

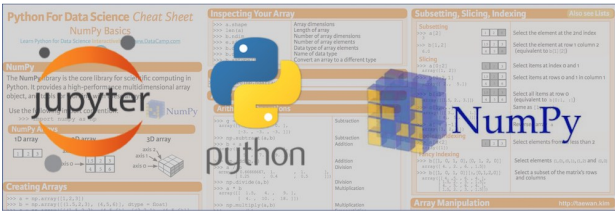
TAEWAN.KIM 블로그

STEP BY STEP - 우공이산(愚公移山) (HTTP://TAEWAN.KIM/)

MENU

파이썬 데이터 사이언스 Cheat Sheet: NumPy 기초, 기본

January 16, 2018 Machine Learning (/categories/machine-learning)



파이썬 기반 데이터 분석 환경에서 NumPy¹는 행렬 연산을 위한 핵심 라이브러리입니다. NumPy는 “**Numerical Python**”의 약자로 대규모 다차원 배열과 행렬 연산에 필요한 다양한 함수를 제공합니다. 특히 메모리 버퍼에 배열 데이터를 저장하고 처리하는 효율적인 인터페이스를 제공합니다. 파이썬 list 객체를 개선한 NumPy의 ndarray 객체를 사용하면 더 많은 데이터를 더 빠르게 처리할 수 있습니다.

NumPy는 다음과 같은 특징을 갖습니다.

- 강력한 N 차원 배열 객체
- 정교한 브로드캐스팅(Broadcast) 기능
- C/C++ 및 포트란 코드 통합 도구
- 유용한 선형 대수학, 푸리에 변환 및 난수 기능
- 범용적 데이터 처리에 사용 가능한 다차원 컨테이너

본 문서는 cs231n 강좌의 **Python Numpy Tutorial** (<http://cs231n.github.io/python-numpy-tutorial/>) 문서와 DataCamp의 **Python For Data Science Cheat Sheet NumPy Basics**

SEARCH...

Like Share 147 people like this

TOC (목차)

1. NumPy 배열
2. 배열 생성
2.1 파이썬 배열로 NumPy 배열 생성
2.2 배열 생성 및 초기화
2.3 데이터 생성 함수
2.4 난수 기반 배열 생성
2.5 약속된 난수
3. NumPy 입출력
4. 데이터 타입
5. 배열 상태 검사(Inspecting)
6. 도움말
7. 배열 연산
7.1 배열 일반 연산
7.2 브로드캐스팅
7.3 벡터연산
8. 배열 복사
9. 배열 정렬
10. 서브셋, 슬라이싱, 인덱싱
10.1 요소 선택
10.2 슬라이싱(Slicing)
10.3 불린 인덱싱 (Boolean Indexing)
10.4 팬시 인덱싱(Fancy Indexing)

(https://s3.amazonaws.com/assets.datacamp.com/blog_asset/s/Numpy_Python_Cheat_Sheet.pdf) 문서를 참조하여 작성하였습니다.

1. NumPy 배열

NumPy는 과학 연산을 위한 파이썬 핵심 라이브러리입니다. NumPy는 고성능 다차원 배열과 이런 배열을 처리하는 다양한 함수와 툴을 제공합니다.

파이썬에서 NumPy를 사용할 때, 다음과 같이 numpy 모듈을 “np”로 임포트하여 사용합니다.

```
import numpy as np
```

NumPy 라이브러리 버전은 다음과 같이 확인 할 수 있습니다.

In [1]:

```
import numpy as np
```

In [2]:

```
np.__version__
```

Out[2]:

'1.13.3'

NumPy 배열은 <그림 1>과 같이 다차원 배열을 지원합니다. NumPy 배열의 구조는 “**Shape**”으로 표현됩니다. Shape은 배열의 구조를 파이썬 튜플 자료형을 이용하여 정의합니다. 예를 들어 28X28 컬러 사진은 높이가 28, 폭이 28, 각 픽셀은 3개 채널(RGB)로 구성된 데이터 구조를 갖습니다. 즉 컬러 사진 데이터는 Shape이 (28, 28, 3)인 3차원 배열입니다. 다차원 배열은 입체적인 데이터 구조를 가지며, 데이터의 차원은 여러 갈래의 데이터 방향을 갖습니다. 다차원 배열의 데이터 방향을 axis로 표현할 수 있습니다. 행방향(높이), 열방향(폭), 채널 방향은 각각 axis=0, axis=1 그리고 axis=2로 지정됩니다. NumPy 집계(Aggregation) 함수는 배열 데이터의 집계 방향을 지정하는 axis 옵션을 제공합니다.

11. 배열 변환

11.1 전치(Transpose)

11.2 배열 형태 변경

11.3 배열 요소 추가 삭제

11.4 배열 결합

11.5 배열 분리

12. 참고자료

관련도서



(<http://www.yes24.com/Product/Goods/69335909>)

최신글

[2020/Books:05] '엔터프라이즈 데이터 플랫폼 구축' 리뷰 (/book/enterprise_data_platform/)

OCI Data Science 한글 폰트 설정 (/cloud/setup_hangul_font_on_data_science/)

[OCFS2]OCI Shared Block Volume 구성 (/cloud/oci_shared_block_volume/)

OCI Cloud Shell: 브라우저 기반 가상 터미널 (/cloud/oci_cloud_shell/)

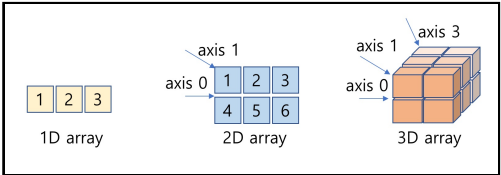


그림 1: NumPy 1차원, 2차원 및 3차원 배열과 Axis

NumPy 배열의 axis와 관련해서는 “**Numpy에서 np.sum 함수의 axis 이해** (http://taewan.kim/post/numpy_sum_axis/)” 문서를 참조하시기 바랍니다.

본 문서에서는 NumPy 객체의 정보를 출력하는 용도로 다음 pprint 함수를 공통으로 사용합니다.

```
def pprint(arr):
    print("type:{}".format(type(arr)))
    print("shape: {}, dimension: {}, dtype:{}".format(arr.shape, arr.ndim, arr.dtype))
    print("Array's Data:\n", arr)
```

2. 배열 생성

NumPy 배열은 numpy.ndarray 객체입니다. 이 절에서는 NumPy 배열(numpy.ndarray 객체) 생성 방법을 소개합니다.

2.1 파이썬 배열로 NumPy 배열 생성

파이썬 배열을 인자로 NumPy 배열을 생성할 수 있습니다. 파라미터로 list 객체와 데이터 타입(dtype)을 입력하여 NumPy 배열을 생성합니다. dtype을 생략할 경우, 입력된 list 객체의 요소 타입이 설정됩니다.

- 파이썬 1차원 배열(list)로 NumPy 배열 생성

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

S3FS를 이용한 OCI Object Storage 파일 시스템 마운트 (/cloud/mounting_oci_objectstorage_bucket_on_linux_mac/)

OCI에서 리눅스 VM 루트 파티션 (Root Partition) 확장 (/cloud/extending_root_partition_on_oci_linux/)

[2020/Books:04] '내 몸 치유력' 후기 (/book/%EB%82%B4%EB%AA%B8%EC%B9%98%EC%9C%A0%EB%A0%A5/)

[2020/Books:03] IT 개발자의 영어 필살기 (/book/english_for_developer/)

[2020/Books:02] '결론부터 써라' 후기 (/book/%EA%B2%B0%EB%A1%A0%EB%B6%80%ED%84%B0%EC%8D%A8%EB%9D%BC/)

[2020/Books:01] '지적자본론' 후기 (/book/%EC%A7%80%EC%A0%81%EC%9E%90%EB%B3%B8%EB%A1%A0/)

카테고리

(<http://taewan.kim/categories/>)

Bigdata (<http://taewan.kim/categories/bigdata>)

Blogs.oracle.com (<http://taewan.kim/categories/blogs.oracle.com>)

Book (<http://taewan.kim/categories/book>)

Cloud (<http://taewan.kim/categories/cloud>)

```
In [3]:
pprint(a)

type:<class 'numpy.ndarray'>
shape: (3,), dimension: 1, dtype:int64
Array's Data:
[1 2 3]
```

- 파이썬 2차원 배열로 NumPy 배열 생성, 원소 데이터 타입 지정

```
In [4]:
arr = [(1,2,3), (4,5,6)]
a= np.array(arr, dtype = float)
```

```
In [5]:
pprint(a)

type:<class 'numpy.ndarray'>
shape: (2, 3), dimension: 2, dtype:float64
Array's Data:
[[ 1.  2.  3.]
 [ 4.  5.  6.]]
```

- 파이썬 3차원 배열로 NumPy 배열 생성, 원소 데이터 타입 지정

```
In [6]:
arr = np.array([[[1,2,3], [4,5,6]], [[3,2,1], [4,5,6]]], dtype = float)
a= np.array(arr, dtype = float)
```

```
In [7]:
pprint(a)

type:<class 'numpy.ndarray'>
shape: (2, 2, 3), dimension: 3, dtype:float64
Array's Data:
[[[ 1.  2.  3.]
 [ 4.  5.  6.]]
 [[ 3.  2.  1.]
 [ 4.  5.  6.]]]
```

2.2 배열 생성 및 초기화

Graalvm (<http://taewan.kim/categories/graalvm>)

It-Life (<http://taewan.kim/categories/it-life>)

Java (<http://taewan.kim/categories/java>)

Language (<http://taewan.kim/categories/language>)

Life (<http://taewan.kim/categories/life>)

Machine-Learning (<http://taewan.kim/categories/machine-learning>)

Math (<http://taewan.kim/categories/math>)

Minsu (<http://taewan.kim/categories/minsu>)

Mysql (<http://taewan.kim/categories/mysql>)

Oracle (<http://taewan.kim/categories/oracle>)

Seminar (<http://taewan.kim/categories/seminar>)

Tech-Event (<http://taewan.kim/categories/tech-event>)

Tech-Tip (<http://taewan.kim/categories/tech-tip>)

NumPy는 원하는 shape으로 배열을 설정하고, 각 요소를 특정 값으로 초기화하는 zeros, ones, full, eye 함수를 제공합니다. 또한, 파라미터로 입력한 배열과 같은 shape의 배열을 만드는 zeros_like, ones_like, full_like 함수도 제공합니다. 이 함수를 이용하여 배열 생성하고 초기화할 수 있습니다.

np.zeros 함수

- zeros(shape, dtype=float, order='C')
- 지정된 shape의 배열을 생성하고, 모든 요소를 0으로 초기화

In [2]:

```
a = np.zeros((3, 4))
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (3, 4), dimension: 2, dtype:float64
Array's Data:
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
```

np.ones 함수

- np.ones(shape, dtype=None, order='C')
- 지정된 shape의 배열을 생성하고, 모든 요소를 1로 초기화

In [3]:

```
a = np.ones((2,3,4),dtype=np.int16)
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (2, 3, 4), dimension: 3, dtype:int16
Array's Data:
[[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
 [[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]]
```

Tip

(<http://taewan.kim/categories/tip>)

Youtube

(<http://taewan.kim/categories/youtube>)

맛집

(<http://taewan.kim/categories/%eb%a7%9b%ec%a7%91>)

번역

(<http://taewan.kim/categories/%eb%b2%88%ec%97%ad>)

오라클-클라우드

(<http://taewan.kim/categories/%ec%98%a4%eb%9d%bc%ed%81%b4-%ed%81%b4%eb%9d%bc%ec%9a%b0%eb%93%9c>)

짜투리

(<http://taewan.kim/categories/%ec%a7%9c%ed%88%ac%eb%a6%ac>)

SNS(SOCIAL NETWORK SERVICE)



([https://github.com/oracloud-](https://github.com/oracloud-kr-team)



kr-team)

([https://www.slideshare.net/ssu](https://www.slideshare.net/ssusercda07e)



sercda07e)

(<https://www.youtube.com/channel/UCbojr3TLlqeDqpBURRdb>



_lg)

(mailto:taewanme@gmail.com)

관심 사이트

In [6]:

```
a = np.empty((4,2))
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (4, 2), dimension: 2, dtype:float64
Array's Data:
[[ 0.00000000e+000  6.91240343e-310]
 [ 6.91240500e-310  5.39088070e-317]
 [ 5.39084907e-317  6.91239798e-310]
 [ 3.16202013e-322  6.91239798e-310]]
```

like 함수

- numpy는 지정된 배열과 shape이 같은 행렬을 만드는 like 함수를 제공합니다.
- np.zeros_like
- np.ones_like
- np.full_like
- np.empty_like

In [7]:

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

```
type:<class 'numpy.ndarray'>
shape: (2, 3), dimension: 2, dtype:int64
Array's Data:
[[1 1 1]
 [1 1 1]]
```

2.3 데이터 생성 함수

NumPy는 주어진 조건으로 데이터를 생성한 후, 배열을 만드는 데이터 생성 함수를 제공합니다.

- numpy.linspace
- numpy.arange
- numpy.logspace

np.linspace 함수

- `numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)`
- start부터 stop의 범위에서 num개를 균일한 간격으로 데이터를 생성하고 배열을 만드는 함수
- 요소 개수를 기준으로 균등 간격의 배열을 생성

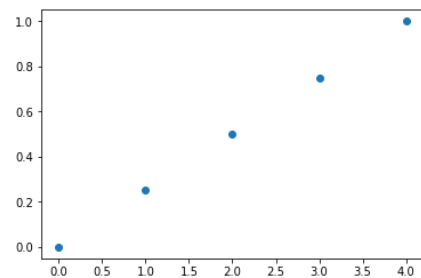
In [2]:

```
a = np.linspace(0, 1, 5)
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (5,), dimension: 1, dtype:float64
4
Array's Data:
[ 0.    0.25  0.5   0.75  1.   ]
```

In [3]:

```
# linspace의 데이터 추출 시각화
import matplotlib.pyplot as plt
plt.plot(a, 'o')
plt.show()
```

**np.arange 함수**

- `numpy.arange([start,] stop[, step,], dtype=None)`
- start부터 stop 미만까지 step 간격으로 데이터 생성한 후 배열을 만들
- 범위내에서 간격을 기준 균등 간격의 배열
- 요소의 개수가 아닌 데이터의 간격을 기준으로 배열 생성

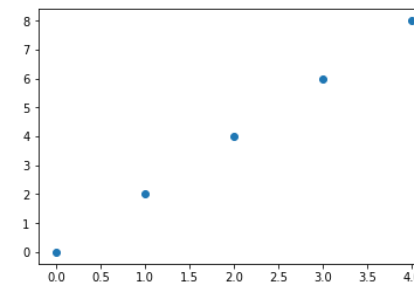
In [4]:

```
a = np.arange(0, 10, 2, np.float)
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (5,), dimension: 1, dtype:float64
4
Array's Data:
[ 0.  2.  4.  6.  8.]
```

In [5]:

```
# arange의 데이터 추출 시각화
import matplotlib.pyplot as plt
plt.plot(a, 'o')
plt.show()
```

**np.logspace 함수**

- `numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)`
- 로그 스케일의 linspace 함수
- 로그 스케일로 지정된 범위에서 num 개수만큼 균등 간격으로 데이터 생성한 후 배열 만들

In [6]:

```
a = np.logspace(0.1, 1, 20, endpoint=True)
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (20,), dimension: 1, dtype:float64
64
Array's Data:
[ 1.25892541  1.40400425  1.565802
 1.74624535  1.94748304
 2.1719114   2.42220294  2.70133812
 3.0126409   3.35981829
 3.74700446  4.17881006  4.66037703
 5.19743987  5.79639395
 6.46437163  7.2093272  8.04013161
 8.9666781   10.   ]
```

np.full 함수

- np.full(shape, fill_value, dtype=None, order='C')
- 지정된 shape의 배열을 생성하고, 모든 요소를 지정한 "fill_value"로 초기화

In [4]:

```
a = np.full((2,2),7)
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (2, 2), dimension: 2, dtype:int64
Array's Data:
[[7 7]
 [7 7]]
```

np.eye 함수

- np.eye(N, M=None, k=0, dtype=<class 'float'>)
- (N, N) shape의 단위 행렬(Unit Matrix)을 생성

In [5]:

```
np.eye(4)
```

Out[5]:

```
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

np.empty 함수

- empty(shape, dtype=float, order='C')
- 지정된 shape의 배열 생성
- 요소의 초기화 과정에 없고, 기존 메모리값을 그대로 사용
- 배열 생성비용이 가장 저렴하고 빠름
- 배열 사용 시 주의가 필요(초기화를 고려)

Steemit 블로그

(<http://steemit.com/@taewan.kim>)

Docker Hub for taewan.kim

(<https://hub.docker.com/u/taewanme/>)

Github REPOSITORY for

Notebooks/a>

(<https://github.com/taewanme/notebooks4til>)

(<https://github.com/taewanme/notebooks4til>)

(<https://github.com/taewanme/notebooks4til>)

(<https://github.com/taewanme/notebooks4til>)

(<https://github.com/taewanme/notebooks4til>)

LICESNE

(<https://github.com/taewanme/notebooks4til>)

(<https://github.com/taewanme/notebooks4til>)

(<http://creativecommons.org/licenses/by-nc-sa/4.0/>)

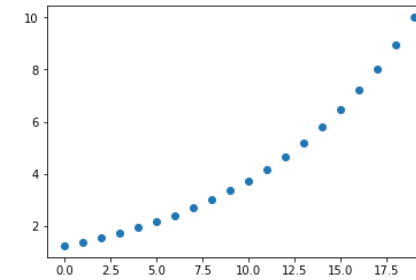
이 저작물은 크리에이티브 커먼즈 저작자표시-비영리-동일조건변경허락 4.0 국제 라이선스

(<http://creativecommons.org/licenses/by-nc-sa/4.0/>)에 따라 이용할 수 있습니다.



In [7]:

```
# Logspace의 데이터 추출 시각화
import matplotlib.pyplot as plt
plt.plot(a, 'o')
plt.show()
```

**2.4 난수 기반 배열 생성**

NumPy는 난수 발생 및 배열 생성을 생성하는 numpy.random 모듈을 제공합니다. 이 절에서는 이 모듈의 함수 사용법을 소개합니다.

numpy.random 모듈은 다음과 같은 함수를 제공합니다.

- np.random.normal
- np.random.rand
- np.random.randn
- np.random.randint
- np.random.random

np.random.normal

- normal(loc=0.0, scale=1.0, size=None)
- 정규 분포 확률 밀도에서 표본 추출
- loc: 정규 분포의 평균
- scale: 표준편차

In [2]:

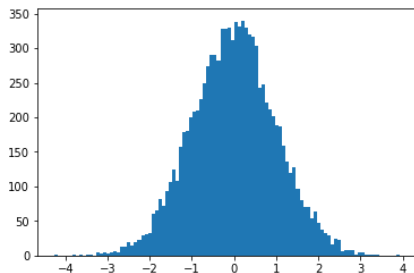
```
mean = 0
std = 1
a = np.random.normal(mean, std, (2, 3))
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (2, 3), dimension: 2, dtype:float64
Array's Data:
[[ 1.4192442 -2.0771293  1.84898108]
 [-0.12303317  1.04533993  1.94901387]]
```

- np.random.normal이 생성한 난수는 정규 분포의 형상을 갖습니다.
- 다음 예제는 정규 분포로 10000개 표본을 뽑은 결과를 히스토그램으로 표현한 예입니다.
- 표본 10000개의 배열을 100개 구간으로 구분할 때, 정규 분포 형태를 보이고 있습니다.

In [3]:

```
data = np.random.normal(0, 1, 10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=100)
plt.show()
```



np.random.rand

- numpy.random.rand(d0, d1, ..., dn)
- Shape이 (d0, d1, ..., dn) 인 배열 생성 후 난수로 초기화
- 난수: [0, 1)의 균등 분포(Uniform Distribution) 형상으로 표본 추출
- Gaussina normal

In [4]:

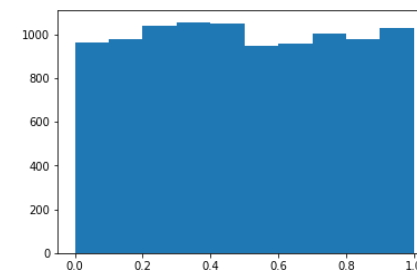
```
a = np.random.rand(3,2)
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (3, 2), dimension: 2, dtype:float64
Array's Data:
[[ 0.1258167  0.25474262]
 [ 0.25514046  0.0918946 ]
 [ 0.19843316  0.73586066]]
```

- np.random.rand는 균등한 비율로 표본 추출
- 다음 예제는 균등 분포로 10000개를 표본 추출한 결과를 히스토그램으로 표현한 예입니다.
- 표본 10000개의 배열을 10개 구간으로 구분했을 때 균등한 분포 형태를 보이고 있습니다.

In [5]:

```
data = np.random.rand(10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=10)
plt.show()
```



np.random.randn

- numpy.random.randn(d0, d1, ..., dn)
- (d0, d1, ..., dn) shape 배열 생성 후 난수로 초기화
- 난수: 표준 정규 분포(standard normal distribution)에서 표본 추출

In [6]:

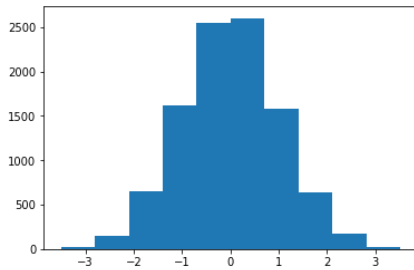
```
a = np.random.randn(2, 4)
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (2, 4), dimension: 2, dtype:float64
Array's Data:
[[ 0.81791892  0.74967685 -0.20023471
  0.76089888]
 [-1.13037451 -0.52569743 -1.33934774
  0.75105868]]
```

- np.random.randn은 정규 분포로 표본 추출
- 다음 예제는 정규 분포로 10000개를 표본 추출한 결과를 히스토그램으로 표현한 예입니다.
- 표본 10000개의 배열을 10개 구간으로 구분했을때 정규 분포 형태를 보이고 있습니다.

In [7]:

```
data = np.random.randn(10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=10)
plt.show()
```



np.random.randint

- numpy.random.randint(low, high=None, size=None, dtype='i')
- 지정된 shape으로 배열을 만들고 low 부터 high 미만의 범위에서 정수 표본 추출

In [8]:

```
a = np.random.randint(5, 10, size=(2, 4))
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (2, 4), dimension: 2, dtype:int64
Array's Data:
[[5 5 6 6]
 [7 9 7 9]]
```

In [9]:

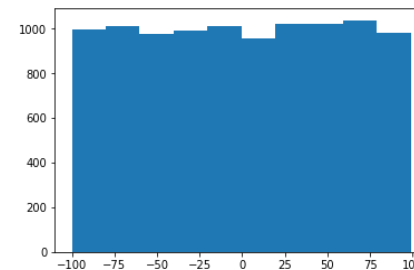
```
a = np.random.randint(1, size=10)
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (10,), dimension: 1, dtype:int64
Array's Data:
[0 0 0 0 0 0 0 0 0 0]
```

- 100에서 100의 범위에서 정수를 균등하게 표본 추출합니다.
- 다음 예제에서 균등 분포로 10000개를 표본 추출한 결과를 히스토그램으로 표현한 예입니다.
- 표본 10000개의 배열을 10개 구간으로 구분했을때 균등한 분포 형태를 보이고 있습니다.

In [10]:

```
data = np.random.randint(-100, 100, 10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=10)
plt.show()
```



np.random.random

- np.random.random(size=None)
- 난수: [0., 1.)의 균등 분포(Uniform Distribution)에서 표본 추출

In [11]:

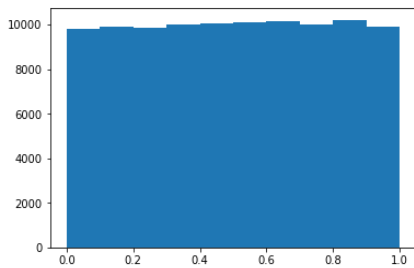
```
a = np.random.random((2, 4))
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (2, 4), dimension: 2, dtype:float64
Array's Data:
[[ 0.92646678  0.02811114  0.97379431
  0.86712785]
 [ 0.18829149  0.78809537  0.52076073
  0.71967828]]
```

- np.random.random은 균등 분포로 표본을 추출합니다.
- 다음 예제는 정규 분포로 10000개를 표본 추출한 결과를 히스토그램으로 표현한 예입니다.
- 표본 10000개의 배열을 10개 구간으로 구분했을때 정규 분포 형태를 보이고 있습니다.

In [12]:

```
data = np.random.random(100000)
import matplotlib.pyplot as plt
plt.hist(data, bins=10)
plt.show()
```



2.5 약속된 난수

무작위 수를 만드는 난수는 특정 시작 숫자로부터 난수처럼 보이는 수열을 만드는 알고리즘의 결과물입니다. 따라서 시작점을 설정함으로써 난수 발생을 재연할 수 있습니다. 난수의 시작점을 설정하는 함수가 np.random.seed 입니다.

- random 모듈의 함수는 실행할 때 마다 무작위 수를 반환합니다.

In [2]:

```
np.random.random((2, 2))
```

Out[2]:

```
array([[ 0.37177011,  0.80381439],
       [ 0.98299691,  0.91079526]])
```

In [3]:

```
np.random.randint(0, 10, (2, 3))
```

Out[3]:

```
array([[0, 1, 7],
       [0, 2, 3]])
```

In [4]:

```
np.random.random((2, 2))
```

Out[4]:

```
array([[ 0.4573231 ,  0.1649216 ],
       [ 0.76895461,  0.96333133]])
```

In [5]:

```
np.random.randint(0, 10, (2, 3))
```

Out[5]:

```
array([[4, 5, 4],
       [5, 1, 1]])
```

- np.random.seed 함수를 이용한 무작위수 재연
- np.random.seed(100)을 기준으로 동일한 무작위수로 초기화된 배열이 만들어 지고 있습니다.

In [6]:

```
# seed 값을 설정하여 아래에서 난수가 재연 가능하도록 함
np.random.seed(100)
```

In [7]:

```
np.random.random((2, 2))
```

Out[7]:

```
array([[ 0.54340494,  0.27836939],
       [ 0.42451759,  0.84477613]])
```



```
In [8]:
np.random.randint(0, 10, (2, 3))

Out[8]:
array([[4, 2, 5],
       [2, 2, 2]])
```

```
In [9]:
# seed 값 재 설정
np.random.seed(100)
```

```
In [10]:
# 위 난수의 재연
np.random.random((2, 2))
```

```
Out[10]:
array([[ 0.54340494,  0.27836939],
       [ 0.42451759,  0.84477613]])
```

```
In [11]:
# 위 난수의 재연
np.random.randint(0, 10, (2, 3))
```

```
Out[11]:
array([[4, 2, 5],
       [2, 2, 2]])
```

3. NumPy 입출력

NumPy는 배열 객체를 바이너리 파일 혹은 텍스트 파일에 저장하고 로딩하는 기능을 제공합니다.

함수명	기능	파일포맷
np.save()	NumPy 배열 객체 1개를 파일에 저장	바이너리
np.savez()	NumPy 배열 객체 복수개를 파일에 저장	바이너리
np.load()	NumPy 배열 저장 파일로 부터 객체 로딩	바이너리
np.loadtxt()	텍스트 파일로 부터 배열 로딩	텍스트
np.savetxt()	텍스트 파일에 NumPy 배열 객체 저장	텍스트

- 예제로 다음과 같은 a, b 두 개 배열을 사용합니다.

```
In [2]:
a = np.random.randint(0, 10, (2, 3))
b = np.random.randint(0, 10, (2, 3))
pprint(a)
pprint(b)

type:<class 'numpy.ndarray'>
shape: (2, 3), dimension: 2, dtype:int64
4
Array's Data:
[[8 7 4]
 [4 3 8]]
type:<class 'numpy.ndarray'>
shape: (2, 3), dimension: 2, dtype:int64
4
Array's Data:
[[6 9 8]
 [9 7 7]]
```

배열 객체 저장

np.save 함수와 np.savez 함수를 이용하여 배열 객체를 파일로 저장할 수 있습니다.

- np.save: 1개 배열 저장, 확장자: npy
- np.savez: 복수 배열을 1개의 파일에 저장, 확장자: pnz
- 배열 저장 파일은 바이너리 형태입니다.

- 1개 배열 파일에 저장

```
In [3]:
# a 배열 파일에 저장
np.save("./my_array1", a)
```

```
In [4]:
# 파일 조회
!ls -al my_array1*

-rw-r--r-- 1 root root 128  1월 17  201
8 my_array1.npy
```

- 복수 배열을 1개 파일에 저장

```
In [5]:
# a, b 두 개 배열을 파일에 저장
np.savez("my_array2", a, b)
```

In [6]:

```
# 파일 조회
!ls -al my_array2*
```

```
-rw-r--r-- 1 root root 466  1월 17  201
8 my_array2.npz
```

파일로 부터 배열 객체 로딩

- npy와 npz 파일은 np.load 함수로 읽을 수 있습니다.

In [7]:

```
# 1개 배열 로딩
np.load("./my_array1.npy")
```

Out[7]:

```
array([[8, 7, 4],
       [4, 3, 8]])
```

In [8]:

```
# 복수 파일 로딩
npzfiles = np.load("./my_array2.npz")
npzfiles.files
```

Out[8]:

```
['arr_0', 'arr_1']
```

In [9]:

```
npzfiles['arr_0']
```

Out[9]:

```
array([[8, 7, 4],
       [4, 3, 8]])
```

In [10]:

```
npzfiles['arr_1']
```

Out[10]:

```
array([[6, 9, 8],
       [9, 7, 7]])
```

텍스트 파일 로딩

텍스트 파일을 np.loadtxt 로 로딩 할 수 있습니다.

- np.loadtxt(fname, dtype=<class 'float'>, comments='#', delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0)
 - fname: 파일명
 - dtype: 데이터 타입
 - comments: comment 시작 부호
 - delimiter: 구분자
 - skiprows: 제외 라인 수(header 제거용)

In [11]:

```
# 데이터 파일 위치
!ls -al ./data/simple.csv
```

```
-rw-r--r-- 1 root root 15  1월 14 06:32
./data/simple.csv
```

In [12]:

```
# 데이터 파일 내용
!cat ./data/simple.csv
```

```
1 2 3
4 5 6
```

In [13]:

```
#기본 데이터 타입은 float을 설정됩니다.
#파일 데이터 배열로 로딩 및 데이터 타입 지정
np.loadtxt("./data/simple.csv")
```

Out[13]:

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

In [14]:

```
#dtype 속성으로 데이터 타입 변경 가능합니다.
#파일 데이터 배열로 로딩 및 데이터 타입 지정
np.loadtxt("./data/simple.csv", dtype=np.int)
```

Out[14]:

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

- 텍스트를 포함한 파일이 경우 dtyped으로 컬럼 명과 데이터 타입을 설정해야 합니다.

In [15]:

```
# 대상 데이터 파일 조회
!ls -al ./data/president_height.csv

-rw-r--r-- 1 root root 987  1월 13  07:0
4 ./data/president_height.csv
```

In [16]:

```
# 데이터 파일 내용
# 헤더 라인: 1개
# 문자열 포함
!head -n 3 ./data/president_height.csv
```

```
order,name,height(cm)
1,George Washington,189
2,John Adams,170
```

문자열을 포함하는 텍스트 파일 로딩

- president_height.csv 파일은 숫자와 문자를 모두 포함하는 데이터 파일입니다.
- dtype을 이용하여 컬럼 타입을 지정하여 로딩합니다.
- delimiter와 skiprows를 이용하여 구분자와 무시해야 할 라인을 지정합니다.

In [17]:

```
# dtype에 dict 형식으로 데이터 타입 지정
data = np.loadtxt("./data/president_height.csv", del
imiter=";", skiprows=1, dtype={
'names': ("order","name","height"),
'formats':('i', 'S20', 'f')
})
# 배열 데이터 출력
data[:3]
```

Out[17]:

```
array([(1, b'George Washington', 18
9.), (2, b'John Adams', 170.),
(3, b'Thomas Jefferson', 189.)],
      dtype=[('order', '<i4'), ('name',
'S20'), ('height', '<f4')])
```

배열 객체 텍스트 파일로 저장

- np.savetxt 함수를 이용하여 배열 객체를 텍스트 파일로 저장할 수 있습니다.
- np.savetxt(fname, X, fmt='%18e', delimiter=' ', newline='\n', header="", footer="", comments='# ')

In [18]:

```
# 데모 데이터 생성
data = np.random.random((3, 4))
pprint(data)

type:<class 'numpy.ndarray'>
shape: (3, 4), dimension: 2, dtype:floa
t64
Array's Data:
[[ 0.21554899  0.56103576  0.71822224
 0.42060378]
 [ 0.59906291  0.51097642  0.37703684
 0.48276954]
 [ 0.1889987  0.62604535  0.88074236
 0.01603881]]
```

In [19]:

```
# 배열 객체 텍스트 파일로 저장
np.savetxt("./data/saved.csv", data, delimiter=",")
```

In [20]:

```
# 파일 조회
!ls -al ./data/saved.csv

-rw-r--r-- 1 root root 300  1월 17  201
8 ./data/saved.csv
```

In [21]:

```
# 파일 내용 조회
!cat ./data/saved.csv

2.155489877688239186e-01,5.610357577617
570701e-01,7.18222391102696113e-01,4.2
06037807135534212e-01
5.990629079875264829e-01,5.109764156401
283008e-01,3.770368358788609431e-01,4.8
27695415663437739e-01
1.889986982230907886e-01,6.260453490415
701649e-01,8.807423613788849526e-01,1.6
03880803818769074e-02
```

In [22]:

```
# 데이터 파일 로딩
np.loadtxt('./data/saved.csv', delimiter=',')
```

Out[22]:

```
array([[ 0.21554899,  0.56103576,  0.71
822224,  0.42060378],
[ 0.59906291,  0.51097642,  0.3770
3684,  0.48276954],
[ 0.1889987 ,  0.62604535,  0.8807
4236,  0.01603881]])
```

4. 데이터 타입

NumPy는 다음과 같은 데이터 타입을 지원합니다. 배열을 생성할 때 dtype속성으로 다음과 같은 데이터 타입을 지정할 수 있습니다.

- np.int64 : 64 비트 정수 타입
- np.float32 : 32 비트 부동 소수 타입
- np.complex : 복소수 (128 float)
- np.bool : 불린 타입 (Trur, False)
- np.object : 파이썬 객체 타입
- np.string_ : 고정자리 스트링 타입
- np.unicode_ : 고정자리 유니코드 타입

5. 배열 상태 검사(Inspecting)

NumPy는 배열의 상태를 검사하는 다음과 같은 방법을 제공합니다.

배열 속성 검사 항목	배열 속성 확인 방법	예시	결과
배열 shape	np.ndarray.shape 속성	arr.shape	(5, 2, 3)
배열 길이	일차원의 배열 길이 확인	len(arr)	5
배열 차원	np.ndarray.ndim 속성	arr.ndim	3
배열 요소 수	np.ndarray.size 속성	arr.size	30
배열 타입	np.ndarray.dtype 속성	arr.dtype	dtype('float64')
배열 타입 명	np.ndarray.dtype.name 속성	arr.dtype.name	float64
배열 타입 변환	np.ndarray.astype 함수	arr.astype(np.int)	배열 타입 변환

NumPy 배열 객체는 다음과 같은 방식으로 속성을 확인할 수 있습니다.

In [2]:

```
#데모 배열 객체 생성
arr = np.random.random((5,2,3))
```

In [3]:

```
#배열 타입 조회
type(arr)
```

Out[3]:

numpy.ndarray

In [4]:

```
# 배열의 shape 확인
arr.shape
```

Out[4]:

(5, 2, 3)

In [5]:

```
# 배열의 길이
len(arr)
```

Out[5]:

5

In [6]:

```
# 배열의 차원 수
arr.ndim
```

Out[6]:

3

In [7]:

```
# 배열의 요소 수: shape(k, m, n) ==> k*m*n
arr.size
```

Out[7]:

30

In [8]:

```
# 배열 타입 확인
arr.dtype
```

Out[8]:

dtype('float64')

```
In [9]:
# 배열 타입명
arr.dtype.name
```

```
Out[9]:
'float64'
```

```
In [10]:
# 배열 요소를 int로 변환
# 요소의 실제 값이 변환되는 것이 아님
# View의 출력 타입과 연산을 변환하는 것
arr.astype(np.int)
```

```
Out[10]:
array([[0, 0, 0],
       [0, 0, 0]],

      [[0, 0, 0],
       [0, 0, 0]],

      [[0, 0, 0],
       [0, 0, 0]],

      [[0, 0, 0],
       [0, 0, 0]])
```

```
In [11]:
# np.float으로 타입을 다시 변환하면 np.int 변환 이전 값
으로 모든 원소 값이 복원됨
arr.astype(np.float)
```

```
Out[11]:
array([[ 0.16384724,  0.14102935,  0.1
3880948],
       [ 0.75155573,  0.41813362,  0.322
00517]],

      [[ 0.15771594,  0.15517552,  0.735
18959],
       [ 0.45756098,  0.47543577,  0.577
1006 ]],

      [[ 0.21845306,  0.10044739,  0.637
70484],
       [ 0.36300707,  0.67294698,  0.928
7396 ]],

      [[ 0.17435257,  0.90261209,  0.163
89656],
       [ 0.8118577 ,  0.36231545,  0.261
70523]],

      [[ 0.41849605,  0.44730829,  0.038
15196],
       [ 0.03938546,  0.45850899,  0.150
59227]])
```

6. 도움말

NumPy의 모든 API는 np.info 함수를 이용하여 도움말을 확인할 수 있습니다

In [2]:

```
np.info(np.ndarray.dtype)
```

Data-type of the array's elements.

Parameters

None

Returns

d : numpy dtype object

See Also

numpy.dtype

Examples

```
>>> x
array([[0, 1],
       [2, 3]])
>>> x.dtype
dtype('int32')
>>> type(x.dtype)
<type 'numpy.dtype'>
```

In [3]:

```
np.info(np.squeeze)
```

squeeze(a, axis=None)

Remove single-dimensional entries from the shape of an array.

Parameters

a : array_like

Input data.

axis : None or int or tuple of ints, optional

.. versionadded:: 1.7.0

Selects a subset of the single-dimensional entries in the shape. If an axis is selected with shape entry greater than one, an error is raised.

Returns

squeezed : ndarray

The input array, but with all or a subset of the dimensions of length 1 removed. This is always `a` itself or a view into `a`.

Raises

ValueError

If `axis` is not `None`, and an axis being squeezed is not of length 1

See Also

expand_dims : The inverse operation, adding singleton dimensions
 reshape : Insert, remove, and combine dimensions, and resize existing ones

Examples

```
>>> x = np.array([[[0], [1], [2]]])
>>> x.shape
(1, 3, 1)
>>> np.squeeze(x).shape
(3,)
>>> np.squeeze(x, axis=0).shape
(3, 1)
>>> np.squeeze(x, axis=1).shape
Traceback (most recent call last):
...
ValueError: cannot select an axis to squeeze out which has size not equal to one
>>> np.squeeze(x, axis=2).shape
(1, 3)
```

7. 배열 연산

7.1 배열 일반 연산

NumPy는 기본 연산자를 연산자 재정의하여 배열(행렬) 연산에 대한 직관적으로 표현을 강화하였습니다. 모든 산술 연산 함수는 np 모듈에 포함되어 있습니다.

산술 연산(Arithmetic Operations)

NumPy는 기본 연산자를 연산자 재정의하여 배열(행렬) 연산에 대한 직관적으로 표현을 강화하였습니다. 모든 산술 연산 함수는 np 모듈에 포함되어 있습니다.

In [2]:

```
# arange로 1부터 10 미만의 범위에서 1씩 증가하는 배열 생성
# 배열의 shape을 (3, 3)으로 지정
a = np.arange(1, 10).reshape(3, 3)
pprint(a)
```

```
shape: (3, 3), dimension: 2, dtype:int64
4
Array's Data
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [3]:

```
# arange로 9부터 0까지 범위에서 1씩 감소하는 배열 생성
# 배열의 shape을 (3, 3)으로 지정
b = np.arange(9, 0, -1).reshape(3, 3)
pprint(b)
```

```
shape: (3, 3), dimension: 2, dtype:int64
4
Array's Data
[[9 8 7]
 [6 5 4]
 [3 2 1]]
```

배열 연산: 뺄셈, -

In [4]:

```
a - b
```

Out[4]:

```
array([[ -8,  -6,  -4],
       [-2,   0,   2],
       [ 4,   6,   8]])
```

In [5]:

```
np.subtract(a, b)
```

Out[5]:

```
array([[ -8,  -6,  -4],
       [-2,   0,   2],
       [ 4,   6,   8]])
```

배열 연산: 덧셈, +

In [6]:

```
a+b
```

Out[6]:

```
array([[10, 10, 10],
       [10, 10, 10],
       [10, 10, 10]])
```

In [7]:

```
np.add(a, b)
```

Out[7]:

```
array([[10, 10, 10],
       [10, 10, 10],
       [10, 10, 10]])
```

배열 연산: 나눗셈, /

In [8]:

```
a/b
```

Out[8]:

```
array([[ 0.11111111,  0.25      ,  0.42
857143],
       [ 0.66666667,  1.        ,  1.5
],
       [ 2.33333333,  4.        ,  9.
]])
```

In [9]:

```
np.divide(a, b)
```

Out[9]:

```
array([[ 0.11111111,  0.25      ,  0.42
857143],
       [ 0.66666667,  1.        ,  1.5
],
       [ 2.33333333,  4.        ,  9.
]])
```

배열 연산: 곱셈, *

In [10]:

```
a*b
```

Out[10]:

```
array([[ 9, 16, 21],
       [24, 25, 24],
       [21, 16,  9]])
```

In [11]:

```
np.multiply(a, b)
```

Out[11]:

```
array([[ 9, 16, 21],
       [24, 25, 24],
       [21, 16,  9]])
```

배열 연산: 지수

In [12]:

```
np.exp(b)
```

Out[12]:

```
array([[ 8.10308393e+03,  2.98095799e
+03,  1.09663316e+03],
       [ 4.03428793e+02,  1.48413159e+0
2,  5.45981500e+01],
       [ 2.00855369e+01,  7.38905610e+0
0,  2.71828183e+00]])
```

배열 연산: 제곱근

In [13]:

```
np.sqrt(a)
```

Out[13]:

```
array([[ 1.        ,  1.41421356,  1.73
205081],
       [ 2.        ,  2.23606798,  2.4494
8974],
       [ 2.64575131,  2.82842712,  3.
]])
```

배열 연산: sin

In [14]:

```
np.sin(a)
```

Out[14]:

```
array([[ 0.84147098,  0.90929743,  0.14
112001],
       [-0.7568025 , -0.95892427, -0.2794
155 ],
       [ 0.6569866 ,  0.98935825,  0.4121
1849]])
```

배열 연산: cos

In [15]:

```
np.cos(a)
```

Out[15]:

```
array([[ 0.54030231, -0.41614684, -0.98
99925 ],
       [-0.65364362,  0.28366219,  0.9601
7029],
       [ 0.75390225, -0.14550003, -0.9111
3026]])
```

배열 연산: tan

In [16]:

```
np.tan(a)
```

Out[16]:

```
array([[ 1.55740772, -2.18503986, -0.14
254654],
       [ 1.15782128, -3.38051501, -0.2910
0619],
       [ 0.87144798, -6.79971146, -0.4523
1566]])
```


배열 연산: log

In [17]:

`np.log(a)`

Out[17]:

```
array([[ 0.          ,  0.69314718,  1.09
861229],
       [ 1.38629436,  1.60943791,  1.7917
5947],
       [ 1.94591015,  2.07944154,  2.1972
2458]])
```

배열 연산: dot product, 내적

In [18]:

`np.dot(a, b)`

Out[18]:

```
array([[ 30,  24,  18],
       [ 84,  69,  54],
       [138, 114,  90]])
```

비교 연산(Comparison)**배열의 요소별 비교 (Element-wise)**

- 기본 연산자를 이용하여 요소별 비교를 할 수 있습니다.

In [19]:

`a == b`

Out[19]:

```
array([[False, False, False],
       [False,  True, False],
       [False, False, False]], dtype=bool)
```

In [20]:

`a > b`

Out[20]:

```
array([[False, False, False],
       [False, False,  True],
       [ True,  True,  True]], dtype=bool)
```

배열 비교 (Array-wise)

- 두 배열 전체는 np.array_equal 함수를 사용하여 비교합니다.

In [21]:

`np.array_equal(a, b)`

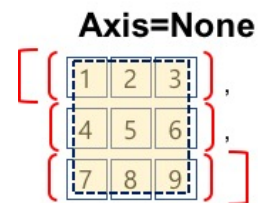
Out[21]:

False

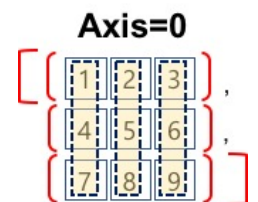
집계 함수(Aggregate Functions)

NumPy의 모든 집계 함수는 집계 함수는 AXIS를 기준으로 계산됩니다. 집계 함수에 AXIS를 지정하지 않으면 axis=None입니다. axis=None, 0, 1은 다음과 같은 기준으로 생각할 수 있습니다.

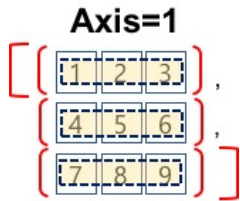
- axis=None
 - axis=None은 전체 행렬을 하나의 배열로 간주하고 집계 함수의 범위를 전체 행렬로 정의합니다.



- axis=0
 - axis=0은 행을 기준으로 각 행의 동일 인덱스의 요소를 그룹으로 합니다.
 - 각 그룹을 집계 함수의 범위로 정의합니다.



- axis=1
 - axis=1은 열을 기준으로 각 열의 요소를 그룹으로 합니다.
 - 각 그룹을 집계 함수의 범위로 정의합니다.



- axis 관련해서는 **NumPy에서 np.sum 함수의 axis 이해** (http://taewan.kim/post/numpy_sum_axis/)를 참조하시기 바랍니다.

In [22]:

```
# arange로 1부터 10미만의 범위에서 1씩 증가하는 배열 생성
# 배열의 shape을 (3, 3)으로 지정
a = np.arange(1, 10).reshape(3, 3)
pprint(a)
```

shape: (3, 3), dimension: 2, dtype:int64

4

Array's Data

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[ndarray 배열 객체].sum(), np.sum(): 합계

지정된 axis를 기준으로 요소의 합을 반환합니다.

In [23]:

```
a.sum(), np.sum(a)
```

Out[23]:

(45, 45)

In [24]:

```
a.sum(axis=0), np.sum(a, axis=0)
```

Out[24]:

```
(array([12, 15, 18]), array([12, 15, 18]))
```

In [25]:

```
a.sum(axis=1), np.sum(a, axis=1)
```

Out[25]:

```
(array([ 6, 15, 24]), array([ 6, 15, 24]))
```

[ndarray 배열 객체].min(), np.min(): 최소값

지정된 axis를 기준으로 요소의 최소값을 반환합니다.

In [26]:

```
a.min(), np.min(a)
```

Out[26]:

(1, 1)

In [27]:

```
a.min(axis=0), np.min(a, axis=0)
```

Out[27]:

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

In [28]:

```
a.min(axis=1), np.min(a, axis=1)
```

Out[28]:

```
(array([1, 4, 7]), array([1, 4, 7]))
```

[ndarray 배열 객체].max(), np.max(): 최대값

지정된 axis를 기준으로 요소의 최대값을 반환합니다.

In [29]:

`a.max(), np.max(a)`

Out[29]:

(9, 9)

In [30]:

`a.max(axis=0), np.max(a, axis=0)`

Out[30]:

(array([7, 8, 9]), array([7, 8, 9]))

In [31]:

`a.max(axis=1), np.max(a, axis=1)`

Out[31]:

(array([3, 6, 9]), array([3, 6, 9]))

[ndarray 배열 객체].cumssum(), np.cumsum(): 누적 합계

지정된 axis를 기준으로 각 요소의 누적 합의 결과를 반환합니다.

In [32]:

`a.cumsum(), np.cumsum(a)`

Out[32]:

(array([1, 3, 6, 10, 15, 21, 28, 36, 45]),
array([1, 3, 6, 10, 15, 21, 28, 36, 45]))

In [33]:

`a.cumsum(axis=0), np.cumsum(a, axis=0)`

Out[33]:

(array([[1, 2, 3],
 [5, 7, 9],
 [12, 15, 18]]), array([[1, 2,
3],
 [5, 7, 9],
 [12, 15, 18]]))

In [34]:

`a.cumsum(axis=1), np.cumsum(a, axis=1)`

Out[34]:

(array([[1, 3, 6],
 [4, 9, 15],
 [7, 15, 24]]), array([[1, 3,
6],
 [4, 9, 15],
 [7, 15, 24]]))**[ndarray 배열 객체].mean(), np.mean(): 평균**

지정된 axis를 기준으로 요소의 평균을 반환합니다.

In [35]:

`a.mean(), np.mean(a)`

Out[35]:

(5.0, 5.0)

In [36]:

`a.mean(axis=0), np.mean(a, axis=0)`

Out[36]:

(array([4., 5., 6.]), array([4.,
5., 6.]))

In [37]:

`a.mean(axis=1), np.mean(a, axis=1)`

Out[37]:

(array([2., 5., 8.]), array([2.,
5., 8.]))**np.mean(): 중앙값**

지정된 axis를 기준으로 요소의 중앙값을 반환합니다.

In [38]:

`np.median(a)`

Out[38]:

5.0

In [39]:

```
np.mean(a, axis=0)
```

Out[39]:

```
array([ 4.,  5.,  6.])
```

In [40]:

```
np.mean(a, axis=1)
```

Out[40]:

```
array([ 2.,  5.,  8.])
```

np.corrcoef(): (상관계수)Correlation coefficient

In [41]:

```
np.corrcoef(a)
```

Out[41]:

```
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

[ndarray 배열 객체].std(), np.std(): 표준편차

지정된 axis를 기준으로 요소의 표준 편차를 계산합니다.

In [42]:

```
a.std(), np.std(a)
```

Out[42]:

```
(2.5819888974716112, 2.5819888974716112)
```

In [43]:

```
a.std(axis=0), np.std(a, axis=0)
```

Out[43]:

```
(array([ 2.44948974,  2.44948974,  2.44948974]),
 array([ 2.44948974,  2.44948974,  2.44948974]))
```

In [44]:

```
a.std(axis=1), np.std(a, axis=1)
```

Out[44]:

```
(array([ 0.81649658,  0.81649658,  0.81649658]),
 array([ 0.81649658,  0.81649658,  0.81649658]))
```

7.2 브로드캐스팅

Shape이 같은 두 배열에 대한 이항 연산은 배열의 요소별로 수행됩니다. 두 배열 간의 Shape이 다를 경우 두 배열 간의 형상을 맞추는 <그림 2>의 Broadcasting 과정을 거칩니다.

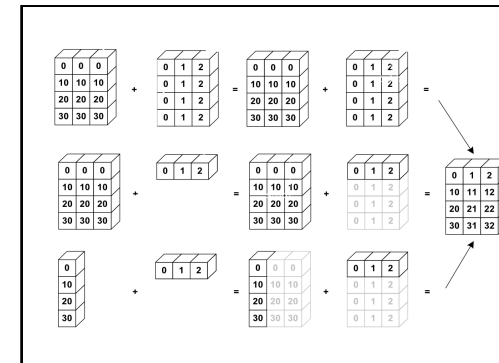


그림 2: 브로드캐스트 작동 원리

- 그림 2 참조:

<https://mathematica.stackexchange.com/questions/99171/how-to-implement-the-general-array-broadcasting-method-from-numpy>
(<https://mathematica.stackexchange.com/questions/99171/how-to-implement-the-general-array-broadcasting-method-from-numpy>)

In [2]:

```
# 데모 배열 생성
a = np.arange(1, 25).reshape(4, 6)
pprint(a)
b = np.arange(25, 49).reshape(4, 6)
pprint(b)

shape: (4, 6), dimension: 2, dtype:int6
4
Array's Data
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
shape: (4, 6), dimension: 2, dtype:int6
4
Array's Data
[[25 26 27 28 29 30]
 [31 32 33 34 35 36]
 [37 38 39 40 41 42]
 [43 44 45 46 47 48]]
```

Shape이 같은 배열의 이항 연산

- Shape이 같은 두 배열을 이항 연산 할 경우 위치가 같은 요소 단위로 수행됩니다.

In [3]:

a+b

Out[3]:

```
array([[26, 28, 30, 32, 34, 36],
       [38, 40, 42, 44, 46, 48],
       [50, 52, 54, 56, 58, 60],
       [62, 64, 66, 68, 70, 72]])
```

Shape이 다른 두 배열의 연산

- Shape이 다른 두 배열 사이의 이항 연산에서 브로드캐스팅 발생
- 두 배열을 같은 Shape으로 만든 후 연산을 수행

Case 1: 배열과 스칼라

배열과 스칼라 사이의 이항 연산 시 스칼라를 배열로 변형합니다.

In [4]:

```
# 데모 배열 생성
a = np.arange(1, 25).reshape(4, 6)
pprint(a)

shape: (4, 6), dimension: 2, dtype:int6
4
Array's Data
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

In [5]:

a+100

Out[5]:

```
array([[101, 102, 103, 104, 105, 106],
       [107, 108, 109, 110, 111, 112],
       [113, 114, 115, 116, 117, 118],
       [119, 120, 121, 122, 123, 124]])
```

- a + 100은 다음과 같은 과정을 거쳐 처리 됩니다.

In [6]:

```
# step 1: 스칼라 배열 변경
new_arr = np.full_like(a, 100)
pprint(new_arr)
```

```
shape: (4, 6), dimension: 2, dtype:int6
4
Array's Data
[[100 100 100 100 100 100]
 [100 100 100 100 100 100]
 [100 100 100 100 100 100]
 [100 100 100 100 100 100]]
```

In [7]:

```
# step 2: 배열 이항 연산
a+new_arr
```

Out[7]:

```
array([[101, 102, 103, 104, 105, 106],
       [107, 108, 109, 110, 111, 112],
       [113, 114, 115, 116, 117, 118],
       [119, 120, 121, 122, 123, 124]])
```

Case 2: Shape이 다른 배열들의 연산

In [8]:

```
# 데모 배열 생성
a = np.arange(5).reshape((1, 5))
pprint(a)
b = np.arange(5).reshape((5, 1))
pprint(b)
```

```
shape: (1, 5), dimension: 2, dtype:int64
4
Array's Data
[[0 1 2 3 4]]
shape: (5, 1), dimension: 2, dtype:int64
4
Array's Data
[[0]
 [1]
 [2]
 [3]
 [4]]
```

In [9]:

a+b

Out[9]:

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

7.3 벡터연산

NumPy는 벡터 연산을 지원합니다. NumPy의 집합 연산에는 Vectorization이 적용되어 있습니다. 배열 처리에 대한 벡터 연산을 적용할 경우 처리 속도가 100배 이상 향상됩니다. 머신러닝에서 선형대수 연산을 처리할 때 매우 높은 효과가 있을 수 있습니다.

아래 테스트에서 천만 개의 요소를 갖는 배열의 요소를 합산하는 코드를 반복문으로 처리하면 2.5초가 소요됩니다. 이 연산을 벡터 연산으로 처리하면 10밀리세컨드로 단축됩니다.

In [1]:

```
import numpy as np
# sample array
a = np.arange(10000000)
```

- loop 문을 이용한 연산: 처리 2.7초

In [2]:

result = 0

In [3]:

```
%%time
for v in a:
    result += v
```

```
CPU times: user 2.57 s, sys: 0 ns, tota
l: 2.57 s
Wall time: 2.57 s
```

In [4]:

result

Out[4]:

49999995000000

- numpy의 벡터연산: 10 ms

In [5]:

```
%%time
result = np.sum(a)
```

```
CPU times: user 10 ms, sys: 0 ns, tota
l: 10 ms
Wall time: 16.3 ms
```

In [6]:

result

Out[6]:

49999995000000

8. 배열 복사

ndarray 배열 객체에 대한 slice, subset, indexing이 반환하는 배열은 새로운 객체가 아닌 기존 배열의 뷰입니다. 반환한 배열의 값을 변경하면 원본 배열에 반영됩니다. 따라서 기본 배열로부터 새로운 배열을 생성하기 위해서는 copy 함수로 명시적으로 사용해야 합니다. copy 함수로 복사된 원본 배열과 사본 배열은 완전히 다른 별도의 객체입니다.

[ndarray 배열 객체].copy(), np.copy()

In [2]:

```
# 대모용 배열
a = np.random.randint(0, 9, (3, 3))
pprint(a)
```

```
shape: (3, 3), dimension: 2, dtype:int6
4
Array's Data
[[1 8 2]
 [3 1 7]
 [2 8 4]]
```

In [3]:

```
copied_a1 = np.copy(a)
```

In [4]:

```
# 복사된 배열의 요소 업데이트
copied_a1[:, 0]=0 # 배열의 전체 row의 첫번째 요소 0으로
업데이트
pprint(copied_a1)
```

```
shape: (3, 3), dimension: 2, dtype:int6
4
Array's Data
[[0 8 2]
 [0 1 7]
 [0 8 4]]
```

In [5]:

```
# 복사본 배열 변경이 원본에 영향을 미치지 않음
pprint(a)
```

```
shape: (3, 3), dimension: 2, dtype:int6
4
Array's Data
[[1 8 2]
 [3 1 7]
 [2 8 4]]
```

In [6]:

```
# np.copy()를 이용한 복사
copied_a2 = np.copy(a)
pprint(copied_a2)
```

```
shape: (3, 3), dimension: 2, dtype:int6
4
Array's Data
[[1 8 2]
 [3 1 7]
 [2 8 4]]
```

9. 배열 정렬

ndarray 객체는 axis를 기준으로 요소 정렬하는 sort 함수를 제공합니다.

In [2]:

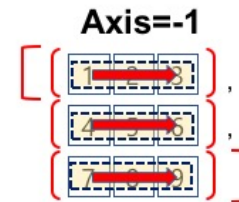
```
# 배열 생성
unsorted_arr = np.random.random((3, 3))
pprint(unsorted_arr)
```

```
shape: (3, 3), dimension: 2, dtype:float64
Array's Data
[[ 0.2850756  0.39471278  0.62599575]
 [ 0.36466771  0.2226558  0.78950711]
 [ 0.16414313  0.95836532  0.43830591]]
```

In [3]:

```
# 대모를 위한 배열 복사
unsorted_arr1 = unsorted_arr.copy()
unsorted_arr2 = unsorted_arr.copy()
unsorted_arr3 = unsorted_arr.copy()
```

- [ndarray 객체].sort() axis의 기본 값은 -1입니다.
- axis=-1은 현재 배열의 마지막 axis를 의미합니다.
- 현재 unsorted_arr의 마지막 axis는 1입니다.
- [ndarray 객체].sort()와 [ndarray 객체].sort(axis=1)의 결과는 동일합니다.

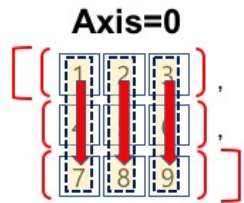


In [4]:

```
# 배열 정렬
unsorted_arr1.sort()
pprint(unsorted_arr1)
```

```
shape: (3, 3), dimension: 2, dtype:float64
Array's Data
[[ 0.2850756  0.39471278  0.62599575]
 [ 0.2226558  0.36466771  0.78950711]
 [ 0.16414313  0.43830591  0.95836532]]
```

- axis=0의 정렬 방향은 다음 그림과 같습니다.

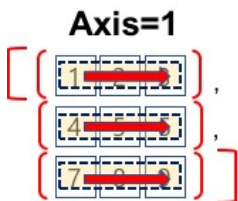


In [5]:

```
#배열 정렬, axis=0
unsorted_arr2.sort(axis=0)
pprint(unsorted_arr2)
```

```
shape: (3, 3), dimension: 2, dtype:float64
Array's Data
[[ 0.16414313  0.2226558  0.43830591]
 [ 0.2850756  0.39471278  0.62599575]
 [ 0.36466771  0.95836532  0.78950711]]
```

- axis=1의 정렬 방향은 다음 그림과 같습니다.



In [6]:

```
#배열 정렬, axis=1
unsorted_arr3.sort(axis=1)
pprint(unsorted_arr3)
```

```
shape: (3, 3), dimension: 2, dtype:float64
Array's Data
[[ 0.2850756  0.39471278  0.62599575]
 [ 0.2226558  0.36466771  0.78950711]
 [ 0.16414313  0.43830591  0.95836532]]
```

10. 서브셋, 슬라이싱, 인덱싱

배열의 개별 요소, 요소 집합을 참조하는 방법을 소개합니다.

10.1 요소 선택

배열의 각 요소는 axis 인덱스 배열로 참조할 수 있습니다. 1차원 배열은 1개 인덱스, 2차원 배열은 2개 인덱스, 3차원 인덱스는 3개 인덱스로 요소를 참조할 수 있습니다.

인덱스로 참조한 요소는 값 참조, 값 수정이 모두 가능합니다.

In [2]:

```
# 데모 배열 생성
a0 = np.arange(24) # 1차원 배열
pprint(a0)
a1 = np.arange(24).reshape((4, 6)) #2차원 배열
pprint(a1)
a2 = np.arange(24).reshape((2, 4, 3)) # 3차원 배열
pprint(a2)
```

```
shape: (24,), dimension: 1, dtype:int64
Array's Data
[ 0  1  2  3  4  5  6  7  8  9 10 11 12
 13 14 15 16 17 18 19 20 21 22 23]
shape: (4, 6), dimension: 2, dtype:int64
4
Array's Data
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]]
shape: (2, 4, 3), dimension: 3, dtype:int64
Array's Data
[[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

 [[12 13 14]
 [15 16 17]
 [18 19 20]
 [21 22 23]]]
```

1차원 배열 요소 참조 및 변경

In [3]:

```
a0[5] # 5번 인덱스 요소 참조
```

Out[3]:

5

In [4]:

```
# 5번 인덱스 요소 업데이트
a0[5] = 1000000
```


In [5]:

```
pprint(a0)
```

```
shape: (24,), dimension: 1, dtype:int64
Array's Data
[  0      1      2      3
 4 100000      6      7      8
   9     10     11     12     13
14     15     16     17
   18     19     20     21     22
23]
```

2 차원 배열 요소 참조 및 변경

In [6]:

```
pprint(a1)
```

```
shape: (4, 6), dimension: 2, dtype:int64
4
Array's Data
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]]
```

In [7]:

```
# 1행 두번째 컬럼 요소 참조
a1[0, 1]
```

Out[7]:

1

In [8]:

```
# 1행 두번째 컬럼 요소 업데이트
a1[0, 1]=10000
```

In [9]:

```
pprint(a1)
```

```
shape: (4, 6), dimension: 2, dtype:int64
4
Array's Data
[[  0 10000      2      3      4      5]
 [  6   7   8   9  10  11]
 [ 12  13  14  15  16  17]
 [ 18  19  20  21  22  23]]
```

3 차원 배열 요소 참조 및 변경

In [10]:

```
pprint(a2)
```

```
shape: (2, 4, 3), dimension: 3, dtype:int64
Array's Data
[[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

 [[12 13 14]
 [15 16 17]
 [18 19 20]
 [21 22 23]]]
```

In [11]:

```
# 2 번째 행, 첫번째 컬럼, 두번째 요소 참조
a2[1, 0, 1]
```

Out[11]:

13

In [12]:

```
a2[1, 0, 1]=10000
```

In [13]:

```
pprint(a2)
```

```
shape: (2, 4, 3), dimension: 3, dtype:int64
Array's Data
[[[  0      1      2]
 [  3      4      5]
 [  6      7      8]
 [  9     10     11]]

 [[ 12 10000     14]
 [ 15     16     17]
 [ 18     19     20]
 [ 21     22     23]]]
```

10.2 슬라이싱(Slicing)

여러개의 배열 요소를 참조할 때 슬라이싱을 사용합니다. 슬라이싱은 axis 별로 범위를 지정하여 실행합니다. 범위는 *from_iindex* : *to_iindex* 형태로 지정합니다. from_index는 범위의 시작 인덱스이며, to_index는 범위의 종료 인덱스입니다. 요소 범위를 지정할 때 to_index는 결과에 포함되지 않습니다. *from_iindex* : *to_iindex*의 범위 지정에서 from_index는 생략 가능합니다. 생략할 경우 0을 지정한 것으로 간주합니다. to_index 역시 생략 가능합니다. 이 경우 마지막 인덱스로 설정됩니다. 따라서 ":" 형태로 지정된 범위는 전체 범위를 의미합니다.

from_index와 to_index에 음수를 지정하면 이것은 반대 방향을 의미합니다. 예를 들어서 -1은 마지막 인덱스를 의미합니다.

슬라이싱은 원본 배열의 뷰입니다. 따라서 슬라이싱 결과의 요소를 업데이트하면 원본에 반영됩니다.

In [1]:

```
# 데모 배열 생성
a1 = np.arange(1, 25).reshape((4, 6)) #2차원 배열
pprint(a1)
```

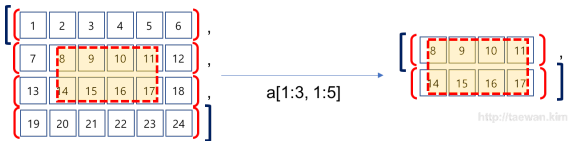
shape: (4, 6), dimension: 2, dtype:int64

4

Array's Data

```
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

가운데 요소 가져오기



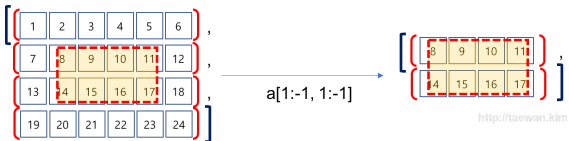
In [2]:

```
a1[1:3, 1:5]
```

Out[2]:

```
array([[ 8,  9, 10, 11],
       [14, 15, 16, 17]])
```

- 음수 인덱스를 이용한 범위 설정
 - 음수 인덱스는 지정한 axis의 마지막 요소로부터 반대 방향의 인덱스입니다.
 - -1은 마지막 요소의 인덱스를 의미합니다.
 - 다음 슬라이싱은 위 슬라이싱과 동일한 결과를 만듭니다.



In [3]:

```
a1[1:-1, 1:-1]
```

Out[3]:

```
array([[ 8,  9, 10, 11],
       [14, 15, 16, 17]])
```

슬라이싱 업데이트

In [4]:

```
# 데모 대상 배열 조회
pprint(a1)
```

shape: (4, 6), dimension: 2, dtype:int64

4

Array's Data

```
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

In [5]:

```
# 슬라이싱 배열
slide_arr = a1[1:3, 1:5]
pprint(slide_arr)
```

shape: (2, 4), dimension: 2, dtype:int64

4

Array's Data

```
[[ 8  9 10 11]
 [14 15 16 17]]
```

In [6]:

```
# 슬라이싱 결과 배열에 슬라이싱을 적용하여 4개 요소 참조
slide_arr[:, 1:3]
pprint(slide_arr)
```

shape: (2, 4), dimension: 2, dtype:int64

4

Array's Data

```
[[ 8  9 10 11]
 [14 15 16 17]]
```

In [7]:

```
# 슬라이싱을 적용하여 참조한 4개 요소 업데이트 및 슬라이싱 배열 조회
slide_arr[:, 1:3]=99999
pprint(slide_arr)
```

shape: (2, 4), dimension: 2, dtype:int64

4

Array's Data

```
[[ 8 99999 99999 11]
 [14 99999 99999 17]]
```

```
In [8]:  
  
# 원본 배열에 반영된 결과 확인  
pprint(a1)  
  
shape: (4, 6), dimension: 2, dtype:int64  
4  
Array's Data  
[[ 1 2 3 4 5 6]  
 [ 7 8 99999 99999 11 12]  
 [ 13 14 99999 99999 17 18]  
 [ 19 20 21 22 23 24]]
```

10.3 불린 인덱싱(Boolean Indexing)

NumPy는 불린 인덱싱은 배열 각 요소의 선택 여부를 True, False 지정 하는 방식입니다. 해당 인덱스의 True만을 조회합니다.

```
In [2]:  
  
# 데모 배열 생성  
a1 = np.arange(1, 25).reshape((4, 6)) #2차원 배열  
pprint(a1)  
  
type:<class 'numpy.ndarray'>  
shape: (4, 6), dimension: 2, dtype:int64  
4  
Array's Data:  
[[ 1 2 3 4 5 6]  
 [ 7 8 9 10 11 12]  
 [13 14 15 16 17 18]  
 [19 20 21 22 23 24]]
```

- a1 배열에서 요소의 값이 짝수인 요소들의 총 합은?

```
In [3]:  
  
# 짝수인 요소 확인  
# numpy broadcasting을 이용하여 짝수인 배열 요소 확인  
even_arr = a1%2==0  
pprint(even_arr)  
  
type:<class 'numpy.ndarray'>  
shape: (4, 6), dimension: 2, dtype:bool  
Array's Data:  
[[False True False True False True]  
 [False True False True False True]  
 [False True False True False True]  
 [False True False True False True]]
```

```
In [4]:  
  
# a1[a1%2==0] 동일한 의미입니다.  
a1[even_arr]  
  
Out[4]:  
  
array([ 2, 4, 6, 8, 10, 12, 14, 16,  
 18, 20, 22, 24])
```

```
In [5]:  
  
np.sum(a1)
```

```
Out[5]:  
  
300
```

Boolean Indexing의 응용

- 2014년 시애크 강수량 데이터: ./data/seattle2014.csv
- 2014년 시애크를 1월 평균 강수량은?

```
In [6]:  
  
!head -n 3 ./data/seattle2014.csv  
  
STATION,STATION_NAME,DATE,PRCP,SNWD,SNO  
W,TMAX,TMIN,AWND,WDF2,WDF5,WSF2,WSF5,WT  
01,WT05,WT02,WT03  
GHCND:USW00024233,SEATTLE TACOMA INTERN  
ATIONAL AIRPORT WA US,20140101,0,0,0,7  
2,33,12,340,310,36,40,-9999,-9999,-999  
9,-9999  
GHCND:USW00024233,SEATTLE TACOMA INTERN  
ATIONAL AIRPORT WA US,20140102,41,0,0,1  
06,61,32,190,200,94,116,-9999,-9999,-99  
99,-9999
```

```
In [7]:  
  
# 데이터 로딩  
import pandas as pd  
rains_in_seattle = pd.read_csv("./data/seattle2014.c  
sv")  
rains_arr = rains_in_seattle['PRCP'].values  
print("Data Size:", len(rains_arr))
```

Data Size: 365

```
In [8]:  
  
# 날짜 배열  
days_arr = np.arange(0, 365)
```


Tranpose는 행렬의 인덱스가 바뀌는 변환입니다. 행렬의 Tranpose의 예는 다음과 같습니다.

$$\begin{bmatrix} 1 & 2 \end{bmatrix}^T = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

- 다음은 NumPy에서 행렬을 전치하기 위하여 [numpy.ndarray 객체].T 속성을 사용합니다.
- 이 속성은 전치된 행렬을 반환합니다.

In [1]:

```
# 행렬 생성
a = np.random.randint(1, 10, (2, 3))
pprint(a)
```

```
shape: (2, 3), dimension: 2, dtype:int64
4
Array's Data
[[6 6 1]
 [4 1 3]]
```

In [2]:

```
#행렬의 전치
pprint(a.T)
```

```
shape: (3, 2), dimension: 2, dtype:int64
4
Array's Data
[[6 4]
 [6 1]
 [1 3]]
```

11.2 배열 형태 변경

[numpy.ndarray 객체]는 배열의 Shape을 변경하는 ravel 메서드와 reshape 메서드를 제공합니다.

- ravel은 배열의 shape을 1차원 배열로 만드는 메서드입니다.
- reshape은 데이터 변경없이 지정된 shape으로 변환하는 메서드입니다.

[numpy.ndarray 객체].ravel()

배열을 1차원 배열로 반환하는 메서드입니다. numpy.ndarray 배열 객체의 View를 반환합니다. ravel 메서드가 반환하는 배열은 원본 배열이 뷰입니다. 따라서 ravel 메서드가 반환하는 배열의 요소를 수정하면 원본 배열 요소에도 반영됩니다.

In [1]:

```
# 데모 배열 생성
a = np.random.randint(1, 10, (2, 3))
pprint(a)
```

```
shape: (2, 3), dimension: 2, dtype:int64
4
Array's Data
[[7 9 6]
 [2 1 1]]
```

In [2]:

```
a.ravel()
```

Out[2]:

```
array([7, 9, 6, 2, 1, 1])
```

- ravel이 반환하는 배열은 a 행렬의 view입니다.
- 반환 행렬의 데이터를 변경하면 a 행렬도 변경됩니다.

In [3]:

```
b = a.ravel()
pprint(b)
```

```
shape: (6,), dimension: 1, dtype:int64
Array's Data
[7 9 6 2 1 1]
```

In [4]:

```
b[0]=99
pprint(b)
```

```
shape: (6,), dimension: 1, dtype:int64
Array's Data
[99 9 6 2 1 1]
```

In [5]:

```
# b 배열 변경이 a 행렬에 반영되어 있습니다.
pprint(a)
```

```
shape: (2, 3), dimension: 2, dtype:int64
4
Array's Data
[[99 9 6]
 [ 2 1 1]]
```

[numpy.ndarray 객체].reshape()

[numpy.ndarray 객체]의 shape을 변경합니다. 실제 데이터를 변경하는 것은 아닙니다. 배열 객체의 shape 정보만을 수정합니다.

```
In [1]:  
  
# 대상 행렬 속성 확인  
a = np.random.randint(1, 10, (2, 3))  
pprint(a)  
  
shape: (2, 3), dimension: 2, dtype:int64  
4  
Array's Data  
[[7 7 8]  
 [3 9 1]]  
  
In [2]:  
  
result = a.reshape((3, 2, 1))  
pprint(result)  
  
shape: (3, 2, 1), dimension: 3, dtype:in  
nt64  
Array's Data  
[[[7]  
 [7]]  
  
 [[8]  
 [3]]  
  
 [[9]  
 [1]]]
```

11.3 배열 요소 추가 삭제

배열의 요소를 변경, 추가, 삽입 및 삭제하는 resize, append, insert, delete 함수를 제공합니다.

resize()

- np.resize(a, new_shape)
- np.ndarray.resize(new_shape, refcheck=True)
- 배열의 shape과 크기를 변경합니다.

np.resize와 np.reshape 함수는 배열의 shape을 변경한다는 부분에서 유사합니다. 차이점은 reshape 함수는 배열 요소 수를 변경하지 않습니다. reshape 전후 배열의 요소 수는 같습니다. 반면에 resize는 shape을 변경하는 과정에서 배열 요소 수를 줄이거나 늘립니다.

- 일반적인 resize 사용 방법

```
In [2]:  
  
# 배열 생성  
a = np.random.randint(1, 10, (2, 6))  
pprint(a)  
  
type:<class 'numpy.ndarray'>  
shape: (2, 6), dimension: 2, dtype:int64  
4  
Array's Data:  
[[1 5 4 2 7 4]  
 [4 8 4 4 9 9]]  
  
In [3]:  
  
# shape 변경 - 요소 수 변경 없음  
a.resize((6, 2))  
pprint(a)  
  
type:<class 'numpy.ndarray'>  
shape: (6, 2), dimension: 2, dtype:int64  
4  
Array's Data:  
[[1 5]  
 [4 2]  
 [7 4]  
 [4 8]  
 [4 4]  
 [9 9]]
```

- 요소 수가 늘어난 변경

In [4]:

```
# 배열 생성
a = np.random.randint(1, 10, (2, 6))
pprint(a)

type:<class 'numpy.ndarray'>
shape: (2, 6), dimension: 2, dtype:int64
Array's Data:
[[5 1 8 4 5 3]
 [2 8 9 2 2 6]]
```

In [5]:

```
# 요소속 12개에서 20개로 늘어남
# 늘어난 요소는 0으로 채워짐
a.resize((2, 10))
pprint(a)

type:<class 'numpy.ndarray'>
shape: (2, 10), dimension: 2, dtype:int64
Array's Data:
[[5 1 8 4 5 3 2 8 9 2]
 [2 6 0 0 0 0 0 0 0 0]]
```

- 요소 수가 늘어난 변경

In [6]:

```
# 배열 생성
a = np.random.randint(1, 10, (2, 6))
pprint(a)

type:<class 'numpy.ndarray'>
shape: (2, 6), dimension: 2, dtype:int64
Array's Data:
[[7 4 5 9 8 6]
 [9 7 5 1 3 1]]
```

In [7]:

```
# 요소속 12개에서 9개로 줄임
# 이전 데이터 삭제
a.resize((3, 3))
pprint(a)

type:<class 'numpy.ndarray'>
shape: (3, 3), dimension: 2, dtype:int64
Array's Data:
[[7 4 5]
 [9 8 6]
 [9 7 5]]
```

append()

- np.append(arr, values, axis=None)
- 배열의 끝에 값을 추가

np.append 함수는 arr의 끝에 values(배열)을 추가합니다. axis로 배열이 추가되는 방향을 지정할 수 있습니다.

In [8]:

```
# 데모 배열 생성
a = np.arange(1, 10).reshape(3, 3)
pprint(a)
b = np.arange(10, 19).reshape(3, 3)
pprint(b)

type:<class 'numpy.ndarray'>
shape: (3, 3), dimension: 2, dtype:int64
Array's Data:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
type:<class 'numpy.ndarray'>
shape: (3, 3), dimension: 2, dtype:int64
Array's Data:
[[10 11 12]
 [13 14 15]
 [16 17 18]]
```

case 1: axis을 지정하지 않을 경우

axis를 지정하지 않으면 배열은 1차원 배열로 변형되어 결합됩니다.

In [9]:

```
# axis 지정 없이 추가
result = np.append(a, b)
pprint(result)

type:<class 'numpy.ndarray'>
shape: (18,), dimension: 1, dtype:int64
Array's Data:
[ 1  2  3  4  5  6  7  8  9 10 11 12 13
 14 15 16 17 18]
```

In [10]:

```
# 원본 배열을 변경하는 것이 아니며 새로운 배열이 생성됩니다.
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (3, 3), dimension: 2, dtype:int64
Array's Data:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

case 2: axis=0 지정

In [11]:

```
# axis = 0
# axis 0 방향으로 b 배열 추가
result = np.append(a, b, axis=0)
pprint(result)
```

```
type:<class 'numpy.ndarray'>
shape: (6, 3), dimension: 2, dtype:int64
Array's Data:
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]
 [13 14 15]
 [16 17 18]]
```

- axis = 0 설정 시, shape[0]를 제외한 나머지 shape은 같아야 합니다.
- shape[0]를 제외한 나머지 shape이 다를 경우 append는 오류를 발생합니다.

In [12]:

```
different_sahpe_arr = np.arange(10, 20).reshape(2, 5)
pprint(different_sahpe_arr)
```

```
type:<class 'numpy.ndarray'>
shape: (2, 5), dimension: 2, dtype:int64
Array's Data:
[[10 11 12 13 14]
 [15 16 17 18 19]]
```

In [13]:

```
# 기존 축을 제외한 shape이 다른 배열의 append: 오류 발생
np.append(a, different_sahpe_arr, axis=0)
```

```
ValueErrorTraceback (most recent call last)
<ipython-input-13-62a27e99a457> in <module>()
      1 # 기존 축을 제외한 shape이 다른
      배열의 append: 오류 발생
----> 2 np.append(a, different_sahpe_arr, axis=0)
/usr/local/lib/python3.5/dist-packages/numpy/lib/function_base.py in append(arr, values, axis)
      5150         values = ravel(values)
      5151         axis = arr.ndim-1
-> 5152         return concatenate((arr, values), axis=axis)
ValueError: all the input array dimensions except for the concatenation axis must match exactly
```

case 3: axis=1 지정

- axis = 1 설정 시, shape[1]을 제외한 나머지 shape은 같아야 합니다.
- shape[1]를 제외한 나머지 shape이 다를 경우 append는 오류를 발생합니다.

In [14]:

```
# axis = 1
# axis 1 방향으로 b 배열 추가
result = np.append(a, b, axis=1)
pprint(result)
```

```
type:<class 'numpy.ndarray'>
shape: (3, 6), dimension: 2, dtype:int64
Array's Data:
[[ 1  2  3 10 11 12]
 [ 4  5  6 13 14 15]
 [ 7  8  9 16 17 18]]
```

- axis = 1 설정 시, shape[1]를 제외한 나머지 shape은 같아야 합니다.
- shape[1]를 제외한 나머지 shape이 다를 경우 append는 오류를 발생합니다.

In [15]:

```
# shape이 다른 배열 생성
different_sahpe_arr = np.arange(10, 20).reshape(5, 2)
)
pprint(different_sahpe_arr)
```

```
type:<class 'numpy.ndarray'>
shape: (5, 2), dimension: 2, dtype:int64
Array's Data:
[[10 11]
 [12 13]
 [14 15]
 [16 17]
 [18 19]]
```

In [16]:

```
# 기준 축을 제외한 shape이 다른 배열의 append: 오류 발생
np.append(a, different_sahpe_arr, axis=1)
```

```
ValueErrorTraceback (most recent call last)
<ipython-input-16-c3bfec5cee97> in <module>()
      1 # 기준 축을 제외한 shape이 다른 배열의 append: 오류 발생
----> 2 np.append(a, different_sahpe_arr, axis=1)
/usr/local/lib/python3.5/dist-packages/numpy/lib/function_base.py in append(arr, values, axis)
      5150         values = ravel(values)
      5151         axis = arr.ndim-1
-> 5152         return concatenate((arr, values), axis=axis)
ValueError: all the input array dimensions except for the concatenation axis must match exactly
```

insert()

- np.insert(arr, obj, values, axis=None)
- axis를 지정하지 않으며 1차원 배열로 변환
- 추가할 방향을 axis로 지정

In [17]:

```
# 데모 배열 생성
a = np.arange(1, 10).reshape(3, 3)
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (3, 3), dimension: 2, dtype:int64
Array's Data:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [18]:

```
# a 배열을 일차원 배열로 변환하고 1번 index에 99 추가
np.insert(a, 1, 999)
```

Out[18]:

```
array([[ 1, 999,  2,  3,  4,  5,
        6,  7,  8,  9]])
```

In [19]:

```
# a 배열의 axis 0 방향 1번 인덱스에 추가
# index가 1인 row에 999가 추가됨
np.insert(a, 1, 999, axis=0)
```

Out[19]:

```
array([[ 1,  2,  3],
       [999, 999, 999],
       [ 4,  5,  6],
       [ 7,  8,  9]])
```

In [20]:

```
# a 배열의 axis 1 방향 1번 인덱스에 추가
# index가 1인 column에 999가 추가됨
np.insert(a, 1, 999, axis=1)
```

Out[20]:

```
array([[ 1, 999,  2,  3],
       [ 4, 999,  5,  6],
       [ 7, 999,  8,  9]])
```

delete()

- np.delete(arr, obj, axis=None)
- axis를 지정하지 않으며 1차원 배열로 변환
- 삭제할 방향을 axis로 지정
- delete 함수는 원본 배열을 변경하지 않으며 새로운 배열을 반환

In [21]:

```
# 데모 배열 생성
a = np.arange(1, 10).reshape(3, 3)
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (3, 3), dimension: 2, dtype:int64
Array's Data:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [22]:

```
# a 배열을 일차원 배열로 변환하고 1번 index 삭제
np.delete(a, 1)
```

Out[22]:

```
array([1, 3, 4, 5, 6, 7, 8, 9])
```

In [23]:

```
# a 배열의 axis 0 방향 1번 인덱스인 행을 삭제한 배열을
# 생성하여 반환
np.delete(a, 1, axis=0)
```

Out[23]:

```
array([[1, 2, 3],
       [7, 8, 9]])
```

In [24]:

```
# a 배열의 axis 1 방향 1번 인덱스인 열을 삭제한 배열을
# 생성하여 반환
np.delete(a, 1, axis=1)
```

Out[24]:

```
array([[1, 3],
       [4, 6],
       [7, 9]])
```

11.4 배열 결합

배열과 배열을 결합하는 np.concatenate, np.vstack, np.hstack 함수를 제공합니다.

배열 결합

np.concatenate

- concatenate((a1, a2, ...), axis=0)
- a1, a2....: 배열

In [2]:

```
# 데모 배열
a = np.arange(1, 7).reshape((2, 3))
pprint(a)
b = np.arange(7, 13).reshape((2, 3))
pprint(b)
```

```
shape: (2, 3), dimension: 2, dtype:int64
4
Array's Data
[[1 2 3]
 [4 5 6]]
shape: (2, 3), dimension: 2, dtype:int64
4
Array's Data
[[ 7  8  9]
 [10 11 12]]
```

In [3]:

```
# axis=0 방향으로 두 배열 결합, axis 기본값=0
result = np.concatenate((a, b))
result
```

Out[3]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [4]:

```
# axis=0 방향으로 두 배열 결합, 결과 동일
result = np.concatenate((a, b), axis=0)
result
```

Out[4]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [5]:

```
# axis=1 방향으로 두 배열 결합, 결과 동일
result = np.concatenate((a, b), axis=1)
result
```

Out[5]:

```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

수직 방향 배열 결합

np.vstack

- np.vstack(tup)
 - tup: 튜플
- 튜플로 설정된 여러 배열을 수직 방향으로 연결 (axis=0 방향)
- np.concatenate(tup, axis=0)와 동일

In [6]:

```
# 데모 배열
a = np.arange(1, 7).reshape((2, 3))
pprint(a)
b = np.arange(7, 13).reshape((2, 3))
pprint(b)
```

```
shape: (2, 3), dimension: 2, dtype:int6
4
Array's Data
[[1 2 3]
 [4 5 6]]
shape: (2, 3), dimension: 2, dtype:int6
4
Array's Data
[[ 7  8  9]
 [10 11 12]]
```

In [7]:

```
np.vstack((a, b))
```

Out[7]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [8]:

```
# 4개 배열을 튜플로 설정
np.vstack((a, b, a, b))
```

Out[8]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12],
       [ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

수평 방향 배열 결합

np.hstack

- np.hstack(tup)
 - tup: 튜플
- 튜플로 설정된 여러 배열을 수평 방향으로 연결 (axis=1 방향)
- np.concatenate(tup, axis=1)와 동일

In [9]:

```
# 데모 배열
a = np.arange(1, 7).reshape((2, 3))
pprint(a)
b = np.arange(7, 13).reshape((2, 3))
pprint(b)
```

```
shape: (2, 3), dimension: 2, dtype:int6
4
Array's Data
[[1 2 3]
 [4 5 6]]
shape: (2, 3), dimension: 2, dtype:int6
4
Array's Data
[[ 7  8  9]
 [10 11 12]]
```

In [10]:

```
np.hstack((a, b))
```

Out[10]:

```
array([[ 1,  2,  3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12]])
```

In [11]:

```
np.hstack((a, b, a, b))
```

Out[11]:

```
array([[ 1,  2,  3,  7,  8,  9,  1,  2,
        3,  7,  8,  9],
       [ 4,  5,  6, 10, 11, 12,  4,  5,
        6, 10, 11, 12]])
```

11.5 배열 분리

NumPy는 배열을 수직, 수평으로 분할하는 함수를 제공합니다.

- `np.hsplit()`: 지정한 배열을 수평(행) 방향으로 분할
- `np.vsplit()`: 지정한 배열을 수직(열) 방향으로 분할

배열 수평 분할

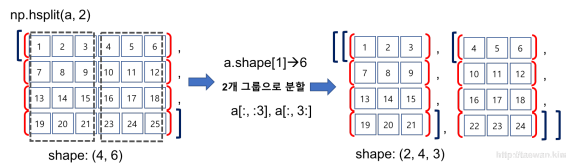
- `np.hsplit(ary, indices_or_sections)`
- 배열을 수평 방향(컬럼 방향)으로 분할하는 함수

In [2]:

```
# 분할 대상 배열 생성
a = np.arange(1, 25).reshape((4, 6))
pprint(a)
```

```
shape: (4, 6), dimension: 2, dtype:int64
4
Array's Data
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

수평 방향으로 배열을 두 그룹으로 분할



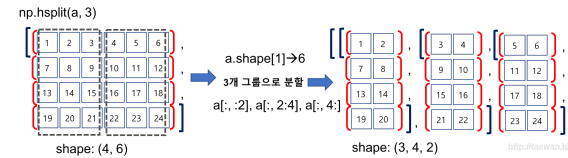
In [3]:

```
# 수평으로 두 그룹으로 분할하는 함수
result = np.hsplit(a, 2)
result
```

Out[3]:

```
[array([[ 1,  2,  3],
        [ 7,  8,  9],
        [13, 14, 15],
        [19, 20, 21]]), array([[ 4,  5,  6],
        [10, 11, 12],
        [16, 17, 18],
        [22, 23, 24]])]
```

수평 방향으로 배열을 세 그룹으로 분할



In [4]:

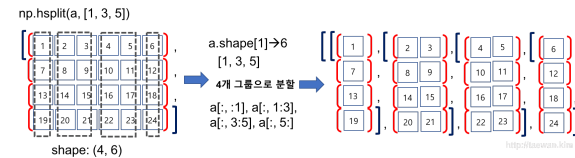
```
# 수평으로 두 그룹으로 분할하는 함수
result = np.hsplit(a, 3)
result
```

Out[4]:

```
[array([[ 1,  2],
        [ 7,  8],
        [13, 14],
        [19, 20]]), array([[ 3,  4],
        [ 9, 10],
        [15, 16],
        [21, 22]]), array([[ 5,  6],
        [11, 12],
        [17, 18],
        [23, 24]])]
```

수평 방향으로 여러 구간으로 구분

- `np.hsplit`의 두 번째 파라미터에 구간 설정 배열을 전달하여 여러 배열로 구분합니다.



In [5]:

```
np.hsplit(a, [1, 3, 5])
```

Out[5]:

```
[array([[ 1],
       [ 7],
       [13],
       [19]]), array([[ 2,  3],
       [ 8,  9],
       [14, 15],
       [20, 21]]), array([[ 4,  5],
       [10, 11],
       [16, 17],
       [22, 23]]), array([[ 6],
       [12],
       [18],
       [24]])]
```

배열 수직 분할

- `np.vsplit(ary, indices_or_sections)`
- 배열을 수직 방향(행 방향)으로 분할하는 함수

In [6]:

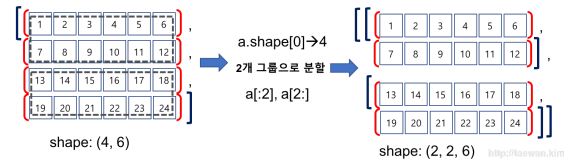
```
# 분할 대상 배열 생성
a = np.arange(1, 25).reshape((4, 6))
pprint(a)
```

```
shape: (4, 6), dimension: 2, dtype:int64
4
```

```
Array's Data
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

수직 방향으로 배열을 두 개 그룹으로 분할

np.vsplit(a, 2)



In [7]:

```
result=np.vsplit(a, 2)
result
```

Out[7]:

```
[array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]]), array
([[13, 14, 15, 16, 17, 18],
       [19, 20, 21, 22, 23, 24]])]
```

In [8]:

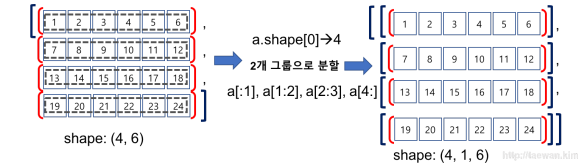
```
np.array(result).shape
```

Out[8]:

(2, 2, 6)

수직 방향으로 배열을 4 개 그룹으로 분할

np.vsplit(a, 4)



In [9]:

```
result=np.vsplit(a, 4)
result
```

Out[9]:

```
[array([[1, 2, 3, 4, 5, 6]]),
 array([[ 7,  8,  9, 10, 11, 12]]),
 array([[13, 14, 15, 16, 17, 18]]),
 array([[19, 20, 21, 22, 23, 24]])]
```

In [10]:

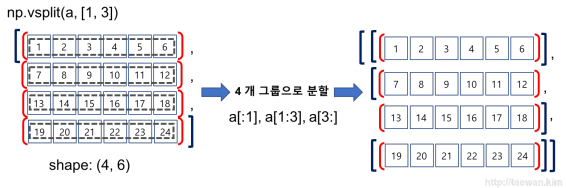
```
np.array(result).shape
```

Out[10]:

(4, 1, 6)

수직 방향으로 여러 구간으로 구분

- np.hsplit의 두 번째 파라미터에 구간 설정 배열을 전달하여 여러 배열로 구분합니다.



© 2020 taewan.kim 블로그.

In [11]:

```
# row를 1, 2-3, 4번째 라인으로 구분
np.vsplit(a, [1, 3])
```

Out[11]:

```
[array([[1, 2, 3, 4, 5, 6]]), array([[
7,  8,  9, 10, 11, 12],
      [13, 14, 15, 16, 17, 18]]), array
([[19, 20, 21, 22, 23, 24]])]
```

12. 참고자료

- Python Numpy Tutorial - <http://cs231n.github.io/> (<http://cs231n.github.io/python-numpy-tutorial/>)
- Python For Data Science Cheat Sheet NumPy Basics by DataCamp (https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf)
- docs.scipy.org: Quickstart tutorial (<https://docs.scipy.org/doc/numpy-dev/user/quickstart.html>)

1. Numpy는 /NUM-pee/ 혹은 /NUM-pai/ 로 발음합니다.
<https://www.youtube.com/watch?v=Uqd7pgpO2-g>
(<https://www.youtube.com/watch?v=Uqd7pgpO2-g>)
[return]

[NUMPY \(/TAGS/NUMPY/\)](#) [PYTHON \(/TAGS/PYTHON/\)](#)

[MACHINE LEARNING \(/TAGS/MACHINE-LEARNING/\)](#)

[DATA SCIENCE \(/TAGS/DATA-SCIENCE/\)](#)

[CHEAT SHEET \(/TAGS/CHEAT-SHEET/\)](#)

[가이드 \(/TAGS/%EA%B0%80%EC%9D%B4%EB%93%9C/\)](#)

[사용법 \(/TAGS/%EC%82%AC%EC%9A%A9%EB%B2%95/\)](#)

[김태완 \(/TAGS/%EA%B9%80%ED%83%9C%EC%99%84/\)](#)



작성자: 김태완

1999년 부터 Java, Framework, Middleware, SOA, DB Replication, Cache, CEP, NoSQL, Big Data, Cloud를 키워드로 살아왔습니다. 현재는 빅데이터와 Machine Learning을 중점에 두고 있습니다.
E-mail: taewanme@gmail.com

«PREVIOUS


[번역] JShell 사용자 가이드
(http://taewan.kim/post/trans_js_hell/)

NEXT»


Cloud Native Java:GraalVM
(@Oracle Developer Meetup)
(http://taewan.kim/post/graalvm_seminar/)

11 Comments

Sort by Oldest




Add a comment...



김장훈

여태 봤던 어느 자료보다 제일 좋습니다. 잘 보고 갑니다.


Like · Reply · 2y



김태완

감사합니다.


Like · Reply · 1 · 2y



임진규

감사합니다 잘 정리되어있어서 배우기도 쉽네요.


Like · Reply · 1 · 2y · Edited



김태완

감사합니다. ^^


Like · Reply · 1 · 2y



Jeong Hun Lee

너무 좋은 자료 감사드립니다.
한 가지 이해가 안되는 부분이 있어 질문 드립니다.
최상단 목차 1. 내용 중 '다차원 배열의 데이터 방향을 axis로 표현할 수 있습니다. 행방향(높이), 열방향(폭), 채널 방향은 각각 axis=0, axis=1 그리고 axis=2로 지정됩니다'
다른 글 중 'Numpy axis이해' 부분에선 그림 상 axis=0을 row 인 폭으로 제가 잘못 이해하여 혼동이 옵니다. 이 바보에게 가르침을 내려주셨으면 합니다.
너무나 좋은 글 다시 한 번 감사드립니다


Like · Reply · 1y



김한빛

캐글 입문하기 위해서 numpy를 공부하려하는데 정말 좋은 자료 감사합니다!! 제가 돌머리라서 이해가 잘 안 됐는데 이 글은 정말 속속 들어오네요 ㅎㅎ
다만 초반에 shape를 설명할 때, shaep라고 오타를 치신 듯합니다. 이 부분을 수정하면 더욱 좋은 자료가 될 것 같습니다.
좋은 글에 다시 한번 감사드립니다!


Like · Reply · 1 · 1y



김한빛

shape 오타가 문서 전체에서 2건이 있는 것 같습니다. 제가 오타 지적은 가급적 하지 않는 편인데 이렇게 좋은 글에는 오타가 없었으면 하는 바람에 이렇게 댓글을 쓰게 되네요 ㅠ


Like · Reply · 1y



김태완

좋은 지적 감사드려요. 수정했습니다. 정말 감사드려요.


Like · Reply · 1 · 1y



김한빛

김태완 빠른 수정 감사합니다 ㅎㅎ 잘 정리해주신 튜토리얼 덕분에 캐글 타이타닉 도전하고 있는 중입니다!


Like · Reply · 1y



김지형

좋은 자료 감사합니다. 학교 후배들 학습용 만드는 자료에 본 자료를 참고하여 재구성하고 싶는데 그래도 될까요?

Like · Reply · 1 · 1y



김태완

문제 없습니다. 레퍼런스로 출처만 남겨 주세요. 감사합니다.

Like · Reply · 1y

**임도형**

아주 훌륭한 자료 입니다. 정말 훌륭합니다.

기반하여 노트북 자료를 작성하여 교육에 사용해도 될까요?
당연히 출처는 밝히겠습니다.

Like · Reply · 1y

**김태완**

감사합니다. 사용해 주세요. 많이 써주세요. 출처 남겨 주시면 감사하겠습니다.

Like · Reply · 1y

**오혜신**

좋은 자료 감사합니다! 혹시 포스팅에 주피터노트북을 어떻게 임베드시키셨는지 궁금한데, 알려주실 수 있나요??

Like · Reply · 1y

**김태완**

안녕하세요 절대적인 노가다입니다.
hugo template에 jupyter css를 포함하는 별도 템플릿을 만들었습니다.

Like · Reply · 1y

**오혜신**

김태완 오... 그렇군요 — 알려주셔서 감사합니다!

Like · Reply · 1y

**장윤수**

최고의 자료네요 잘보고갑니다.

Like · Reply · 1y

**김태완**

감사합니다.

Like · Reply · 1y

**박희경**

좋은 자료 감사합니다. 공부하는데 큰 도움이 되고 있습니다.

Like · Reply · 1y