1. Teacher.py

#!/usr/bin/env python (shebang라인)

#/usr/bin에 있는 env는 인자로 넘어온 python파일을 동적으로 찾아서 실행해준다. 결과적으로 스크립트가 Python 스크립트로 실행되는지 확인한다.

import rospy

import는 다른 위치에 있는 라이브러리를 사용할 수 있게 선언해준다.

#rospy는 Python 용 ROS 클라이언트 라이브러리이다. 이 라이브러리가 import 되어야 ROS 노드를 작성할 수 있게 된다.

from std_msgs.msg import String

std_msgs.msg 는 게시를 위해 std_msgs/String 메시지 유형을 재사용할 수 있게 한다.

rospy.init_node('teacher')

#teacher이라는 이름을 갖는 노드를 생성한다.

pub = rospy.Publisher('my_topic', String)

my_topic이라는 이름의 토픽을 발행하며 그 안에 들어갈 데이터는 String 타입이다.

rate = rospy.Rate(2)

#아래 루프에서 1초에 2번 반복할 수 있도록 설정한다.(0.5초)

while not rospy.is_shutdown():

#강제 종료하기 전까지 루프를 돌린다.

    pub.publish('call me please')

'call me please'라는 값을 토픽에 담아 발행한다.

    rate.sleep()

#0.5초 지날 때까지 잠깐 쉰다.

2. Student.py

#!/usr/bin/env python

#/usr/bin에 있는 env는 인자로 넘어온 python파일을 동적으로 찾아서 실행해준다.

import rospy

import는 다른 위치에 있는 라이브러리를 사용할 수 있게 선언해준다.

#rospy는 Python 용 ROS 클라이언트 라이브러리이다. 이 라이브러리가 import 되어야 ROS 노드를 구현할 수 있게 된다.

from std_msgs.msg import String

std_msgs.msg 는 게시를 위해 std_msgs/String 메시지 유형을 재사용할 수 있게 한다.

def callback(msg):

    print msg.data

#새로운 메시지를 수신 받으면 그 메시지의 데이터를 출력한다

rospy.init_node('student')

#student라는 이름을 갖는 노드를 생성한다.

sub = rospy.Subscriber('my_topic', String, callback)

#my_topic이라는 토픽을 구독하며 토픽 안에 담긴 데이터는 String이다. 토픽이 도착하면 callback 함수를 호출한다.

rospy.spin()

종료될 때까지 계속 기다리게 한다.

3. Teacher.py와 Student.py의 동작 결과

Teacher.py 노드가 my_topic이라는 이름의 topic에 “call me please” 문자열을 담아 보내면, student 노드가 해당 topic을 받아 그 안에 담긴 문자열을 꺼내 화면에 출력한다.