

제2장 파이썬과 넘파이 보습

파이썬이란?

- 영어와 유사한 문법을 사용하기 때문에 파이썬으로 작성된 코드는 읽기 쉽다.
- 빠르게 코드를 작성하고 테스트할 수 있다.
- 성능 좋은 라이브러리

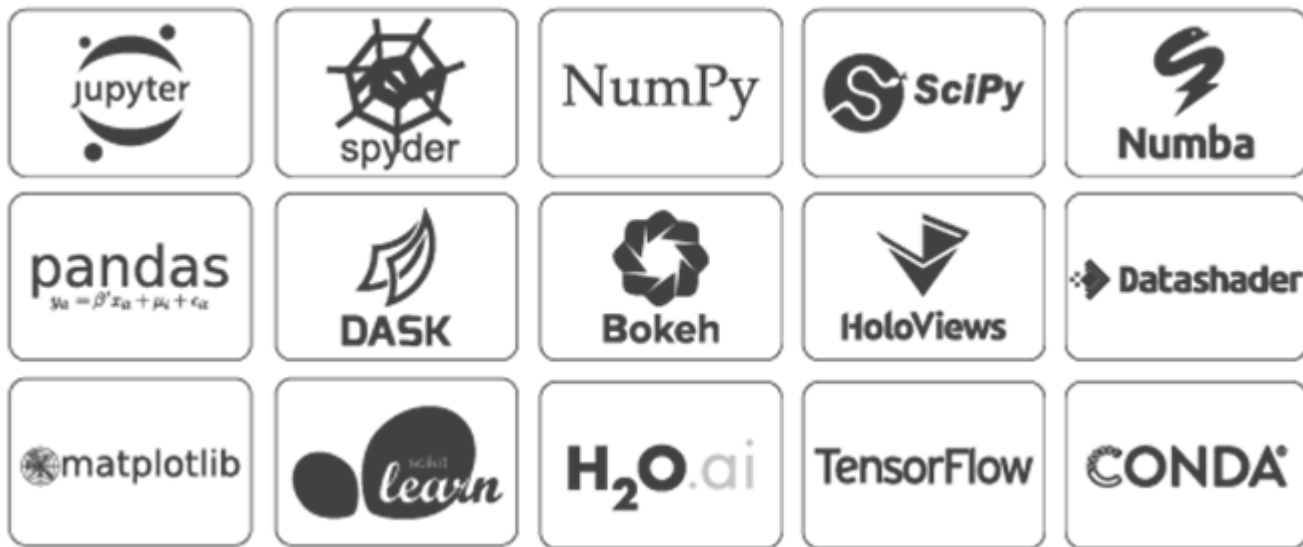


라이브러리들은 pip 명령어로
쉽게 설치됩니다.



그림 2-1 파이썬의 라이브러리

- 인기 있는 라이브러리가 거의 모두 포함되어 있다.





아나콘다 다운로드

Windows  MacOS  Linux 

Python 3.8

64-Bit Graphical Installer (457 MB)

32-Bit Graphical Installer (403 MB)

Python 3.8

64-Bit Graphical Installer (435 MB)

64-Bit Command Line Installer (428 MB)

Python 3.8

64-Bit (x86) Installer (529 MB)

64-Bit (Power8 and Power9) Installer (279 MB)

클릭한다.

ADDITIONAL INSTALLERS

The archive has older versions of Anaconda Individual Edition installers. The Miniconda installer homepage can be found [here](#).

- 현재 파이썬의 최신 버전은 3.9이지만 텐서플로우는 아직도 3.7 버전을 사용할 수 있다. 이렇게 되면 충돌이 생겨서 최신 버전의 파이썬에서는 텐서플로우는 실행되지 않는다



가상 환경 base



가상 환경 deep



가상환경 생성

```
(base) C:\Users\deep> conda create -n deep python=3.7
```

```
(base) C:\Users\deep> conda activate deep
```

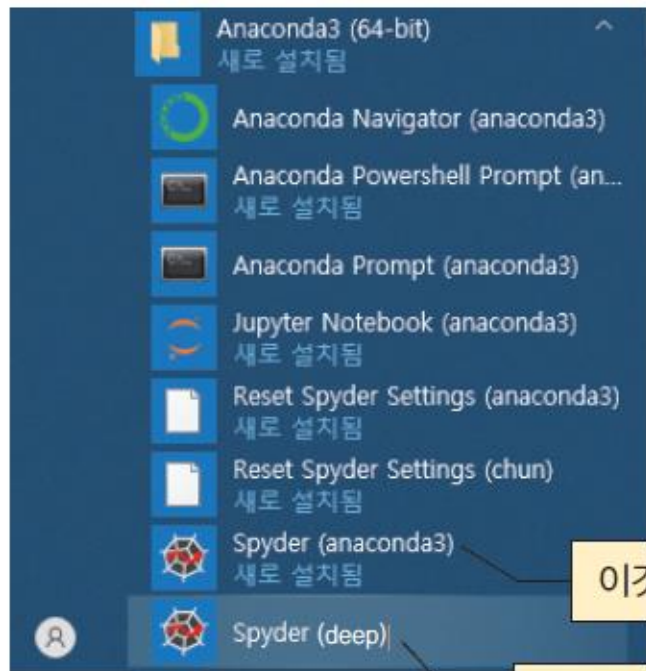
```
(deep) C:\Users\deep> conda install spyder
```

```
(deep) C:\Users\deep> conda install tensorflow
```

책
오
타
!



가상환경 생성



이것이 base 버전(기본)이다.

이것이 deep 버전이다.

지금부터 가상 환경
deep을 사용합니다.





파이썬의 개발 도구

특징	스파이더	주피터 노트북	구글의 코랩
설치 필요?	YES	YES	NO
인터페이스	GUI	웹 브라우저	웹 브라우저
인터넷 연결 필요?	NO	NO	YES
코드와 텍스트 통합	NO	YES	YES
데이터 파일 사용	로컬/리모트	로컬/리모트	구글 드라이브
CPU 사용	내 컴퓨터의 CPU	내 컴퓨터의 CPU	구글 클라우드 CPU



Home Page - Select or create a x +

localhost:8888/tree

jupyter

Quit Logout

Files Running Clusters

Select items to perform actions on them.

0 /

- 3D Objects
- AndroidStudioProjects
- Contacts
- Desktop
- Documents
- Downloads
- eclipse
- eclipse-workspace
- Favorites
- Links
- Music
- OneDrive
- Pictures
- Saved Games

localhost:8888/tree# Searches

Upload New ↕ ↺

Python 3

Other:

- Text File
- Folder
- Terminal

3일 전

21분 전

4달 전

4달 전

한 달 전

한 달 전

한 달 전

18일 전

한 달 전

한 달 전

한 달 전



주피터 노트북(소스 입력과 실행)

The screenshot shows the Jupyter Notebook interface in a web browser. The browser tab is titled 'Untitled3 - Jupyter Notebook'. The address bar shows 'localhost:8888/notebooks/Untitled3.ipynb?kernel_name=python3'. The Jupyter logo and 'Untitled3' are visible. Below the menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) is a toolbar with icons for adding, deleting, and running cells. The main area contains two cells. The first cell is a code cell with the text 'In [1]: print("Hello World!")' followed by the output 'Hello World!'. This cell is highlighted with a red rectangle. The second cell is a markdown cell with the text '# 자신의 이름을 출력하는 프로그램' followed by 'print() 함수를 이용하여 이름을 출력합니다.' This cell is also highlighted with a red rectangle. A yellow callout box points to the first cell with the text '코드를 입력한다. Shift+Enter 키를 누르면 코드가 실행된다.' Another yellow callout box points to the second cell with the text '설명을 입력한다. "Markdown"이라고 한다.' A third yellow callout box at the bottom right points to the entire interface with the text '주피터 노트북'.

코드를 입력한다.
Shift+Enter 키를 누르면
코드가 실행된다.

```
In [1]: print("Hello World!")  
Hello World!
```

자신의 이름을 출력하는 프로그램
print() 함수를 이용하여 이름을 출력합니다.

설명을 입력한다.
"Markdown"이라고 한다.

주피터 노트북



주피터 노트북(저장)

The screenshot shows the Jupyter Notebook web interface in a browser. The address bar indicates the URL is `localhost:8888/notebooks/Untitled3.ipynb?kernel_name=python3`. The notebook title is "Untitled3" and it shows the last checkpoint was 3 minutes ago (autosaved). The "File" menu is open, and the "Save and Checkpoint" option is highlighted with a red box and a circled number 1. The notebook content includes a code cell with `print("Hello World!")` and a text cell with the Korean text "자신의 이름을 출력하는 프로그램" (Program to output your name). Below the text cell, there is a code cell with `print("나의 이름은 철수입니다.")` and another text cell with the output "철수의 이름은 철수입니다." (Chulso's name is Chulso).

Home Page - Select or create x x Untitled3 - Jupyter Notebook x +

localhost:8888/notebooks/Untitled3.ipynb?kernel_name=python3

jupyter Untitled3 Last Checkpoint: 3분 전 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

New Notebook Open... Make a Copy... Save as... Rename... **Save and Checkpoint** Revert to Checkpoint Print Preview Download as Trusted Notebook Close and Halt

```
print("Hello World!")
```

Hello World!

① 자신의 이름을 출력하는 프로그램

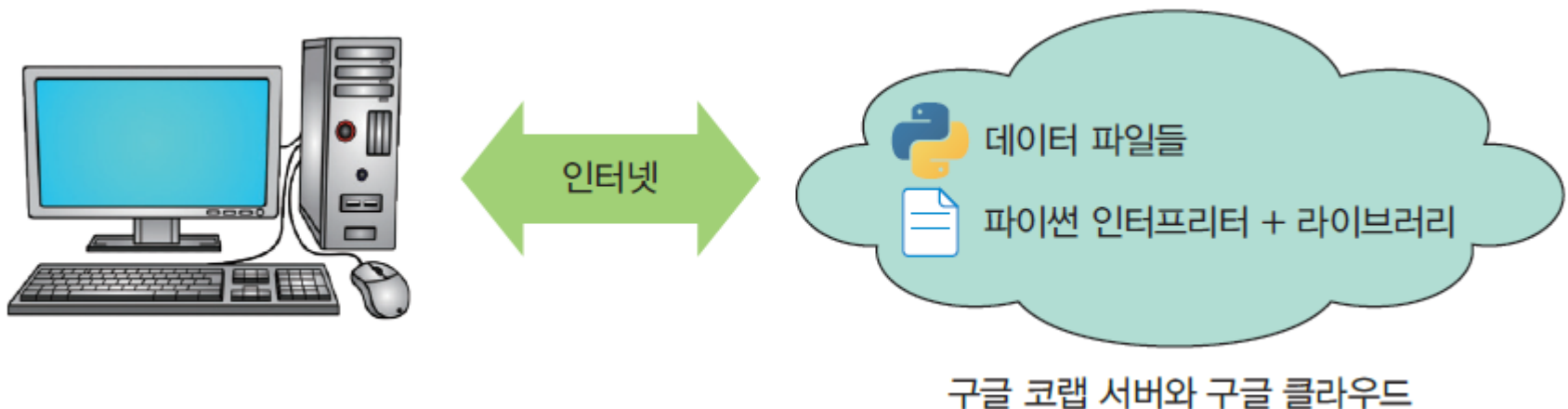
```
print() 함수를 이용하여 이름을 출력합니다.
```

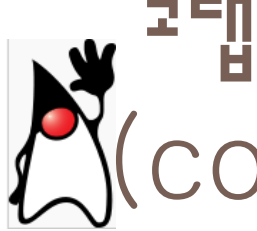
```
print("나의 이름은 철수입니다.")
```

철수의 이름은 철수입니다.

localhost:8888/notebooks/Untitled3.ipynb?kernel_name=python3#

- 구글 Colab은 주피터 노트북 개념을 클라우드로 확장한 것이다.
- 텍스트를 이미지, HTML, LaTeX 등과 함께 하나의 문서로 통합
- Colab 메모장을 만들면 Google 드라이브 계정에 저장



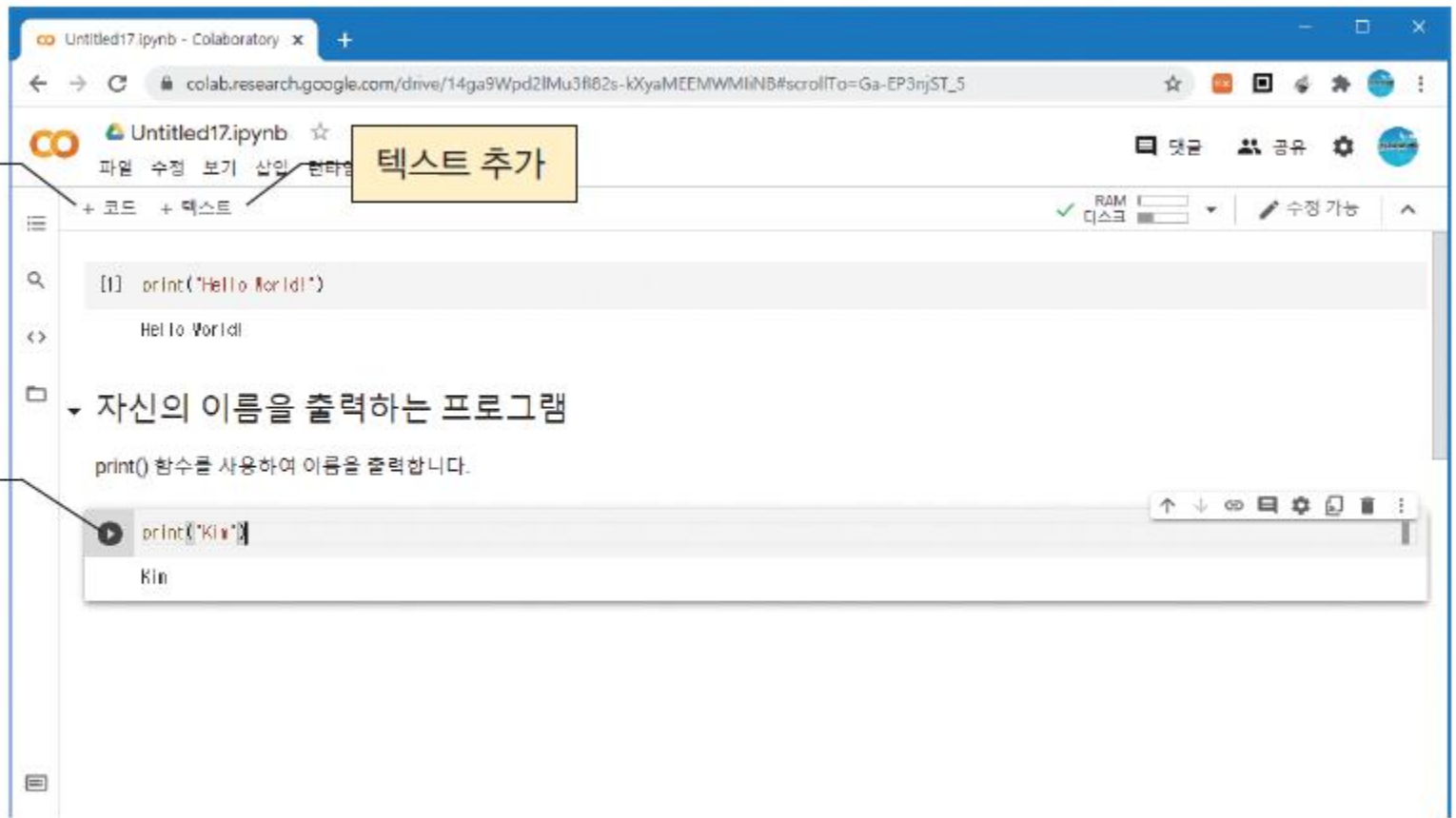


(colabcolab.research.google.com)

코드 추가

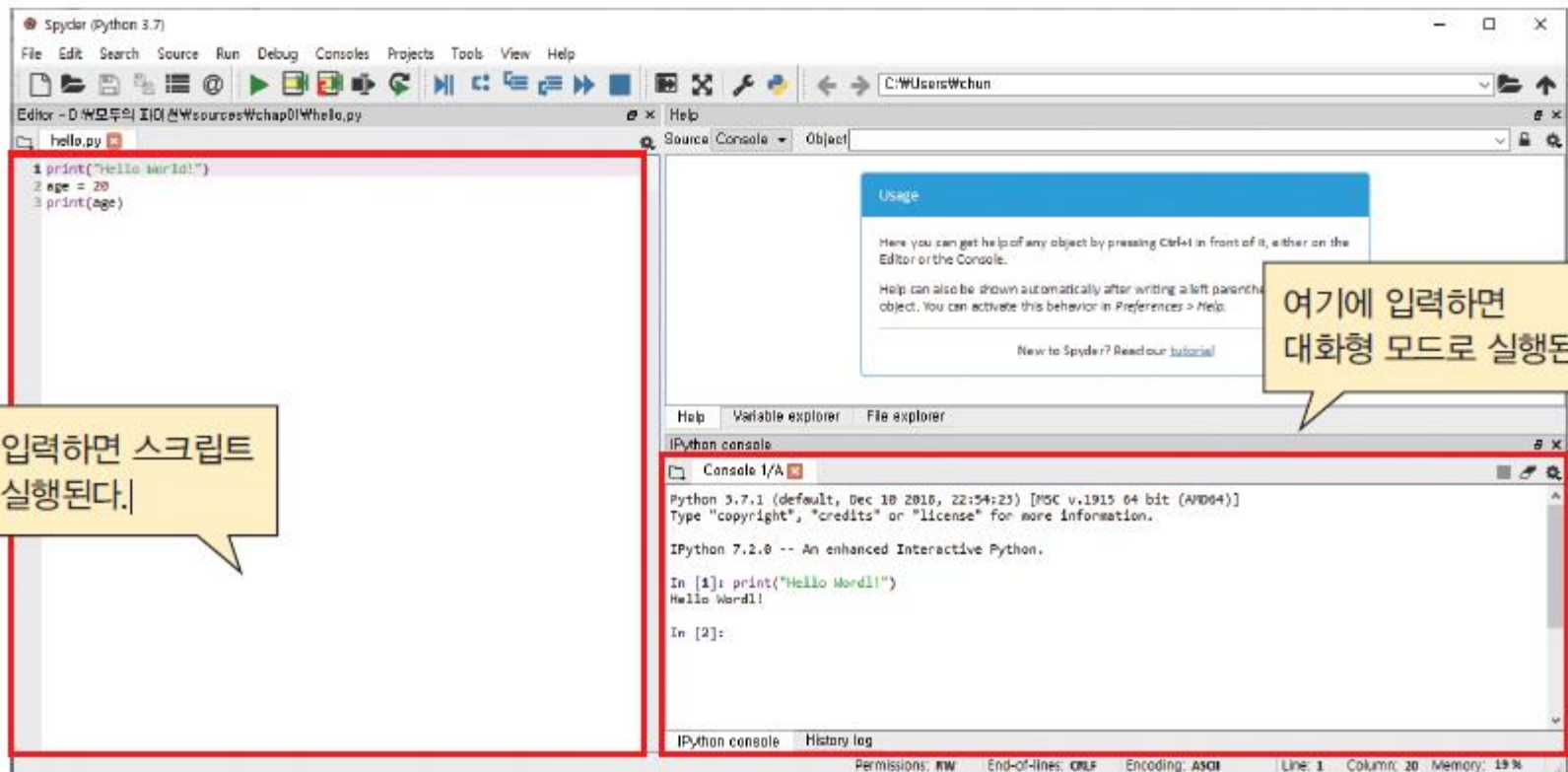
텍스트 추가

코드
실행 버튼





- 스파이더는 고급 편집 기능, 분석, 디버깅, 프로파일링 등의 기능이 있으며 변수 조사, 대화식 실행, 심층 검사, 시각화 기능을 갖춘 포괄적인 개발 도구





파이썬 보습(상술편)

```
>>> 3.14 * 10 * 10  
314.0
```

```
>>> 3.14 * 10**2  
314.0
```



파이썬 기본 (자료형)

```
>>> type(10)
Int
```

```
>>> type(3.14)
Float
```

```
>>> type("python")
str
```




파이썬 보습(변수)

```
>>> r = 20
>>> PI = 3.14          # 원주율 정의
>>> area = PI * r**2
>>> area
1256.0
```



파이썬 보습(리스트)

```
>>> lst = [ 10, 20, 30, 40, 50]
```

리스트 정의

```
>>> lst
```

리스트 출력

```
[10, 20, 30, 40, 50]
```

```
>>> lst[2]
```

리스트의 원소 접근

```
30
```

```
>>> lst[2] = 90
```

세 번째 원소를 90으로 변경

```
>>> lst
```

리스트 출력

```
[10, 20, 90, 40, 50]
```

```
>>> len(lst)
```

리스트의 길이 출력

```
5
```



파이썬 리스트 (슬라이싱)

```
>>> lst[0:3]  
[10, 20, 90]
```

인덱스 0부터 2까지를 추출한다.

```
>>> lst[2:]  
[90, 40, 50]
```

인덱스 2부터 끝까지를 추출한다.

```
>>> lst[:3]  
[10, 20, 90]
```

인덱스 0부터 2까지를 추출한다.

```
>>> lst[:-1]  
[10, 20, 90, 40]
```

처음부터 마지막 원소 앞까지 추출한다.



파이썬 보습(딕셔너리)

```
>>> car = { 'HP':200, 'make': "BNW" }    # 딕셔너리 정의
```

```
>>> car['HP']                            # 원소에 접근  
200
```

```
>>> car['color'] = "white"               # 새 원소 추가  
>>> car  
{'HP': 200, 'make': 'BNW', 'color': 'white'}
```



파이썬 복습(if-else 문)

```
>>> temp = -10
>>> if temp < 0 :
...     print("영하입니다.")          # 들여쓰기를 해야 한다.
... else
...     print("영상입니다.")
...
영하
```



파이썬 반복(for 문)

```
>>> for i in [1, 2, 3, 4, 5] :  
...     print(i, end=" ")  
...  
1 2 3 4 5
```



파이썬 보습(하루)

```
>>> def sayHello():  
...     print("Hello!")  
...  
>>> sayHello()  
Hello  
  
>>> def sayHello(name):  
...     print("Hello! "+name)  
...  
>>> sayHello("Kim")  
Hello! Kim
```



파이썬 보습(클래스)

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def sayHello(self):
        print("Hello 나의 이름은 " + self.name)

p1 = Person("John", 36)
p1.sayHello()
```

Hello 나의 이름은 John



딥러닝 개발에 사용되는 라이브러리

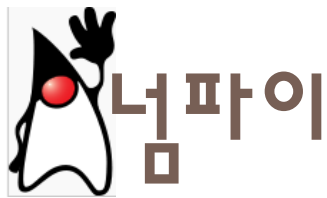
라이브러리 이름	웹사이트	설명
넘파이(Numpy)	https://numpy.org	효율적인 행렬 연산 라이브러리
맷플롯립(Matplotlib)	https://matplotlib.org	다양한 그래프를 그리는 라이브러리
사이킷런(Scikit-learn)	https://scikit-learn.org	전통적인 머신러닝 라이브러리
텐서플로우(TensorFlow)	https://tensorflow.org	딥러닝을 지원하는 라이브러리
케라스(Keras)	https://keras.io	고수준의 딥러닝 라이브러리
파이토치(PyTorch)	https://pytorch.org	페이스북에서 만든 딥러닝 라이브러리
판다스(Pandas)	https://pandas.pydata.org	데이터 처리를 위한 라이브러리



라이브러리 설치

- conda 또는 pip 사용

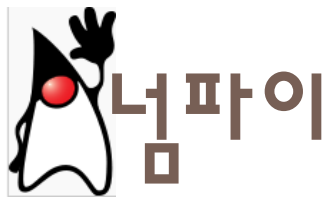
```
(base) C:\Users\kim> activate deep
(deep) C:\Users\kim> pip install gtts
Collecting gtts
Using cached gTTS-2.2.2-py3-none-any.whl (25 kB)
...
Successfully installed gtts-2.2.2
```



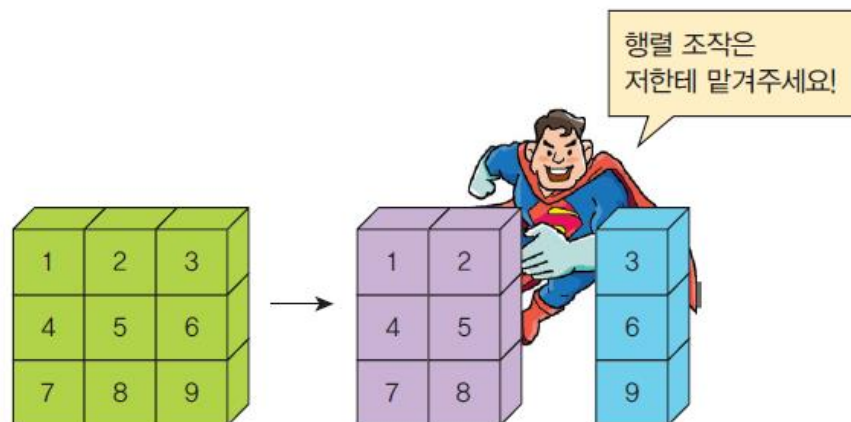
- 파이썬 리스트는 데이터가 비연속인 위치에 저장된다. 따라서 대용량 데이터를 처리할 때 상당히 불리하다.
- C언어 스타일의 배열은 데이터를 연속적인 위치에 저장해서 아주 효율적으로 데이터를 처리할 수 있다. -> 넘파이

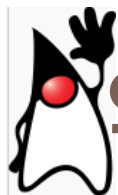


그림 2-3 파이썬 리스트와 넘파이 배열의 비교



- NumPy(Numerical Python)
- 과학 및 공학 분야에서 사용되는 오픈 소스 라이브러리
- Pandas, SciPy, Matplotlib, scikit-learn 등 과학 패키지에서 사용
- 다차원 배열 데이터 구조를 포함 - ndarray
- 다양한 수학적 행렬 연산을 수행하는 데 사용
- > pip install numpy





왜 딥러닝에서는 넘파이가 중요한가?

- 훈련 샘플은 2차원 행렬이나 3차원 행렬에 저장된다.

	특징 #1	특징 #2	특징 #3	특징 #4	...		
샘플 #1							
샘플 #2							
샘플 #3							
...							



왜 딥러닝에서는 넘파이가 중요한가?

- 훈련 샘플은 2차원 행렬이나 3차원 행렬에 저장된다.
- 훈련 샘플에 특징을 제외하거나 추가, 또는 훈련샘플 중에서 몇 개를 테스트 용으로 남겨두기도 -> 행렬 조작 -> 넘파이를 통해야

샘플

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
50	6.4	3.5	4.5	1.2	Versicolor
150	5.9	3.0	5.0	1.8	Virginica

특징

레이블



넘파이 포함하기

- 우리는 `numpy`를 `np`로 단축하여 표기한다. 코딩 시간을 절약할 수 있으며 코드를 표준화하기 위해서이다.

```
>>> import numpy as np
```



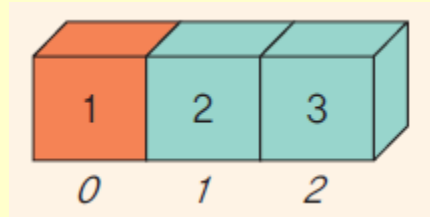
넘파이 배열 생성하기

```
>>> import numpy as np
```

```
>>> a = np.array([1, 2, 3])
```

```
>>> a  
array([1, 2, 3])
```

```
>>> a[0]  
1
```



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

배열
객체

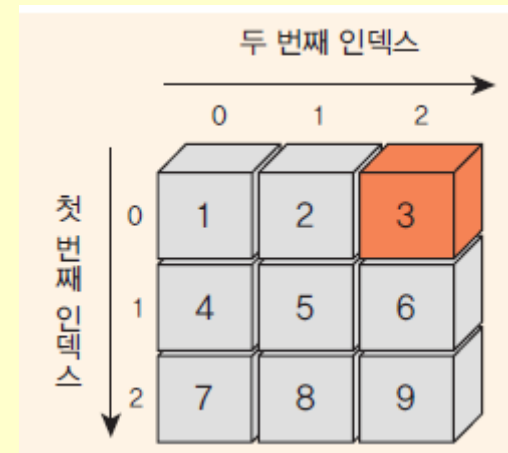
생성자 함수

파이썬 리스트

2차원 넘파이 배열

```
>>> b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> b
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

>>> b[0][2]
3
```





넘파이 배열의 속성

- 배열의 차원 및 항목 수는 형상(shape)에 의해 정의된다. 배열의 형상은 각 차원의 크기를 지정하는 정수의 튜플이다.
- 넘파이에서는 차원을 축(axes)이라고 한다

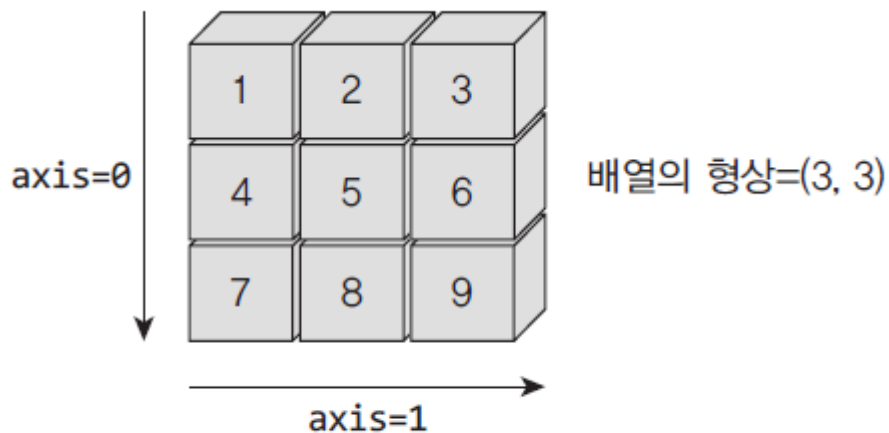
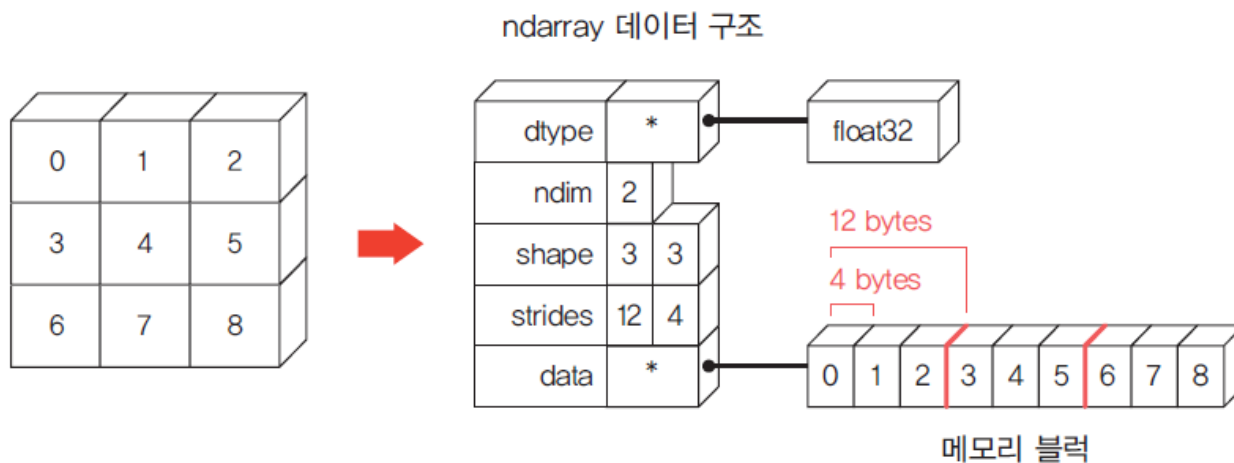


그림 2-4 넘파이 배열의 형상



넘파이 배열의 속성

속성	설명
ndim	축의 개수. 2차원 배열이면 ndim은 2이다.
shape	배열의 형상. 형상은 정수 튜플로 나타낸다. 예를 들어서 n개의 행과 m개의 열이 있는 행렬의 경우, shape는 (n, m)이다.
size	배열 안에 있는 요소들의 총 개수
dtype	배열 요소의 자료형, numpy.int32, numpy.int16, numpy.float64가 그 예이다.
itemsize	배열을 이루는 요소의 크기로서 단위는 바이트이다. 예를 들어, float64은 itemsize가 8이다.
data	실제 데이터가 저장되는 메모리 블록의 주소





넘파이 배열의 속성

```
>>> import numpy as np
>>> a = np.array([[ 0, 1, 2],
                  [ 3, 4, 5],
                  [ 6, 7, 8]])
>>> a.shape           # 배열의 형상
(3, 3)
>>> a.ndim            # 배열의 차원 개수
2
>>> a.dtype           # 요소의 자료형
dtype('int32')
>>> a.itemsize        # 요소 한개의 크기
4                      # 오타
>>> a.size            # 전체 요소의 개수
9
```



스파이더에서의 넘파이 배열 디버깅

Variable explorer

Download Save Print Copy

Name	Type	Size	
a	int32	(3, 3)	<pre>[[0 1 2] [3 4 5]]</pre>

a - NumPy array

	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8

Variable explorer File explorer Help



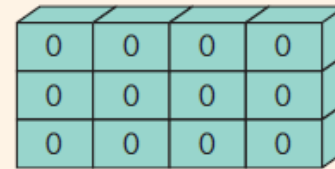
zeros(), eye(), ones() 함수

```
>>> np.zeros( (3, 4) )  
array([[ 0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.]])
```

```
>>> np.ones((3, 4))  
array([[1, 1, 1, 1],  
       [1, 1, 1, 1],  
       [1, 1, 1, 1]])
```

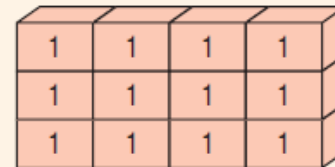
```
>>> np.eye(3)  
array([[1., 0., 0.],  
       [0., 1., 0.],  
       [0., 0., 1.]])
```

(3, 4)는 배열의 형상(행의 개수, 열의 개수)



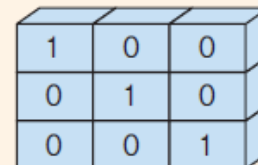
0	0	0	0
0	0	0	0
0	0	0	0

np.zeros((3, 4))



1	1	1	1
1	1	1	1
1	1	1	1

np.ones((3, 4))



1	0	0
0	1	0
0	0	1

np.eye(3)



arange() 함수

```
>>> np.arange(5)  
array([0, 1, 2, 3, 4])
```

시작값을 지정하려면 다음과 같이 한다.

```
>>> np.arange(1, 6)  
array([1, 2, 3, 4, 5])
```

증가되는 간격을 지정하려면 다음과 같이 한다.

```
>>> np.arange(1, 10, 2)  
array([1, 3, 5, 7, 9])
```

np.arange(start, stop, step)

시작값

종료값

간격



linspace() 함수

```
>>> np.linspace(0, 10, 100)
array([ 0.         ,  0.1010101 ,  0.2020202 ,  0.3030303 ,  0.4040404 ,
        ...
        8.08080808,  8.18181818,  8.28282828,  8.38383838,  8.48484848,
        8.58585859,  8.68686869,  8.78787879,  8.88888889,  8.98989899,
        9.09090909,  9.19191919,  9.29292929,  9.39393939,  9.49494949,
        9.5959596 ,  9.6969697 ,  9.7979798 ,  9.8989899 , 10.         ])
```

np.linspace(start, stop, num)

시작값

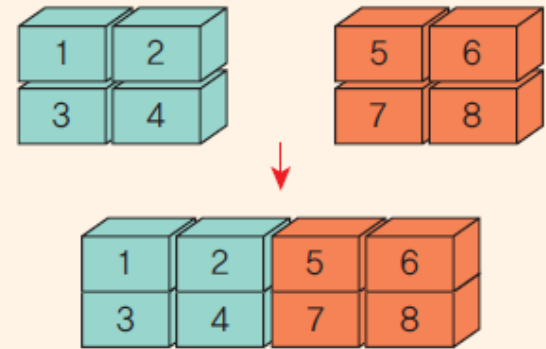
종료값

개수

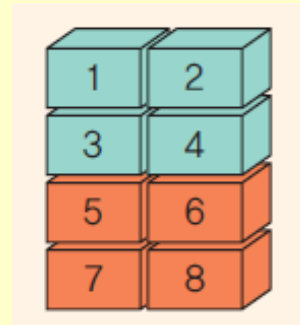


2개의 배열 합치기

```
>>> x = np.array([[1, 2], [3, 4]])  
>>> y = np.array([[5, 6], [7, 8]])  
  
>>> np.concatenate((x, y), axis=1)  
array([[1, 2, 5, 6],  
       [3, 4, 7, 8]])
```



```
>>> np.vstack((x, y))  
array([[1, 2],  
       [3, 4],  
       [5, 6],  
       [7, 8]])
```





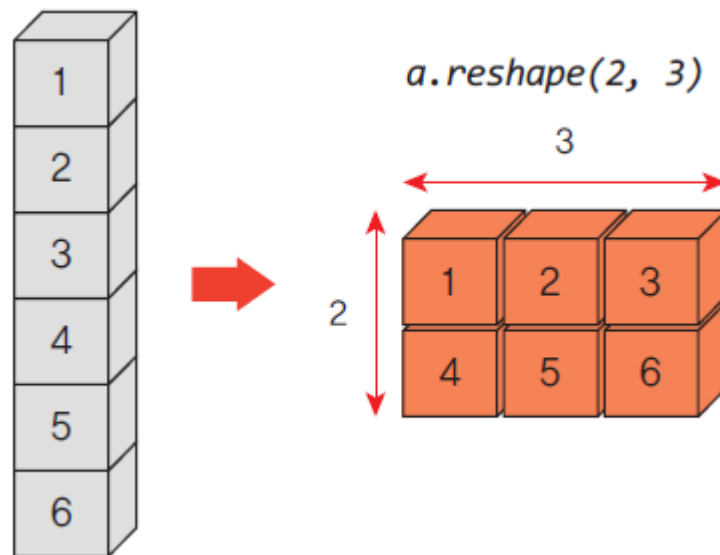
reshape() 함수

```
new_array = old_array.reshape((2, 3))
```

새로운 배열

원래의 배열

새로운 배열의 형상





reshape() 함수

```
>>> a = np.arange(12)
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

a에 대하여 reshape(3, 4)를 호출하면 1차원 배열이 2차원 배열로 바뀌게 된다.

```
>>> a.reshape(3, 4)
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
>>> a.reshape(6, -1)
array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
```



배열 분할하기

```
>>> array = np.arange(30).reshape(-1, 10)
>>> array
Array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]])
```

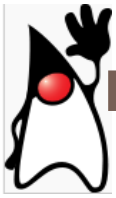
```
>>> arr1, arr2 = np.split(array, [3], axis=1)
```

```
>>> arr1
```

```
array([[ 0,  1,  2],
       [10, 11, 12],
       [20, 21, 22]])
```

```
>>> arr2
```

```
array([[ 3,  4,  5,  6,  7,  8,  9],
       [13, 14, 15, 16, 17, 18, 19],
       [23, 24, 25, 26, 27, 28, 29]])
```



배열에 새로운 축 추가하기

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

```
>>> a.shape
```

```
(6,)
```

```
>>> a1 = a[np.newaxis, :]
```

```
>>> a1
```

```
array([[1, 2, 3, 4, 5, 6]])
```

```
>>> a1.shape
```

```
(1, 6)
```

```
>>> a2 = a[:, np.newaxis]
```

```
array([[1],
```

```
       [2],
```

```
       [3],
```

```
       [4],
```

```
       [5],
```

```
       [6]])
```

```
>>> a2.shape
```

```
(6, 1)
```

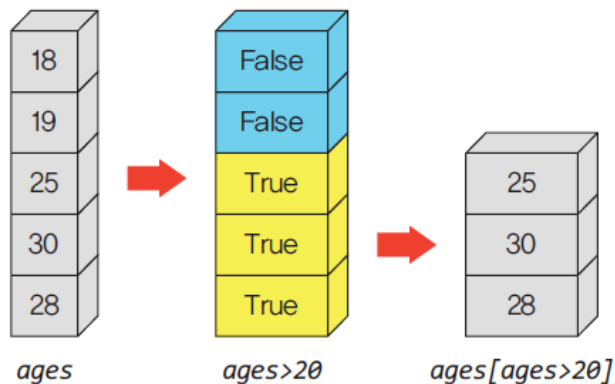


인덱싱과 슬라이싱

```
>>> ages = np.array([18, 19, 25, 30, 28])
>>> ages[1:3] # 인덱스 1에서 인덱스 2까지
array([19, 25])
>>> ages[:2] # 인덱스 0에서 인덱스 1까지
array([18, 19])
```

논리적인 인덱싱 (logical indexing)

```
>>> y = ages > 20
>>> y
array([False, False,  True,  True,  True])
>>> ages[ages > 20]
array([25, 30, 28])
```

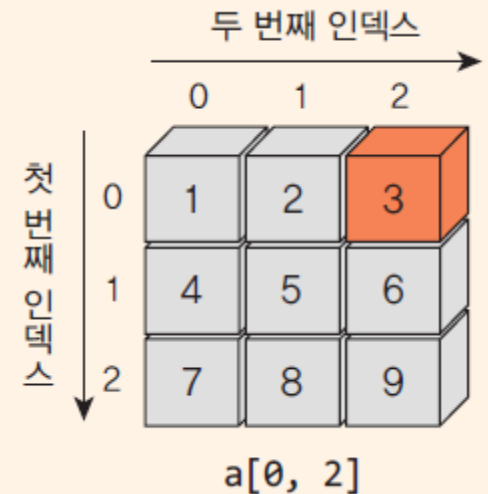


조건을 주어서 배열 중에서 원하는 요소들을 선택할 수 있습니다.



2차원 배열의 인덱싱

```
>>> a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
>>> a[0, 2]  
3  
>>> a[0, 0] = 12  
>>> a  
array([[12, 2, 3],  
       [ 4, 5, 6],  
       [ 7, 8, 9]])
```





2차원 배열의 슬라이싱

```
>>> a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
>>> a[0:2, 1:3]  
array([[2, 3],  
       [5, 6]])
```



a[0:2, 1:3]



2차원 배열의 슬라이싱

data

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[0]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[1,:]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[:,2]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[0:2,0:2]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[0:2,2:4]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

data[:,2,2]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)

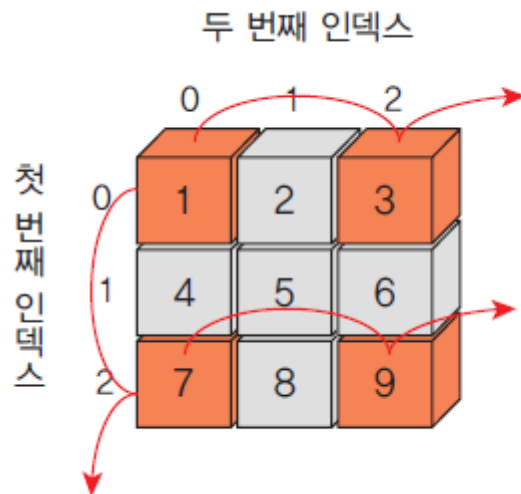
data[1::2,1::2]

(0, 0)	(0, 1)	(0, 2)	(0, 3)
(1, 0)	(1, 1)	(1, 2)	(1, 3)
(2, 0)	(2, 1)	(2, 2)	(2, 3)
(3, 0)	(3, 1)	(3, 2)	(3, 3)



2차원 배열의 슬라이싱

`a[::2, ::2]`

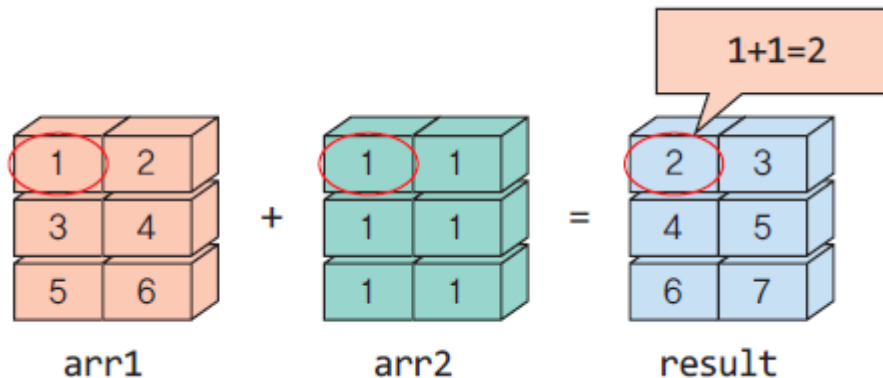


`data[::2, ::2]`

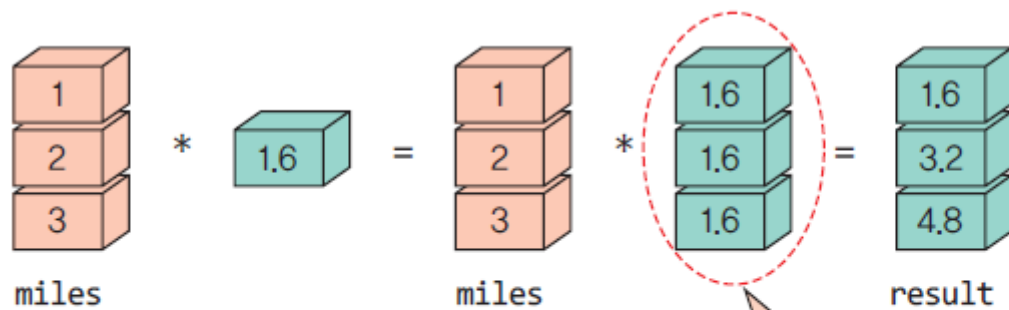


배열과 배열의 연산

```
>>> arr1 = np.array([[1, 2], [3, 4], [5, 6]])
>>> arr2 = np.array([[1, 1], [1, 1], [1, 1]])
>>> result = arr1 + arr2 # 넘파이 배열에 + 연산이 적용된다.
>>> result
array([[2, 3],
       [4, 5],
       [6, 7]])
```



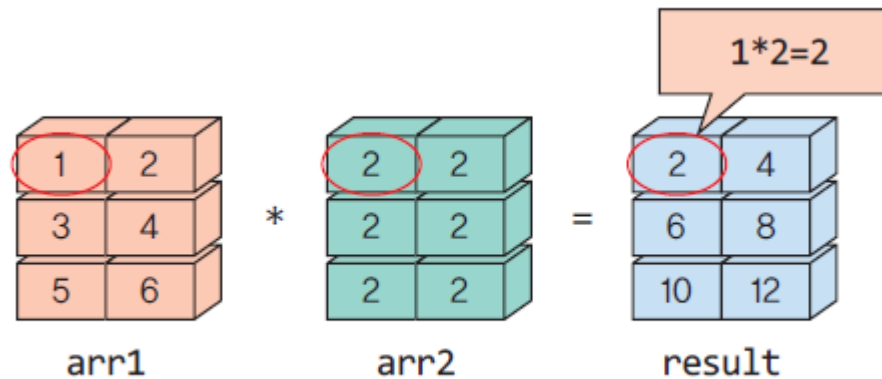
```
>>> miles = np.array([1, 2, 3])  
>>> result = miles * 1.6  
>>> result  
array([1.6, 3.2, 4.8])
```



넘파이는 산술 연산이 가능하도록 행렬의 차원을 맞춘다.

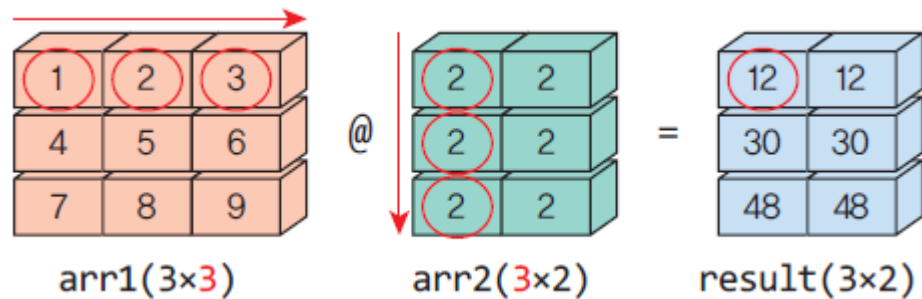


```
>>> arr1 = np.array([[1, 2], [3, 4], [5, 6]])
>>> arr2 = np.array([[2, 2], [2, 2], [2, 2]])
>>> result = arr1 * arr2
>>> result
array([[ 2,  4],
       [ 6,  8],
       [10, 12]])
```





```
>>> arr1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> arr2 = np.array([[2, 2], [2, 2], [2, 2]])
>>> result = arr1 @ arr2 # arr1.dot(arr2)로 하여도 된다.
array([[12, 12],
       [30, 30],
       [48, 48]])
```





넘파이 배열에 함수를 적용하려면?

```
>>> A = np.array([0, 1, 2, 3])  
>>> 10 * np.sin(A)  
array([0.          , 8.41470985, 9.09297427, 1.41120008])
```

넘파이의 `sin()` 함수를 적용하려면 배열의 요소에 모두 `sin()` 함수가 적용된다.



넘파이 배열 메소드

```
>>> a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
>>> a.sum()  
45  
>>> a.min()  
1  
>>> a.max()  
9
```

1	2	3
4	5	6
7	8	9

.max() = 9

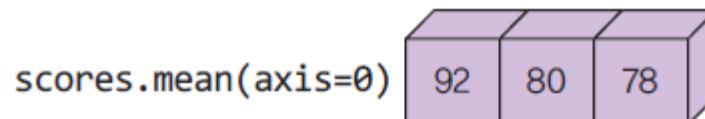
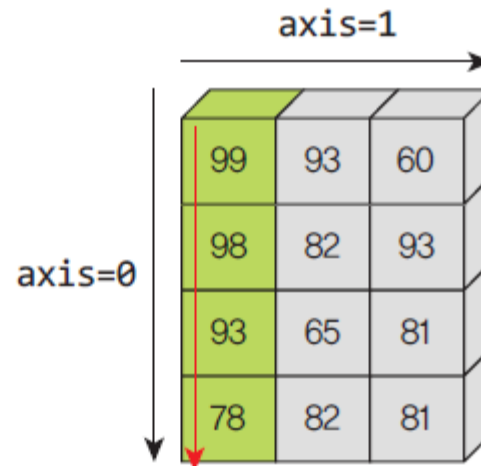
1	2	3
4	5	6
7	8	9

.min() = 1



계산을 특정 한 행이나 열만을 가지고도 할 수 있다

```
>>> scores = np.array([[99, 93, 60], [98, 82, 93],  
...:                   [93, 65, 81], [78, 82, 81]])  
>>> scores.mean(axis=0)  
array([92. , 80.5 , 78.75])
```





이 분포에서 난수 생성하기

`a = np.random.rand(5, 3)`

새로운 배열

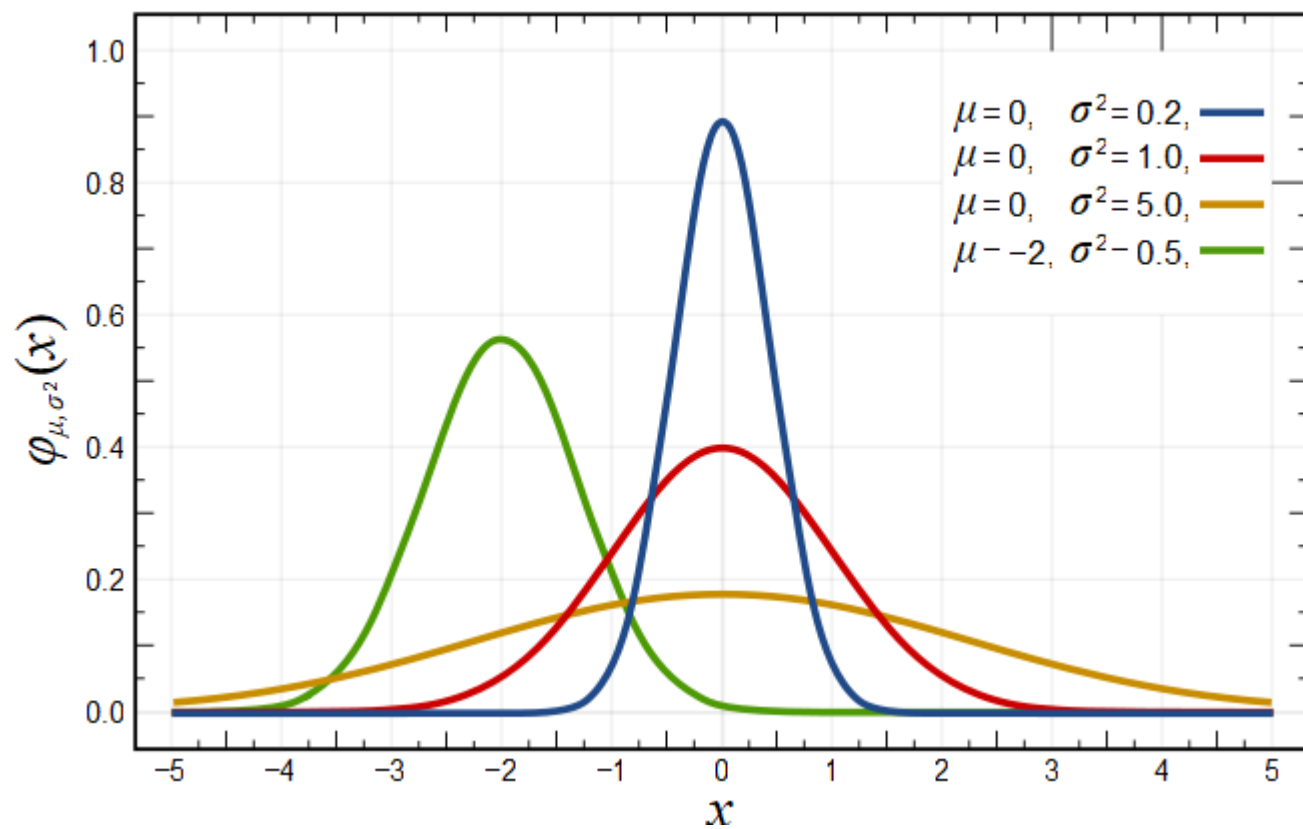
행의 개수 열의 개수

```
>>> np.random.seed(100)
>>> np.random.rand(5)
array([0.54340494, 0.27836939, 0.42451759, 0.84477613, 0.00471886])

>>> np.random.rand(5, 3)
array([[0.12156912, 0.67074908, 0.82585276],
       [0.13670659, 0.57509333, 0.89132195],
       [0.20920212, 0.18532822, 0.10837689],
       [0.21969749, 0.97862378, 0.81168315],
       [0.17194101, 0.81622475, 0.27407375]])
```



정규 분포에서 난수 생성하기





정규분포에서 난수 생성하기

```
>>> np.random.randn(5)
array([ 0.78148842, -0.65438103,  0.04117247, -0.20191691, -0.87081315])
```

```
>>> np.random.randn(5, 4)
array([[ 0.22893207, -0.40803994, -0.10392514,  1.56717879],
       [ 0.49702472,  1.15587233,  1.83861168,  1.53572662],
       [ 0.25499773, -0.84415725, -0.98294346, -0.30609783],
       [ 0.83850061, -1.69084816,  1.15117366, -1.02933685],
       [-0.51099219, -2.36027053,  0.10359513,  1.73881773]])
```

```
>>> m, sigma = 10, 2
>>> m + sigma*np.random.randn(5)
array([ 8.56778091, 10.84543531,  9.77559704,  9.09052469,  9.48651379])
```



정규분포에서 난수 생성하기

```
>>> mu, sigma = 0, 0.1      # 평균과 표준 편차
>>> np.random.normal(mu, sigma, 5)
array([ 0.15040638,  0.06857496, -0.01460342, -0.01868375, -0.1467971 ])
```

```
a = np.random.normal(loc=0.0, scale=1.0, size=None)
```

평균

표준편차

배열의 차원



고유 항목 및 개수를 얻는 방법

```
>>> a = np.array([11, 11, 12, 13, 14, 15, 16, 17, 12, 13, 11, 14, 18, 19, 20])
>>> unique_values = np.unique(a)
>>> unique_values
array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

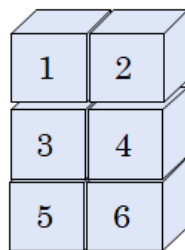


전치행렬 계산하기

```
import numpy as np
```

```
arr = np.array([[1, 2], [3, 4], [5, 6]])  
print(arr.T)
```

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



arr



arr.T



다차원 배열의 평탄화

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




CSV 파일 읽고 쓰기

- 대부분의 학습 샘플들은 **CSV** 파일에 저장되어서 제공된다.
- 판다스(**Pandas**) 라이브러리를 사용

```
import numpy as np
import pandas as pd

x = pd.read_csv('countries.csv', header=0).values
print(x)
```

```
[[['KR' 'Korea' 98480 'Seoul' 48422644]
 ['US' 'USA' 9629091 'Washington' 310232863]
 ['JP' 'Japan' 377835 'Tokyo' 127288000]
 ['CN' 'China' 9596960 'Beijing' 1330044000]
 ['RU' 'Russia' 17100000 'Moscow' 140702000]]]
```



- 매트플롯은 **GNUplot**처럼 그래프를 그리는 라이브러리이다.
- 매트플롯은 무료이고 오픈 소스이다.
- 2가지 방식의 사용법
 - 함수 호출 방식으로 사용하는 방법
 - 객체 지향 방식으로 사용하는 방법

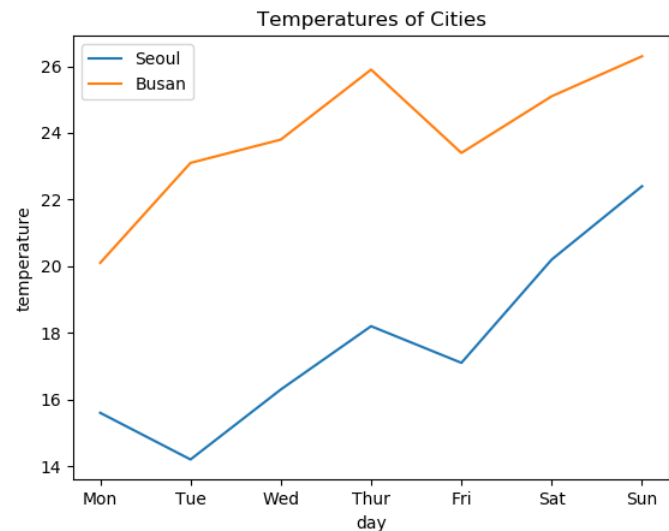


```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
X = [ "Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun" ]
Y1 = [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4]
Y2 = [20.1, 23.1, 23.8, 25.9, 23.4, 25.1, 26.3]
```

```
plt.plot(X, Y1, label="Seoul")
plt.plot(X, Y2, label="Busan")
plt.xlabel("day")
plt.ylabel("temperature")
plt.legend(loc="upper left")
plt.title("Temperatures of Cities")
plt.show()
```

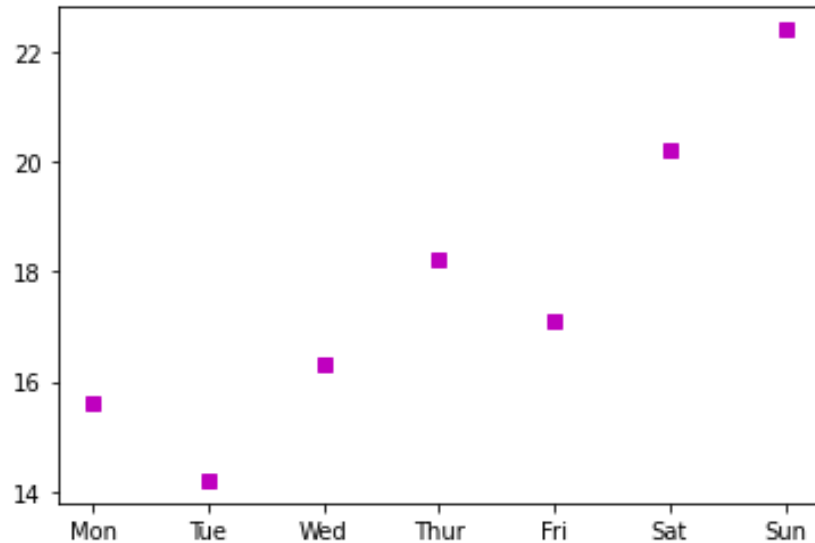
분리시켜서 그려도 됨





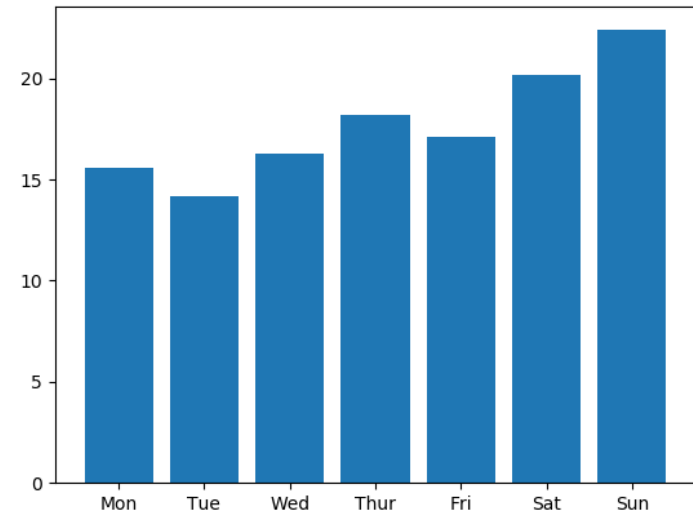
```
import matplotlib.pyplot as plt  
%matplotlib inline
```

```
X = [ "Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun" ]  
plt.plot(X, [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4], "sm")  
plt.show()
```



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

```
X = [ "Mon", "Tue", "Wed", "Thur", "Fri", "Sat", "Sun" ]  
Y = [15.6, 14.2, 16.3, 18.2, 17.1, 20.2, 22.4]  
plt.bar(X, Y)  
plt.show()
```

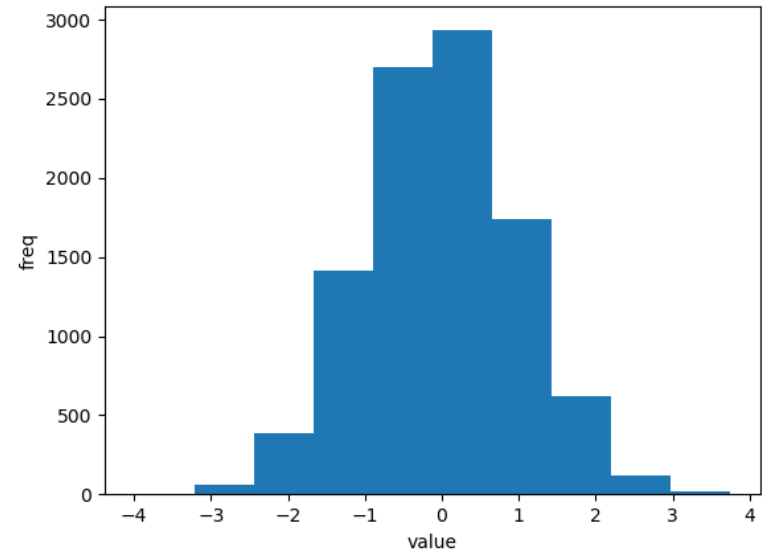




```
import matplotlib.pyplot as plt  
import numpy as np
```

```
numbers = np.random.normal(size=10000)
```

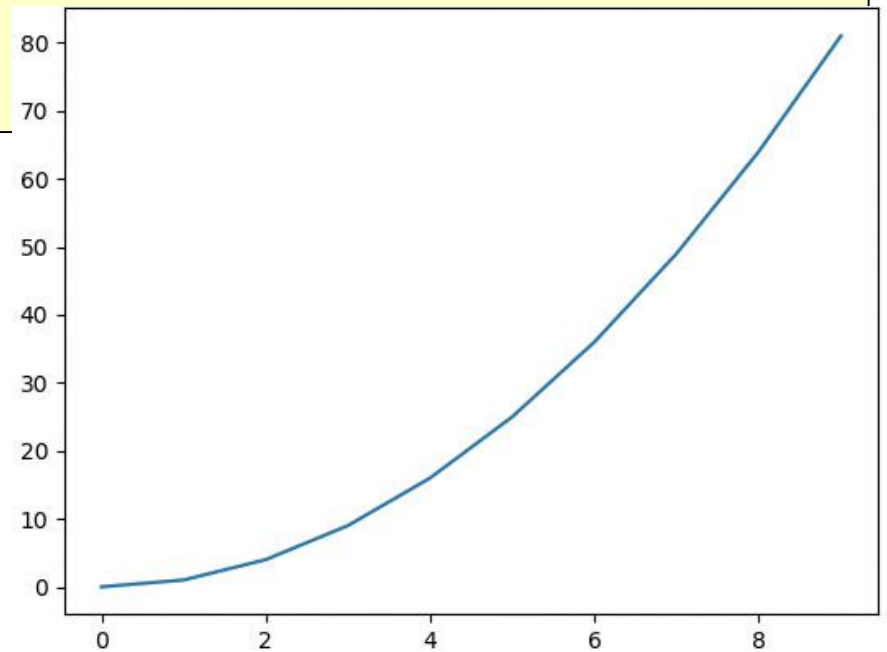
```
plt.hist(numbers)  
plt.xlabel("value")  
plt.ylabel("freq")  
plt.show()
```





넘파이와 맷플로틀

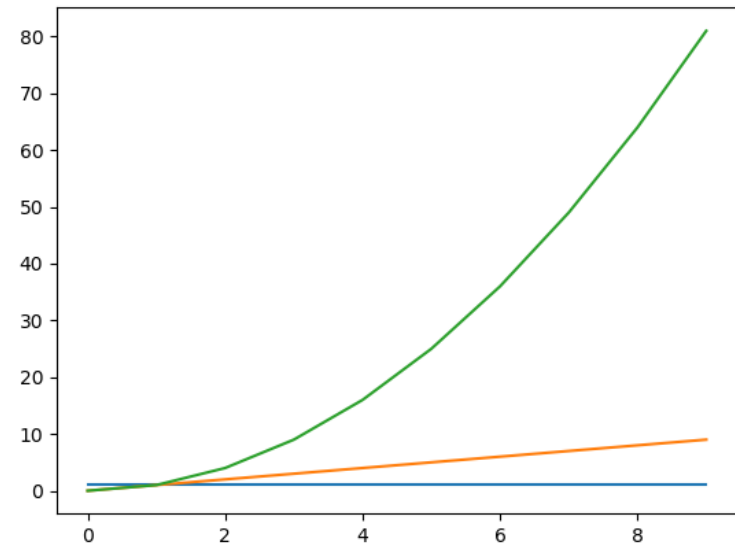
```
import matplotlib.pyplot as plt
import numpy as np
X = np.arange(0, 10)
Y = X**2
plt.plot(X, Y)
plt.show()
```





넘파이와 맷플로틀스

```
X = np.arange(0, 10)
Y1 = np.ones(10)
Y2 = X
Y3 = X**2
plt.plot(X, Y1, X, Y2, X, Y3)
plt.show()
```





Lab: 시그모이드 함수 미분값

```
import matplotlib.pyplot as plt
import numpy as np
```

```
def sigmoid(x):
    s=1/(1+np.exp(-x))
    ds=s*(1-s)
    return s,ds
```

시그모이드 함수 1차 미분 함수

```
X = np.linspace(-10, 10, 100)
Y1, Y2 = sigmoid(X)
```

```
plt.plot(X, Y1, X, Y2)
plt.xlabel("x")
plt.ylabel("Sigmoid(X), Sigmoid'(X)")
plt.show()
```

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \sigma(x) (1 - \sigma(x))$$

