec 동안 기다리도록 합시다.
  (보통의 경우 10 ~ 20msec 정도로 하면 됩니다.)

01

장애물 감지하기

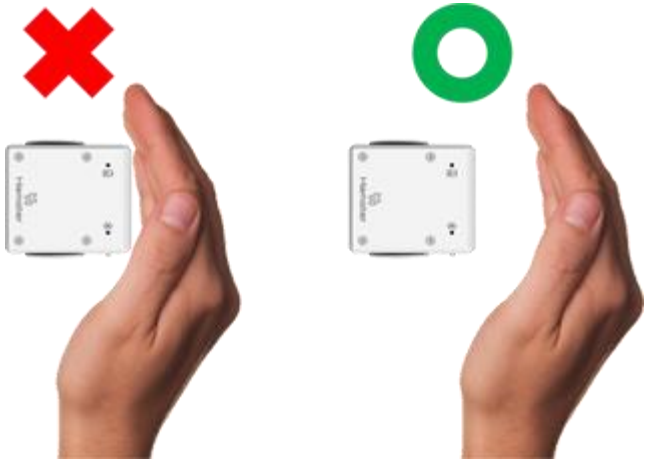


- 햄스터 로봇의 앞을 손으로 막고 손을 앞뒤로 움직여 로봇과의 거리를 다르게 하면서 왼쪽 근접 센서와 오른쪽 근접 센서의 값이 어떻게 달라지는지 관찰해 봅시다.
- 손을 너무 가까이 가져 가면 센서 값이 측정되지 않을 수도 있습니다.
- 하얀색 종이로 하였을 때, 다양한 색상의 물체로 하였을 때 어떤 변화가 생기는지도 관찰해 봅시다.
- 같은 거리에서 장애물의 색이 밝을수록 근접 센서의 값이 더 커진다는 것을 알 수 있는데, 밝은 색일수록 반사되는 광량이 많아지기 때문입니다.
- 센서의 값들을 관찰하고 분석하는 것은 매우 중요하기 때문에 앞으로 새로운 센서를 사용할 때마다 반드시 이와 같은 방법으로 관찰하도록 합시다.

01

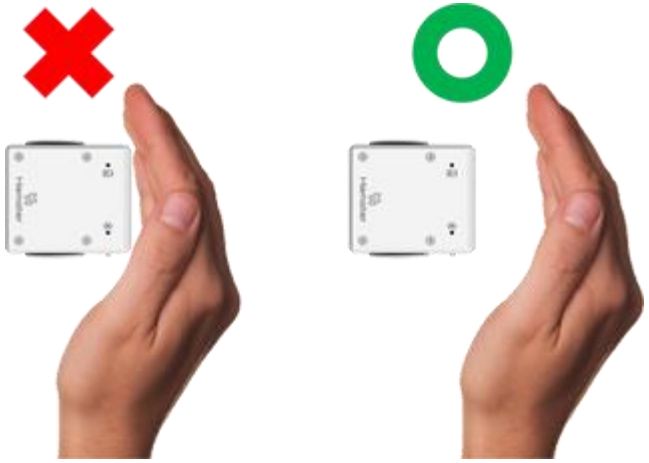
장애물 감지하기

- 햄스터 로봇이 앞으로 이동하다가 앞에 장애물이 나타나면 정지하게 해봅시다.
- 40이라는 숫자는 장애물과의 거리에 따라 센서 값이 어떻게 되는지 관찰하여 정해주면 됩니다.



```
from roboid import *

hamster = HamsterS()

while hamster.left_proximity() < 40 and hamster.right_proximity() < 40:
    hamster.wheels(30)  # 앞으로 이동한다.
    wait(20) # 너무 빨리 반복하지 않도록 한다.

hamster.stop() # 정지한다.
```

01 장애물 감지하기



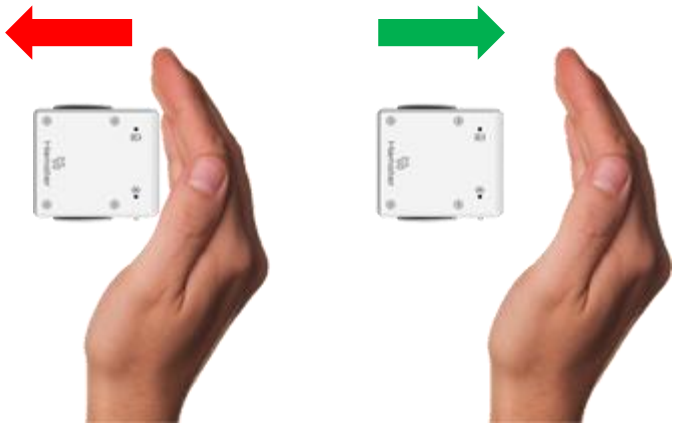
- 손을 치우면 햄스터 로봇이 다시 앞으로 이동하게 해봅시다.

```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.left_proximity() < 40 and hamster.right_proximity() < 40:
        hamster.wheels(30)  # 앞으로 이동한다.
    else:
        hamster.stop() # 정지한다.

    wait(20) # 너무 빨리 반복하지 않도록 한다.
```

- 햄스터 로봇이 제자리에서 왼쪽으로 돌다가 앞에 장애물이 있으면 정지하게 해봅시다.

02

밀당 로봇



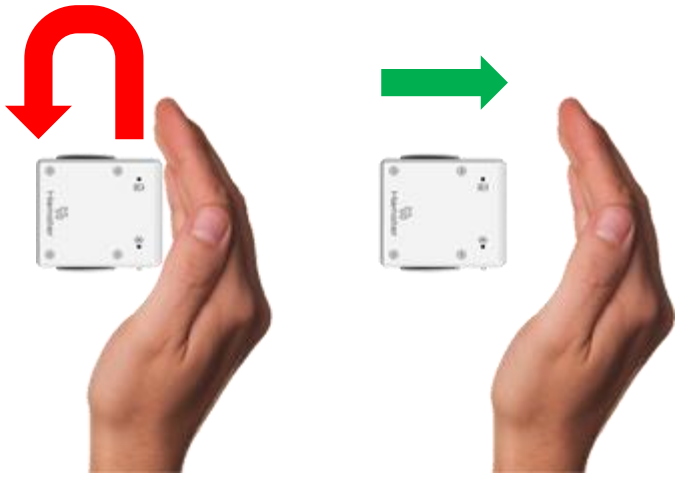
- 햄스터 로봇이 앞으로 이동하다가 앞쪽의 장애물과 가까우면 뒤로 물러나고 장애물과 멀어지면 다시 앞으로 이동하여, 장애물과 일정 거리를 두고 앞뒤로 이동하게 해봅시다.

```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.left_proximity() < 40 and hamster.right_proximity() < 40:
        hamster.wheels(30)  # 앞으로 이동한다.
    else:
        hamster.wheels(-30)  # 뒤로 이동한다.

    wait(20) # 너무 빨리 반복하지 않도록 한다.
```

- 장애물과 가까우면 근접 센서의 값이 40보다 커져서 뒤로 물러나고, 뒤로 물러나 장애물과 멀어지면 근접 센서의 값이 40보다 작아져서 다시 앞으로 이동하는 것을 반복합니다.

- 숫자 40을 변경하여 장애물과의 거리를 조정해 봅시다.

03 햄스터 탈출을 막아라



- 앞으로 이동하는 햄스터 로봇의 앞을 손으로 막으면 회전하여 방향을 바꾸고 다시 앞으로 이동하도록 해봅시다.

```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.left_proximity() < 40 and hamster.right_proximity() < 40:
        hamster.wheels(30)  # 앞으로 이동한다.
    else:
        hamster.wheels(-30, 30)  # 1초 동안 왼쪽으로 회전한다.
        wait(1000)

    wait(20) # 너무 빨리 반복하지 않도록 한다.
```

- 햄스터 로봇과 손 간의 거리에 따라 회전하는 시간이 달라지도록 해봅시다.

04

손으로 연주하기

- 햄스터 로봇과 손 간의 거리에 따라 음정을 변경해 봅시다.

```
from roboid import *

hamster = HamsterS()

pitch = 0
while True:
  proximity = hamster.left_proximity()
  if proximity < 10: # 거리가 너무 멀면 0으로 설정한다.

     proximity = 0

# 이전으 음정과  현재 근접 센서 값을 조합하여 음정이 부드럽게 변하도록 한다.
  pitch = (pitch * 9 + proximity)/10.0
  hamster.note(pitch)

  wait(20)  # 너무 빨리 반복하지 않도록 설정한다.
```

05

애완로봇

- 햄스터 로봇의 앞에 손을 멀리 가져가면 따라 오고 가깝게 가져가면 뒤로 도망가는 애완 로봇을 만들어 봅시다.

```
from roboid import *

hamster = HamsterS()

while True:
 proximity = hamster.left_proximity() # 왼쪽 근접 센서 값
 if proximity > 15:
  left_speed = (40-proximity)* 4 # 거리가 멀면 앞으로, 가까우면 뒤로 이동한다.
 else:
  left_speed = 0 # 거리가 너무 멀면 정지한다.

 proximity= hamster.right_proximity() # 오른쪽 근접 센서 값
 if proximity > 15:
  right_speed = (40-proximity)* 4 # 거리가 멀면 앞으로, 가까우면 뒤로 이동한다.
 else:
  right_speed = 0 # 거리가 너무 멀면 정지한다.

 hamster.wheels(left_speed, right_speed)

 wait(20) # 너무 빨리 반복하지 않도록 한다.
```

05 애완로봇

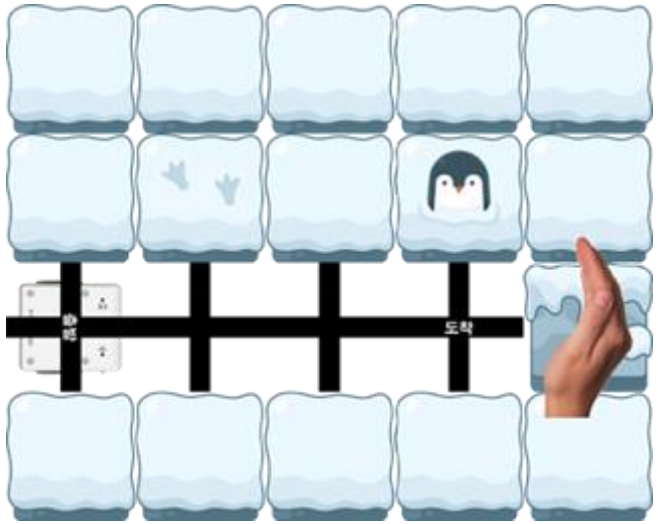
- 공통적인 부분을 함수로 만들면 다음과 같이 좀 더 간략하게 코드를 작성할 수 있습니다.

```
from roboid import *

hamster = HamsterS()

def calc_speed(proximity):
 if proximity > 15:
  return (40-proximity)* 4 # 거리가 멀면 앞으로, 가까우면 뒤로 이동한다.
 else:
  return 0 # 거리가 너무 멀면 정지한다.

while True:
 left_speed= calc_speed(hamster.left_proximity())
 right_speed= calc_speed(hamster.right_proximity())

 hamster.wheels(left_speed, right_speed)

 wait(20) # 너무 빨리 반복하지 않도록 한다.
```

- 햄스터 로봇과 손 간의 거리가 가까우면 근접 센서의 값이 40보다 커져서 바퀴의 속도가 음수가 되고, 거리가 멀면 근접 센서의 값이 40보다 작아져서 바퀴의 속도가 양수가 됩니다.
- 왼쪽 바퀴의 속도는 왼쪽 근접 센서 값에 의해 결정되고, 오른쪽 바퀴의 속도는 오른쪽 근접 센서 값에 의해 결정되기 때문에 손을 가져가는 방향에 따라 햄스터 로봇의 움직임이 달라집니다.
- 손바닥을 약간 둥근 모양으로 하면 햄스터 로봇이 더 잘 움직입니다.

06 얼음 나라

- 햄스터 로봇이 펭귄 친구를 만나러 얼음 나라에 갔습니다. 얼음 나라는 길이 얼음으로 되어 있어서 너무 미끄럽습니다. 길 위에서는 방향을 바꿀 수 없어요. 벽이 나타날 때까지 계속 앞으로 미끄러져 이동하다가 벽을 잡고 방향을 바꾸어야 합니다.
- 아래 그림에서 햄스터 로봇이 펭귄 앞까지 이동하려면 어떻게 움직여야 할지 생각해 봅시다. 도착 지점으로 이동한 후에는 펭귄 친구와 얘기할 수 있도록 햄스터 로봇이 펭귄 방향을 바라보아야 합니다.



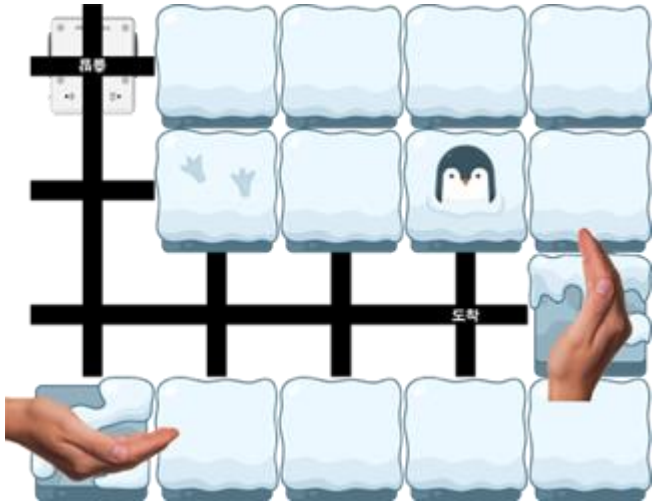
- 햄스터 로봇이 아직 너무 어려서 그림으로 그려진 벽은 알 수가 없으니 말판 위에 벽을 세워 놓아야 합니다. 그림의 손과 방향을 맞추어 손을 올려놓도록 합시다.

```
from roboid import *

hamster = HamsterS()

def can_move_forward():
    return hamster.left_proximity() < 40 and hamster.right_proximity() < 40

while(can_move_forward()):
    hamster.board_forward()
hamster.board_left()
```

- 1차시(38p)에서 내려받은 '말판 내려 받기(PDF)'의 도안(move_on_board.pdf, 13~16p)을 활용하세요.
- 근접 센서의 값은 상황에 따라 적절한 값으로 변경해도 됩니다.

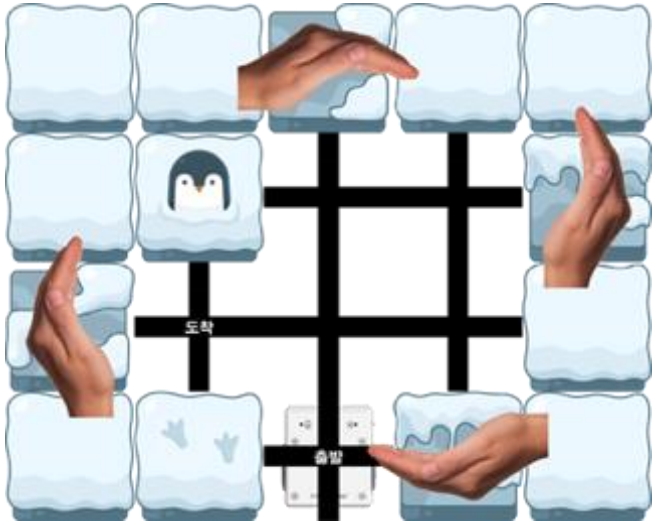
06

얼음 나라



- 두 번째 펭귄 친구도 찾아 봅시다.

```
from roboid import *

hamster = HamsterS()

def can_move_forward():
    return hamster.left_proximity() < 40 and hamster.right_proximity() < 40

for i in range(2):
    while(can_move_forward()):
        hamster.board_forward()
    hamster.board_left()
```



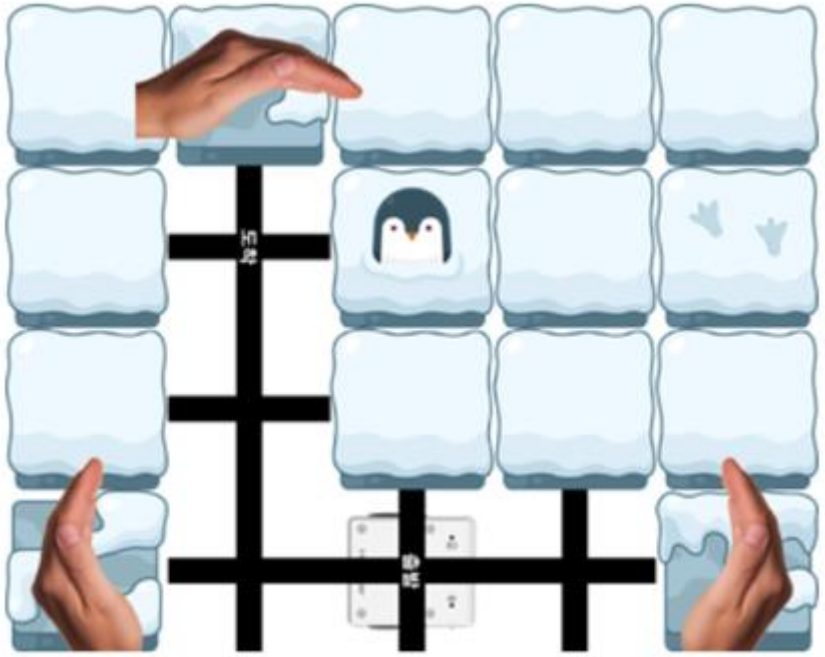
- 세 번째 펭귄 친구도 찾아 봅시다.

```
from roboid import *

hamster = HamsterS()

def can_move_forward():
    return hamster.left_proximity() < 40 and hamster.right_proximity() < 40

for i in range(4):
    while(can_move_forward()):
        hamster.board_forward()
    hamster.board_right()
```

06

얼음 나라

- 네 번째 펭귄 친구도 찾아 봅시다.



```
from roboid import *

hamster = HamsterS()

def can_move_forward():
    return hamster.left_proximity() < 40 and hamster.right_proximity() < 40


while(can_move_forward()):
    hamster.board_forward()
hamster.board_right()
hamster.board_right()


for i in range(2):
    while(can_move_forward()):
        hamster.board_forward()
    hamster.board_right()
```

6차시

바닥센서 사용하기

01

정지선 지키기

- 햄스터 로봇의 바닥 센서는 종이 위에 그려진 선이나 책상의 가장자리 등을 검출할 때 사용합니다.

- 바닥 센서는 근접 센서와 마찬가지로 적외선을 방출하는 IR-LED와 적외선을 감지하는 광 트랜지스터로 이루어져 있습니다.
  다른 점은 IR-LED와 광 트랜지스터가 한 쌍으로 이루어져 왼쪽 바닥 센서와 오른쪽 바닥 센서, 각각 별도로 구성되어 있다는 것입니다.

- IR-LED가 방출하는 적외선이 바닥에 반사되어 들어오는 광량을 광 트랜지스터가 측정합니다.
  바닥 센서가 밝은 색의 종이 또는 물체 위에 있으면 반사된 광량이 많아서 측정되는 값이 증가하고, 어두운 색 위에 있으면 반사된 광량이 적어서 측정되는 값이 감소합니다.

01

정지선 지키기

- 햄스터 로봇이 공중에 떠 있으면, 즉 바닥 센서 아래에 아무 것도 없으면 반사된 빛이 없어 0의 값을 가집니다.
  바닥 센서가 출력하는 값의 범위는 0 ~ 255이지만 바닥에서 가장 밝은 색(흰색에 가까운 색 중에서 가장 밝은 색)을 100으로 자동 보정하기 때문에 코드 작성 시 사용하는 값의 범위는 0 ~ 100입니다.

- A4 용지를 준비하고 검은색 테이프 또는 펜으로 정지선을 표시합시다. 미리 제작된 파일을 프린터로 인쇄해도 됩니다.
  이때 바닥 센서가 정지선을 놓치지 않고 감지할 수 있도록 정지선의 폭은 1cm 이상으로 합시다.

01

정지선 지키기

- 코드를 작성하기 전에 우선 바닥의 색상에 따라 바닥 센서의 값이 어떻게 달라지는지 관찰해 봅시다.
- 바닥 센서가 A4 용지의 하얀색 위에 있을 때와 검은색 위에 있을 때 센서 값이 어떻게 다른지 확인해 봅시다.
- 바닥 센서가 하얀색과 검은색의 경계 위에 있을 때, 즉 하얀색과 검은색이 바닥 센서에 반 쯤 걸쳐 있을 때 센서 값이 어떻게 되는지도 관찰해 봅시다.

```
from roboid import *

hamster = HamsterS()

while True:
 print(hamster.left_floor(), hamster.right_floor())  # Python 3.x.x
 #print hamster.left_floor(), hamster.right_floor # Python 2.7.x
 wait(20)  # 너무 빨리 반복하지 않도록 한다.
```

01

정지선 지키기

- 햄스터 로봇이 앞으로 이동하다가 정지선을 만나면 정지하게 해봅시다.

```
from roboid import *

hamster = HamsterS()

while hamster.left_floor()>20 and hamster.right_floor()>20: # 정지선을 만날 때까지
 hamster.wheels(30) # 앞으로 이동한다.
 wait(20) # 너무 빨리 반복하지 않도록 한다.

hamster.stop() # 정지한다.
```

- 숫자 20을 변경하여 햄스터 로봇이 정지하는 지점을 조정해 봅시다.

02 일정 간격의 검은색 선 개수 세기

햄스터 로봇이 공중에 떠 있으면 반사된 빛이 없어 바닥 센서의 값이 0이 되기 때문에 검은색 선 위에 있는 것과 같아 집니다.
검은색 선의 개수를 세는 동안
햄스터 로봇을 손으로 들지 않도록 합니다.

- A4 용지를 준비하고 검은색 테이프 또는 펜으로 검은색 선을 여러 개 표시합시다. 미리 제작된 파일을 프린터로 인쇄해도 됩니다. 이때 바닥 센서가 검은색 선을 놓치지 않고 감지할 수 있도록 검은색 선의 폭은 1cm 이상으로 합시다. 검은색 선 간의 거리는 햄스터 로봇이 1초 동안 앞으로 이동하는 거리보다 멀어야 합니다.
- 검은색 선을 감지하는 것은 앞에서 한 것과 같은 방법으로 하면 됩니다. 검은색 선을 지나가는 것은 여러 가지 방법이 있는데, 우선 쉽게 하기 위해 일정 시간 동안 앞으로 이동하는 방법으로 검은색 선을 지나가게 합시다.

```
from roboid import *

hamster = HamsterS()

hamster.wheels(30)  # 앞으로 이동한다.

count = 0
while True:
  if hamster.left_floor()< 20 or hamster.right_floor()<20:  # 검은색 선 위에 있으면
   #개수를 하나 증가시키고 화면에 출력한다.
    count+=1
    print('count:', count)  # python 3.x.x
    #print 'count:', count #python 2.7.x
    hamster.wheels(30)  # 1초 동안 앞으로 이동한다.
    wait(1000)
  wait(20)  # 너무 빨리 반복하지 않도록 한다.
```

- 검은색 선의 개수를 세다가 네 번째 검은색 선을 만나면 정지하도록 해봅시다.

03 임의 간격 및 굵기의 검은색 선 개수 세기

- 검은색 선 간의 거리를 짧게 하면 햄스터 로봇이 1초 동안 앞으로 이동할 때 검은색 선을 하나 이상 지나갈 수 있습니다. 검은색 선을 굵게 하면 하나의 선을 여러 번 셀 수도 있습니다. 앞에서 작성한 코드를 아래의 실습판에서 실행하여 선의 개수를 잘 세는지 관찰해 봅시다.
    - 하얀색과 검은색의 경계를 감지해 봅시다. 하얀색에서 검은색으로 바뀔 때 개수를 세면 됩니다.

```
from roboid import *

hamster = HamsterS()

hamster.wheels(30) # 앞으로 이동한다.
count = 0
white = False

while True:
    if hamster.left_floor()>80 and hamster.right_floor()>80:
        white = True # 하얀색 종이 위에 있다.
    elif white and (hamster.left_floor()< 20 or hamster.right_floor()<20):
                    # 하얀색에서 검은색으로 바뀌면
        white =False
        # 개수를 하나 증가시키고 화면에 출력한다.
        count +=1
        print('count:',count)

    wait(20) #너무 빨리 반복하지 않도록 한다.
```

04 바코드 읽기

- 검은색 선 위에 있는 동안 시간을 측정하여 선의 두께를 알아 봅시다.
- 굵은 선과 가는 선, 두 가지로 구분하여 바코드를 인식합니다.

```
from roboid import *

hamster = HamsterS()

hamster.wheels(30) # 앞으로 이동한다.

tick = 0
code = ''

while True:
    if hamster.left_floor() > 80 and hamster.right_floor() > 80: # 하얀색 종이 위에 있으면
        if tick > 0:
            if tick > 40: # 선이 굵다.
                code += '-'
            else: # 선이 가늘다.
                code += '.'

            print(code) # 바코드 확인.
        tick = 0

    elif hamster.left_floor() < 20 and hamster.right_floor() < 20 : # 검은색 선 위에 있으면
        tick += 1

    wait(20) # 너무 빨리 반복하지 않도록 한다.
```

- 바코드 읽기 실습판 내려 받기(PDF) :
  http://hamster.school/repository/tutorial/class/stop_line_4_barcode_pdf.zip

05 라인트레이싱

- 왼쪽 센서 + 오른쪽 가장자리
- 검은색 선의 오른쪽 가장자리에서 왼쪽 센서의 위치에 따라 선을 벗어나지 않도록 동작한다.

05 라인트레이싱

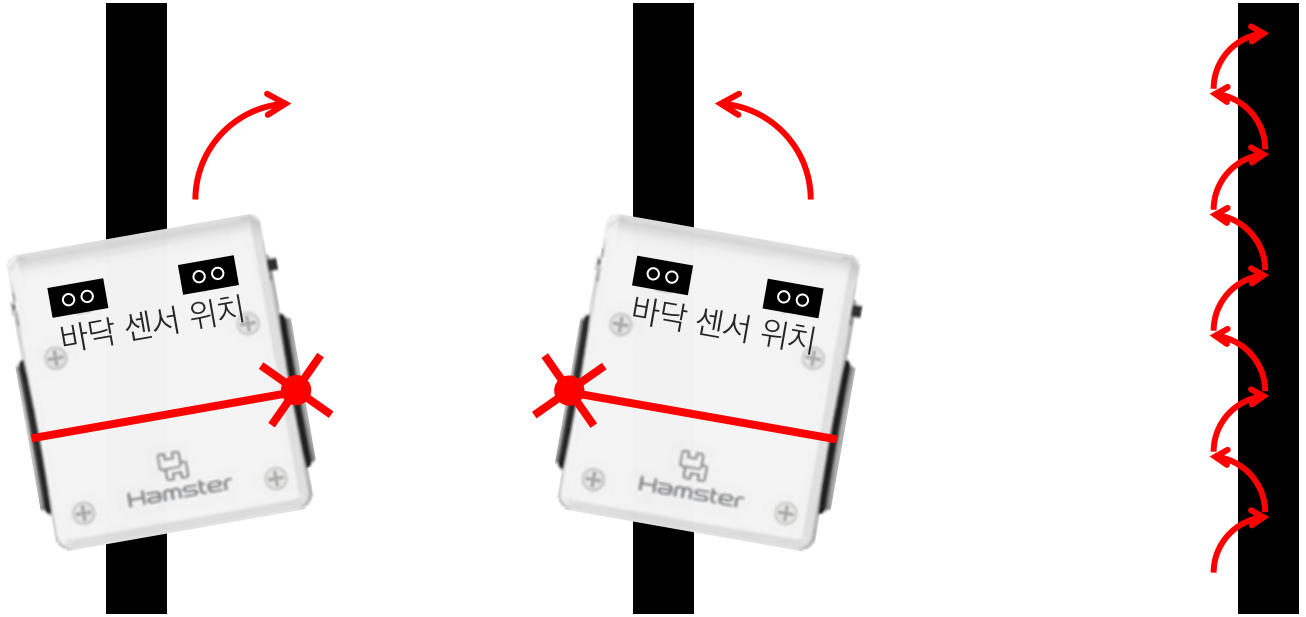
- 왼쪽 센서 + 오른쪽 가장자리
- 검은색 선의 오른쪽 가장자리에서 왼쪽 센서의 위치에 따라 선을 벗어나지 않도록 동작한다.


```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.left_floor() > 50:
        hamster.wheels(0, 30)
    else:
        hamster.wheels(30, 0)

    wait(10) # 너무 빨리 반복하지 않도록 한다.
```

05

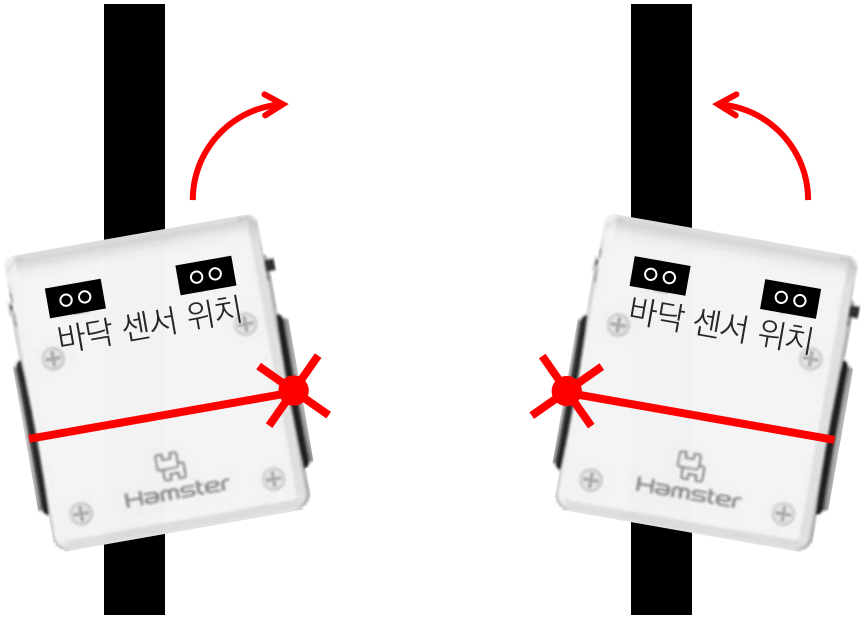
라인트레이싱

- 왼쪽 센서 + 왼쪽 가장자리
- 검은색 선의 왼쪽 가장자리에서 왼쪽 센서의 위치에 따라 선을 벗어나지 않도록 동작한다.



05 라인트레이싱

- 왼쪽 센서 + 왼쪽 가장자리
- 검은색 선의 왼쪽 가장자리에서 왼쪽 센서의 위치에 따라 선을 벗어나지 않도록 동작한다.




```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.left_floor() > 50:
        hamster.wheels(30, 0)
    else:
        hamster.wheels(0, 30)

    wait(10) # 너무 빨리 반복하지 않도록 한다.
```

05

라인트레이싱

- 양쪽 바닥 센서
- 검은색 선의 양쪽 가장자리에서 양쪽 센서의 위치에 따라 선을 벗어나지 않도록 동작한다.

05 라인트레이싱

- 양쪽 바닥 센서
- 검은색 선의 양쪽 가장자리에서 양쪽 센서의 위치에 따라 선을 벗어나지 않도록 동작한다.


```
from roboid import *

hamster = HamsterS()

while True:
    hamster.wheels(30, 30)
    if hamster.left_floor() < 50:
        hamster.wheels(-30, 30)
    elif hamster.right_floor() < 50:
        hamster.wheels(30, -30)

    wait(10) # 너무 빨리 반복하지 않도록 한다.
```

7차시

밝기센서와 가속도센서 사용하기

01

어둠이 무서워요

- 햄스터 로봇의 앞면에는 근접 센서의 광 트랜지스터와 같은 위치에 빛의 밝기를 감지하는 밝기 센서가 있습니다. 밝기 센서는 0부터 65535 Lux까지의 값을 가지며, 밝은 수록 값이 커집니다.

- 코드를 작성하기 전에 우선 밝기 센서의 값을 관찰해 봅시다.
  햄스터 로봇의 밝기 센서를 손으로 가리거나 모니터 화면 또는 형광등과 같이 밝은 곳을 향했을 때 센서 값이 어떻게 달라지는지 관찰해 봅시다.

```
from roboid import *

hamster = HamsterS()

while True:
    print(hamster.light())
    wait(20) # 너무 빨리 반복하지 않도록 한다.
```

01 어둠이 무서워요

- 햄스터 로봇을 옷이나 종이 박스로 덮으면 버저 소리를 내게 합시다.

```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.light() < 10:
        hamster.buzzer(1000)
    else:
        hamster.buzzer(0)

    wait(20) # 너무 빨리 반복하지 않도록 한다.
```

01 어둠이 무서워요

- 어두울 수록 높은 음을 내게 해봅시다.
  밝기 센서의 값이 100일 때 버저 음의 높이는 100, 밝기 센서의 값이 5일 때 버저 음위 높이는 2000이 되도록 하고, 그 사이를 일정 비율로 버저 음이 높아지도록 하면 (버저 음) = 2100 – (밝기) * 20이 됩니다.

```
from roboid import *

hamster = HamsterS()

while True:
    value = hamster.light()
    print(value)

    if value < 5:
        hamster.buzzer(2000)
    elif value > 100:
        hamster.buzzer(0)
    else:
        hamster.buzzer(2100 - value * 20)

    wait(20) # 너무 빨리 반복하지 않도록 한다.
```

02 빛으로 말해요 (2인 1조)

- 모스 부호를 만들어 햄스터 친구에게 전달해 봅시다.

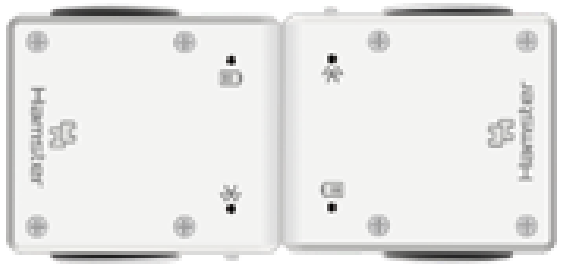
  첫 번째 햄스터 로봇은 하얀색 LED를 짧게 또는 길게 깜빡여서 모스 부호를 표현합니다.

```
from roboid import *

hamster = HamsterS()

def blink_short():
    hamster.leds(hamster.COLOR_NAME_WHITE)
    wait(200)
    hamster.leds(0)
    wait(200)

def blink_long():
    hamster.leds(hamster.COLOR_NAME_WHITE)
    wait(1000)
    hamster.leds(0)
    wait(200)
```

```
# 첫 번째 모스 부호 (.--.)
def send_morse1():
    blink_short()
    blink_long()
    blink_long()
    blink_short()

# 두 번째 모스 부호 (-..-)
def send_morse2():
    blink_long()
    blink_short()
    blink_short()
    blink_long()

send_morse1()
```

02 빛으로 말해요 (2인 1조)

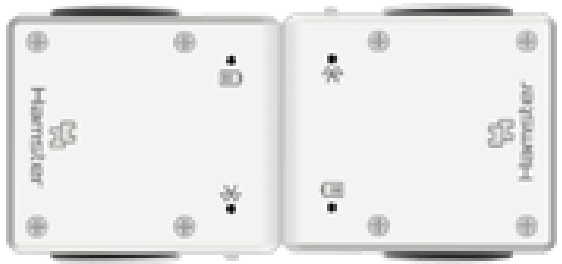
- 두 번째 햄스터 로봇은 밝기 센서를 사용하여 LED의 밝기를 감지하고 LED가 켜진 시간을 측정하여 모스 부호를 알아냅니다.
  첫 번째 모스 부호에 대해서는 잠시 뒤로 물러났다가 왼쪽으로 회전하고, 두번째 모스 부호에 대해서는 잠시 뒤로 물러났다가 오른쪽으로 회전하도록 합시다.



```
from roboid import *

hamster = HamsterS()

# 첫 번째 모스 부호에 대한 동작
def act1():
    hamster.wheels(-30)
    wait(1000)
    hamster.wheels(-30, 30)
    wait(1000)
    hamster.stop()

# 두 번째 모스 부호에 대한 동작
def act2():
    hamster.wheels(-30)
    wait(1000)
    hamster.wheels(30, -30)
    wait(1000)
    hamster.stop()
```

```
tick = 0
code = ''

while True:
    if hamster.light() < 10: # 어두우면
        if tick > 0:
            if tick > 40: # 밝은 시간이 길다.
                code += '-'
            else: # 밝은 시간이 짧다.
                code += '.'

            print(code) # 모스 부호 확인.
            if code == '.--.':
                code = ''
                act1()
            if code == '-..-':
                code = ''
                act2()

        tick = 0

    elif hamster.light() > 20: #밝으면
        tick += 1

    wait(20) # 너무 빨리 반복하지 않도록 한다.
```

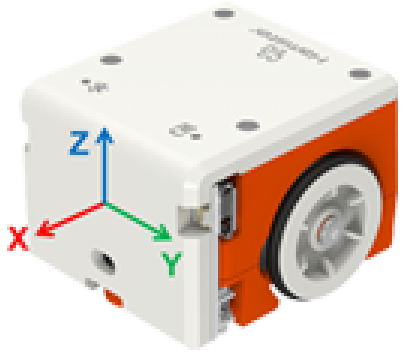
02 빛으로 말해요 (2인 1조)



- 두 대의 햄스터 로봇을 서로 마주 보게 하여 뽀뽀하듯 가까이 붙여 놓습니다.
- 두 번째 햄스터 로봇의 코드를 먼저 실행하여 모스 부호를 받을 준비가 되면, 첫 번째 햄스터 로봇의 코드를 실행하여 모스 부호를 전달하도록 합시다.
- 다양한 모스 부호를 만들어 친구에게 비밀스러운 명령을 전달해 봅시다.

03

엉덩이 치면 앞으로 가기

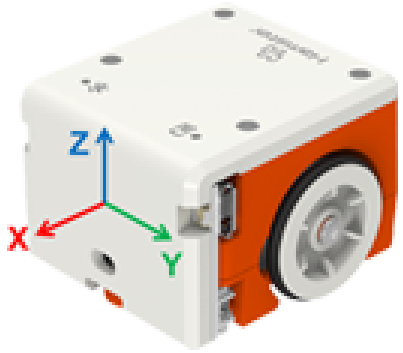


- 햄스터 로봇의 3축 가속도 센서 좌표계는 로봇의 앞쪽 방향이 X축, 왼쪽 방향이 Y축, 위쪽 방향이 Z축의 양수 방향이며,
  각각 -32768 ~ 32767의 값을 가집니다.

- 코드를 작성하기 전에 우선 가속도 센서의 값을 관찰해 봅시다.
  햄스터 로봇을 뒤집거나 옆으로 눕혔을 때 가속도 값이 어떻게 달라지는지 관찰해 봅시다.
  햄스터 로봇을 손에 쥐고 이리저리 손목을 움직이면서 가속도 값을 확인해 봅시다.

```
from roboid import *

hamster = HamsterS()

while True:
    print(hamster.acceleration_x(), hamster.acceleration_y(), hamster.acceleration_z())
    wait(20) # 너무 빨리 반복하지 않도록 한다.
```

03 엉덩이 치면 앞으로 가기



- 햄스터 로봇의 엉덩이를 손가락으로 툭 치면 햄스터 로봇의 X축 방향으로 가속도가 작용합니다. 이를 감지하여 1초 동안 앞으로 가게 해봅시다.

```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.acceleration_x() > 2000: # 엉덩이를 툭 쳤으면
        hamster.wheels(30) # 앞으로 이동한다.
        wait(1000)
        hamster.stop() # 1초 후에 정지한다.

    wait(20) # 너무 빨리 반복하지 않도록 한다.
```

- 어느 방향으로든 햄스터 로봇을 손가락으로 툭 치면 1초 동안 앞으로 가도록 코드를 수정해 봅시다.

04 긴급구조

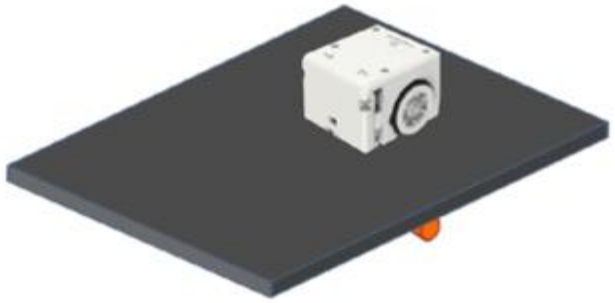
- 햄스터 로봇에 내장된 가속도 센서를 사용하여 햄스터 로봇이 넘어졌다는 것을 감지해 봅시다.

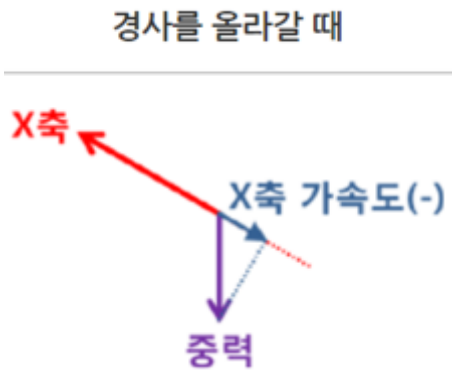
- 햄스터 로봇이 똑바로 서있으면 중력에 의해 아래쪽으로만 가속도가 생기므로 가속도의 Z축 성분만 음수로 크기가 큰 값을 가지고 X축과 Y축 성분의 크기는 작은 값을 가집니다. 햄스터 로봇이 앞뒤로 혹은 왼쪽, 오른쪽으로 넘어졌을 때는 Z축 성분의 크기가 작아지고 X축 또는 Y축 성분의 크기는 커진다는 것을 알 수 있습니다. 따라서 Z축 성분의 크기가 작거나 양수이면 햄스터 로봇이 넘어졌다고 판단할 수 있습니다.

```
from roboid import *

hamster = HamsterS()

tick = 0
while True:
  if hamster.acceleration_z()> -2048:   # z축 가속도 값이 작거나 양수이면
  # 일정 시간 간격으로 삐삐삐~ 소리 낸다.
    tick += 1
    if tick <=10: # 0.2초까지
       hamster.buzzer(1000)
    elif tick <=20: # 0.4초까지
       hamster.buzzer(0)
    else:
       tick = 0
 else:
    hamster.buzzer(0)
    tick = 0

 wait(20)  #너무 빨리 반복하지 않도록 한다.
```

05

시소놀이

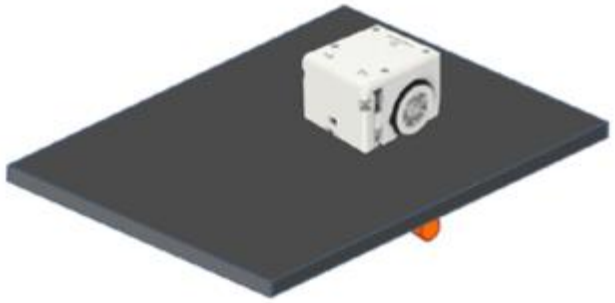


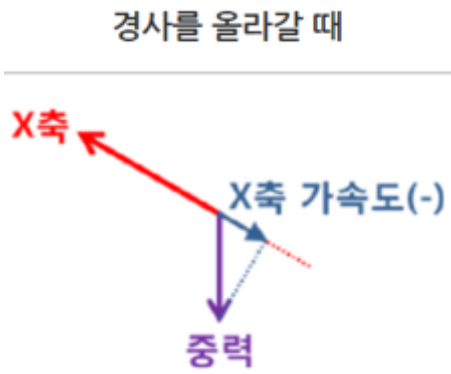


- 다음 그림과 같이 얇은 판을 연필이나 볼펜 위에 올려 시소처럼 움직이는 경사로를 만듭니다.
- 미로판의 검은색 바닥 판 한 장을 얇은 판으로 사용하여도 됩니다. 햄스터 로봇이 경사로를 올라갈 수 있도록 한 쪽 끝에 올려 놓습니다.
- 햄스터 로봇의 x축 가속도 값을 관찰해 보면 중력이 아래쪽으로 작용하기 때문에 수평으로 놓여진 경우에는 아주 작은 값을 가집니다.
- 경사로를 내려갈 때는 x축 가속도 값이 양수 값으로 절대치가 커지고, 경사로를 올라갈 때는 음수 값으로 절대치가 커집니다.

05

시소놀이





- 왼쪽 그림과 같이 관찰 결과를 바탕으로 시소 놀이를 만들어 봅시다. 햄스터 로봇이 경사로를 올라가서 중앙을 넘어가면 햄스터 로봇의 무게 때문에 얇은 판의 앞쪽이 시소처럼 아래로 기울어집니다.

- 이 때문에 햄스터 로봇의 x축 가속도 값이 바뀌게 되고 이를 감지하여 햄스터 로봇이 뒤로 이동하게 합니다.

- 뒤로 이동하면서 중앙을 넘어가면 얇은 판의 뒤쪽이 아래로 기울어집니다. 이를 감지하여 다시 앞으로 이동하는 것을 반복하게 해봅시다.

```
from roboid import *

hamster = HamsterS()
while True:
 accX = hamster.acceleration_x()  # x축 가속도 값
 if accX > 1500: # 앞쪽이 아래로 기울어졌다.
   hamster.wheels(-30)  # 뒤로 이동한다.
 elif accX < -1500: # 뒤쪽이 아래로 기울어졌다.
   hamster.wheels(30)  # 앞으로 이동한다.

 wait(20)  # 너무 빨리 반복하지 않도록 한다.
```

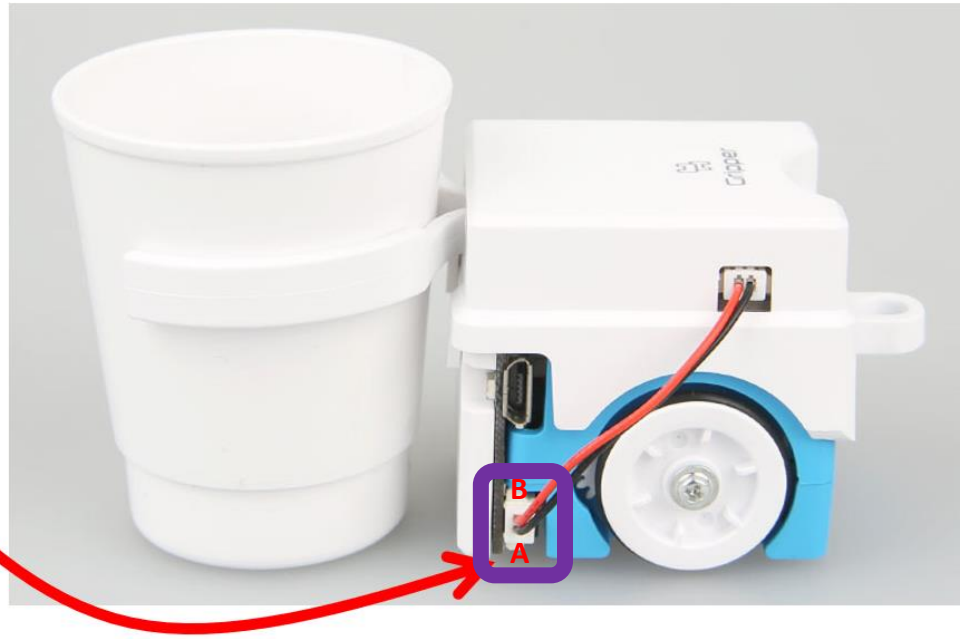
8차시

그리퍼 활용

그리퍼 활용– 연결 방법

- 본 예제는 햄스터 s를 이용한 것으로 예제 활용을 위해 햄스터 S와 햄스터용 그리퍼를 준비해 주세요.
  햄스터용 그리퍼는 햄스터와 햄스터 S 모두 사용 가능합니다.
- 햄스터에 그리퍼를 씌운 후 좌우측 선을 햄스터의 확장 포트에 연결해 사용 준비를 해주세요.

연결 방법은 유튜브에서 확인하실 수 있습니다 . → [영상으로 확인하기]



그리퍼 활용– 관련 메소드

그리퍼를 사용하기 위해서는 햄스터에서 제공하는 확장포트(a포트와 b포트)를 이용해야 합니다.
관련 함수와 메소드는 오른쪽 그림과 같습니다.

io_mode_a(mode)

io_mode_b(mode)

output_a(value)

output_b(value)

open_gripper()

close_gripper()

release_gripper()

그리퍼 활용– 관련 메소드

io_mode_a(mode)

A포트에 대해 입력/출력 모드를 결정할 수 있는 메소드입니다. 그리퍼를 사용하기 위해 반드시 사용해야 되며, 파라미터는 디지털 출력모드로 지정해주어야 합니다.

io_mode_a(mode) 버전 1.3.0부터

외부 확장 포트 중에서 포트 A의 입출력 모드를 설정한다.

파라미터:

- mode: 포트 A의 입출력 모드(정수)

입출력 모드	상수 값	설명
IO_MODE_ANALOG_INPUT (아날로그 입력 모드)	0	입력 전압을 8비트 ADC로 측정한다. 최대 입력 전압인 3.3V가 입력되면 255의 값을 가진다.
IO_MODE_DIGITAL_INPUT (디지털 입력 모드)	1	입력 전압을 0과 1로 변환한다. 입력 전압이 1.6V 이상이면 1로 하고, 아니면 0으로 한다.
IO_MODE_SERVO_OUTPUT (아날로그 서보 출력 모드)	8	외부 서보 제어용 PWM 신호를 출력한다.
IO_MODE_PWM_OUTPUT (PWM 출력 모드)	9	듀티비(0 ~ 255 단계)에 따른 PWM 파형을 출력한다.
IO_MODE_DIGITAL_OUTPUT (디지털 출력 모드)	10	디지털 값 LOW(0) 또는 HIGH(1)를 출력한다.

```
from roboid import *

hamster = Hamster()

# 포트 A의 입출력 모드를 아날로그 입력 모드로 설정한다.
hamster.io_mode_a(Hamster.IO_MODE_ANALOG_INPUT)
```

그리퍼 활용- 관련 메소드

output_a(value)

output_b(value)

A와 B포트에 대해 디지털 출력
(H또는 L)을 결정하는 메소드입니다. 파라미터에 0을 넣어주면 LOW, 1을 넣어주면 HIGH로 출력됩니다.

output_a(value) 버전 1.3.0부터

외부 확장 포트 중에서 포트 A로 출력하는 신호 값을 설정한다.

외부 확장 포트 A의 입출력 모드에 따라 설정할 수 있는 값의 범위가 달라진다.

파라미터:

- value: 포트 A로 출력하는 신호 값(정수)

입출력 모드	값의 범위	설명
IO_MODE_SERVO_OUTPUT (아날로그 서보 출력 모드)	0 ~ 180 (0: off)	외부 서보 제어용 PWM 신호를 출력하며, 20msec의 주기를 갖는 PWM 파형에서 ON 상태의 펄스 폭을 설정한다. 유효한 값은 1 ~ 180도이며, 펄스 폭 1.0msec ~ 2.0msec에 대응된다. (90도는 1.5msec에 대응) 0(off)인 경우에는 펄스 없이 항상 0이 출력되며, 180보다 큰 값을 입력하면 최대 펄스 폭인 2.5msec로 제한된다.
IO_MODE_PWM_OUTPUT (PWM 출력 모드)	0 ~ 255	20msec의 주기를 갖는 PWM 파형에서 ON 상태의 듀티비를 0 ~ 255 단계로 설정한다.
IO_MODE_DIGITAL_OUTPUT (디지털 출력 모드)	0 또는 1	디지털 출력(0 또는 1)을 설정한다. 0을 입력하면 LOW, 0이 아닌 값을 입력하면 HIGH 값을 출력한다.

```
from roboid import *

hamster = Hamster()

hamster.io_mode_a(Hamster.IO_MODE_PWM_OUTPUT)
# PWM 출력을 50으로 설정한다.
hamster.output_a(50)
```

그리퍼 활용- 관련 메소드

open_gripper()

close_gripper()

release_gripper()

그리퍼를 제어할 수 있는 메소드입니다. 앞서, output_a 또는 output_b 메소드로 그리퍼를 제어할 수 도 있지만, 해당 메소드들 또한 그리퍼를 제어할 수 있습니다.

open_gripper() 버전 1.4.0부터

집게를 연다.

close_gripper() 버전 1.4.0부터

집게를 닫는다.

release_gripper() 버전 1.4.0부터

집게의 전원을 끄고 자유롭게 움직일 수 있도록 한다.

open_gripper 또는 close_gripper 메소드를 사용하여 집게를 열거나 닫으면 상태를 유지하기 위해 집게의 전원이 계속 켜져 있다. 상태를 계속 유지할 필요가 없는 경우에는 배터리 소모를 줄이기 위해 집게의 전원을 끄는 것이 좋다.

```
from roboid import *

hamster = Hamster()

# 집게를 열고 전원을 끈다.
hamster.open_gripper()
hamster.release_gripper()
```

그리퍼 활용- 예제

그리퍼를 제어하는 방법은 두 가지가 있습니다.

1. 직접 입/출력 제어모드를 통해 그리퍼를 제어하는 방법
2. 라이브러리 내, 그리퍼 제어 관련 메소드 이용하는 방법

03 그리퍼 활용- 예제(열고 닫기)

- 1초에 한번씩 그리퍼를 열고 닫는 예제
- 직접 입/출력 제어모드 활용하여 그리퍼 제어

```
from roboid import *

hamster = HamsterS()
hamster.io_mode_a(hamster.IO_MODE_DIGITAL_OUTPUT)
hamster.io_mode_b(hamster.IO_MODE_DIGITAL_OUTPUT)

# 그리퍼 닫기 〉( a : b ) = ( 0 : 1)
# 그리퍼 열기 〉( a : b ) = ( 1 : 0)

# gripper 함수 〉1==OPEN,  0 == CLOSE
def gripper(state):
    if(state==1): # 그리퍼 벌리기
        hamster.output_a(1)
        hamster.output_b(0)
    elif (state==0): # 그리퍼 오므리기
        hamster.output_a(0)
        hamster.output_b(1)
    elif (state==-1): # 그리퍼 동작 x
        hamster.output_a(0)
        hamster.output_b(0)

while True:
    gripper(1)
    wait(1000)
    gripper(0)
    wait(1000)
```

03 그리퍼 활용- 예제(열고 닫기)

- 1초에 한번씩 그리퍼를 열고 닫는 예제
- 그리퍼 제어 관련 메소드 이용

```
from roboid import *

hamster = HamsterS()
hamster.io_mode_a(hamster.IO_MODE_DIGITAL_OUTPUT)
hamster.io_mode_b(hamster.IO_MODE_DIGITAL_OUTPUT)


while True:
    hamster.open_gripper()
    wait(1000)
    hamster.close_gripper()
    wait(1000)
```

03

그리퍼 활용– 예제(키보드 입력을 통한 제어)

- 키보드 입력을 통한 그리퍼 제어
- 1을 누르는 동안 그리퍼를 오므리고 2를 누르는 동안 그리퍼를 벌립니다.

```
from roboid import *
import keyboard

hamster = HamsterS()
hamster.io_mode_a(hamster.IO_MODE_DIGITAL_OUTPUT)
hamster.io_mode_b(hamster.IO_MODE_DIGITAL_OUTPUT)

# 그리퍼 닫기 〉( a : b ) = ( 0 : 1)
# 그리퍼 열기 〉( a : b ) = ( 1 : 0)

# gripper 함수 〉1 == OPEN,  0 == CLOSE , -1 == OFF
def gripper(state):
    if(state==1): # 그리퍼 벌리기
        hamster.output_a(1)
        hamster.output_b(0)
    elif (state==0): # 그리퍼 오므리기
        hamster.output_a(0)
        hamster.output_b(1)
    elif (state==-1): # 그리퍼 동작 x
        hamster.output_a(0)
        hamster.output_b(0)

while True:
    if keyboard.is_pressed("1"): # 키보드에서 1이 눌리면
        gripper(0)
    elif keyboard.is_pressed("2"): # 키보드에서 2이 눌리면
        gripper(1)
    elif keyboard.is_pressed("0"): # 키보드에서 0이 눌리면
        gripper(-1)
    wait(20) # 너무 빨리 반복하지 않도록함
```

03

그리퍼 활용– 예제(센서 입력을 통한 제어)

- 밝기 센서 입력을 통한 그리퍼 제어
- 1을 누르면 오므리고 2를 누르면 벌리고 다른 것을 누르면 그리퍼가 꺼지도록 합니다.
- 밝기 센서를 이용하여 물체가 바로 앞에 접근했을 때 그리퍼로 해당 물체를 잡으려 합니다.

```
from roboid import *

hamster = HamsterS()
hamster.io_mode_a(hamster.IO_MODE_DIGITAL_OUTPUT)
hamster.io_mode_b(hamster.IO_MODE_DIGITAL_OUTPUT)

# 그리퍼 닫기 〉( a : b ) = ( 0 : 1)
# 그리퍼 열기 〉( a : b ) = ( 1 : 0)

# gripper 함수 〉 1==OPEN,  0 == CLOSE
def gripper(state):
    if(state==1): # 그리퍼 벌리기
        hamster.output_a(1)
        hamster.output_b(0)
    elif (state==0): # 그리퍼 오므리기
        hamster.output_a(0)
        hamster.output_b(1)
    elif (state==-1): # 그리퍼 동작 x
        hamster.output_a(0)
        hamster.output_b(0)

while True:
    print(hamster.light())
    if hamster.light()>50: # 밝기 센서의 값이 50 이상일 경우
            gripper(1)
    else: gripper(0)
    wait(20) # 너무 빨리 반복하지 않도록함
```

03 그리퍼 활용– 예제(센서 입력을 통한 제어)

- 근접 센서를 이용한 그리퍼 제어
- 주행하다 근접 센서 값이 커지면 그리퍼로 해당 물체를 집으려 합니다.

```
from roboid import *

hamster = HamsterS()
hamster.open_gripper() # 그리퍼 열리게 설정
While True:
   hamster.wheels(30) # 양쪽 바퀴의 속도를 30으로 설정한다.
   if hamster.left_proximity()>50 and hamster.right_proximity()>50:
      # 양쪽 근접 센서가 50보다 크면
      hamster.stop() # 햄스터 정지
      wait(1000) # 1초간 대기
      hamster.close_gripper() # 그리퍼 닫음

   wait(20) # 너무 빨리 동작하지 않도록 설정
```

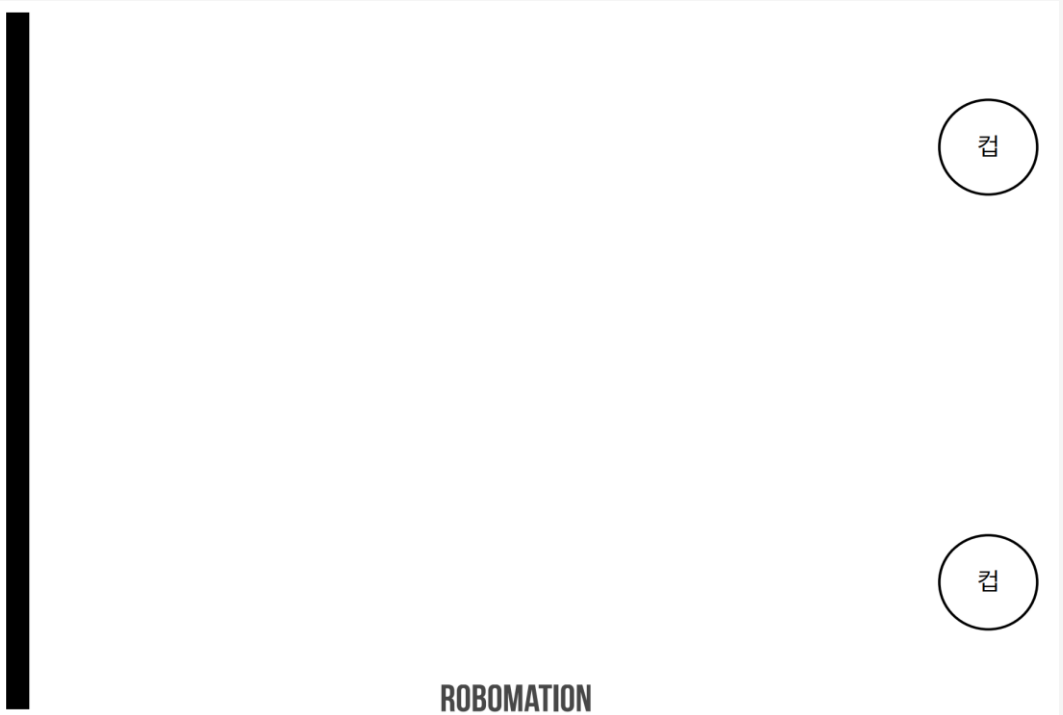
03 그리퍼 활용- 예제(키보드 조작모드 활용)

- 위의 수동 조작 모드 코드를 실행하면 햄스터를 원하는대로 움직이고 컵을 잡을 수 있습니다.
- 친구와 함께 컵을 지정된 위치에 두고 누가 더 먼저 컵을 들고 돌아오는지 대결해봅시다.

영상보기 클릭!

- 그림을 참고해서 “컵”이라고 쓴 곳에 그리퍼용 컵을 두고 남은 빈 공간에 종이컵 등과 같은 장애물을 친구들 각자의 창의성을 발휘하여 설치한 다음, 검은 선 위에서 출발하여 누가 컵을 먼저 가지고 돌아오는지 시합해보세요.

- 도안 출력 (A3 크기로 출력): https://robomation.net/?p=4465



그리퍼 활용– 예제(키보드 조작모드 활용)

```
from roboid import *
import keyboard

hamster = HamsterS()
hamster.io_mode_a(hamster.IO_MODE_DIGITAL_OUTPUT)
hamster.io_mode_b(hamster.IO_MODE_DIGITAL_OUTPUT)

# 그리퍼 닫기 〉(a : b) = (0 : 1)
# 그리퍼 열기 〉(a : b) = (1 : 0)

while True:
    key= Keyboard.read()  # 키보드 이벤트를 얻는다.
    if keyboard.is_pressed("up"):   # 위 방향키를 눌렀을 경우
        hamster.wheels(30)
    elif keyboard.is_pressed("down"):   # 아래 방향키를 눌렀을 경우
        hamster.wheels(-30)
    elif keyboard.is_pressed("right"):   # 오른쪽 방향키를 눌렀을 경우
        hamster.wheels(30, -30)
    elif keyboard.is_pressed("left"):   # 왼쪽 방향키를 눌렀을 경우
        hamster.wheels(-30, 30)
    elif keyboard.is_pressed("z"):   # z키를 눌렀을 경우
        hamster.close_gripper()
    elif keyboard.is_pressed("x"):   # x키를 눌렀을 경우
        hamster.open_gripper()
    elif keyboard.is_pressed("c"):   # c키를 눌렀을 경우
        hamster.release_gripper()
    elif keyboard.is_pressed(" "):   # 스페이스 바를 눌렀을 경우
        hamster.wheels(0)
    wait(20)   # 너무 빨리 반복하지 않도록 한다.
```

03 그리퍼 활용- 예제(심부름꾼 햄스터!)

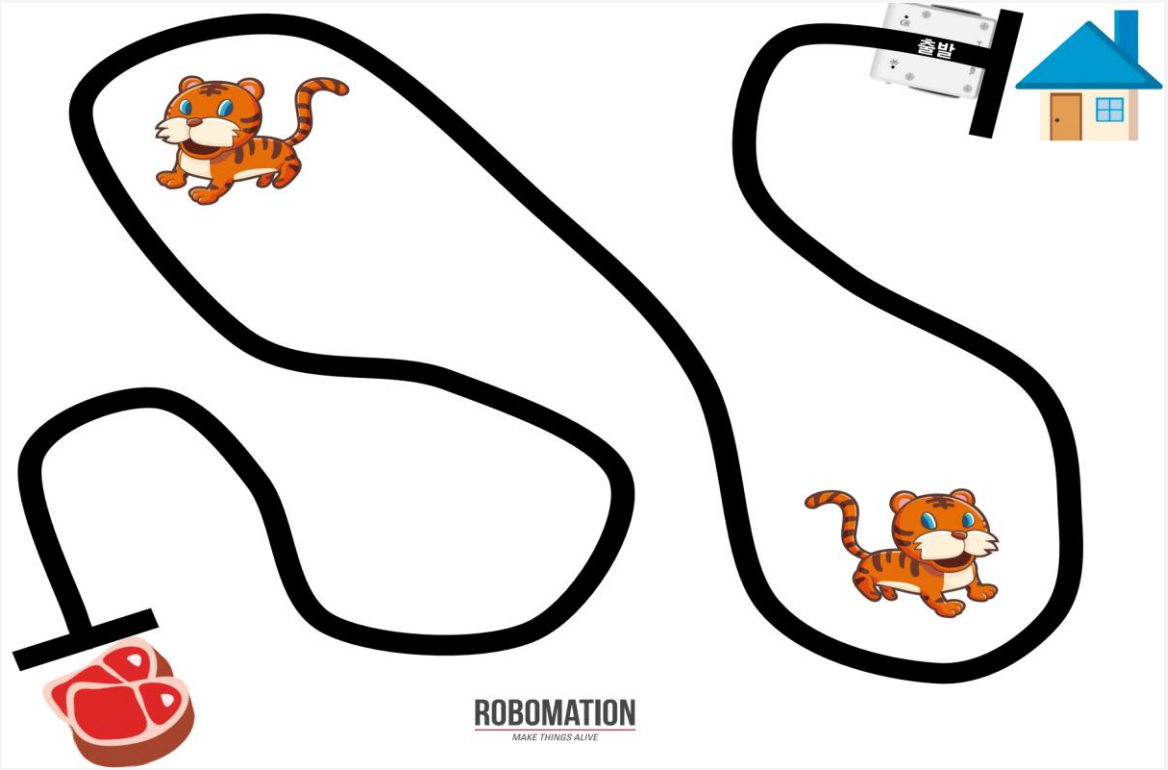
- 햄스터가 집에서 출발하여 고기를 사서 돌아오는 심부름을 해야합니다.
- 햄스터가 오가는 길에 있는 무시무시한 호랑이를 피해 고기를 사서 집으로 무사히 돌아올 수 있도록 명령해주세요.

영상보기 클릭!

- 고기 그림을 미리 오려서 준비해주세요.
- 검정색 라인을 잘 따라가면 호랑이를 피할 수 있습니다.
- 왼쪽 아래의 고기 그림에 도착하면 햄스터를 잠시 멈추게 한 후 손으로 직접 고기를 넣어주고 뒤를 돌아 집으로 돌아가면 됩니다.

- 햄스터가 로봇마다 모터 속도가 조금씩 차이가 나기 때문에 왼쪽 또는 오른쪽으로 몇 초 동안 돌아야 햄스터가 정확히 뒤로 돌지 직접 테스트 한 후 활동을 진행하세요.(힌트: 약 2초)

- 도안 출력 (A3크기로 출력): https://robomation.net/?p=4465



그리퍼 활용- 예제(심부름꾼 햄스터!)

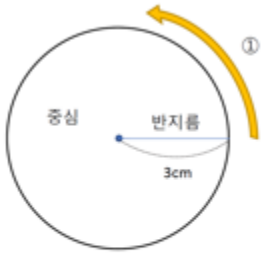
```
from roboid import *

hamster = HamsterS()
hamster.io_mode_a(hamster.IO_MODE_DIGITAL_OUTPUT)
hamster.io_mode_b(hamster.IO_MODE_DIGITAL_OUTPUT)

# 그리퍼 닫기 〉(a : b) = (0 : 1)
# 그리퍼 열기 〉(a : b) = (1 : 0)

hamster.close_gripper()
while True:
    print(hamster.left_floor(), hamster.right_floor())# 양쪽 바닥 센서 값 확인
    if (hamster.left_floor()>60 or hamster.right_floor()>60): #라인을 따라 갈 때 (양쪽 바닥 센서가 60보다 클 때)
     hamster.line_tracer_mode(3) #mode(3): 양쪽 모두 인식
     wait(20) # 너무 빨리 반복하지 않도록 한다.

    elif (hamster.left_floor()<40 and hamster.right_floor()<40): # 햄스터의 양쪽 센서 모두 검은 정지선을 밟을 때
     hamster.line_tracer_mode(3)
     hamster.stop()
     hamster.open_gripper() # 그리퍼 열기
     hamster.release_gripper()
     wait(500)
     hamster.wheels(-22,22) # 반바퀴 회전
```

8차시

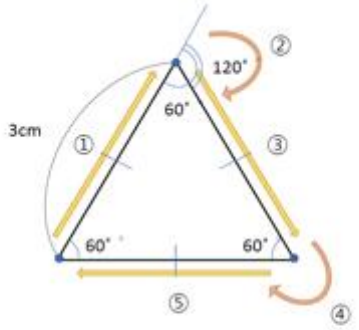
펜홀더 활용
(햄스터S 용)

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터의 앞쪽 방향으로 원을 그릴 수 있도록 해봅시다.

```
from roboid import *

hamster = HamsterS()

# 펜 위치는 오른쪽이며 햄스터가 왼쪽으로 돌며 원을 그립니다.
hamster.right_pen_circle_left(degree=360, radius=3, velocity=40)
```



반지름의 길이 : 3cm

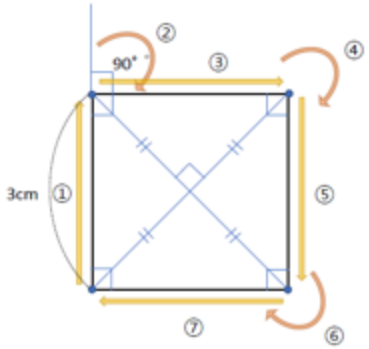


한 변의 길이 : 3cm

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터의 앞쪽 방향으로 정삼각형을 그릴 수 있도록 해봅시다.

```
from roboid import *

hamster = HamsterS()

# 펜 위치는 오른쪽입니다.
for _ in range(3):
    hamster.move_forward(cm=3, velocity=40) # 3cm만큼 앞으로 이동합니다.

    # 오른쪽 펜을 기준으로 제자리에서 120도 만큼 회전합니다.
    hamster.pivot_right_pen(degree=120, velocity=40)
```

초급



한 변의 길이 : 3cm

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터의 앞쪽 방향으로 정사각형을 그릴 수 있도록 해봅시다.

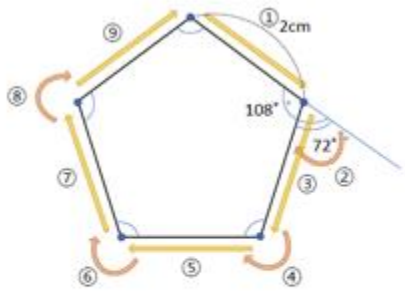
```
from roboid import *

hamster = HamsterS()

# 펜 위치는 오른쪽입니다.
for _ in range(4):
    hamster.move_forward(cm=3, velocity=40) # 3cm만큼 앞으로 이동합니다.

    # 오른쪽 펜을 기준으로 제자리에서 90도 만큼 회전합니다.
    hamster.pivot_right_pen(degree=90, velocity=40)
```

01

초급

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터의 앞쪽 방향으로 정오각형을 그릴 수 있도록 해봅시다.

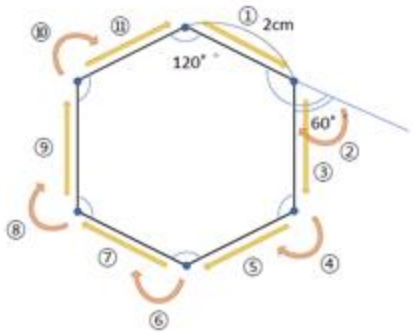
```
from roboid import *

hamster = HamsterS()

# 펜 위치는 오른쪽입니다.
for _ in range(5):
    hamster.move_forward(cm=2, velocity=40) # 2cm만큼 앞으로 이동합니다.

    # 오른쪽 펜을 기준으로 제자리에서 72도 만큼 회전합니다.
    hamster.pivot_right_pen(degree=72, velocity=40)
```



한 변의 길이 : 2cm

01

초급



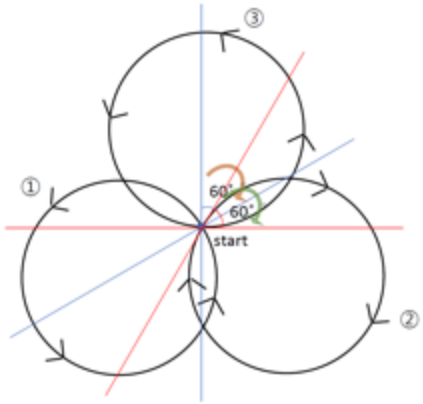
한 변의 길이 : 2cm

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터의 앞쪽 방향으로 정육각형을 그릴 수 있도록 해봅시다.

```
from roboid import *

hamster = HamsterS()

# 펜 위치는 오른쪽입니다.
for _ in range(6):
    hamster.move_forward(cm=2, velocity=40) # 2cm만큼 앞으로 이동합니다.

    # 오른쪽 펜을 기준으로 제자리에서 60도 만큼 회전합니다.
    hamster.pivot_right_pen(degree=60, velocity=40)
```

02

중급



원의 반지름: 2cm

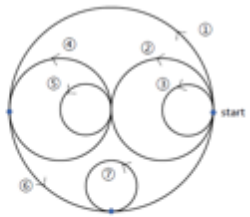
- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터로 원을 겹쳐서 그릴 수 있도록 해봅시다.
- .right_pen_circle_left 에서 같은 메소드라도 반지름에 따라 각각 다른 방향으로 동작할 수 있습니다. 이를 유의하여 주어진 방향대로 그림을 그릴 수 있도록 각도 부호를 유의하여 코드를 작성해주세요.

```
from roboid import *

hamster = HamsterS()

# 펜 위치는 오른쪽입니다.
hamster.right_pen_circle_left(degree=-360, radius=2, velocity=40)
# 오른쪽 펜을 기준으로 제자리에서 60도 만큼 회전합니다.
hamster.pivot_right_pen(degree=60, velocity=40)
hamster.right_pen_circle_left(degree=360, radius=2, velocity=40)
hamster.pivot_right_pen(degree=60, velocity=40)
hamster.right_pen_circle_left(degree=-360, radius=2, velocity=40)
```

02

중급

1. 얼굴 그리기
2. 오른쪽 눈 그리기
3. 오른쪽 눈동자 그리기
4. 왼쪽 눈 그리기
5. 왼쪽 눈동자 그리기
6. 입 모양 그리기



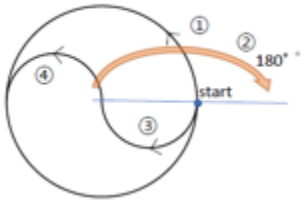
얼굴의 반지름: 2cm
눈의 반지름: 1cm
눈동자, 입의 반지름: 0.5cm

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터의 앞쪽 방향으로 얼굴 표정을 그릴 수 있도록 해봅시다.
- .right_pen_circle_left 에서 같은 메소드라도 반지름에 따라 각각 다른 방향으로 동작할 수 있습니다. 이를 유의하여 주어진 방향대로 그림을 그릴 수 있도록 각도 부호를 유의하여 코드를 작성해주세요.

```
from roboid import *

hamster = HamsterS()

# 펜 위치는 오른쪽입니다.
hamster.right_pen_circle_left(degree=-360, radius=2, velocity=40)
hamster.right_pen_circle_left(degree=-360, radius=1, velocity=40)
hamster.right_pen_circle_left(degree=-360, radius=0.5, velocity=40)
hamster.right_pen_circle_left(degree=-180, radius=2, velocity=40)
hamster.right_pen_circle_left(degree=-540, radius=1, velocity=40)
hamster.right_pen_circle_left(degree=-360, radius=0.5, velocity=40)
hamster.right_pen_circle_left(degree=-180, radius=1, velocity=40)
hamster.right_pen_circle_left(degree=-100, radius=2, velocity=40)
hamster.right_pen_circle_left(degree=-360, radius=0.5, velocity=40)
```

02

중급



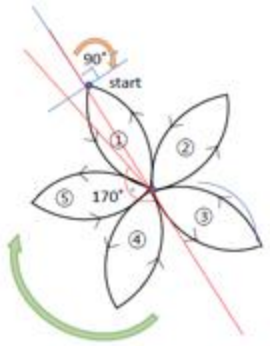
큰 원의 반지름: 2cm

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터로 태극 무늬를 그릴 수 있도록 해봅시다.
- .right_pen_circle_left 에서 같은 메소드라도 반지름에 따라 각각 다른 방향으로 동작할 수 있습니다. 이를 유의하여 주어진 방향대로 그림을 그릴 수 있도록 각도 부호를 유의하여 코드를 작성해주세요.

```
from roboid import *

hamster = HamsterS()

# 펜 위치는 오른쪽입니다.
hamster.right_pen_circle_left(degree=-360, radius=2, velocity=40)
# 오른쪽 펜을 기준으로 제자리에서 180도 만큼 회전합니다.
hamster.pivot_right_pen(degree=180, velocity=40)
hamster.right_pen_circle_right(degree=180, radius=1, velocity=40)
hamster.right_pen_circle_left(degree=-180, radius=1, velocity=40)
```

02

중급



꽃잎의 반지름: 2cm

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터로 꽃무늬를 그릴 수 있도록 해봅시다.
- .right_pen_circle_left 에서 같은 메소드라도 반지름에 따라 각각 다른 방향으로 동작할 수 있습니다. 이를 유의하여 주어진 방향대로 그림을 그릴 수 있도록 각도 부호를 유의하여 코드를 작성해주세요.

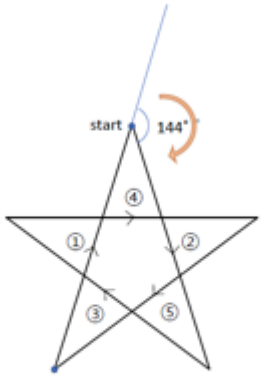
```
from roboid import *

hamster = HamsterS()

for _ in range(5):
    # 펜 위치는 오른쪽입니다.
    hamster.right_pen_circle_left(degree=-90, radius=2, velocity=40)
    # 오른쪽 펜을 기준으로 제자리에서 90도 만큼 회전합니다.
    hamster.pivot_right_pen(degree=90, velocity=40)
    hamster.right_pen_circle_right(degree=-90, radius=2, velocity=40)
    hamster.pivot_right_pen(degree=170, velocity=40)
```

02

중급

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터로 여러 가지 별을 그릴 수 있도록 해봅시다.

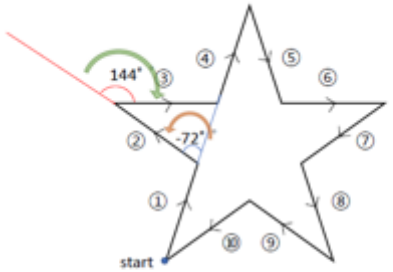
```
from roboid import *

hamster = HamsterS()
# 펜 위치는 오른쪽입니다.
for _ in range(5):
    hamster.move_forward(cm=3, velocity=40)
    # 오른쪽 펜을 기준으로 제자리에서 144도 만큼 회전합니다.
    hamster.pivot_right_pen(degree=144, velocity=40)
```



한 변의 길이: 3cm

02

중급



한 변의 길이: 2cm

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터로 여러 가지 별을 그릴 수 있도록 해봅시다.
- 각도의 부호에 유의하여 코드를 작성해주세요.

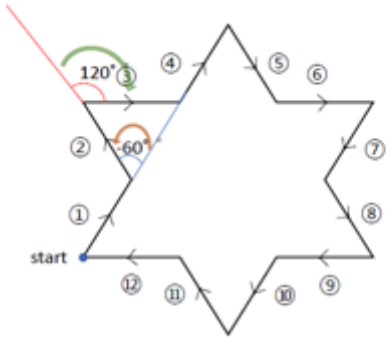
```
from roboid import *

hamster = HamsterS()
# 펜 위치는 오른쪽입니다.
for _ in range(5):
    hamster.move_forward(cm=2, velocity=40)
    # 오른쪽 펜을 기준으로 제자리에서 -72도 만큼 회전합니다.
    hamster.pivot_right_pen(degree=-72, velocity=40)
    hamster.move_forward(cm=2, velocity=40)
    # 오른쪽 펜을 기준으로 제자리에서 144도 만큼 회전합니다.
    hamster.pivot_right_pen(degree=144, velocity=40)
```

02

중급

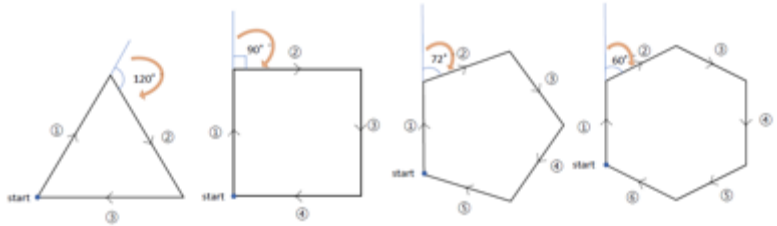
- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터로 여러 가지 별을 그릴 수 있도록 해봅시다.
- 각도의 부호에 유의하여 코드를 작성해주세요.

```
from roboid import *

hamster = HamsterS()
# 펜 위치는 오른쪽입니다.
for _ in range(6):
    hamster.move_forward(cm=2, velocity=40)
    # 오른쪽 펜을 기준으로 제자리에서 -60도 만큼 회전합니다.
    hamster.pivot_right_pen(degree=-60, velocity=40)
    hamster.move_forward(cm=2, velocity=40)
    # 오른쪽 펜을 기준으로 제자리에서 120도 만큼 회전합니다.
    hamster.pivot_right_pen(degree=120, velocity=40)
```



한 변의 길이: 2cm

03 고급



한 변의 길이: 2cm

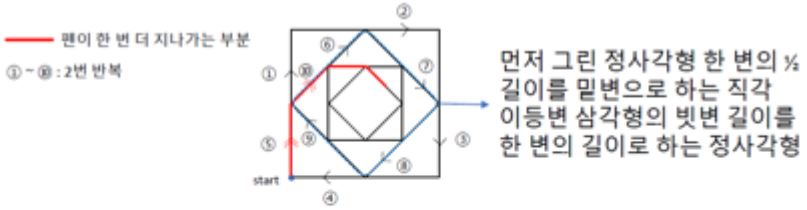
모든 도형의 외각의 합: 360°
➔ 정 n각형 한 모서리의 외각의 크기: (360/n)°

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터로 여러 다각형을 그릴 수 있도록 해봅시다.
- 코드를 실행하면 Run화면에 n:이 나타납니다.
- 원하는 도형의 변수를 정수로 입력하고 “enter 키”를 입력하면 햄스터S가 해당 도형을 그립니다.

```
from roboid import *

hamster = HamsterS()

# 펜 위치는 오른쪽입니다.

num = int(input("n: ")) # 정수 입력 받기.

for _ in range(num):
    hamster.move_forward(cm=2, velocity=40)
    hamster.pivot_right_pen(degree=360/num, velocity=40)
```

```
HamsterS[0] Connected: COM25 F8:43:F0:5D:F3:D2
n:3

Process finished with exit code 0
```

03 고급



가장 큰 사각형의 길이: 2cm

내각의 크기가 45도, 45도, 90도인 직각이등변 삼각형

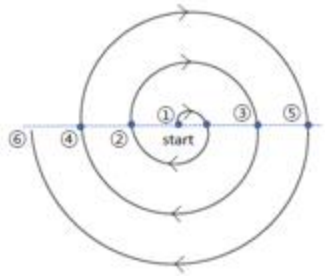
세 변의 길이의 비: a = $\sqrt{2}$(2n), $\sqrt{2}$ = 1.414

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터로 여러 개의 정사각형을 그릴 수 있도록 해봅시다.
- 코드를 실행하면 Run화면에 n:이 나타납니다.
- 원하는 변의 길이를 정수로 입력하고 “enter 키”를 입력하면 햄스터S가 도형을 그립니다.

```
from roboid import *

hamster = HamsterS()

# 펜 위치는 오른쪽입니다.

num = int(input("n: ")) # 정수 입력 받기.

for _ in range(4):
    for _ in range(4):
        hamster.move_forward(cm=num, velocity=40)
        hamster.pivot_right_pen(degree=90, velocity=40)
    num /= 2
    hamster.move_forward(cm=num, velocity=40)
    hamster.pivot_right_pen(degree=45, velocity=40)
    num *= 1.414
```

03

고급



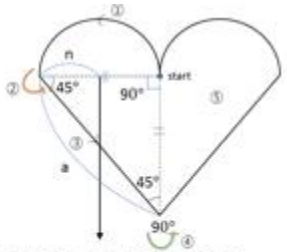
반원: 180°

반지름이 0.5cm씩 커지는 반원 6개

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터로 소용돌이를 그릴 수 있도록 해봅시다.
- 코드를 실행하면 Run화면에 n:이 나타납니다.
- 원하는 반지름 값을 정수로 입력하고 “enter 키”를 입력하면 햄스터S가 해당 도형을 그립니다.

```
from roboid import *

hamster = HamsterS()

# 펜 위치는 오른쪽입니다.

num = float(input(”n: ”)) # 실수 입력 받기.

for _ in range(6):
    hamster.left_pen_circle_right(degree=180, radius=num, velocity=40)
    num += 0.5
```

03 고급



반원의 반지름: ncm

내각의 크기가 45도, 45도, 90도인 직각이등변 삼각형
세 변의 길이의 비: a = √2 (2n)

- 햄스터의 펜의 위치는 오른쪽입니다.
- 햄스터로 하트 모양을 그릴 수 있도록 해봅시다.
- 코드를 실행하면 Run화면에 n:이 나타납니다.
- 원하는 반지름 값을 정수로 입력하고 “enter 키”를 입력하면 햄스터S가 해당 도형을 그립니다..

```
from roboid import *
from math import sqrt

hamster = HamsterS()

# 펜 위치는 오른쪽입니다.

num = float(input(”n: ”)) # 실수 입력 받기.

hamster.right_pen_circle_left(degree=-180, radius=num, velocity=40)
hamster.pivot_right_pen(degree=-45, velocity=40)
hamster.move_forward(cm=((sqrt(2)) * (2 * num)))
hamster.pivot_right_pen(degree=-90, velocity=40)
hamster.move_forward(cm=((sqrt(2)) * (2 * num)))
hamster.pivot_right_pen(degree=-45, velocity=40)
hamster.right_pen_circle_left(degree=-180, radius=num, velocity=40)
```

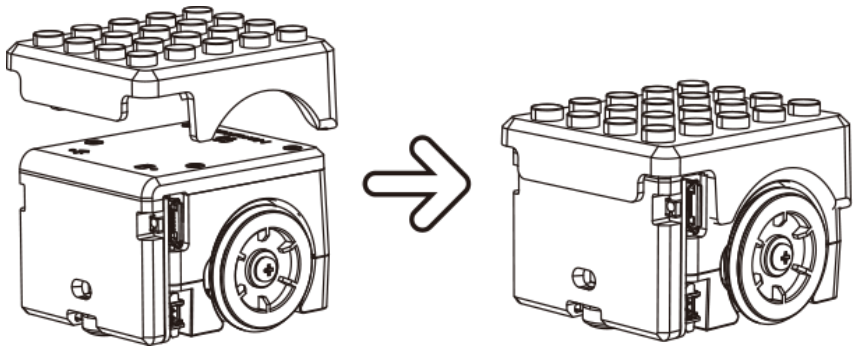
9차시

미로찾기 커버 활용 - 슬라럼

01

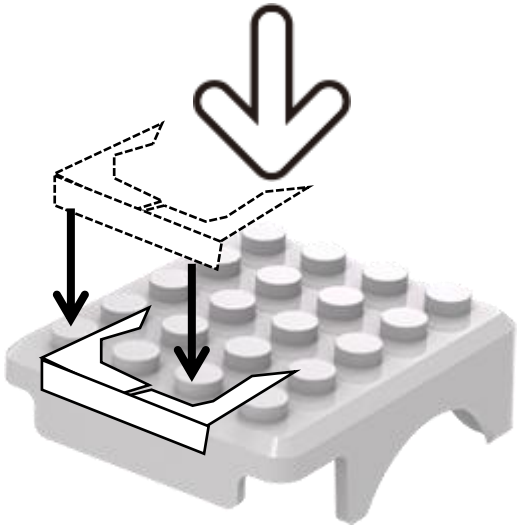
컵따라 돌기

- 커버 조립

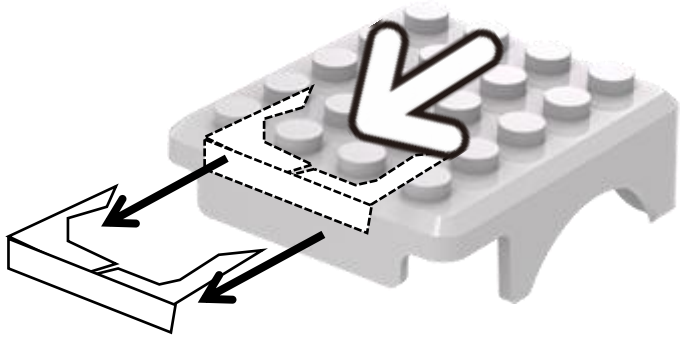


- 반사판 조립

위에서 누르면 끼울 수 있어요

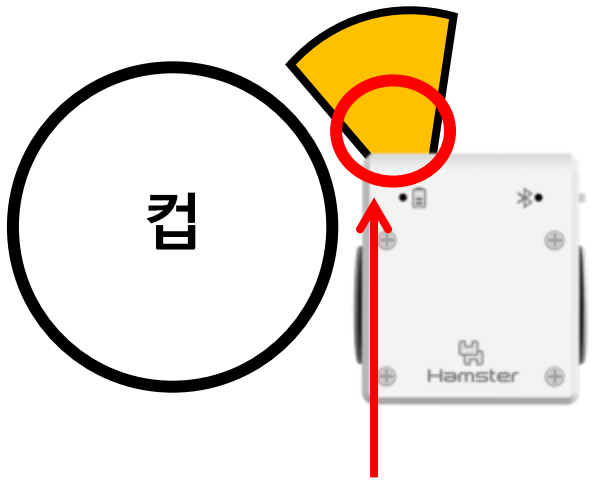


뒤에서 앞으로 밀면 뺄 수 있어요



01

컵 따라 돌기 (Beginner)



왼쪽 근접 센서

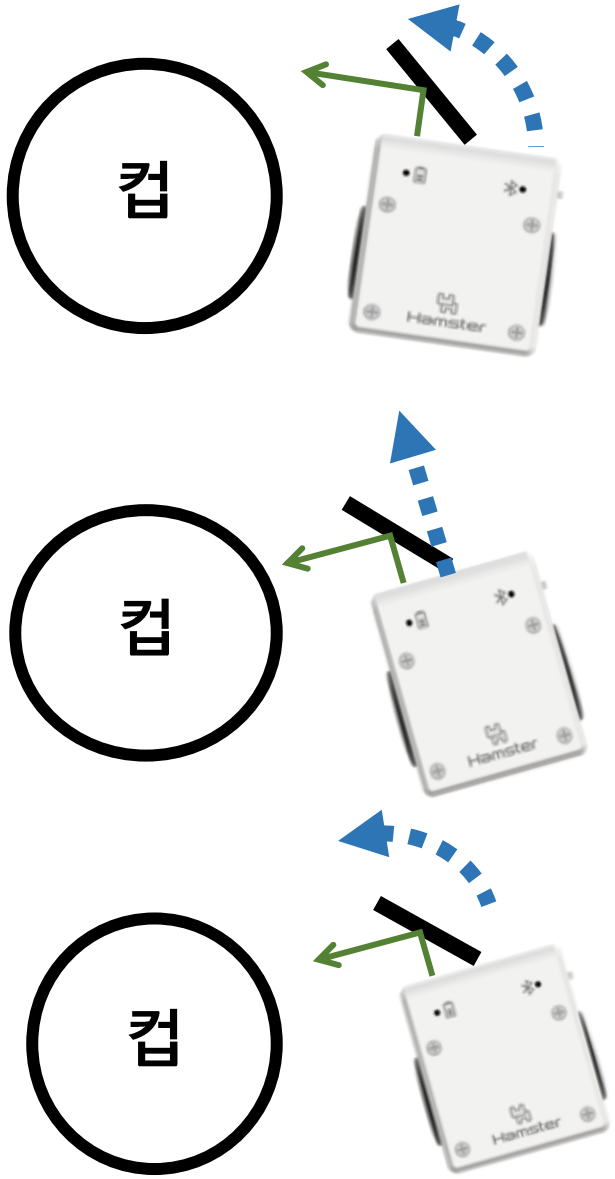
- 왼쪽 근접 센서의 값에 따라 양쪽 바퀴의 속도가 바뀌며 회전합니다. 왼쪽 센서 값이 10보다 커지면 왼쪽 바퀴가 오른쪽 바퀴보다 빨리 돌아 컵에 부딪히지 않게 주행하고, 왼쪽 센서 값이 10보다 작아지면 오른쪽 바퀴가 왼쪽보다 빨리 돌아 컵 쪽으로 주행합니다.

```
from roboid import *

hamster = HamsterS()

while True:
  if hamster.left_proximity() >10: # 왼쪽 근접 센서의 값이 10보다 커지면
     hamster.wheels(30,0) # 왼쪽 바퀴의 속도를 30, 오른쪽 바퀴의 속도를 0으로 설정한다.
   else: # 왼쪽 근접 센서의 값이 10보다 작으면
     hamster.wheels(0,30) # 왼쪽 바퀴의 속도를 0, 오른쪽 바퀴의 속도를 30으로 설정한다.
   wait(10)
```

- 왼쪽 바퀴와 오른쪽 바퀴의 속도를 변경하면서 로봇이 회전하는 원의 크기가 어떻게 달라지는지 관찰해 봅시다.
- 종이컵의 바깥을 돌 수 있도록 왼쪽 바퀴와 오른쪽 바퀴의 속도를 조정해 봅시다.

01

컵 따라 돌기 (Intermediate) –개선된 컵 따라 돌기

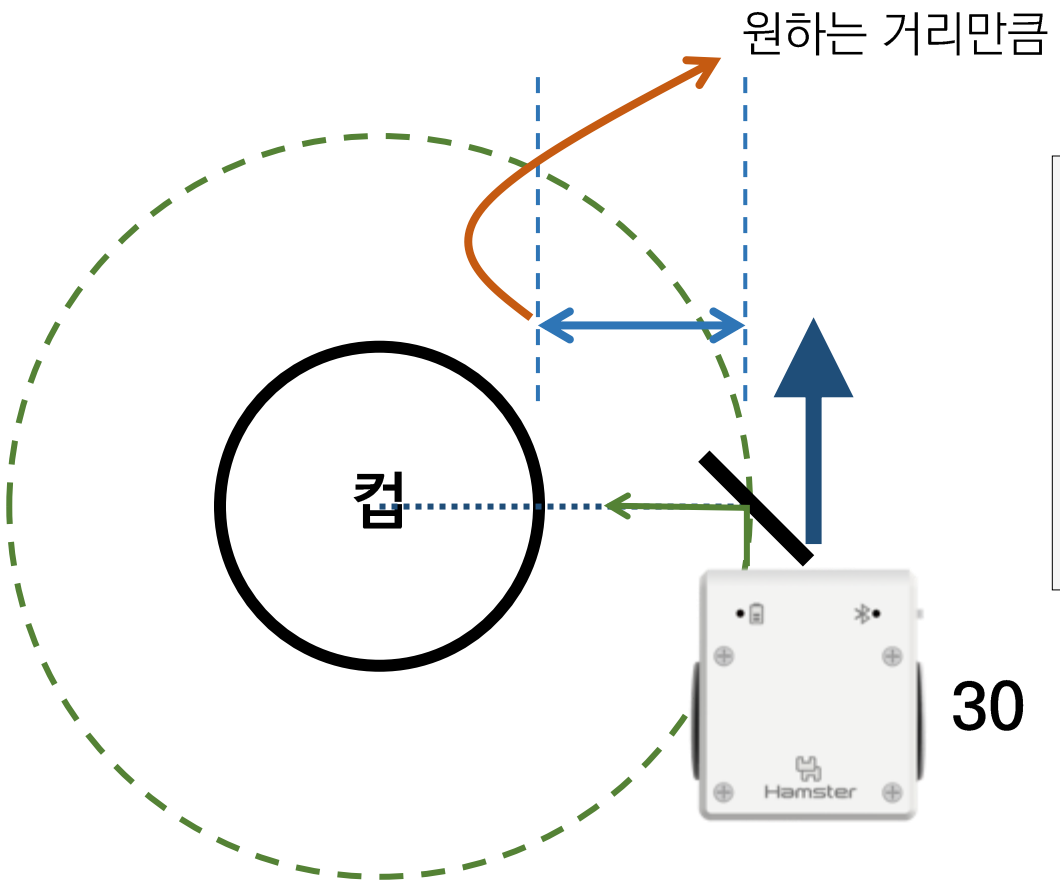


- 왼쪽 근접 센서의 값에 따라 양쪽 바퀴의 속도가 바뀌며 회전합니다. 왼쪽 센서 값이 10보다 커지면 양쪽 바퀴가 같은 속도로 컵에 부딪히지 않게 주행하고, 왼쪽 센서 값이 10보다 작아지면 오른쪽 바퀴가 왼쪽보다 빨리 돌아 컵 쪽으로 주행합니다.

```
from roboid import *

hamster = HamsterS()

while True:# 왼쪽 바퀴의 속도를 20, 오른쪽 바퀴의 속도를 40으로 설정한다.
  if hamster.left_proximity() >10: # 왼쪽 근접 센서의 값이 10보다 커지면
     hamster.wheels(30,30) # 왼쪽 바퀴의 속도를 30, 오른쪽 바퀴의 속도를 30으로 설정한다.
  else: # 왼쪽 근접 센서의 값이 10보다 작아지면
     hamster.wheels(0,30) # 왼쪽 바퀴의 속도를 0, 오른쪽 바퀴의 속도를 30으로 설정한다.
  wait(10)
```

01 원하는 거리만큼 떨어져서 돌기 (Advanced)

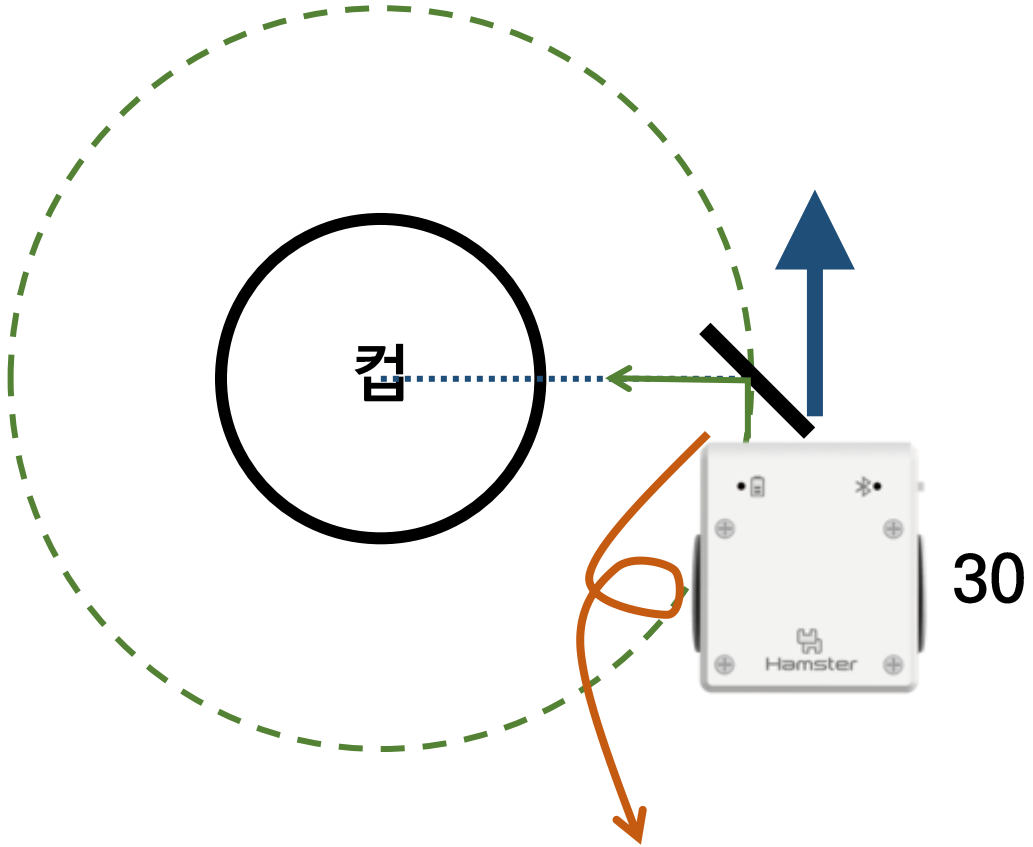
센서 위치를 컵의 중심에
맞춘다 !!




```
from roboid import *

hamster = HamsterS()

while True:
 print(hamster.left_proximity())  # 원하는 센서의 값 확인
 wait(10) #너무 빨리 반복하지 않도록 한다.
```

01 원하는 거리만큼 떨어져서 돌기 (Advanced)

센서 위치를 컵의 중심에
맞춘다 !!



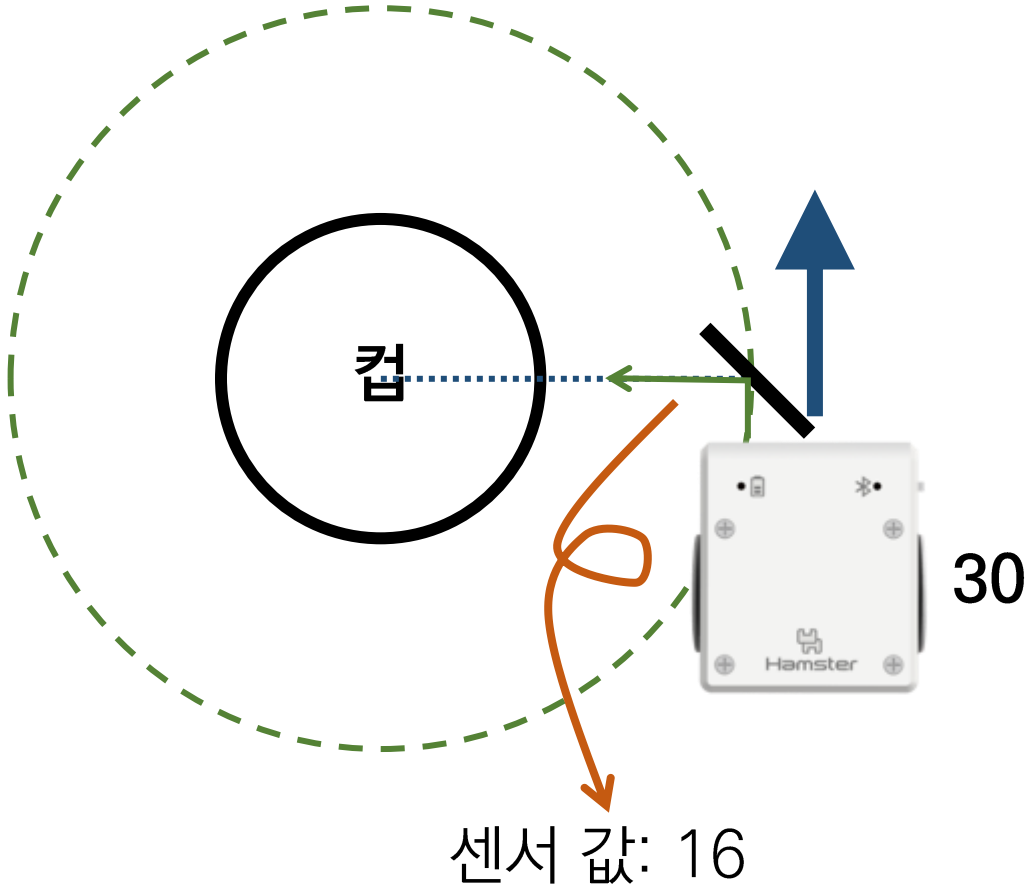
- 왼쪽 근접 센서의 값으로 양쪽 바퀴의 속도가 바뀌며 회전합니다. 오른쪽 바퀴는 항상 30의 속도이고 왼쪽 바퀴의 속도는 왼쪽 센서의 값으로 설정하여 주행합니다.

```
from roboid import *

hamster = HamsterS()

while True:
 hamster.wheels(hamster.left_proximity(),30)
 # 왼쪽 바퀴의 속도는 왼쪽 센서의 값으로 오른쪽 바퀴의 속도는 30으로 설정한다.
 wait(10)
```

센서 값: 16

01 원하는 거리만큼 떨어져서 돌기 (Advanced)

센서 위치를 컵의 중심에
맞춘다 !!



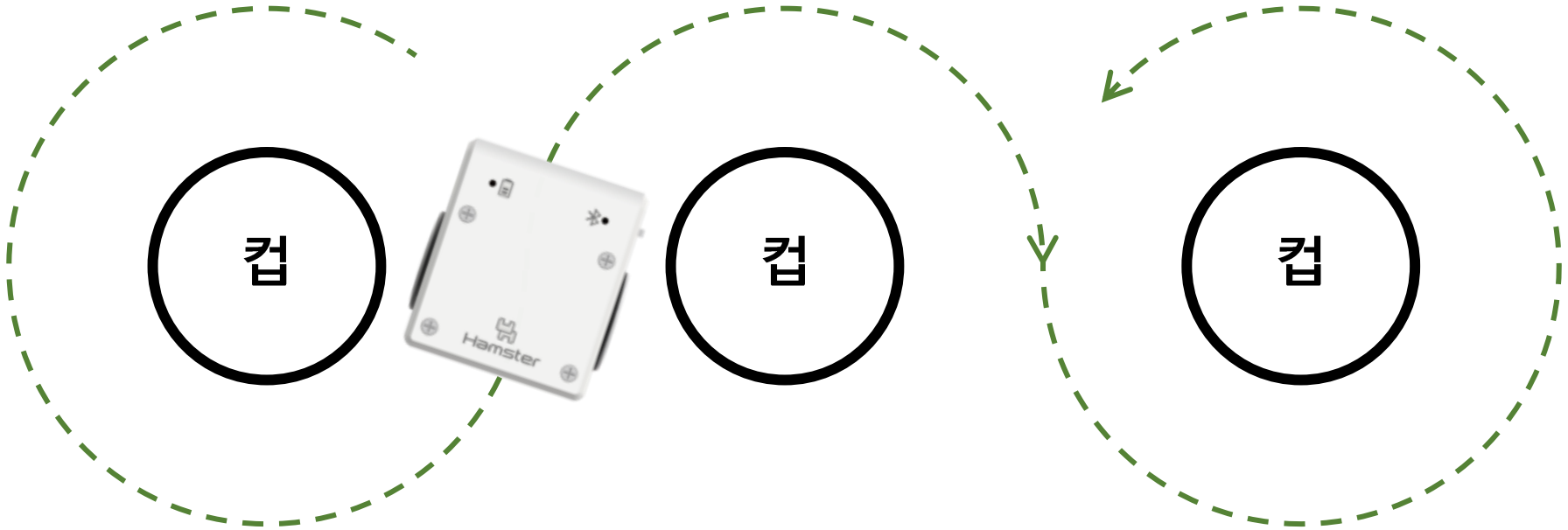
- 왼쪽 근접 센서의 값으로 양쪽 바퀴의 속도가 바뀌며 회전합니다. 오른쪽 바퀴는 항상 30의 속도이고 왼쪽 바퀴의 속도는 왼쪽 센서의 값으로 설정하여 주행합니다.

```
from roboid import *

hamster = HamsterS()

while True:
 hamster.wheels(hamster.left_proximity()*1.875,30)  # 왼쪽 바퀴의 속도는 왼쪽 센서의 값(16)과 기존 바퀴 속도(30)을 센서(16)로 나눈 값을 곱하여 설정한다.
    오른쪽 바퀴의 속도를 30으로 설정한다.
 wait(10)
```

30 ÷ 16 = 1.875

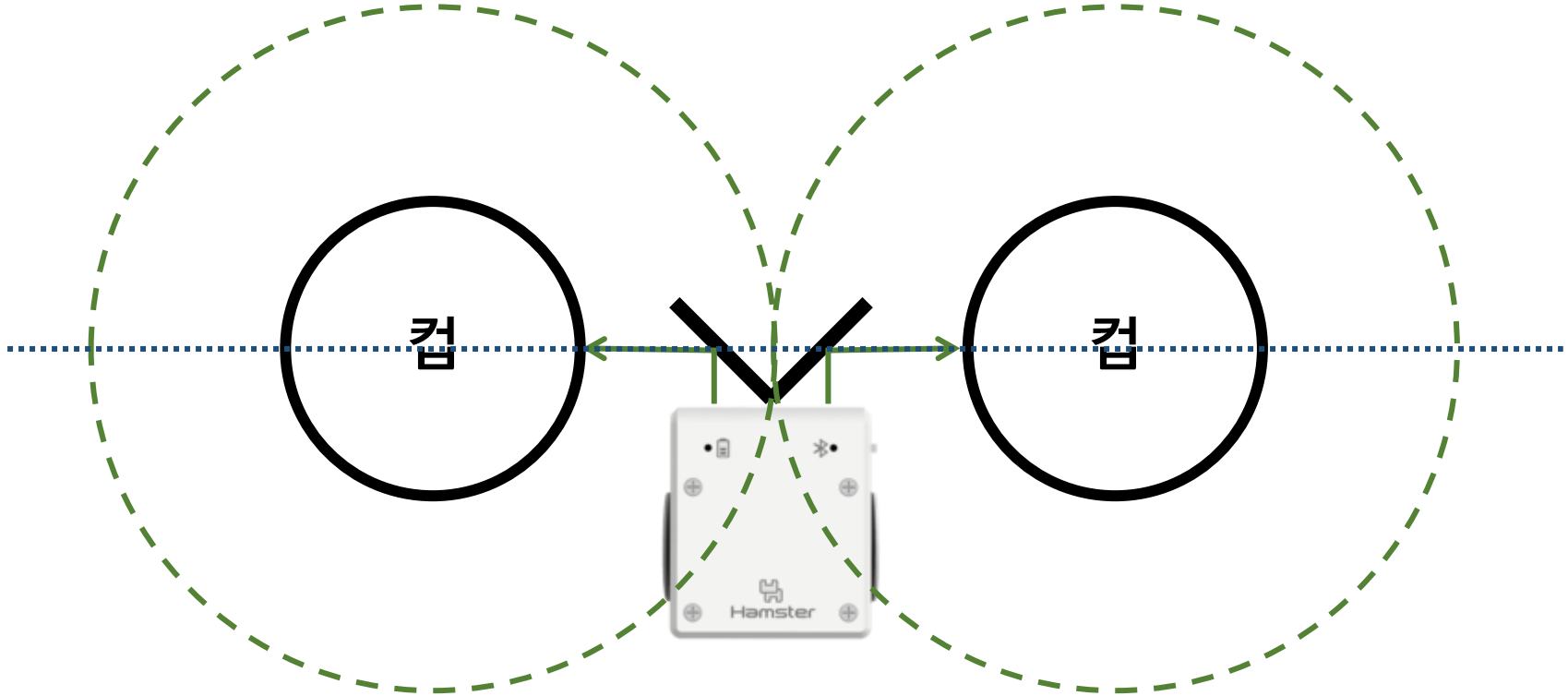
02

슬라럼



02

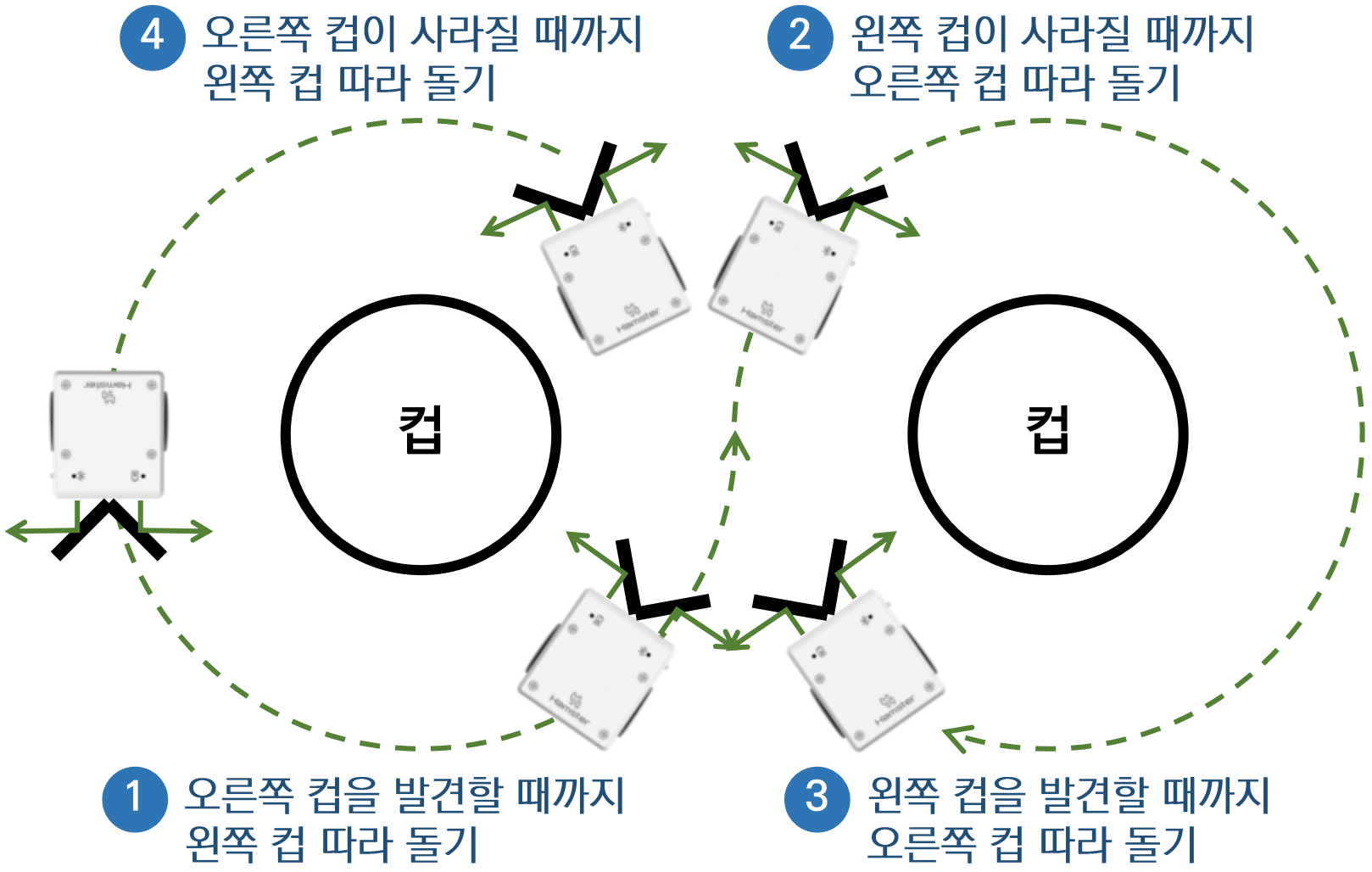
슬라럼



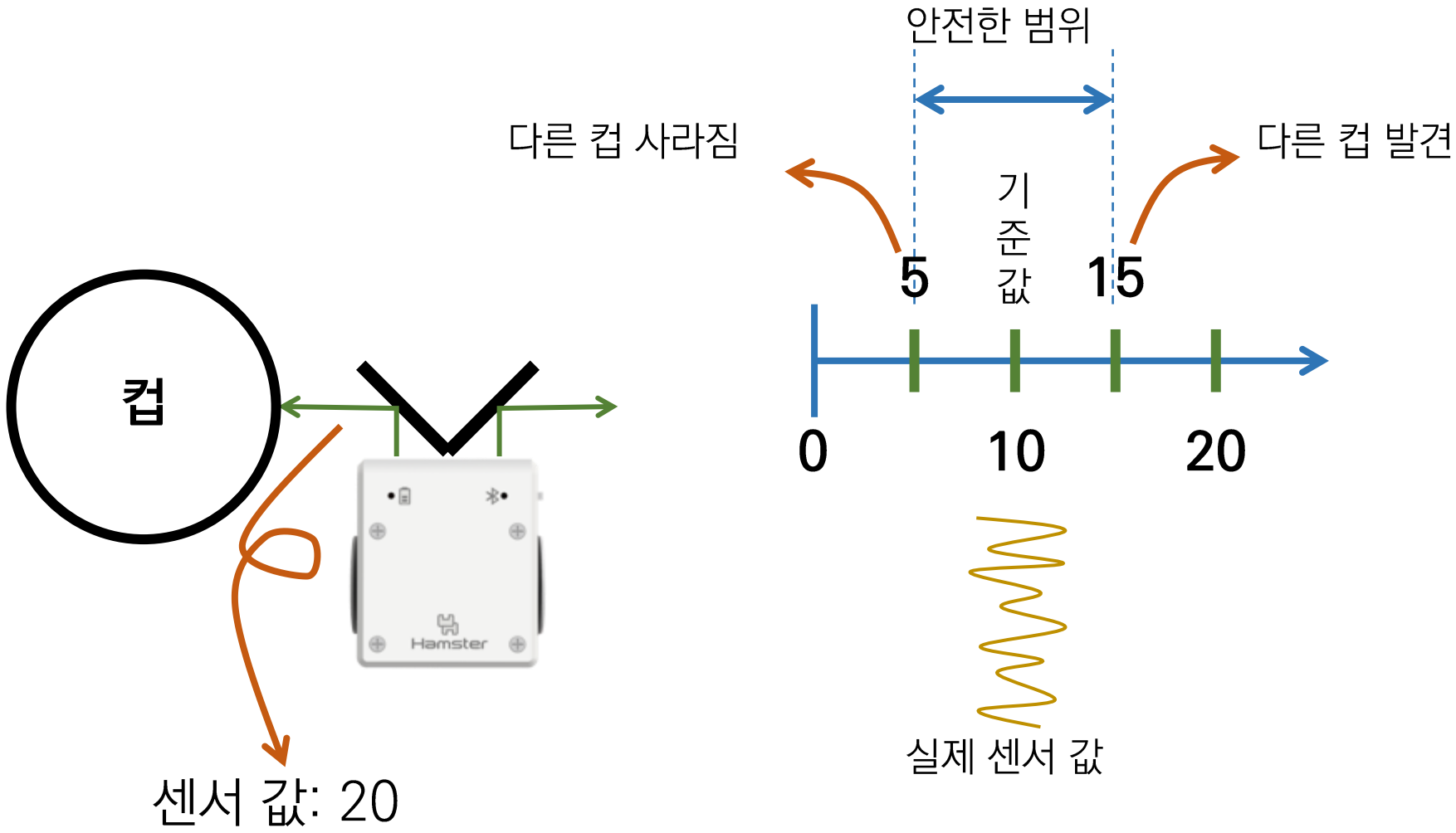
컵 중간으로 지나가게 하는 방법은?

02

슬라럼


4 오른쪽 컵이 사라질 때까지
왼쪽 컵 따라 돌기
2 왼쪽 컵이 사라질 때까지
오른쪽 컵 따라 돌기
컵
컵
1 오른쪽 컵을 발견할 때까지
왼쪽 컵 따라 돌기
3 왼쪽 컵을 발견할 때까지
오른쪽 컵 따라 돌기

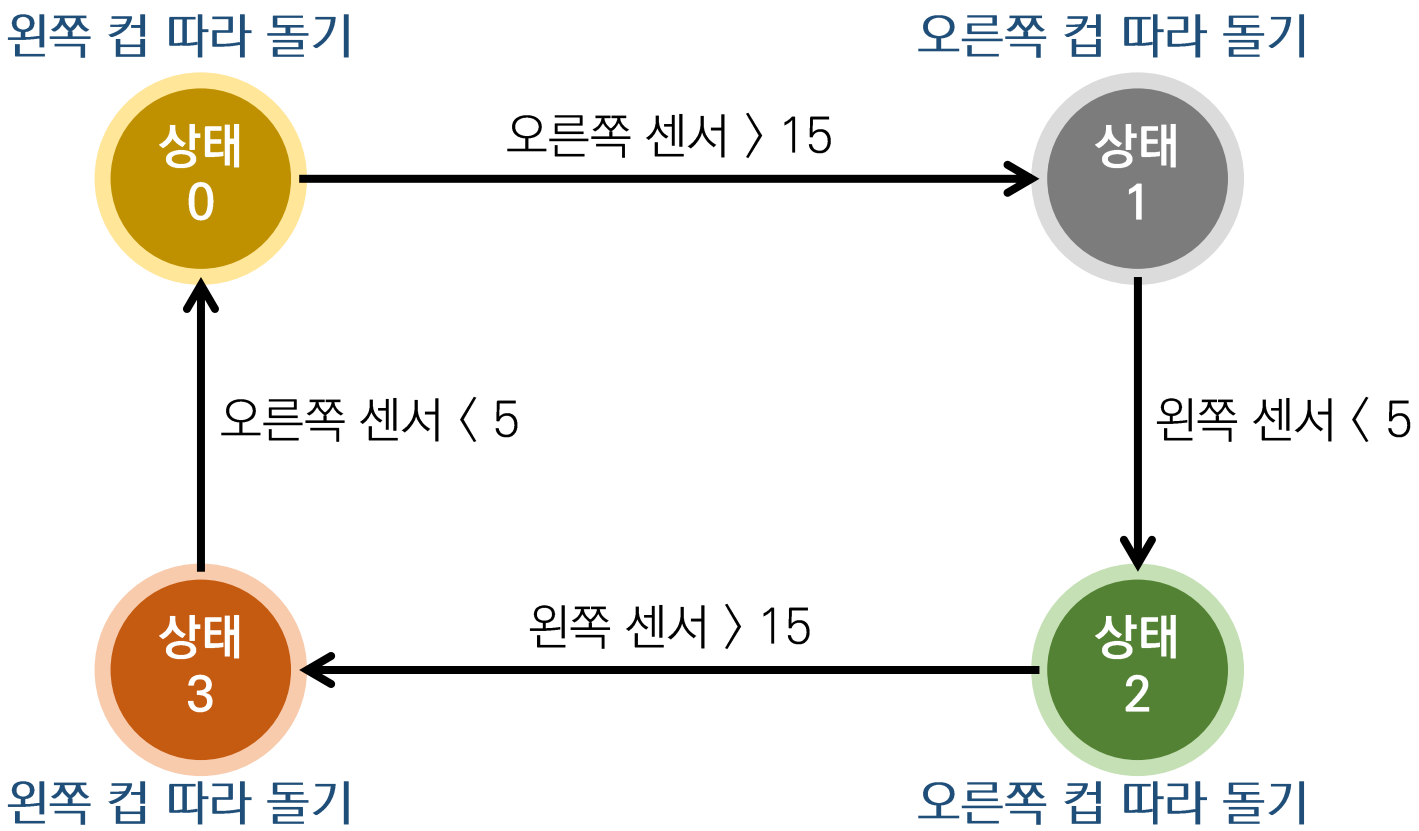
슬라럼



02

슬라럼

- 다이어그램


왼쪽 컵 따라 돌기
오른쪽 컵 따라 돌기
상태 0
상태 1
오른쪽 센서 〉 15
오른쪽 센서 〈 5
왼쪽 센서 〈 5
상태 3
상태 2
왼쪽 센서 〉 15
왼쪽 컵 따라 돌기
오른쪽 컵 따라 돌기

02

슬라럼

```
from roboid import *

hamster = HamsterS()

state= 0
count= 0 # 노이즈 값 세기
while True:
 left= hamster.left_proximity()
 right= hamster.right_proximity()
 if state==0:
    hamster.wheels(left*2.5, 50)
    if right > 15: count+=1
    else: count=0
    if count>10:
      count= 0
      state= 1

 elif state==1:
    hamster.wheels(50, right *2.5)
    if left < 5: count += 1
    else: count = 0
    if count > 10:
       count = 0
       state = 2
```

```
elif state==2:
   hamster.wheels(50, right*2.5)
   if left > 15: count+=1
   else: count=0
   if count>10:
     count= 0
     state= 3

else:
   hamster.wheels(left*2.5, 50)
   if left < 5: count += 1
   else: count = 0
   if count > 10:
      count = 0
      state = 0
wait(10)
```

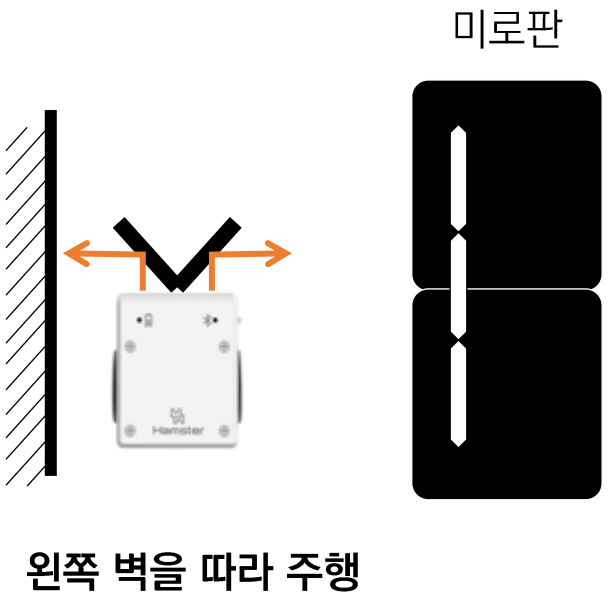
10차시

미로찾기 커버 활용 - 미로탈출

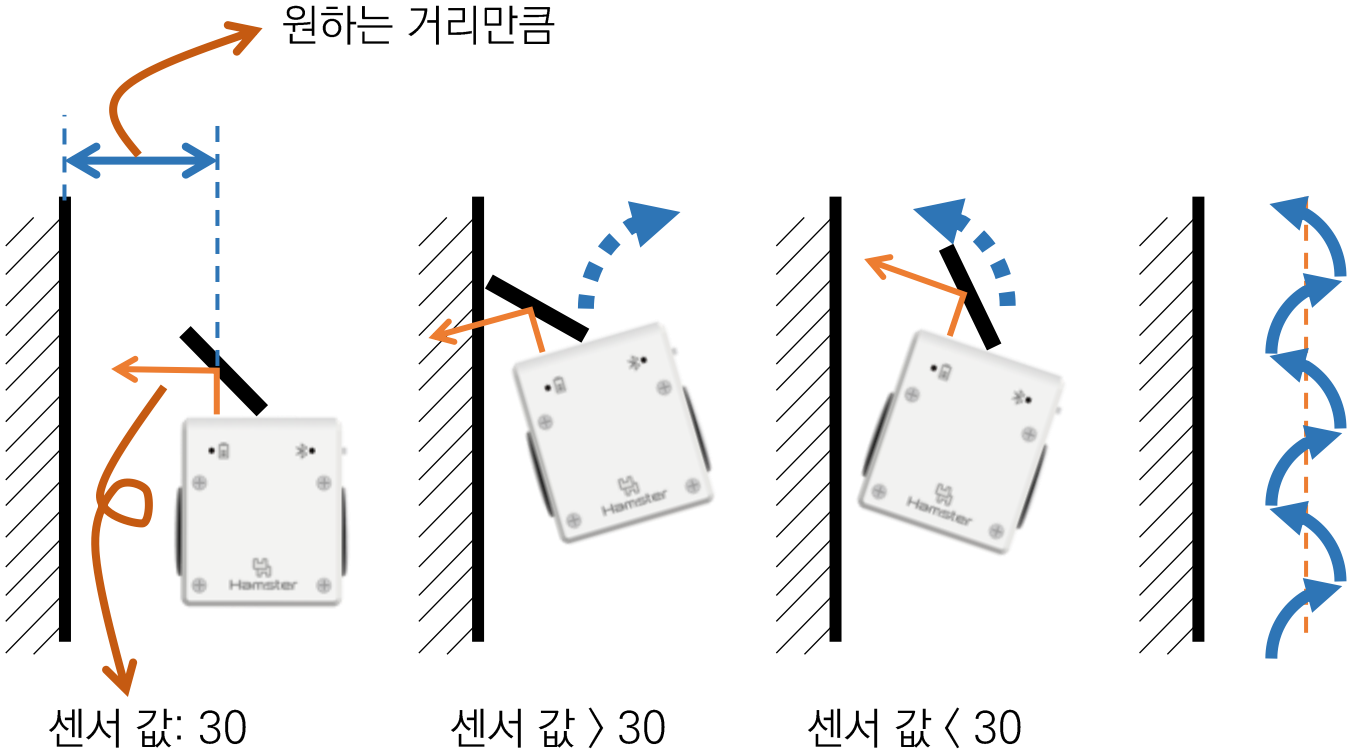
01

미로 탈출

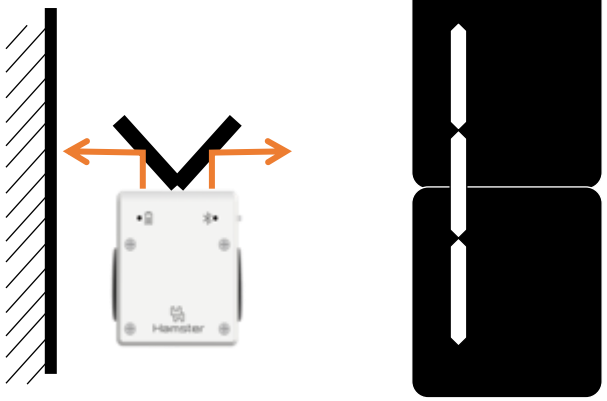
- 한 쪽 센서 사용하여 벽을 따라갈 수 있도록 해봅시다.



왼쪽 벽을 따라 주행



01 미로 탈출



왼쪽 벽을 따라 주행

미로판

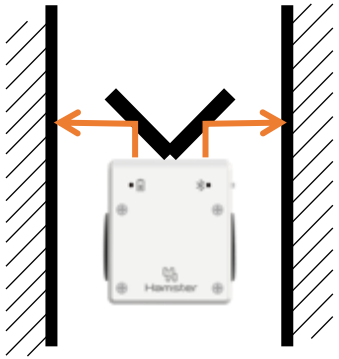
- 한 쪽 센서 사용하여 벽을 따라갈 수 있도록 해봅시다.
- 반사판을 끼워주세요.

```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.left_proximity() > 35:
        hamster.wheels(50, 0)
    else:
        hamster.wheels(0, 50)
    wait(10)
```

01

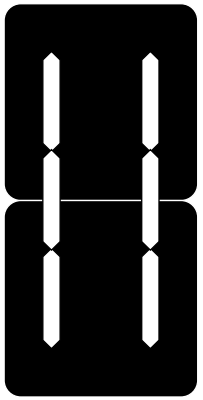
미로 탈출

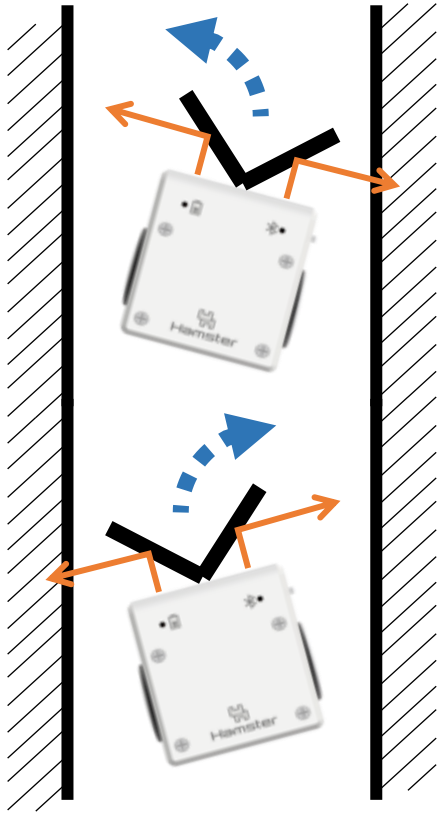
- 양쪽 센서 사용하여 벽을 따라갈 수 있도록 해봅시다.
- 반사판을 끼워주세요.



복도 중앙으로 주행

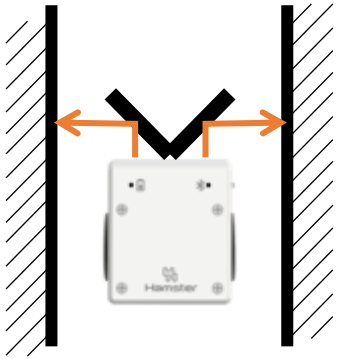
미로판





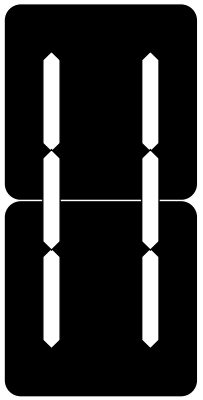
01 미로 탈출

- 양쪽 센서 사용하여 벽을 따라갈 수 있도록 해봅시다.
- 반사판을 끼워주세요.



복도 중앙으로 주행

미로판

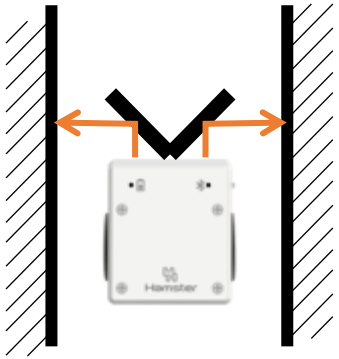


```
from roboid import *

hamster = HamsterS()

while True:
    hamster.wheels(50, 50)
    if hamster.left_proximity() > 35:
        hamster.wheels(50, -50)
    elif hamster.right_proximity() > 35:
        hamster.wheels(-50, 50)
    wait(10)
```

01

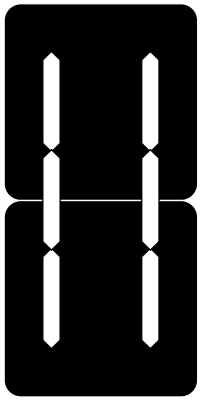
미로 탈출

- 양쪽 센서 사용하여 벽을 따라갈 수 있도록 해봅시다.
- 왼쪽 센서 값과 오른쪽 센서 값의 차이를 이용해서 주행해 보도록 합시다.

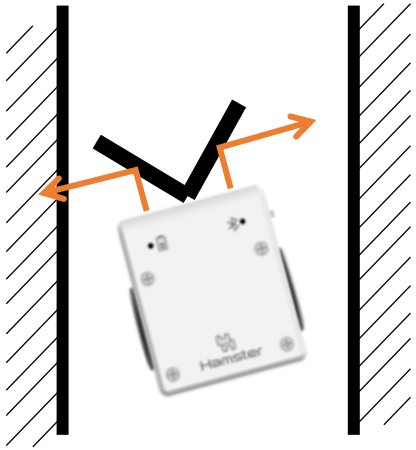


복도 중앙으로 주행

미로판



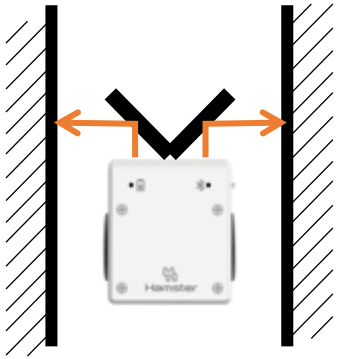
왼쪽 센서 값 〉 오른쪽 센서 값



```
hamster.left_proximity() > hamster.right_proximity()

hamster.left_proximity() - hamster.right_proximity() > 0
```

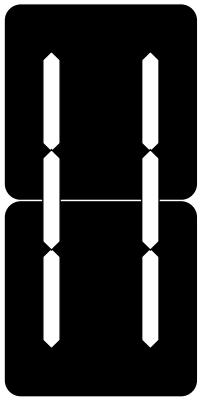
01

미로 탈출



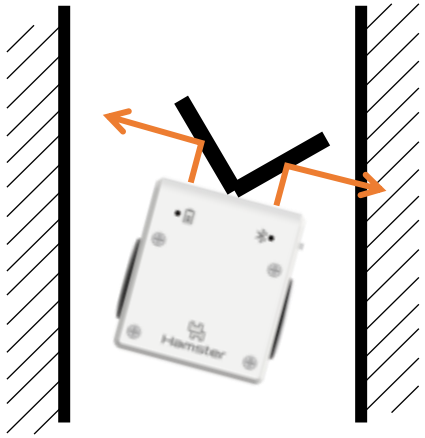
복도 중앙으로 주행

미로판



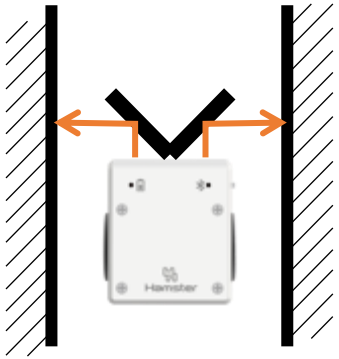
- 양쪽 센서 사용하여 벽을 따라갈 수 있도록 해봅시다.
- 왼쪽 센서 값과 오른쪽 센서 값의 차이를 이용해서 주행해 보도록 합시다.

왼쪽 센서 값 < 오른쪽 센서 값



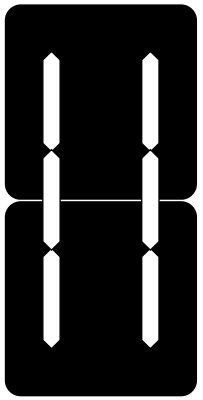
```
hamster.left_proximity() < hamster.right_proximity()

hamster.left_proximity() - hamster.right_proximity() < 0
```

01 미로 탈출



복도 중앙으로 주행

미로판

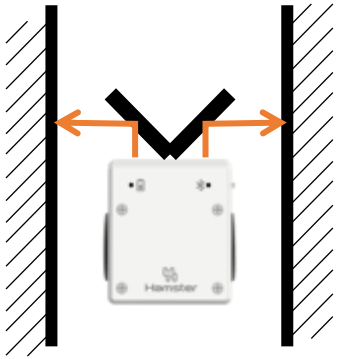


- 양쪽 센서 사용하여 벽을 따라갈 수 있도록 해봅시다.
- 왼쪽 센서 값과 오른쪽 센서 값의 차이를 이용해서 주행해 보도록 합시다.

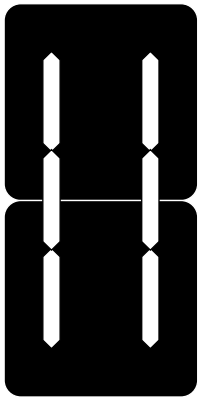
```
from roboid import *

hamster = HamsterS()

while True:
hamster.wheels(50, 50)
    diff = hamster.left_proximity() - hamster.right_proximity()
    if diff > 0:
        hamster.wheels(50, -50)
    elif diff < 0:
        hamster.wheels(-50, 50)
    wait(10)
```

01

미로 탈출

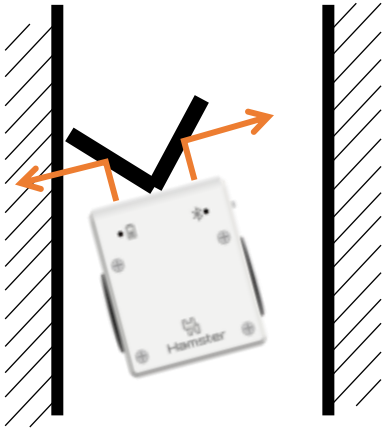


복도 중앙으로 주행

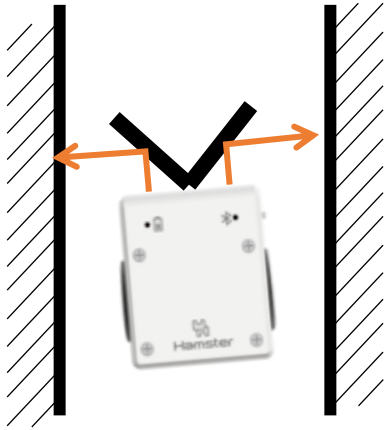


- 양쪽 센서 사용하여 벽을 따라갈 수 있도록 해봅시다.
- 왼쪽 센서와 오른쪽 센서의 차이에 따라 속도를 조절할 수 있도록 해봅시다.
- 차이가 큰 경우 더 많이 움직이고, 차이가 적은 경우 더 조금 움직여야 한다는 것을 코드로 표현해보도록 합시다.

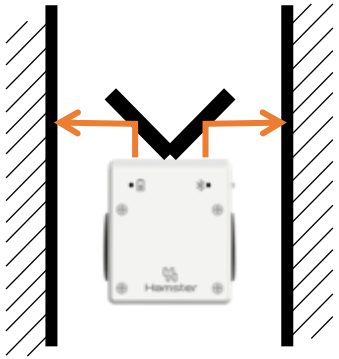
오른쪽으로 많이 움직여야...



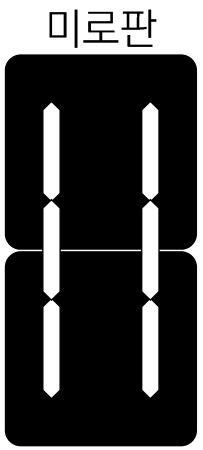
오른쪽으로 조금 움직이면...



01 미로 탈출



복도 중앙으로 주행



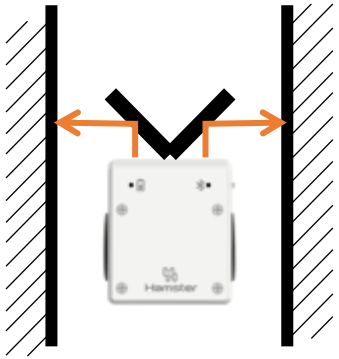
- 양쪽 센서 사용하여 벽을 따라갈 수 있도록 해봅시다.
- 왼쪽 센서와 오른쪽 센서의 차이에 따라 속도를 조절할 수 있도록 해봅시다.
- 차이가 큰 경우 더 많이 움직이고, 차이가 적은 경우 더 조금 움직여야 한다는 것을 코드로 표현해보도록 합시다.

```
from roboid import *

hamster = HamsterS()

while True:

    diff = hamster.left_proximity() - hamster.right_proximity()

    hamster.wheels(diff, -diff)

    wait(10)
```

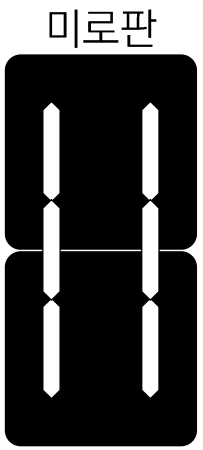
- 위 코드를 실행해보고 문제점이 무엇인지 알아봅시다.

01

미로 탈출



복도 중앙으로 주행



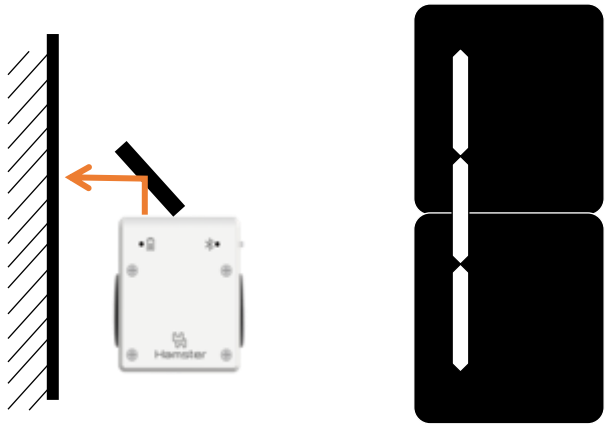
- 양쪽 센서 사용하여 벽을 따라갈 수 있도록 해봅시다.
- 왼쪽 센서와 오른쪽 센서의 차이에 따라 속도를 조절할 수 있도록 해봅시다.
- 차이가 큰 경우 더 많이 움직이고, 차이가 적은 경우 더 조금 움직여야 한다는 것을 코드로 표현해보도록 합시다.

```
from roboid import *

hamster = HamsterS()

while True:

    diff = hamster.left_proximity() - hamster.right_proximity()

    hamster.wheels(50 + diff * 1.3, 50 - diff * 1.3)

    wait(10)
```

01

미로 탈출

미로판



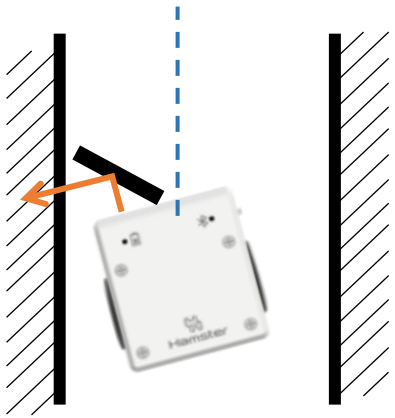
왼쪽 벽을 따라 주행

- 왼쪽 반사판으로 교체해 주세요.

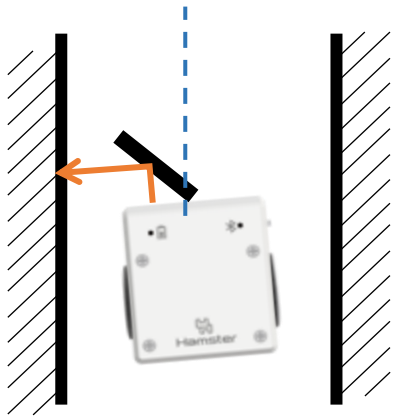
- 왼쪽 벽을 따라 주행하는 코드 또한 일정 조건이 아니라 차이를 이용하여 코드를 작성해봅시다.

```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.left_proximity() > 35:
        hamster.wheels(50, 0)
    else:
        hamster.wheels(0, 50)
    wait(10)
```

오른쪽으로 많이 움직여야...

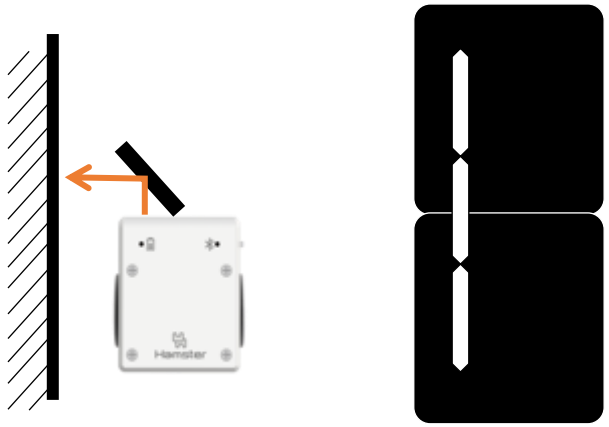


오른쪽으로 조금 움직이면...



01 미로 탈출

미로판



왼쪽 벽을 따라 주행

- 왼쪽 반사판으로 교체해 주세요.

- 왼쪽 벽을 따라 주행하는 코드 또한 일정 조건이 아니라 차이를 이용하여 코드를 작성해봅시다.

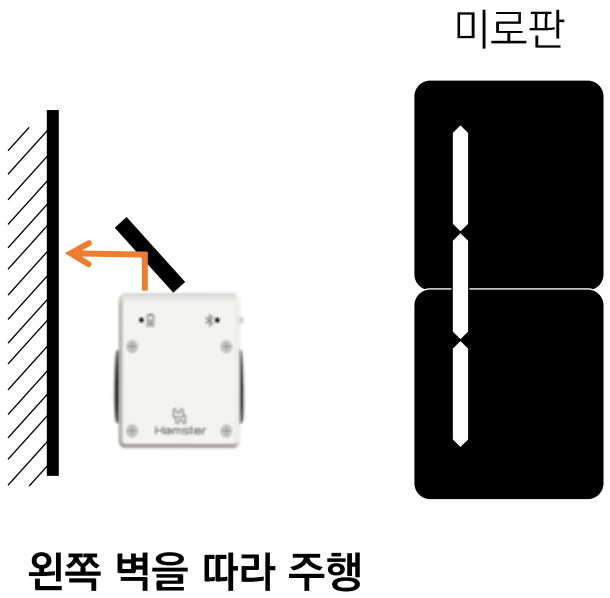
```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.left_proximity() > 35:
        hamster.wheels(50, 0)
    else:
        hamster.wheels(0, 50)
    wait(10)
```

```
from roboid import *

hamster = HamsterS()

while True:
    diff = hamster.left_proximity() - 35

    hamster.wheels(50 + diff * 0.8, 50 - diff * 0.8)

    wait(10)
```

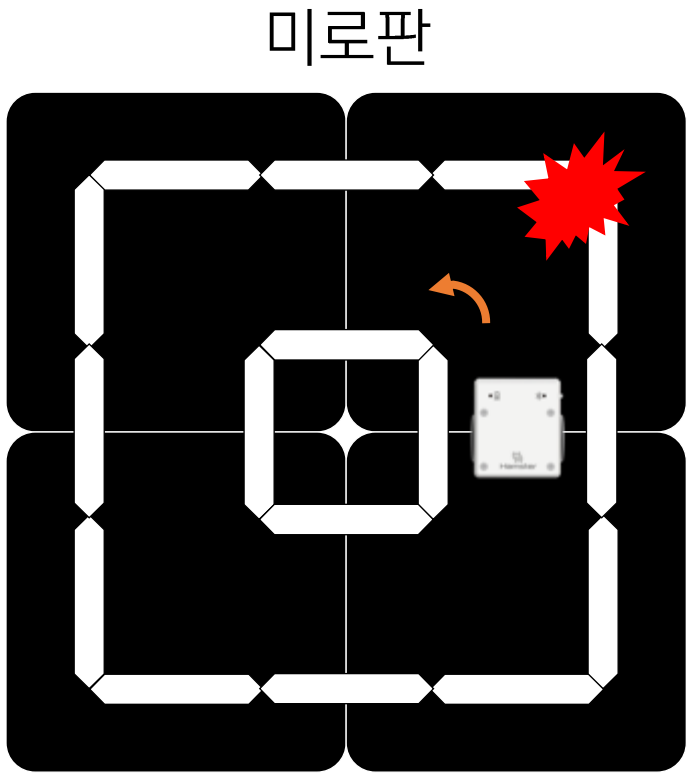
01

미로 탈출

- 미로에서 모퉁이에 부딪히는지 확인해봅시다.



왼쪽 벽을 따라 주행



01 미로 탈출

- 막힌 길에서 제자리를 돌아보도록 합시다.

- 센서 값을 활용해 언제까지 돌아야 하는지 생각해보고 코드를 작성해봅시다.

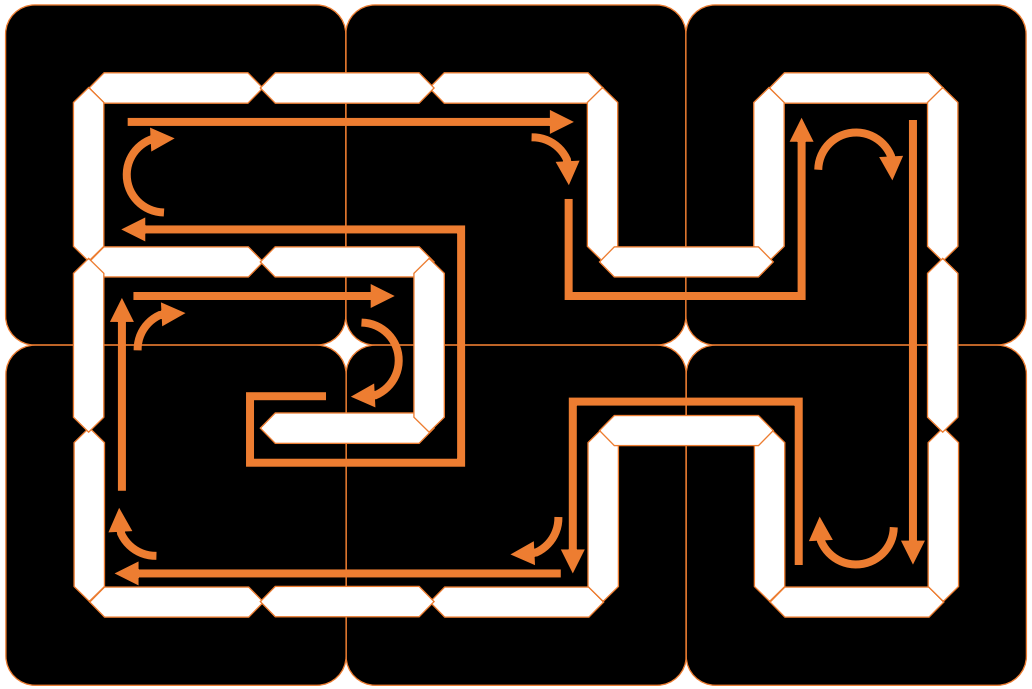
```
from roboid import *

hamster = HamsterS()

while True:
    left = hamster.left_proximity()
    right = hamster.right_proximity()

    if right > 47:
        while right > 5:
            hamster.wheels(50, -50)
            right = hamster.right_proximity()
            wait(10)
    else:
        diff = left - 35
        hamster.wheels(50 + diff * 0.8, 50 - diff * 0.8)

    wait(10)
```

01

미로 탈출

- 좌수법 (좌선법): 왼쪽 -> 앞쪽 -> 오른쪽

- 왼쪽 벽을 따라 주행
- 앞쪽에 벽이 있으면 벽이 없을 때까지 오른쪽으로 회전



01 미로 탈출

- 좌수법 (좌선법): 왼쪽 -> 앞쪽 -> 오른쪽

- 왼쪽 벽을 따라 주행합니다.
- 앞쪽에 벽이 있으면 벽이 없을 때까지 오른쪽으로 회전합니다.

- 왼쪽 반사판을 끼워주세요.

```
from roboid import *

hamster = HamsterS()

while True:
    left = hamster.left_proximity()
    right = hamster.right_proximity()

    if right > 47:
        while right > 5:
            hamster.wheels(50, -50)
            right = hamster.right_proximity()
            wait(10)
    else:
        diff = left - 35
        hamster.wheels(50 + diff * 0.8, 50 - diff * 0.8)

    wait(10)
```

01

미로 탈출

- **최단 경로 찾기 알고리즘**

- 좌수법(좌선법), 우수법(우선법)
- 깊이 우선 탐색, 너비 우선 탐색
- 다익스트라 알고리즘
- A* 알고리즘

- **미로 제작 알고리즘**

- http://weblog.jamisbuck.org/2011/2/7/maze-generation-algorithm-recap

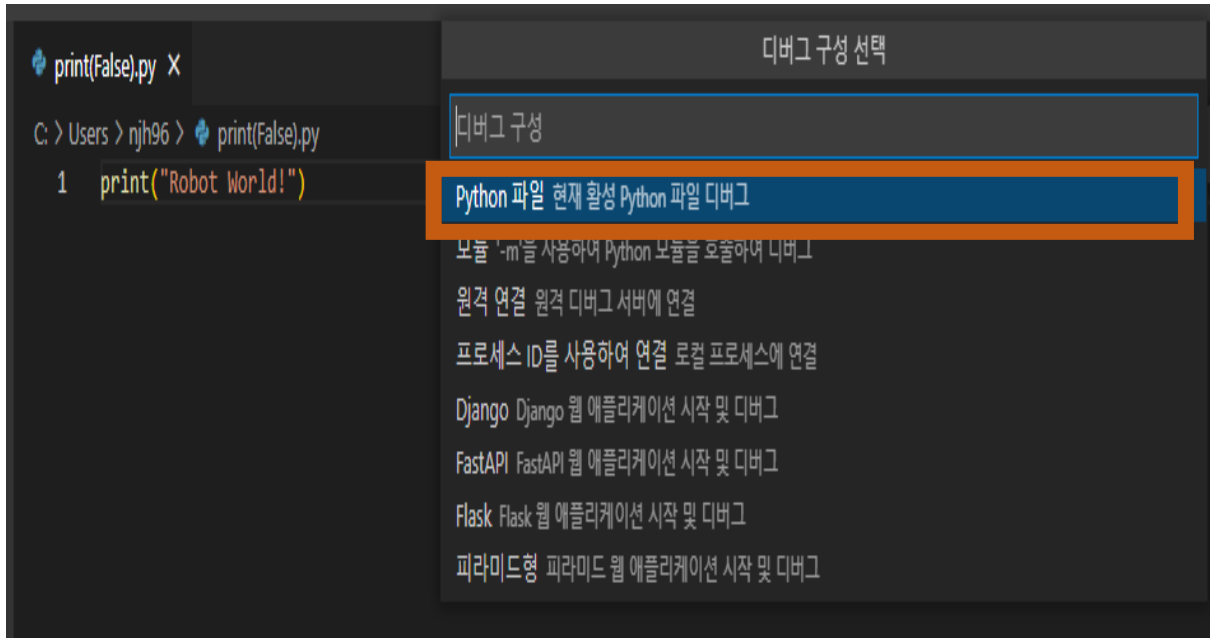
11차시

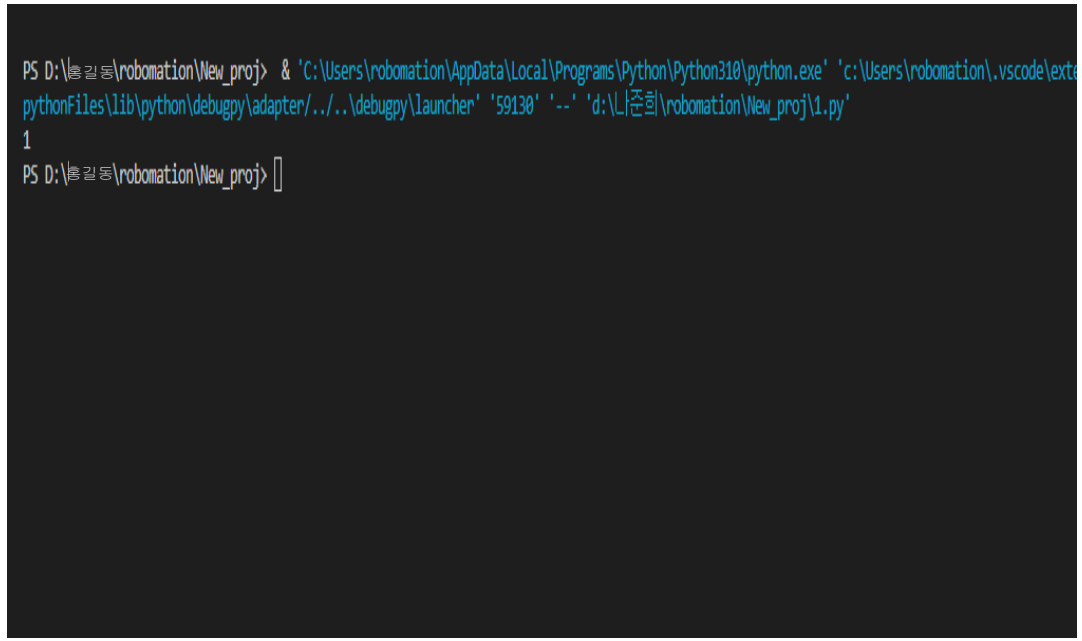
영상처리 준비

01 라이브러리 설치하기(roboid ai)

- VS code에서 New File을 열어줍니다. 예제 코드를 작성 후 F5를 눌러 실행시킨 다음 디버그 구성에서 Python파일을 선택합니다.

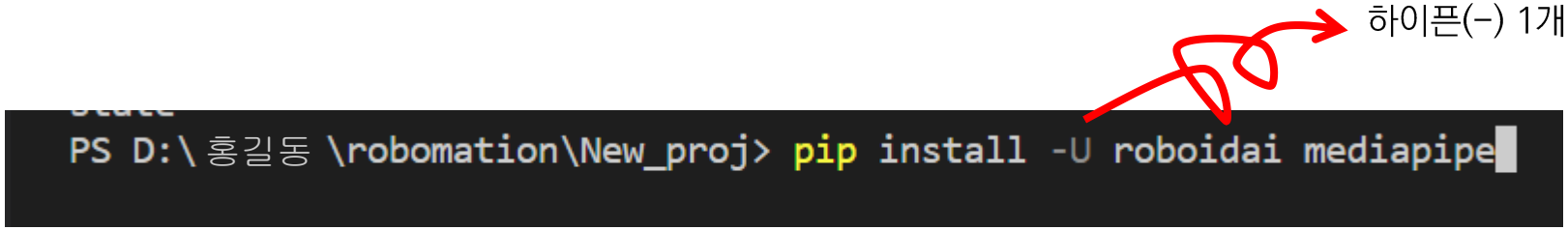
- 실행 결과, 아래에 오른쪽 그림과 같이 Terminal 창에 결과가 출력됩니다.
  (헷갈리면 1차시 라이브러리 설치 항목을 참고해주세요.)





01 라이브러리 설치하기(roboid ai)

- 터미널 창에 “pip install –U roboidai mediapipe”를 입력하여 roboidai 라이브러리와 mediapipe 라이브러리를 설치합니다.



- Roboidai와 mediapipe를 함께 설치하는 이유:
  - mediapipe는 인체를 대상으로 하는 비전 인식 기능들을 AI 모델 개발과 기계학습까지 마친 상태로 제공되는 서비스라, 다양한 프로그램언어에서 사용하기 편하게 라이브러리 형태로 모듈화로 제공되어 AI기능을 활용한 응용 프로그램개발이 가능합니다.

02

카메라 화면 불러오기

- ai.Camera를 이용하여 카메라 화면을 불러봅시다.

Camera 내부 인자:

Id: 카메라 종류

- USB 카메라➔ usb0, usb1...
- IP 카메라 ➔ ip0, ip1...

Flip: 카메라 화면 출력 형태

- 'h': 좌우 대칭
- 'v': 원점 대칭
- 'a': 상하 대칭

```
import roboidai as ai #roboidai 라이브러리를 ai라고 불러옵니다.

cam = ai.Camera('usb0') # ai의 기능 중 하나인 Camera를 불러와 내장 카메라('usb0')를 cam으로 저장합니다.

while True:
    image= cam.read()  # 이미지를 1장씩 읽습니다.

    cam.show(image)  # 읽은 이미지를 창에 띄웁니다.
    # 이미지를 1장씩 읽고 띄우지만, while문을 통해 행위를 반복하여 영상으로 띄워줍니다.
    if cam.check_key() == 'esc': break # esc키가 입력 시, 프로그램이 종료됩니다.
```

병렬처리 및 이벤트

- 아래와 같이 음악을 연주하는 동안 다른 일을 할 수 없습니다.

```
for _ in range(2):
    hamster.note(“C4”, 0.5)
    hamster.note(“E4”, 0.5)
    hamster.note(“G4”, 0.5)

for _ in range(3):
    hamster.note(“A4”,0.5)

hamster.note(“G4”,1)
hamster.note(“OFF”,0.5)
```

- 마찬가지로 아래와 같이 작성하면 햄스터의 led를 조정하는 동안 다른 일을 할 수 없습니다.

```
for _ in range(5):
    hamster.leds(“RED”, ”RED”)
    wait(500)
    hamster.leds(“OFF”, “OFF”)
    wait(500)
```

- 이 두가지 일을 동시에 하려면 어떻게 해야 할까요?
- parallel 함수를 사용하면 여러 가지 일을 동시에 할 수 있습니다.

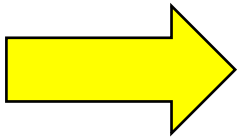
병렬처리 및 이벤트

- 병렬 처리: 연산처리의 성능을 개선하기 위해 이용

- 기본 예제

```
for _ in range(2):
    hamster.note("C4", 0.5)
    hamster.note("E4", 0.5)
    hamster.note("G4", 0.5)

for _ in range(3):
    hamster.note("A4",0.5)

hamster.note("G4",1)
hamster.note("OFF",0.5)
```

```
for _ in range(5):
    hamster.leds("RED", "RED")
    wait(500)
    hamster.leds("OFF", "OFF")
    wait(500)
```



- 병렬 처리

```
from roboid import *

hamster =  HamsterS()

def play_music():
    for _ in range(2):
        hamster.note("C4", 0.5)
        hamster.note("E4", 0.5)
        hamster.note("G4", 0.5)
    for _ in range(3):
        hamster.note("A4",0.5)
    hamster.note("G4",1)
    hamster.note(0, 0.5)


def blink():
    for _ in range(5):
        hamster.leds("RED","RED")
        wait(500)
        hamster.leds("OFF","OFF")
        wait(500)
parallel(play_music(), blink())
```

병렬처리 및 이벤트

- 햄스터 로봇의 앞을 손으로 막을 때마다 일정 시간 뒤로 이동하게 해봅시다.

```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.left_proximity() > 30 or hamster.right_proximity() > 30:
        hamster.wheels(-30) # 뒤로 이동한다.
        wait(1000) # 1초
        hamster.stop() # 정지한다.
    wait(20) # 너무 빨리 반복하지 않도록 한다.
```

- 이번에는 햄스터 로봇의 바닥 센서가 검은색 선을 만날 때마다 삐 소리를 내도록 해봅시다.

```
from roboid import *

hamster = HamsterS()

while True:
    if hamster.left_floor() < 20 or hamster.right_floor() <20:
        hamster.beep()

    wait(20) # 너무 빨리 반복하지 않도록 한다.
```

03 병렬처리 및 이벤트

- 이벤트 처리: 각 조건에 해당하는 상황이 발생하는 것을 이벤트라고 합니다. 이번에는 이벤트 방식으로 코드를 작성해 봅시다.
- 우선 각각의 상황을 판단하는 함수와 각 상황에 대한 동작을 수행하는 함수를 만듭니다. 이 때, 상황을 판단하는 함수는 True 또는 False를 반환해야 합니다.

```
from roboid import *

hamster = HamsterS()

def when_hand_found(): # 손이 감지되었을 때( 양쪽 근접 센서 값 중 1개라도 30이 넘을 때)의 이벤트
    return hamster.left_proximity()>30 or hamster.right_proximity()>30

def do_move_backward(): # 후진 후 1초 있다 정지의 이벤트
    hamster.wheels(-30)
    wait(1000)
    hamster.stop()

def when_on_black_line(): # 검은 선 위에 있을 때( 양쪽 바닥 센서 값 중 1개라도 20보다 작을 때)
    return hamster.left_floor() < 20 or hamster.right_floor() <20

def do_sound(): # 햄스터가 삡소리
    hamster.beep()

when_do(when_hand_found, do_move_backward)
when_do(when_on_black_line, do_sound)

wait(-1) # Ctrl+C를 누를 때까지 계속 기다립니다.
```

병렬처리 및 이벤트

- WHILE-DO

```
from roboid import *

hamster = HamsterS()

def while_hand_found(): # 손이 감지되었을 때( 양쪽 근접 센서 값 중 1개라도 30이 넘을 때)의 이벤트
    return hamster.left_proximity()>30 or hamster.right_proximity()>30

def do_move_backward(): # 후진 후 1초 있다 정지의 이벤트
    hamster.wheels(-30)
    wait(1000)
    hamster.stop()

def when_on_black_line(): # 검은 선 위에 있을 때( 양쪽 바닥 센서 값 중 1개라도 20보다 작을 때)
    return hamster.left_floor() < 20 or hamster.right_floor()<20

def do_sound(): # 햄스터가 삡소리
    hamster.beep()

while_do(while_hand_found, do_move_backward)
when_do(when_on_black_line, do_sound)

wait(-1) # Ctrl+C를 누를 때까지 계속 기다립니다.
```

- WAIT UNTIL

```
from roboid import *

hamster = HamsterS()

def hand_found(): # 손이 감지되었을 때( 양쪽 근접 센서 값 중 1개라도 30이 넘을 때)의 이벤트
    return hamster.left_proximity()>30 or hamster.right_proximity()>30

wait_until(hand_found)
hamster.wheels(-30)
wait(1000)
hamster.stop()
```

12차시

사물검출/정지선 지키기

영상처리 관련 클래스, 메소드 안내

- 사물 검출– ObjectDetector 클래스의 메소드

ObjectDetector(multi=False, lang='en')
사물 검출 객체를 생성합니다.
파라미터: • **multi**: **True**: 모든 사물 검출, **False**: 영역이 가장 큰 사물 하나만 검출 • **lang**: 레이블 언어, **'en'** 또는 **'ko'** 반환 값: • 사물 검출 객체

load_model(folder=None)
모델 파일을 읽습니다.
파라미터: • **folder**: object.weights 파일과 object.cfg 파일이 있는 폴더의 경로 **None**이면 c:/roboid/model 폴더로 함 반환 값: • **True**: 성공, **False**: 실패

영상처리 관련 클래스, 메소드 안내

- 사물 검출– ObjectDetector 클래스의 메소드

download_model(folder=None, overwrite=False)
모델 파일을 인터넷에서 내려 받습니다.
파라미터: • **folder**: 모델 파일을 내려 받을 폴더의 경로, 폴더가 존재하지 않으면 생성함 **None**이면 c:/roboid/model 폴더에 내려 받음 • **overwrite**: 파일이 이미 있는 경우 **True**: 덮어 씀, **False**: 내려 받지 않음

detect(image, conf_threshold=0.5, nms_threshold=0.4, gpu=False)
사물을 검출합니다.
파라미터: • **image**: 사물을 검출할 입력 영상 (nunpy.ndarray) • **conf_threshold**: 신뢰도가 **conf_threshold** 이상인 경우에만 사물로 인정 • **nms_threshold**: Non-Maximum Suppression, 중복된 박스 제거하기 위한 문턱 값 • **gpu**: **True**: GPU 사용, **False**: GPU 사용 안함 반환 값: • **True**: 사물 검출 성공, **False**: 실패

영상처리 관련 클래스, 메소드 안내

- 사물 검출– ObjectDetector 클래스의 메소드

draw_result(image, colors=None, show_conf=False)
검출된 사물을 표시합니다.
파라미터: • **image**: 입력 영상 (nunpy.ndarray) • **colors**: 각 사물의 사각형 테두리 및 레이블 색깔들, **None**: 자동 • **show_conf**: **True**: 신뢰도 표시, **False**: 표시 안함 반환 값: • 검출된 사물 영역이 표시된 영상 (numpy.ndarray)

영상처리 관련 클래스, 메소드 안내

- 사물 검출– ObjectDetector 클래스의 메소드

get_box()

검출된 사물 영역 좌표를 반환합니다.

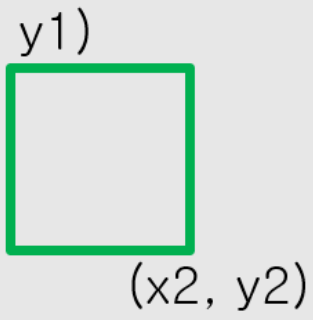
반환 값:

- 객체 생성 시 **ObjectDetector(multi=False)**로 한 경우
  - 사물 영역 좌표 리스트 [x1, y1, x2, y2] 또는 **None** (검출된 사물이 없는 경우)
- 객체 생성 시 **ObjectDetector(multi=True)**로 한 경우
  - 사물 영역 좌표 리스트의 리스트

  [[x1, y1, x2, y2],

  …

  [x1, y1, x2, y2]]



영상처리 관련 클래스, 메소드 안내

- 사물 검출– ObjectDetector 클래스의 메소드

get_label()
검출된 사물의 레이블을 반환합니다.
반환 값: • 객체 생성 시 **ObjectDetector(multi=False)**로 한 경우 - 레이블 문자열 • 객체 생성 시 **ObjectDetector(multi=True)**로 한 경우 - 레이블 문자열의 리스트

get_conf()
검출된 사물의 신뢰도를 반환합니다.
반환 값: • 객체 생성 시 **ObjectDetector(multi=False)**로 한 경우 - 신뢰도 값 0.0 ~ 1.0 • 객체 생성 시 **ObjectDetector(multi=True)**로 한 경우 - 신뢰도 값의 리스트 [0.9, 0.6, …, 0.8]

영상처리 관련 클래스, 메소드 안내

- 사물 검출- 80개의 레이블

person	bicycle	car	motorcycle	airplane	bus	train	truck	boat	traffic light
사람	자전거	자동차	오토바이	비행기	버스	기차	트럭	배	신호등

fire hydrant	stop sign	parking meter	bench	bird	cat	dog	horse	sheep	cow
소화전	정지 신호	주차 미터기	벤치	새	고양이	개	말	양	소

elephant	bear	zebra	giraffe	backpack	umbrella	handbag	tie	suitcase	frisbee
코끼리	곰	얼룩말	기린	배낭	우산	핸드백	넥타이	여행 가방	원반

skis	snowboard	sports ball	kite	baseball bat	baseball glove	skateboard	surfboard	tennis racket	bottle
스키	스노보드	공	연	야구 방망이	야구 글러브	스케이트보드	서프보드	테니스 채	병

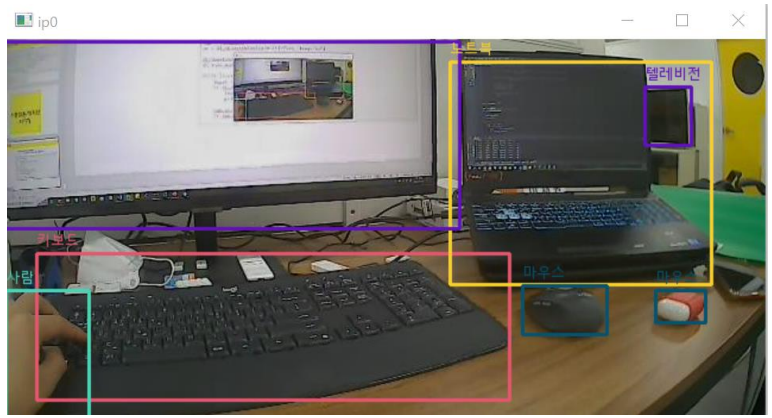
wine glass	cup	fork	knife	spoon	bowl	banana	apple	sandwich	orange
포도주 잔	컵	포크	칼	숟가락	그릇	바나나	사과	샌드위치	오렌지

broccoli	carrot	hot dog	pizza	donut	cake	chair	couch	potted plant	bed
브로콜리	당근	핫도그	피자	도넛	케이크	의자	소파	화분	침대

dining table	toilet	tv	laptop	mouse	remote	keyboard	cell phone	microwave	oven
식탁	변기	텔레비전	노트북	마우스	리모컨	키보드	휴대 전화	전자레인지	오븐

toaster	sink	refrigerator	book	clock	vase	scissors	teddy bear	hair drier	toothbrush
토스터	싱크대	냉장고	책	시계	꽃병	가위	곰 인형	헤어드라이어	칫솔

02 사물 검출 실습



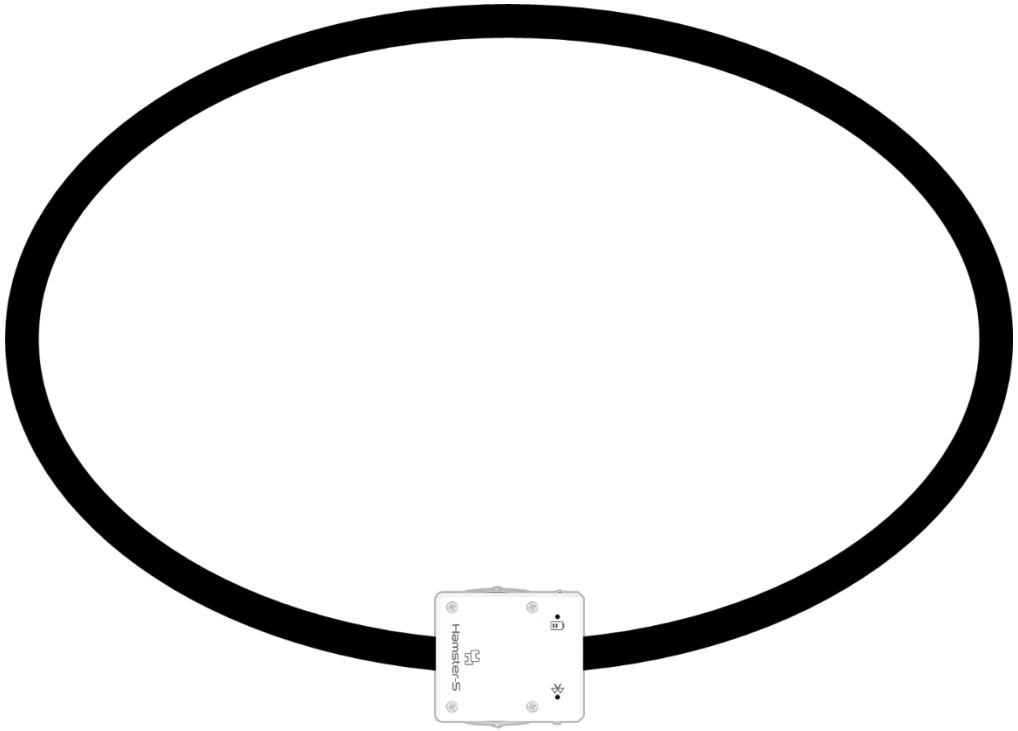

```
['마우스', '키보드', '노트북', '텔레비전', '마우스']
['노트북', '키보드', '마우스', '텔레비전', '마우스', '사람']
['키보드', '노트북', '텔레비전', '마우스', '마우스', '사람']
['키보드', '노트북', '텔레비전', '마우스', '마우스', '사람']
['키보드', '마우스', '노트북', '텔레비전', '마우스', '사람']
['키보드', '마우스', '노트북', '텔레비전', '마우스', '사람']
['키보드', '노트북', '마우스', '텔레비전', '마우스', '사람']
['키보드', '노트북', '마우스', '텔레비전', '마우스', '사람']
['키보드', '노트북', '마우스', '텔레비전', '마우스', '사람']
['키보드', '노트북', '마우스', '텔레비전', '마우스', '사람', '텔레비전']
['마우스', '노트북', '키보드', '텔레비전', '마우스', '텔레비전']
['마우스', '키보드', '노트북', '텔레비전', '마우스', '사람']
['노트북', '마우스', '키보드', '텔레비전', '마우스', '텔레비전', '사람']
['키보드', '마우스', '텔레비전', '마우스', '노트북', '사람', '마우스']
```

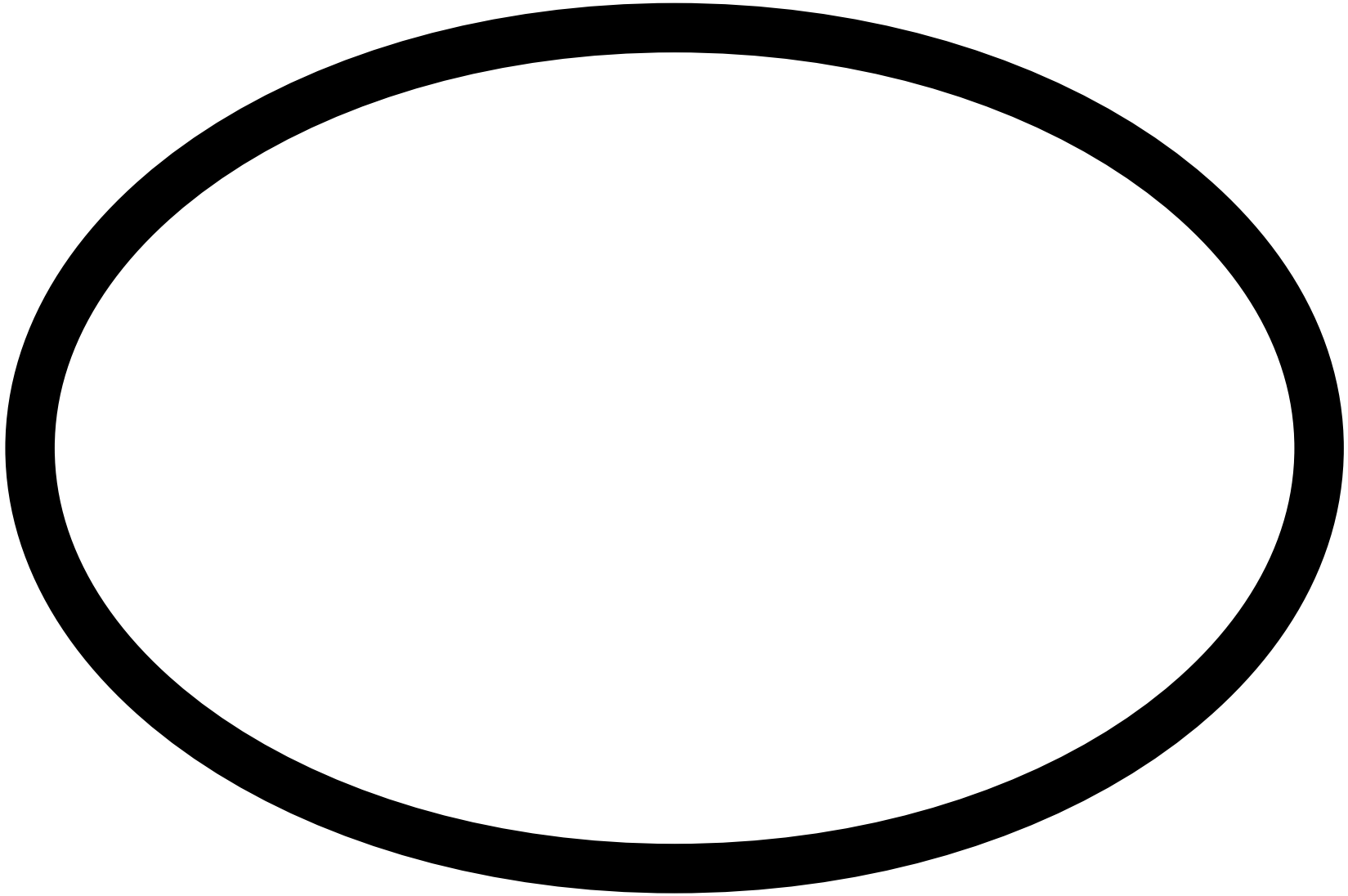
- 사물 검출 실습을 위한 코드를 작성해봅시다.
- 관련 메소드는 11차시 영상처리 관련 클래스, 메소드 안내에 있습니다.

```
import roboidai as ai

cam = ai.Camera('usb0') # USB 카메라를 사용한다(웹캠).
dt = ai.ObjectDetector(multi=Ture, lang='ko') # 모든 사물 검출하여 한국어로 사물 검출 객체 반환

dt.download_model() # 모델 파일을 인터넷에서 내려 받는다.
dt.load_model() #모델 파일을 읽는다.

while True:
    image = cam.read()
    if dt.detect(image):# 사물이 검출되면
        image = dt.draw_result(image) 검출된 사물을 image라고 저장한다.
        print(dt.get_label()) # 해당 사물의 라벨을 출력한다.

    cam.show(image)
    if cam.check_key() == 'esc': break
```

03 정지 신호 지키기

- 정지 신호 지키기를 위한 준비물이 필요합니다.






STOP

03 정지 신호 지키기

- 정지 신호 인식을 위한 코드를 작성해봅시다.
- 관련 메소드는 11차시 영상처리 관련 클래스, 메소드 안내에 있습니다.


```
from roboid import *
import roboidai as ai

hamster = HamsterS()

cam = ai.Camera('usb0') # USB 카메라를 사용한다(웹캠).
dt = ai.ObjectDetector(multi=Ture, lang='ko') # 모든 사물 검출하여 한국어로 사물 검출 객체 반환

dt.download_model() # 모델 파일을 인터넷에서 내려 받는다.
dt.load_model() #모델 파일을 읽는다.

hamster.line_both()

while True:
    image = cam.read()
    if dt.detect(image):# 사물이 검출되면
        image = dt.draw_result(image) # 검출된 사물을 image에 표시한다.
        print(dt.get_label()) # 해당 사물의 라벨을 출력한다.

        if '정지 신호' in dt.get_label():
            break

    cam.show(image)
    if cam.check_key() == 'esc': break

hamster.stop()
wait(100)
```

03 정지 신호 지키기

- 정지 신호 인식을 위한 코드를 작성해봅시다.
- 관련 메소드는 11차시 영상처리 관련 클래스, 메소드 안내에 있습니다.

```
from roboid import *
import roboidai as ai

hamster = HamsterS()

cam = ai.Camera('usb0') # USB 카메라를 사용한다(웹캠).
dt = ai.ObjectDetector(multi=Ture, lang='ko') # 모든 사물 검출하여 한국어로 사물 검출 객체 반환

dt.download_model() # 모델 파일을 인터넷에서 내려 받는다.
dt.load_model() #모델 파일을 읽는다.

hamster.line_both()

while True:
    image = cam.read()
    if dt.detect(image):# 사물이 검출되면
        image = dt.draw_result(image) 검출된 사물을 image라고 저장한다.
        print(dt.get_label()) # 해당 사물의 라벨을 출력한다.
        if '정지 신호' in dt.get_label():
            break
    cam.show(image)
    if cam.check_key() == 'esc': break

hamster.stop()
wait(100)
```

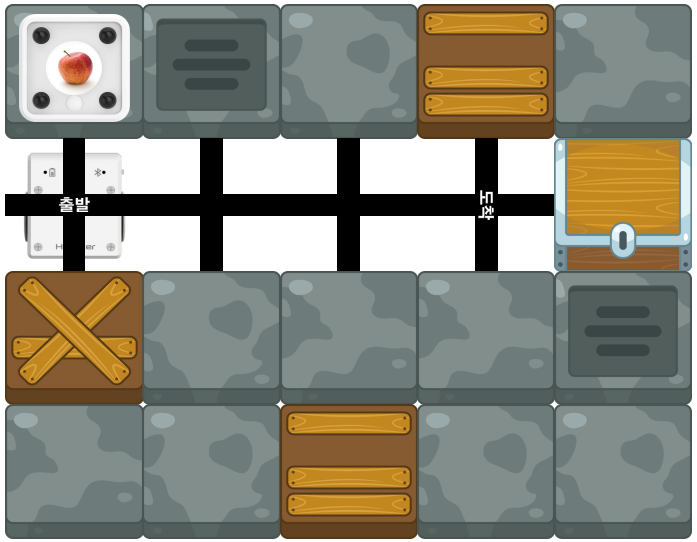
- 물체 검출 중 '정지 신호' 가 검출 된다면 break를 통해 whlie문을 탈출하게 됩니다.
- while문 이후에 햄스터를 멈추는 명령을 통해 햄스터를 멈추고 프로그램이 종료합니다.

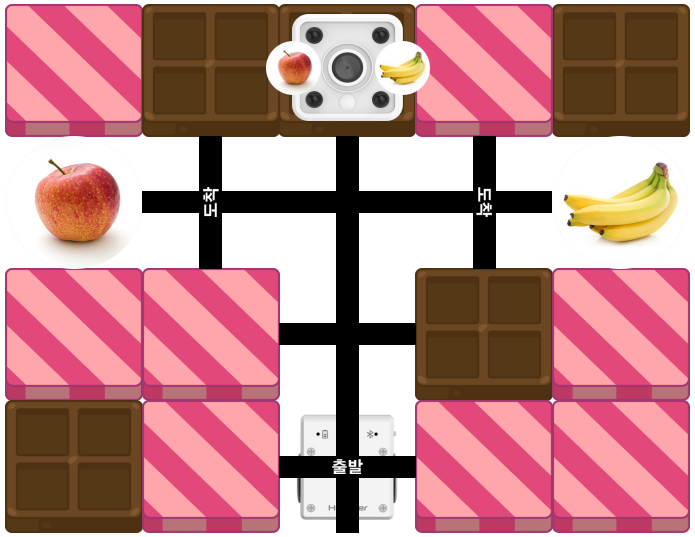
13차시

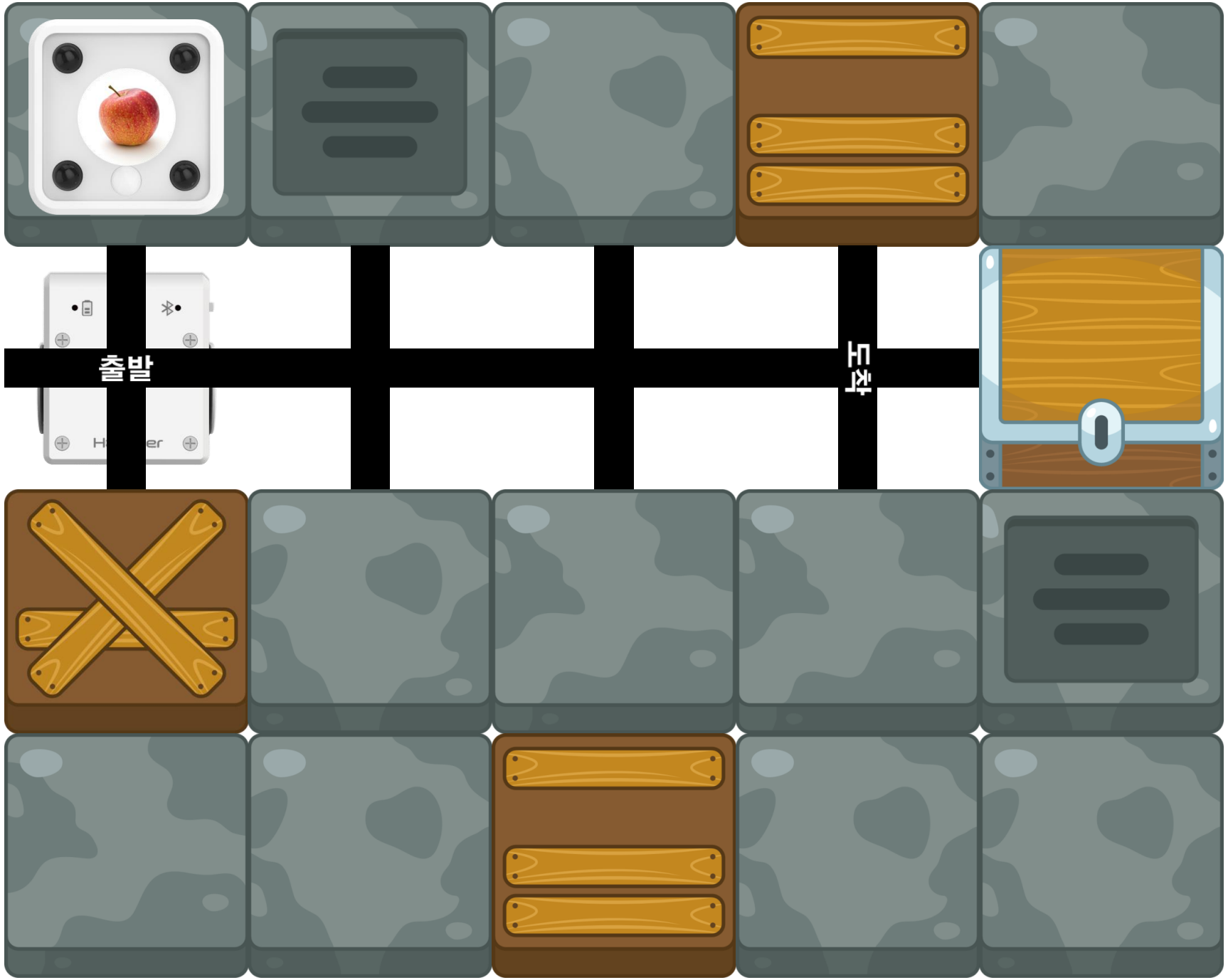
말판 이동

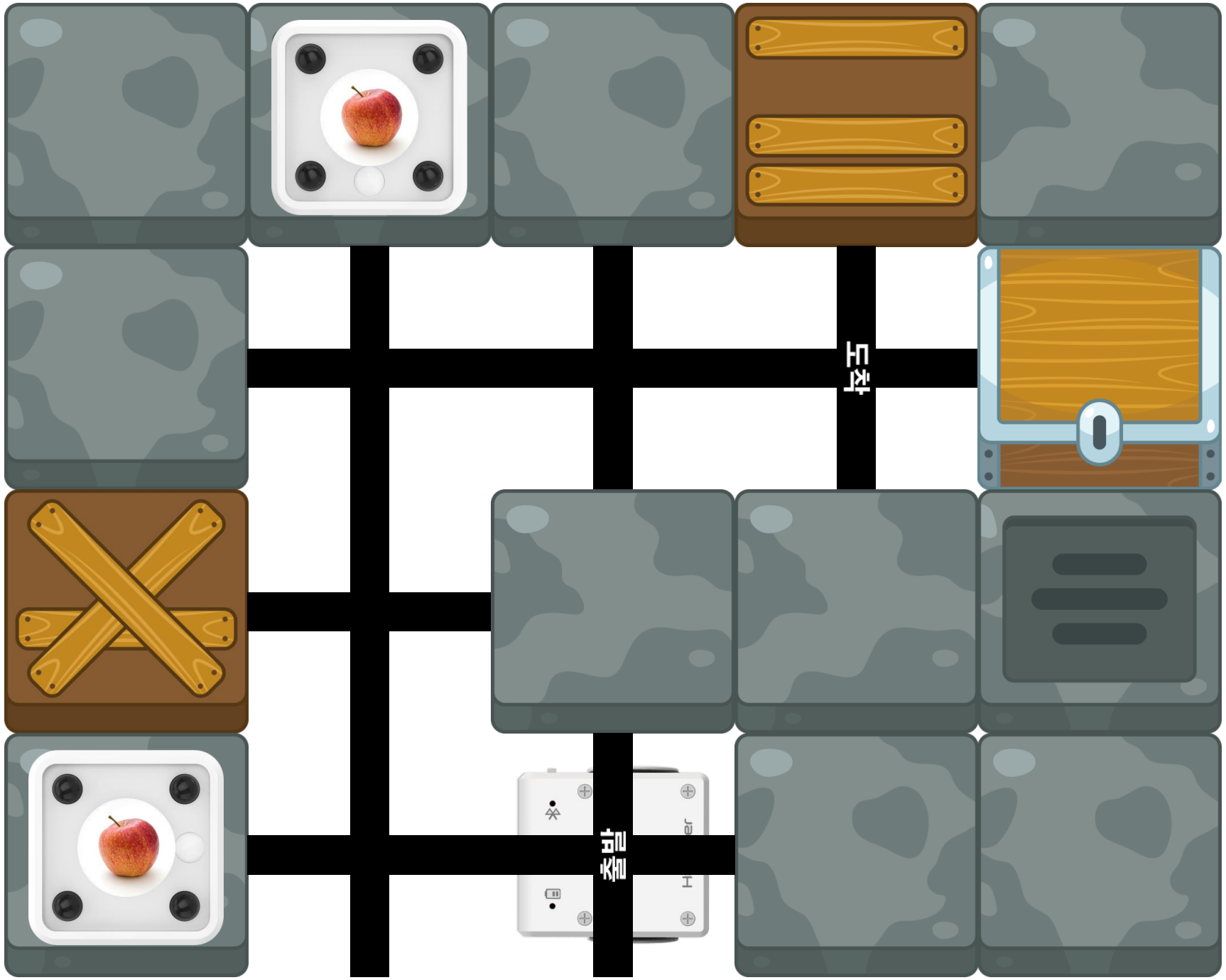
말판 이동

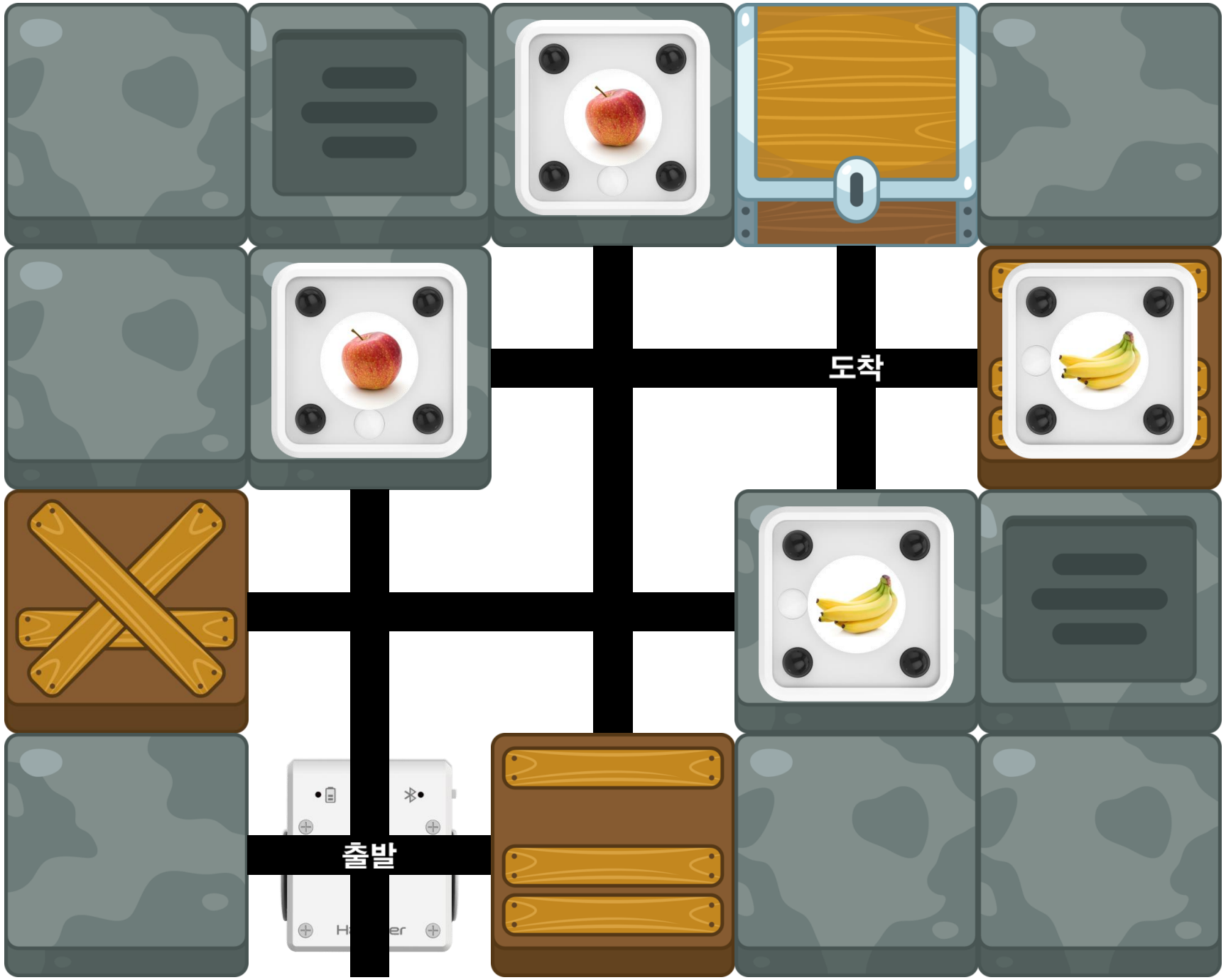
- 활동지를 준비합니다.

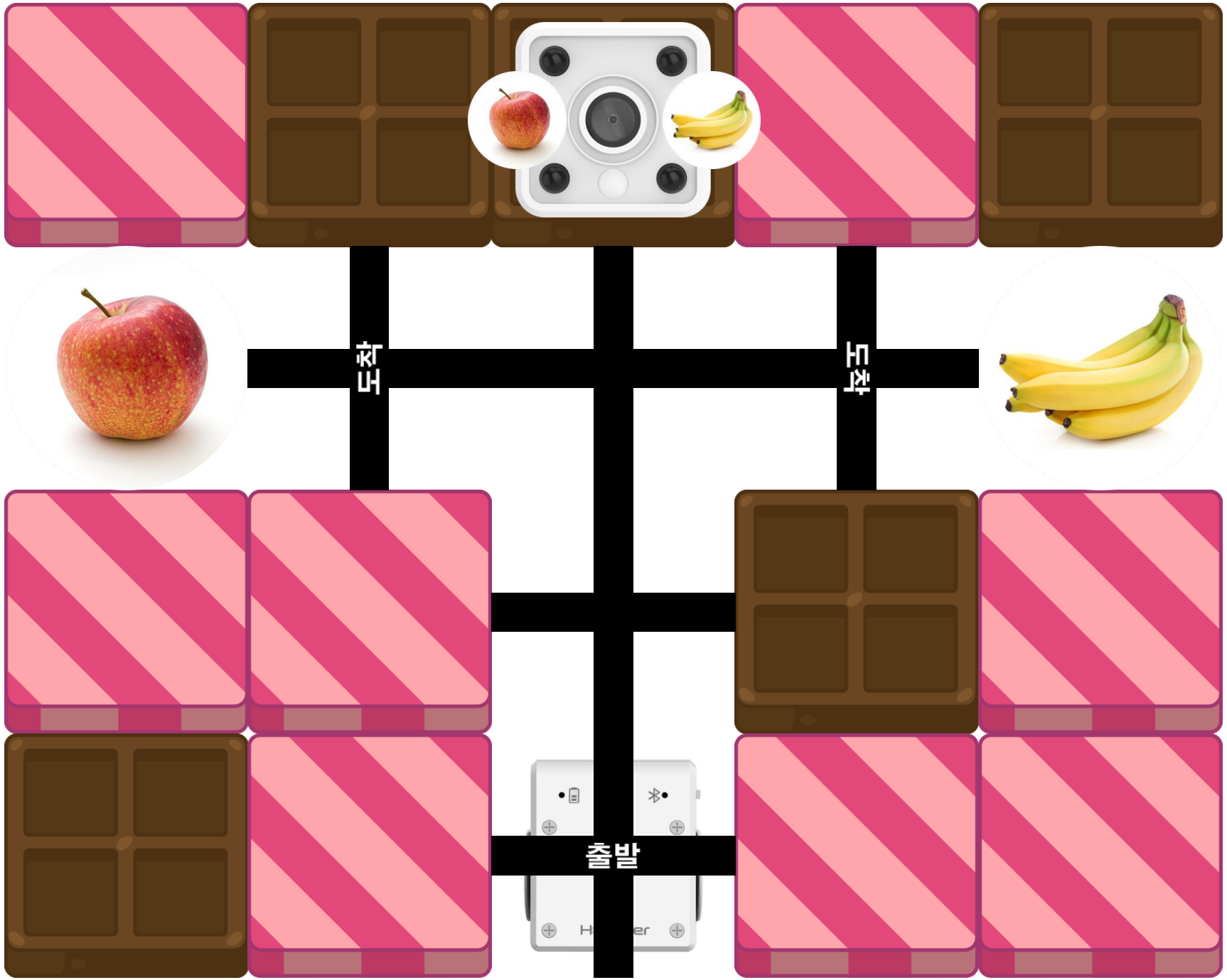


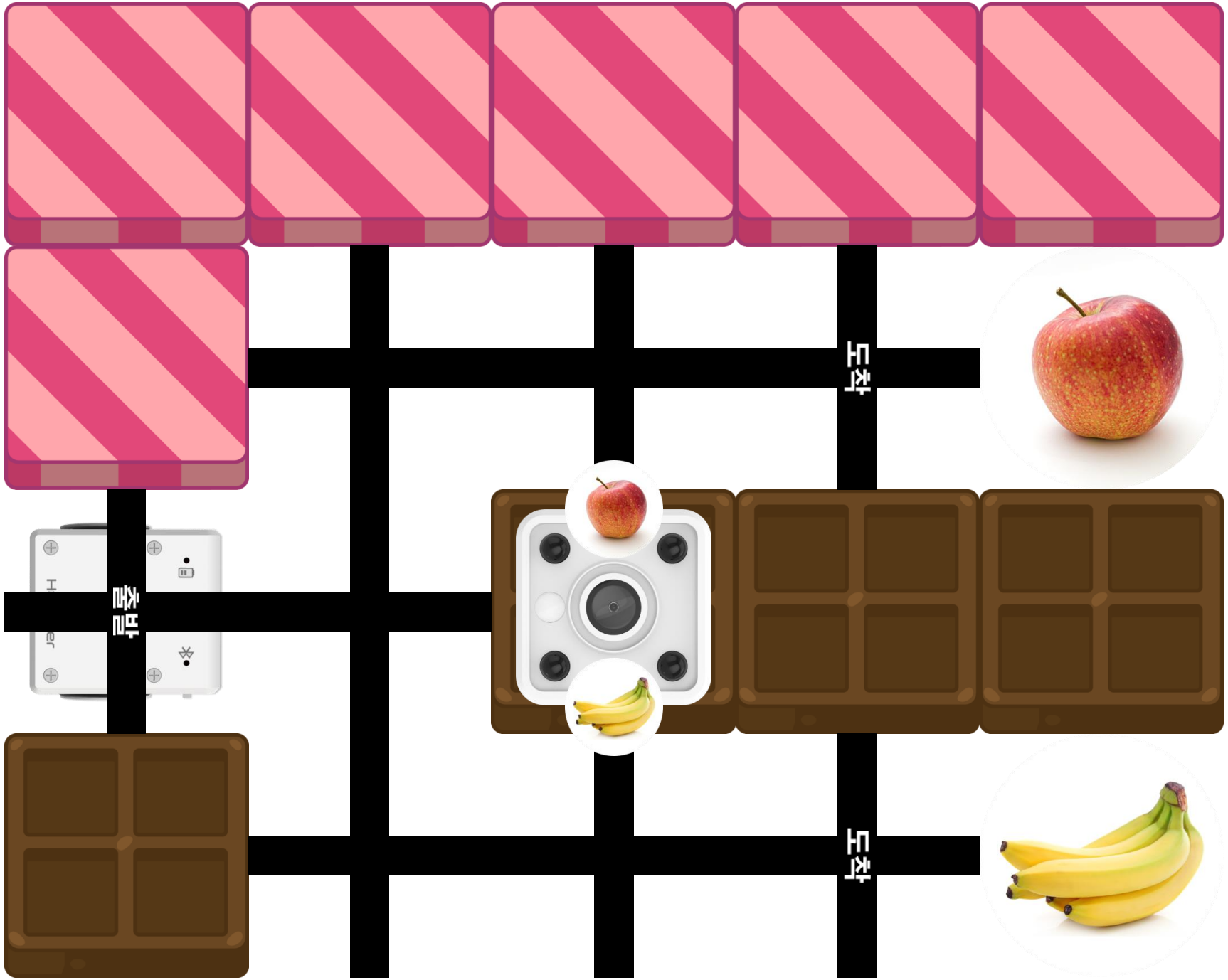



출발
도착


도착
출발

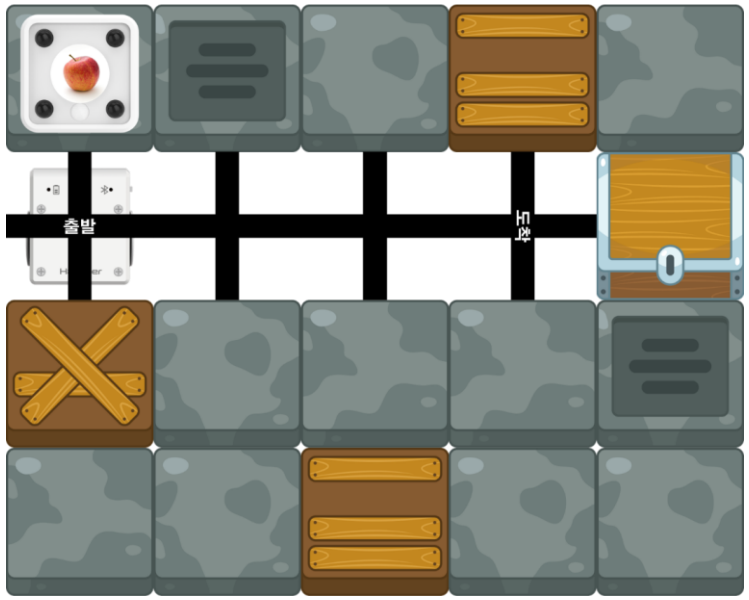

도착
출발


도착
도착
출발


도착
출발
도착

01 말판 이동

- 말판 이동 1

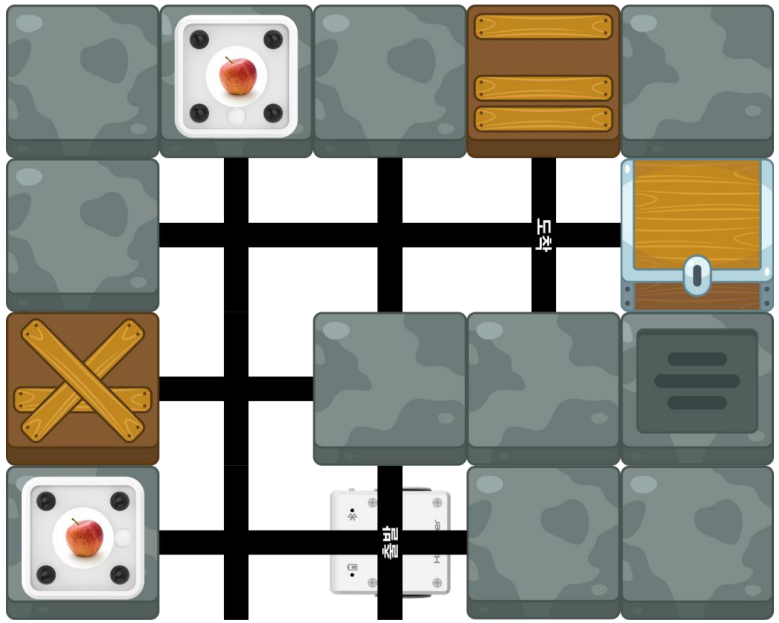


- 햄스터 로봇이 출발 위치에서 사과를 인식하면 사과를 들고 도착 위치까지 배달하고자 합니다.(그리퍼 사용 x, 사과를 인식하면 들었다고 가정) 카메라로 사과를 어떻게 인식할지 생각해 봅시다.

```
from roboid import *
import roboidai as ai

hamster = HamsterS()

cam = ai.Camera('usb0') # USB 카메라를 사용한다.
dt = ai.ObjectDetector(multi= True, lang='ko') # 모든 사물 검출하여 한국어로 사물 검출 객체 반환

dt.download_model() # 모델 파일을 인터넷에서 내려 받는다.
dt.load_model() #모델 파일을 읽는다.

while True:
    image= cam.read()
    if dt.detect(image): # 사물이 검출되면
        image= dt.draw_result(image) # 검출된 사물을 image라고 저장한다.
        print(dt.get_label()) # 해당 사물의 라벨을 출력한다.
        if '사과' in dt.get_label(): # 인식된 라벨이 사과이면
            hamster.board_right() # 오른쪽 방향으로 제자리에서 90도 회전하고
            for _ in range(3): # 전진을 3번 반복한다.
                hamster.board_forward()
            break

    cam.show(image)
    if cam.check_key() == 'esc': break # esc키를 누르면 프로그램이 종료된다.
```

01 말판 이동

- 말판 이동 2

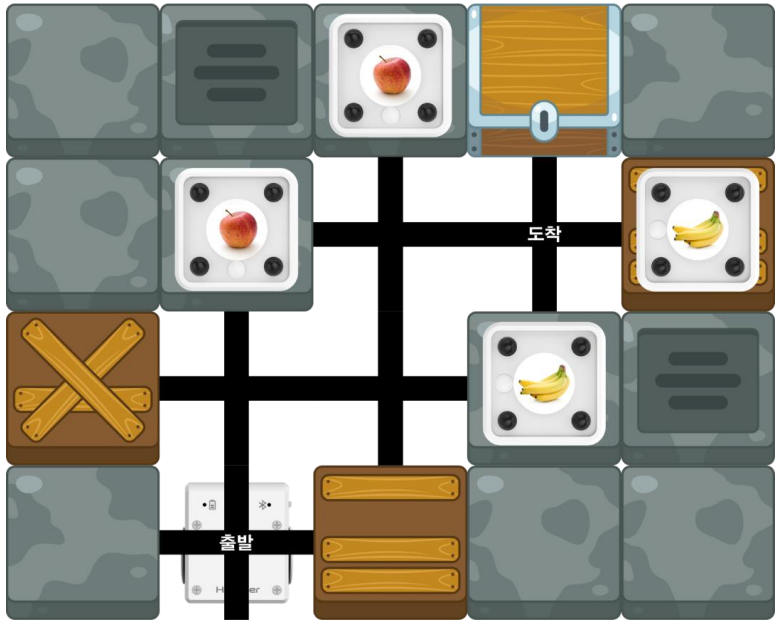


- 이번에는 사과를 인식했을 때의 동작을 2번하여 도착 위치에 가려 합니다.
- 함수를 사용해서 어떻게 구현할 지 생각해봅시다.

```
from roboid import *
import roboidai as ai

hamster = HamsterS()

cam = ai.Camera('usb0') # USB 카메라를 사용한다.
dt = ai.ObjectDetector(multi= True, lang='ko') # 모든 사물 검출하여 한국어로 사물 검출 객체 반환
dt.download_model() # 모델 파일을 인터넷에서 내려 받는다.
dt.load_model() #모델 파일을 읽는다.

def wait_until_apple(): # 사과 인식 때
    while True:
        image = cam.read()
        if dt.detect(image): # 사물이 검출되면
            image = dt.draw_result(image) # 검출된 사물을 image라고 저장한다.
            if '사과' in dt.get_label(): # 인식된 라벨이 사과이면
                break # 반복문을 종료한다.
        cam.show(image)
        if cam.check_key() == 'esc': break # esc키를 누르면 프로그램이 종료된다.

hamster.board_forward() # 앞으로 1칸 전진한다.
for _ in range(2): # for 문 안의 내용을 2번 반복한다.
    wait_until_apple()
    hamster.board_right() # 오른쪽 방향으로 제자리에서 90도 회전하고
    hamster.board_forward() # 앞으로 2칸 전진한다.
    hamster.board_forward()
```

01 말판 이동

- 말판 이동 3-1

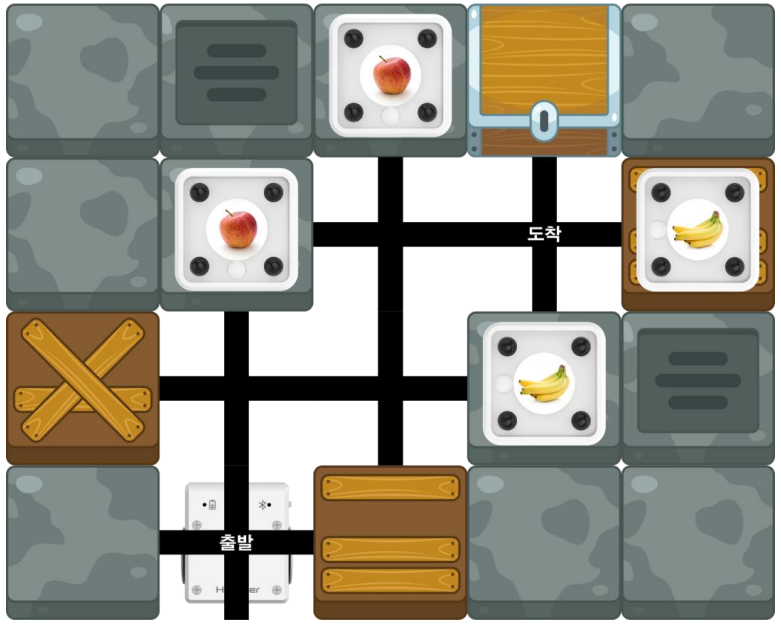


- 이번에는 사과와 바나나를 인식했을 때 각각의 동작을 행하여 도착 위치에 가려 합니다.
- 함수를 사용해서 어떻게 구현할 지 생각해봅시다.

```
from roboid import *
import roboidai as ai

hamster = HamsterS()

cam = ai.Camera('usb0') # USB 카메라를 사용한다.
dt = ai.ObjectDetector(multi= True, lang='ko') # 모든 사물 검출하여 한국어로 사물 검출 객체 반환
dt.download_model() # 모델 파일을 인터넷에서 내려 받는다.
dt.load_model() #모델 파일을 읽는다.

def wait_until_fruit(fruit): # 매개변수가 fruit인 함수 선언
    while True:
        image = cam.read()
        if dt.detect(image): # 사물이 검출되면
            image = dt.draw_result(image) # 검출된 사물을 image라고 저장한다.
            if fruit in dt.get_label(): # 인식된 라벨이  매개변수 fruit이라면
                break # 반복문을 종료한다.
        cam.show(image)
        if cam.check_key() == 'esc': break # esc키를 누르면 프로그램이 종료된다.

for _ in range(2): # for 문 안의 내용을 2번 반복한다.
    hamster.board_forward() # 앞으로 1칸 전진한다.
    wait_until_fruit('사과') # 매개변수가 사과일 때( 즉, 인식된 라벨이 사과일 때)
    hamster.board_right() # 오른쪽 방향으로 제자리에서 90도 회전한다.
    hamster.board_forward() # 앞으로 1칸 전진한다.
    wait_until_fruit('바나나') # 매개변수가 바나나일 때( 즉, 인식된 라벨이 바나나일 때)
    hamster.board_left() # 왼쪽 방향으로 제자리에서 90도 회전한다.
```

01 말판 이동

- 말판 이동 3-2

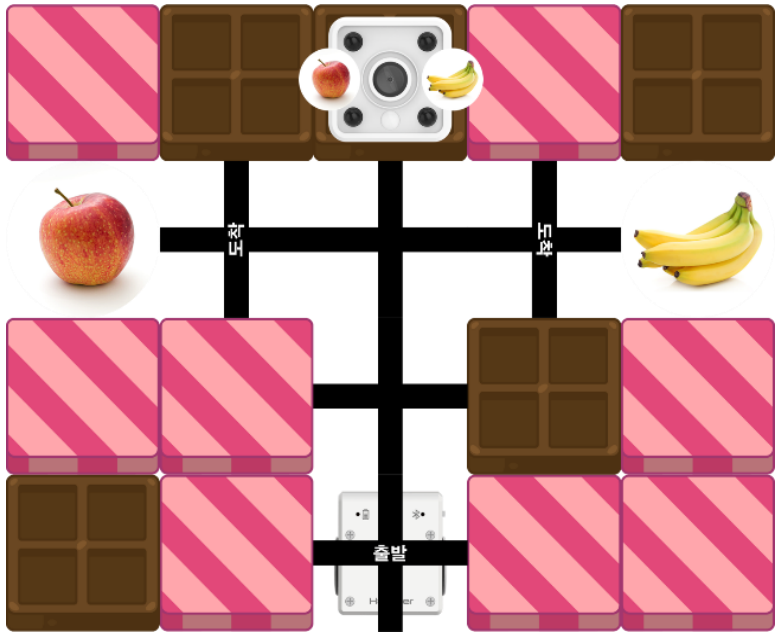


- 함수의 매개변수를 리스트 형태로 사용해서 어떻게 구현할 지 생각해봅시다.

```
from roboid import *
import roboidai as ai

hamster = HamsterS()
cam = ai.Camera('usb0') # USB 카메라를 사용한다.
dt = ai.ObjectDetector(multi= True, lang='ko') # 모든 사물 검출하여 한국어로 사물 검출 객체 반환
dt.download_model() # 모델 파일을 인터넷에서 내려 받는다.
dt.load_model() #모델 파일을 읽는다.

def wait_until_fruit(fruit): # 매개변수가 fruit인 함수 선언
    while True:
        image = cam.read()
        if dt.detect(image): # 사물이 검출되면
            image = dt.draw_result(image) # 검출된 사물을 image라고 저장한다.
            if any([x in dt.get_label() for x in fruit]):
            # 인식된 라벨이 매개변수 fruit이라면
                break # 반복문을 종료한다.
        cam.show(image)
        if cam.check_key() == 'esc': break # esc키를 누르면 프로그램이 종료된다.

for _ in range(4): # for 문 안의 내용을 4번 반복한다.
    hamster.board_forward() # 앞으로 1칸 전진한다.
    wait_until_fruit(['사과', '바나나']) # 매개변수가 사과와 바나나로 구성된 리스트일 때,
    if '사과' in dt.get_label(): # 인식된 라벨이 사과라면
        hamster.board_right() # 오른쪽 90도 제자리 회전
    else: # 사과가 아니라면 (즉, 인식된 라벨이 바나나라면)
        hamster.board_left() # 왼쪽 90도 제자리 회전
hamster.stop() # 정지
wait(100)
```

01 말판 이동

- 말판 이동 4

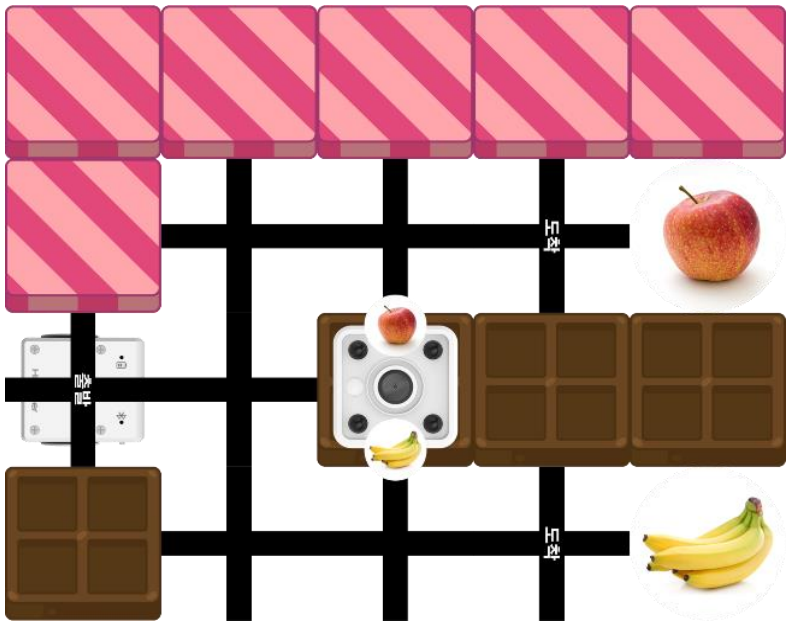


- 사과와 바나나 중 먼저 인식되는 물체에 따라 주어진 행동을 해봅시다.

```
from roboid import *
import roboidai as ai

hamster = HamsterS()
cam = ai.Camera('usb0') # USB 카메라를 사용한다.
dt = ai.ObjectDetector(multi= True, lang='ko') # 모든 사물 검출하여 한국어로 사물 검출 객체 반환
dt.download_model() # 모델 파일을 인터넷에서 내려 받는다.
dt.load_model() #모델 파일을 읽는다.

def wait_until_fruit(fruit): # 매개변수가 fruit인 함수 선언
    while True:
        image = cam.read()
        if dt.detect(image): # 사물이 검출되면
            image = dt.draw_result(image) # 검출된 사물을 image라고 저장한다.
            if any([x in dt.get_label() for x in fruit]):
            # 인식된 라벨이 매개변수 fruit이라면
                break # 반복문을 종료한다.
        cam.show(image)
        if cam.check_key() == 'esc': break # esc키를 누르면 프로그램이 종료된다.
hamster.board_forward()
hamster.board_forward() # 앞으로 2칸 전진한다.
wait_until_fruit(['사과', '바나나']) # 매개변수가 사과와 바나나로 구성된 리스트일 때,
if '사과' in dt.get_label(): # 인식된 라벨이 사과라면
    hamster.board_right() # 오른쪽 90도 제자리 회전
else: # 사과가 아니라면 (즉, 인식된 라벨이 바나나라면)
    hamster.board_left() # 왼쪽 90도 제자리 회전
hamster.board_forward() # 앞으로 1칸 전진한다.
hamster.stop() # 정지
wait(100)
```

01 말판 이동

- 말판 이동 5



```
from roboid import *
import roboidai as ai

hamster = HamsterS()
cam = ai.Camera('usb0') # USB 카메라를 사용한다.
dt = ai.ObjectDetector(multi= True, lang='ko') # 모든 사물 검출하여 한국어로 사물 검출 객체 반환
dt.download_model() # 모델 파일을 인터넷에서 내려 받는다.
dt.load_model() #모델 파일을 읽는다.
def wait_until_fruit(fruit): # 매개변수가 fruit인 함수 선언
    while True:
        image = cam.read()
        if dt.detect(image): # 사물이 검출되면
            image = dt.draw_result(image) # 검출된 사물을 image라고 저장한다.
            if any([x in dt.get_label() for x in fruit]):
                break # 반복문을 종료한다.
        cam.show(image)
        if cam.check_key() == 'esc': break # esc키를 누르면 프로그램이 종료된다.
hamster.board_forward() # 앞으로 2칸 전진한다.
wait_until_fruit(['사과', 바나나']) # 매개변수가 사과와 바나나로 구성된 리스트일 때,
if '사과' in dt.get_label(): # 인식된 라벨이 사과라면
    hamster.board_left() # 왼쪽 90도 제자리 회전
    hamster.board_forward()
    hamster.board_right() # 오른쪽 90도 제자리 회전
else: # 사과가 아니라면 (즉, 인식된 라벨이 바나나라면)
    hamster.board_right()
    hamster.board_forward()
    hamster.board_left() # 왼쪽 90도 제자리 회전
hamster.board_forward() # 앞으로 1칸 전진한다.
hamster.board_forward()
hamster.stop() # 정지
wait(100)
```

14차시

얼굴 검출 /나이,성별 ,얼굴 표정

영상처리 관련 클래스, 메소드 안내

- 얼굴 검출– FaceDetector 클래스의 메소드

FaceDetector(threshold=0.5)
얼굴 검출 객체를 생성합니다.
파라미터: • **threshold**: 신뢰도가 **threshold** 이상인 경우에만 얼굴로 인정 **반환 값**: • 얼굴 검출 객체

detect(image, padding=0)
얼굴을 검출합니다.
파라미터: • **image**: 얼굴을 검출할 입력 영상 (nunpy.ndarray) • **padding**: 검출한 얼굴 영역을 왼쪽, 오른쪽, 위, 아래 **padding**만큼 영역을 넓힘 **반환 값**: • **True**: 얼굴 검출 성공, **False**: 실패

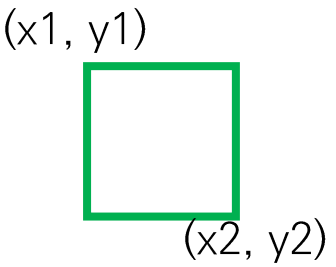
영상처리 관련 클래스, 메소드 안내

- 얼굴 검출– FaceDetector 클래스의 메소드

draw_result(image, color=(0,255,0), thickness=2, clone=False)
검출된 얼굴 영역을 표시합니다.

파라미터:
• **image**: 입력 영상 (nunpy.ndarray)
• **color**: 얼굴 영역 사각형 테두리 색깔, BGR 순서
• **thickness**: 얼굴 영역 사각형 테두리 두께 (픽셀)
• **clone**:
  - **True**: 입력 영상을 복제한 영상에 얼굴 영역 표시 (원본 유지)
  - **False**: 복제 안 함, 입력 영상에 얼굴 영역 표시
반환 값:
• 얼굴 영역이 표시된 영상 (numpy.ndarray)

get_box()
검출된 얼굴 영역 사각형의 좌표를 반환합니다.
반환 값: • 얼굴 영역 좌표 리스트 [x1, y1, x2, y2] 또는 **None** (검출된 얼굴이 없는 경우)



영상처리 관련 클래스, 메소드 안내

- 얼굴 검출– FaceDetector 클래스의 메소드

get_conf()
검출된 얼굴 영역의 신뢰도를 반환합니다.
반환 값: • 신뢰도 값 0.0 ~ 1.0

get_width()
검출된 얼굴 영역 사각형의 폭을 반환합니다.
반환 값: • 검출된 얼굴 영역 사각형의 폭

get_height()
검출된 얼굴 영역 사각형의 높이를 반환합니다.
반환 값: • 검출된 얼굴 영역 사각형의 높이

get_area()
검출된 얼굴 영역 사각형의 넓이를 반환합니다.
반환 값: • 검출된 얼굴 영역 사각형의 넓이

영상처리 관련 클래스, 메소드 안내

- 얼굴 검출– FaceDetector 클래스의 메소드

get_xy(id='all')
검출된 얼굴에서 각 요소의 좌표를 반환합니다.
파라미터: • **id**: **'all'**, **'left eye'**, **'right eye'**, **'nose'**, **'mouth'**, **'left ear'**, **'right ear'** **반환 값**: • **id**가 **'all'**: 모든 요소의 좌표 리스트 (numpy.ndarray) • 그 외: 해당 얼굴 요소의 좌표 (numpy.ndarray)

crop(image, clone=False)
image에서 검출된 얼굴 영역을 잘라내어 반환합니다.
파라미터: • **image**: 얼굴 영역을 잘라 낼 입력 영상 (nunpy.ndarray) • **clone**: - **True**: 입력 영상을 복제한 후 얼굴 영역을 잘라 냄 (원본 유지) - **False**: 복제 안 함 **반환 값**: • 잘라 낸 얼굴 영상 (numpy.ndarray) 또는 **None** (검출된 얼굴이 없는 경우)

영상처리 관련 클래스, 메소드 안내

- 나이,성별,표정 검출

AgeGenderDetector()
객체를 생성합니다.
반환 값: • 나이·성별 검출 객체

load_age_model(folder=None)
나이에 대한 모델 파일을 읽습니다.
파라미터: • **folder**: age.caffemodel 파일과 age.prototxt 파일이 있는 폴더의 경로 **None**이면 c:/roboid/model 폴더로 함 **반환 값**: • **True**: 성공, **False**: 실패

영상처리 관련 클래스, 메소드 안내

- 나이,성별,표정 검출

load_gender_model(folder=None)
성별에 대한 모델 파일을 읽습니다.
파라미터: • **folder**: gender.caffemodel 파일과 gender.prototxt 파일이 있는 폴더의 경로 **None**이면 c:/roboid/model 폴더로 함 **반환 값**: • **True**: 성공, **False**: 실패

download_age_model(folder=None, overwrite=False)
나이에 대한 모델 파일을 인터넷에서 내려 받습니다.
파라미터: • **folder**: 모델 파일을 내려 받을 폴더의 경로, 폴더가 존재하지 않으면 생성함 **None**이면 c:/roboid/model 폴더에 내려 받음 • **overwrite**: 파일이 이미 있는 경우 **True**: 덮어 씀, **False**: 내려 받지 않음

영상처리 관련 클래스, 메소드 안내

- 나이,성별,표정 검출

download_gender_model(folder=None, overwrite=False)
성별에 대한 모델 파일을 인터넷에서 내려 받습니다.
파라미터: • **folder**: 모델 파일을 내려 받을 폴더의 경로, 폴더가 존재하지 않으면 생성함 **None**이면 c:/roboid/model 폴더에 내려 받음 • **overwrite**: 파일이 이미 있는 경우 **True**: 덮어 씀, **False**: 내려 받지 않음

detect(image, gpu=False)
나이와 성별을 검출합니다.
파라미터: • **image**: 입력 영상 (nunpy.ndarray) • **gpu**: **True**: GPU 사용, **False**: GPU 사용 안함 **반환 값**: • **True**: 성공, **False**: 실패

영상처리 관련 클래스, 메소드 안내

- 나이,성별,표정 검출

draw_result(image, color=(0,255,0), thickness=2, clone=False)

얼굴 영역과 결과 레이블을 표시합니다.

파라미터:
- **image**: 입력 영상 (nunpy.ndarray)
- **color**: 얼굴 영역 사각형 테두리 및 레이블 색깔, BGR 순서
- **thickness**: 얼굴 영역 사각형 테두리 두께 (픽셀)
- **clone**:
  - **True**: 입력 영상을 복제한 영상에 결과 표시 (원본 유지)
  - **False**: 복제 안 함, 입력 영상에 결과 표시

반환 값:
- 결과가 표시된 영상 (numpy.ndarray)

영상처리 관련 클래스, 메소드 안내

- 나이,성별,표정 검출

get_age_index()
검출된 나이의 인덱스를 반환합니다.
반환 값: • 검출된 나이의 인덱스, 검출 실패 시 -1 ['0-2', '4-6', '8-12', '15-20', '25-32', '38-43', '48-53', '60-100']에 대한 인덱스

get_age_label()
검출된 나이의 레이블을 반환합니다.
반환 값: • 검출된 나이의 레이블, 검출 실패 시 빈 문자열 '' '0-2', '4-6', '8-12', '15-20', '25-32', '38-43', '48-53', '60-100' 중에서 하나

get_age_conf()
검출된 나이의 신뢰도를 반환합니다.
반환 값: • 검출된 나이의 신뢰도 0.0 ~ 1.0

영상처리 관련 클래스, 메소드 안내

- 나이,성별,표정 검출

get_gender_index()
검출된 성별의 인덱스를 반환합니다.
반환 값: • 검출된 성별의 인덱스, 검출 실패 시 -1 ['male', 'female']에 대한 인덱스

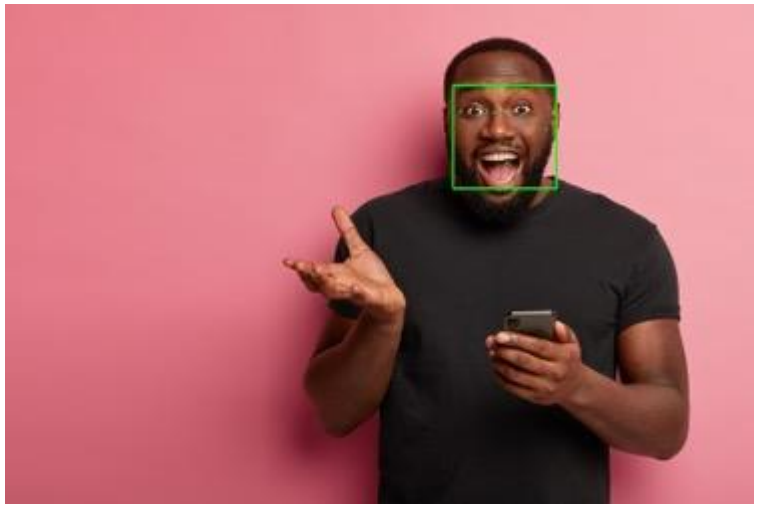
get_gender_label()
검출된 성별의 레이블을 반환합니다.
반환 값: • 검출된 성별의 레이블, 검출 실패 시 빈 문자열 '' 'male', 'female' 중에서 하나

get_age_conf()
검출된 성별의 신뢰도를 반환합니다.
반환 값: • 검출된 성별의 신뢰도 0.0 ~ 1.0

01

얼굴 검출 실습

- 얼굴 검출 실습을 위한 코드를 작성해봅시다.
- 관련 메소드는 11차시 영상처리 관련 클래스, 메소드 안내에 있습니다.



```
import roboidai as ai

cam = ai.Camera('usb0') # USB 카메라를 사용한다(웹캠).
dt = ai.FaceDetector() # 얼굴 검출 클래스 객체 생성

while True:
    image = cam.read()
    if dt.detect(image):# 사물이 검출되면
        image = dt.draw_result(image) # 검출된 사물을 image에 표시한다.

    cam.show(image)
    if cam.check_key() == 'esc': break
```

01 얼굴 검출 실습

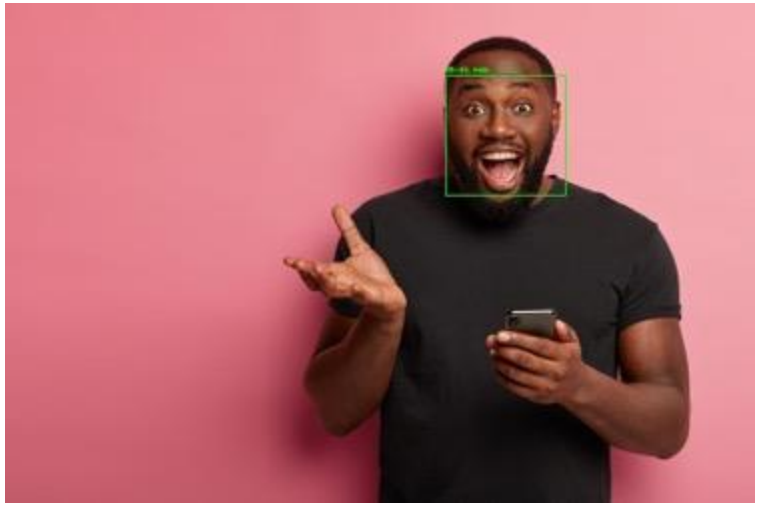
- 얼굴 검출 실습을 위한 코드를 작성해봅시다.
- 관련 메소드는 11차시 영상처리 관련 클래스, 메소드 안내에 있습니다.

- padding을 사용해보고, 신뢰도를 출력해봅시다.
- padding을 사용하면 검출한 얼굴 영역을 왼쪽, 오른쪽, 위, 아래 padding만큼 영역을 넓힙니다.

```
import roboidai as ai

cam = ai.Camera('usb0', flip='h') # USB 카메라를 사용한다(웹캠), 수평 반전
dt = ai.FaceDetector() # 얼굴 검출 클래스 객체 생성

while True:
    image = cam.read()
    if dt.detect(image, padding=20):# 얼굴이 검출되면
        print(dt.get_box(), dt.get_conf()) # 검출된 얼굴 영역 사각형의 좌표와 신뢰도를 출력한다.
        image = dt.crop(image) # image에서 검출된 얼굴 영역을 잘라내어 반환한다.

    cam.show(image)
    if cam.check_key() == 'esc': break
```

02 나이, 성별 검출

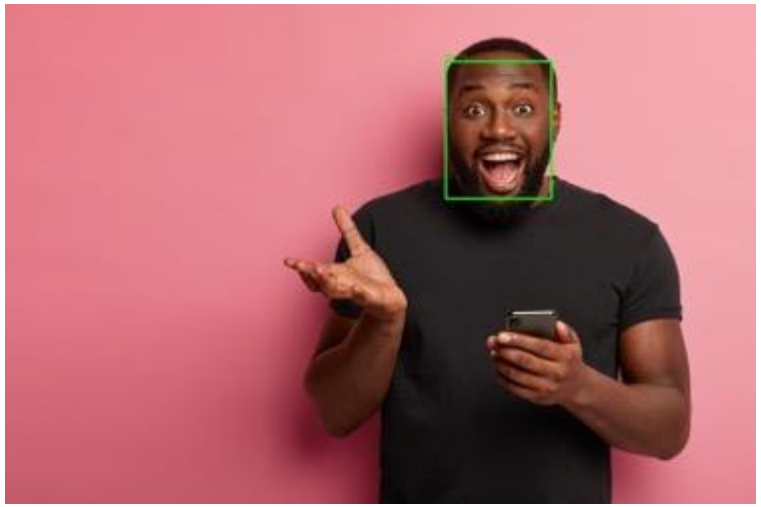
- 나이, 성별 검출 실습을 위한 코드를 작성해봅시다.
- 관련 메소드는 11차시 영상처리 관련 클래스, 메소드 안내에 있습니다.



```
import roboidai as ai

cam = ai.Camera('usb0', flip='h') # USB 카메라를 사용한다(웹캠), 수평 반전
dt = ai.AgeGenderDetector() # 나이, 성별 검출 클래스 객체 생성

dt.download_age_model() # 모델 파일을 인터넷에서 내려 받는다.
dt.download_gender_model()

dt.load_age_model() #모델 파일을 읽는다.
dt.load_gender_model()

while True:
    image = cam.read()
    if dt.detect(image):# 인식 성공 시
        image = dt.draw_result(image) # 검출된 사물을 image에 표시한다.

    cam.show(image)
    if cam.check_key() == 'esc': break
```

03 얼굴 표정 검출

- 얼굴 표정 검출 실습을 위한 코드를 작성해봅시다.
- 관련 메소드는 11차시 영상처리 관련 클래스, 메소드 안내에 있습니다.



```
import roboidai as ai

cam = ai.Camera('usb0', flip='h') # USB 카메라를 사용한다(웹캠), 수평 반전
dt = ai.FacialExpression('ko') # 얼굴 표정 검출 클래스 객체 생성

dt.load_model() #모델 파일을 읽는다.

while True:
    image = cam.read()
    if dt.detect(image):# 인식 성공 시
        image = dt.draw_result(image, show_conf=True) # 결과와 신뢰도를 image에 표시한다.

    cam.show(image)
    if cam.check_key() == 'esc': break
```

15차시

손으로 운전하기

영상처리 관련 클래스, 메소드 안내

- 손 검출– HandPose 클래스의 메소드

`HandPose()`
손 검출 객체를 생성합니다.
반환 값: • 손 검출 객체

`load_model(both_hands=False, threshold=0.5)`
모델을 불러옵니다.
파라미터: • `both_hands`: `True`이면 양손, `False`이면 한 손 • `threshold`: 신뢰도가 `threshold` 이상인 경우에만 손으로 인정 **반환 값**: • `True`: 성공, `False`: 실패

영상처리 관련 클래스, 메소드 안내

- 손 검출– HandPose 클래스의 메소드

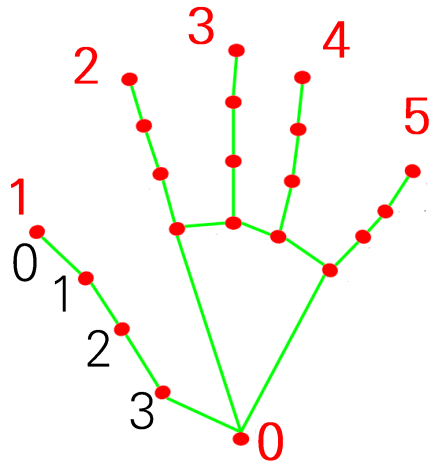
detect(image)
손을 검출합니다.
파라미터: • **image**: 손을 검출할 입력 영상 (nunpy.ndarray) **반환 값**: • **True**: 손 검출 성공, **False**: 실패

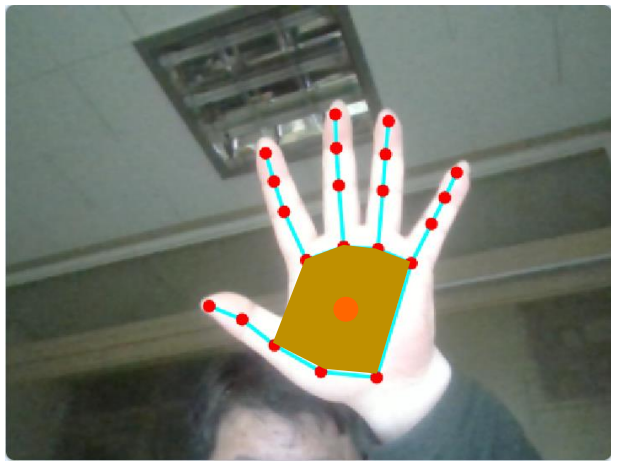
draw_result(image, clone=False)
검출된 손을 표시합니다.
파라미터: • **image**: 입력 영상 (nunpy.ndarray) • **clone**: - **True**: 입력 영상을 복제한 영상에 결과 표시 (원본 유지) - **False**: 복제 안 함, 입력 영상에 결과 표시 **반환 값**: • 검출된 손 결과가 표시된 영상 (numpy.ndarray)

영상처리 관련 클래스, 메소드 안내

- 손 검출– HandPose 클래스의 메소드

`get_xy(hand, id='all', index=0)`
손의 각 랜드마크의 x, y 좌표를 반환합니다.
파라미터: • **hand**: `'left'` 또는 `'right'` • **id**: – `0`: 손목, `1`: 엄지, `2`: 검지, `3`: 중지, `4`: 약지, `5`: 새끼 – `'all'`: 모든 랜드마크, `'hand'`: 손, `'palm'`: 손바닥 • **index**: 손가락의 경우 마디 번호, 위에서 아래로 `0, 1, 2, 3` **반환 값**: • `[x, y]`의 리스트 • id가 `'all'`인 경우에는 `[[x, y], [x, y], ..., [x, y]]`

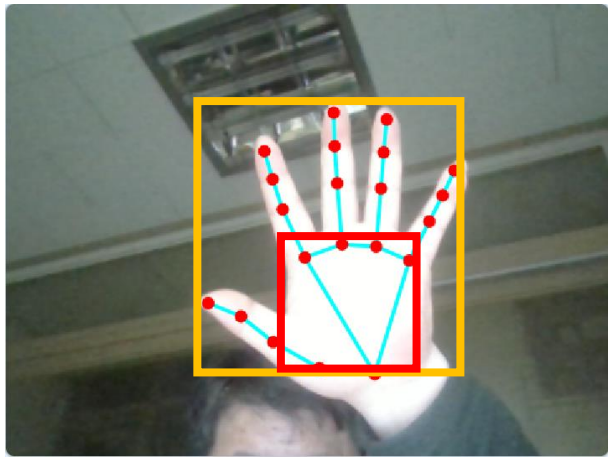




영상처리 관련 클래스, 메소드 안내

- 손 검출– HandPose 클래스의 메소드

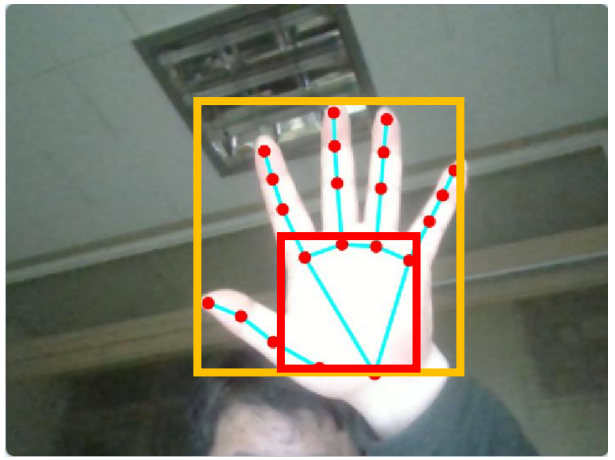
`get_box(hand, id='all')`
손 또는 손바닥의 사각형 영역을 반환합니다.
파라미터: • `hand`: `'left'` 또는 `'right'` • `id`: `'all'`: 모든 영역, `'hand'`: 손, `'palm'`: 손바닥 **반환 값**: • `[x1, y1, x2, y2]`의 리스트 • `id`가 `'all'`인 경우에는 `[[x1,y1,x2,y2], ..., [x1,y1,x2,y2]]`



영상처리 관련 클래스, 메소드 안내

- 손 검출– HandPose 클래스의 메소드

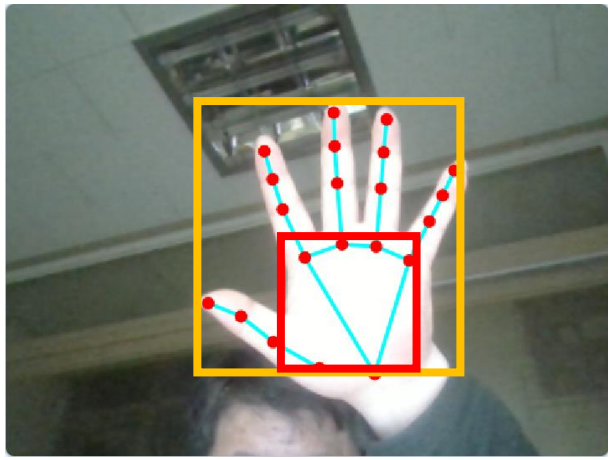
`get_width(hand, id='all')`
손 또는 손바닥의 사각형 영역의 폭을 반환합니다.
파라미터: • **hand**: `'left'` 또는 `'right'` • **id**: `'all'`: 모든 영역, `'hand'`: 손, `'palm'`: 손바닥 **반환 값**: • 폭 값 • id가 `'all'`인 경우에는 `[폭, …, 폭]`



영상처리 관련 클래스, 메소드 안내

- 손 검출– HandPose 클래스의 메소드

`get_height(hand, id='all')`
손 또는 손바닥의 사각형 영역의 높이를 반환합니다.
파라미터: • **hand**: `'left'` 또는 `'right'` • **id**: `'all'`: 모든 영역, `'hand'`: 손, `'palm'`: 손바닥 **반환 값**: • 높이 값 • id가 `'all'`인 경우에는 **[높이, …, 높이]**

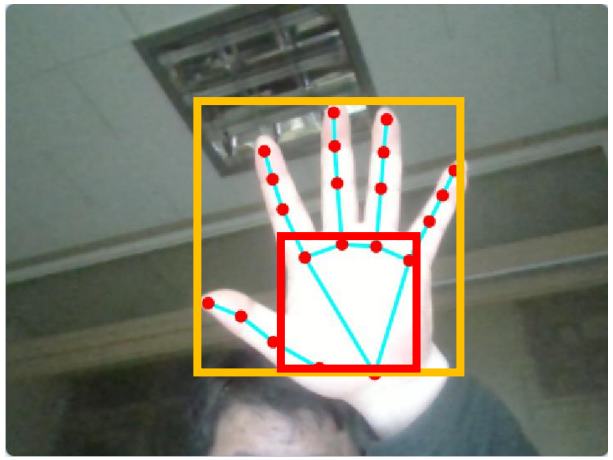


영상처리 관련 클래스, 메소드 안내

- 손 검출– HandPose 클래스의 메소드

`get_area(hand, id='all')`
손 또는 손바닥의 사각형 영역의 넓이를 반환합니다.
파라미터: • **hand**: `'left'` 또는 `'right'` • **id**: `'all'`: 모든 영역, `'hand'`: 손, `'palm'`: 손바닥 **반환 값**: • 넓이 값 • id가 `'all'`인 경우에는 **[넓이, …, 넓이]**

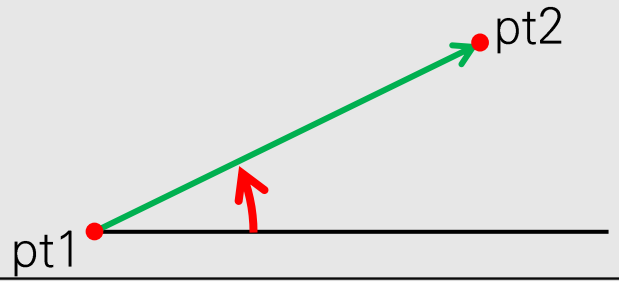
넓이 = 폭 x 높이



영상처리 관련 클래스, 메소드 안내

- 손 검출– HandPose 클래스의 메소드

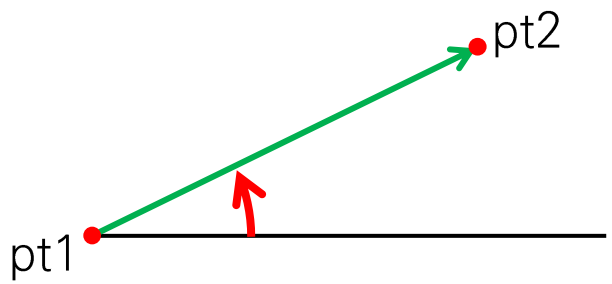
distance(pt1, pt2)
두 점 사이의 거리를 반환합니다.
파라미터: • **pt1**: 첫 번째 점의 **[x, y]** • **pt2**: 두 번째 점의 **[x, y]** **반환 값**: • 거리 값 (픽셀)

degree(pt1, pt2)
pt1에서 pt2로 향하는 직선과 수평선 사이의 각도를 도(degree)로 반환합니다.
파라미터: • **pt1**: 첫 번째 점의 **[x, y]** • **pt2**: 두 번째 점의 **[x, y]** **반환 값**: • 각도 값 (degree) 

영상처리 관련 클래스, 메소드 안내

- 손 검출– HandPose 클래스의 메소드

radian(pt1, pt2)
pt1에서 pt2로 향하는 직선과 수평선 사이의 각도를 라디안으로 반환합니다.
파라미터: • **pt1**: 첫 번째 점의 **[x, y]** • **pt2**: 두 번째 점의 **[x, y]** **반환 값**: • 각도 값 (radian)

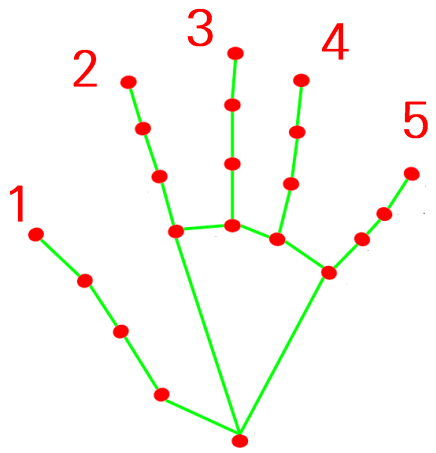


영상처리 관련 클래스, 메소드 안내

- 손 검출– HandPose 클래스의 메소드

`get_feature(filter='all')`
특징 벡터를 반환합니다.
파라미터: • `filter`: `'all'` 또는 튜플 `(손가락번호, ...)` 또는 리스트 `[손가락번호, ...]` **반환 값**: • 특징 벡터 (numpy.ndarray)

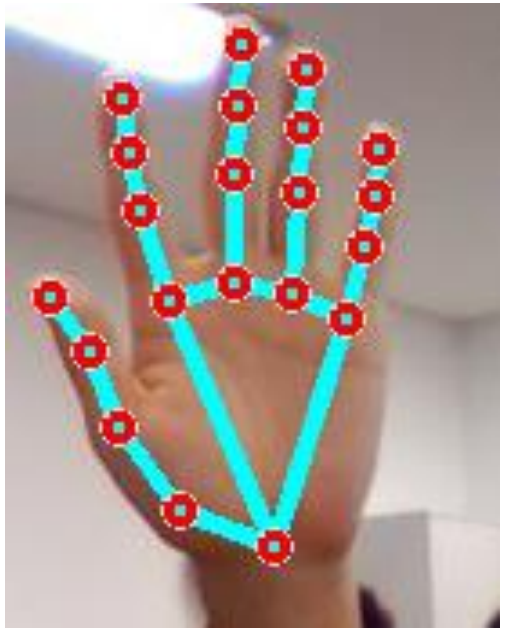
손가락 번호




```
get_feature()
get_feature('all')
get_feature((1, 3))
get_feature([2, 4, 5])
```

02 손으로 운전하기

- 손 찾기

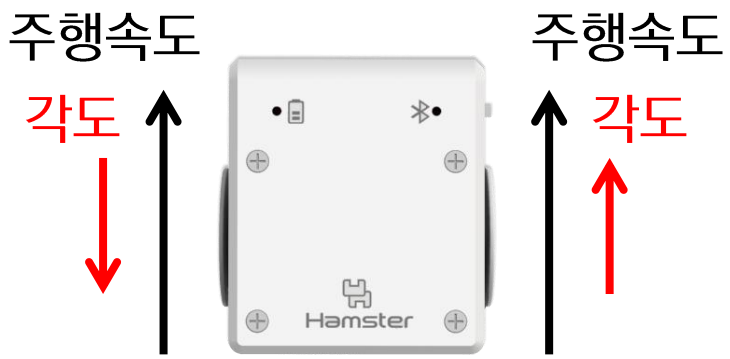


- 손을 찾는 예제를 실행해 봅시다.
- 손을 감지하기 위해서는 HandPose()라는 클래스를 이용합니다.

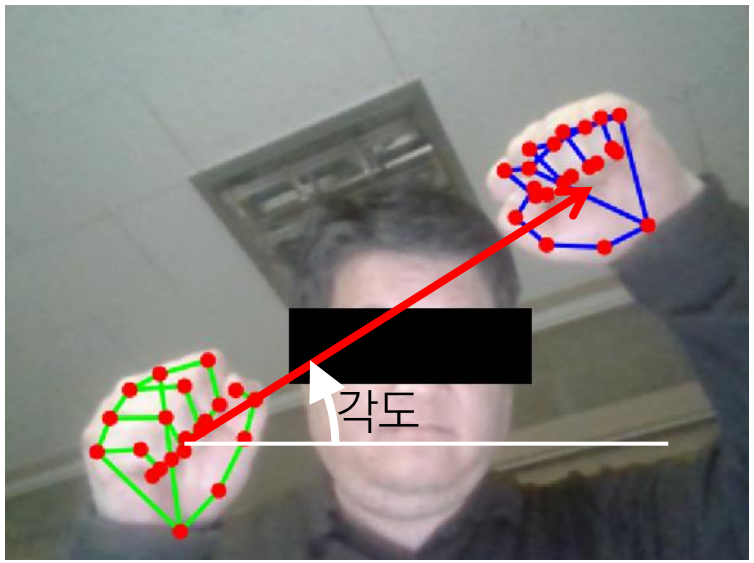
```
import roboidai as ai

cam = ai.Camera('usb0', flip='h')
dt = ai.HandPose() # 손 검출 클래스인 HandPose()를 dt로 저장한다.

dt.load_model() # 손 검출 모델을 불러온다.

while True:
    image = cam.read()
    if dt.detect(image): # 영상에서 손이 검출되면
        image = dt.draw_result(image) #검출될 손을 표시한다.

    cam.show(image)
    if cam.check_key() == 'esc': break
hamster.wheels(30, 30) # 왼쪽 바퀴와 오른쪽 바퀴의 속도를 30으로 설정한다.
```

02 손으로 운전하기

- 손으로 운전하기



주행속도 – 각도× α 주행속도 + 각도× α

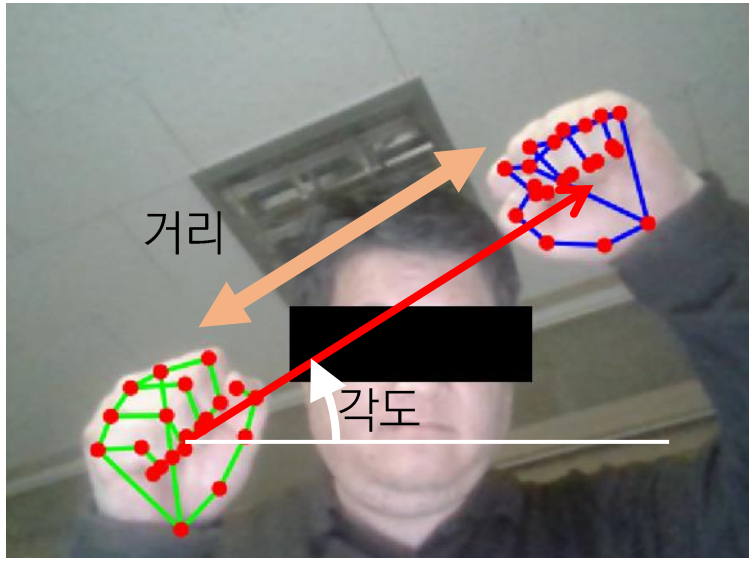


- 손으로 운전하는 예제를 실행해봅시다.
- 두 손 위치의 각도 차이를 이용하여 햄스터를 주행하게 합시다.
- 클래스와 메소드는 〈11차시 영상처리 준비〉를 참고하세요.

```
import roboidai as ai
from roboid import *

hamster = HamsterS()
cam = ai.Camera('usb0', flip='h')
dt = ai.HandPose()
dt.load_model(both_hands = True) # True로 양손 모두를 검출한다.

while True:
    image = cam.read()
    if dt.detect(image): #손이 검출되면
        left_xy= dt.get_xy('left','palm') #왼손바닥의 좌표와 오른손바닥의 좌표를 반환
        right_xy= dt.get_xy('right','palm')
        angle = dt.degree(left_xy, right_xy) #두 손바닥 간의 각도를 angle에 저장
        print(angle)
        if angle is not None: # angle이 계산될 때, angle에 따라 햄스터 주행
            hamster.wheels(30-angle* 0.5, 30+ angle* 0.5)
        else: # angle이 없을 때, 정지
            hamster.stop()
        image = dt.draw_result(image)
    else: # 손이 검출안되면 정지
        hamster.stop()

    cam.show(image)
    if cam.check_key() == 'esc': break # esc를 누르면 프로그램을 종료한다.
```

02 손으로 운전하기

- 속도 + 방향으로 운전하기



- 속도와 방향으로 운전하는 예제를 실행해봅시다.
- 두 손 위치의 각도 차이를 이용하여 햄스터를 주행하게 합시다.
- 클래스와 메소드는 〈11차시 영상처리 준비〉를 참고하세요.

```
import roboidai as ai
from roboid import *

hamster = HamsterS()
cam = ai.Camera('usb0', flip='h')
dt = ai.HandPose()
dt.load_model(both_hands = True) # True로 양손 모두를 검출한다.

while True:
    image = cam.read()
    if dt.detect(image): #손이 검출되면
        left_xy= dt.get_xy('left','palm') #왼손바닥의 좌표와 오른손바닥의 좌표를 반환
        right_xy= dt.get_xy('right','palm')
        dist= dt.distance(left_xy, right_xy)#두 손바닥 간의 거리를 dist에 저장
        angle = dt.degree(left_xy, right_xy) #두 손바닥 간의 각도를 angle에 저장
        print(dist, angle)
        if angle is not None: # angle이 계산될 때, angle에 따라 햄스터 주행
            hamster.wheels((dist/5-20.0)-angle*0.5, (dist/5-20.0)+angle*0.5)
        else: # angle이 없을 때, 정지
            hamster.stop()
        image = dt.draw_result(image)
    else: # 손이 검출안되면 정지
        hamster.stop()
    cam.show(image)
    if cam.check_key() == 'esc': break # esc를 누르면 프로그램을 종료한다.
```

16차시

손모양 인식
(티처블 머신)

01 영상 수집

- 영상 수집을 위한 클래스, 메소드 안내

`collect_image(cam, folder)`
카메라를 사용하여 스페이스 키를 누를 때마다 영상을 촬영하여 저장합니다. ESC 키를 누르면 종료합니다.
파라미터: • `cam`: 카메라 객체 • `folder`: 영상을 저장할 폴더, 폴더가 존재하지 않으면 생성함

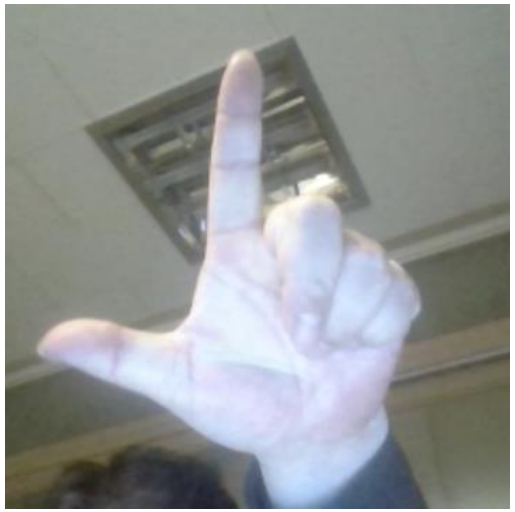
`record_image(cam, folder, interval_msec=100, frames=20, countdown=3)`
카메라를 사용하여 카운트다운 후에 `interval_msec` 간격으로 `frames`개의 영상을 촬영하여 저장합니다. ESC 키를 누르면 종료합니다.
파라미터: • `cam`: 카메라 객체 • `folder`: 영상을 저장할 폴더, 폴더가 존재하지 않으면 생성함 • `interval_msec`: 영상을 저장하는 시간 간격 (msec) • `frames`: 저장할 영상 개수 • `countdown`: 카운트다운 수

01 영상 수집

- 영상 수집을 위한 코드를 작성해봅시다.
- 스페이스 키를 누를 때마다 사진을 저장합니다.

```
import roboidai as ai
import roboidai.lab.ko as lab

cam = ai.Camera('usb0', flip='h', square=True) # USB 카메라를 사용한다(웹캠). 수평 반전, 정사각형 화면
lab.collect_image(cam, 'c:/Example/scissors')
```

c:/Example/scissors



c:/Example/rock

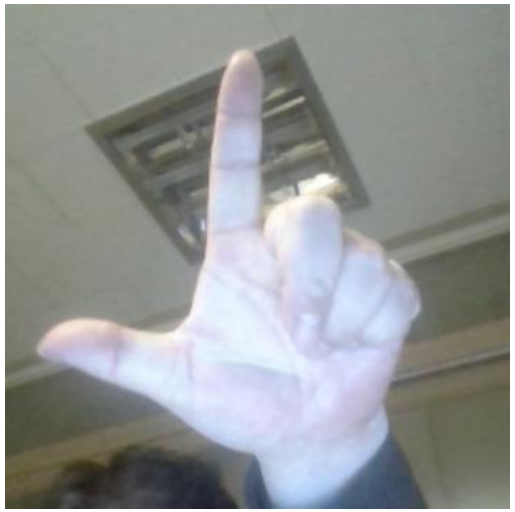
c:/Example/paper

01 영상 수집

- 영상 수집을 위한 코드를 작성해봅시다.
- 200msec(0.2초)를 기준으로 30개의 영상을 촬영하여 저장합니다.

```
import roboidai as ai
import roboidai.lab.ko as lab

cam = ai.Camera('usb0', flip='h', square=True) # USB 카메라를 사용한다(웹캠). 수평 반전, 정사각형 화면
lab.record_image(cam, 'c:/Example/scissors', 200, 30)
```

c:/Example/scissors

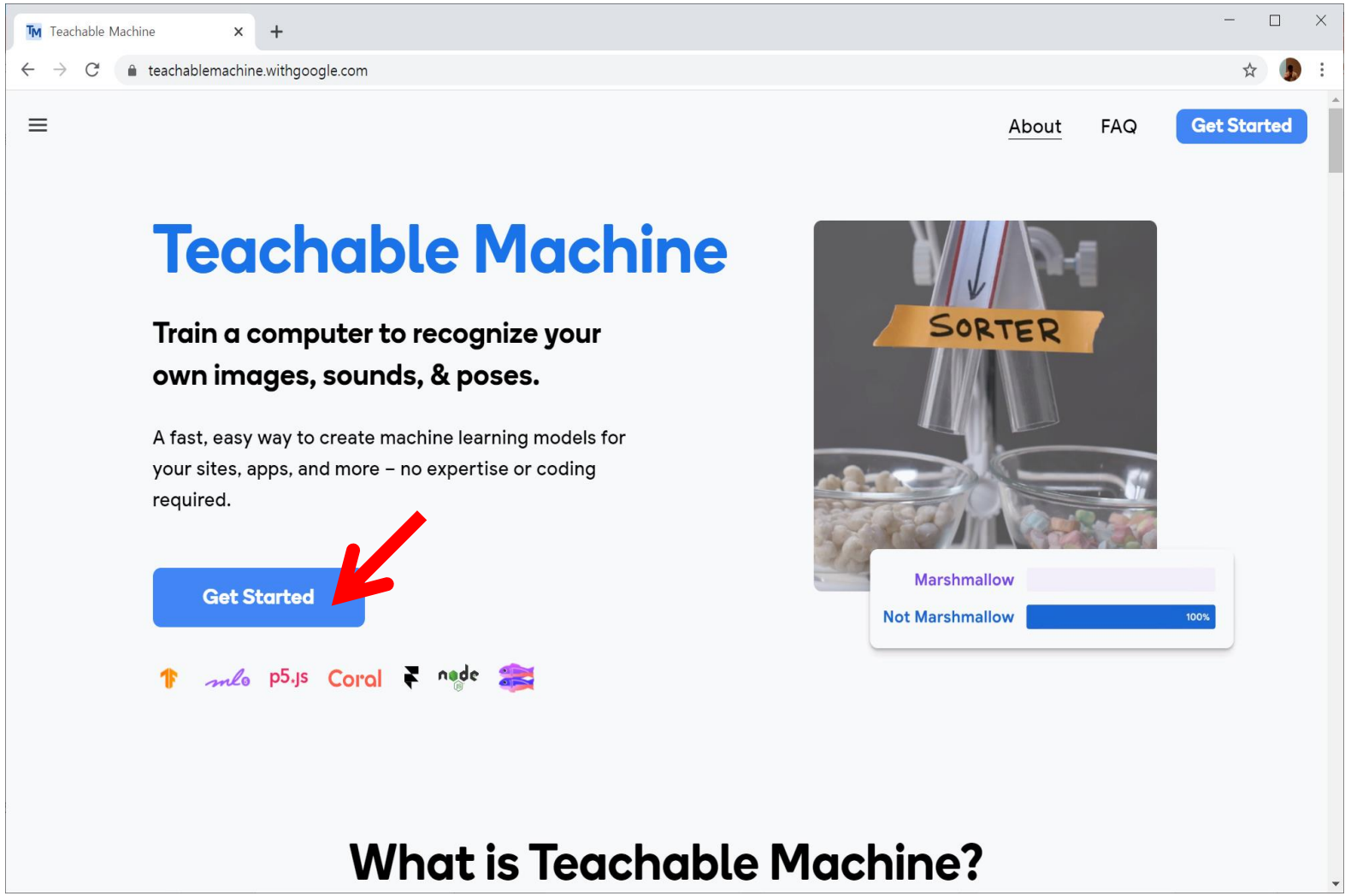


c:/Example/rock

c:/Example/paper

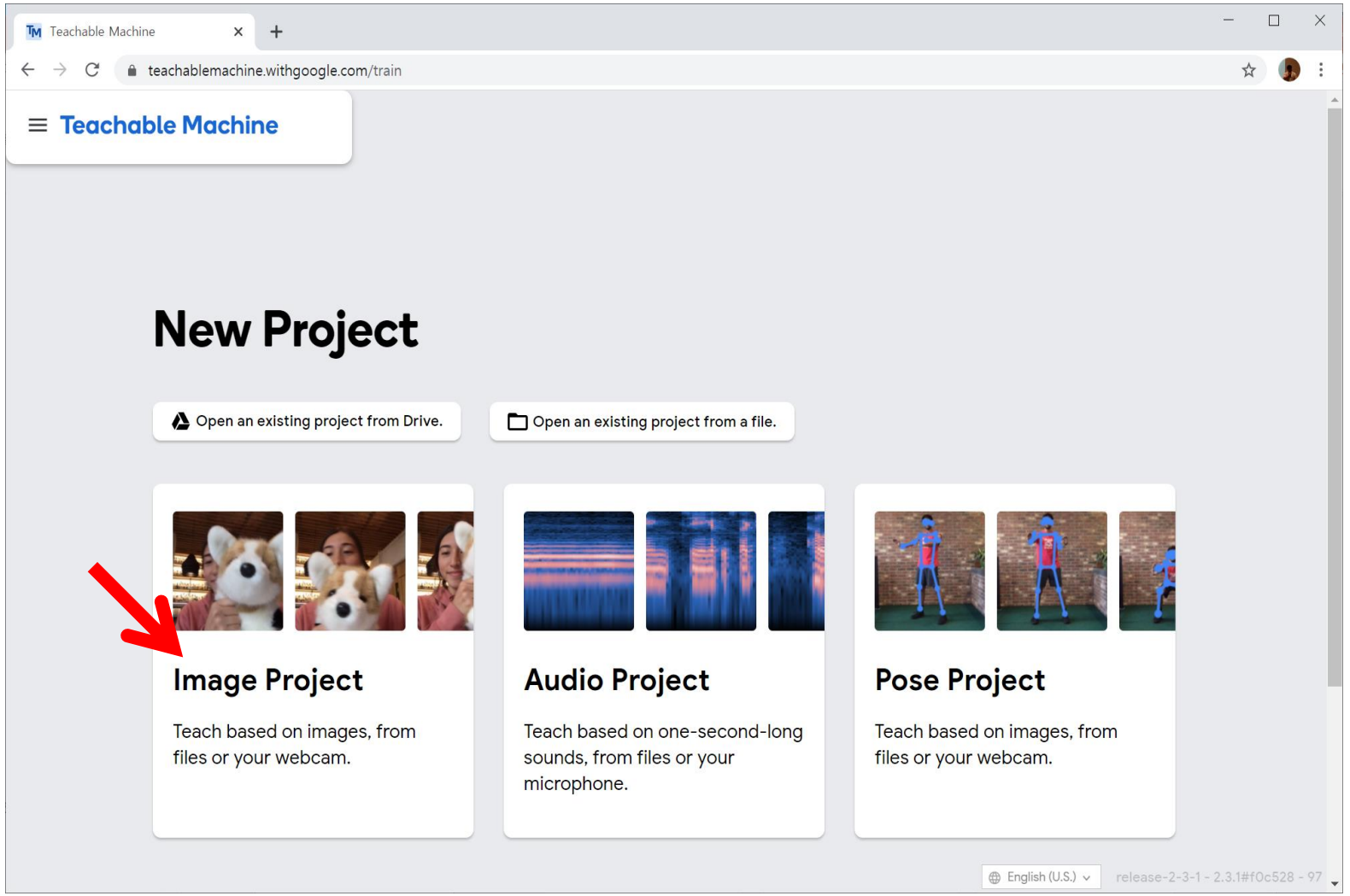
영상 학습

- teachablemachine.withgoogle.com에 접속합니다.



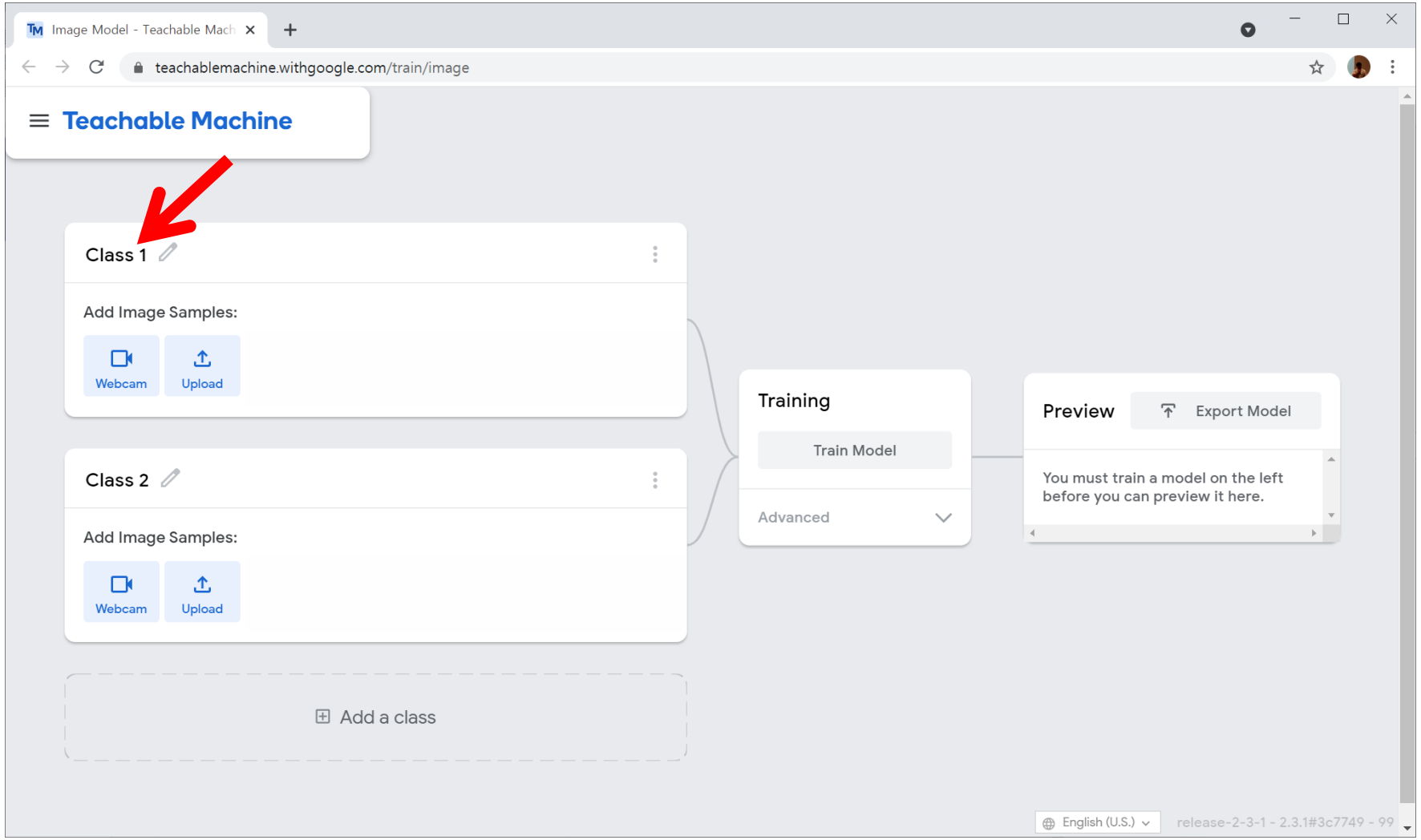
영상 학습

- Image Project 를 클릭합니다. (표준 이미지 모델)



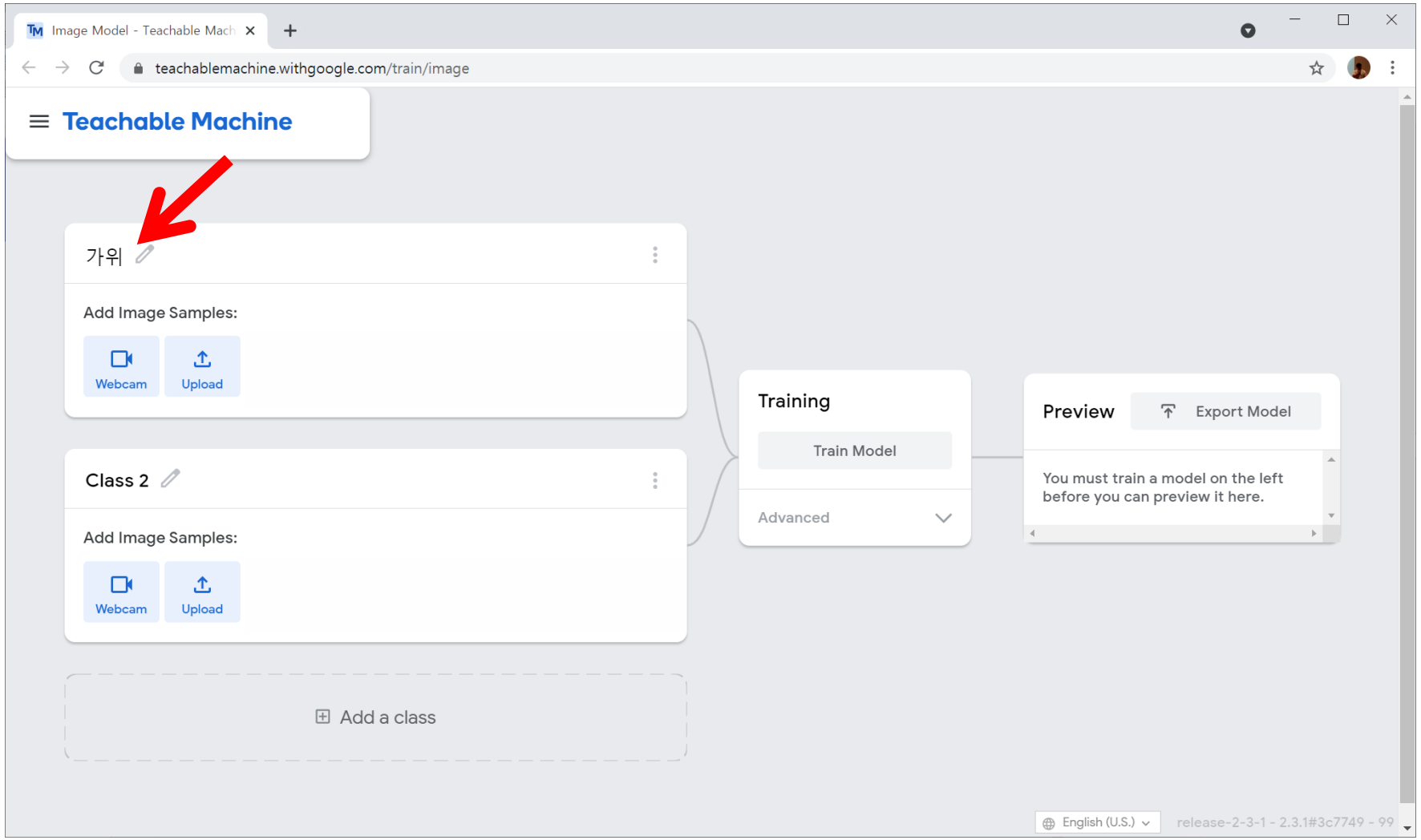
영상 학습

- Class 1을 클릭합니다.



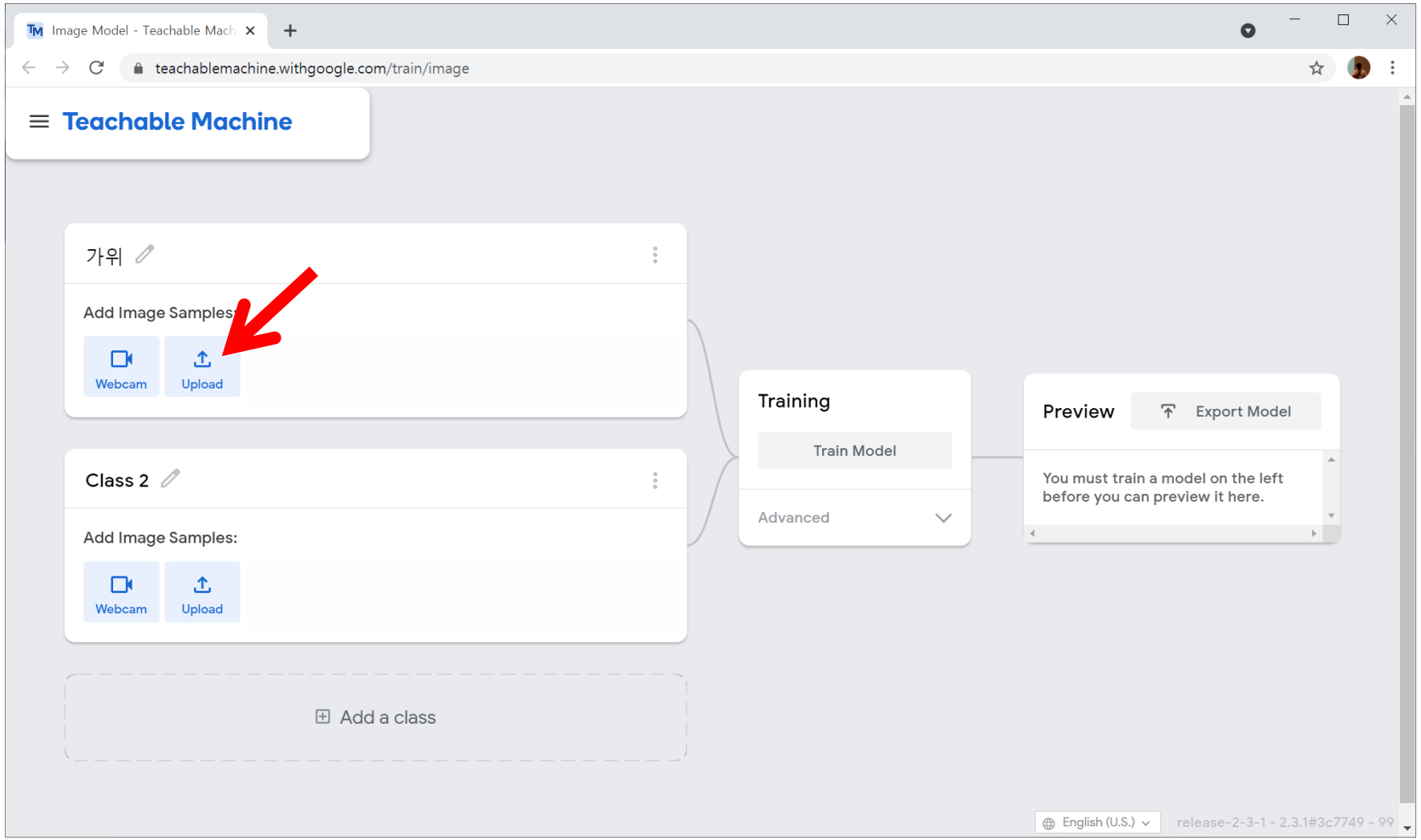
영상 학습

- 클래스 이름을 '가위'로 변경해줍니다.



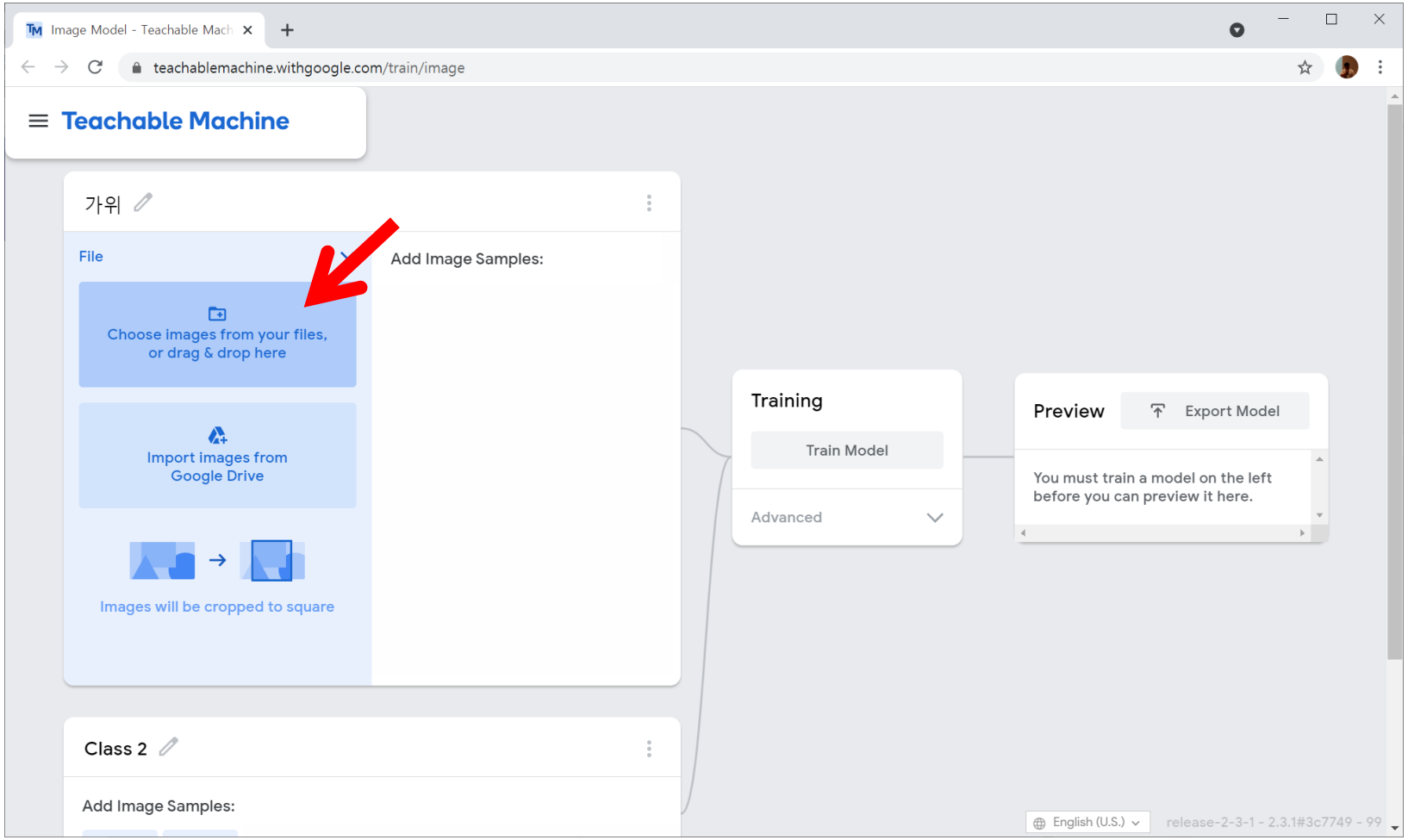
영상 학습

- Upload를 클릭합니다.



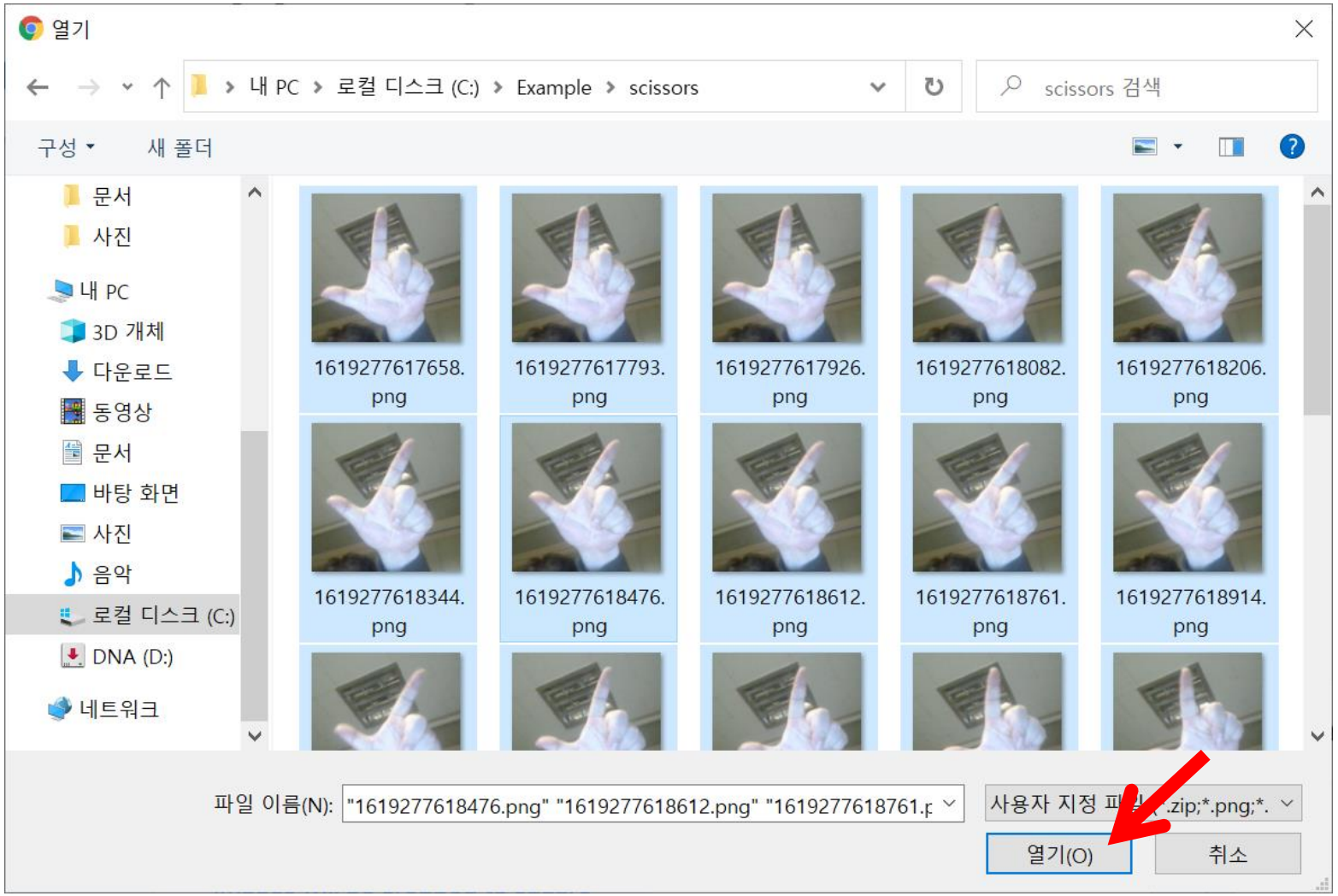
영상 학습

- Choose images from your files 를 클릭합니다.



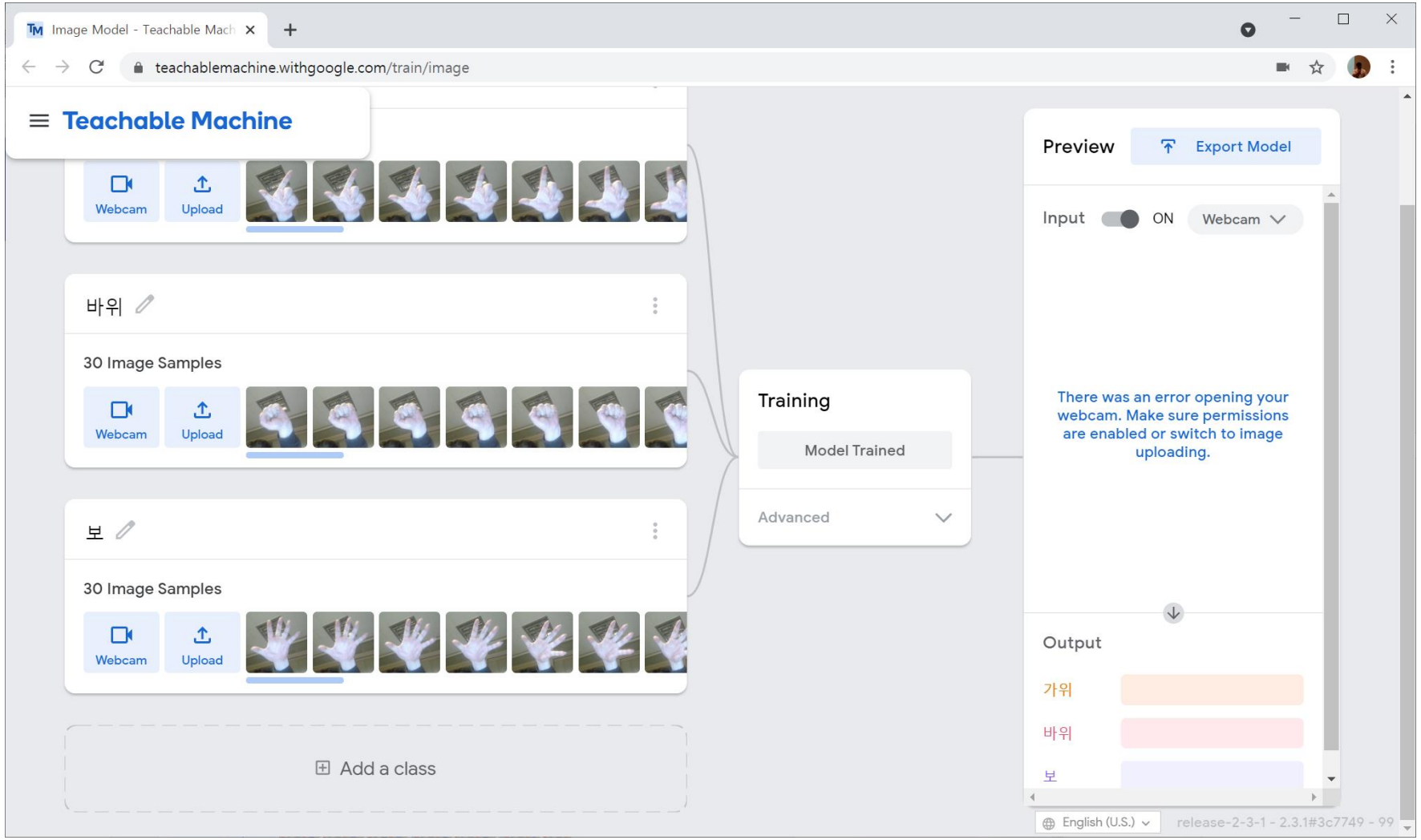
영상 학습

- 영상을 저장한 폴더에서 가위 영상을 모두 선택합니다. -> **열기** 버튼을 클릭해줍니다.



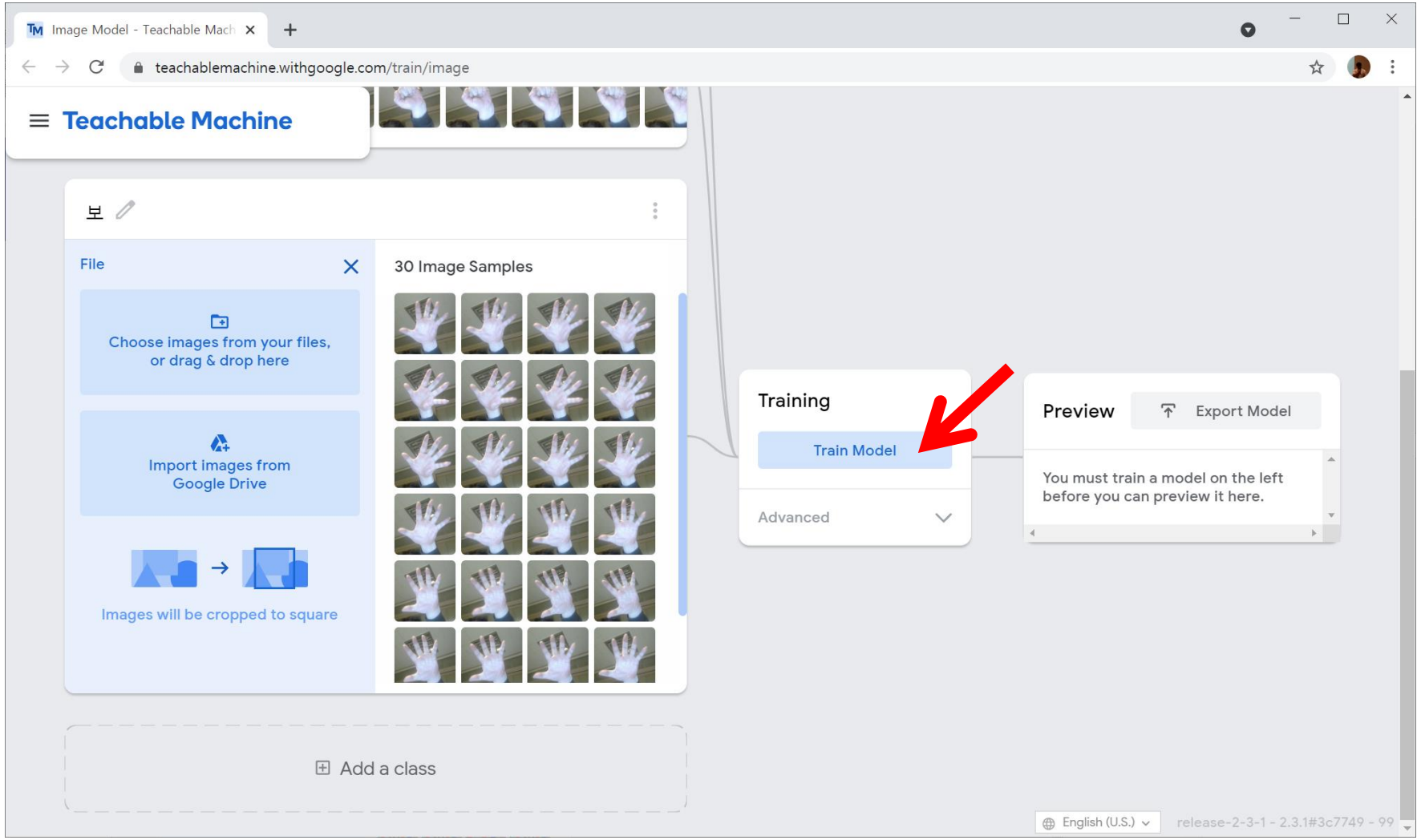
영상 학습

- **Class 2와 Class 3**도 마찬가지로 '바위', '보' 사진을 업로드 해줍니다.



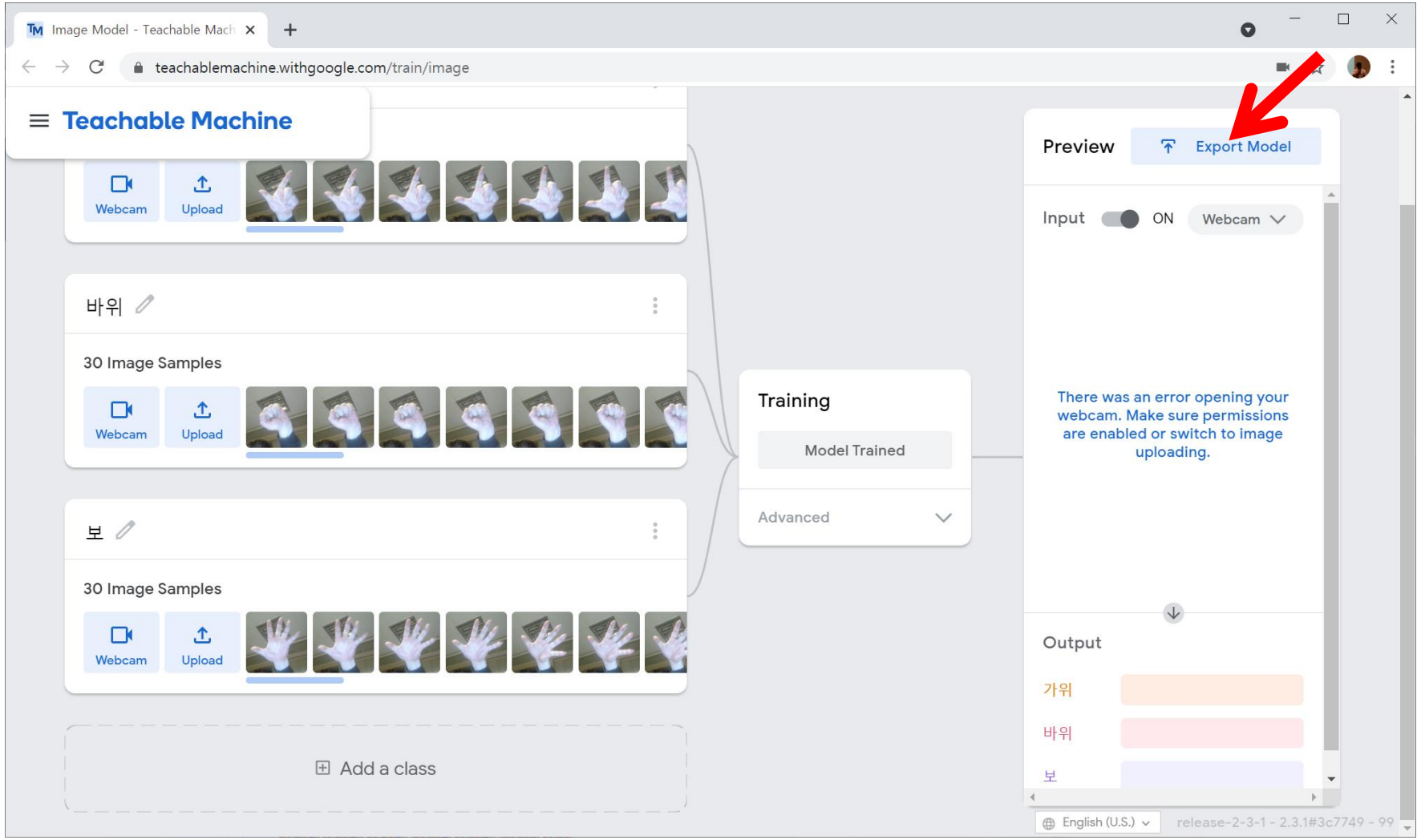
영상 학습

- 사진을 모두 업로드했으면, **Train Model**을 클릭합니다.



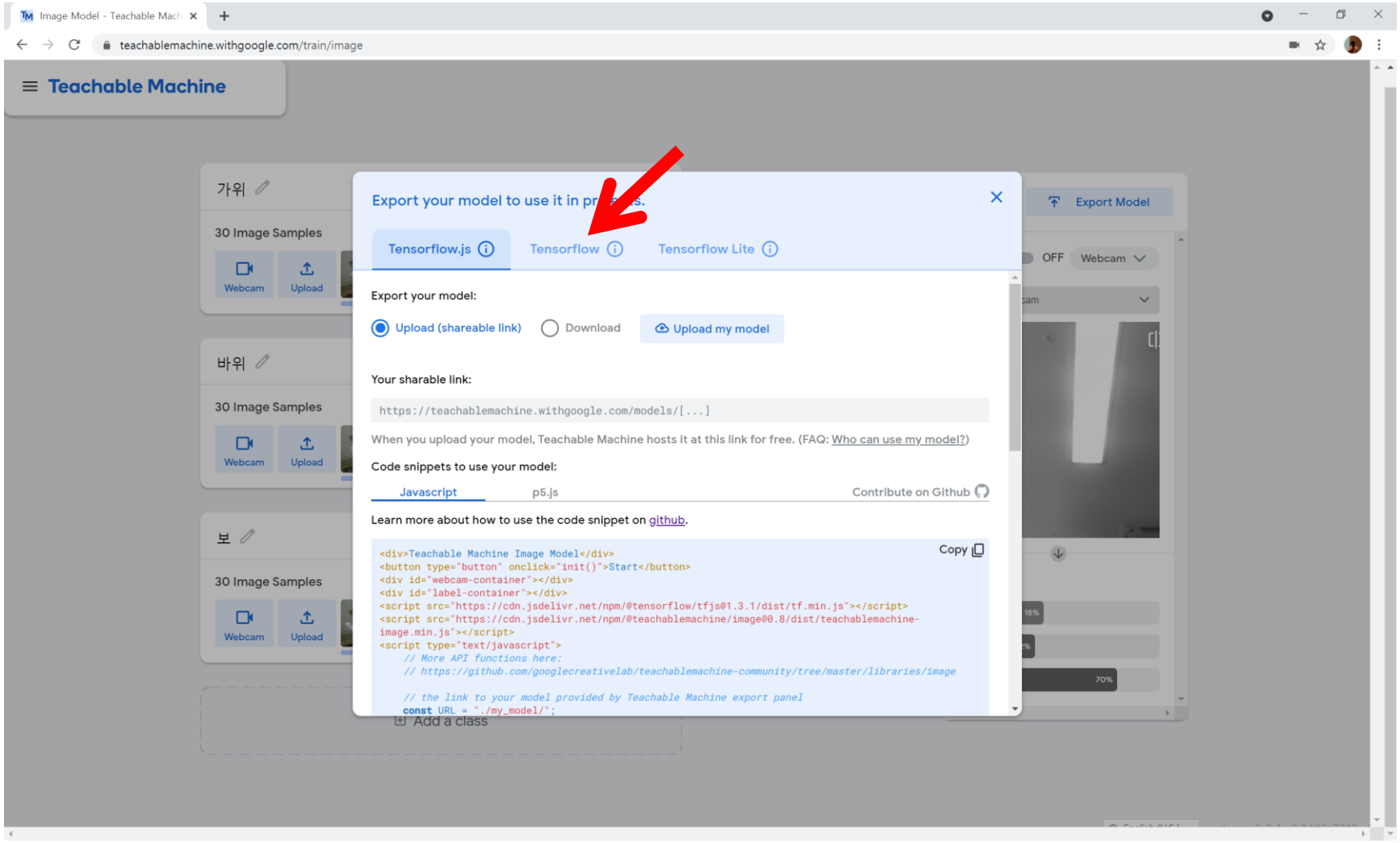
영상 학습

- 학습이 완료되면 **Export Model**을 클릭합니다.



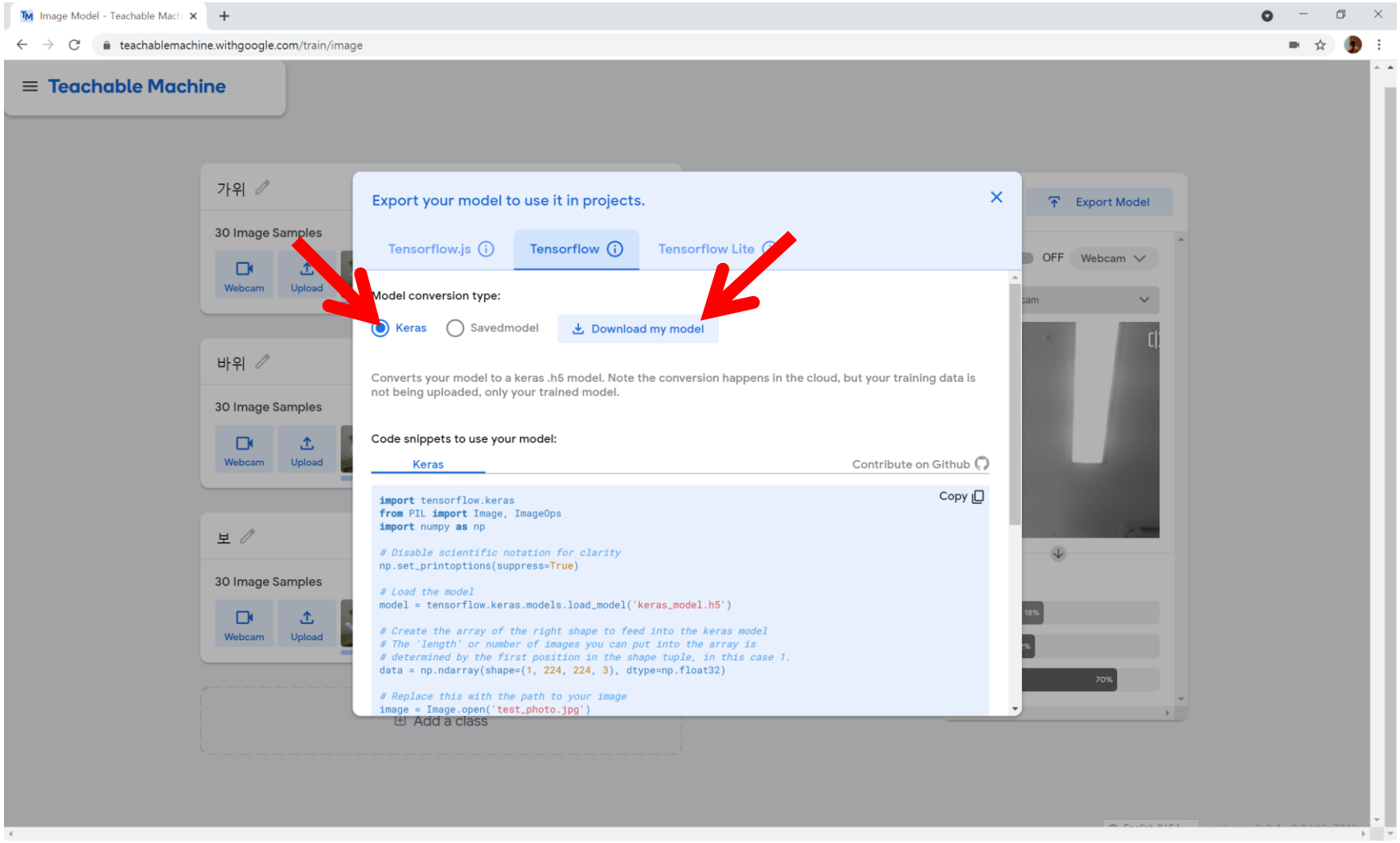
영상 학습

- Tensorflow 를 클릭합니다.



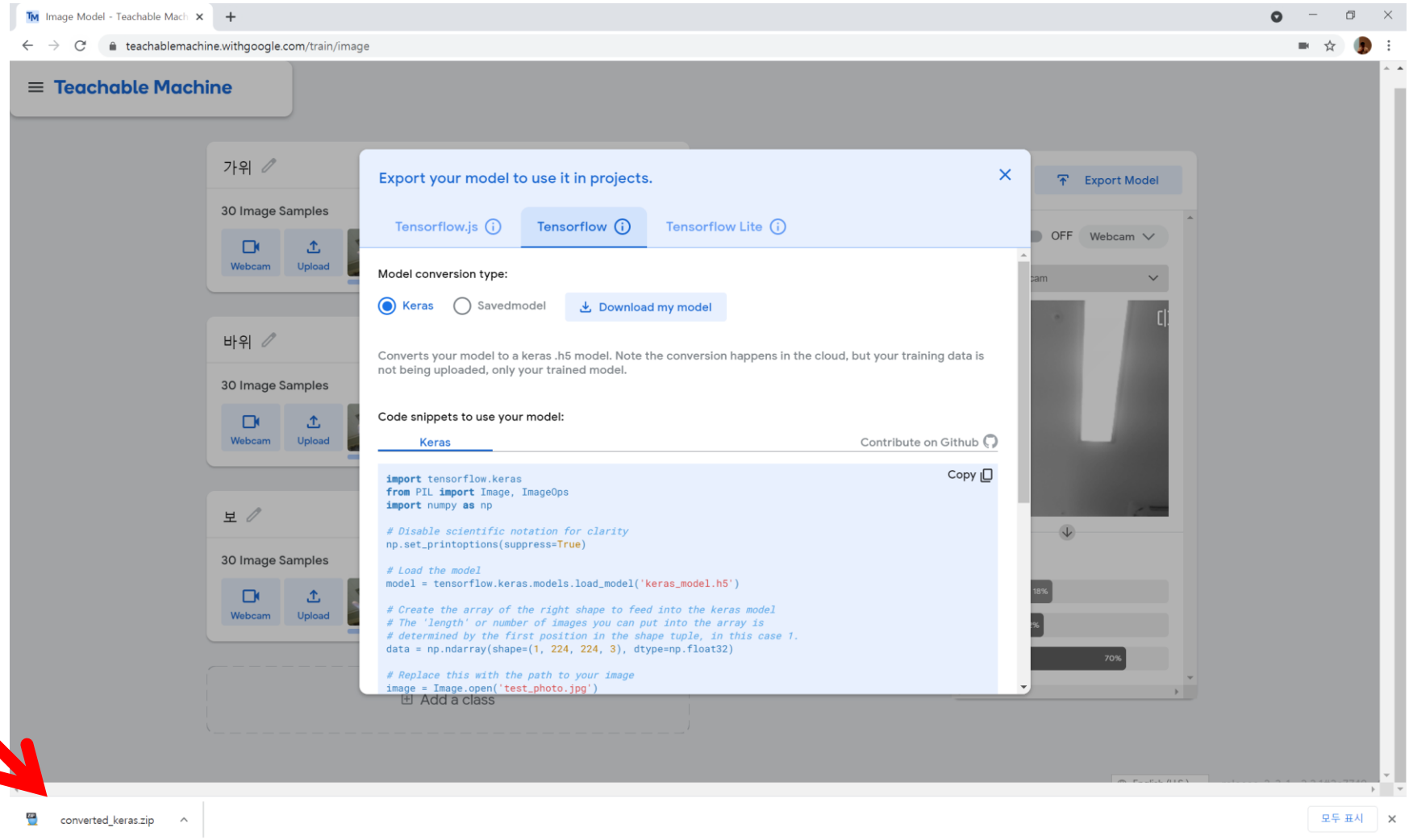
영상 학습

- Keras 를 선택한 후, Download my model 을 클릭합니다.



영상 학습

- 변환이 끝나면 파일이 다운로드 됩니다.



영상 학습

- 다운로드된 converted_keras.zip 파일을 적절한 위치에서 압축을 풀어 줍니다.
  ex) c:/Example/가위바위보

- 압축 파일 내에는 2개의 파일이 들어 있습니다.
  - keras_model.h5
  - labels.txt

가위바위보 실습

- TmImage 클래스, 메소드 안내

predict(image, threshold=0.5)
영상을 인식합니다.
파라미터: • **image**: 인식할 입력 영상 (nunpy.ndarray) • **threshold**: 신뢰도가 **threshold** 이상인 경우에만 인식된 것으로 인정 반환 값: • **True**: 인식 성공, **False**: 실패

get_label()
인식 결과의 레이블을 반환합니다.
반환 값: • 신뢰도 값이 가장 큰 클래스의 레이블

가위바위보 실습

- TmImage 클래스, 메소드 안내

get_conf()
인식 결과의 신뢰도를 반환합니다.
반환 값: • 신뢰도 값이 가장 큰 클래스의 신뢰도

get_all_labels()
모든 클래스의 레이블 목록을 반환합니다.
반환 값: • 모든 클래스의 레이블 목록 (numpy.ndarray)

get_all_confs()
모든 클래스의 신뢰도 목록을 반환합니다.
반환 값: • 모든 클래스의 신뢰도 목록 (numpy.ndarray)

03 가위바위보 실습

- 학습한 가위바위보 모델을 이용해 이를 분류하는 코드를 작성해봅시다.

```
import roboidai as ai

tmi = ai.TmImage()
tmi.load_model('c:/Example/가위바위보')

cam = ai.Camera('usb0', flip='h', square=True) # USB 카메라를 사용한다(웹캠), 수평 반전, 정사각형 이미지
cam.count_down(3)

while True:
    image = cam.read()
    if tmi.predict(image):# 인식 성공 시
        label = tmi.get_label() # 인식한 결과를 반환합니다.
        print(label) # 인식한 결과를 출력합니다.

    cam.show(image)
    if cam.check_key() == 'esc': break
```

03 가위바위보 실습

- 학습한 가위바위보 모델을 이용해 이를 분류하는 코드를 작성해봅시다.

- 신뢰도가 일정 값 이상인 경우에만 결과가 나타나도록 해봅시다.

```
import roboidai as ai

tmi = ai.TmImage()
tmi.load_model('c:/Example/가위바위보')

cam = ai.Camera('usb0', flip='h', square=True) # USB 카메라를 사용한다(웹캠), 수평 반전, 정사각형 이미지
cam.count_down(3)

while True:
    image = cam.read()
    if tmi.predict(image):# 인식 성공 시
        label = tmi.get_label() # 인식한 결과를 반환합니다.

        conf = tmi.get_conf() # 신뢰도를 반환합니다.
        if conf > 0.8:
            print(label, conf) # 인식한 결과와 신뢰도를 출력합니다.

    cam.show(image)
    if cam.check_key() == 'esc': break
```

03 가위바위보 실습

- 학습한 가위바위보 모델을 이용해 이를 분류하는 코드를 작성해봅시다.

- 햄스터와 함께 동작해봅시다.

```
import roboidai as ai

tmi = ai.TmImage()
tmi.load_model('c:/Example/가위바위보')

cam = ai.Camera('usb0', flip='h', square=True) # USB 카메라를 사용한다(웹캠), 수평 반전, 정사각형 이미지
cam.count_down(3)

while True:
    image = cam.read()
    if tmi.detect(image):# 인식 성공 시
        label = tmi.get_label() # 인식한 결과를 반환합니다.
        conf = tmi.get_conf() # 신뢰도를 반환합니다.
        if conf > 0.8:
            print(label, conf) # 인식한 결과와 신뢰도를 출력합니다.
            if label == '가위':
                hamster.wheels(30, 30)
            elif label == '바위':
                hamster.wheels(-30, -30)
            elif label == '보':
                hamster.stop()

cam.show(image)
    if cam.check_key() == 'esc': break
```

03 가위바위보 실습

- 학습한 가위바위보 모델을 이용해 이를 분류하는 코드를 작성해봅시다.

- 동일한 코드를 .predict()의 매개변수를 활용해 약간 수정해봅시다.

```
import roboidai as ai

tmi = ai.TmImage()
tmi.load_model('c:/Example/가위바위보')

cam = ai.Camera('usb0', flip='h', square=True) # USB 카메라를 사용한다(웹캠), 수평 반전, 정사각형 이미지
cam.count_down(3)

while True:
    image = cam.read()
    if tmi.detect(image, 0.8):# 인식 성공 시. 신뢰도가 0.8 이상일 때만 인식 인정
        label = tmi.get_label() # 인식한 결과를 반환합니다.
        conf = tmi.get_conf() # 신뢰도를 반환합니다.

        if label == '가위':
            hamster.wheels(30, 30)
        elif label == '바위':
            hamster.wheels(-30, -30)
        elif label == '보':
            hamster.stop()

    cam.show(image)
    if cam.check_key() == 'esc': break
```

번외

자율주행 햄스터 AI카메라 활용

영상처리 관련 클래스, 메소드 안내

- 트랙 검출

find_track_xy(image, output, color, hrange, srange=(50,255), vrange=(50,255), window_height=-1, min_area=0)

영상에서 h, s, v 범위에 해당하는 Blob을 찾아 무게 중심을 반환합니다.

파라미터:

- **image**: 입력 영상 (nunpy.ndarray)
- **output**: 결과를 표시할 영상 (numpy.ndarray)
- **color**: 결과를 표시할 색깔 (B, G, R)
- **hrange**: Hue의 범위 (최소, 최대), 범위가 2개 이면 (최소1, 최대1, 최소2, 최대2)
- **srange**: Saturation의 범위 (최소, 최대)
- **vrange**: Value의 범위 (최소, 최대)
- **window_height**: 초록색 및 파란색 선을 검출하는 영상 부분의 세로 방향 높이, 영상 제일 아래에서 window_height 높이 만큼의 영상에서 초록색 및 파란색 선을 검출함, -1이면 영상 전체를 사용
- **min_area**: 발견된 Blob의 넓이가 **min_area**보다 작으면 Blob으로 인정하지 않음

반환 값:

- **(x, y)** 튜플: 무게 중심의 x, y 좌표

영상처리 관련 클래스, 메소드 안내

- 트랙 검출

find_green_track_xy(image, output, h_range=(40, 80), s_range=(50,255), v_range=(50,255), window_height=-1, min_area=0)

영상에서 초록색 Blob을 찾아 무게 중심을 반환합니다.

파라미터:

- **image**: 입력 영상 (nunpy.ndarray)
- **output**: 결과를 표시할 영상 (numpy.ndarray)
- **hrange**: Hue의 범위 (최소, 최대), 범위가 2개 이면 (최소1, 최대1, 최소2, 최대2)
- **srange**: Saturation의 범위 (최소, 최대)
- **vrange**: Value의 범위 (최소, 최대)
- **window_height**: 초록색 및 파란색 선을 검출하는 영상 부분의 세로 방향 높이, 영상 제일 아래에서 window_height 높이 만큼의 영상에서 초록색 및 파란색 선을 검출함, -1이면 영상 전체를 사용
- **min_area**: 발견된 Blob의 넓이가 **min_area**보다 작으면 Blob으로 인정하지 않음

반환 값:

- **(x, y)** 튜플: 무게 중심의 x, y 좌표

영상처리 관련 클래스, 메소드 안내

- 트랙 검출

find_blue_track_xy(image, output, h_range=(100, 140), s_range=(50,255), v_range=(50,255), window_height=-1, min_area=0)

영상에서 파란색 Blob을 찾아 무게 중심을 반환합니다.

파라미터:

- **image**: 입력 영상 (nunpy.ndarray)
- **output**: 결과를 표시할 영상 (numpy.ndarray)
- **hrange**: Hue의 범위 (최소, 최대), 범위가 2개 이면 (최소1, 최대1, 최소2, 최대2)
- **srange**: Saturation의 범위 (최소, 최대)
- **vrange**: Value의 범위 (최소, 최대)
- **window_height**: 초록색 및 파란색 선을 검출하는 영상 부분의 세로 방향 높이, 영상 제일 아래에서 window_height 높이 만큼의 영상에서 초록색 및 파란색 선을 검출함, -1이면 영상 전체를 사용
- **min_area**: 발견된 Blob의 넓이가 **min_area**보다 작으면 Blob으로 인정하지 않음

반환 값:

- **(x, y)** 튜플: 무게 중심의 x, y 좌표

영상처리 관련 클래스, 메소드 안내

- 트랙 검출

find_red_track_xy(image, output, h_range=(0, 10, 170, 180), s_range=(50,255), v_range=(50,255), window_height=-1, min_area=0)

영상에서 빨간색 Blob을 찾아 무게 중심을 반환합니다.

파라미터:

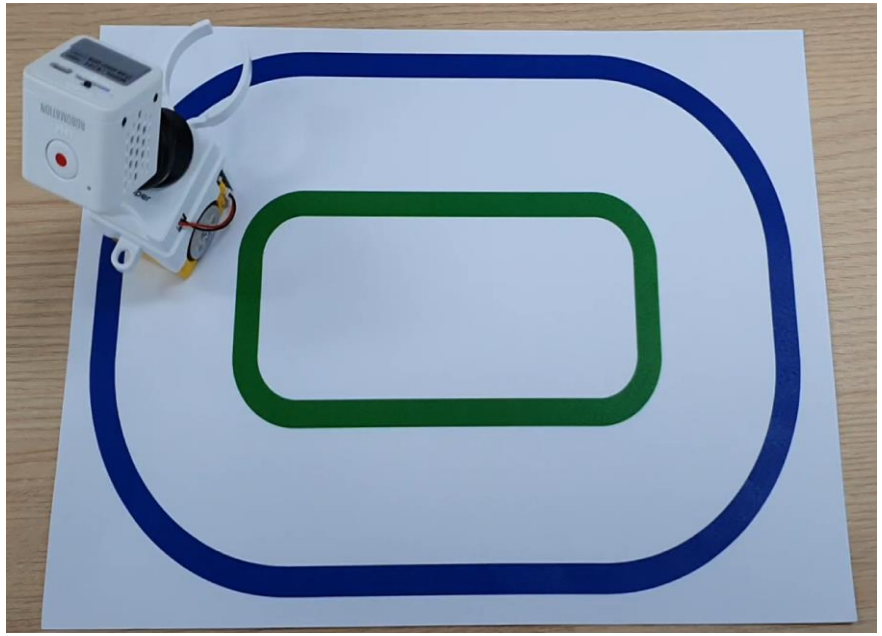
- **image**: 입력 영상 (nunpy.ndarray)
- **output**: 결과를 표시할 영상 (numpy.ndarray)
- **hrange**: Hue의 범위 (최소, 최대), 범위가 2개 이면 (최소1, 최대1, 최소2, 최대2)
- **srange**: Saturation의 범위 (최소, 최대)
- **vrange**: Value의 범위 (최소, 최대)
- **window_height**: 초록색 및 파란색 선을 검출하는 영상 부분의 세로 방향 높이, 영상 제일 아래에서 window_height 높이 만큼의 영상에서 초록색 및 파란색 선을 검출함, -1이면 영상 전체를 사용
- **min_area**: 발견된 Blob의 넓이가 **min_area**보다 작으면 Blob으로 인정하지 않음

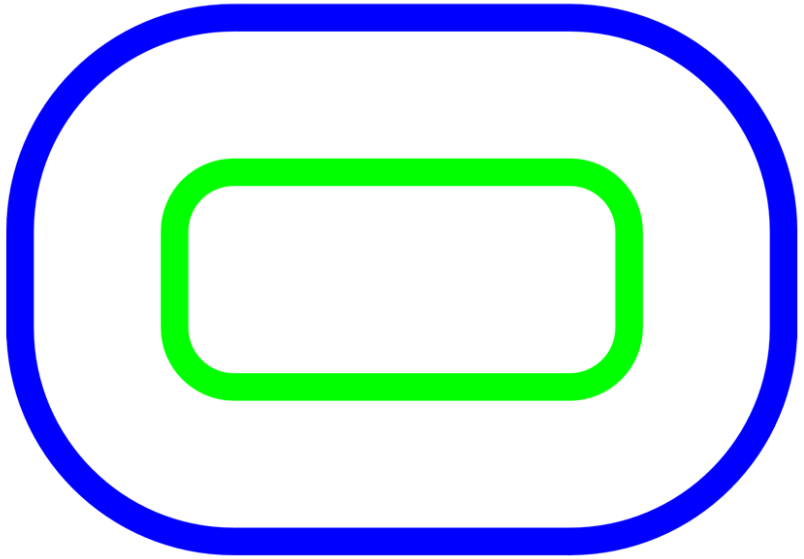
반환 값:

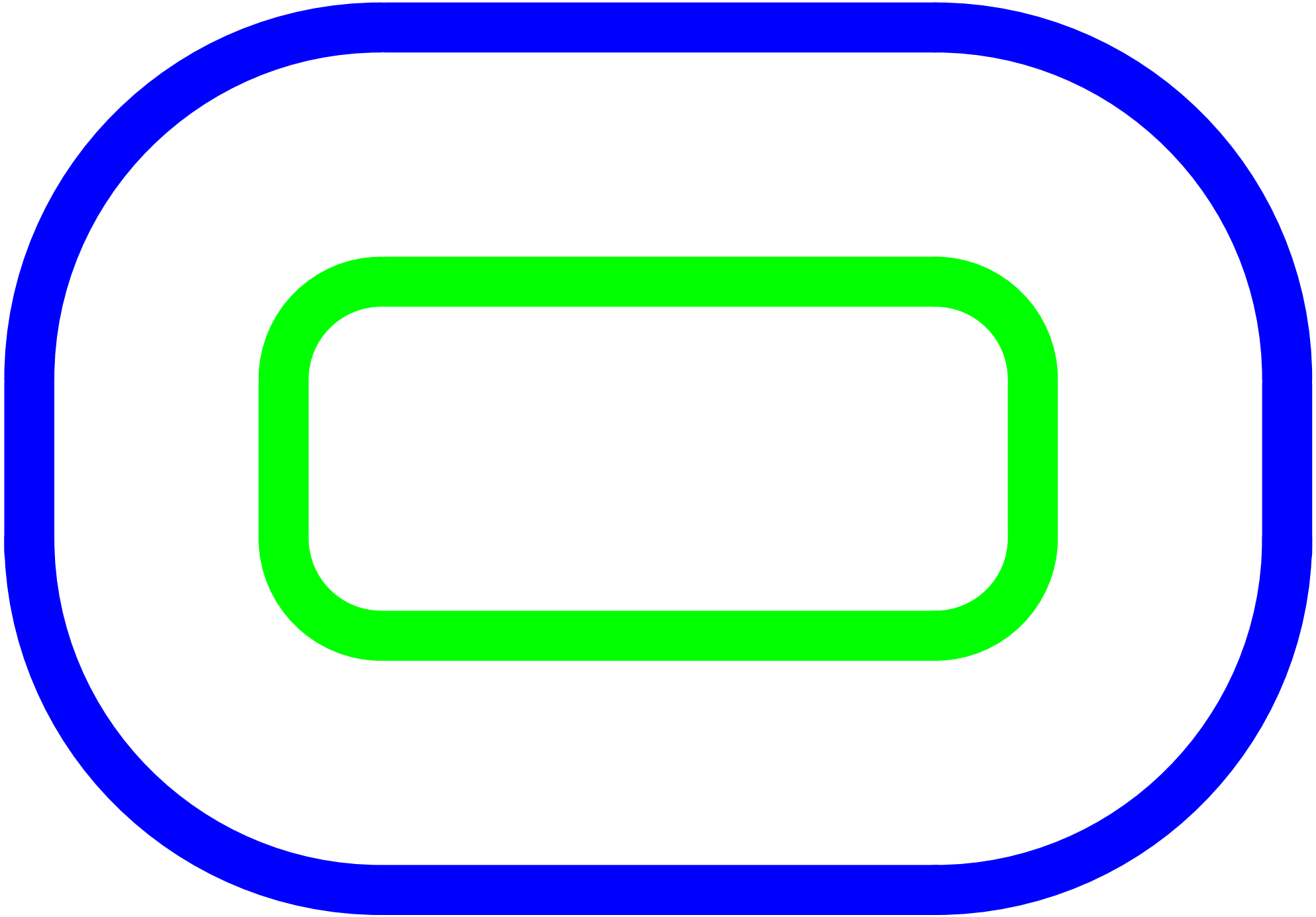
- **(x, y)** 튜플: 무게 중심의 x, y 좌표

자율주행

- 준비물
  - 활동지
  - 햄스터 AI 카메라







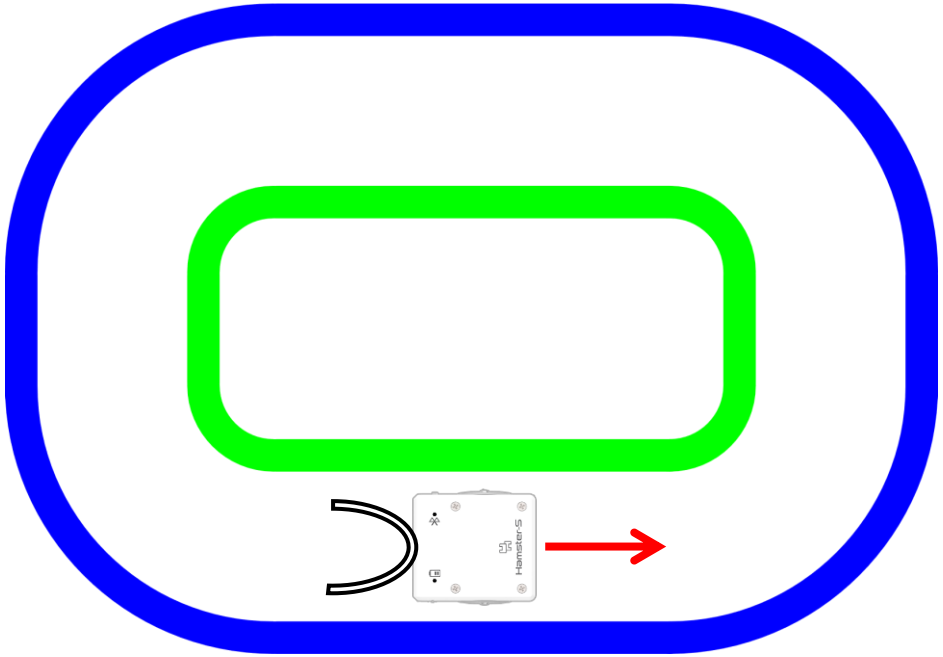
자율주행

- 받침대 활용 시 카메라 설치
  - 카메라 받침대를 햄스터 로봇의 앞쪽 끝에 부착하고 뒤쪽 아래로 기울입니다.
  - 카메라가 뒤쪽 아래를 바라보도록 카메라를 카메라 받침대에 부착합니다.
  - 카메라가 삐뚤지 않게 해주세요!

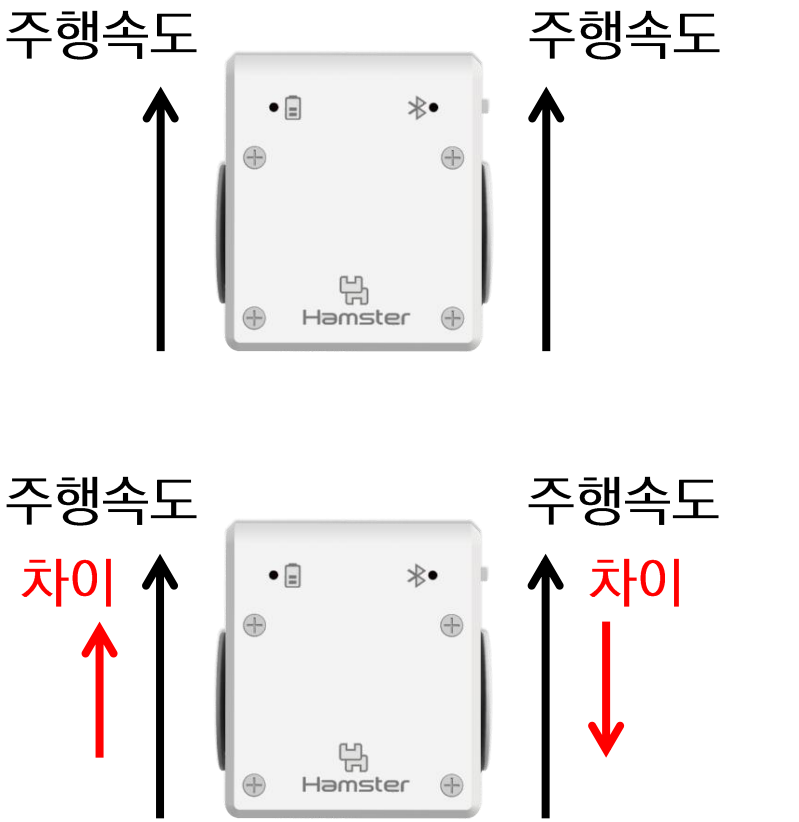
자율주행

- 로봇 배치

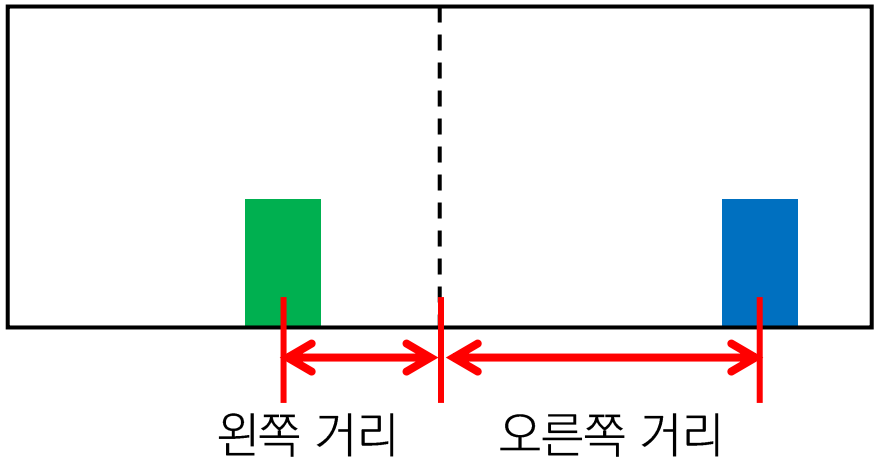
  - 햄스터 로봇을 활동지에 올려 놓습니다. 반시계 방향으로 이동합니다.
  - 후진으로 주행하므로 거꾸로 놓아야 합니다.



자율주행

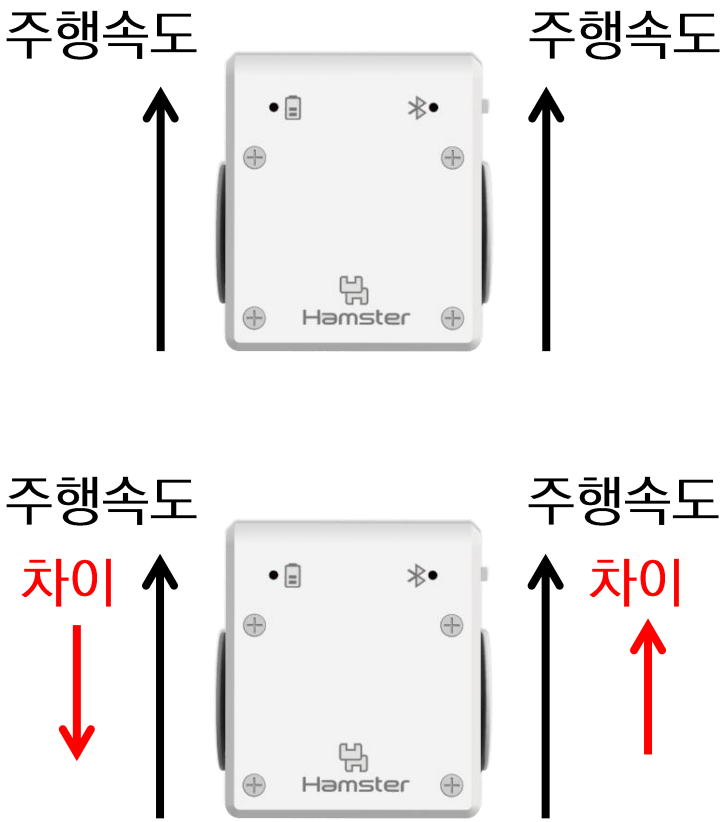


- 차이 = 오른쪽 거리 – 왼쪽 거리
- 차이를 기존 주행 속도에 더하고(왼쪽 바퀴) 빼게(오른쪽 바퀴) 되면 차이가 커질 수록, 왼쪽 바퀴의 속도가 오른쪽보다 빨라져 오른쪽으로 회전하는 형태로 주행하게 된다.

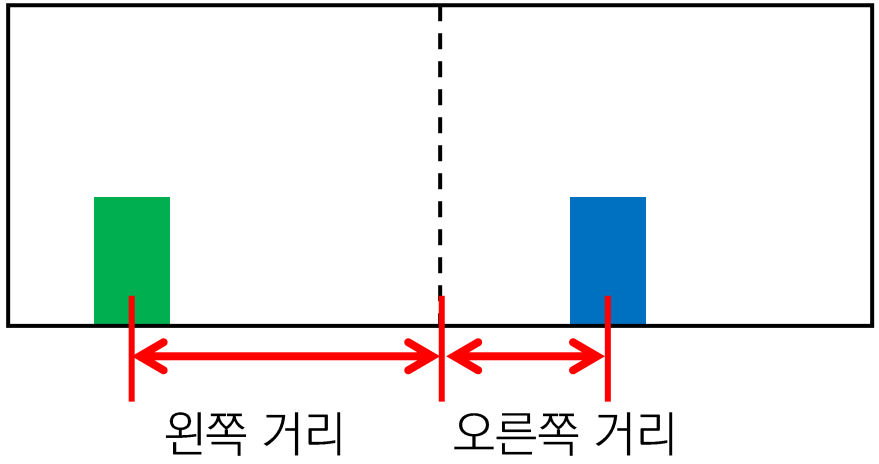


차이 = 오른쪽 거리 – 왼쪽 거리 〉 0

자율주행



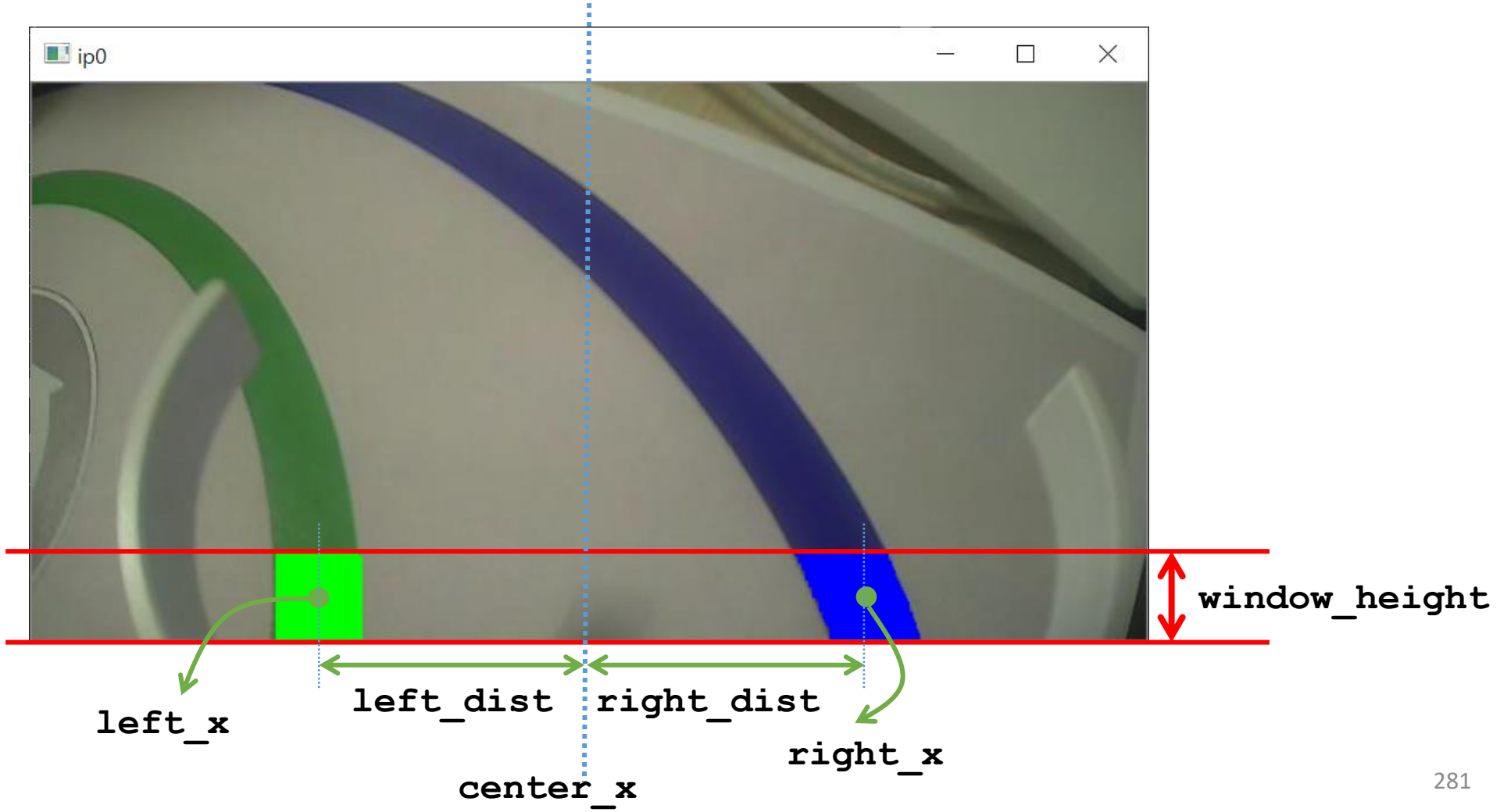
- 차이 = 오른쪽 거리 – 왼쪽 거리
- 차이를 기존 주행 속도에 더하고(왼쪽 바퀴) 빼게(오른쪽 바퀴) 되면 차이가 작아질 수록, 오른쪽 바퀴의 속도가 왼쪽보다 빨라져 왼쪽으로 회전하는 형태로 주행하게 된다.



차이 = 오른쪽 거리 – 왼쪽 거리 〈 0

자율주행

- 트랙 검출 관련 클래스와 메소드는 11차시(영상처리 준비)를 참고하세요.



02 자율주행

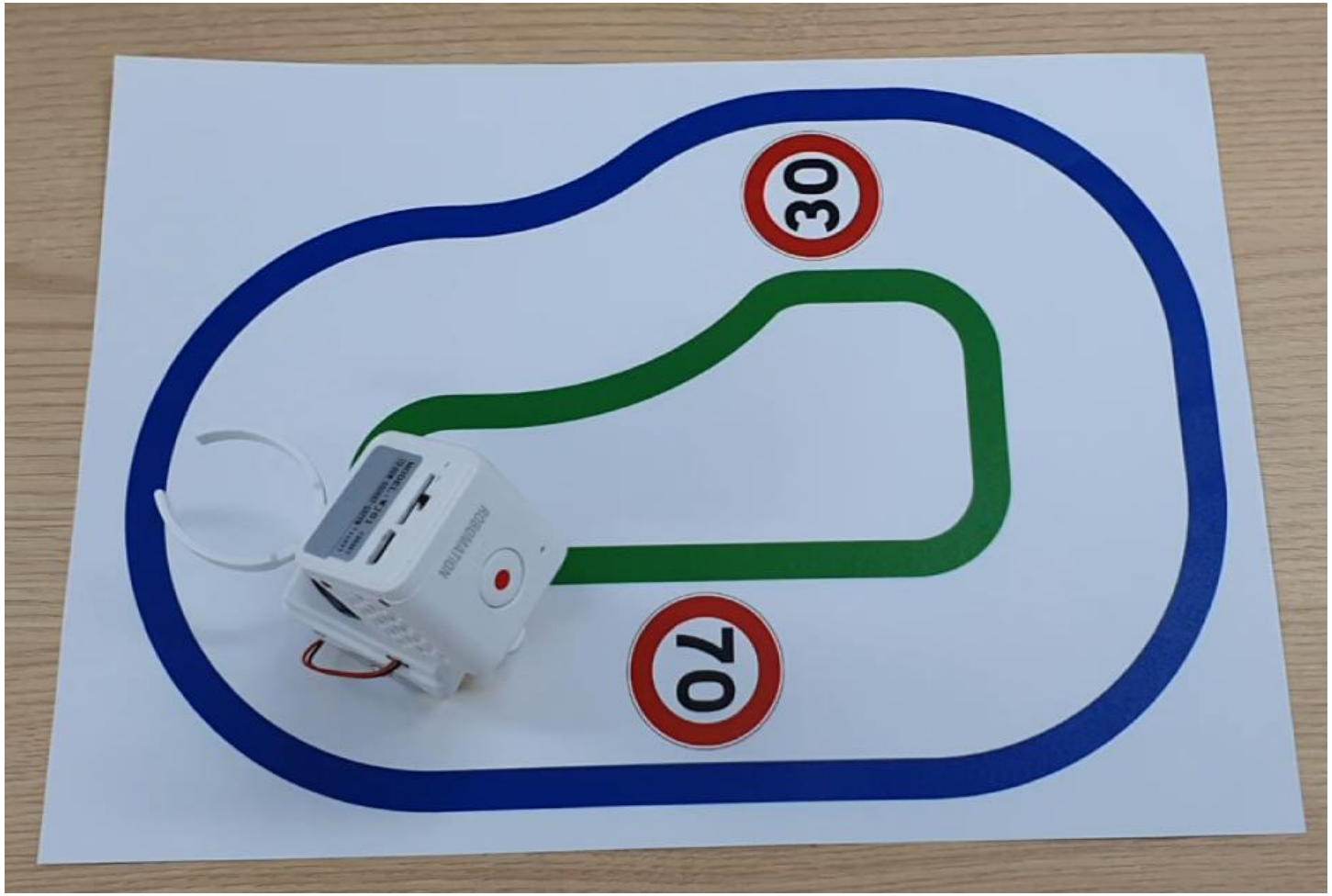
- 자율 주행

```
from roboid import *
import roboidai as ai
import roboidai.lab as lab

cam = ai.Camera('ip0') #외장 카메라로 사용한다.
hamster = HamsterS()

velocity = -70 # 후진 주행이 원칙이므로 기본 속도를 음수로 설정한다. 양수를 입력하면 전진합니다.

def control_hamster(center_x, left_x, right_x):
#매개변수는 화면의 중심선, 왼쪽 선, 오른쪽 선
    left_dist = abs(center_x - left_x) # 화면 중심 선과 왼쪽에서 인식 되는 선 사이의 절대 거리
    right_dist = abs(center_x - right_x) # 화면 중심 선과 오른쪽에서 인식 되는 선 사이의 절대 거리
    diff = right_dist - left_dist # 오른쪽 거리와 왼쪽 거리의 "차이 " 를 diff에 저장
    hamster.wheels(velocity + 0.1* diff, velocity -0.1*diff)
    # 왼쪽 바퀴와 오른쪽 바퀴의 속도를 -70에서 차이의 10%만 추가하여 설정한다.
while True:
    image = cam.read()
    if image is not None:
        width = image.shape[1]
        height = image.shape[0]
        output = image.copy()
        left_x, _ = lab.find_green_track_xy(image,output,window_height=50) # 영상에서 초록색 Blob을 찾아 무게중심을 반환합니다.
        right_x, _ = lab.find_blue_track_xy(image,output,window_height=50) # 영상에서 파란색 Blob을 찾아 무게중심을 반환합니다.
        if left_x < 0: left_x = 0
        # 초록색 Blob의 무게중심이 음수면 0으로 초기화
        if right_x < 0: right_x =width
        # 파란색 Blob의 무게중심이 음수면 0으로 width로 초기화
        control_hamster(width//2, left_x, right_x)
        # 함수 실행
        cam.show(output)
        if cam.check_key(10) == 'esc': break


hamster.stop()
wait(500)
```

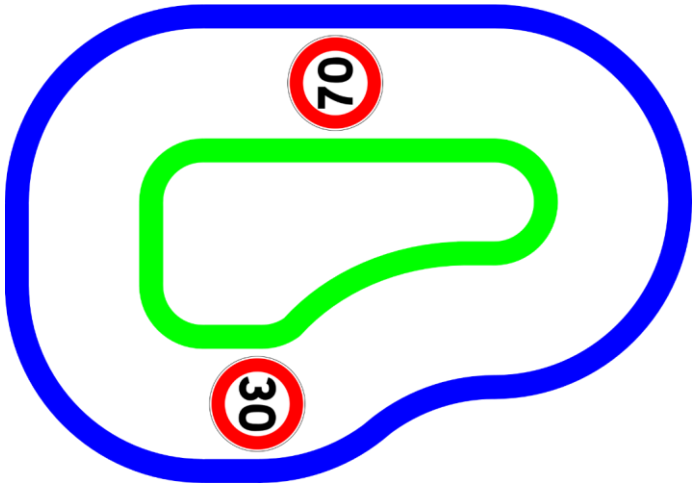
02

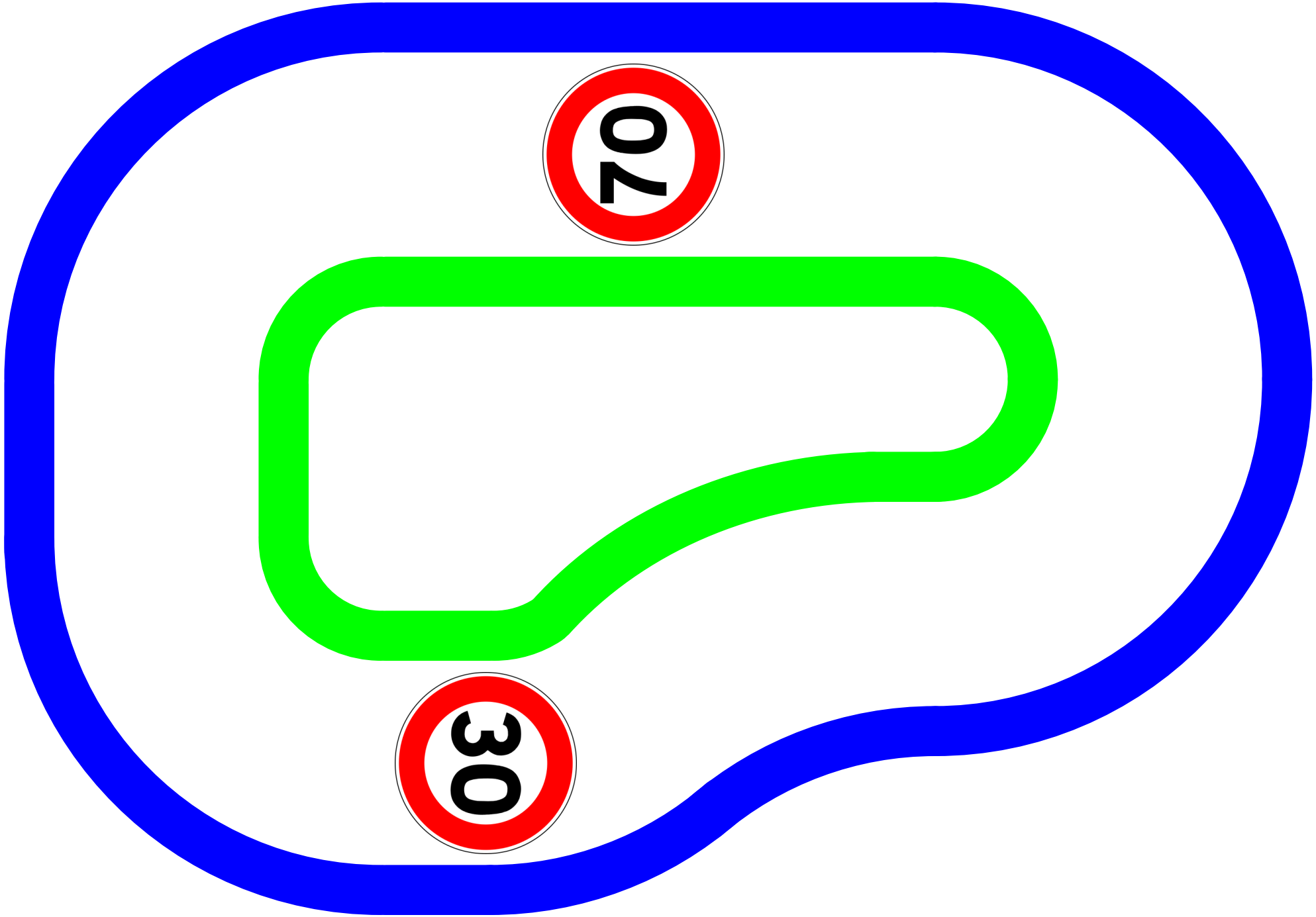
자율주행

- 속도 감지 자율 주행 자동차



- 준비물
  - 활동지
  - 햄스터 AI 카메라

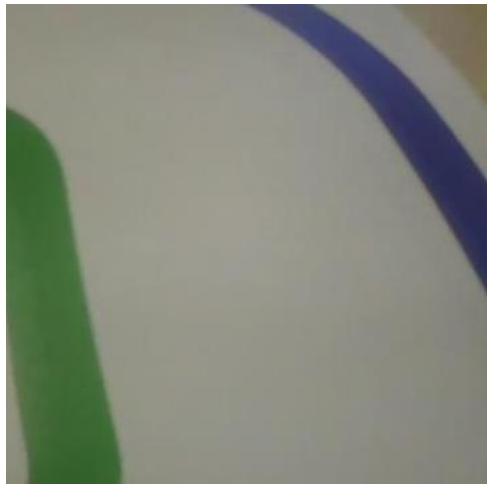



07
30

자율주행

- 속도 감지 자율 주행 자동차
  - 이미지 학습은 16차시의 티처블 머신 사용법 참고하세요!
  - 직접 훈련한 모델을 사용하여 코드를 작성하세요.

- 학습 시킬 이미지: 트랙, 속도 30 시그널, 속도 70 시그널



02 자율주행

- 속도 감지 자율 주행 자동차

```
from roboid import *
import roboidai as ai
import roboidai.lab as lab

cam = ai.Camera('ip0') #외장 카메라 사용
hamster = HamsterS()

tmi = ai.TmImage() # 티처블 머신 이미지 프로젝트 객체 생성
tmi.load_model('c:/Example/speed')
velocity = -50 # 후진 주행이 원칙이므로 기본 속도를 음수로 설정한다.
VEL_LOW = -30
VEL_HIGH = -70

def control_hamster(center_x, left_x, right_x):
    left_dist = abs(center_x - left_x)
    right_dist = abs(center_x - right_x)
    diff = right_dist - left_dist
    hamster.wheels(velocity + 0.1* diff, velocity -0.1*diff)

cam.count_down(3)
Candidates = []
while True:
    image = cam.read()
    if image is not None:
        width = image.shape[1] #햄스터 카메라는 320x640의 이미지 크기
          # width를 320의 크기로 저장
        height = image.shape[0] # height를 640의 크기로 저장
        output = image.copy()
```

티처블 머신에서 다운로드하여 압축을 푼 모델 파일(keras_model.h5, labels.txt)이 있는 폴더

로봇이 너무 흔들리면 0.1을 더 작은 값으로 줄이세요

```
        left_x, _ =
lab.find_green_track_xy(image,output,window_height
=50) #영상에서 초록색 Blob을 찾아 무게중심을 반환합니다.
        right_x, _ =
lab.find_blue_track_xy(image,output,window_height=
50) #영상에서 파란색 Blob을 찾아 무게중심을 반환합니다.
        _, red_y =
lab.find_red_track_xy(image,output,min_area=1000)
#빨간 색 stop 시그널은 빨간 색 물체 인식의 y좌표를 사용합니다.
 # min_area인 1000보다 작으면 인정하지 않습니다.
        if left_x < 0: left_x = 0
        if right_x < 0: right_x =width
        if red_y > height //2:
            if tmi.predict(image):
                label = tmi.get_label()
                if label =='30':
candidates.append(VEL_LOW) # 인식된 label이 30일 때
                elif label=='70':
 # VEL_LOW를 candidates에 추가
candidates.append(VEL_HIGH) #인식된 label이 70일 때 -70추가
# VEL_HIGH를 candidates에 추가
        elif len(candidates) >0:
            print(candidates)
            velocity= VEL_HIGH if
 # elif문에서 추가한 candidates리스트를 가지고 속도를 바꾸기 때문에
  red_y 〉height//2가 만족되는 동안에는 계속 기본 속도로 주행하게
됩니다.
```

02 자율주행

- 속도 감지 자율 주행 자동차

```
candidates.count(VEL_HIGH) > candidates.count(VEL_LOW) else VEL_LOW
# candidates의 원소 중 VEL_HIGH가 VEL_LOW의 수 보다 많을 경우 속도를 VEL_HIGH로
# 아닐 경우 VEL_LOW로 설정
            candidates=[] # candidates 초기화
            print('velocity to', velocity)
        control_hamster(width//2, left_x, right_x)
        cam.show(output)
        if cam.check_key(10) == 'esc': break # esc키 클릭 시, 프로그램 종료


hamster.stop()
wait(500)
```

감사합니다.

ROBOMATION
MAKE THINGS ALIVE