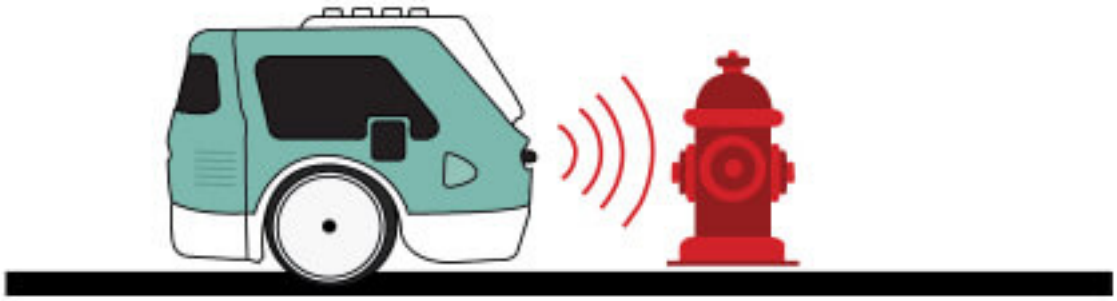
장애물 회피(전면 IR)

자율주행차는 긴급 상황에서 장애물을 피할 수 있어야 합니다.

적외선 대신 **LIDAR**(Light Imaging Detection and Ranging) 및 **RADAR**와 같은 고급 센서를 사용합니다.
LIDAR는 레이저 광을 사용하여 환경을 매핑하는 반면 RADAR는 전파를 사용합니다.
이 기술 없이도 Zumi는 여전히 장애물을 피할 수 있습니다!



라이브러리 가져오기

In [ ]:

```
from zumi.zumi import Zumi
import time

zumi = Zumi()
```

보행자를 위한 정지

자율주행차가 전방에서 위험을 감지하는 것은 매우 중요합니다!

여기에서는 Zumi가 전면 왼쪽 또는 전면 오른쪽 IR 센서로 여러분의 손을 감지할 때까지 `forward_step()`을 사용하여 앞으로 운전하도록 Zumi를 코딩합니다. 그리고 다시 손을 떼면 Zumi가 계속 주행해야 합니다.

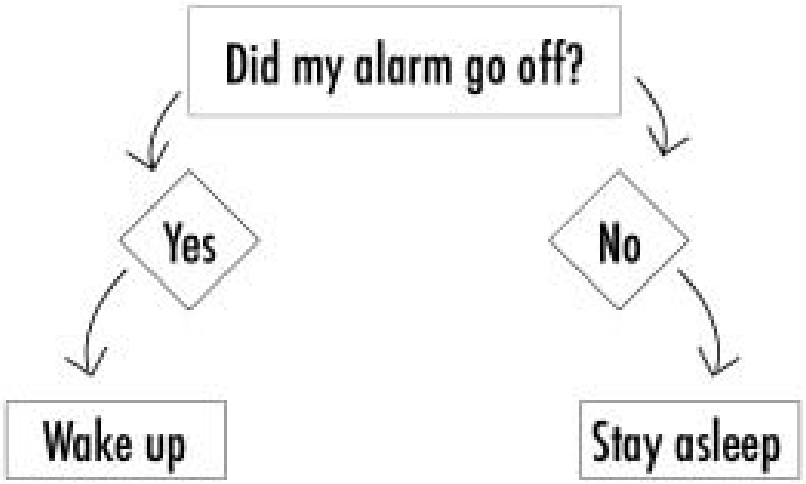
이제부터 주미는 스스로 판단하고 결정을 내려야할 때가 온 것입니다.

논리

논리란 무엇인가요? 여러분은 누군가가 논리적이고 이성적이라는 것을 들어본 적이 있을 것입니다.
논리는 올바르게 추리해가는 과정과 원리입니다.
우리가 결정을 내릴 때, 수학을 할 때, 또는 비교와 대조를 할 때, 우리는 상황에 근거하여 최선의 해결책을 찾습니다.

조건문
If 문 또는 조건문은 코드 내에서 결정을 내리는 데 사용됩니다.
우리는 실제로 조건문을 매일 사용합니다!

- 만약 알람이 울리면, 나는 일어날 거야. 그렇지 않으면 나는 계속 잠을 잘 것이다. 배가 고프다면 뭔가를 먹을 거야.
- 밖에 햇볕이 쨍쨍 내리쬔다면 선크림을 발라야겠다.
- 내 방이 깨끗하다면 용돈을 받을 거야. 그렇지 않으면 나는 용돈을 받지 못할거야.



알람은 깨어나기 위해 참이어야 하는 **조건**입니다. 그렇지 않고 조건이 거짓이면 프로그램은 다른 것을 실행합니다.
파이썬에서 if 문은 다음과 유사한 형식을 따릅니다.

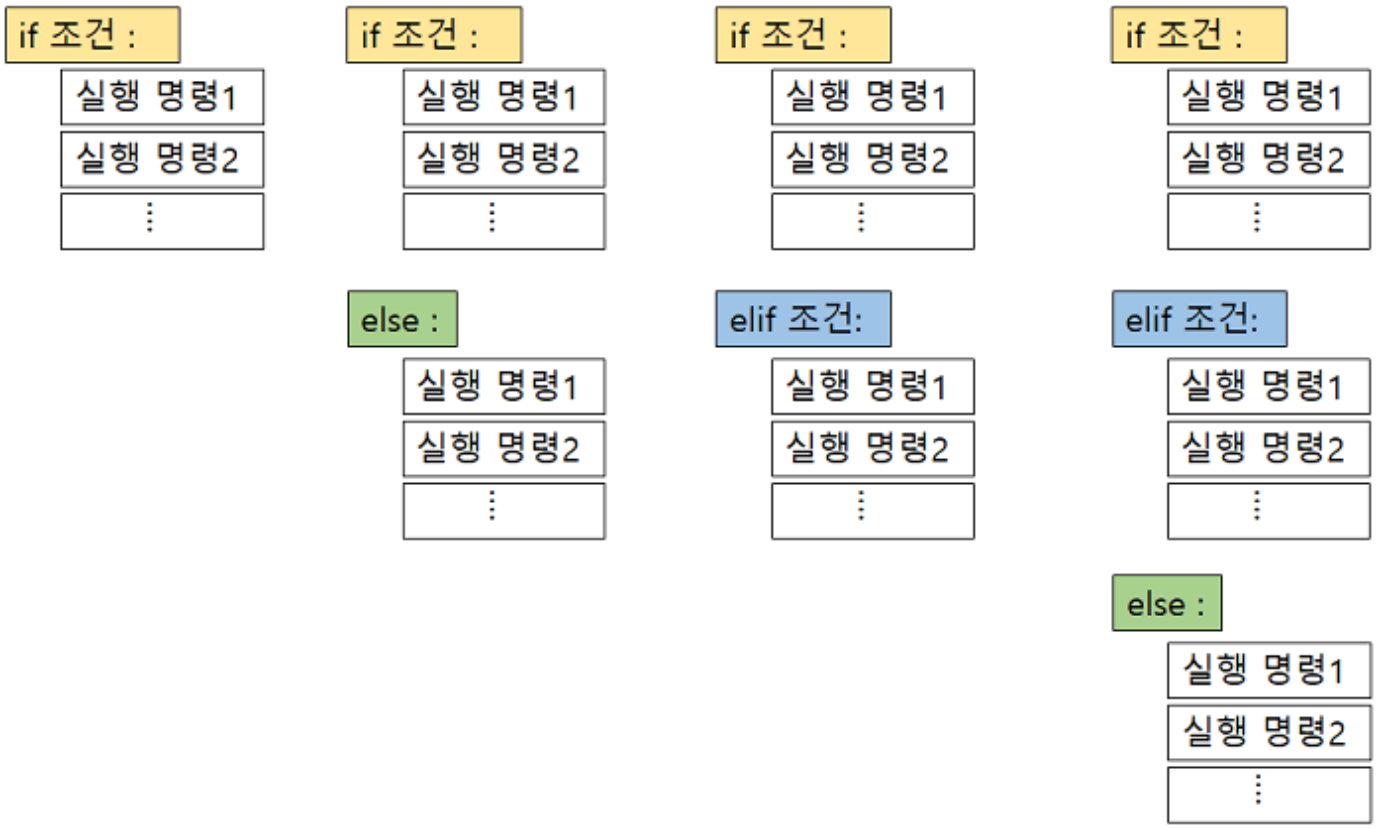
```
알람이  울리면:
      일어나겠습니다
기타:
      저는  계속  자겠습니다
```

우리가 논리적인 근거를 바탕으로 판단을 하는 매 순간 상황이 모두 if 조건문이라고 할 수 있습니다.
우리가 조건문을 사용한다는 것을 이제 아셨을 것입니다.

이제 우리는 주미 혹은 컴퓨터에서는 어떻게 조건문을 사용하는지, 어떤 방식으로 사용하는지만 알면 됩니다.

---

if문
파이썬에서 조건문의 구조는 다음과 같습니다. 살펴볼까요?



if-else 구조

- 2가지의 가능한 결과를 판단 할 수 있습니다.
  조건이 참이면 if문 바로 다음 명령들을 실행합니다.
  조건이 거짓이면 else문 다음 명령들을 실행합니다.
  else문은 if문 없이 사용할 수 없습니다.

if-elif 구조

- 3개 이상의 가능한 결과를 원하는 경우에 사용합니다.
  조건이 참이면 if문 바로 다음 명령들을 실행합니다.
  조건이 거짓이면 다음 elif문의 조건을 확인하고 참이면 바로 다음 명령들을 실행합니다.
  elif문의 조건이 거짓이고, 다음 elif문이 더있다면 조건을 다시 확인합니다.

if-elif-else 구조

- 조건이 참이면 if문 바로 다음 명령들을 실행합니다.
  조건이 거짓이면 다음 elif문의 조건을 확인하고 참이면 바로 다음 명령들을 실행합니다.
  elif문의 조건이 거짓이고, 다음 elif문이 더 있다면 조건을 다시 확인합니다.
  elif문이 더이상 없고 else 문으로 이동한다면 else문 다음 명령들을 실행합니다.

기억해야 할 몇 가지 if 조건문의 규칙들:

- 하나의 조건만 확인하는 경우 조건이 참이면 **If** 아래에 들여쓰기 된 명령을 수행합니다.
  그렇지 않으면 **else** 또는 다음 조건문으로 넘어갑니다.

- **else**에는 조건이 없습니다. **if-else** 문을 사용할 때 **else**가 마지막 옵션이기 때문입니다.
  컴퓨터가 다른 모든 조건을 확인한 후 **else** 옵션에 도달하면 **else** 조건은 자동으로 참(true)이 되며, 해당 명령을 실행합니다.

- **else**를 사용할때는 **if** 조건이 거짓인 경우, 어떤 명령을 실행해야 할 때입니다.
  if 조건이 거짓일 때, 아무 명령도 실행하지 않는다면 **else**를 사용할 필요가 없습니다.

if문 예제

아래 코드 셀을 실행하기 전에 변수 x와 y의 값을 확인하세요.
출력이 어떻게 될지 예측할 수 있습니까?

In [ ]:

```
x = 4
y = 3

if x > y:
    print("x 가 y 보다 큽니다.")
```

In [ ]:

```
x = 4
y = 3

if x > y:
    print("x 가 y 보다 큽니다.")
else :
    print("x 가 y 보다 크지 않습니다")
```

다음 셀에서는 x와 y 값을 바꿔가며 실행해봅시다. 어떻게 바뀌나요?

In [ ]:

```
x = 10
y = 30

if x > y:
    print("x 가 y 보다 큽니다.")
elif x == y:
    print("x 와 y가 같습니다.")
else:
    print("x 가 y 보다 작습니다.")
```

조건문의 순서

조건문에서 순서가 중요할까요?
다음 예제를 실행해 봅시다.

In [ ]:

```
sensor = 20
if(sensor < 40):
    print("sensor가 40보다 작습니다.")
elif(sensor < 20):
    print("sensor가 20보다 작습니다.")
```

In [ ]:

```
sensor = 20
if(sensor < 20):
    print("sensor가 40보다 작습니다.")
elif(sensor < 40):
    print("sensor가 20보다 작습니다
```

비교연산자

x,y를 비교할 때 쓰인 기호를 비교연산자라고 합니다.
이 등호는 수학시간에 봐서 친숙해 보일지 모르지만, 코드에서는 약간 다른 사용 방식을 가지고 있습니다.
비교연산자는 조건 내에서 값을 비교하는 데 사용할 수 있습니다.
조건문이 참이면 그 아래의 문장이 실행됩니다.

- > 보다큼
- < 보다작다
- >= 보다크거나같음
- <= 보다작거나같음
- == 같음
- ! = 같지않음

논리 연산자

여러개의 연산자를 동시에 비교해야하는 경우에 논리 연산자를 사용합니다.

- and 연산자 : and 연산은 조건 A 와 조건 B가 모두 참이면 참(True)을 반환합니다.
- or 연산자 : or 연산은 조건 A, B 둘중 하나라도 참(True) 이면 참(True)이 나오게 됩니다.
- not 연산자 : not 연산은 조건을 반대로 반환합니다.

- and 연산
  True and True => True
  True and False => False
  False and True => False
  False and False => False
- or 연산
  True or True => True

True or False => True
False or True => True
False or False => False

- not 연산
  not True => False
  not False => True

다음 예제를 실행해보고 x와 y 값을 바꿔가며 실행해봅시다. 어떻게 바뀌나요?

In [ ]:

```
x = 10
y = 20

if x == 10 and y == 20:
    print("x와 y의 비교식 값이 모두 참이므로 결과는 참입니다.")
else:
    print("x와 y의 비교식 값에 거짓이 있으므로 결과는 거짓입니다.")
```

In [ ]:

```
x = 10
y = 20

if x == 10 or y == 20:
    print("x와 y의 비교식 값 중에 참이 있으므로 결과는 참입니다.")
else:
    print("x와 y의 비교식 값이 모두 거짓이므로 결과는 거짓입니다.")
```

In [ ]:

```
x = 10

if not x == 10:
    print("x의 비교식 값이 거짓이므로 결과는 참입니다.")
else:
    print("x의 비교식 값이 참이므로 결과는 거짓입니다.")
```

장애물 회피1 (전면 오른쪽 IR)

우리의 목표는 Zumi의 센서가 감지되었을 때 혹은 그렇지 않은 경우에 작동하는 코드를 작성하는 것입니다.

즉, **주미는 *만약에(if)* ir_readings[0](정면 오른쪽 센서) 값이 100 보다 작을 때만 작동하려고 합니다.**

파이썬으로 코드를 작성하기 전에 사람이 이해할 수 있는 일반 언어로 작성된 프로그램인 의사 코드를 작성하는 것이 도움이 됩니다.
아래 의사 코드를 보고 작동 내용을 확인하세요.

센서 확인
정면 오른쪽 센서 값이 100보다 작다면 정지
그렇지 않다면 주행
300회 반복

for 반복문, forward_step() 및 get_all_IR_data()을 사용하여 Zumi가 앞으로 나아갈 수 있도록 아래 코드를 작성하세요.
!! Try and Finally 문을 사용하여 모터가 계속 회전하는 것을 방지합니다.

In [ ]:

```
# 주미의 각도를 0으로 설정하기 위해 자이로 센서를 초기화
zumi.reset_gyro()
try:
    for x in range(300):
        ir_readings = zumi.get_all_IR_data()
        front_right_ir = ir_readings[0]
        #  나머지 조건문을 작성해주세요

finally:
    zumi.stop()
print("Done!")
```

해결 방안

해결 방안을 보려면 클릭하세요!

```
# 주미의 각도를 0으로 설정하기 위해 자이로 센서를 초기화
zumi.reset_gyro()
try:
    for x in range(300):
        ir_readings = zumi.get_all_IR_data()
        front_right_ir = ir_readings[0]
        if front_right_ir < 100:
            zumi.stop()
        else:
            zumi.forward_step(40, 0)
finally:
    zumi.stop()
print("Done!")
```

장애물 회피2 (전면 오른쪽 IR + 전면 왼쪽 IR)

이번에는 전면 왼쪽 또는 전면 오른쪽 IR 센서에 물체가 감지되면 정지하고, 감지되지 않으면 주행하는 코드를 작성하려고 합니다.
마찬가지로 forward_step()을 사용하여 앞으로 운전하도록 Zumi를 코딩합니다.

의사 코드

이번에는 이 미션의 의사코드를 작성하게 됩니다!
아래 셀에 의사 코드의 각 줄을 **주석**으로 작성합니다. 줄 시작 부분에 # 기호(#)를 넣으면 됩니다.
컴퓨터는 이를 무시하기때문에, 코드와 함께 설명을 제공하는 좋은 방법입니다.

In [ ]:

```
# 여기에 작성하세요.
```

해결 방안

해결 방안을 보려면 클릭하세요!

```
# 전방 왼쪽과 오른쪽의 센서 값을 읽습니다.
# 전방 왼쪽과 오른쪽의 센서 값이 100보다 작다면 주미가 정지합니다.
# 전방 왼쪽과 오른쪽의 센서 값이 100보다 작지 않다면 주미는 전진합니다.
# 300번 반복합니다.
```

의사 코드를 통해 각 코드 줄을 파이썬으로 변환하고 아래 for 반복문을 채우세요.
양쪽 두개의 센서를 사용해야하므로 **or** 논리 연산자를 사용해야 합니다.

In [ ]:

```
zumi.reset_gyro()
try:
    for x in range(300):
        # 여기에 코드를 쓰세요.

finally:
    zumi.stop() # 주미가 정지해야 합니다.
    print("완료!")
```

해결 방안

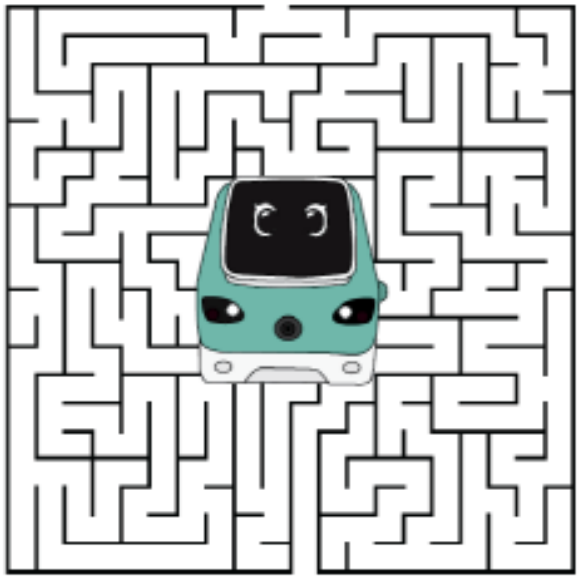
해결 방안을 보려면 클릭하세요!

```
# 주미의 각도를 0으로 설정하기 위해 자이로 센서를 초기화
zumi.reset_gyro()
try:
    for x in range(300):
        ir_readings = zumi.get_all_IR_data()
        front_right_ir = ir_readings[0]
        front_left_ir = ir_readings[5]
        if front_right_ir < 100 or front_left_ir < 100:
            zumi.stop()
        else:
            zumi.forward_step(40, 0)
finally:
```

```
    zumi.stop()
print("완료!")
```

주미 미로 탈출

충돌 회피는 모든 자동차의 필수 기능이지만 자율 주행 자동차는 장애물을 우회해야 합니다.
이 섹션에서는 주미가 장애물을 만났을 때 방향을 바꾸는 방법에 대해 알아봅니다.



이번에는 멈추고나서 장애물이 사라지기를 기다리는 대신에, 어느쪽 센서가 감지했는지에 따라 왼쪽이나 오른쪽으로 회전하려고 합니다.

heading 초기화

프로그램 시작 부분에서 heading 변수를 0으로 설정합니다.
매개변수에 보통 내가 회전할 각도 또는 방향을 입력하지만, 여기서는 `heading` 변수를 사용합니다.
변수를 사용하는 이유는 주미를 방해하는 장애물에 따라 방향이 계속 바뀌기 때문입니다!

```
heading=0
```

모든 프로그램에서 heading을 0으로 설정하여 곧바로 시작할 수 있도록 하는 것이 매우 중요합니다!

heading이 0인 상태로 `forward_step()` 을 사용하여 다음을 실행해봅시다.

In [ ]:

```
heading = 0
zumi.reset_gyro() # 리셋 자이로를 실행하면 현재 주미가 바라보는 방향의 각도가 0이 됩니다.

for i in range(40):
    zumi.forward_step(20, heading) #  heading 방향으로 이동합니다.

zumi.stop()
```

현재 바라보는 방향으로 주미가 직진하는 걸 확인하였나요.

heading의 값이 현재 각도인 0도이기 때문입니다.

heading 변경

`turn_left()` 또는 `turn_right()`를 호출하는 대신 Zumi의 각도를 더하거나 빼서 heading을 변경하도록 해봅시다.

heading 값을 변경하는 두개의 예제를 실행해보고 차이를 눈으로 확인해봅시다.
먼저 heading 값에 90을 더해봅시다. 어떻게 움직이나요?

In [ ]:

```
heading = 0
zumi.reset_gyro() # 리셋 자이로를 실행하면 현재 주미가 바라보는 방향의 각도가 0이 됩니다.

for i in range(40):
    zumi.forward_step(20, heading)

heading = heading + 90

for i in range(40):
    zumi.forward_step(20, heading)

zumi.stop()
```

이번에는 heading 값에 90을 빼봅시다. 어떻게 움직이나요?

In [ ]:

```
heading = 0
zumi.reset_gyro()

for i in range(40):
    zumi.forward_step(20, heading)

heading = heading - 90

for i in range(40):
    zumi.forward_step(20, heading)

zumi.stop()
```

heading에 값을 더하면 왼쪽으로 회전하고, heading에서 값을 빼면 오른쪽으로 회전하는 것을 확인하였나요?.

`turn_left()` 또는 `turn_right()`를 사용하지 않고, 현재 각도를 기준으로 회전하는것을 보았을 것입니다.

우리가 작성할 프로그램의 목표는 오른쪽 센서가 감지되면 왼쪽으로, 왼쪽 센서가 감지되면 오른쪽으로 회전하는 것입니다.
만약 두 센서가 모두 감지되었다면 장애물이 주미 바로 앞에 있는 것입니다!
이 때, 주미는 막다른 골목이 될 수 있는 이 상황을 벗어나기 위해 180도를 회전해서 빠져나와야 합니다.

다음과 같이 사용됩니다.

예시:

```
front_right_ir < 100인 경우:
    heading = heading + 30
```

오른쪽 센서가 감지된다는 것은 **왼쪽**으로 가는 것을 의미하므로 빼기가 아니라 **각도를 **더하기**해야 합니다. 절대 각도를 기억하는 데 도움이 필요하면 강의 자이로스코프를 다시 살펴보세요.

의사 코드

파이썬으로 코드를 작성하기 전에 사람이 이해할 수 있는 일반 언어로 작성된 프로그램인 의사 코드를 작성하는 것이 도움이 됩니다.
아래 의사 코드를 보고 작동 내용을 확인하세요.

```
heading을 0으로 설정
IR 센서 확인
전방 우측 센서가 감지되면 방향을 양수만큼 변경하여 왼쪽으로 이동
그렇지 않고 전면 왼쪽 센서가 감지되면 방향을 음수로 변경하여 오른쪽으로 이동
그렇지 않고 두 센서가 모두 감지되면 정지하고 방향을 180(양수 또는 음수)으로 변경하여 방향을 반대로 합니다..
새로운 heading 값으로 계속 직진
반복!
```

논리 연산자
코드를 작성하기 전에 세 번째 if 문에서 둘 다 감지되었는가를 확인하는지 살펴보세요.
이러한 경우에 파이썬에서는 **and** 연산자를 사용합니다!

```
front_right_ir < 100 and front_left_ir < 100인 경우:
    # 정지하고 180도 방향 변경하는 코드
```

In [ ]:

```
zumi.reset_gyro()

heading = 0

try:
    for x in range(1000):
    # 여기에 코드를 작성해주세요.

finally:
    zumi.stop()
```

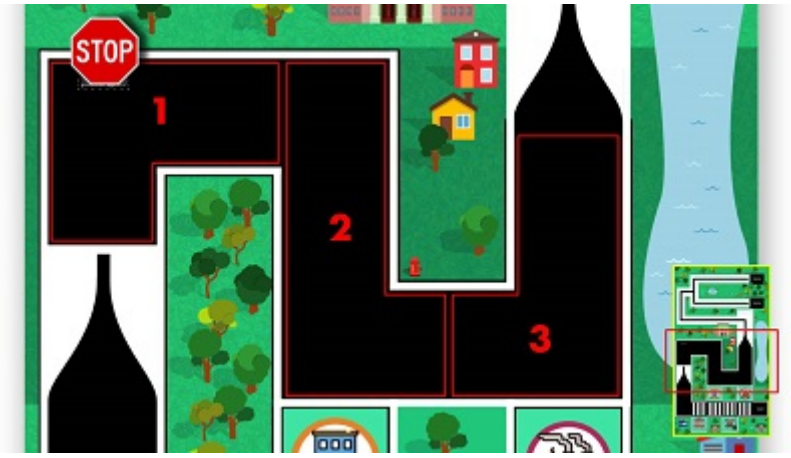
해결 방안

해결 방안을 보려면 클릭하세요!

```
zumi.reset_gyro()
heading = 0
```

```
try:
    for x in range(1000):
        ir_readings = zumi.get_all_IR_data()
        front_right_ir = ir_readings[0]
        front_left_ir = ir_readings[5]
        if front_right_ir < 100:
            zumi.stop()
            heading = heading + 30
            zumi.turn(heading,0.5)
        if front_left_ir < 100:
            zumi.stop()
            heading = heading - 30
            zumi.turn(heading,0.5)
        if front_right_ir < 100 and front_left_ir < 100:
            zumi.stop()
            zumi.reverse()
            # update the heading since we have changed heading
            heading = heading - 180
            zumi.turn(heading)
        else:
            zumi.forward_step(20, heading)
finally:
    zumi.stop()
print("완료!")
```

대회 : 어떻게 사용되나요?



대회에서 일부 구간에는 벽이 있는 미로 구간이 있습니다. 이 구간의 경우 Zumi의 IR 센서를 사용하여 통과하세요!

forward_avoid_collision() 함수

장애물을 피하는 또 다른 방법은 `forward_avoid_collision()` 함수를 사용하는 것입니다. 이 기능을 사용하면 Zumi가 지정된 속도와 시간 동안 전진합니다. 전면 IR 센서가 무언가를 감지하면 Zumi가 멈춥니다.

In [ ]:

```
# Zumi를 속도 40, 지속 시간 1초로 전진시킵니다. 전면 IR 센서로 무언가가 감지되면 Zumi가 멈춥니다.
zumi.forward_avoid_collision(speed=40, duration=1.0,left_th=100,right_th=100)
```

또 다른 유사한 기능인 `reverse_avoid_collision()`은 후면 IR 센서로 무언가를 감지할 때마다 Zumi를 멈추게 하는 데 사용할 수 있습니다.

In [ ]:

```
# Zumi를 속도 40, 지속 시간 1초로 역주행합니다. 후방 IR 센서로 무언가가 감지되면 Zumi가 멈춥니다.
zumi.reverse_avoid_collision(speed=40, duration=1.0,left_th=100,right_th=100)
```

left_th와 right_th의 값은 센서 임계 값으로 zumi 마다 다르기 때문에 가지고 있는 zumi의 센서 값이나 원하는 물체와의 거리에 따라서 다르게 입력합니다.

응용

전방 센서를 사용하여 장애물이 감지되면 정지 후 회전하는 예제입니다.

In [ ]:

```
# 센서 값을 꼭 확인하세요!!!
left_sensor_threshold = 100 # 각각의 주미에 맞는 전방 왼쪽 센서 임계 값을 입력합니다.
right_sensor_threshold = 100 # 각각의 주미에 맞는 전방 오른쪽 센서 임계 값을 입력합니다.

#주행과 관련된 센서들을 초기화 합니다.
zumi.reset_drive()


# 전진 중 장애물 감지되면 정지
zumi.forward_avoid_collision(speed=20,duration=5,desired_angle=0,left_th=left_sensor_threshold,righ

# 오른쪽으로 90도 회전
zumi.turn_right(desired_angle=90, duration=1)

# 전진 중 장애물 감지되면 정지
zumi.forward_avoid_collision(speed=20,duration=5,desired_angle=-90,left_th=left_sensor_threshold,ri

# 왼쪽으로 90도 회전
zumi.turn_left(desired_angle=90, duration=1)
```

이번에는 heading을 사용하여 이동해봅시다.

In [ ]:

```
# 센서 값을 꼭 확인하세요!!!
left_sensor_threshold = 100 # 각각의 주미에 맞는 전방 왼쪽 센서 임계 값을 입력합니다.
right_sensor_threshold = 100 # 각각의 주미에 맞는 전방 오른쪽 센서 임계 값을 입력합니다.

#주행과 관련된 센서들을 초기화 합니다.
zumi.reset_drive()

# 현재 각도를 읽습니다.
now_angle = zumi.read_z_angle()

# 전진 중 장애물 감지되면 정지
now_angle,time_elapsed = zumi.forward_avoid_collision(speed=40, duration=1.0, desired_angle=now_ang
time.sleep(1) # 1초 기다리기

# 제자리에서 오른쪽으로 회전
now_angle = now_angle - 90  # 목표 각도를 현재위치에서 90도를 뺀 각도로 변경
zumi.turn(now_angle,1,25,0) # 목표각도로 오른쪽으로 회전

# 전진 중 장애물 감지되면 정지
now_angle,time_elapsed = zumi.forward_avoid_collision(speed=40, duration=1.0, desired_angle=now_ang
time.sleep(1) # 1초 기다리기

# 제자리에서 왼쪽으로 회전
now_angle = now_angle + 90  # 목표 각도를 현재위치에서 90도를 더한 각도로 변경
zumi.turn(now_angle,1,25,0) # 목표각도로 왼쪽으로 회전
```