실습 Zero
:인공지능에 자주 쓰이는 기초 문법들

소프트웨어학과 양희경

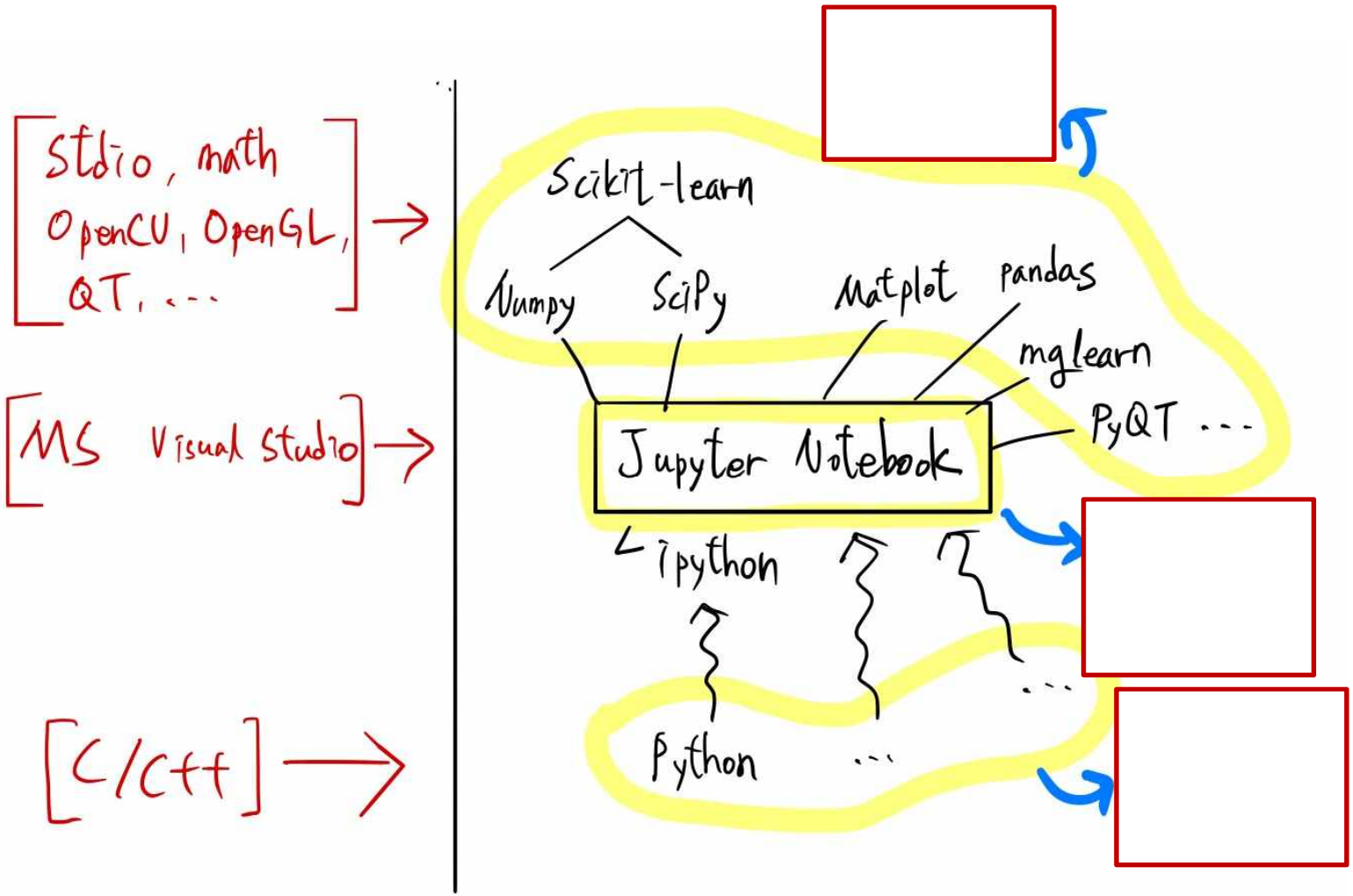
실습 내용

0. Colab 접속 및 Jupyter notebook 사용법

1. 파이썬 기초(Optional)

2. 벡터, 행렬 연산, 그래프 그리기

머신러닝 프레임웍


stdio, math
OpenCV, OpenGL,
QT, ...
MS Visual Studio
C/C++
Scikit-learn
Numpy
SciPy
Matplot
pandas
mglearn
PyQT ...
Jupyter Notebook
ipython
Python

0. Colab

- Colaboratory (Colab) 이란?
  - Google Colaboratory = Google Drive + Jupyter Notebook
  - Python3 을 지원하는 Jupyter Notebook 의 구글 커스텀 버전

추가 지식

Jupyter Notebook 이란?

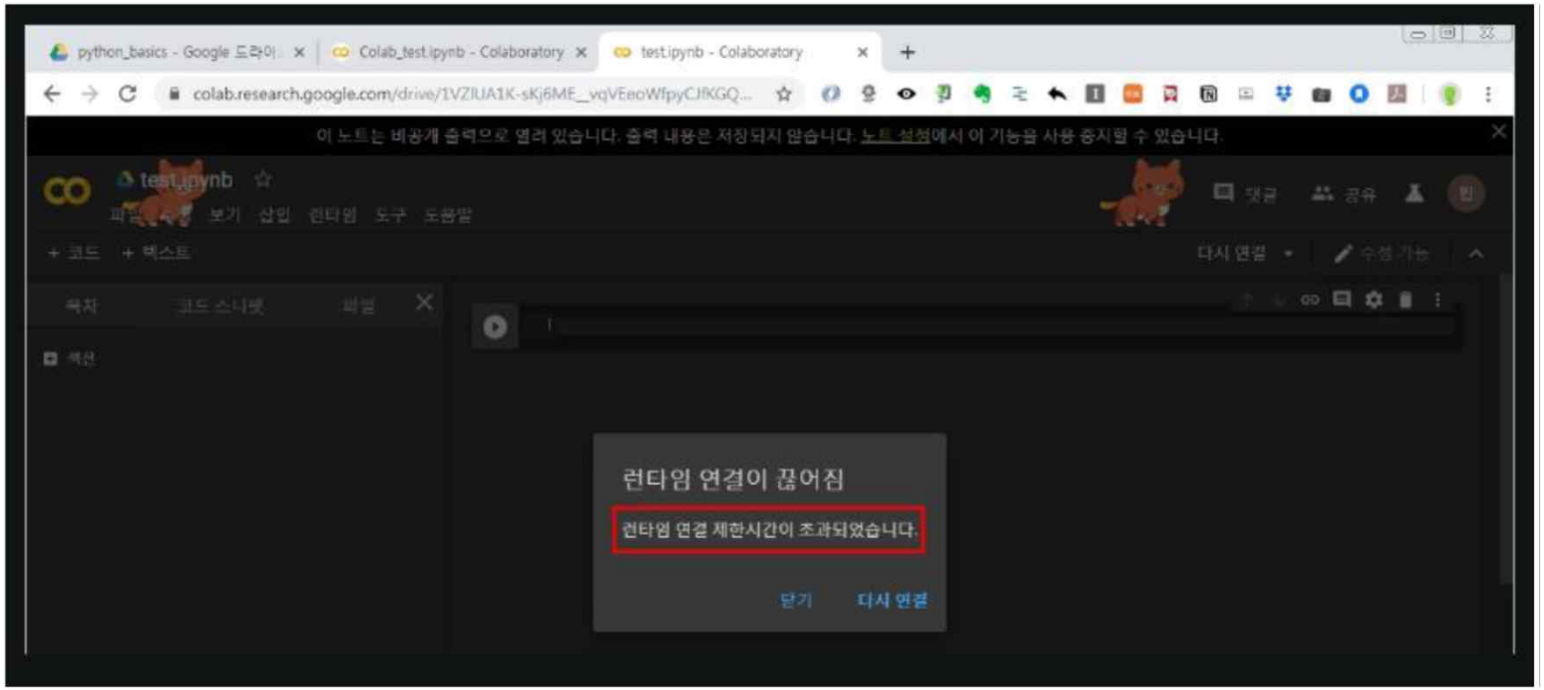
웹 브라우저에서 파이썬 코드를 작성하고 실행할 수 있는 개발 도구

0. Colab

- Colab 의 장점
  - 구글 드라이브와 연동이 가능하다.
  - GitHub 와 연동이 가능하다.
  - **GPU 가속을 무료로 받을 수 있다.** (무료버전 12시간 제약)
  - 협업 가능

0. Colab

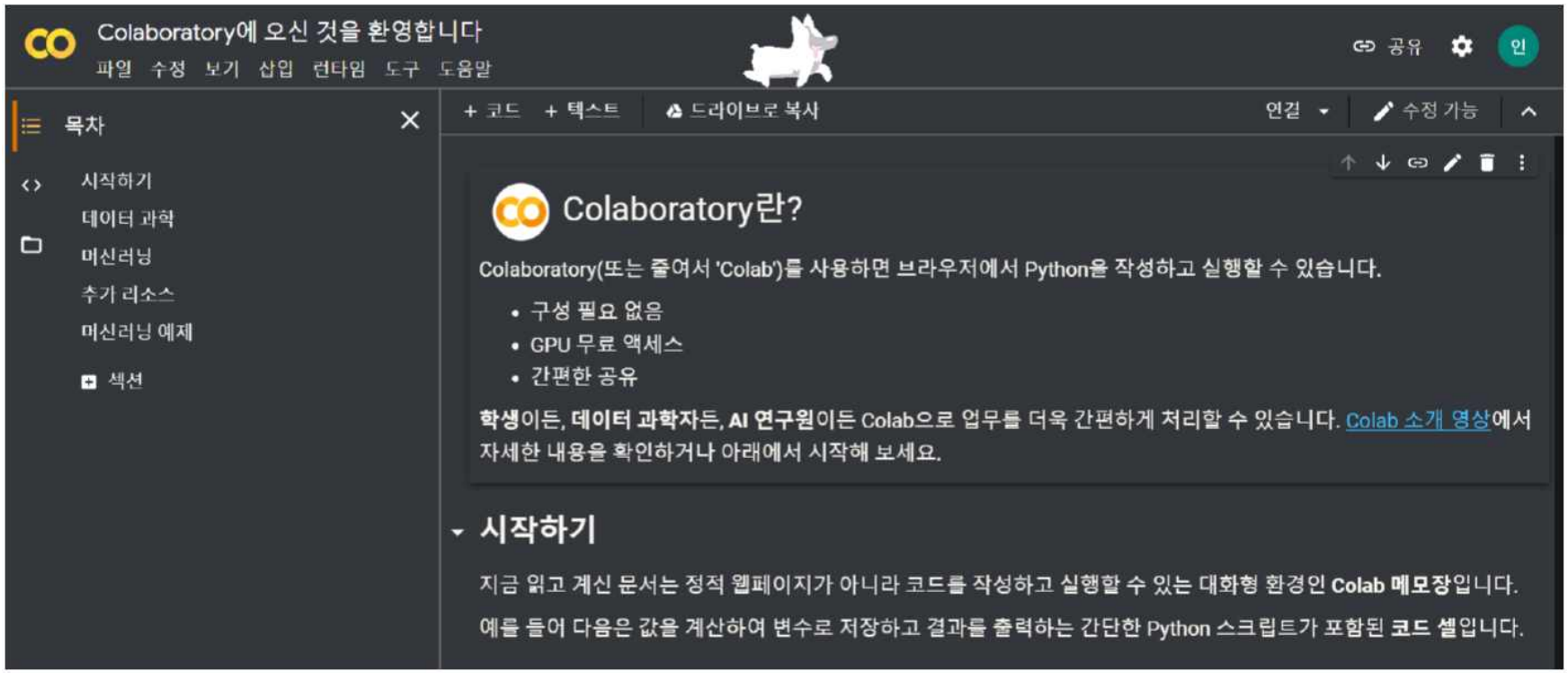
- Colab 유의할 점
  - 최대 세션 유지 시간은 **12시간**
    (데이터는 소멸되나 소스코드는 자동 저장됨)



https://theorydb.github.io/dev/2019/08/23/dev-ml-colab/

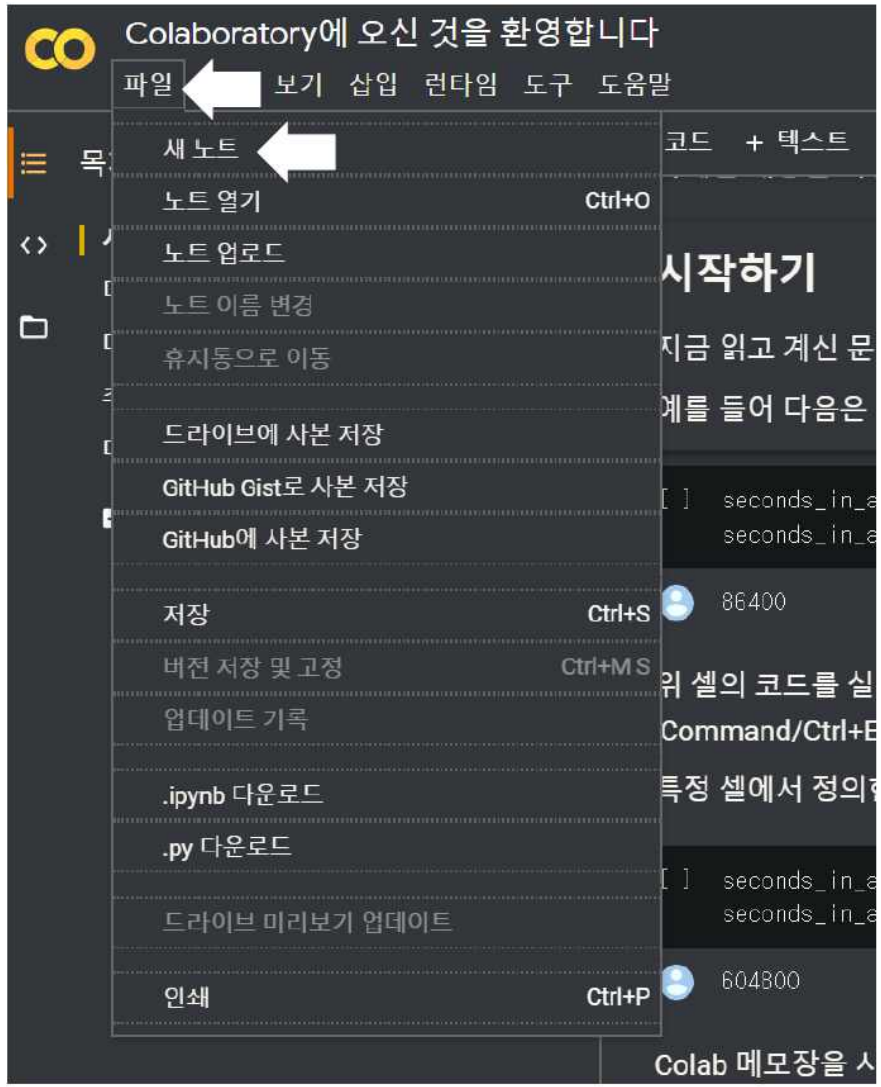
0. Colab

- Colab 사용 방법
  - 1. Google 로그인 후 https://colab.research.google.com/ 접속



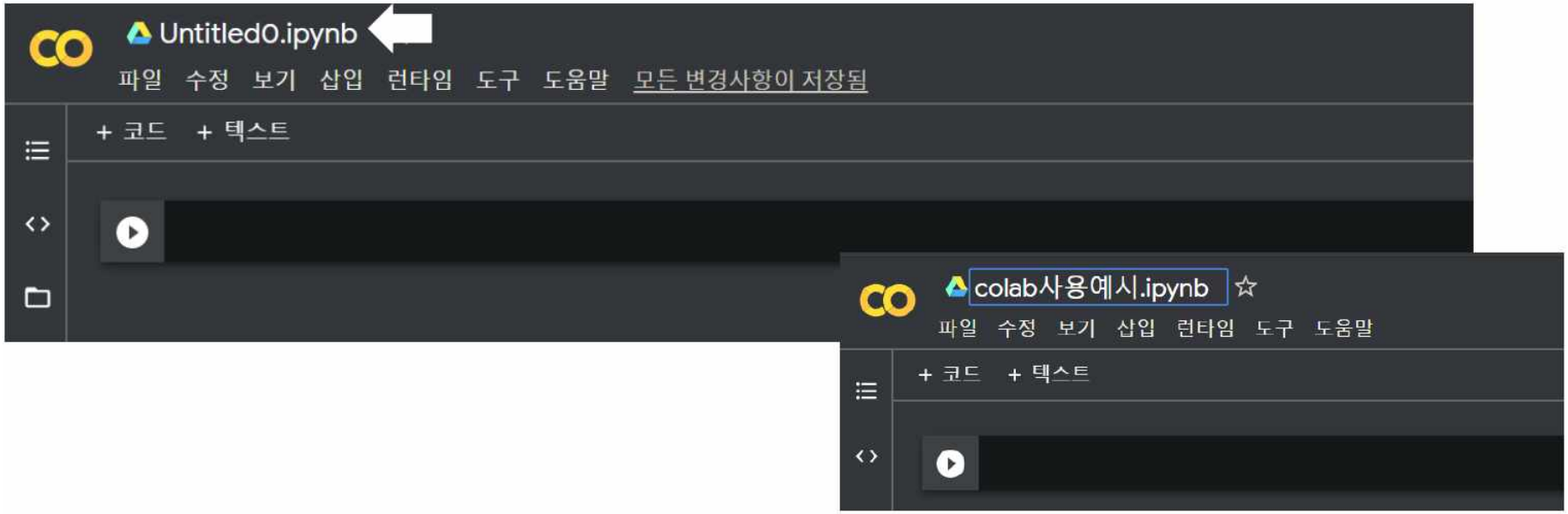
0. Colab

- Colab 사용 방법
  - 2. 파일 → 새 노트



0. Colab

- Colab 사용 방법
  - 3. 파일 이름을 클릭하여 이름 수정 가능 (Ctrl+S 로 저장)

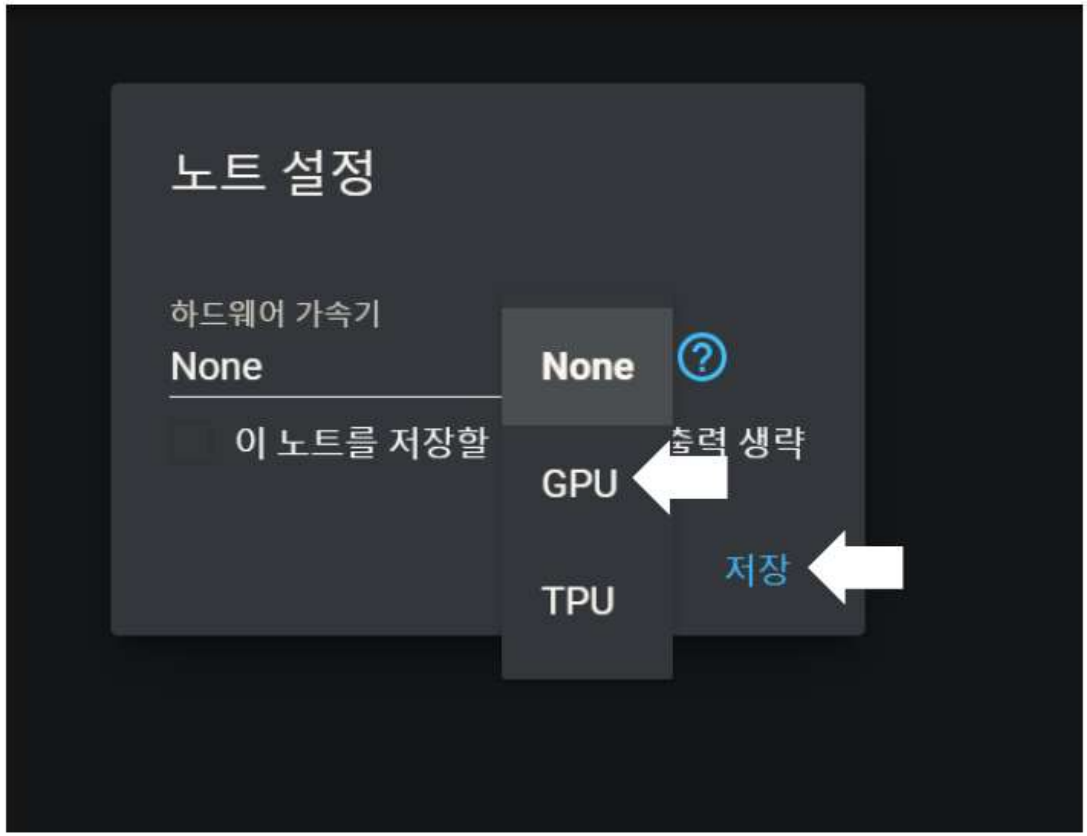


0. Colab

- Colab 사용 방법
  - 4. (Option) GPU 가속시, 수정→ 노트 설정(또는 런타임 → 런타임 유형 변경)

0. Colab

- Colab 사용 방법
  - 4. (Option) GPU 선택→ 저장



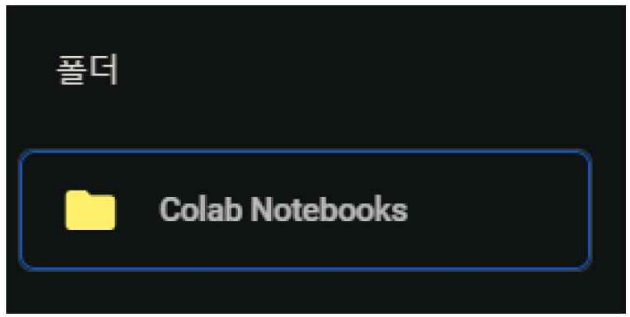
0. Colab

- Colab 사용 방법
  - 4. (Option) 주석 처리된 .cuda() 를 주석 해제하여 Run

```
for j, [imgs, labels] in enumerate(dloader):
    img = Variable(imgs,volatile=True).cuda()
    #label = Variable(labels) #y
    label = Variable(labels).cuda()
    # .cuda() : GPU에 로드되기 위함. 만약 CPU로 설정되어 있다면 에러남
```

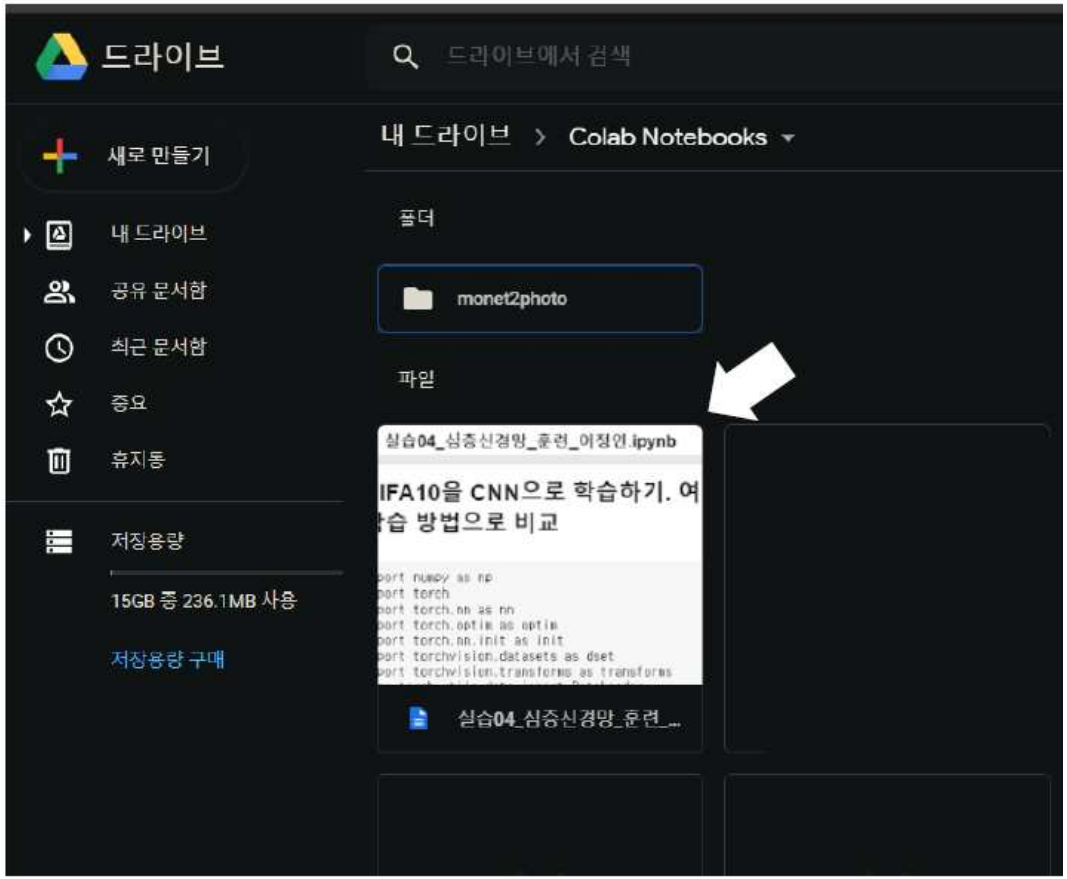
0. Colab

- Colab 사용 방법
  - 5. 구글 드라이브→ 내 드라이브→ Colab 폴더가 생성되었는 지 확인



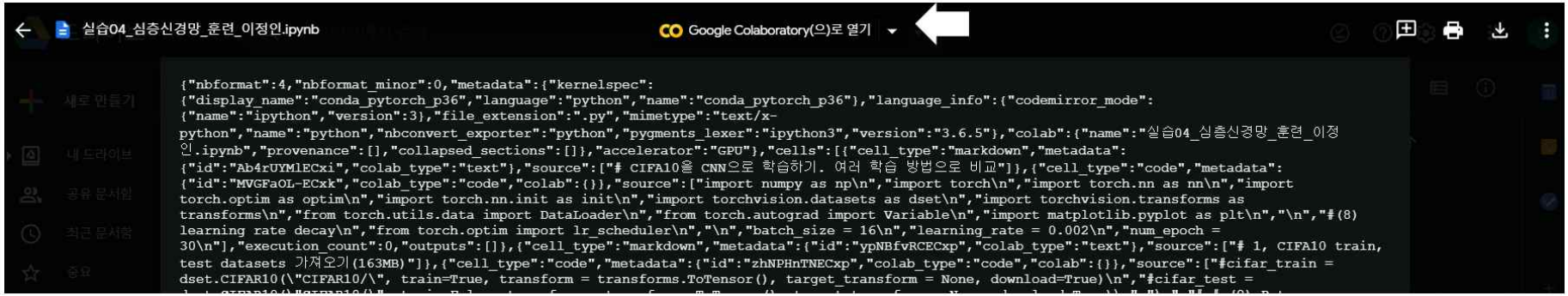
0. Colab

- Jupeter 작업물을 Colab 에서 사용할 경우
  - 1. Jupyter에서 ipynb로 다운로드한 파일을 colab 폴더에 업로드



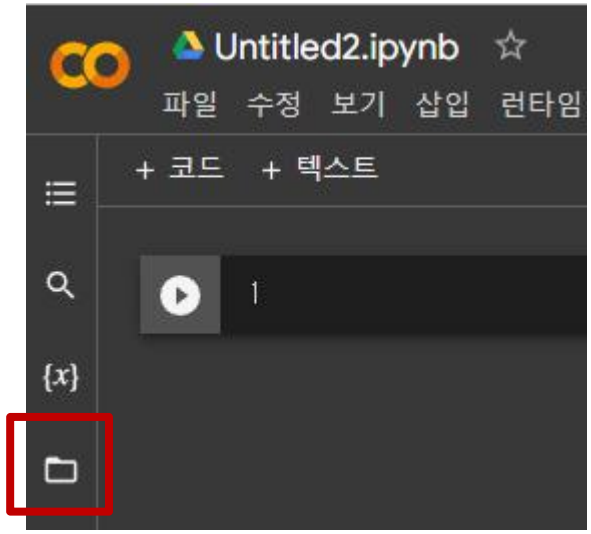
0. Colab

- Jupeter 작업물을 Colab 에서 사용할 경우
  - 2. Google Colaboratory 로 열기 클릭!



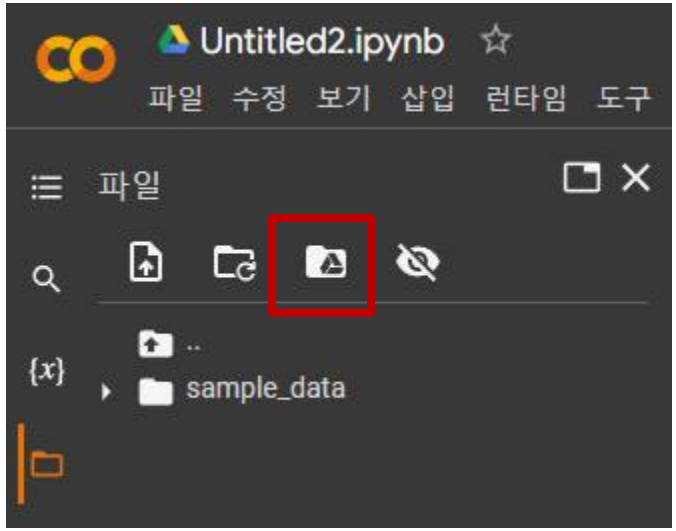
0. Colab

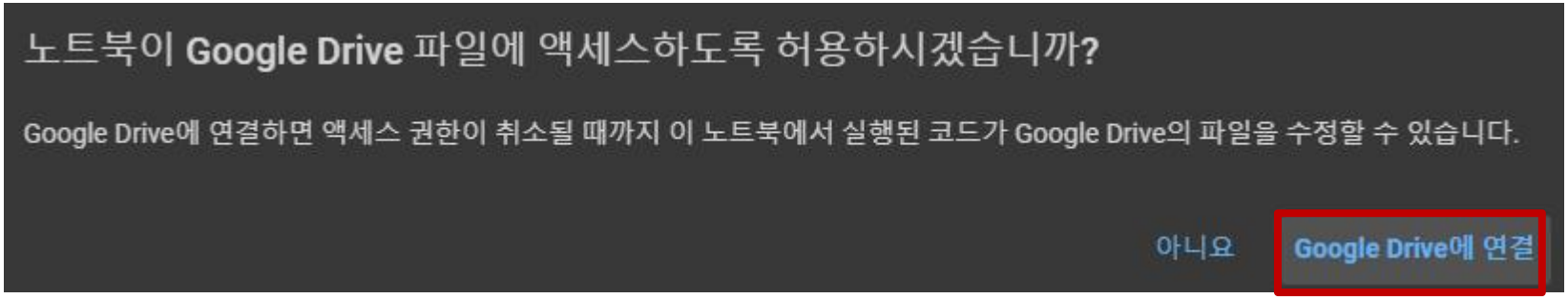
- 구글 드라이브 연동 방법
  - 왼쪽 폴더 아이콘 클릭



0. Colab

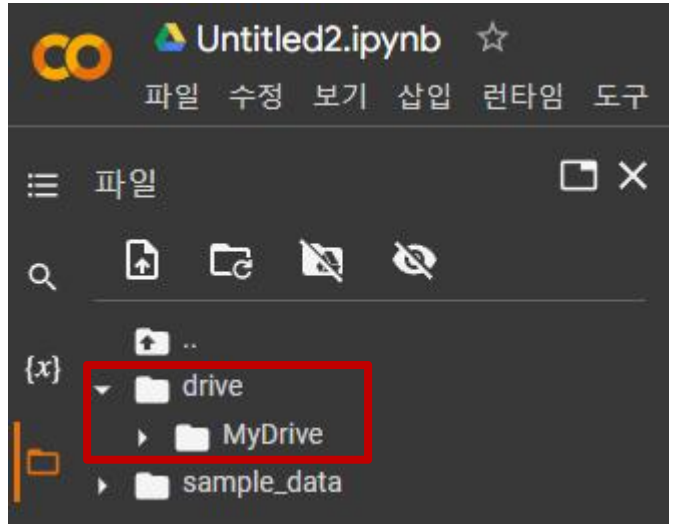
- 구글 드라이브 연동 방법
  - '드라이브 마운트' 클릭
  - 'Google Drive에 연결' 클릭





0. Colab

- 구글 드라이브 연동 방법
  - Colab 내에서 파일 경로 지정해 줄 때 "drive/MyDrive/…" 로 접근



0. Colab

- Colab 관련 유용한 사이트
  - 1. Colab 명령어

    https://theorydb.github.io/dev/2019/08/23/dev-ml-colab/

  - 2. GitHub 연동 방법

    https://league-cat.tistory.com/305

실습 내용

0. Colab 접속 및 Jupyter notebook 사용법

1. 파이썬 기초(Optional)

2. 벡터, 행렬 연산, 그래프 그리기

1. 파이썬 기초

1. 파이썬 기초(Optional)

프린트

```
print ("Hello, world")

# integer
x = 3
print ("정수: %01d, %02d, %03d, %04d, %05d"
      % (x,x,x,x,x))

# float
x = 256.123
print ("실수: %.0f, %.1f, %.2f"
      % (x,x,x))

# string
x = "Hello, world"
print ("문자열: [%s]" % (x))
```

```
Hello, world
정수: 3, 03, 003, 0003, 00003
실수: 256, 256.1, 256.12
문자열: [Hello, world]
```

1. 파이썬 기초

반복문, 조건문

```
[2]  1 contents = ["Regression", "Classification", "SVM", "Clustering", "Dimension reduction",
     2             "NN", "CNN", "AE", "GAN", "RNN"]
     3 for con in contents:
     4   if con in ["Regression", "Classification", "SVM", "Clustering", "Dimension reduction"]:
     5     print ("%s 은(는) 기계학습 내용입니다." %con)
     6   elif con in ["CNN"]:
     7     print ("%s 은(는) convolutional neural network 입니다." %con)
     8   else:
     9     print ("%s 은(는) 심층학습 내용입니다." %con)
```

```
Regression 은(는) 기계학습 내용입니다.
Classification 은(는) 기계학습 내용입니다.
SVM 은(는) 기계학습 내용입니다.
Clustering 은(는) 기계학습 내용입니다.
Dimension reduction 은(는) 기계학습 내용입니다.
NN 은(는) 심층학습 내용입니다.
CNN 은(는) convolutional neural network 입니다.
AE 은(는) 심층학습 내용입니다.
GAN 은(는) 심층학습 내용입니다.
RNN 은(는) 심층학습 내용입니다.
```

1. 파이썬 기초

반복문과 인덱스

```
for (i,con) in enumerate(contents):
    print ("[%d/%d]: %s" %(i, len(contents), con))
```

```
[0/10]: Regression
[1/10]: Classification
[2/10]: SVM
[3/10]: Clustering
[4/10]: Demension reduction
[5/10]: NN
[6/10]: CNN
[7/10]: AE
[8/10]: GAN
[9/10]: RNN
```

1. 파이썬 기초

함수

```
def sum(a,b):
    return a+b

x = 10.0
y = 20.0
print ("%.1f + %.1f = %.1f" %(x, y, sum(x,y)))
```

```
10.0 + 20.0 = 30.0
```

1. 파이썬 기초

리스트

```
a = []
b = [1,2,3]
c = ["Hello", ",", "world"]
d = [1,2,3,"x","y","z"]
x = []
print (x)

x.append('a')
print (x)

x.append(123)
print (x)

x.append(["a", "b"])
print x
```

```
[]
['a']
['a', 123]
['a', 123, ['a', 'b']]
```

1. 파이썬 기초

딕셔너리(dictionary)

```
dic = dict()
dic["name"] = "Heekyung"
dic["town"] = "Goyang city"
dic["job"] = "Assistant professor"
print dic
```

```
{'town': 'Goyang city', 'job': 'Assistant professor', 'name': 'Heekyung'}
```

1. 파이썬 기초

클래스

```
class Student:
    # 생성자
    def __init__(self, name):
        self.name = name
    # 메써드
    def study(self, hard=False):
        if hard:
            print "%s 학생은 열심히 공부합니다." %self.name
        else:
            print "%s 학생은 공부합니다." %self.name

s = Student('Heekyung')
s.study()
s.study(hard=True)
```

```
Heekyung 학생은 공부합니다.
Heekyung 학생은 열심히 공부합니다.
```

실습 내용

0. Colab 접속 및 Jupyter notebook 사용법

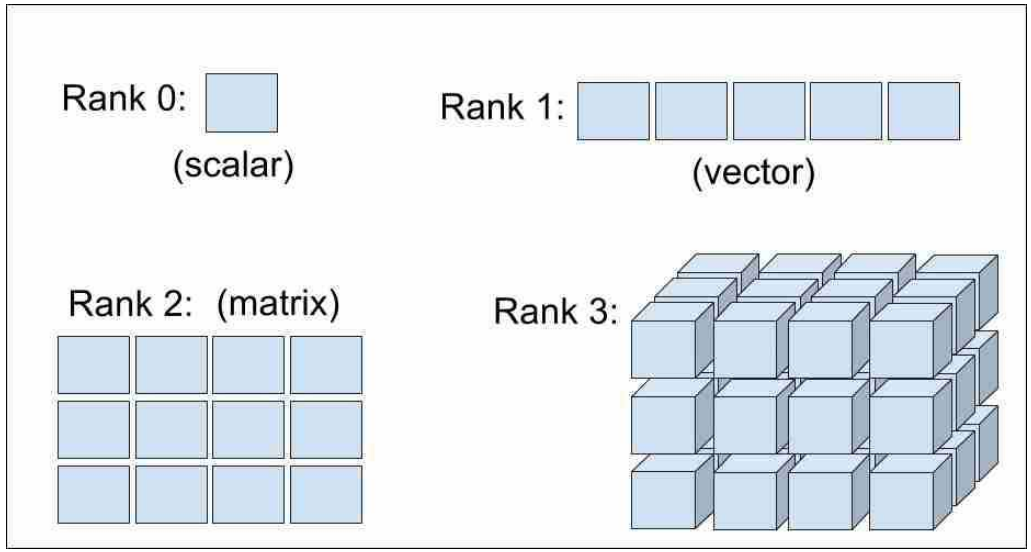
1. 파이썬 기초(Optional)

2. 벡터, 행렬 연산, 그래프 그리기

2. 벡터, 행렬 연산, 그래프 그리기

- 텐서(Tensor)?
  - 다차원 배열을 일반화한 것
  - Scalar, vector, matrix 등이 포함됨

- 랭크(Rank)?
  - 텐서의 차원
  - Rank 0 tensor: Scalar
  - Rank 1 tensor: Vector
  - Rank 2 tensor: Matrix
  - Rank n tensor



© mc.ai

2. 벡터, 행렬 연산, 그래프 그리기

라이브러리(패키지) 로드

```
import numpy as np
```

2. 벡터, 행렬 연산, 그래프 그리기

프린트

```
def print_val(x):
    print "Type:", type(x)
    print "Shape:", x.shape
    print "값:₩n", x
    print " "
```

2. 벡터, 행렬 연산, 그래프 그리기

rank 1 np array

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

x[0] = 5
print_val(x)
```

```
Type: <type 'numpy.ndarray'>
Shape: (3,)
값:
[1 2 3]

Type: <type 'numpy.ndarray'>
Shape: (3,)
값:
[5 2 3]
```

2. 벡터, 행렬 연산, 그래프 그리기

rank 2 np array

```
y = np.array([[1,2,3], [4,5,6]])
print_val(y)
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 3)
값:
[[1 2 3]
 [4 5 6]]
```

rank 2 zeros

```
a = np.zeros((2,2))
print_val(a)
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[0. 0.]
 [0. 0.]]
```

rank 2 ones

```
a = np.ones((3,2))
print_val(a)
```

```
Type: <type 'numpy.ndarray'>
Shape: (3, 2)
값:
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

rank 2 단위 행렬(identity matrix)

```
a = np.eye(3,3)
print_val(a)
```

```
Type: <type 'numpy.ndarray'>
Shape: (3, 3)
값:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

2. 벡터, 행렬 연산, 그래프 그리기

랜덤 행렬(uniform: 0~1 사이 모든 값들이 나올 확률이 같음)

```
a = np.random.random((4,4))
print_val(a)
```

```
Type: <type 'numpy.ndarray'>
Shape: (4, 4)
값:
[[0.51188499 0.24694867 0.88542043 0.3671004 ]
 [0.56884716 0.92298505 0.17967754 0.4287874 ]
 [0.01890182 0.61939264 0.95876775 0.96522488]
 [0.88609591 0.89879732 0.39578545 0.05220255]]
```

랜덤 행렬(Gaussian: 0을 평균으로 하는 가우시안 분포를 따르는 랜덤값)

```
a = np.random.randn(4,4)
print_val(a)
```

```
Type: <type 'numpy.ndarray'>
Shape: (4, 4)
값:
[[-1.09098861  1.94289236  1.54194447  0.95220594]
 [ 1.71442595 -0.96202191 -0.37320804 -1.32446336]
 [-1.97234896  0.0756659  -1.05772135  0.34975056]
 [ 0.70735129  0.64681658 -0.09711383  0.34148813]]
```

2. 벡터, 행렬 연산, 그래프 그리기

np array indexing

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print_val(a)
```

```
Type: <type 'numpy.ndarray'>
Shape: (3, 4)
값:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

```
b = a[:2, 1:3]  # 행 0~1, 열 1~2
print_val(b)
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[2 3]
 [6 7]]
```

행렬의 n번째 행 얻기

```
row1 = a[1, :]  # 1번째 행
print_val(row1)
```

```
Type: <type 'numpy.ndarray'>
Shape: (4,)
값:
[5 6 7 8]
```

2. 벡터, 행렬 연산, 그래프 그리기

행렬의 원소별 연산

```
m1 = np.array([[1,2], [3,4]], dtype=np.float64)
m2 = np.array([[5,6], [7,8]], dtype=np.float64)
```

```
# elementwise sum
print_val(m1 + m2)
print_val(np.add(m1, m2))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[ 6.  8.]
 [10. 12.]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[ 6.  8.]
 [10. 12.]]
```

2. 벡터, 행렬 연산, 그래프 그리기

```
# elementwise difference
print_val(m1 - m2)
print_val(np.subtract(m1, m2))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[-4. -4.]
 [-4. -4.]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[-4. -4.]
 [-4. -4.]]
```

```
# elementwise product
print_val(m1 * m2)
print_val(np.multiply(m1, m2))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[ 5. 12.]
 [21. 32.]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[ 5. 12.]
 [21. 32.]]
```

2. 벡터, 행렬 연산, 그래프 그리기

```
# elementwise division
print_val(m1 / m2)
print_val(np.divide(m1, m2))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[0.2        0.33333333]
 [0.42857143 0.5       ]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

```
# elementwise square root
print_val(np.sqrt(m1))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

2. 벡터, 행렬 연산, 그래프 그리기

행렬 연산

```
m1 = np.array([[1,2], [3,4]])  # (2,2)
m2 = np.array([[5,6], [7,8]])  # (2,2)
v1 = np.array([9,10])  # (2,1)   # [[9,10]]  (1,2)
v2 = np.array([11,12])  # (2,1)

print_val(m1)
print_val(m2)
print_val(v1)
print_val(v2)
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[1 2]
 [3 4]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[5 6]
 [7 8]]

Type: <type 'numpy.ndarray'>
Shape: (2,)
값:
[ 9 10]

Type: <type 'numpy.ndarray'>
Shape: (2,)
값:
[11 12]
```

2. 벡터, 행렬 연산, 그래프 그리기

벡터-벡터 연산

```
print_val(v1.dot(v2))
print_val(np.dot(v1, v2))
```

```
Type: <type 'numpy.int64'>
Shape: ()
값:
219

Type: <type 'numpy.int64'>
Shape: ()
값:
219
```

2. 벡터, 행렬 연산, 그래프 그리기

벡터-행렬 연산

```
print_val(m1.dot(v1))  # (2,2) x (2,1) -> (2,1)
print_val(np.dot(m1, v1))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2,)
값:
[29 67]

Type: <type 'numpy.ndarray'>
Shape: (2,)
값:
[29 67]
```

2. 벡터, 행렬 연산, 그래프 그리기

행렬-행렬 연산

```
print_val(m1.dot(m2))
print_val(np.dot(m1, m2))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[19 22]
 [43 50]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[19 22]
 [43 50]]
```

2. 벡터, 행렬 연산, 그래프 그리기

전치 행렬 (transpose)

```
print_val(m1)
print_val(m1.T)
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[1 2]
 [3 4]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[1 3]
 [2 4]]
```

2. 벡터, 행렬 연산, 그래프 그리기

합

```
print_val(np.sum(m1))  # 행렬의 모든 원소의 합
print_val(np.sum(m1, axis=0))  # shape[0] (행) 을 압축시키자. (2,2) -> (2,)
print_val(np.sum(m1, axis=1))  # shape[1] (열) 을 압축시키자. (2,2) -> (2,)
```

```
Type: <type 'numpy.int64'>
Shape: ()
값:
10

Type: <type 'numpy.ndarray'>
Shape: (2,)
값:
[4 6]

Type: <type 'numpy.ndarray'>
Shape: (2,)
값:
[3 7]
```

2. 벡터, 행렬 연산, 그래프 그리기

```
m1 = np.array([[1,2,3], [4,5,6]])
print m1
print m1.shape  # (2,3)
```

```
[[1 2 3]
 [4 5 6]]
```

```
print np.sum(m1)
print np.sum(m1, axis=0)  # shape[0] (행) 을 압축시키자. (2,3) ->
print np.sum(m1, axis=1)  # shape[1] (열) 을 압축시키자. (2,3) ->
```

```
21
```

2. 벡터, 행렬 연산, 그래프 그리기

zeros-like

```
m1 = np.array([[1,2,3],
               [4,5,6],
               [7,8,9],
               [10,11,12]])
m2 = np.zeros_like(m1)  # m1과 같은 형태의 0으로 이루어진 np array
print_val(m1)
print_val(m2)
```

```
Type: <type 'numpy.ndarray'>
Shape: (4, 3)
값:
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

Type: <type 'numpy.ndarray'>
Shape: (4, 3)
값:
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
```

2. 벡터, 행렬 연산, 그래프 그리기

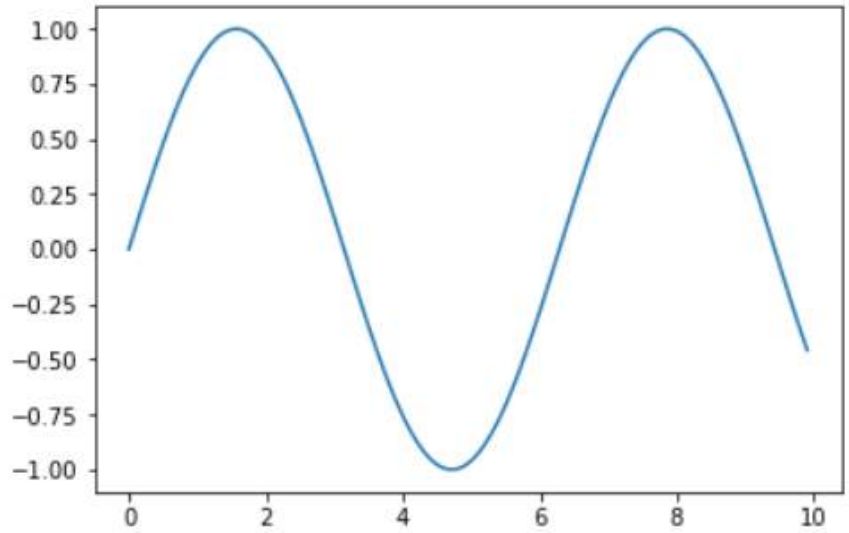
Matplot library

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# sin 커브
x = np.arange(0, 10, 0.1)  # 0~10 까지 0.1 간격의 숫자 배열
y = np.sin(x)

plt.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x7f202b9d7810>]
```



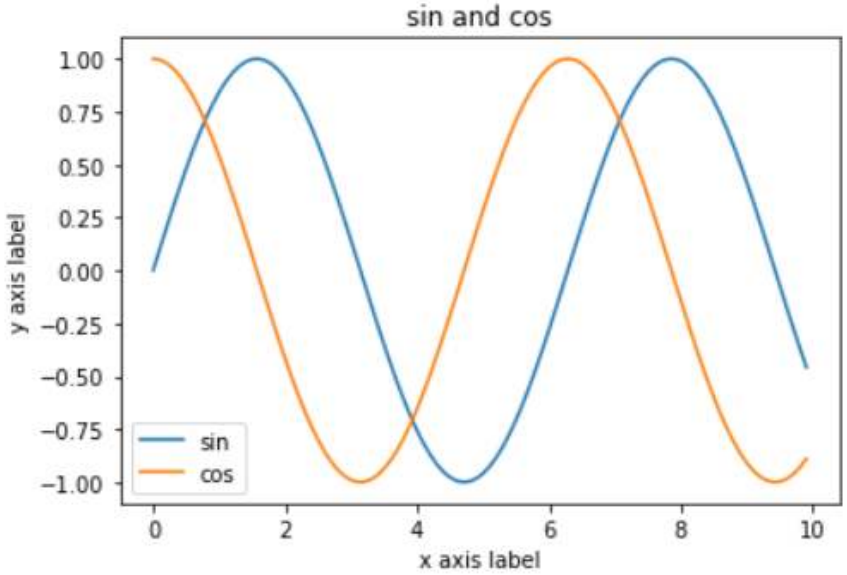
2. 벡터, 행렬 연산, 그래프 그리기

한 번에 두 개 그래프 그리기

```
y_sin = np.sin(x)
y_cos = np.cos(x)

plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('sin and cos')
plt.legend(['sin', 'cos'])

plt.show()
```



2. 벡터, 행렬 연산, 그래프 그리기

Subplot

```
plt.subplot(2, 1, 1)  # (2,1) 형태 플랏의 첫 번째 자리에 그리겠다
plt.plot(x, y_sin)
plt.title('sin')

plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('cos')

plt.show()
```