7. 넘파이

1) 넘파이(Numpy)란?

지금까지 데이터의 중요성에 대해서는 충분히 이해 하셨으리라 믿습니다. 인공 신경망을 설계하고, 직접 프로그래밍할 때 가장 많이 사용하는 데이터가 바로 배열입니다. 이러한 배열을 인공 신경망에 적합하게 생성하는 방법이 있습니다. 바로 넘파이(Numpy) 라이브러리를 사용하는 것입니다.



우리가 이번 장에서 사용할 넘파이는 배열이나 행렬의 계산에 필요한 함수들을 모아 놓은 파이썬 라이브러리 입니다

2) 넘파이 설치 및 설정하기

파이썬의 라이브러리는 자동으로 설치되지 않으므로 직접 설치하여야 합니다.

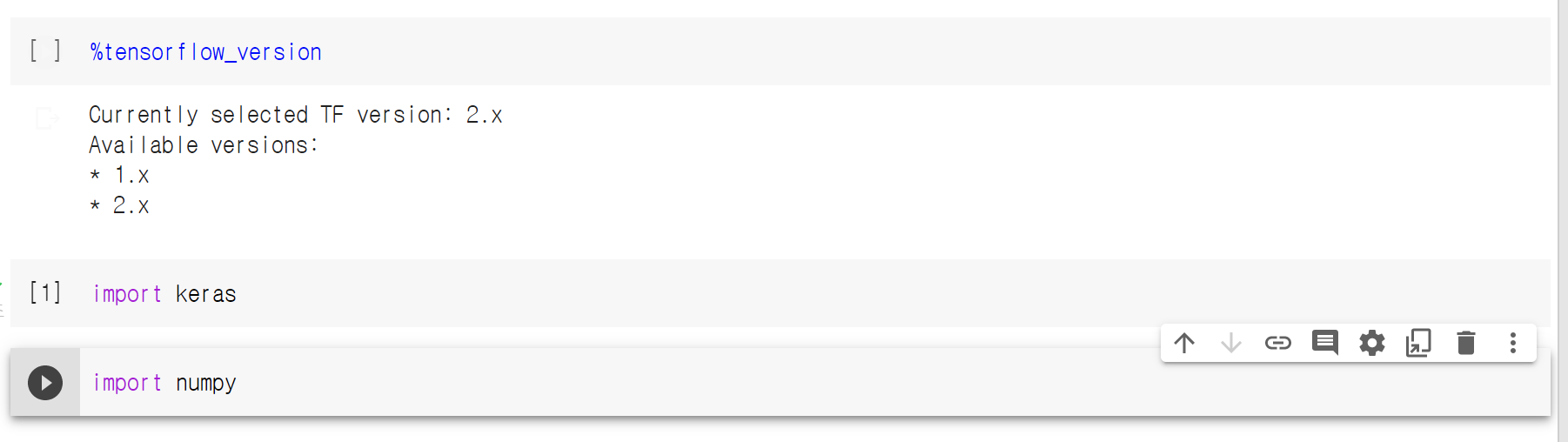
다만 우리가 사용하고 있는 코랩에서는 코랩을 지원하고 있기 때문에 코랩을 사용하고 계신다면 넘파이를 따로 설치하지 않으셔도 됩니다.

넘파이를 비롯한 파이썬 라이브러리를 설치하는 방법 중 하나는 pip 명령어를 사용하는 것입니다.

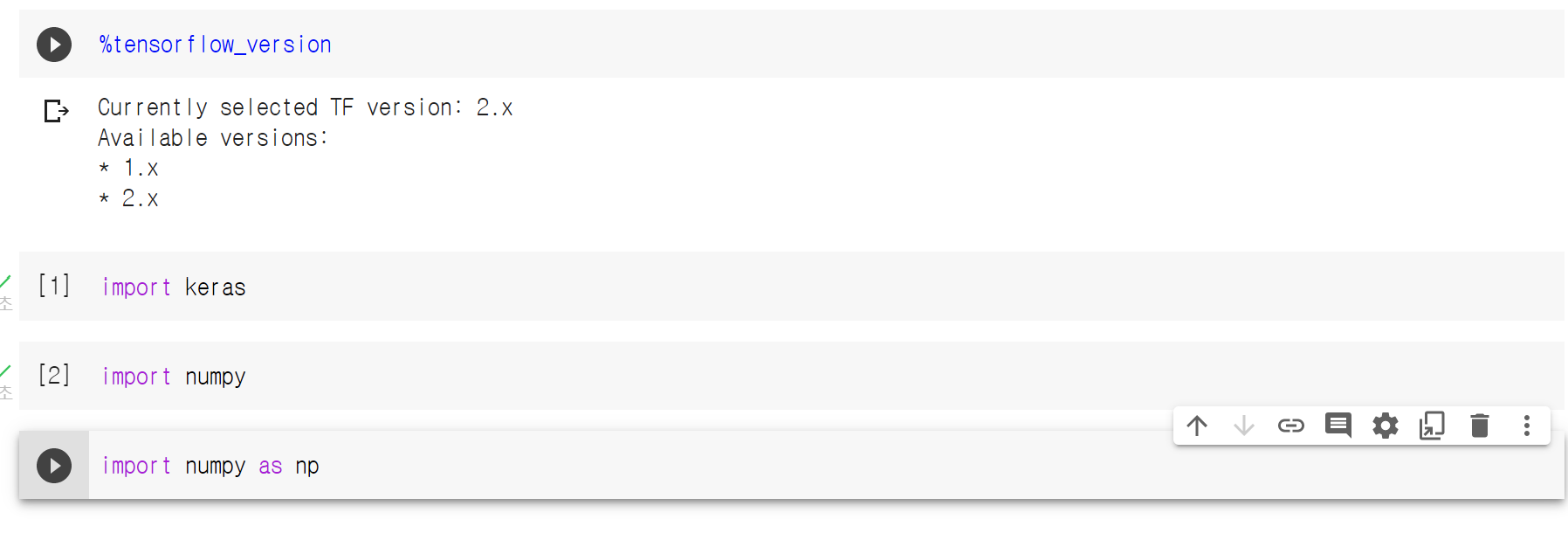
pip install numpy # pip 도구를 사용해서, numpy 를, install(설치)하겠다. 코랩을 사용하신다면 입력하지 않으셔도 됩니다!!

구글 코랩에는 기본적으로 넘파이가 설치되어 있지만 케라스와 마찬가지로 불러올 때는 import문을 사용해야 합니다

Import numpy



라이브러리를 불러왔으면 사용을 해야 합니다. 하지만 매번 numpy를 타이핑하기는 너무 귀찮을 수 있습니다. 그래서 우리는 numpy를 줄여서 np라고 부르기로 합시다.



앞으로 인공지능 코딩을 진행하며 다양한 라이브러리를 사용하게 될 겁니다. 이 라이브러리를 이용하기 위해서 import와 as는 자주 사용하게 될 것이니 기억해 두는 것이 좋을 것입니다.

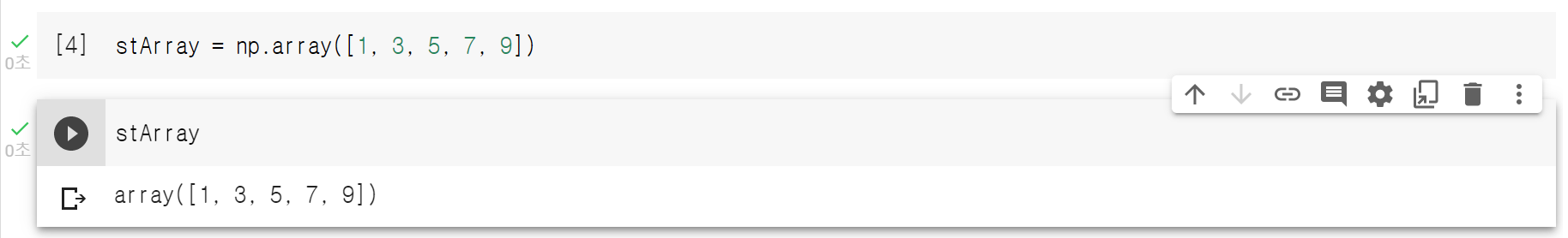
3) 넘파이로 배열 만들기

넘파이의 array() 함수를 사용하면 넘파이 배열을 만들 수 있습니다. 아까 넘파이를 np라고 부르기로 했으니 바로 사용해봅시다. 배열 이름은 우리 책의 이름(Standard of new Technology)을 따서 stArray로 합시다.

stArray = np.array([1, 3, 5, 7, 9]) # np(as로 설정해 준 numpy의 별칭)으로 1, 3, 5, 7, 9가 담긴 배열을 만들고 그 이름은 stArray로 하겠다.

* 여기서 np.array의 . 는 np(numpy)의 내부의 함수(기능)인 array를 불러오겠다는 뜻입니다. 앞으로 ~~.~~라는 형태를 얼마나 잘 이용하는지가 프로그래밍 혹은 딥러닝 제작 실력을 보는 척도 중 하나가 될 것입니다.

stArray # 잘 만들어졌나 확인하는 겁니다.

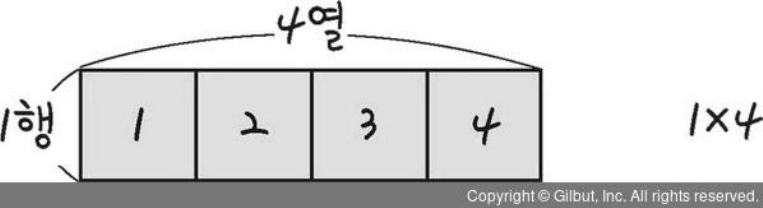


아래와 같이 나왔다면 잘 따라오셨습니다.

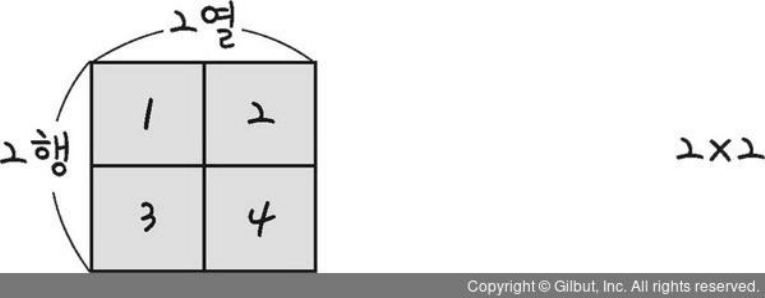
4) 다양한 배열

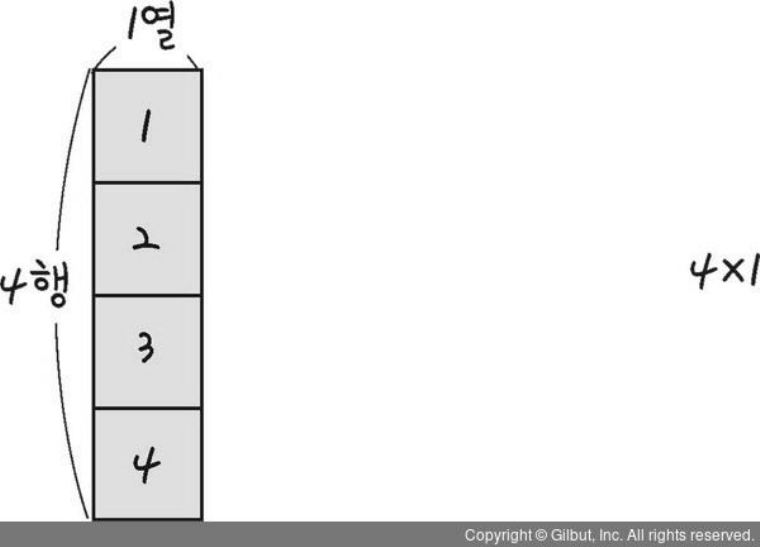
무작정 배열을 만들기만 한다고 제대로 사용할 수는 없겠죠. 배열에 대한 이해가 필요합니다. 데이터를 어떠한 형태로 배열에 넣는지가 인공 신경망을 설계할 때 중요하기 때문입니다

먼저 1, 2, 3, 4 3개의 원소로 이루어진 배열을 머리속에 떠올려봅시다



이런 형태인가요? 이것은 1행으로 이루어진 배열입니다. 외에도 다양한 행으로 이루어진 배열을 구성할 수도 있죠.



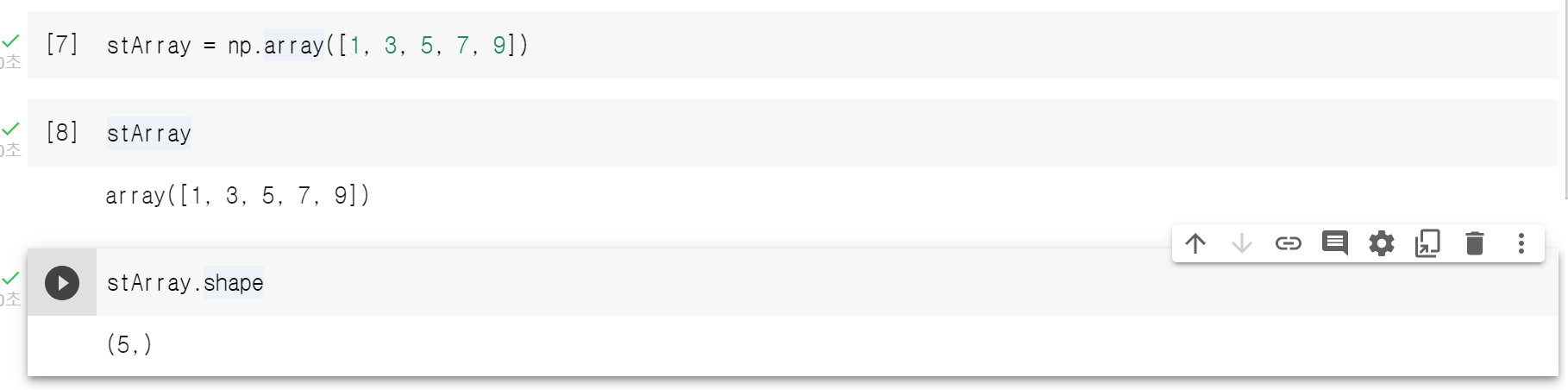


이렇게 말이죠.

우리가 인공 신경망을 설계하고 딥러닝 모델을 만들 때 어떤 배열의 형태로 입력할 것인지를 결정해야 합니다. 그리고 그렇게 하기 위해선 배열을 의도한 대로 구성하는 능력이 필요합니다.

위에서 배열의 ‘형태’라는 말을 사용했습니다. 우리가 만든 배열의 형태를 한번 볼까요? 우리가 만든 stArray의 형태를 보기 위해 stArray 안의 shape 함수를 사용합시다 (정확히는 np로 만든 a배열의 기능을 사용하는 것입니다)

stArray.shape



(5,)가 나온다면 가로로 5개의 줄(원소)이 있다는 뜻입니다.

5) 넘파이 2차원 배열 만들기

방금 만든 배열은 한줄로 이루어진 1차원 배열이었습니다. 이제 2차원 배열을 만들기 전에 2차원 배열의 모양에 대해 봅시다

1차원 배열은 우리가 아까 만든 stArray의 [1, 3, ,5, 7, 9]처럼 각 값이 1차원의 형태로, 즉 1줄로 구성된 것을 의미합니다



이렇게 말이죠

그럼 2차원 배열의 모양은 어떨까요?

이때 행렬이 필요하게 됩니다. 가로 줄은 ‘행(row)’, 세로 줄은 ‘열(column)’이라고 합니다.

배열을 다룰 때 아주 중요하니 반드시 알고 계셔야 합니다

2차원 배열을 다루는 법을 익히기 위해 위 사진과 같은 배열을 만들어봅시다.

stArray2 = np.array([[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]])

만들었으면 확인도 해야죠

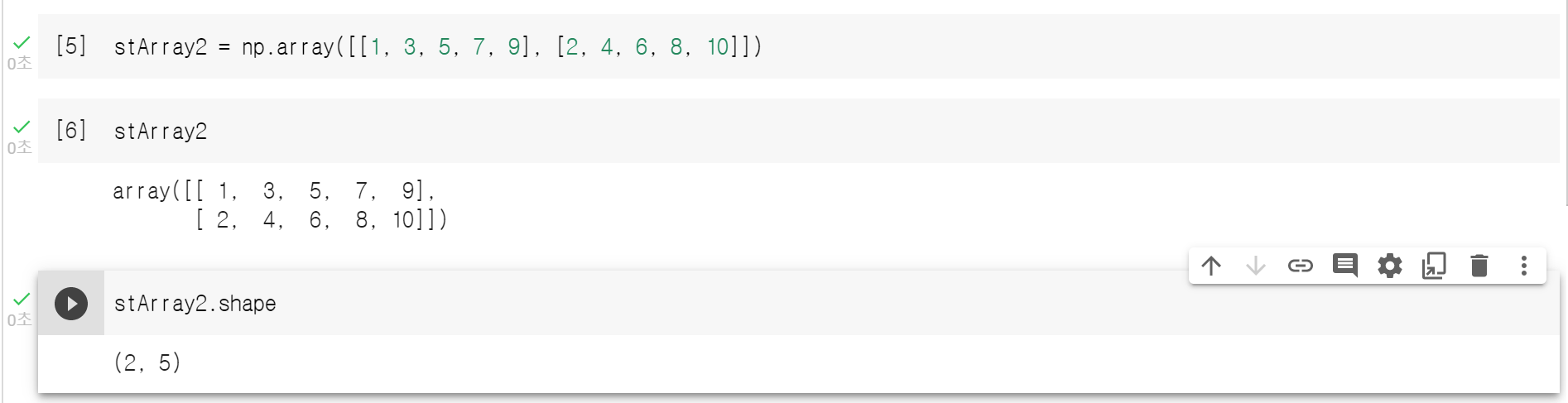
stArray2



친절하게도 행마다 줄을 구분해줘서 보기가 편합니다

1번 행에 1, 3, 5, 7, 9 원소가, 2번 행에 2, 4, 6, 8, 10 원소가 들어있는 모습을 보실 수 있습니다. 이제 shape 명령어를 통해 형태를 보겠습니다

stArray2.shape



2행(가로 2줄), 5열(세로 5줄)이라고 알려줍니다

6) 배열 형태 바꾸기

넘파이의 강력한 기능 중 하나는 배열의 형태를 바꿀 수 있다는 점입니다. 이때 사용하는 함수가 reshape(바꿀 형태)입니다. 인공지능 프로그래밍에서 상당히 많이 사용되는 기능이지요.

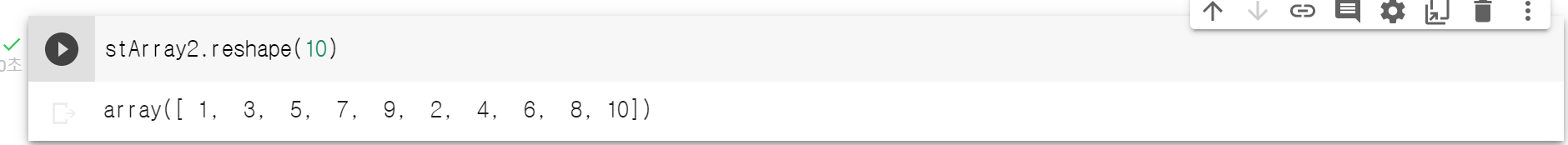
바로 사용해봅시다. 방금 만든 (2, 5) 형태의 stArray2를 (5, 2)형태로 바꿔봅시다

stArray2.reshape(5, 2)

테이블이(가) 표시된 사진

자동 생성된 설명

이번엔 각자 능력껏 stArray를 1차원 배열로 만들어봅시다



stArray2.reshape(10)

stArray2.reshape(10, ) 모두 가능합니다

7) 다양한 넘파이 함수 살펴보기

**zeros()** 함수는 모든 원소가 0인 행렬을 만들어 줍니다. 일단 사용해보며 감을 익힙시다

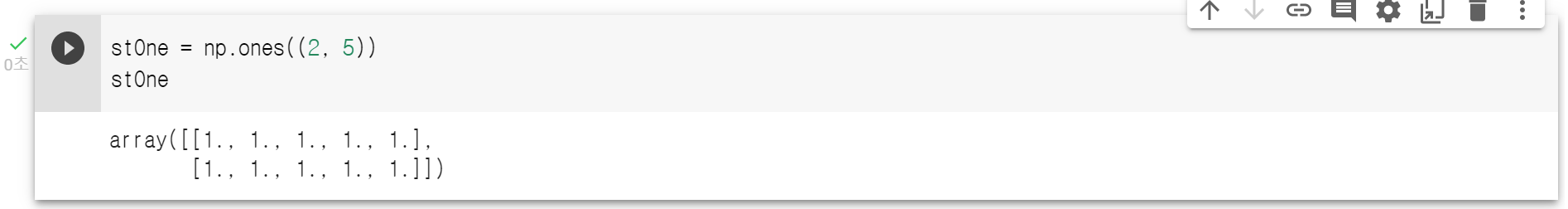
stZero = np.zeros((5, 2)) # 모든 원소가 0인 5행, 2열의 배열을 만들어 stZero라는 이름에 할당한다.

stZero # 제대로 만들어졌나 확인

테이블이(가) 표시된 사진

자동 생성된 설명

**ones()** 함수는 모든 원소가 0인 행렬을 만들어 줍니다. 이번엔 스스로 stOne이라는 이름의 배열에 2행, 5열의 1 배열을 만들어보세요



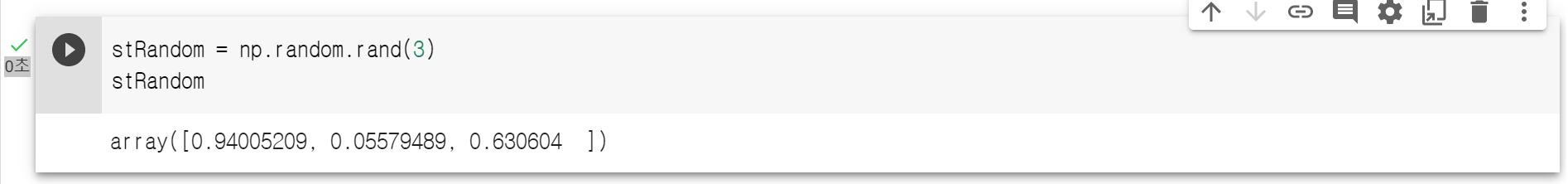
이렇게 생성과 확인을 한 코드 셀로 할 수도 있습니다. 출력하는 코드가 여러 줄이면 하나만 표시될 수 있으니 너무 남용하지는 마세요.

이제 random이라는 라이브러리의 함수들을 사용할 것입니다.

np(numpy) 라이브러리 안의 random 라이브러리 안의 함수이니 사용할 땐 np.random.~~~()가 되겠네요.

**rand()** 함수는 0부터 1 사이의 랜덤한 값(균일한 확률)을 생성해 배열로 만들어주는 함수입니다. 이렇게 만들어진 값은 한 곳에 몰리지 않았으며 다양한 용도로 사용됩니다.

stRandom = np.random.rand(3) # np 안의 random 안의 rand를 이용해 3개의 값을 생성한다.



아마 직접 실습하시게 되면 교재와 다른 숫자가 나올 것입니다. 임의의 랜덤한 수이기 때문에 매우 정상적인 일입니다

이렇게 만들어진 랜덤 값이 정말 균일한지 확인해보겠습니다.

import matplotlib.pyplot as plt #matplotlib 라이브러리를 불러와 별칭을 plt로 정한다.

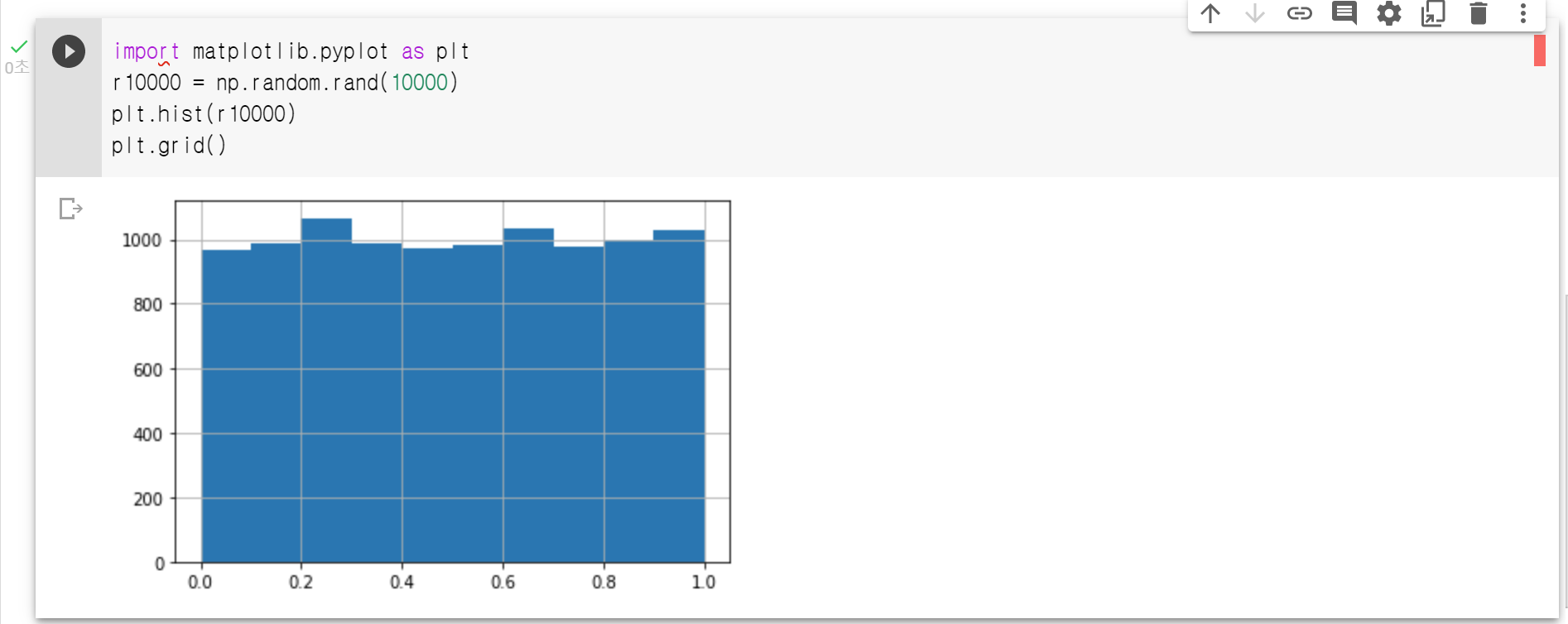
r10000 = np.random.rand(10000) # rand를 통하여 10000개의 랜덤한 값을 만들어

r10000에 저장한다.

plt.hist(r10000) # r10000을 plt의 hist를 사용해 히스토그램으로

표시한다.

plt.grid() # 히스토그램을 격자무늬 형태로 표시한다.



랜덤이기에 완벽하진 않지만 얼추 균등한 그래프가 나왔습니다.

* 히스토그램은 X좌표는 각 숫자 값(r10000에 들어있는 각 원소 하나하나의 값), Y좌표는 그 값을 가진 원소의 개수를 나타냅니다.

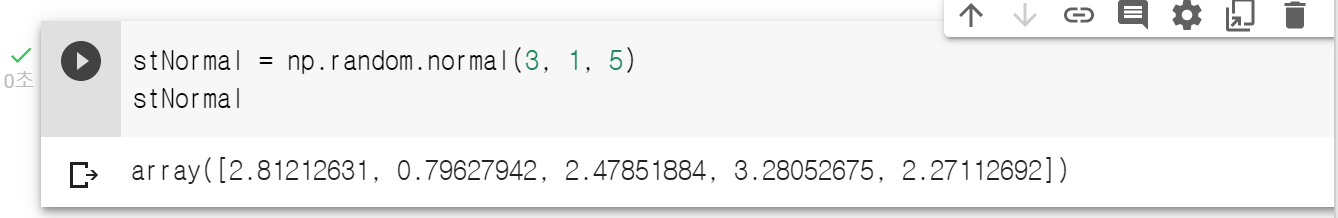
위 사용한 명령어들을 아직은 100% 이해하지 못하셔도 괜찮습니다. 지금은 aaa.bbb()를 이용하면 aaa에 있는 bbb라는 함수를 사용한다 정도만 이해하셔도 됩니다

**normal()** 함수는 정규 분포(가우시안 분포)로 값을 생성합니다. 이 때 평균과 표준편차를 정해 줄 수 있습니다. normal(평균, 표준편차, 생성할 개수)로 말이죠.

평균이 3, 표준편차가 1인 정규 분포 무작위 값 5개를 만들어보겠습니다

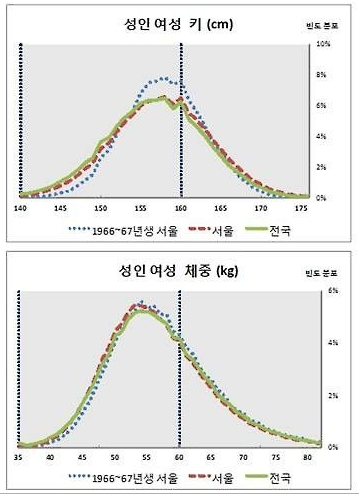
stNormal = np.random.normal(3, 1, 5)

stNormal



혹시 정규 분포가 뭔지 아시나요?

다양한 자연 현상을 볼 때 그 사례는 평균에 집중되고, 그 평균에서 멀어질수록 그 사례가 적어지는 경향을 보입니다. 이해하기 쉽게 실제 사례로 봅시다



어떤가요? 이제 좀 감이 오시나요? 이렇게 일반적인 분포를 정규 분포라고 합니다

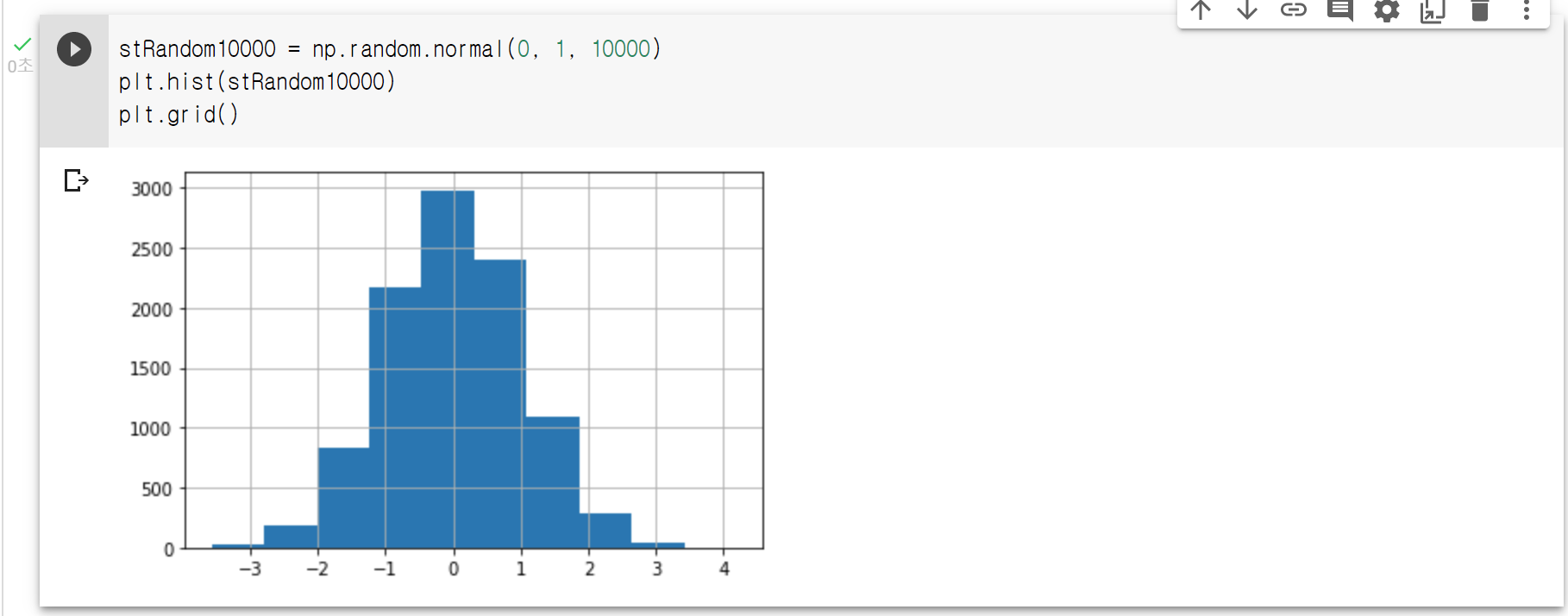
이 정규 분포는 통계학에서 아주 널리 쓰이며, 아주 많은 데이터를 다뤄야하는 딥러닝에서도 빠질 수 없는 존재입니다.

서론은 여기까지 하고 정규분포를 따르는 무작위 값을 만드는 normal()을 이용해 한번 만들어볼까요?

stRandom10000 = np.random.normal(0, 1, 10000) # 평균이 0, 표준편차는 1입니다.

plt.hist(stRandom10000)

plt.grid()

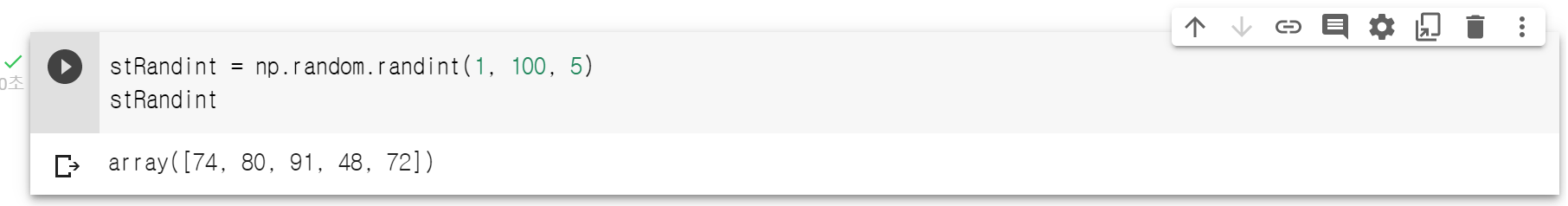


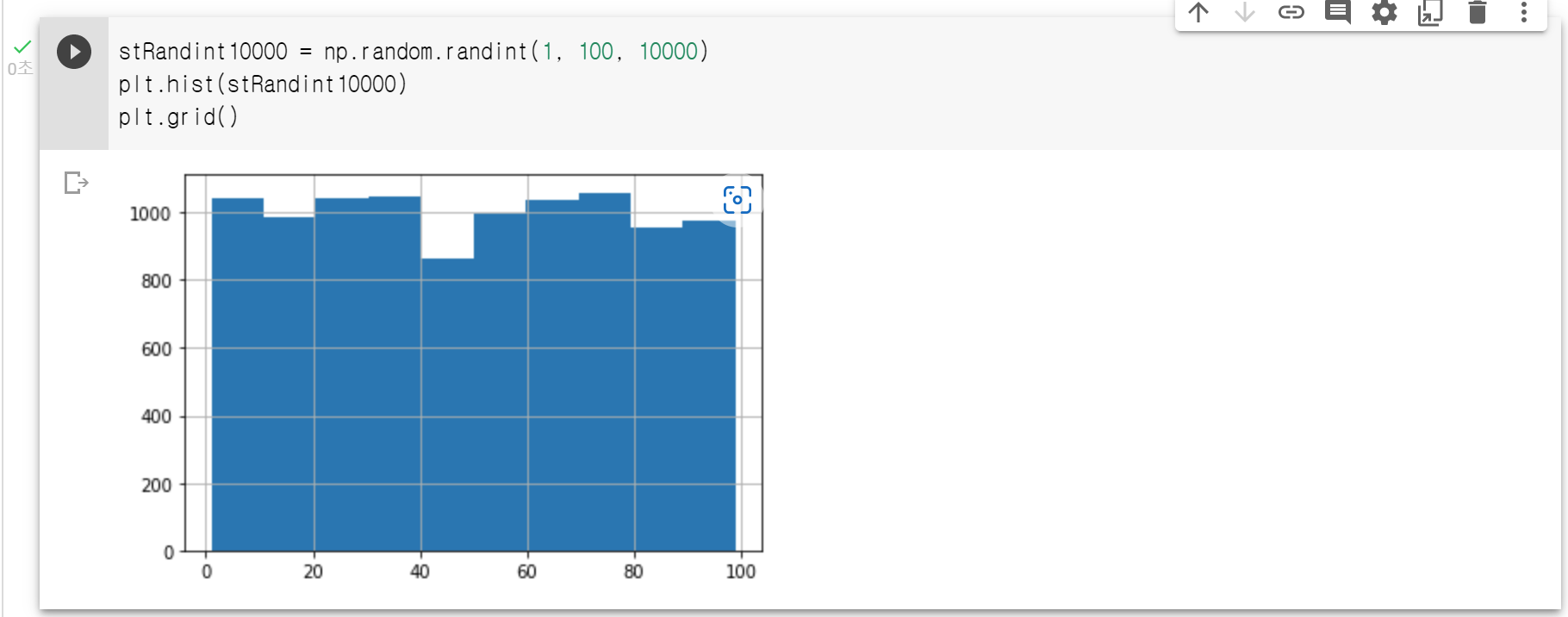
0에 근접한 값들이 잘 나와줬습니다.

**randint()** 함수는 특정한 값 사이에서 무작위 정수 값을 생성하는 함수입니다. 눈치가 빠르신 분은 ‘rand’om + ‘int’eger 라는 것을 눈치채셨을겁니다.

stRandint = np.random.randint(1, 100, 5) # 1과 100 사이의 값 5개를 만든다.

stRandint





randint는 이름의 rand에서 알 수 있듯이 각 값이 균일한 확률로 생성된다는 것을 알 수 있습니다

지금까지의 내용을 보면 컴퓨터는 아무런 규칙 없이 무작위의 수를 만드는 것 같지만 사실은 특정한 알고리즘에 의해서 만든 것입니다. 즉 규칙이 있다는 말이지요. 그 규칙에는 기준이 있고, 그 기준을 정하는 것이 바로 seed() 함수입니다.

같은 씨앗을 심으면 같은 식물이 자라나는 것처럼 생각하시면 됩니다.

먼저 비교를 위해 seed가 없는 무작위 값과, seed가 있는 무작위 값을 만들어 비교해봅시다

print(np.random.rand(5))

print(np.random.rand(5)) # 각각 다른 숫자가 출력될 것입니다.

테이블이(가) 표시된 사진

자동 생성된 설명

np.random.seed(12345)

print(np.random.rand(5))

np.random.seed(12345) # 참고로 seed는 일회용입니다. 다시 사용하려면

다시 seed를 입력해야 합니다

print(np.random.rand(5)) # 아까와 같은 값이 나오는 것을 볼 수 있습니다

텍스트이(가) 표시된 사진

자동 생성된 설명

이제 라이브러리에 대해서 감을 좀 잡으셨나요? 파이썬 프로그래밍을 다루는 것이 아닌, 딥러닝 개발을 목표로 하는 책이기에 더 자세한 내용을 다루기는 어렵지만 이정도만 이해하신다면 책을 따라오기에는 충분할 것입니다