

ROS2 강습회

대한기계학회 IT융합부문

명지대학교 최동일

dongilc@mju.ac.kr

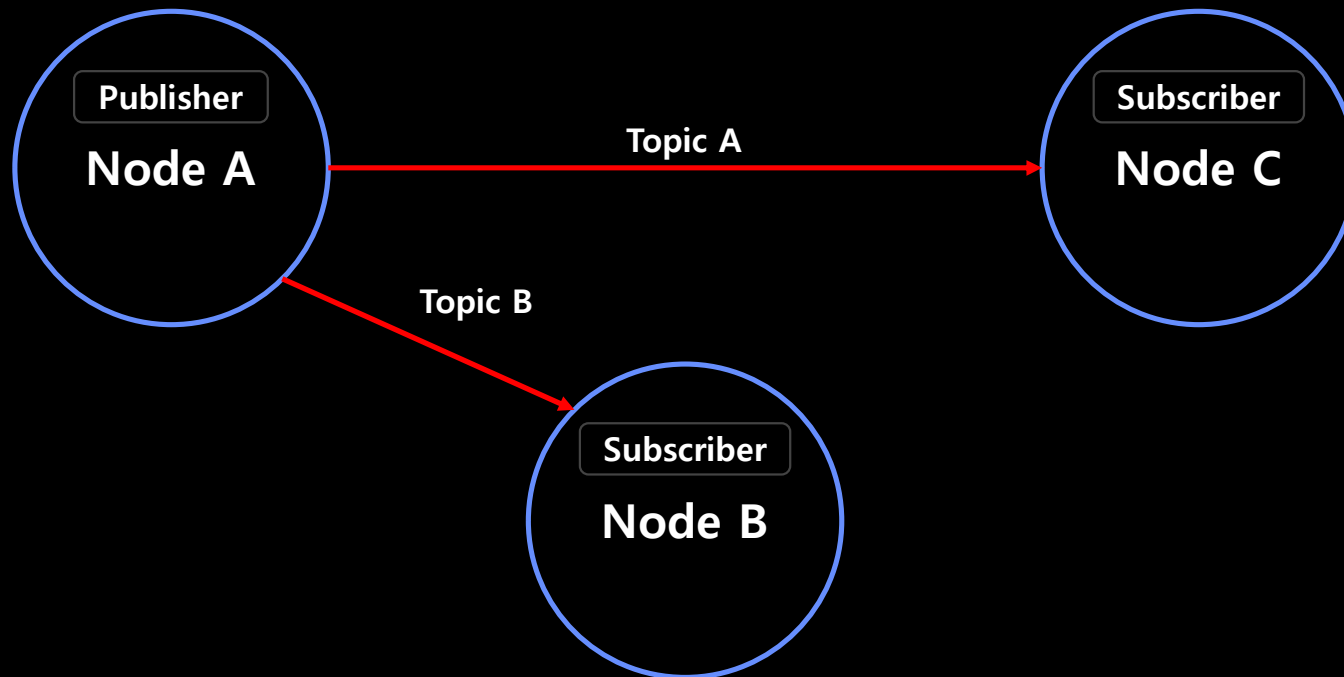
2023. 08. 11

3교시

파이썬으로 ROS2 토픽 다루기 실습

토픽(topic)

- 비동기식 단방향 메시지 송수신 방식
- Publisher와 Subscriber 간의 통신
- Publisher: msg 인터페이스 형태의 메시지를 발행
- Subscriber: 메시지를 구독



파이썬 설치 및 준비 (ubuntu 에서 설치시)

```
sudo apt install python3-pip
```

- pip: Python에서 모듈을 관리하는 관리자

```
pip3 install --upgrade pip
```

```
pip3 install jupyter ipywidgets pyyaml bqplot
```

- pip로 jupyter, ipywidegets, pyyaml, bqplot을 설치
- 모듈 설치가 끝났으면 재부팅

```
jupyter notebook
```

- 재부팅 후 에러가 난다면 아래 명령 실행 후 다시 재부팅

```
sudo apt install jupyter-core
```

Jupyter의 기본 사용

- jupyter에서 만들고 실행하는 코드는 python이라는 폴더에 넣어 두는 것으로 함

```
mkdir python
```

```
cd python
```

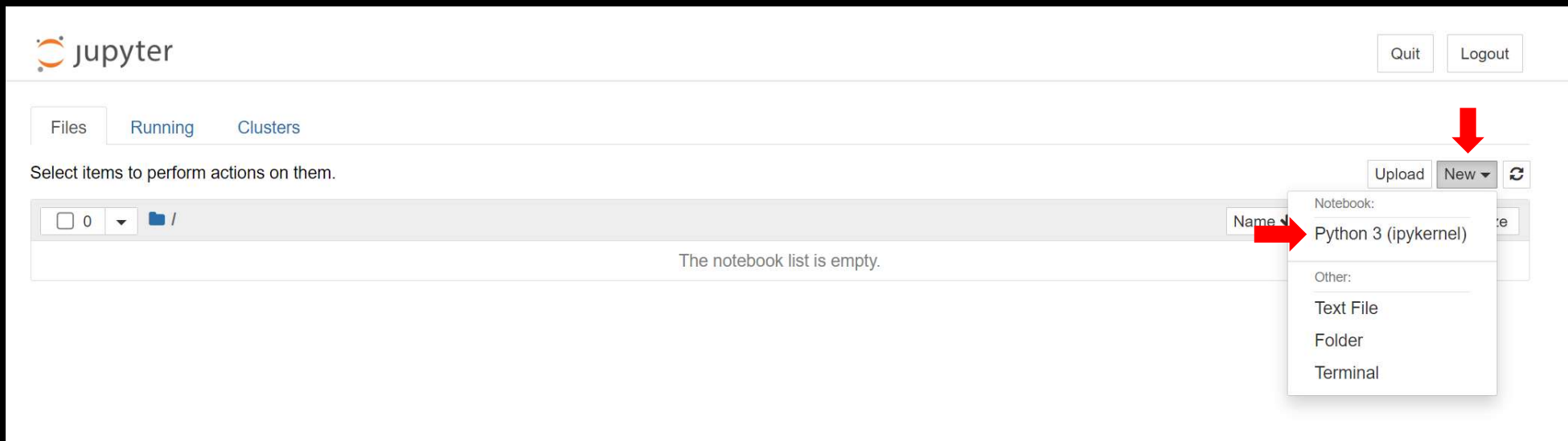
```
pw@pinklab:~$  
pw@pinklab:~$ ls  
catkin_ws  Documents  Music      Public     Videos  
Desktop    Downloads  Pictures   Templates  
pw@pinklab:~$  
pw@pinklab:~$ mkdir python  
pw@pinklab:~$  
pw@pinklab:~$ cd python/  
pw@pinklab:~/python$  
pw@pinklab:~/python$
```

```
jupyter notebook
```

- jupyter에서 사용하는 코드는 확장명이 ipynb

Jupyter의 기본 사용

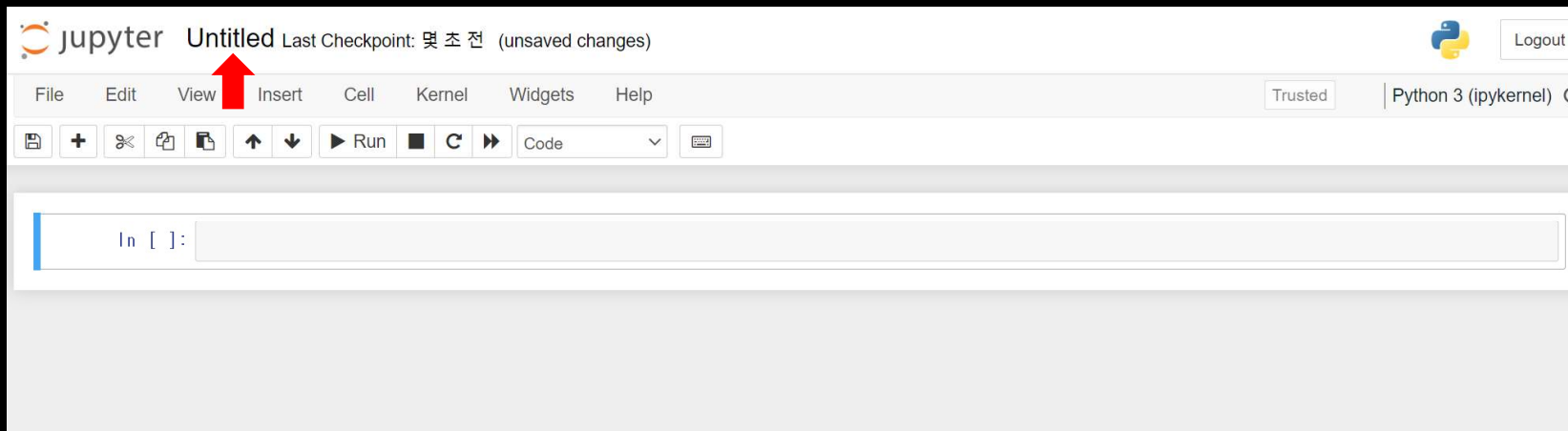
- 새로 만든 폴더에서 jupyter를 실행했기 때문에 아래와 같이 빈 화면



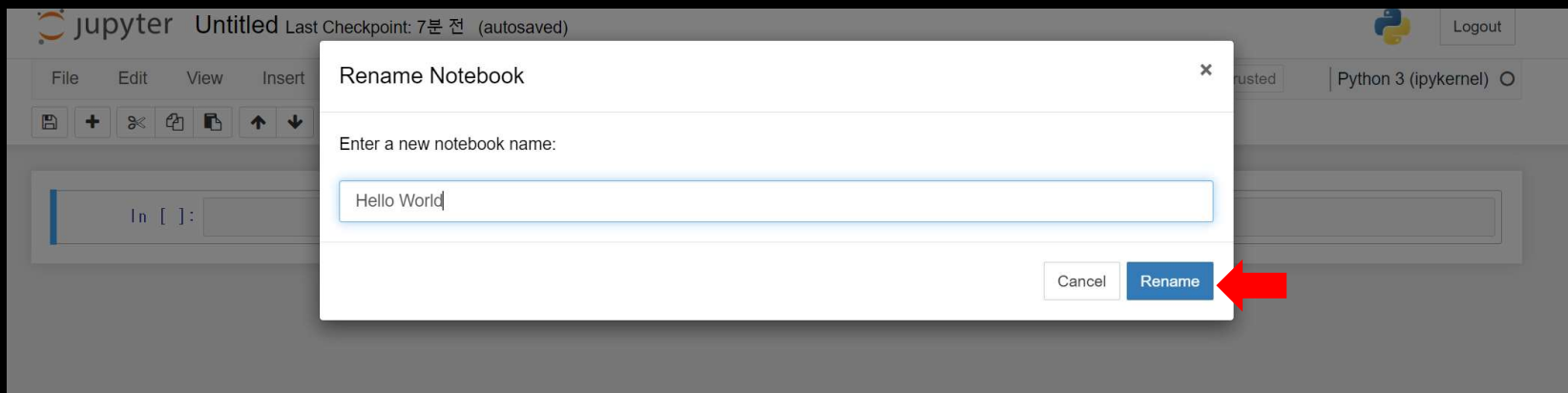
- 화면 우측 상단의 New라는 버튼을 누르고 Python3를 클릭

Jupyter의 기본 사용

- 코드를 입력할 수 있는 빈 화면 생성

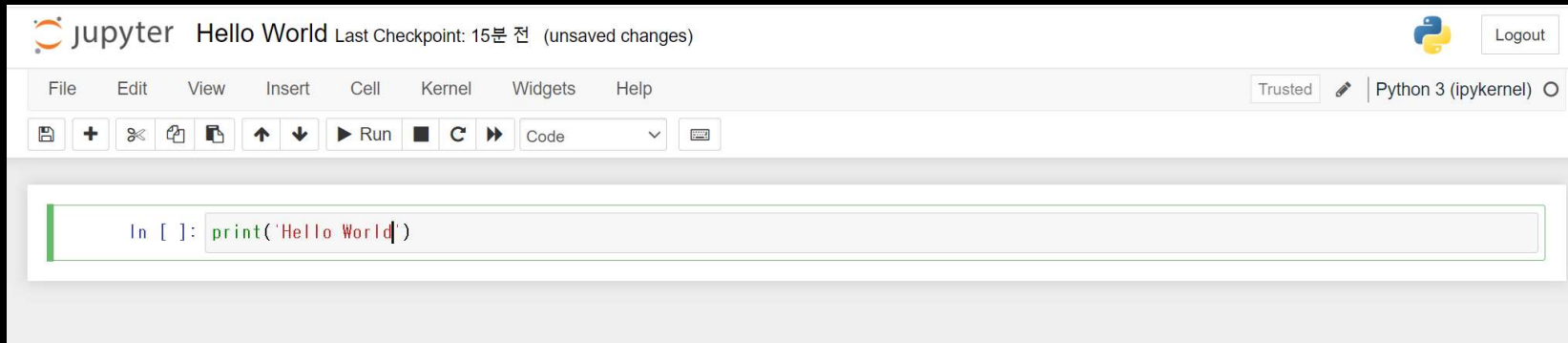


- 상단에 Untitled라는 글자를 클릭하면 문서의 제목을 변경할 수 있는 창이 뜬다. (Hello World라고 입력)

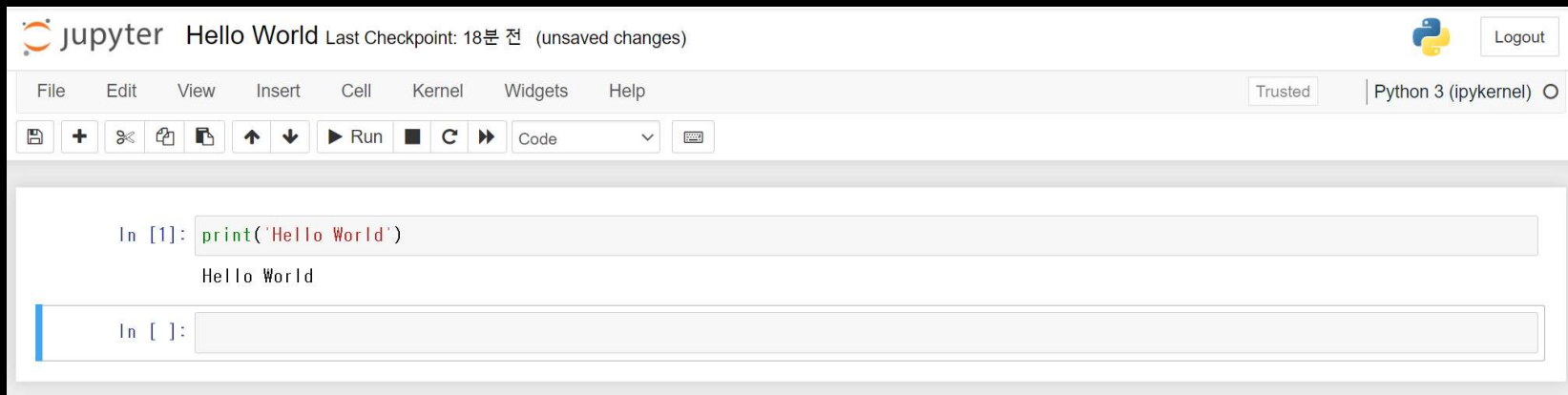


Jupyter의 기본 사용

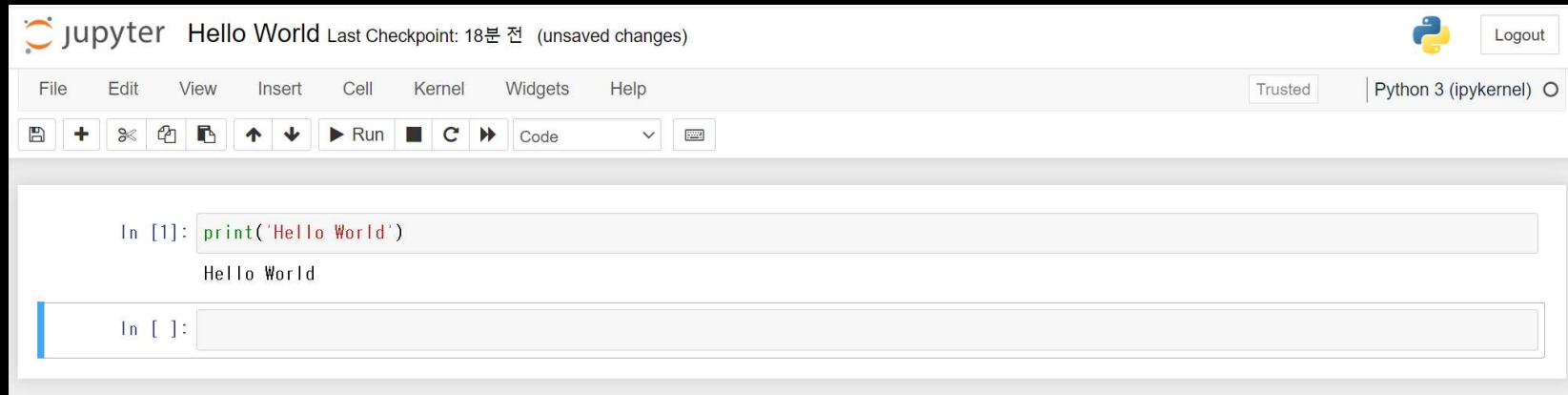
- 빈칸을 마우스로 클릭하고, 아래에 나타나 있는 print문을 입력



- 구문을 다 입력한 후 SHIFT + Enter 키를 누르면 Hello World가 코드 아래에 출력(print)

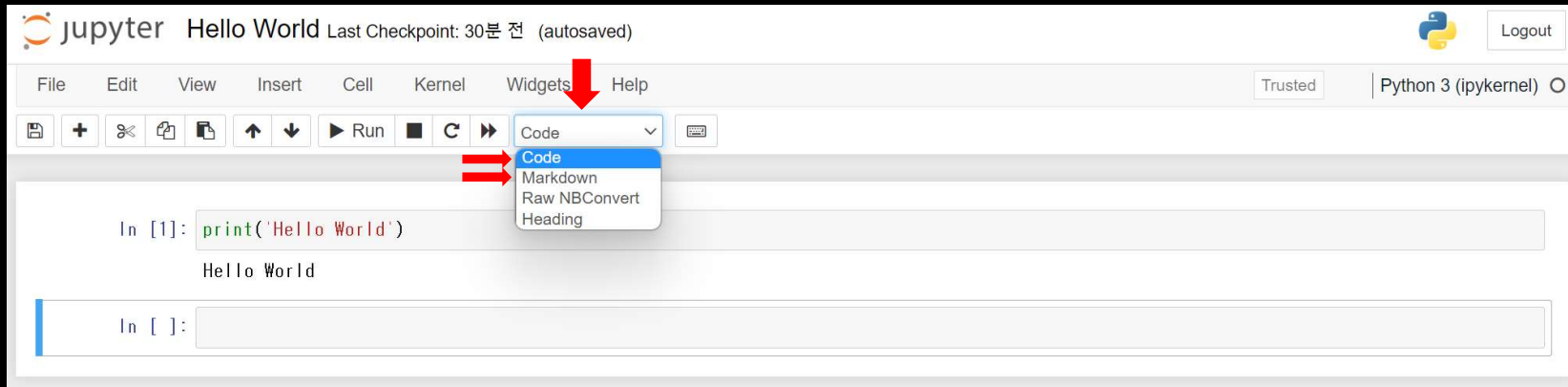


Jupyter의 기본 사용



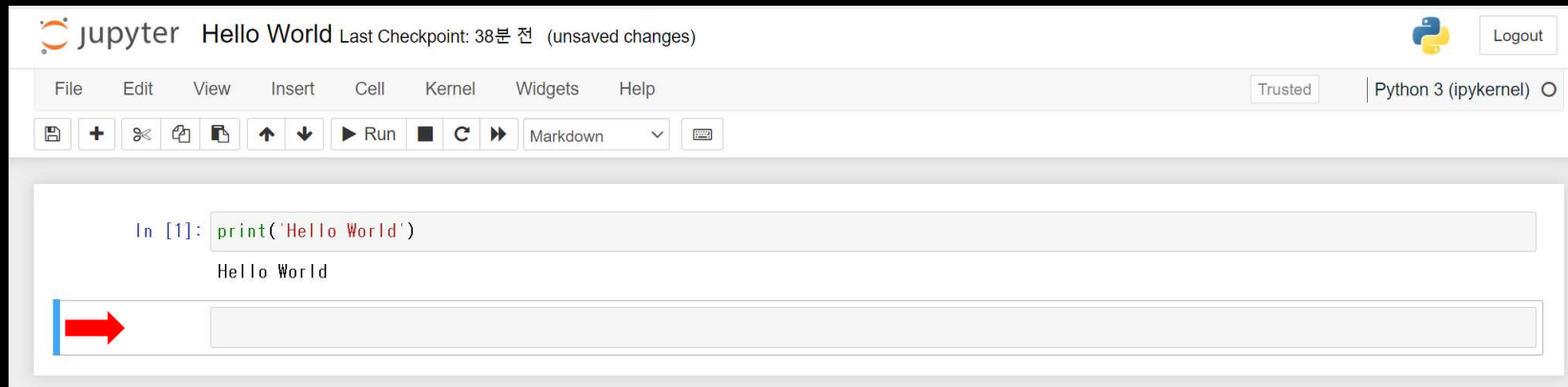
- 코드를 입력하는 영역 바로 아래에서 코드의 출력이 나타나는 것이 Jupyter의 환경
- 좌측 In [1]이라는 번호가 실행한 순서대로 나타남
- Jupyter는 코드가 적힌 순서대로 실행되는 것이 아니라 각 코드 블록의 실행 순서를 마음대로 정할 수 있기 때문에 실행 순서 번호를 주의해서 보아야함
- 코드가 실행된 후 PC를 끄고 다시 실행해서 ipynb 파일을 읽어보면 위의 그림과 같은 모습을 유지함

Markdown 문서



- 위의 그림에서 표시된 부분을 클릭하면 코드를 입력하는 Code와 그 아래에 Markdown이 보임
- Markdown은 Jupyter 문서를 단순 코드가 아니라 문서로서의 가치를 가지게 만듦
- Jupyter의 마크다운에서는 LaTeX 문법을 이용해서 수식 입력이 가능하고 나중에 간단한 설정으로 PDF로 출력해보면 문서로서도 그럴듯하게 만들어짐
- 연습하는 코드를 정리할 때 Markdown을 한 번 활용해 보시기를 권장

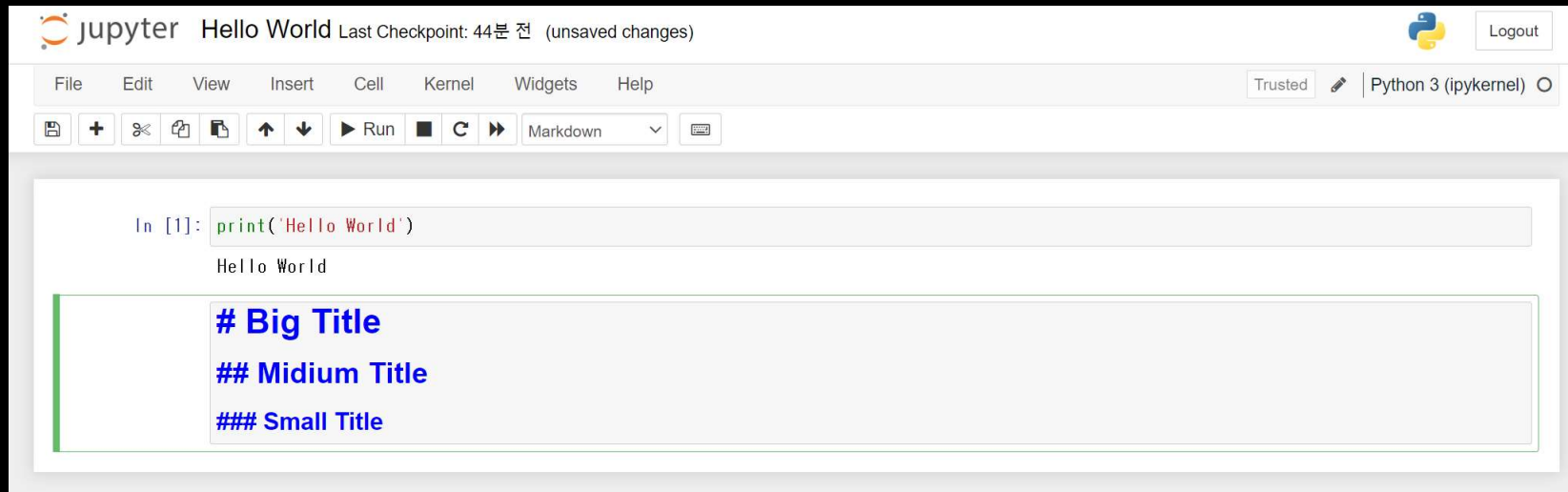
Markdown 문서



- Markdown을 선택하면 코드를 입력할 때 나타나는 In []이라는 글자가 사라짐

Markdown 문서

<제목 달기>



- # 한 개, 두 개, 세 개는 각각 큰 제목, 중간 제목, 작은 제목을 의미

Markdown 문서

<제목 달기>

```
In [1]: print('Hello World')
Hello World

Big Title

Midium Title

Small Title

In [ ]:
```

- Shift + Enter를 입력해서 실행
- 위 그림처럼 제목이 단계별로 나뉘서 나타남
- 나중에 별도의 위젯을 설치하면 PDF로 변환되거나 HTML로 변환할 수 있는데 이때 제목 설정은 자동으로 목차를 생성할 때 유리

Markdown 문서

<목록 만들기>

- 새로 만들어진 코드입력 블록도 Markdown으로 설정

```
Big Title

Midium Title

Small Title

### List without number
* Item
* Item

### List with number
1. Item
2. Item
```

- 별표(*)를 입력한 뒤 한 칸 띄워서 글을 작성하면 번호 없는 목록
- 숫자로 시작하면 번호 목록

```
Big Title

Midium Title

Small Title

List without number
• Item
• Item

List with number
1. Item
2. Item

In [ ]:
```

<실행 결과>

Markdown 문서

<강조와 기울임>

List with number

1. Item
2. Item

Italic : **italic**

Bold : ****Bold****

both : ******both******

- 별표 한 개, 두 개, 세 개로 각각 둘러싸면 기울임(italic), 굵은 글씨, 둘 다 적용하는 것 가능

List with number

1. Item
2. Item

Italic : *Italic*

Bold : **Bold**

both : ***both***

<실행 결과>

Markdown 문서

<인용문>

Italic : *Italic*

Bold : **Bold**

both : ***both***

Normal Text

> quote

Normal Text

- 문장의 맨 앞부분에 닫히는 꺾쇠괄호(>)를 사용하면 인용문 실행

Italic : *Italic*

Bold : **Bold**

both : ***both***

Normal Text

quote

Normal Text

<실행 결과>

In []:

Markdown 문서

<코드블록>

- 문서로서(실행할 목적이 아닌) 코드를 문법적 강조까지 하면서 기록하는 것을 코드블록이라고 함

```
Normal Text

```python
def hello_world():
 print('Hello World')
```
```

- 키보드 자판 1 옆의 키(아포스트로피)로 작성 ****작은따옴표가 아닙니다.****
- 아포스트로피 기호를 연달아 세 번 사용하고 적용하고 싶은 언어를 명시
- 코드 작성이 끝난 후 다시 아포스트로피 세 개를 연달아 써서 마무리

```
Normal Text

def hello_world():
    print('Hello World')
```

In []:

<실행 결과>

Jupyter로 토픽 구독

< 준비 >

- 불필요한 혼선을 막기 위해 이전에 실행해 둔 터미널은 모두 끄고 다시 실행
- Home 경로의 python 폴더로 이동한 후 galactic을 실행하고 jupyter notebook을 실행

```
drcl-rss@drclrss-MS-7B23: ~/python
drcl-rss@drclrss-MS-7B23:~$ cd python/
drcl-rss@drclrss-MS-7B23:~/python$ jupyter notebook
[I 2023-07-21 13:22:10.264 ServerApp] Package notebook took 0.0000s to import
[I 2023-07-21 13:22:10.270 ServerApp] Package jupyter_lsp took 0.0062s to import
```

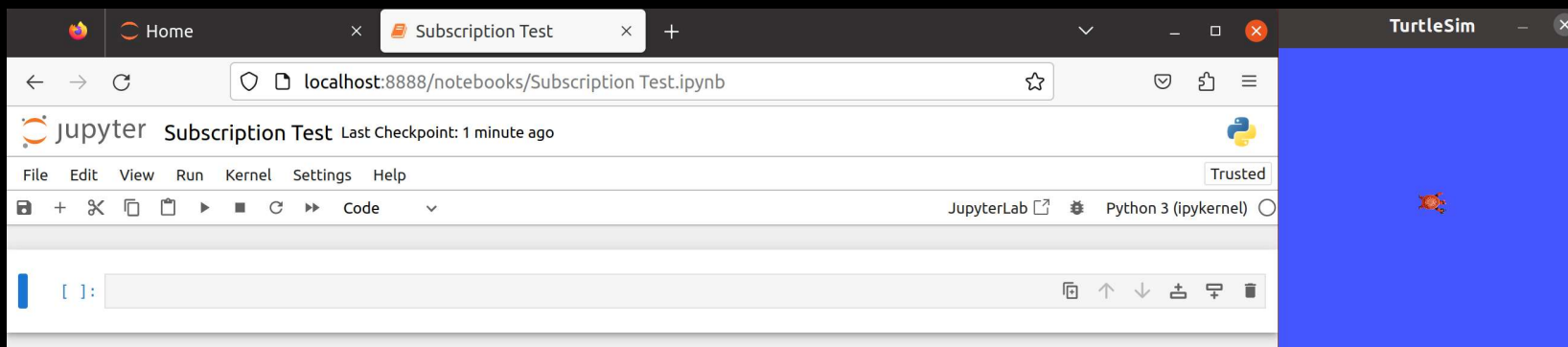
- ros2 run 명령으로 turtlesim 패키지의 turtlesim_node를 실행

```
drcl-rss@drclrss-MS-7B23: ~/python
drcl-rss@drclrss-MS-7B23:~/python$ ros2 run turtlesim turtlesim_node
[INFO] [1689913436.294046029] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1689913436.296178665] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
```

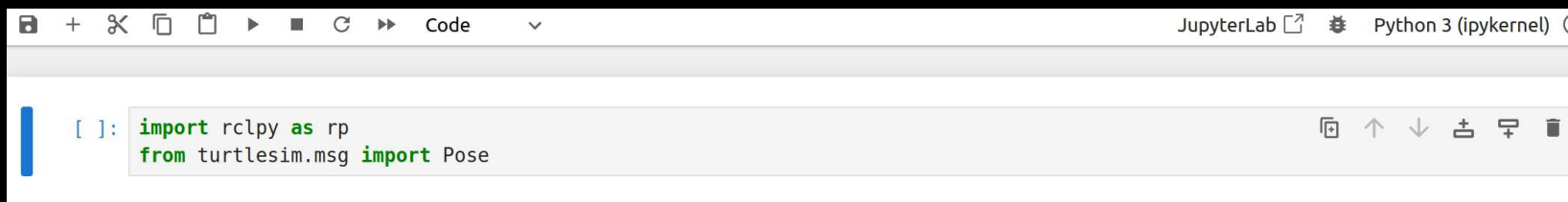
Jupyter로 토픽 구독

< 준비 >

- Jupyter에서 Subscription Test라는 이름의 ipynb 파일 생성



< import >



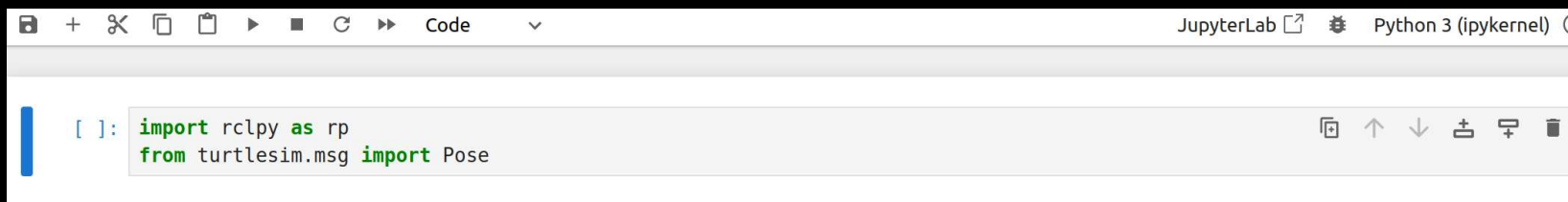
- 첫 줄은 ROS2를 python에서 사용할 수 있게 해주는 모듈을 import 하는 것
- 그 뒤에 as를 사용해서 rp를 한 것은 rclpy 모듈을 불러와서(import) 앞으로는 rp라고 부르겠다는 의미
- as는 alias를 뜻함

Jupyter로 토픽 구독

< import >

```
drcl-rss@drclrss-MS-7B23:~/python$ ros2 topic list -t
/parameter_events [rcl_interfaces/msg/ParameterEvent]
/rosout [rcl_interfaces/msg/Log]
/turtle1/cmd_vel [geometry_msgs/msg/Twist]
/turtle1/color_sensor [turtlesim/msg/Color]
/turtle1/pose [turtlesim/msg/Pose]
```

- turtlesim_node의 토픽 중에서 /turtle1/pose라는 토픽을 구독(subscription)할 예정
- 원하는 토픽을 다루기 위해서는 토픽의 데이터 타입을 topic list -t 또는 topic info 명령으로 확인해야 함
- /turtle1/pose라는 토픽은 turtlesim/msg/Pose라는 데이터 타입을 사용
- 앞 시간에 확인한 turtlesim/msg/Pose는 x, y, theta와 linear, angular, velocity로 구성



The screenshot shows a JupyterLab window with a code editor. The code in the cell is:

```
[ ]: import rclpy as rp
      from turtlesim.msg import Pose
```

The window title bar indicates 'JupyterLab' and 'Python 3 (ipykernel)'.

- 두 번째 줄은 python에서 turtlesim/msg/Pose 데이터 타입을 사용하기 위해 어떻게 import하는지 보여줌
- 위 처럼 import하면 Pose라는 클래스 사용가능

Jupyter로 토픽 구독

< rclpy 초기화 및 노드 생성 >

`rp.init()`

- Python에서 rclpy를 초기화하기 위한 코드

`test_node = rp.create_node('sub_test')`

- sub_test라는 이름의 노드를 만들기 위한 코드

```
[1]: import rclpy as rp
    from turtlesim.msg import Pose

[2]: rp.init()
    test_node = rp.create_node('sub_test')
```

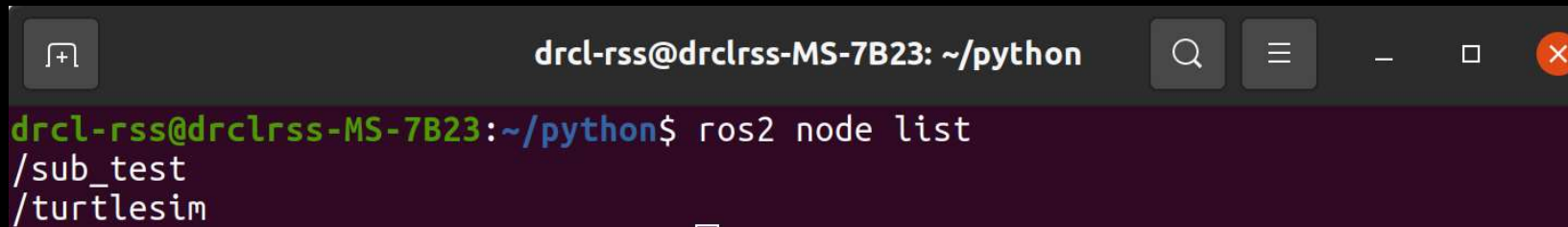
[]: |



- 여기까지 실행했다면, /turtlesim과 /sub_test라는 노드가 실행된 것

Jupyter로 토픽 구독

< rclpy 초기화 및 노드 생성 >



```
drcl-rss@drclrss-MS-7B23: ~/python
drcl-rss@drclrss-MS-7B23:~/python$ ros2 node list
/sub_test
/turtlesim
```

- 노드가 실행되었는지 확인하는 방법은 `ros2 node list` 명령으로 확인 가능

Jupyter로 토픽 구독

< Subscription에서 실행할 callback 함수 작성 >

- 즉, 토픽을 받을 때마다 어떤 일을 수행하게 하는 함수를 callback 함수라고 하며 미리 작성해 두어야함

```
[2]: rp.init()
test_node = rp.create_node('sub_test')

[3]: def callback(data):
      print("--->")
      print("/turtle1/pose : ", data)
      print("X: ", data.x)
      print("Y: ", data.y)
      print("Theta : ", data.theta)
```

- 함수란 언어에서 재사용이 가능하고 호출할 수 있는 것을 의미

def callback(data):

- def라는 키워드를 적고 그 뒤에 원하는 함수 이름을 지정
- 해당 함수가 어떤 입력을 받아야 한다면 괄호 안에 입력 받을 데이터명 기입
- 콜론(:) 기호는 함수 구문이 시작된다는 것을 의미

Jupyter로 토픽 구독

< Subscription에서 실행할 callback 함수 작성 >

```
[3]: def callback(data):  
    print("--->")  
    print("/turtle1/pose : ", data)  
    print("X: ", data.x)  
    print("Y: ", data.y)  
    print("Theta : ", data.theta)
```



- 콜론(:) 기호 입력 후 엔터키를 입력하면 들여쓰기로 몇 칸 정도 앞에서 시작
- Python은 들여쓰기를 사용, 함수(def), 반복문(for), 조건문(if) 등 모두 첫 줄의 마지막은 콜론(:) 기호로 끝나고 그 다음 줄부터 들여쓰기가 된 줄들은 구문(def, for, if 등)에 포함
- callback이 받는 데이터(data)는 turtlesim/msg/Pose로 x, y, theta와 linear, angular velocity로 구성
- x, y, theta를 확인하고 싶을 땐 data.x, data.y, data.theta 이렇게 조회 가능
- 따라서 print 문으로 data.x, data.y, data.theta를 출력해서 확인하기 위해 callback 함수를 구성

Jupyter로 토픽 구독

<토픽 subscriber 만들기>

```
[3]: def callback(data):  
      print("--->")  
      print("/turtle1/pose : ", data)  
      print("X: ", data.x)  
      print("Y: ", data.y)  
      print("Theta : ", data.theta)  
  
[4]: test_node.create_subscription(Pose, '/turtle1/pose', callback, 10)  
  
[4]: <rclpy.subscription.Subscription at 0x7f3fc4515ca0>
```

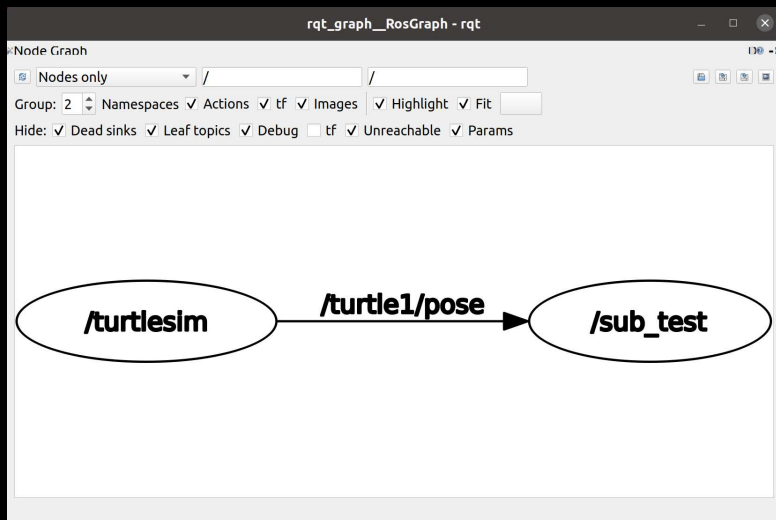
test_node.create_subscription(<data_type>, '<topic_name>', <callback>, <Qos History>)

- data_type: import한 turtlesim의 Pose
- topic_name: /turtle1/pose
- callback: 토픽이 들어오면 실행할 함수 위 사진과 같이 지정
- Qos History: 혼동될 수 있으니 나중에 다시 다룸, 지금은 저장할 샘플의 수라고 언급하도록 하겠음

Jupyter로 토픽 구독

<토픽 subscriber 만들기>

- 진행이 다 되었다면 rqt_graph를 실행



- create_subscription 명령에 의해 /turtlesim 노드에서 /turtle1/pose라는 토픽이, /sub_test라는 노드로 지나가고 있음을 확인할 수 있음

Jupyter로 토픽 구독

<토픽 subscriber 만들기>

rp.spin_once(test_node)

- rclpy(rp)의 spin_once라는 명령에서 선언한 test_node를 위와 같이 지정하면 해당 노드를 구독할 수 있음
- spin_once는 토픽을 한 번만 받아들이는 명령, spin은 토픽을 계속 무한히 구독하게 하는 명령

```
[4]: test_node.create_subscription(Pose, '/turtle1/pose', callback, 10)
[4]: <rclpy.subscription.Subscription at 0x7f3fc4515ca0>
[5]: rp.spin_once(test_node)
--->
/turtle1/pose : turtlesim.msg.Pose(x=5.544444561004639, y=5.544444561004639, theta=0.0, linear_velocity=0.0, angular_velocity=0.0)
X: 5.544444561004639
Y: 5.544444561004639
Theta : 0.0
```

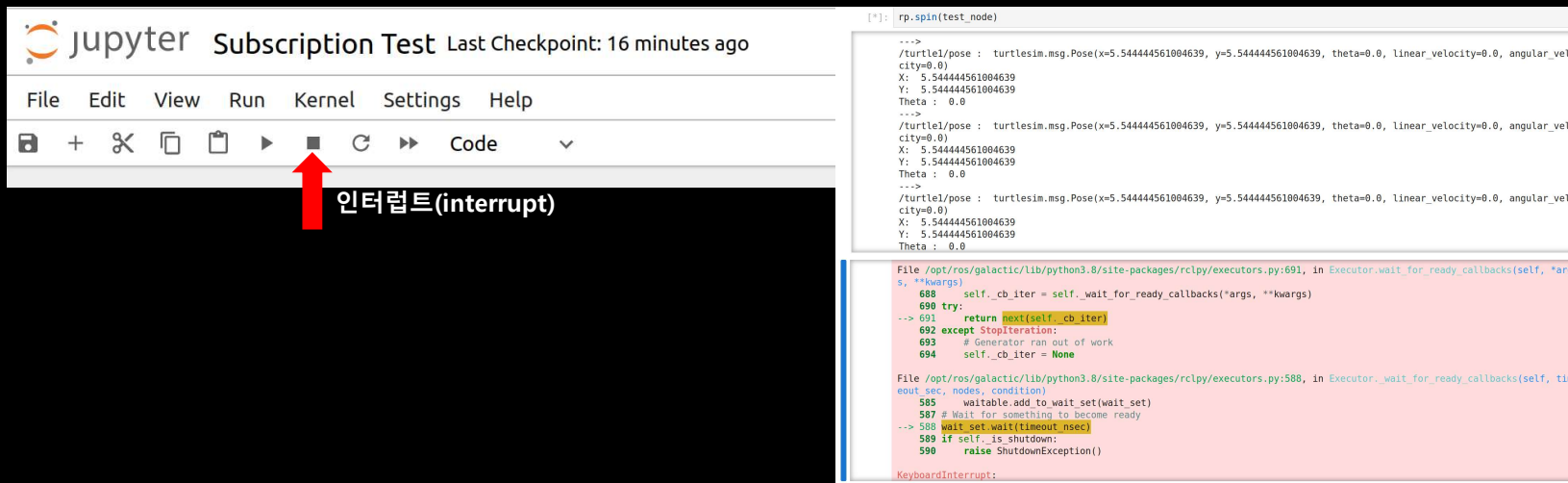
- spin_once가 한 번 실행되면 토픽이 들어올 때까지 기다렸다가, 토픽이 들어오면 지정된 callback 함수가 실행되는 것을 확인

Jupyter로 토픽 구독

<Jupyter 사용에서 유의할 점>

`rp.spin(test_node)`

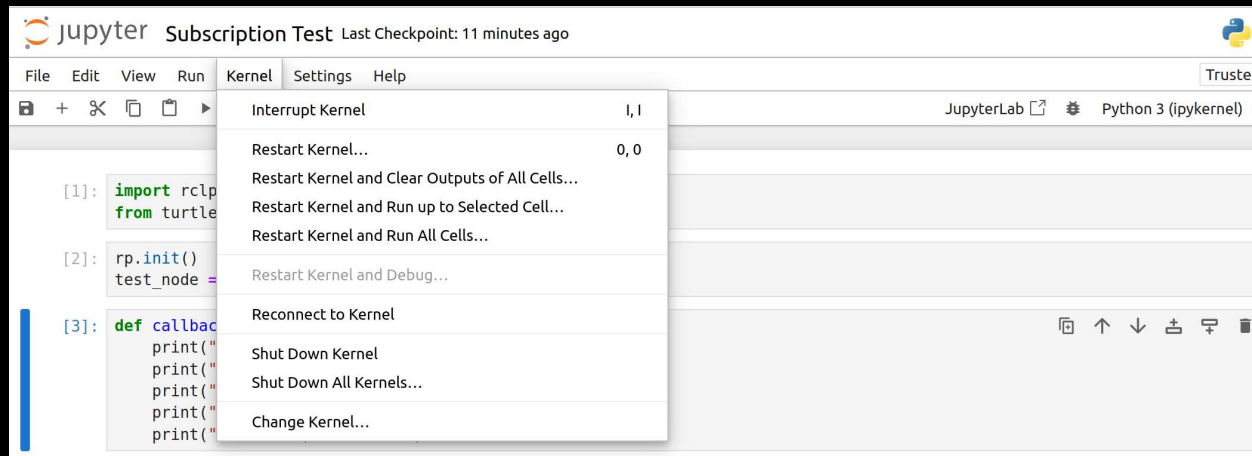
- `spin_once` 대신 `spin`을 사용한 경우 `In[*]`으로 계속 실행되고 있음을 알리는 별표가 나타나고 셀이 멈추지 않음
- 멈추는 방법은 인터럽트(interrupt) 버튼을 누르는 것



- 오른쪽 사진처럼 외부입력으로 멈췄다는 에러가 나오며 정지
- 구독 관련 설정이 지워지는 것은 아니고 그저 `spin` 함수의 동작만 멈춘 상태

Jupyter로 토픽 구독

<Jupyter 사용에서 유의할 점>

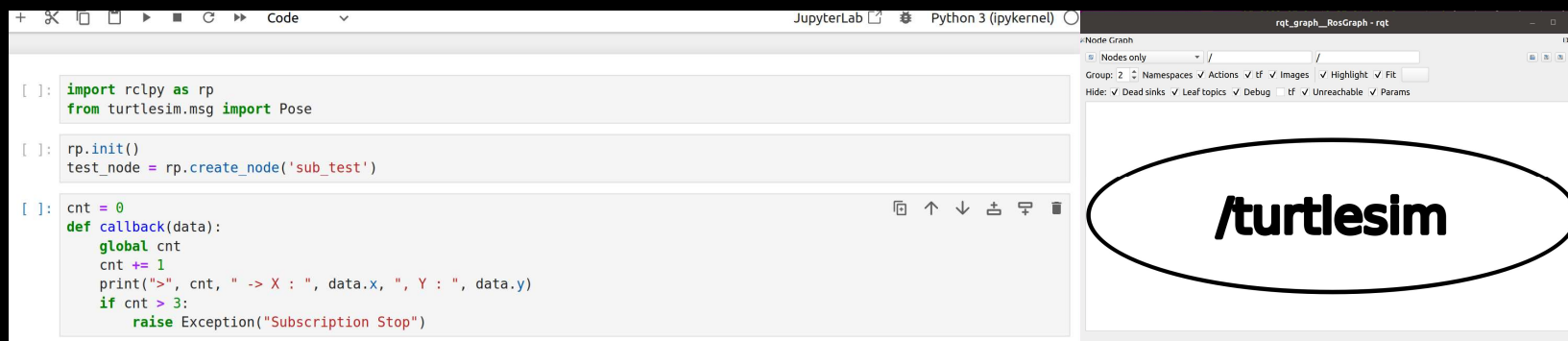


- callback 함수의 내용을 바꾸거나 생성된 노드를 중단하고 싶다면 Python으로 destroy_node 함수를 사용가능
- 하지만 Jupyter 환경에서는 상단 메뉴의 Kernel에서 restart 관련 명령을 선택
- Restart Kernel: 커널 재시작
- Restart Kernel and Clear Output: 실행 결과들을 지우는 것 의미
- Restart Kernel and Run All은 커널과 코드를 다 다시 시작하라는 의미

Jupyter로 토픽 구독

< Jupyter 사용에서 유의할 점 >

- Restart Kernel and Clear Output 선택 시 모든 실행 결과가 지워져 코드만 남고 실행 순서를 의미하는 번호도 사라짐



- rqt_graph를 실행하면 /sub_test라는 실행했던 노드가 사라진 것을 확인할 수 있음
- 한 번 create_node 명령 후에 create_subscription을 하면 callback 함수를 변경하고 다시 실행해도 반영되지 않을 때가 있음 이때 역시 커널을 restart해야함

Jupyter로 토픽 구독

<토픽을 받는 횟수 제한해보기>

```
[ ]: rp.init()
test_node = rp.create_node('sub_test')

[ ]: cnt = 0
def callback(data):
    global cnt
    cnt += 1
    print(">", cnt, " -> X : ", data.x, ", Y : ", data.y)
    if cnt > 3:
        raise Exception("Subscription Stop")
```

- cnt: 구독한 메시지의 구독 횟수를 계산하기 위해 함수(def) 밖에 위치
- 함수 외부의 변수를 다루기 위해 cnt를 global로 선언

global cnt

- callback 함수가 한 번 실행될 때마다 cnt를 1씩 증가시킴

cnt += 1

- cnt 변수의 값과 토픽의 x, y 값만 출력하기 위한 print 작성

print(">", cnt, " -> X : ", data.x, ", Y : ", data.y)

Jupyter로 토픽 구독

<토픽을 받는 횟수 제한해보기>

```
[ ]: rp.init()
test_node = rp.create_node('sub_test')

[ ]: cnt = 0
def callback(data):
    global cnt
    cnt += 1
    print(">", cnt, " -> X : ", data.x, ", Y : ", data.y)
    if cnt > 3:
        raise Exception("Subscription Stop")
```

- cnt가 1부터 시작하여 print 문을 찍고 조건문(if)에서 cnt가 3보다 큰지 확인

if cnt > 3:

- cnt가 1, 2, 3, 4가 될 때에 맞춰 총 네 번의 callback 함수가 실행
- if 문 안의 조건이 참(True)가 되면 들여쓰기가 적용된 if 구문을 실행 즉 cnt가 4가 되면 코드 실행

raise Exception("Subscription Stop")

- Python에서 raise는 예외를 발생시키는 구문, 특정 메시지를 출력하면서 에러를 발생시키는 것이라고 이해

Jupyter로 토픽 구독

<토픽을 받는 횟수 제한해보기>

- 마지막 블록에서 토픽을 구독하는 개수를 제한하고 멈추는 방법은 subscription의 destroy()함수를 불러오거나, node의 destroy_subscription() 함수를 이용해도 가능
- 하지만 두 방법 모두 자연스럽게 부드럽게 토픽 구독을 마치는 것이 아닌 에러를 발생시키므로 토픽을 구독하거나 발행하는 중에 종료하는 행동은 모두 본래 ROS의 의도가 아닌 것 같음
- Jupyter 환경이 아닌 경우 쓸 일 없음
- 일반적으로 토픽은 계속 발행되거나 구독하게 두고, 필요 없으면 ctrl+c로 터미널을 종료하기 때문
- 따라서 Python의 raise 구문으로 그냥 처리

Jupyter로 토픽 구독

<토픽을 받는 횟수 제한해보기>

```
[5]: rp.spin(test_node)

> 1 -> X : 5.544444561004639 , Y : 5.544444561004639
> 2 -> X : 5.544444561004639 , Y : 5.544444561004639
> 3 -> X : 5.544444561004639 , Y : 5.544444561004639
> 4 -> X : 5.544444561004639 , Y : 5.544444561004639

-----
Exception                                Traceback (most recent call last)
Cell In[5], line 1
----> 1 rp.spin(test_node)

File /opt/ros/galactic/lib/python3.8/site-packages/rclpy/__init__.py:196, in spin(node, executor)
    194     executor.add_node(node)
    195     while executor.context.ok():
--> 196         executor.spin_once()
    197 finally:
    198     executor.remove_node(node)

File /opt/ros/galactic/lib/python3.8/site-packages/rclpy/executors.py:713, in SingleThreadedExecutor.spin_once(self, timeout_sec)
```

- cnt가 4까지 실행되고 코드는 멈춤
- 이때 마지막까지 스크롤을 해보면, raise 구문에서 작성한 메시지가 나타나는 것을 알 수 있음

Jupyter로 토픽 구독

<토픽 발행을 위한 rclpy와 메시지 초기화>

- 새 문서를 Python3로 시작하여 Publisher Test라고 이름을 정함



- 토픽을 구독하기 위한 준비를 했던 초반 상황처럼 turtlesim을 실행시킴
- 간단히 필요한 모듈을 import 하고 노드를 초기화하는 코드를 작성하고 실행

```
[1]: import rclpy as rp
      from geometry_msgs.msg import Twist

      rp.init()
      test_node = rp.create_node('pub_test')
```

Jupyter로 토픽 발행

<토픽 발행을 위한 rclpy와 메시지 초기화>

```
[1]: import rclpy as rp
    from geometry_msgs.msg import Twist

    rp.init()
    test_node = rp.create_node('pub_test')
```

from geometry_msgs.msg import Twist

- 다루려고 하는 토픽의 데이터 타입에 따라 import
- cmd_vel이라는 토픽을 발행하려고 함
- cmd_vel은 geometry_msgs의 Twist 데이터 타입 따라서 두 번째 줄처럼 import

rp.init()

- init()으로 초기화

test_node=rp.create_node('pub_test')

- create_node 명령으로 pub_test라는 이름의 노드 생성
- pub_test라는 이름의 노드를 여러 속성을 수정할 수 있도록 test_node라는 이름으로 지정

Jupyter로 토픽 발행

<cmd_vel 토픽의 데이터 타입인 Twist 선언>

- 토픽을 발행하는 것은 메시지의 타입을 알고 그 타입에 맞도록 발행할 내용을 저장하는 것이 우선
- Twist 데이터 타입을 import 했으니 해당 메시지를 msg라는 변수에 지정

```
msg = Twist()
```

```
[1]: import rclpy as rp
    from geometry_msgs.msg import Twist

    rp.init()
    test_node = rp.create_node('pub_test')

[2]: msg = Twist()
    print(msg)

geometry_msgs.msg Twist(linear=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
```

- print 코드를 실행하여 msg라는 변수에 어떤 내용이 있는지 출력
- Twist 데이터형은 3차원 벡터 linear와 angular 두 개로 되어 있고 두 벡터 모두 x, y, z값을 가짐
- 코드는 속도 명령
- msg라는 변수를 만들 때 내부 값들은 모두 0으로 초기화

Jupyter로 토픽 발행

<Python으로 cmd_vel 토픽 간단히 발행해보기>

- linear의 x값을 수정하는 코드를 작성
- 다른 성분들이 0이라는 것을 확인했으니 linear.x만 바꿔주기
- 결과는 print문으로 확인 가능

```
msg.linear.x = 2.0
```

```
[3]: msg.linear.x = 2.0
     print(msg)

geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
```

- test_node라는 변수에서 지정한 pub_test라는 노드가 create_publisher로 토픽을 발행할 수 있음

```
pub = test_node.create_publisher(Twist, '/turtle1/cmd_vel', 10)
```

- 위처럼 데이터 타입(Twist)과 토픽 이름(/turtle1/cmd_vel)을 지정하고 Qos 설정을 해주면 됨

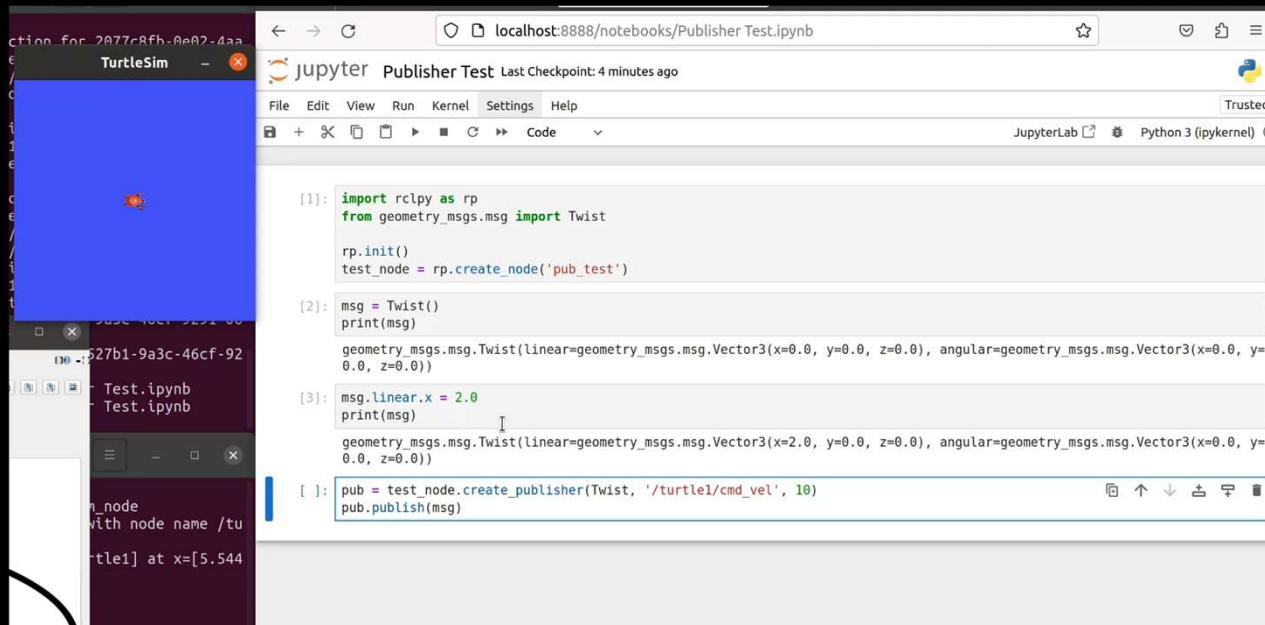
Jupyter로 토픽 발행

<Python으로 cmd_vel 토픽 간단히 발행해보기>

```
[4]: pub = test_node.create_publisher(Twist, '/turtle1/cmd_vel', 10)
      pub.publish(msg)
```

- pub이라는 변수는 토픽을 발행하는 것과 관련된 기능, 설정들을 지정할 수 있음
- 저장해둔 Twist 데이터 타입의 msg를 발행하기 위해 두 번째 줄과 같은 코드 작성

pub.publish(msg)

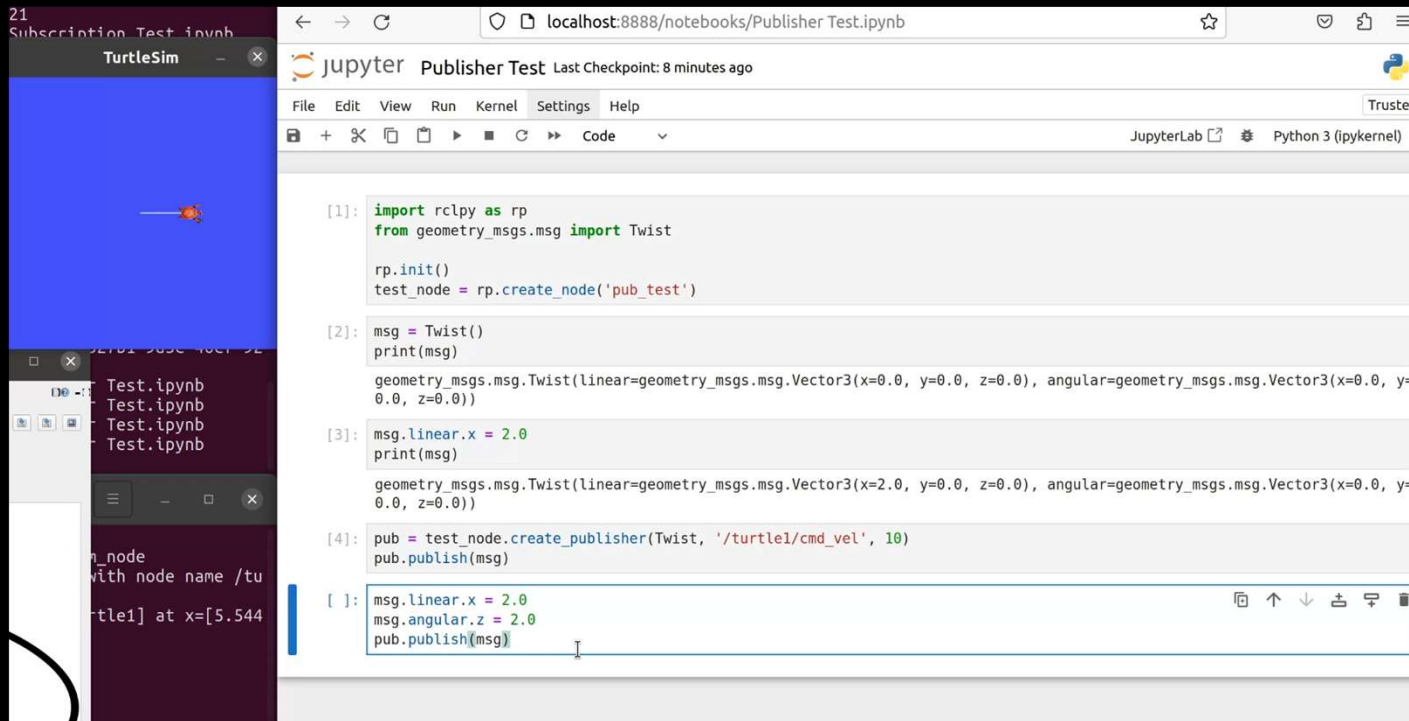


Jupyter로 토픽 발행

<Python으로 cmd_vel 토픽 간단히 발행해보기>

- 추가로 또 움직이게 하고 싶다면 msg를 변경
- 이미 create_publisher 명령으로 publisher를 생성했으므로 publish 명령만 인가

pub.publish(msg)



```
[1]: import rclpy as rp
    from geometry_msgs.msg import Twist

    rp.init()
    test_node = rp.create_node('pub_test')

[2]: msg = Twist()
    print(msg)

    geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))

[3]: msg.linear.x = 2.0
    print(msg)

    geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))

[4]: pub = test_node.create_publisher(Twist, '/turtle1/cmd_vel', 10)
    pub.publish(msg)

[ ]: msg.linear.x = 2.0
    msg.angular.z = 2.0
    pub.publish(msg)
```

- msg의 angular의 z 성분도 변경

Jupyter로 토픽 발행

<ROS에서 timer를 이용해서 토픽 발행하기>

- 일정 주기로 토픽을 발행하기 위해서는 일정 시간을 유지할 필요가 있음
- 이러한 기능은 create_timer를 통해 지정

```
test_node.create_timer(timer_period, timer_callback)
```

- 타이머의 주기(timer_period)와 해당 시간에 뭘 할 건지 콜백 함수(timer_callback)을 지정

```
[6]: cnt = 0

def timer_callback():
    global cnt

    cnt += 1

    print(cnt)
    pub.publish(msg)

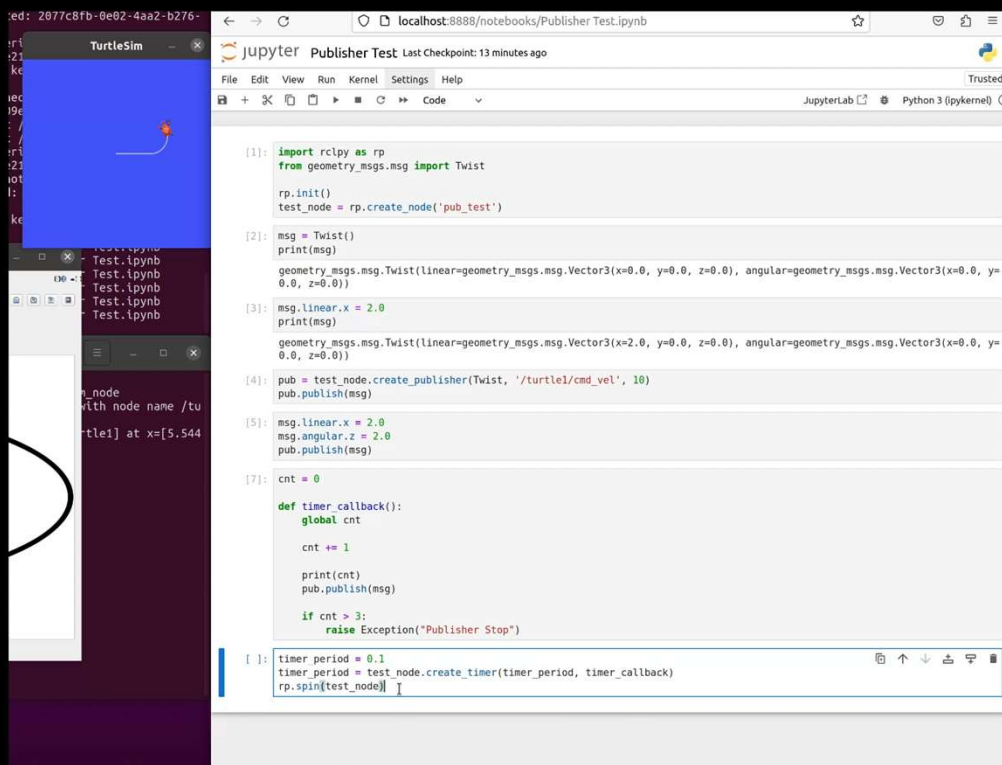
    if cnt > 3:
        raise Exception("Publisher Stop")
```

- 사용한 횟수 제한을 위해 cnt라는 변수에 0을 넣어 두고 timer가 호출될 때마다 cnt를 1씩 증가시키는 코드를 함수(def) 안에 넣음
- timer_callback 함수가 호출될 때마다 msg를 publish 하도록 함
- 조건문(if)는 횟수를 제한하도록 한 것

Jupyter로 토픽 발행

<ROS에서 timer를 이용해서 토픽 발행하기>

- timer_callback과 0.1로 지정한 timer_period를 가지고, test_node가 제공하는 create_timer를 생성
- 0.1초에 한 번씩 timer_callback 함수를 실행, 작성한 대로 cmd_vel 토픽을 0.1초에 한 번씩 발행
- 타이머가 동작할 때까지 기다리도록 spin 명령을 사용



```
[1]: import rclpy as rp
from geometry_msgs.msg import Twist

rp.init()
test_node = rp.create_node('pub_test')

[2]: msg = Twist()
print(msg)
geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))

[3]: msg.linear.x = 2.0
print(msg)
geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=2.0, y=0.0, z=0.0), angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))

[4]: pub = test_node.create_publisher(Twist, '/turtle1/cmd_vel', 10)
pub.publish(msg)

[5]: msg.linear.x = 2.0
msg.angular.z = 2.0
pub.publish(msg)

[7]: cnt = 0

def timer_callback():
    global cnt
    cnt += 1
    print(cnt)
    pub.publish(msg)
    if cnt > 3:
        raise Exception("Publisher Stop")

[ ]: timer_period = 0.1
timer_period = test_node.create_timer(timer_period, timer_callback)
rp.spin(test_node)
```

노드의 종료

- node list로 관찰하다 보면 방금 실행했던 노드를 이제는 사용하지 않는데 node list 명령의 결과에 보이거나 또는 몇 시간 전에 실행했던 노드나 토픽이 관찰되는 것을 볼 수 있음
- 이럴 때는 시간이 지나면 없어지기도 하고, 앞서 언급한 ROS_DOMAIN_ID를 다른 번호로 변경
- 확실한 것은 해당 노드를 종료하는 코드를 실행하는 것

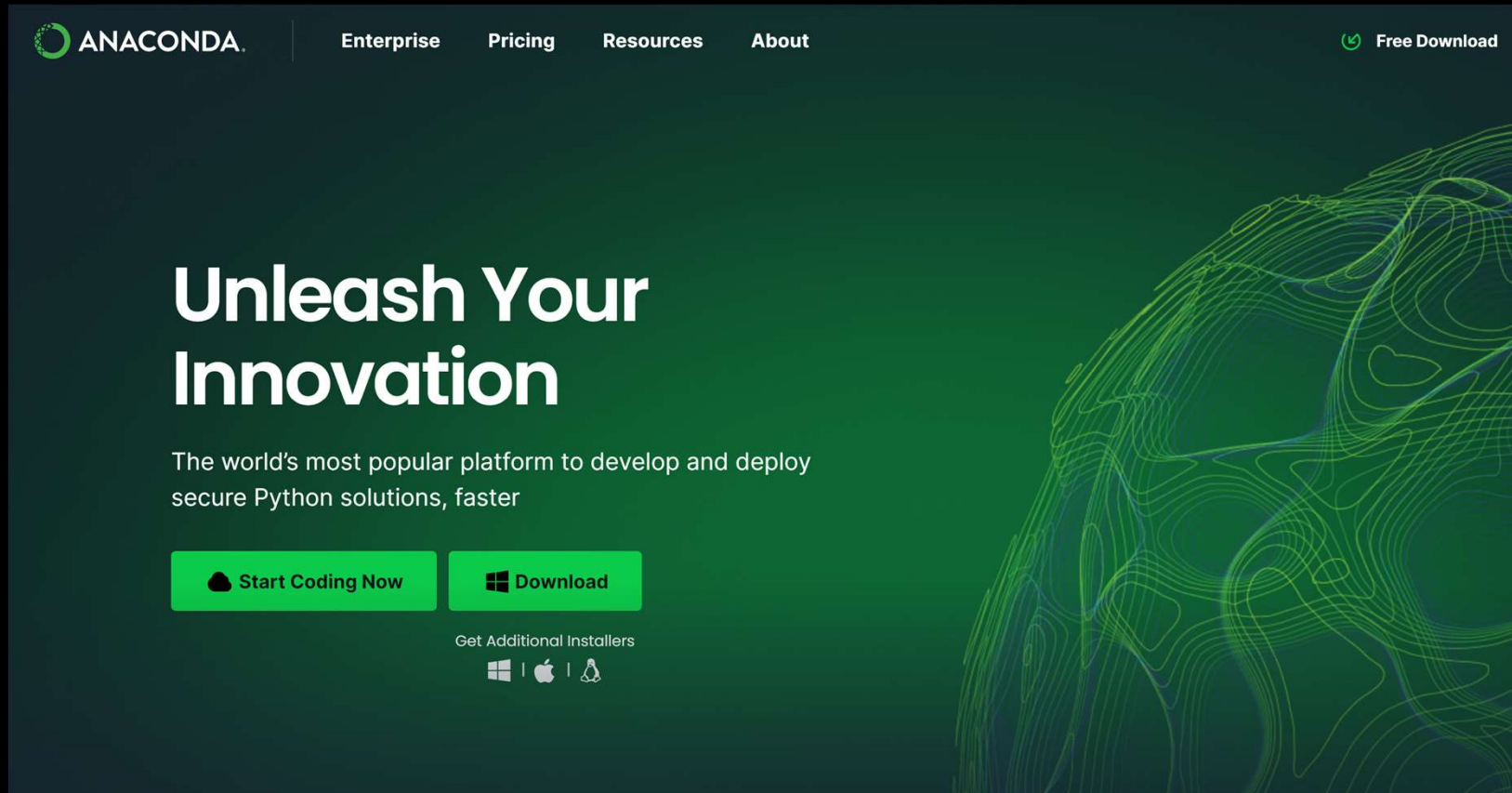
`test_node.destroy_node()`

```
[ ]: test_node.destroy_node()
```



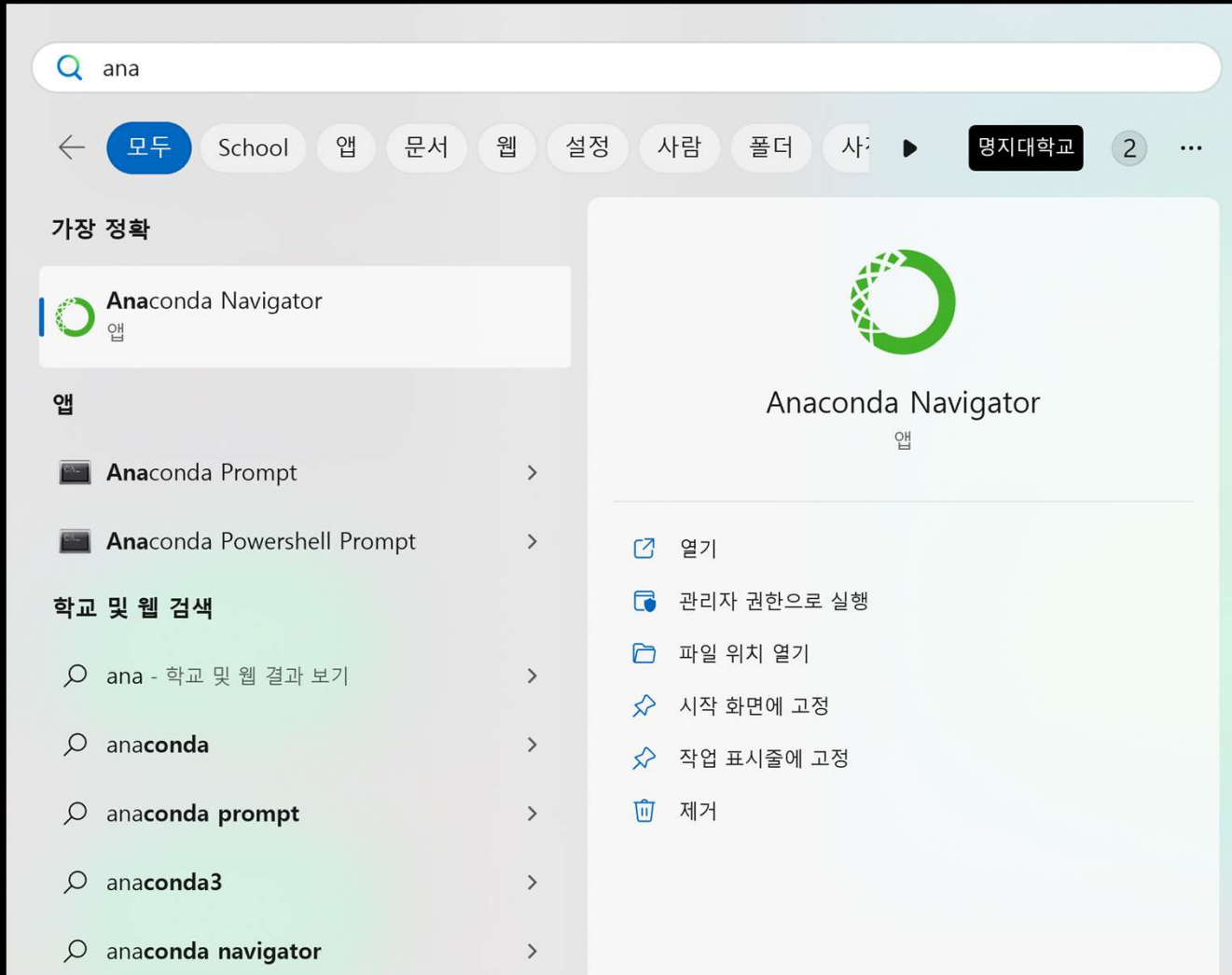
감사합니다.

파이썬 설치 및 준비 (윈도우에서 설치시, 간편한 방법)



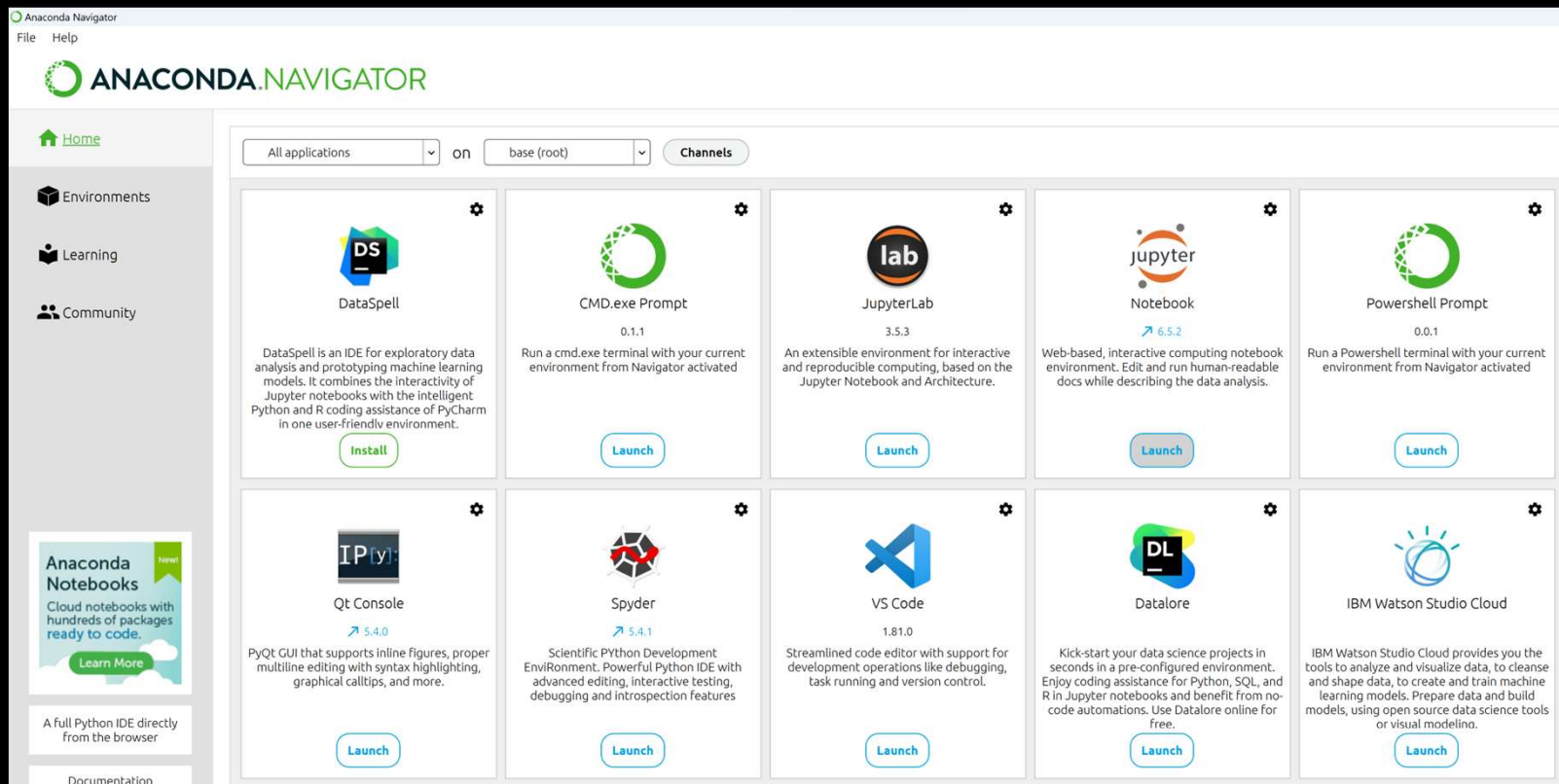
<https://www.anaconda.com/>

파이썬 설치 및 준비 (윈도우에서 설치시, 간편한 방법)



파이썬 설치 및 준비 (Jupyter notebook 실행법 1)

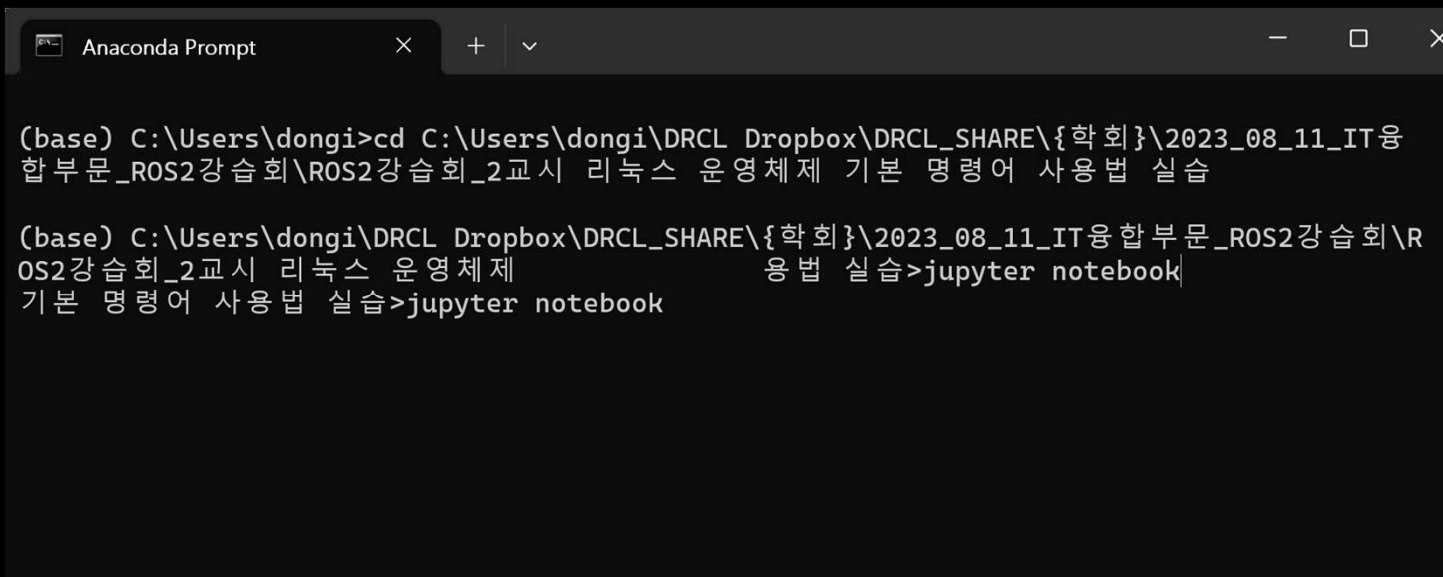
Default 폴더에서 실행하기 : Anaconda Navigator에서 jupyter notebook 실행



파이썬 설치 및 준비 (Jupyter notebook 실행법 2)

특정 폴더에서 실행하기

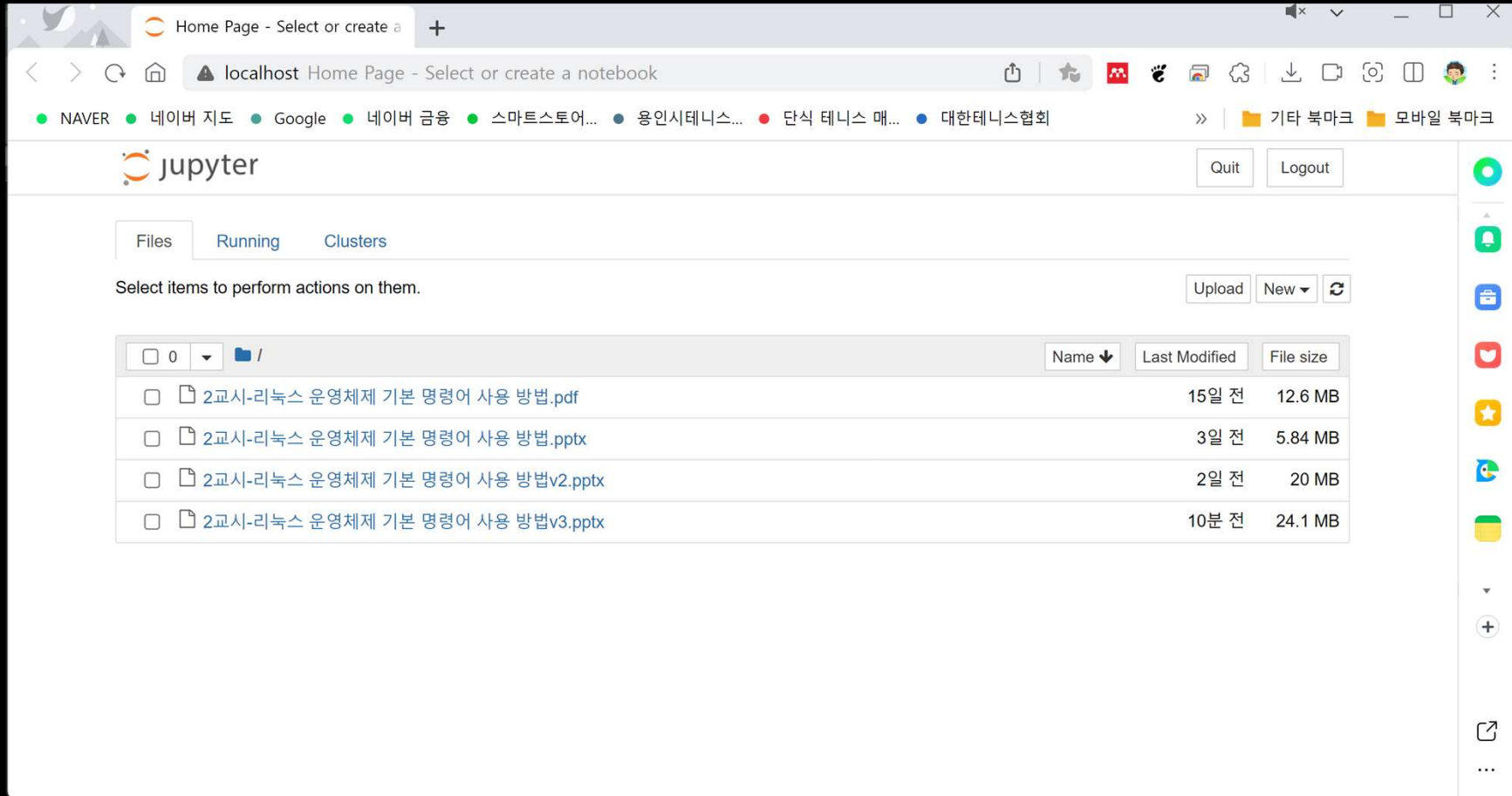
1. Anaconda Prompt 실행
2. 특정 경로로 이동 (cd 명령어 이용)
3. Jupyter notebook 입력



```
Anaconda Prompt
(base) C:\Users\dongi>cd C:\Users\dongi\DRCL Dropbox\DRCL_SHARE\{학회}\2023_08_11_IT융
합부문_ROS2강습회\ROS2강습회_2교시 리눅스 운영체제 기본 명령어 사용법 실습
(base) C:\Users\dongi\DRCL Dropbox\DRCL_SHARE\{학회}\2023_08_11_IT융합부문_ROS2강습회\R
OS2강습회_2교시 리눅스 운영체제
용법 실습>jupyter notebook
기본 명령어 사용법 실습>jupyter notebook
```


파이썬 설치 및 준비 (Jupyter notebook 실행법 2)

특정 폴더에서 실행하기



The screenshot shows the JupyterLab web interface in a browser window. The address bar indicates the URL is localhost. The interface includes a top navigation bar with 'Quit' and 'Logout' buttons. Below this, there are tabs for 'Files', 'Running', and 'Clusters'. The 'Files' tab is active, displaying a file browser. The file browser shows a list of files in the root directory (/). The files are:

| Name | Last Modified | File size |
|----------------------------------|---------------|-----------|
| 2교시-리눅스 운영체제 기본 명령어 사용 방법.pdf | 15일 전 | 12.6 MB |
| 2교시-리눅스 운영체제 기본 명령어 사용 방법.pptx | 3일 전 | 5.84 MB |
| 2교시-리눅스 운영체제 기본 명령어 사용 방법v2.pptx | 2일 전 | 20 MB |
| 2교시-리눅스 운영체제 기본 명령어 사용 방법v3.pptx | 10분 전 | 24.1 MB |