인공지능 개발 환경

Jupyter notebook / TensorFlow
and the more

김 대 환
2022.

Python

- 개발 언어로 파이썬 언어를 많이 사용하는 이유
  - 데이터 과학 분야를 위한 표준 프로그래밍 언어로 발전
  - 범용 프로그래밍 언어의 장점과 매트랩(MATLAB) 같이 특정 분야를 위한 스크립팅 언어의 편리함을 가짐
  - 데이터 적재, 시각화, 통계, 자연어 처리, 이미지 처리 등에 필요한 라이브러리들을 갖춤
  - 그래픽 사용자 인터페이스(GUI)나 웹 서비스 등도 만들 수 있으며, 기존 시스템과 통합이 쉬움

- 필수 라이브러리와 도구
  - 주피터 노트북
  - NumPy
  - SciPy
  - pandas
  - Matplotlib
  - Etc.,

필수 라이브러리와 도구

- **주피터 노트북**
  - 브라우저에서 프로그램 코드를 실행해 주는 대화식 환경
  - 탐색적 데이터 분석에 적합
  - 다양한 프로그래밍 언어를 지원 – 우리는 파이썬만 사용한다
- **Numpy**
  - 파이썬으로 과학 계산에 꼭 필요한 패키지
  - 다차원 배열, 선형 대수, 푸리에 변환 같은 고수준의 수학 함수와 유사 난수 생성기 포함
- **Scipy**
  - 과학 계산용 함수를 모아 놓은 파이썬 패키지
  - 고성능 선형 대수, 함수 최적화, 신호 처리, 특수한 수학 함수와 통계 분포 등을 포함
- **Pandas**
  - 데이터 처리와 분석을 위한 파이썬 라이브러리
  - 데이터 프레임은 엑셀의 스프레드시트와 비슷한 테이블 형태

필수 라이브러리와 도구

- **matplot**
  - 파이썬의 대표적인 과학 계산용 그래프 라이브러리
  - 선 그래프, 히스토그램, 산점도 등을 지원
  - 중요한 통찰을 얻을 수 있도록 데이터와 분석 결과를 다양한 관점에서 시각화

주피터 노트북

- 개요
  - 주피터 노트북이란 오픈소스 (Open source) 기반의 웹 플랫폼으로, 파이썬을 비롯한 다양한 프로그래밍 언어로 코드 작성하고 실행하는 개발 환경
  - 데이터 분석 및 시각화, 모델링, 통계 분석 등, 다양한 분야에서 활용되며 특히 최근 들어 머신 러닝, 딥 러닝에 많이 사용됨
- 특징
  - 데이터를 시각화 (그래프) 하는데 매우 유용
  - 코드, 수식, 시각화 기능들이 포함된 문서 (마크다운)를 생성 및 공유
  - 다양한 프로그래밍 언어를 지원
  - 개발 중간중간 프로그램을 계속해서 실행 하여 확인

주피터 노트북

- **설치 – 리눅스 (Ubuntu 기준)**
  - 설치는 주피터 노트북 서버 설치를 의미
  - 의존성 라이브러리 설치

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install python3-matplotlib -y
$ sudo apt install python3-scipy -y
$ sudo apt install python3-pip
$ sudo apt install -y nodejs npm
$ sudo apt install jupyter-core
```

  - 주피터 노트북 설치

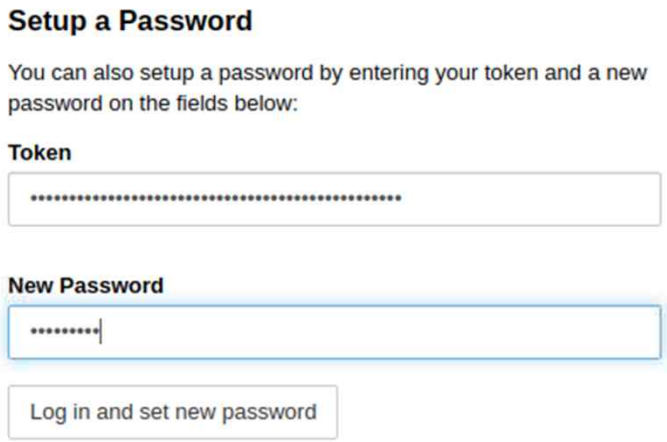
```
$ python3 -m pip install --upgrade pip
$ python3 -m pip install jupyter
$ python3 -m pip install jupyterlab     # 옵션, 주퍼터 노트북의 업그레이드 버전
$ python3 -m pip install ipywidgets     # 옵션, 주피터 노트북 위젯
$ sudo reboot now
```

주피터 노트북

- 설치 – 리눅스 **(Ubuntu 기준)**
  - 주피터 노트북 위젯 활성화

```
$ jupyter nbextension enable --py widgetsnbextension$ python3 –m pip install notebook
```

- 원격 접속 설정
  - 주피터 노트북 접속 암호 및 토큰 생성
    - ✓ 노트북을 처음 실행하면 화면에 토큰이 표시된다. 이 값을 복사한 뒤, 웹 브라우저 로그인 화면에서 입력한다
      - json 파일로 저장되므로 한번만 실행하면 된다



주피터 노트북

- 원격 접속 설정
  - 옵션: 명령어를 이용하여 주피터 노트북 접속 암호 및 토큰 생성
    - ✓ 다음처럼 파이썬 코드를 작성하여 실행
    - ✓ 암호를 입력하고 출력되는 토큰 값을 기억한다

```
from notebook.auth import passwd
passwd()
```

  - "**home/<user name>/.jupyter/jupyter_notebook_config.py**" 파일 생성 및 수정

```
$ jupyter notebook --generate-config
$ vi ~/.jupyter/jupyter_notebook_config.py

c.NotebookApp.password = '복사한 토큰 값 입력'
c.NotebookApp.open_browser = False
c.NotebookApp.notebook_dir = '원하는 작업 디렉터리 경로'
```

주피터 노트북

- 추가 설정
  - **matplotlib**와 **pylab** 라이브러리를 주피터 노트북 내에서 실행하도록 설정

```
$ ipython profile create
$ vi ~/.ipython/profile_default/ipython_config.py

c.InterativeShellApp.matplotlib = 'inline'
c.InterativeShellApp.pylab = 'inline'
```

  - 주피터 노트북 서버 실행

```
$ jupyter notebook --ip=<노트북 서버 주소> --no_browser
```

  - 클라이언트 웹 브라우저를 통한 접속
    - ✓ URL: http://<노트북 서버 주소>:8888

주피터 노트북

- 설치 – **Windows**
  - Anaconda가 설치된 것으로 가정 (https://www.anaconda.com/distribution/)
    - ✓ 아나콘다는 운영체제 맞는 패키지를 미리 준비해 놓은 배포판
    - ✓ 수백 개의 파이썬 패키지를 포함하고 있음
  - Anaconda가 설치되면 주피터 노트북이 함께 설치 됨
    - ✓ 그러나, 최신 버전이 아닌 경우가 있으므로 직접 설치한다
    - ✓ 윈도우 시작 메뉴에서 Anaconda Prompt를 실행

```
c:\> pip install --upgrade pip
c:\> pip install jupyter
```

  - 기타 설정 부분은 앞서 '리눅스에서 주피터 노트북 설치' 부분을 참고
    - ✓ 단, 출력 화면에서 '**jupyter_notebook_config.py**' 와 '**ipython_config.py**' 파일이 생성되는 위치를 확인해 둘 것!!

주피터 노트북

- 설치 – **Windows**
  - 추가 패키지 설치

```
(tf1) c:\> pip install scipy
(tf1) c:\> pip install --upgrade sklearn
(tf1) c:\> pip install --upgrade pandas
(tf1) c:\> pip install --upgrade pandas-datareader
(tf1) c:\> pip install --upgrade matplotlib
(tf1) c:\> pip install --upgrade pillow
(tf1) c:\> pip install --upgrade tensorflow-hub
(tf1) c:\> pip install --upgrade h5py
(tf1) c:\> pip install --upgrade keras
```

가상 환경 (Virtual Environment)

- 가상 환경
  - 사용자가 정한 임의의 디렉터리에 파이썬과 관련 패키지를 함께 넣어 독립적인 개발 환경을 제공
  - 하나의 머신에 여러 개의 가상 환경을 만들어 패키지들을 독립적으로 설치
    - ✓ 패키지 버전에 따른 의존성 문제 해결
- 가상 환경을 위한 툴 설치와 가상 환경 생성 – 리눅스 **(Ubuntu** 기준)

```
$ sudo apt-get install python3-dev python3-venv

# 모든 가상 환경은 venv 디렉터리 아래에 만들기로 한다.
$ mkdir ~/venv

# tf1 이라는 이름의 가상 환경을 만든다
$ python3 -m venv --system-site-package ~/venv/tf1

#  리스트 명령으로 생성된 내용 확인.
$ ls ~/venv/tf1
bin  include  lib  lib64  pyvenv.cfg  share
```

가상 환경 (Virtual Environment)

- 가상 환경 활성화 / 비활성화

```
# 가상 환경 활성화
$ source ~/venv/tf1/bin/activate

# 가상 환경 비활성화
(tf1) $ deactivate
```

- 가상 환경 제거
  - 가상 환경으로 생성한 디렉터리를 삭제한다

```
$ rm -rf ~/venv/tf1
```

가상 환경 (Virtual Environment)

- 가상 환경 생성 – 윈도우 **(Anaconda 기준)**
  - 프로젝트에 따라 설치할 패키지 버전이 다르고, 프로젝트 별로 배타적 관리가 필요할 경우의 해결책
  - 다음은 “tf1”이라는 이름의 가상 환경과 python 3.6 버전을 설치한다

```
c:\> conda update --all
c:\> conda create --name tf1 python=3.6
c:\> conda activate tf1
```

- 가상 환경 활성화 / 비활성화 – 윈도우 **(Anaconda 기준)**

```
# 가상 환경 활성화
C:\> conda activate tf1

# 가상 환경 비활성화
(tf1) c:> conda deactivate
```

가상 환경 (Virtual Environment)

- 가상 환경 제거 **(Anaconda** 기준**) –** 윈도우 **(Anaconda** 기준**)**

```
C:\> conda remove --name <가상환경이름> --all   // 가상환경 삭제
```

- 가상 환경 설정 저장 및 실행 **–** 윈도우 **(Anaconda** 기준**)**

```
C:\> conda env export –n <가상환경 이름> --file <파일 이름>.yaml(yml)   // 저장
C:\> conda env create –f <파일이름>.yml   // 실행
```

텐서플로우 (TensorFlow)

- 소개
  - 텐서플로우는 머신 러닝과 딥 러닝 문제 해결을 위해 알고리즘을 가진 프레임워크
  - 구글에서 2011년에 개발을 시작하여 2015년에 공개한 머신 러닝 라이브러리
  - 기본적으로 C++로 작성되었지만 파이썬, 자바, 고(Go) 등 다양한 언어를 지원
  - 윈도우, 맥, 리눅스, 안드로이드, iOS, 라즈베리 파이 등 다양한 시스템에서 사용할 수 있도록 지원
  - 머신 러닝/딥 러닝을 위한 다양한 라이브러리가(Torch, Caffe, Chainer, CNTK 등) 있지만 활성화된 커뮤니티가 가장 크다.

- 텐서플로 **v1.x vs. v2.x**
  - 2019년 이후, v2.x 버전 배포
  - V2.x 버전은 v1.x 버전과 달리 CPU와 GPU 버전이 통합
  - GPU 버전은 Nvidia의 GPGPU를 사용함으로 CUDA 드라이버와 cuDNN 드라이버가 필요
    - ✓ GPU 버전은 CPU 성능과는 별 관계없이 GPU 성능이 중요

텐서플로우 (TensorFlow)

- **v2.x** 버전에서 개선된 점
  - High-level API를 담당하는 **Keras**가 통합되어 사용의 편의성이 향상
  - 플랫폼 호환성의 향상
    - ✓ **TensorFlow.js**: 자바스크립트 라이브러리로서 모델을 브라우저나 Node.js 에 손쉽게 배포
    - ✓ **TensorFlow Lite**: 모바일, 임베디드 디바이스 등에 배포하기 위해 경량화(Lightweight) 버전의 라이브러리
    - ✓ **TensorFlow Extended**: 대규모 제작 환경에 쓰이는 End-to-End 플랫폼
  - 과감한 API 변경
    - ✓ 많은 API들이 사라졌거나 이름이 변경
  - **Eager execution 지원 – Session**의 개념이 삭제
    - ✓ 1.x 버전에서는 session 안에서 연산이 이루어졌던 것에 비해 session을 선언하고 실행할 필요가 없어졌다

- 텐서플우 v1.x와 v2.x는 호환성이 없으며, v1.x 버전으로 작성된 참고할 만한 좋은 예제들이 많기 때문에 우리는 두 가지 버전 모두를 설치하여 사용한다

텐서플로우 (TensorFlow) v1.x

- 설치 – **AI-LAB-OBJ (Jetson-TX2)**
  - 시스템 패키지 설치

```
$ sudo apt-get install libhdf5-serial-dev hdf5-tools libhdf5-dev zlib1g-dev zip libjpeg8-
dev liblapack-dev libblas-dev gfortran
```

  - pip3 업그레이드

```
$ sudo -H python3 -m pip install -U pip testresources setuptools==49.6.0
```

  - python 의존성 패키지 설치

```
$ sudo -H pip install -U --no-deps numpy==1.19.4 future==0.18.2 mock==3.0.5
keras_preprocessing==1.1.2 keras_applications==1.0.8 gast==0.4.0 protobuf pybind11 cython
pkgconfig

$ sudo -H env H5PY_SETUP_REQUIRES=0 python3 -m pip install -U h5py==3.1.0
```

텐서플로우 (TensorFlow) v1.x

- 설치 – **AI-LAB-OBJ (Jetson-TX2)**
  - 가상 환경에 Nvidia에서 제공하는 텐서플로우 설치 파일
  - 가상 환경 생성 및 활성화

```
$ python3 -m venv --system-site-package ~/venv/tf1
$ source ~/venv/tf1/bin/activate
```

  - pip 업그레이드

```
(tf1) $ python3 -m pip install --upgrade --ignore-installed pip setuptools
```

  - 의존성 파일 설치

```
(tf1) $ python3 -m pip install --no-dependencies numpy grpcio absl-py py-cpuinfo
psutil portpicker six mock requests gast h5py astor termcolor protobuf keras-
applications keras-preprocessing wrapt google-pasta setuptools testresources
```

텐서플로우 (TensorFlow) v1.x

- 설치 – **AI-LAB-OBJ (Jetson-TX2)**
  - 가상 환경에 Nvidia에서 제공하는 텐서플로우 설치 파일
  - Nvidia에서 제공하는 텐서플로우 v1.x 설치

```
(tf1) $ python3 -m pip install --pre --extra-index-url
https://developer.download.nvidia.com/compute/redist/jp/v46 'tensorflow-gpu<2'
```

- 설치 확인
  - 다음 명령 실행으로 유사한 결과를 보이면 설치가 성공한 것이다

```
(tf1) $ python3 -c 'import tensorflow as tf; print(tf.__version__);'

2022-01-26 11:36:26.252454: I
tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened
dynamic library libcudart.so.10.0
1.15.0
```

텐서플로우 (TensorFlow) v2.x 추가 설치

- 가상 환경에서 텐서플로우 **v2.x** 설치
  - xlocale.h 에러 방지를 위해 다음 심볼릭 링크를 만든다

```
$ sudo ln -s /usr/include/locale.h /usr/include/xlocale.h
```

  - 버전 충돌 방지를 위해 가상 환경 하에서 설치
    - ✓ 의존성 파일 등은 Tensorflow v1.x 설치 시에 이미 설치 되었다

```
nvidia@nvidia-desktop:/$ python3 -m venv --system-site-package ~/venv/tf2
nvidia@nvidia-desktop:/$ source ~/venv/tf2/bin/activate

(tf2) nvidia@nvidia-desktop:/$ wget
https://developer.download.nvidia.com/compute/redist/jp/v46/tensorflow/tensorflow-
2.6.2+nv21.12-cp36-cp36m-linux_aarch64.whl

(tf2) nvidia@nvidia-desktop:/$ python3 -m pip install tensorflow-2.6.2+nv21.12-cp36-
cp36m-linux_aarch64.whl
```

텐서플로우 (TensorFlow) v1.x

- 설치 – 윈도우 **(Anaconda** 기준**)**
  - 가상 환경 생성 및 실행

```
(base) c:\> conda update --all
(base) c:\> conda create --name tf1 python=3.6
(base) c:\> conda activate tf1
```

  - pip 업그레이드

```
(tf1) c:\> python –m pip install --upgrade pip
```

  - 텐서플로우 설치

```
(tf1) c:\> pip install tensorflow==1.15.21
```

주피터 노트북과 가상환경 연결

- 가상 환경을 주피터 노트북 커널에 추가
  - 주피터 노트북에서 가상 환경에 설치된 패키지를 이용하려면 노트북 커널에 추가하여 연결해야 한다
  - ipykernel 설치

```
(tf1) $ python3 -m pip install ipykernel
```

  - 가상 환경과 연결하고, 출력 메시지에서 가상 환경에 등록된 것을 확인한다

```
(tf1) $ python3 -m ipykernel install --user --name tf1 --display-name "tf1"

Installed kernelspec tf1 in /home/nvidia/.local/share/jupyter/kernels/tf1
```

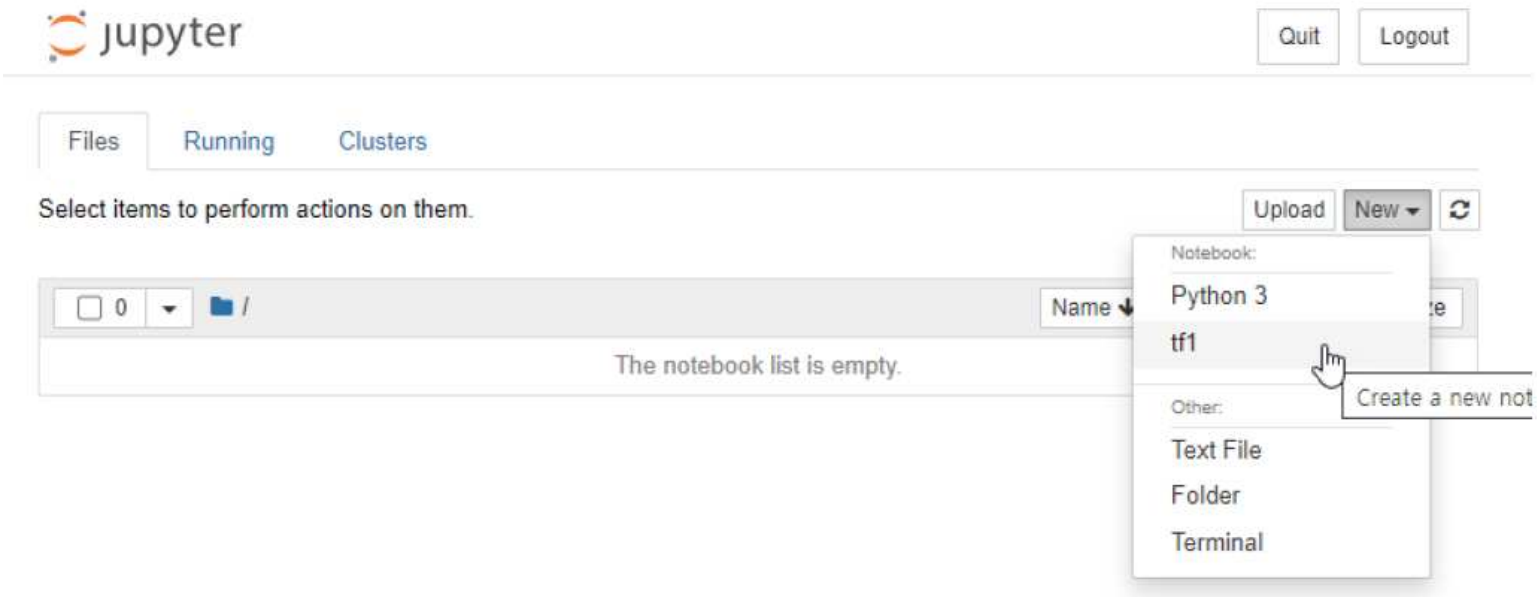
  - 커널을 삭제할 경우 다음 명령 실행

```
(tf1) nvidia@nvidia-desktop:~$ jupyter kernelspec uninstall [가상 환경 이름]
```

주피터 노트북과 가상환경 연결

- 연결 확인
  - 주피터 노트북 서버를 실행하고 웹 브라우저를 통해 접속하여 `**tf1**` 항목이 추가된 것을 확인

```
(tf1) $ jupyter notebook --ip=<서버 IP 주소> --no-browser
```



주피터 노트북 실행 옵션

- 실행 옵션
  - 명령 옵션의 도움말 표시

```
$ jupyter notebook --help
```

  - 실행 속도 향상을 위해 수식 입력 라이브러리 무효화

```
$ jupyter notebook --no-mathjax    # javascript 기반의 수식 입력 라이브러리
```

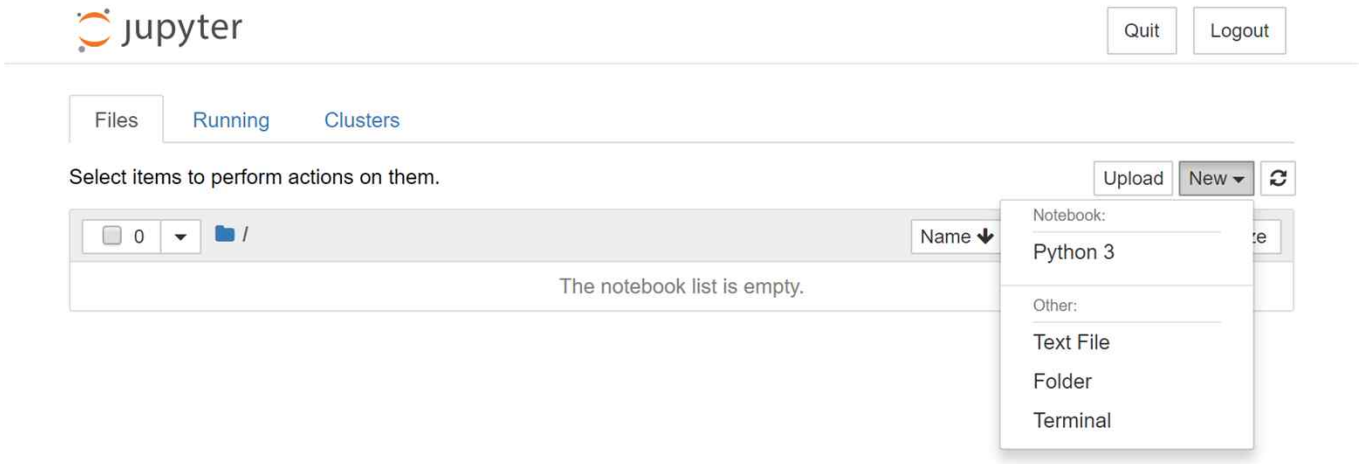
  - 웹 브라우저를 지정하거나, 하지 않거나, 포트번호 설정하기

```
$ jupyter notebook --browser=“Safari!
$ jupyter notebook --no-browser
$ jupyter notebook --port=8889
```

- 실행 종료
  - 명령 창에서 **Ctrl+C** 키를 입력

주피터 노트북 사용

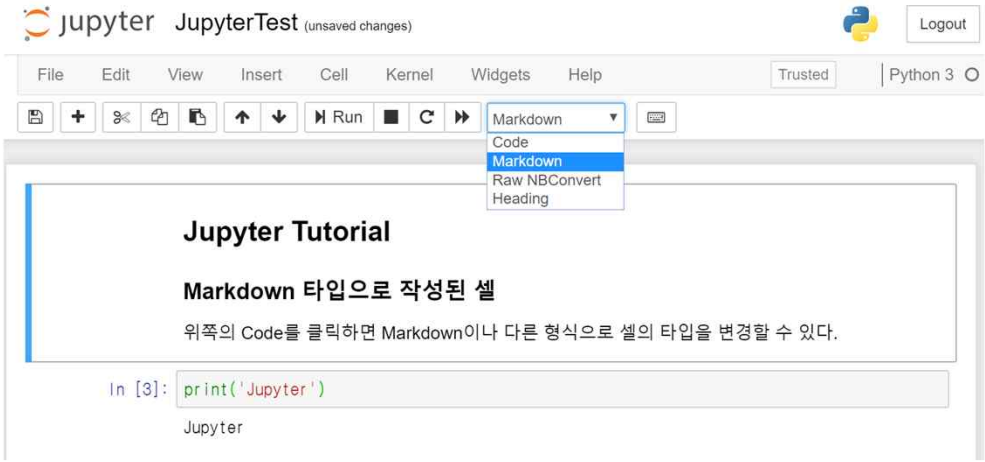
- 새 파일 생성
  - 화면 우측 상단의 **New** 버튼 클릭
    - ✓ Python 3, Text File, Folder, Terminal 등의 옵션을 이용할 수 있다



    - ✓ 기본적으로 Untitled라는 이름으로 생성되며, 언제든지 원하는 이름으로 변경할 수 있다

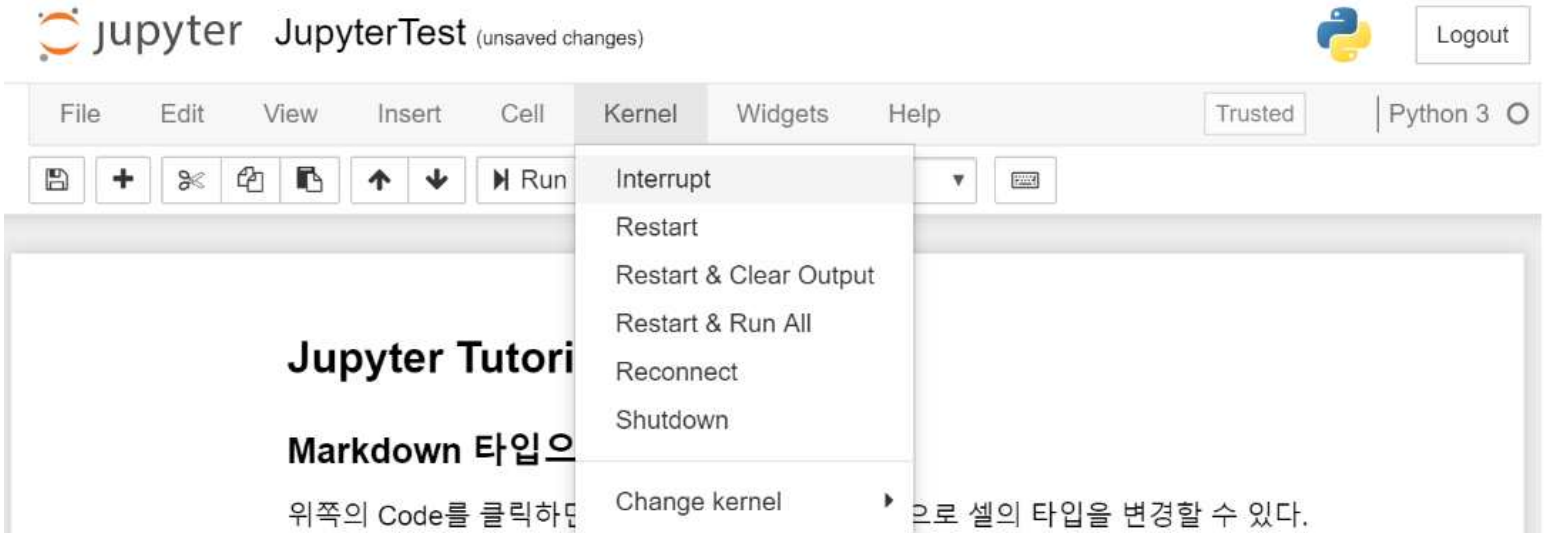
주피터 노트북 사용

- **펀집 / 명령 모드**
  - 편집모드(초록색 테두리): 셀의 내용을 편집 – 엔터 키를 누르면 진입
  - 명령모드(파란색 테두리): 셀 자체를 조작 – Esc 키를 누르면 진입
- **셀 타입**
  - 코드(Code): 일반 코드를 수행할 수 있는 셀 (디폴트, 단축키 = y)
  - 마크다운(Markdown): 마크다운으로 셀의 내용을 작성 (단축키 = m)
    - ✓ Shift + Enter 키를 누르면 마크다운이 실제 보여지는 방식으로 변경되며, 다시 수정하려면 Enter 또는 더블 클릭하면 편집 가능



주피터 노트북 사용

- 셀 실행
  - 셀에 커서를 위치 시킨 후, **Shift + Enter** (현재 셀 아래 새로운 셀을 만듬) 혹은 **Ctrl + Enter** 키를 누른다
    - ✓ 셀 옆 In[], Out[] 의 괄호 안 숫자는 몇 번째 실행했는지를 표시
- 실행 중단 / 재실행
  - 커널(Kernel) 탭 – 커널을 IPython (Interface Python) 아래에 백그라운드로 동작하며 IPython 대화창을 관리



주피터 노트북 사용

- 커널 탭 기능
  - Interrupt: 실행 중인 코드를 강제로 중지
  - Restart: 중지된 코드를 재시작. 코드나 실행 결과는 삭제되지 않는다.
  - Restart & Clear Output: 실행 결과를 삭제하고 코드 재시작
  - Restart & Run All: 모든 셀의 코드를 위에서부터 순차적으로 재실행
  - Reconnect: 인터넷 연결이 끊어졌을 때 연결을 재시도한다
  - Shutdown: 커널을 종료한다. 실행 결과는 삭제되지 않는다
- **Home** 화면의 **New** 메뉴
  - Text File: .py 나 .txt 파일 등을 만듬
    - ✓ 이 파일은 터미널에서 실행시켜야 한다 (IPython 창에서는 읽기만 가능)
  - Folder: 디렉터리 생성
  - Terminal: 윈도우의 CMD 창과 같은 터미널 창을 연다.
    - ✓ .py 파일 실행, 파일 목록 보기 및 삭제 등의 명령이 모두 가능하며, Running 탭에서 중지시킬 수 있다.

주피터 노트북 사용

- 파일 변경, 삭제, 복제
  - Home 화면의 Files 탭 이용 – 터미널 창에서도 가능

주피터 노트북 사용

- 단축키
  - 단축키 정보는 [Help] - [Keyboard Shortcuts] 또는 명령 모드에서 H를 눌러서 표시할 수 있다
  - 공용 단축키

공용 단축키	설명
Shift + Enter	액티브 셀을 실행하고 아래 셀을 선택한다.
Ctrl + Enter	액티브 셀을 실행한다.
Alt + Enter	액티브 셀을 실행하고 아래에 셀을 하나 생성한다.

주피터 노트북 사용

- 단축키
  - 편집모드 단축키

편집 모드 단축키	설명
Ctrl + Z	Undo 명령이다.
Ctrl + Shift + Z	Redo 명령이다.
Tab	자동완성 또는 Indent를 추가한다.

  - 명령모드 단축키

명령 모드 단축키	설명
↑,	셀 선택
A	액티브 코드 셀의 위(Above)에 셀을 하나 생성한다.
B	액티브 코드 셀의 위(Below)에 셀을 하나 생성한다.
Ctrl + S	Notebook 파일을 저장한다.

주피터 노트북 사용

- 단축키
  - 명령모드 단축키

명령 모드 단축키	설명
Shift + L	줄 번호 표시를 토글한다.
D, D	(D 두번 연속으로 타이핑)액티브 코드 셀을 삭제한다.
Z	삭제한 셀을 하나 복원한다.
Y	액티브 코드 셀을 Code 타입(코드를 기술하는 타입)으로 한다.
M	액티브 코드 셀을 Markdown 타입으로 한다.
O, O	커널을 재시작한다.
P	명령 팔레트를 연다.
H	단축키 목록을 표시한다. Enter 키로 숨긴다.

주피터 노트북 사용

- **DocString** 표시
  - 선언한 변수 뒤에 ?를 붙여서 셀을 실행하는 것으로 해당 변수의 상태를 확인
  - 변수를 타이핑한 후 Shift + Tab을 누르면 툴팁이 표시
    - ✓ 툴팁에는 DocString의 일부 내용이 표시
- 이미지 첨부
  - Drag & Drop으로 첨부
- **Shell** 명령어
  - 터미널에서 사용하는 명령을 그대로 쓰되, 맨 앞에 !를 붙임
- 매직 명령어
  - 맨 앞에 %를 붙여 특정 명령을 수행. 파이썬 문법에는 포함되지 않은 IPython 커널이 제공하는 기능.
    - ✓ %는 라인 단위 명령이고, %% 셀 단위 명령이다
  - 코드 셀에서 **%lsmagic** 명령을 수행하면 전체 매직 명령어 리스트를 볼 수 있다

numpy

- 주피터 노트북에서 진행한다
- **numpy** 배열 생성
  - **numpy.array()** 메서스 이용
  - 파이썬 리스트를 인수로 받아 특수한 형태의 배열 **(numpy.ndarray)**을 반환

```
# numpy 가져오기. numpy를 np라는 이름으로 사용.
import numpy as np

x = np.array([1, 2, 3])
X

결과:
array([1, 2, 3])
```

```
type(x)

결과:
numpy.ndarray
```

numpy

- **numpy arange – 특정 범위의 수열**
  - **Syntax: np.arange(시작점(생략 시 0), 끝점(미포함), step size(생략 시 1))**
- **Python의 range 함수와 차이**
  - **range** 함수는 정수 단위만 지원하나, **np.arange**는 실수 단위도 표현 가능하다
  - **range** 함수는 **range iterator** 자료형을 반환하고, **np.arange** 메소드는 **numpy array** 자료형을 반환

```
y = range(10)
Y

결과: range(0, 10)

y = np.arange(10)
Y

결과: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

numpy

- **numpy 산술 연산**
  - 연산에 사용될 원소의 수가 같아야 한다
  - 연산은 원소 별(element-wise)로 수행
  - 넘파이 배열과 단일 값(스칼라 값)의 연산이 가능
    - ✓ 이 기능을 브로드캐스트(broadcast) 라 함

```
x = np.array([1.0, 2.0, 3.0])
y = np.array([2.0, 4.0, 6.0])

# 원소별 덧셈/곱셈
x+y

결과: array([3., 6., 9.])
```

```
# 원소별 곱셈/나눗셈
x * y

결과: array([ 2.,  8., 18.])
```

```
# 스칼라 연산
x = np.array([1.0, 2.0, 3.0])
x / 2.0

결과: array([0.5, 1. , 1.5])
```

numpy

- **numpy N차원 배열**
  - numpy는 다차원 배열을 지원

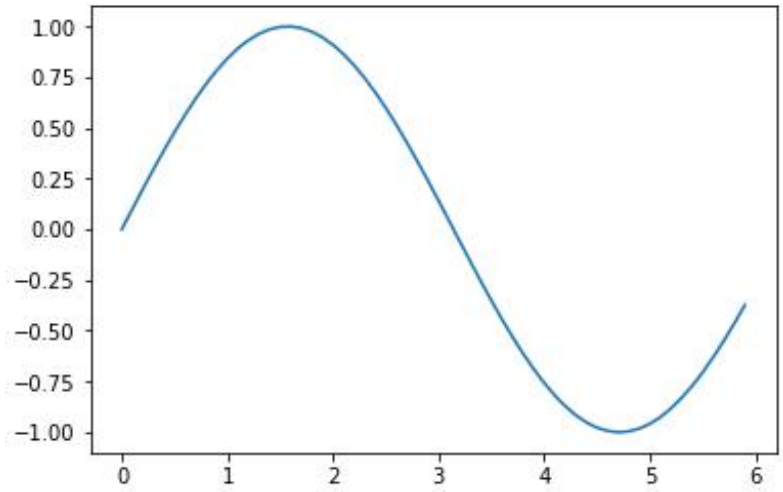
```
# 2x2 행렬
A = np.array([[1, 2], [3, 4]])
print(A)

# 행렬 모양 출력
print(A.shape)

# 행렬 원소의 자료형 출력
print(A.dtype)

출력:
[[1 2]
 [3 4]]
(2, 2)
int64
```

```
B = np.array([[3.0], [0.6]])

# 행렬의 산술 연산
print(A+B)
print(A*B)

# 2차원 행렬에 대한 스칼라 곱
print(A*10)

결과: [[4.  5. ]
 [3.6 4.6]]
[[3.  6. ]
 [1.8 2.4]]
[[10 20]
 [30 40]]
```

matplot

- **matplotlib** 라이브러리
  - 데이터 시각화와 그래프 그리기를 지원
  - 그래프 그리기는 matplotlib의 **pyplot** 모듈을 사용

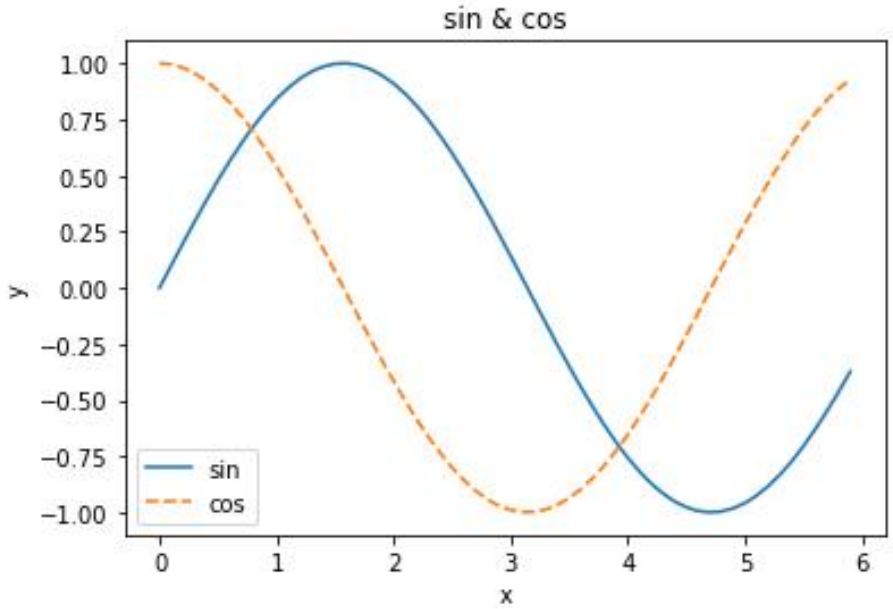
```
import matplotlib.pyplot as plt

# 데이터 준비: 0에서 6까지 0.1 간격으로 생성
x = np.arange(0, 6, 0.1)
y = np.sin(x)

# 그래프 그리기
plt.plot(x, y)
plt.show()
```



matplot

- **pyplot 기능**
  - 그래프의 제목과 축 이름 표시

```
# 데이터 준비
x =  np.arange(0, 6, 0.1)
y1 = np.sin(x)
y2 = np.cos(x)

# 그래프 그리기
plt.plot(x, y1, label='sin')
# cos 함수는 점선으로 표시
plt.plot(x, y2, linestyle="--", label="cos")
plt.xlabel("x")  # x축 이름
plt.ylabel("y")  # y축 이름
plt.title('sic & cos') # 제목
plt.legend()  # 범례 추가
plt.show
```



matplot

- **pyplot 기능**
  - 이미지 표시
    - ✓ **matplotlib.image** 모듈의 **imshow()** 와 **imread()** 메서드 이용

```
from matplotlib.image import imread

# 이미지 가져오기
img = imread('/home/nvidia/nb_examples/lena.png')

plt.imshow(img)
plt.show()
```

  - 현재 지원하는 이미지 타입은 **'.png'** 이므로, 다양한 이미지 형식을 사용하고 싶다면 **cv2, pillow** 등의 다른 패키지들을 이용할 수 있다

pandas

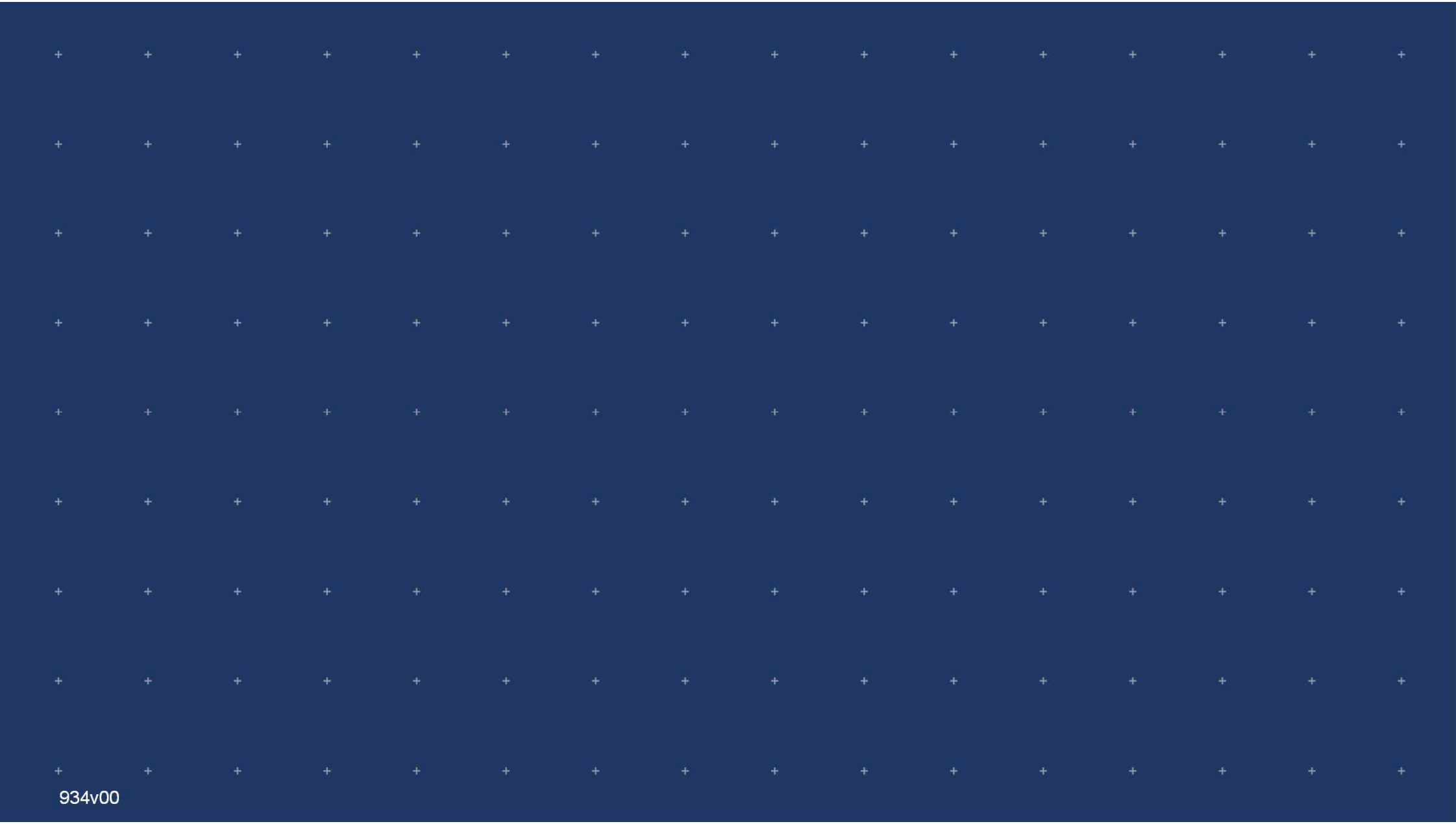
- **pandas DataFrame 생성**

```
from IPython.display import display
import pandas as pd

# 간단한 인명정보를 생성
data = {'Name': ["John", "Anna", "Peter", "Linda"], 'Location' : ["New York",
"Paris", "Berlin", "London"], 'Age' : [24, 13, 53, 33] }

# Pandas Datafram 생성
data_pandas = pd.DataFrame(data)

# IPython.display는 주피터 노트북에서 DataFrame을 이쁘게 출력한다.
display(data_pandas)

# DataFrame에 입력울 정렬하여 추가
my_series = pd.Series({"United Kingdom":"London", "India":"New Delhi", "United
States":"Washington", "Belgium":"Brussels"})
display(pd.DataFrame(my_series))

# DataFrame 입력으로 DataFram 넣기
my_df = pd.DataFrame(data=[4, 5, 6, 7], index=range(0, 4), columns=['A'])
display(pd.DataFrame(my_df))
```

	Age	Location	Name
0	24	New York	John
1	13	Paris	Anna
2	53	Berlin	Peter
3	33	London	Linda

	0
Belgium	Brussels
India	New Delhi
United Kingdom	London
United States	Washington

	A
0	4
1	5
2	6
3	7


934v00