스마트 모빌리티 프로그래밍

Ch 10. 파이썬 확장 패키지 (1)
– NumPy

영남대학교 정보통신공학과

교수 김 영 탁

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- 파이썬 확장 패키지
- **NumPy**
- **NumPy** 유니버설 함수
- 선형대수 (**linear algebra**) 관련 유니버설 함수 및 응용

파이썬 확장 패키지

Python Extension Packages

◆ Python Extension Packages (1)

파이썬 확장 패키지	설 명
NumPy	http://www.numpy.org/ 파이썬 프로그램의 과학 기술 계산을 위하여 필요한 확장 패키지이며, 다음 기능들을 포함: - 우수한 성능의 다차원 배열 - 다양한 함수, 유니버설 함수 - C/C++ 및 Fortran 프로그램 모듈을 결합할 수 있는 기능 - 선형대수, 퓨리에 변환, 랜덤 숫자 생성 및 응용 기능
SciPy	https://www.scipy.org/ 파이썬 기반의 수학, 과학, 공학 분야 프로그램을 오픈 소스 소프트웨어로 설계, 구현 및 활용할 수 있는 환경 제공. NumPy, SciPy, Matplotlib, IPython, Sympy, pandas 등의 사용 환경 제공
Matplotlib	https://matplotlib.org/ Matplotlib은 문서 및 서적 출판에서 사용가능한 품질의 2차원 그래프 및 도형 출력 라이브러리이며, 전문 인쇄용 출력 포맷 지원. Matplotlib은 파이썬 스크립트, IPython 쉘, Jupyter 노트북, 웹 응용 서버 등에서 사용할 수 있으며, 다수의 그래픽 사용자 접속 툴 킷에서 사용가능.

Python Extension Packages

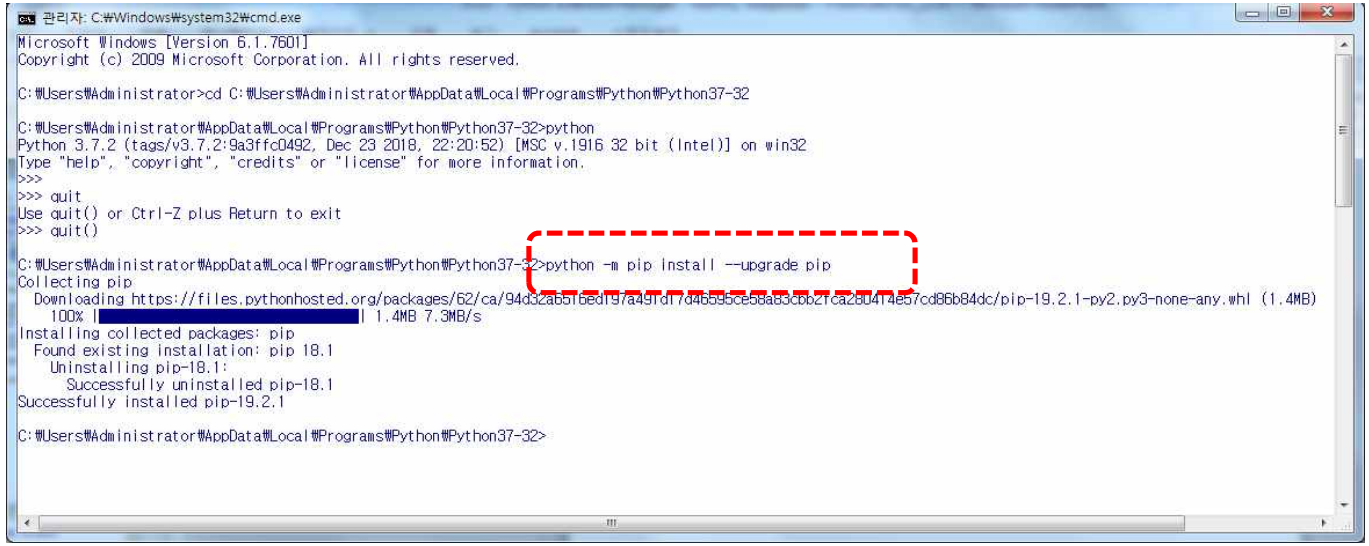
◆ Python Extension Packages (2)

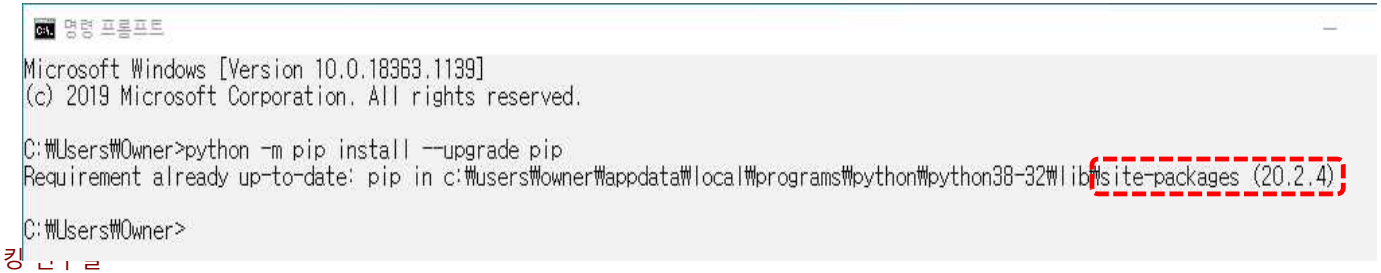
파이썬 확장 패키지	설 명
OpenCV	https://opencv.org/ OpenCV는 실시간 영상 처리가 가능하도록 계산 능력을 강화시킨 C/C++ 라이브러리이며, 멀티 코어 병렬 처리 기능을 활용함. OpenCL 기능을 사용하여 다양한 신호처리 가속 하드웨어 모듈을 활용할 수 있게 함. OpenVS는 BSD 라이센스로 제공되며 교육용 및 상업용에서 무료로 사용 가능. C++, 파이썬, Java 접속 기능이 제공되며, Windows, Linux, Mac OS, iOS 및 Android 환경에서 사용 가능.
Pandas	데이터 분석 확장 패키지 https://pandas.pydata.org/pandas-docs/stable/
TensorFlow	인공지능 분야에 많이 사용되는 확장 패키지
Keras	파이썬으로 구현된 딥러닝 라이브러리이며, TensorFlow위에서 딥러닝 모델을 쉽게 구성할 수 있게 함
R	통계 분석에 많이 사용되는 확장 패키지

Python Package Manager - pip

◆ Windows command 창에서 pip upgrade

> python –m pip install --upgrade pip





Installation/Upgrade of SciPy

◆ Install/Upgrade SciPy for NumPy

> python -m pip install --upgrade scipy

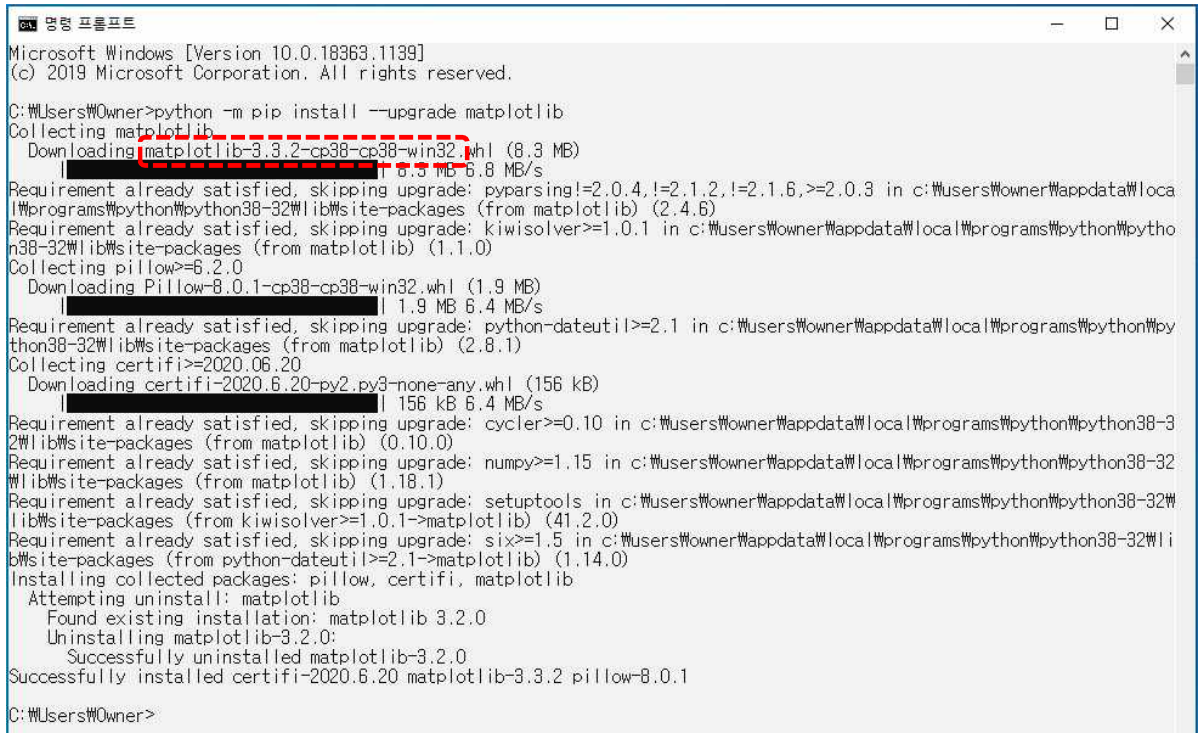
```
C:₩Users₩Administrator₩AppData₩Local₩Programs₩Python₩Python37-32>python -m pip install --upgrade scipy
Collecting scipy
  Downloading https://files.pythonhosted.org/packages/be/cc/6f7842a4d9aa7f51155f849185631e1201df255742de84d724ac33bff723/scipy-1.3.0-cp37-cp37m-win32.whl (27.1MB)
     |████████████████████████████████| 27.1MB 1.3MB/s
Requirement already satisfied, skipping upgrade: numpy>=1.13.3 in c:₩users₩administrator₩appdata₩local₩programs₩python₩python37-32₩lib₩site-packages (from scipy)
Installing collected packages: scipy
  Found existing installation: scipy 1.1.0
    Uninstalling scipy-1.1.0:
      Successfully uninstalled scipy-1.1.0
Successfully installed scipy-1.3.0

C:₩Users₩Administrator₩AppData₩Local₩Programs₩Python₩Python37-32>
```

```
C:₩MyPyPrograms₩Cython₩select_sort>python -m pip install --upgrade scipy
Collecting scipy
  Downloading scipy-1.7.3-cp39-cp39-win_amd64.whl (34.3 MB)
     |████████████████████████████████| 34.3 MB 6.4 MB/s
Requirement already satisfied: numpy<1.23.0,>=1.16.5 in c:₩users₩owner₩appdata₩local₩programs₩python₩python39₩lib₩site-packages (from scipy) (1.21.5)
Installing collected packages: scipy
Successfully installed scipy-1.7.3
```

Installation/Upgrade of MatplotLib

◆ Installation/Upgrade of MatplotLib

> python -m pip install --upgrade matplotlib



NumPy (1)

- ndarray 개요

NumPy

◆ NumPy

- http://www.numpy.org/
- 행렬 (matrix)이나 대규모 다차원 배열을 쉽게 처리할 수 있는 자료구조 기능을 지원하며, 다양한 수치해석과 통계분석 기능을 제공하는 파이썬 라이브러리
- Numarray와 Numeric이라는 오래된 패키지를 계승해서 나온 수학 및 과학 연산을 위한 파이썬 패키지
- 대표적으로 다양한 종류의 배열을 구성할 수 있는 ndarray 자료구조가 제공되며, 주어진 데이터에 대한 통계분석과 수치해석, 선형시스템 해석 등의 다양한 기능을 제공
- NumPy 패키지의 대부분 함수들은 C나 Fortran으로 구현되어 최적화된 상태로 널리 사용되고 있던 함수들을 기반으로 구현되어 있기 때문에 매우 우수한 성능을 제공
- NumPy 패키지는 주로 SciPy 패키지에 포함되어 Matplotlib 등과 함께 사용되는 것이 일반적임
- NumPy Source Code: **https://github.com/numpy/numpy/tree/master/numpy/core/src/npysort**

NumPy의 배열 클래스 - ndarray

◆ ndarray 클래스의 매소드

ndarray 메소드	설명
ndarray.ndim()	• 배열의 차원 (dimension)을 설정 • 배열의 차원은 axes 또는 rank라고도 함
ndarray.shape()	• 배열의 각 차원 크기를 튜플로 반환
ndarray.size()	• 배열의 전체 원소 개수
ndarray.dtype()	• 배열 원소의 자료형 (예: numpy.int32, numpy.int16, numpy.float64)
ndarray.itemsize()	• 배열 원소 자료형의 크기 (바이트 수)
ndarray.data()	• 배열의 버퍼 (이 버퍼를 직접 사용하지는 않음)

NumPy에서 사용가능한 자료형

◆ NumPy 자료형

자료형	typestr 설정	설명
int8 uint8	i1 u1	1-바이트 (8-비트) 부호사용 정수 1-바이트 (8-비트) 부호없는 정수
int16 uint16	i2 u2	2-바이트 (16-비트) 부호사용 정수 2-바이트 (16-비트) 부호없는 정수
int32 uint32	i4 u4	4-바이트 (32-비트) 부호사용 정수 4-바이트 (32-비트) 부호없는 정수
int64 uint64	i8 u8	8-바이트 (64-비트) 부호사용 정수 8-바이트 (64-비트) 부호없는 정수
float16	f2	2-바이트 (16비트) 실수 (float)
float32	f4, f	4-바이트 (32비트) 실수 (float)
float64	f8, d	8-바이트 (64비트) 실수 (float)
float128	f16, g	16-바이트 (128비트) 실수 (float)
complex64	c8	실수부와 허수부 각각 4-bytes (32-bit)로 구성된 복소수
complex128	c16	실수부와 허수부 각각 8-bytes (64-bit)로 구성된 복소수
complex256	c32	실수부와 허수부 각각 16-bytes (128-bit)로 구성된 복소수
bool	?	부울 데이터 (True 또는 False)
object	O	파이썬 객체 (object)
string_, bytes_	S	문자열 (string)
unicode_, str_	U	유니코드 문자열 (unicode string)
void	V	raw data (void)

ndarray 생성에 관련된 NumPy 함수

ndarray 생성관련 NumPy 함수	설명
np.array([[], ...])	• 2차원 리스트를 사용한 ndarray 생성
np.arange(N) np.arange(start, end, stride)	• range() 함수와 유사하며, 0..N-1의 1차원 배열 생성 • start~end 구간의 실수를 stride 간격으로 생성
np.zeros(shape, dtype=float, order='C')	• shape은 (N, M) 튜플 형식으로 N x M 배열의 크기를 지정하며, 배열의 원소가 모두 0.0인 배열을 생성 • dtype은 배열원소의 자료형 지정 • order='C'이면 C-program style로 행 (row) 우선 순서 (row-major order)로 원소 생성 • order='F' 이면 Fortran-program style로 열 (column) 우선 순서 (column-major order)로 원소 생성
np.ones(shape, dtype=float, order='C')	• shape은 (N, M) 튜플 형식으로 N x M 배열의 크기를 지정하며, 배열의 원소가 모두 1.0인 배열을 생성 • dtype은 배열원소의 자료형 지정 • order='C'이면 C-program style로 행 (row) 우선 순서 (row-major order)로 원소 생성 • order='F' 이면 Fortran-program style로 열 (column) 우선 순서 (column-major order)로 원소 생성
np.empty(shape, dtype=float, order='C')	• 초기화 되지 않은 shape 차원의 ndarray 배열 객체를 생성하여 반환
np.full(shape, fill_value, dtype=None, order='C')	• fill_value로 초기화된 shape 차원의 ndarray 배열 객체를 생성하여 반환
np.identity(R, dtype=None)	• R x R 크기의 dtype 자료형 원소로 구성된 2차원 정방 (square) 행렬에서 대각선 원소만 1.0이며, 나머지 원소들은 모두 0.0인 단위 행렬 (identity matrix) 생성

ndarray 생성에 관련된 NumPy 함수

ndarray 생성관련 NumPy 함수	설명
np.eye(R, C=None, k=0, dtype='float')	• R x C 크기의 행렬에서 대각선에서 k 만큼 떨어진 대각선 원소들만 1.0이며, 나머지 원소는 모두 0.0인 행렬 생성 • 만약 C가 설정되지 않으면 행렬의 크기가 R x R로 설정됨
np.linspace(start, end, num=50, endpoint=True, retstep=False, dtype=None)	• start~end 범위에서 등간격으로 num 개 샘플을 생성하여 배열 구성 • endpoint=True이면 end를 포함. retstep=True이면 (samples, step) 튜플을 반환 • dtype=None이면 나머지 인수로 자료형을 추정
np.logspace(start, end, num=50, endpoint=True, base=10.0, dtype=None)	• 행렬의 원소 값이 start~end 범위를 log 스케일에서 등간격으로 샘플을 생성. 배열 요소 값은 base**start~base**end 구간의 값을 가짐 • 배열 요소 간격은 ln(samples)/ln(base)로 설정됨 • y=np.linspace(start, stop), np.power(base, y). astype(dtype)을 적용한 값과 동일
np.power(a, E)	• 배열 a의 각 원소를 배열 E의 각 원소값으로 지수승 계산
X = np.linspace() Y = np.linspace() np.meshgrid(X, Y)	• 좌표벡터 (coordinate vectors) X와 Y를 기반으로 2차원 함수 의 좌표값 쌍 (x, y), 즉 그리드의 행렬 (coordinate matrices)로 반환
np.dtype()	• ndarray 배열의 자료형 지정 • "<i4": little endian, 4-byte integer (itemsie 4, name 'int32') • ">i4": big endian, 4-byte integer • "f": float

리스트를 사용한 ndarray 생성

◆ ndarray 생성 – 리스트 사용

```
# NumPy ndarray – dtype, ndim, shape, size, itemsize, data

import numpy as np
A = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.int32) #dtype = "i4"
print("A : \n", A)
print("A.dtype : ", A.dtype)
print("A.ndim : ", A.ndim)
print("A.shape : ", A.shape)
print("A.size : ", A.size)
print("A.itemsize : ", A.itemsize)
print("A.data : ", A.data)
```

```
>>>
 RESTART: C:/YTK-Progs/2018 Book (Python)/ch 15
A :
 [[1 2 3]
 [4 5 6]]
A.dtype :  int32
A.ndim :  2
A.shape :  (2, 3)
A.size :  6
A.itemsize :  4
A.data :  <memory at 0x04F6B4E0>
>>>
```

arange()

◆ ndarray 생성 – arange() 사용

```
# NumPy ndarray - arange()
import numpy as np
A = np.arange(3)
print("A : ", A)
print("A.type : ", A.dtype)

B = np.arange(3.0)
print("B : ", B)
print("B.type : ", B.dtype)

C = np.arange(3, 10)
print("C : ", C)
print("C.type : ", C.dtype)

D = np.arange(3, 10, 2)
print("D : ", D)
print("D.type : ", D.dtype)
```

```
>>>
 RESTART: C:/YTK-Progs/201
A :  [0 1 2]
A.type :  int32
B :  [0. 1. 2.]
B.type :  float64
C :  [3 4 5 6 7 8 9]
C.type :  int32
D :  [3 5 7 9]
D.type :  int32
>>>
```

ones(), empty(), full()

◆ ndarray 생성 – ones(), empty(), full() 사용

```
# NumPy ndarray - ones(), empty(), full()
import numpy as np
B1 = np.ones((2, 3))
print("B1 : \n", B1)

B2 = np.zeros((2, 3))
print("B2 : \n", B2)

B3 = np.empty((2, 3))
print("B3 : \n", B3)

B4 = np.full((2, 3), 10, dtype=None, order='C')
print("B4 : \n", B4)
```

```
B1 :
 [[1. 1. 1.]
 [1. 1. 1.]]
B2 :
 [[0. 0. 0.]
 [0. 0. 0.]]
B3 :
 [[0. 0. 0.]
 [0. 0. 0.]]
B4 :
 [[10 10 10]
 [10 10 10]]
```

identity(), eye()

◆ ndarray 생성 – identity(), eye() 사용

```
# NumPy ndarray - identity(), eye()
import numpy as np
C1 = np.identity(3)
print("C1 : \n", C1)

D1 = np.eye(3, dtype=int)
print("D1 : \n", D1)

D2 = np.eye(3, k=1, dtype=int)
print("D2 : \n", D2)

D3 = np.eye(3, k=-1, dtype=int)
print("D3 : \n", D3)
```

```
C1 :
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
D1 :
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
D2 :
 [[0 1 0]
 [0 0 1]
 [0 0 0]]
D3 :
 [[0 0 0]
 [1 0 0]
 [0 1 0]]
>>>
```

linspace(), logspace()

◆ ndarray 생성 – linspace(), logspace() 사용

```
# NumPy ndarray - linspace(), logspace()
import numpy as np
E1 = np.linspace(0.0, 10.0, num=5)
print("E1 : \n", E1)

E2 = np.linspace(0.0, 10.0, num=5, endpoint=False, retstep=True)
print("E2 : \n", E2)

E3 = np.logspace(0.1, 1, num=10)
print("E3 : \n", E3)

E4 = np.linspace(0.1, 1, num=10)
print("E4 : \n", E4)

E5 = np.power(10, E4).astype('float64')
print("E5 : \n", E5)
```

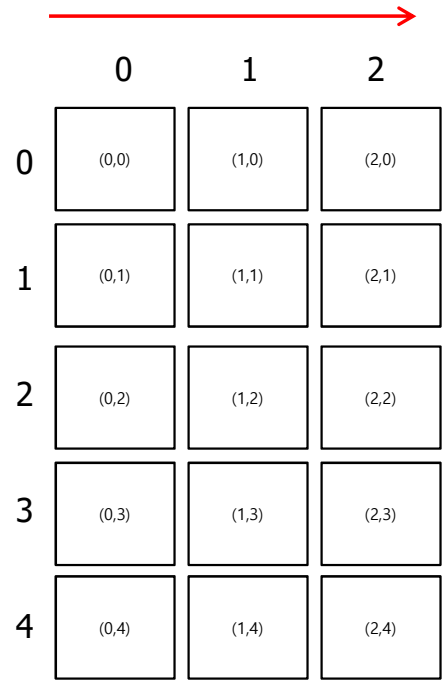
```
E1 :
 [ 0.   2.5  5.   7.5 10. ]
E2 :
 (array([0., 2., 4., 6., 8.]), 2.0)
E3 :
 [ 1.25892541  1.58489319  1.99526231  2.51188643  3.16227766  3.98107171
  5.01187234  6.30957344  7.94328235 10.        ]
E4 :
 [0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
E5 :
 [ 1.25892541  1.58489319  1.99526231  2.51188643  3.16227766  3.98107171
  5.01187234  6.30957344  7.94328235 10.        ]
```

meshgrid()

◆ meshgrid

```
# NumPy ndarray - meshgrid()
import numpy as np

x = np.arange(3)
y = np.arange(5)
print("x : \n", x)
print("y : \n", y)

X,Y = np.meshgrid(x, y) #
print("X : \n", X)
print("Y : \n", Y)
```

```
x :
 [0 1 2]
y :
 [0 1 2 3 4]
X :
 [[0 1 2]
 [0 1 2]
 [0 1 2]
 [0 1 2]
 [0 1 2]]
Y :
 [[0 0 0]
 [1 1 1]
 [2 2 2]
 [3 3 3]
 [4 4 4]]
```



X : 2-D array of x-coordinates
Y : 2_D array of y-coordinates

ndarray 배열 원소의 자료형 지정

◆ dtype()을 사용한 ndarray 배열원소의 자료형 지정 (1)

자료형 지정 유형	예, 설명
array-protocol type strings	dt = np.dtype('i4') # 32-bit 부호 사용 정수 dt = np.dtype('f8') # 64-bit 부동소수점 실수 dt = np.dtype('c16') # 128-bit 복소수 (부동소수점 실수부 및 허수부) dt = np.dtype('a25') # 0으로 종료되는 25-바이트 dt = np.dtype('U25') # 25-문자 문자열
array scalar type	dt = np.dtype(np.int32) # 32-비트 정수 dt = np.dtype(np.complex128) # 128-비트 복소수 (부동소수점 실수부 및 허수부)
byte-order, built-in data type	바이트 저장순서: '<' (little endian), '>' (big endian), '=' (native, 하드웨어 기본설정)
	내장 자료형: i4 (32-비트 정수), f8 (64-비트 부동소수점), 'c16 (64-비트 복소수), a25(0으로 종료되는 25-바이트), U25 (25-문자 문자열)

	dt = np.dtype("<i4") # little endian, 4-byte (32-bit) 정수
	dt = np.dtype(">i4") # big endian, 4-byte (32-bit) 정수

(flexible_dtype, itemsize)	dt = np.dtype((np.void, 10)) # 10바이트 크기의 데이터 블록 np.dtype(('U', 10)) # 10-문자의 유니코드 문자열
(fixed_dtype, shape)	dt = np.dtype((np.int32, (2,2))) # 2×2 정수 서브 배열

ndarray 배열 원소의 자료형 지정

◆ dtype()을 사용한 ndarray 배열원소의 자료형 지정 (2)

자료형 지정 유형	예, 설명

[(field_name, field_dtype, field_shape), ...]	dt = np.dtype([('big', '>i4'), ('little', '<i4')]) # fields big (big-endian 32-bit integer) and little (little-endian 32-bit integer)
	dt = np.dtype([('R','u1'), ('G','u1'), ('B','u1'), ('A','u1')]) # fields R, G, B, A, each being an unsigned 8-bit integer

{'names': ..., 'formats': ..., 'offsets': ..., 'titles': ..., 'itemsize': ...}	dt = np.dtype({'names': ['r','g','b','a'], 'formats': [uint8, uint8, uint8, uint8]}) # r, g, b, a 항목은 각각 8-비트 부호없는 정수
	dt = np.dtype({'names': ['r','b'], 'formats': ['u1', 'u1'], 'offsets': [0, 2], 'titles': ['Red pixel', 'Blue pixel']}) # 8-비트 부호없는 정수로 표현된 r과 b 항목 (제목을 포함), 첫 항목은 바이트 0에 위치, 두 번째 항목은 바이트 2에 위치

(base_dtype, new_dtype)	dt = np.dtype((np.int32,{'real':(np.int16, 0),'imag':(np.int16, 2)}) # 32-비트 정수, 첫 2바이트 정수는 실수부, 나중 2바이트 정수는 허수부 dt = np.dtype((np.int32, (np.int8, 4))) # 8-비트 정수 4개를 서브 배열로 해석되는 32-비트 정수 dt = np.dtype(('i4', [('r','u1'),('g','u1'),('b','u1'),('a','u1')])) # 부호없는 r, g, b, a 항목으로 구성된 4바이트 정수

(Source: https://numpy.org/doc/stable/reference/arrays.dtypes.html)

data type - dtype()

```
# NumPy ndarray - linspace(), logspace()
import numpy as np
dt = np.dtype('<i4')  # < means little endian
print("dt.byteorder : ", dt.byteorder)
print("dt.itemsize : ", dt.itemsize)
print("dt.name : ", dt.name)

dt2 = np.dtype([('name', np.str_, 16), ('scores', np.float64, (2, ))])
A = np.array([('Kim', (80.0, 90.0)), ('Lee', (70.0, 80.0))], dtype=dt2)
print("A : ", A)
print("A[0] : ", A[0])
print("A[0]['name'] : ", A[0]['name'])
print("A[0]['scores'] : ", A[0]['scores'])

print("type(A[0]) : ", type(A[0]))
print("type(A[0]['name']) : ", type(A[0]['name']))
print("type(A[0]['scores']) : ", type(A[0]['scores']))
```

```
dt.byteorder :  =
dt.itemsize :  4
dt.name :  int32
A :  [('Kim', [80., 90.]) ('Lee', [70., 80.])]
A[0] :  ('Kim', [80., 90.])
A[0]['name'] :  Kim
A[0]['scores'] :  [80. 90.]
type(A[0]) :  <class 'numpy.void'>
type(A[0]['name']) :  <class 'numpy.str_'>
type(A[0]['scores']) :  <class 'numpy.ndarray'>
```

NumPy (2)

- ndarray 결합, 분할, 치환
- ndarray 인덱싱, 슬라이싱
- ndarray의 연산

ndarray 결합, 분할, 모양 변경에 관련된 NumPy 함수

◆ ndarray 결합 및 분할

ndarray 분할관련 NumPy 함수	설명
np.vstack()	열을 기준으로 수직 방향으로 배열을 차례로 쌓아 새로운 배열을 생성
np.hstack()	행을 기준으로 수평 방향으로 배열을 차례로 연결시켜 새로운 배열을 생성
np.vsplit()	열을 기준으로 수직 방향으로 배열을 등분하여 새로운 배열을 생성
np.hsplit()	행을 기준으로 수평 방향으로 배열을 등분하여 새로운 배열을 생성

◆ ndarray 모양 변경

ndarray 변환관련 NumPy 함수	설명
np.reshape(shape, order='C')	주어진 배열을 shape 모양의 배열로 변환
np.ravel()	다차원 배열을 1차원 배열로 변환
np.flatten()	다차원 배열을 1차원 배열로 변환
np.transpose()	전치행렬을 반환
np.swapaxes()	전치행렬을 반환

vstack(), vsplit()

```
# NumPy ndarray - vstack(), split()
import numpy as np
A = np.array([1, 2, 3])
B = np.array([4, 5, 6])
C = np.vstack((A, B))
print("A : ", A)
print("B : ", B)
print("C : ", C)

D = np.vstack(([1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]))
print("D : \n", D)
E = np.split(D, 2)
print("E : \n", E)
F = np.split(D, [2, 3])
print("F : \n", F)
```

```
A :  [1 2 3]
B :  [4 5 6]
C :  [[1 2 3]
 [4 5 6]]
D :
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
E :
 [array([[1, 2, 3],
        [4, 5, 6]]), array([[ 7,  8,  9],
        [10, 11, 12]])]
F :
 [array([[1, 2, 3],
        [4, 5, 6]]), array([[7, 8, 9]]), array([[10, 11, 12]])]
```

hstack(), hsplit()

```
# NumPy ndarray - hstack(), hsplit()
import numpy as np
A = np.array([1, 2, 3])
B = np.array([4, 5, 6])
C = np.hstack((A, B))
print("A : ", A)
print("B : ", B)
print("C : ", C)

D = np.array([[1], [2], [3]])
E = np.array([[4], [5], [6]])
F = np.hstack((D, E))
print("D : \n", D)
print("E : \n", E)
print("F : \n", F)


G = np.arange(8).reshape(2,4)
print("G : \n", G)

H = np.hsplit(G, 2)
print("H : \n", H)

K = np.hsplit(G, [2, 3])
print("K : \n", K)
```

```
A :  [1 2 3]
B :  [4 5 6]
C :  [1 2 3 4 5 6]
D :
 [[1]
 [2]
 [3]]
E :
 [[4]
 [5]
 [6]]
F :
 [[1 4]
 [2 5]
 [3 6]]
G :
 [[0 1 2 3]
 [4 5 6 7]]
H :
 [array([[0, 1],
        [4, 5]]), array([[2, 3],
        [6, 7]])]
K :
 [array([[0, 1],
        [4, 5]]), array([[2],
        [6]]), array([[3],
        [7]])]
```

ndarray shape(), reshape()

```
# NumPy ndarray(11) – shape(), reshape()
import numpy as np
A = np.arange(10)
print("A : ", A)
print("A.ndim : ", A.ndim)
print("A.shape : ", A.shape)

B = A.reshape((2, 5))
print("B : ", B)
print("B.ndim : ", B.ndim)
print("B.shape : ", B.shape)

C = A.reshape((5, -1))
print("C : \n", C)
print("A : ", A)

A.shape = (5, 2)
print("After A.shape(5, 2) ==> A : \n", A)

print("B : \n", B)
D = B.ravel()
print("D ( = B.ravel())  : ", D)
E = B.flatten()
print("E ( = B.flatten()): ", E)
```

```
A :  [0 1 2 3 4 5 6 7 8 9]
A.ndim :  1
A.shape :  (10,)
B :  [[0 1 2 3 4]
 [5 6 7 8 9]]
B.ndim :  2
B.shape :  (2, 5)
C :
 [[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
A :  [0 1 2 3 4 5 6 7 8 9]
After A.shape(5, 2) ==> A :
 [[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
B :
 [[0 1 2 3 4]
 [5 6 7 8 9]]
D ( = B.ravel())  :  [0 1 2 3 4 5 6 7 8 9]
E ( = B.flatten()):  [0 1 2 3 4 5 6 7 8 9]
```

행렬의 치환 - transpose()

◆ 행렬의 치환

```
# NumPy ndarray(12) - transpose()
import numpy as np
A = np.arange(15).reshape(5, 3)
print("A : \n", A)
B = A.T
print("B (A.T) : \n", B)

C = A.transpose()
print("C (A.transpose()) : \n", C)
```

```
A :
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]]
B (A.T) :
 [[ 0  3  6  9 12]
 [ 1  4  7 10 13]
 [ 2  5  8 11 14]]
C (A.transpose()) :
 [[ 0  3  6  9 12]
 [ 1  4  7 10 13]
 [ 2  5  8 11 14]]
```

ndarray 인덱싱, 슬라이싱, sum()

◆ ndarray 인덱싱과 슬라이싱에 관련된 NumPy 함수

ndarray 인덱싱과 슬라이싱 관련 NumPy 함수	설명
A[n] A[0], A[-1]	ndarray 행렬 A의 원소를 인덱스로 지정
A[start:end:stride]	ndarray 행렬 A의 원소들 중 start에서 end직전까지 stride 간격으로 지정된 원소들로 구성된 부분 배열
A.sum([axis=0])	ndarray 행렬 A의 원소들의 합산 2차원 행렬에서는 axis=0 (열), axis=1 (행) , axis=-1 (마지막 기준, 행) 기준 합산 3차원 행렬에서는 axis=0 (깊이), axis=1 (열) , axis=2 (행) , axis=-1 (마지막 기준, 행) 기준 합산 axis를 전달하지 않으면 전체 원소를 합산
A.rowsum()	ndarray 행렬 A의 행을 기준으로 원소들을 합산

ndarray에 대한 인덱싱과 슬라이싱

```
# NumPy ndarray indexing, slicing (1)
import numpy as np
A = np.arange(10)
print("A : \n", A)

print("A[0] : ", A[0])
print("A[-1] : ", A[-1])
print("A[:5] : ", A[:5])
print("A[::2] : ", A[::2])

A[:5] = 10
print("After A[:5] = 10 ==> A : \n", A)

B = A[5:] # assignment
print("B (A[5:]) : \n", B)
B[0] = 20
print("After B[0]=20 ==> A : \n", A)

C = A[5:].copy()  # copy
C[0] = 30
print("After C[0]=30 ==> C : \n", C)
print("A : \n", A)
```

```
A :
 [0 1 2 3 4 5 6 7 8 9]
A[0] :  0
A[-1] :  9
A[:5] :  [0 1 2 3 4]
A[::2] :  [0 2 4 6 8]
After A[:5] = 10 ==> A :
 [10 10 10 10 10  5  6  7  8  9]
B (A[5:]) :
 [5 6 7 8 9]
After B[0]=20 ==> A :
 [10 10 10 10 10 20  6  7  8  9]
After C[0]=30 ==> C :
 [30  6  7  8  9]
A :
 [10 10 10 10 10 20  6  7  8  9]
```

2차원 ndarray 행렬에 대한 인덱싱과 슬라이싱

```
# NumPy 2-Dimensional ndarray indexing, slicing
import numpy as np
A = np.arange(12).reshape((3,4))
print("A : \n", A)

print("A[0] : ", A[0])
print("A[:2] : ", A[:2])
B = A[:, :1] # from all rows, select 0-th element
print("B (A[:, :1]) : \n", B)

C = A[:, 1:3] # from all rows,
              # select 1-st and 2-nd elements
print("C (A[:, 1:3]) : \n", C)

D = A[:, 1:3].copy()
print("D (A[:, 1:3].copy()) : \n", D)

A[:, 1:3] = -1
print("After A[:, 1:3] = -1 ==> A : \n", A)
A[:, 1:3] = D
print("After A[:, 1:3] = D ==> A : \n", A)
```

```
A :
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
A[0] :  [0 1 2 3]
A[:2] :  [[0 1 2 3]
 [4 5 6 7]]
B (A[:, :1]) :
 [[0]
 [4]
 [8]]
C (A[:, 1:3]) :
 [[ 1  2]
 [ 5  6]
 [ 9 10]]
D (A[:, 1:3].copy()) :
 [[ 1  2]
 [ 5  6]
 [ 9 10]]
After A[:, 1:3] = -1 ==> A :
 [[ 0 -1 -1  3]
 [ 4 -1 -1  7]
 [ 8 -1 -1 11]]
After A[:, 1:3] = D ==> A :
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

ndarray에 대한 연산

◆ ndarray에 대한 연산

연산	ndarray 수학연산	설명
산술	B = A+n	행렬 A의 각 원소에 n을 더함
	B = A-n	행렬 A의 각 원소에 n을 뺌
	B = A*n	행렬 A의 각 원소에 n을 곱함
	B = A/n	행렬 A의 각 원소에 n을 나눔 (실수 나눗셈)
	B = A//n	행렬 A의 각 원소에 n을 나눔 (정수 나눗셈)
	B = A%n	행렬 A의 각 원소에 n으로 모듈로 계산
	B = A**n	행렬 A의 각 원소에 n 지수승
	B = abs(A*n+m)	행렬 A의 각 원소에 연산을 한 후, 절대값을 계산
관계	A < n	행렬 A의 모든 원소가 n보다 작으면 True
	A > n	행렬 A의 모든 원소가 n보다 크면 True
논리	any()	행렬 A의 원소중 하나라도 True이면 결과는 True; 모든 원소가 False 이면 결과는 False
	all()	행렬 A의 모든 원소가 True이면 결과는 True; 원소 하나라도 False 이면 결과는 False

ndarray 배열에 대한 연산

```
# NumPy ndarray arithmetic operations (1) - add, sub, mult, div
import numpy as np
A = np.arange(10).reshape(2, 5)
print("A : \n", A)
B = A+10
print("B (A+10) : \n", B)
C = A-10
print("C (A-10) : \n", C)
D = A*10
print("D (A*10) : \n", D)
E = A/10
print("E (A/10) : \n", E)
F = A//2
print("F (A//2) : \n", F)
G = A%2
print("G (A%2) : \n", G)
H = A**2
print("H (A**2) : \n", H)
K = A*10 - 50
print("K (A*10 - 50) : \n", K)
M = abs(A*10 - 50)
print("M (abs(A*10 - 50)) : \n", M)
```

```
A :
 [[0 1 2 3 4]
 [5 6 7 8 9]]
B (A+10) :
 [[10 11 12 13 14]
 [15 16 17 18 19]]
C (A-10) :
 [[-10  -9  -8  -7  -6]
 [ -5  -4  -3  -2  -1]]
D (A*10) :
 [[ 0 10 20 30 40]
 [50 60 70 80 90]]
E (A/10) :
 [[0.  0.1 0.2 0.3 0.4]
 [0.5 0.6 0.7 0.8 0.9]]
F (A//2) :
 [[0 0 1 1 2]
 [2 3 3 4 4]]
G (A%2) :
 [[0 1 0 1 0]
 [1 0 1 0 1]]
H (A**2) :
 [[ 0  1  4  9 16]
 [25 36 49 64 81]]
K (A*10 - 50) :
 [[-50 -40 -30 -20 -10]
 [  0  10  20  30  40]]
M (abs(A*10 - 50)) :
 [[50 40 30 20 10]
 [ 0 10 20 30 40]]
```

ndarray에 대한 관계 연산과 논리 연산

```
# NumPy ndarray arithmetic operations (2) - relational, logical
import numpy as np
A = np.arange(10).reshape(2, 5)
print("A : \n", A)
B = A<5
print("B (A<5) : \n", B)
C = A>5
print("C (A>5) : \n", C)
D = (A%2 == 0)
print("D (A%2 == 0) : \n", D)
E = (A%2 != 0)
print("E (A%2 != 0) : \n", E)
F = (A>5).any()
print("F ((A>5).any()) : \n", F)
G = (A>5).all()
print("G ((A>5).all()) : \n", G)
H = np.logical_and(A%2==0, A>5)
print("H (np.logical_and(A%2==0, A>5)) : \n", H)
K = np.logical_or(A%2==0, A>5)
print("K (np.logical_or(A%2==0, A>5)) : \n", K)
M = np.logical_not(A%2==0)
print("M (np.logical_not(A%2==0)) : \n", M)
```

```
A :
 [[0 1 2 3 4]
 [5 6 7 8 9]]
B (A<5) :
 [[ True  True  True  True  True]
 [False False False False False]]
C (A>5) :
 [[False False False False False]
 [False  True  True  True  True]]
D (A%2 == 0) :
 [[ True False  True False  True]
 [False  True False  True False]]
E (A%2 != 0) :
 [[False  True False  True False]
 [ True False  True False  True]]
F ((A>5).any()) :
 True
G ((A>5).all()) :
 False
H (np.logical_and(A%2==0, A>5)) :
 [[False False False False False]
 [False  True False  True False]]
K (np.logical_or(A%2==0, A>5)) :
 [[ True False  True False  True]
 [False  True  True  True  True]]
M (np.logical_not(A%2==0)) :
 [[False  True False  True False]
 [ True False  True False  True]]
```

2차원 ndarray에 대한 sum()

```
# NumPy 2-Dimensional ndarray - sum(), sum(-1)
import numpy as np
A = np.arange(12).reshape((4, 3))
print("A = \n", A)

total_sum = A.sum() # sum all elements
print("A.sum() = \n", total_sum)

sum_0 = A.sum(0) # sum along axis=0 (column)
print("A.sum(0) = \n", sum_0)

sum_1 = A.sum(1) # sum along axis=1 (row)
print("A.sum(1) = \n", sum_1)

rowsum = A.sum(-1) # sum along the last axis (row)
print("A.sum(-1) = \n", rowsum)
```

```
A =
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
A.sum() =
 66
A.sum(0) =
 [18 22 26]
A.sum(1) =
 [ 3 12 21 30]
A.sum(-1) =
 [ 3 12 21 30]
```

3차원 ndarray 에 대한 sum()

```
# NumPy 3-Dimensional ndarray - sum(), sum(-1)
import numpy as np
A = np.arange(27).reshape((3, 3, 3))
print("A = \n", A)

total_sum = A.sum() # sum all elements
print("A.sum() = \n", total_sum)

sum_0 = A.sum(0) # sum along axis=0 (depth)
print("A.sum(0) = \n", sum_0)

sum_1 = A.sum(1) # sum along axis=1 (col)
print("A.sum(1) = \n", sum_1)

sum_2 = A.sum(2) # sum along axis=2 (row)
print("A.sum(2) = \n", sum_2)

sum_last_axis = A.sum(-1) # sum along the last axis (row)
print("(A.sum(-1)) = \n", sum_last_axis)
```

```
A =
 [[[ 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]]]
A.sum() =
 351
A.sum(0) =
 [[27 30 33]
 [36 39 42]
 [45 48 51]]
A.sum(1) =
 [[ 9 12 15]
 [36 39 42]
 [63 66 69]]
A.sum(2) =
 [[ 3 12 21]
 [30 39 48]
 [57 66 75]]
(A.sum(-1)) =
 [[ 3 12 21]
 [30 39 48]
 [57 66 75]]
```

유니버설 함수 (Universal Function, ufunc)

Universal Function (ufunc)

◆ NumPy 유니버설 함수

- NumPy 모듈은 다양한 유니버설 함수를 제공하며, 이 함수를 사용하여 인수로 주어지는 배열의 원소들에 대하여 다양한 수학 연산을 수행할 수 있음
- 유니버설 함수 (ufunc)에는 배열에 적용할 수 있는 기본 산술 연산, 삼각함수 연산, 비트 단위 연산, 논리 연산, 실수 (real number) 관련 연산, 집합 (set) 연산, 배열의 파일 입출력 연산, 배열의 정렬 및 탐색 관련 함수, 난수 관련 함수, 통계처리 및 선형대수 함수 등이 포함되어 있음

◆ 기본적인 유니버설 함수

기본적인 유니버설 함수	설 명
ufunc.reduce(A, axis=0, dtype=None, out=None, keepdims=False)	유니버설 함수 ufunc를 배열 A의 지정된 축 (axis)에 따라 적용 (2차원배열의 axis 지정: 0 column, 1 row; 3차원 배열의 axis 지정: 0 depth, 1 column, 2 row)
ufunc.reduceat(A, indices, axis=0, dtype=None, out=None)	배열 A를 지정된 인덱스 (indices)와 축(axis) 방향으로 차원을 줄임
ufunc.accumulate(A, axis=0, dtype=None, out=None)	배열 A에 유니버설 함수 ufunc를 지정된 축 (axis)에 따라 적용한 결과 합산
ufunc.outer(A, B)	배열 A와 B의 각 원소에 대하여 유니버설 함수 ufunc를 수행
ufunc.at(A, indices, b=None)	유니버설 함수 ufunc를 배열 A의 지정된 인덱스 (indices) 원소에 대하여 수행하며 b 연산수(operand)를 사용

◆ universal function – reduce()

```
# universal function – reduce()

import numpy as np
A = np.arange(10)
print("A : \n", A)
a = np.add.reduce(A)
print("a = np.add.reduce(A) : ", a)

B = np.arange(9).reshape((3, 3))
print("B : \n", B)
r0 = np.add.reduce(B, 0) # sum axis 0 (column)
r1 = np.add.reduce(B, 1) # sum axis 1 (row)
print("r0 :", r0)
print("r1 :", r1)

C = np.arange(8).reshape((2,2,2))
print("C : \n", C)
s0 = np.add.reduce(C, 0) # axis 0 (depth)
s1 = np.add.reduce(C, 1) # axis 1 (column)
s2 = np.add.reduce(C, 2) # axis 2 (row)
print("s0 :", s0)
print("s1 :", s1)
print("s2 :", s2)
```

```
A :
 [0 1 2 3 4 5 6 7 8 9]
a = np.add.reduce(A) :  45
B :
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
r0 : [ 9 12 15]
r1 : [ 3 12 21]
C :
 [[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
s0 : [[ 4  6]
 [ 8 10]]
s1 : [[ 2  4]
 [10 12]]
s2 : [[ 1  5]
 [ 9 13]]
```

수학연산 관련 유니버설 함수 (1)

수학연산 메소드	설 명
add(X, Y[, out])	배열 X와 Y를 더함 (덧셈)
subtract(X, Y[, out])	배열 X에서 Y를 뺌 (뺄셈)
multiply(X, Y[, out])	배열 X에 Y를 곱함 (곱셈)
divide(X, Y[, out]) true_divide(X, Y[, out])	X / Y (배열의 실수 나눗셈)
floor_divide(X, Y[, out])	X // Y (배열의 정수 나눗셈)
logaddexp(X, Y[, out])	log(exp(X) + exp(Y)) 지수승 덧셈에 대한 로그
logaddexp2(X, Y[, out])	log2(2**X + 2**Y) 2의 지수승 덧셈에 대한 밑이 2인 로그
negative(X[, out])	-X (부호 반전)
power(X, Y[, out])	X ** Y (X의 Y 지수승)
remainder(X, Y[, out]) mod(X, Y[, out])	X % Y연산의 나머지이며, 부호는 Y의 부호
fmod(X, Y[, out])	X % Y 연산의 나머지이며, 부호는 X의 부호
absolute(X[, out])	배열 X의 절대값
rint(X[, out])	반올림된 정수 (가장 가까운 정수); 자료형은 변경없음
sign(X[, out])	부호를 반환: -1, 0, 1
conj(X[, out])	켤레 복소수를 반환
exp(X[, out])	지수 (exponent), exp(X)
exp2(X[, out])	2**X

수학연산 관련 유니버설 함수 (2)

수학연산 메소드	설 명
log(X[, out])	자연 로그 (natural log)
log2(X[, out])	log_2
log10(X[, out])	log_10
expm1(X[, out])	exp(X) -1
log1p(X[, out])	log(1+X)
sqrt(X[, out])	X의 제곱근
square(X[, out])	X의 제곱 (X**2)
reciprocal(X[, out])	1/X
ones_like(X[, dtype, order, subok])	배열 X와 동일한 모양 및 자료형을 반환

◆ universal function – add()

```
# universal function - add

import numpy as np
x1 = np.arange(9.0).reshape((3, 3))
print("x1 = \n", x1)
x2 = np.add(x1, x1)
print("x2 = np.add(x1, x1) : \n", x2)
np.add(x1, x2, x1)
print("x1 after np.add(x1, x2, x1) : \n", x1)
```

```
x1 =
 [[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
x2 = np.add(x1, x1) :
 [[ 0.  2.  4.]
 [ 6.  8. 10.]
 [12. 14. 16.]]
x1 after np.add(x1, x2, x1) :
 [[ 0.  3.  6.]
 [ 9. 12. 15.]
 [18. 21. 24.]]
```

삼각함수 관련 유니버설 함수

삼각함수 관련 메소드	설명
sin(x[, out]) cos(x[, out]) tan(x[, out])	사인 함수, 코사인 함수, 탄젠트 함수
arcsin(x[, out]) arccos(x[, out]) arctan(x[, out]) arctan2(x1, x2[, out]	역 사인함수, 역 코사인 함수, 역 탄젠트 함수
hypot(x1, x2[, out])	(x1**2 + x2**2)의 제곱근
sinh(x[, out]) cosh(x[, out]) tanh(x[, out])	쌍곡선 사인, 쌍곡선 코사인, 쌍곡선 탄젠트
arcsinh(x[, out]) arccosh(x[, out]) arctanh(x[, out])	역 쌍곡선 사인, 역 쌍곡선 코사인, 역 쌍곡선 탄젠트
deg2rad(x[, out])	도(°) 단위 각도를 라디안 단위로 변환
rad2deg(x[, out])	라디안 단위 각도를 도(°) 단위로 변환

비트단위 연산 관련 유니버설 함수

비트단위 연산 메소드	설 명
bitwise_and(x1, x2[, out])	비트 단위 AND (x1 AND x2)
bitwise_or(x1, x2[, out])	비트 단위 OR (x1 OR x2)
bitwise_xor(x1, x2[, out])	비트 단위 XOR (x1 XOR x2); exclusive OR
invert(x[, out])	NOT x
left_shift(x1, x2[, out])	x1을 x2 비트 단위만큼 왼쪽으로 시프트
right_shift(x1, x2[, out])	x1을 x2 비트 단위만큼 오른쪽으로 시프트

논리연산 및 관계연산을 위한 유니버설 함수

◆ 논리 연산

논리연산 메소드	설 명
logical_and(x1, x2[, out])	x1와 x2의 논리곱
logical_or(x1, x2[, out])	x1와 x2의 논리합
logical_xor(x1, x2[, out])	x1와 x2의 배타적 논리합
logical_not(x[, out])	x의 반전 (NOT)

◆ 관계 연산

관계연산 메소드	설명
greater(x1, x2[, out])	x1 > x2
greater_equal(x1, x2[, out])	x1 >= x2
less(x1, x2[, out])	x1 < x2
less_equal(x1, x2[, out])	x1 <= x2
not_equal(x1, x2[, out])	x1 != x2
equal(x1, x2[, out])	x1 == x2

min, max 탐색을 위한 유니버설 함수

◆ max, min

min, max 연산 메소드	설 명
maximum(x1, x2[, out])	배열 x1와 x2에서 큰 원소를 선택하여 배열을 생성
minimum(x1, x2[, out])	배열 x1와 x2에서 작은 원소를 선택하여 배열을 생성
fmax(x1, x2[, out])	배열 x1와 x2에서 큰 원소를 선택하여 배열을 생성 만약 비교대상이 없으면 원소가 있는 항목을 사용
fmin(x1, x2[, out])	배열 x1와 x2에서 작은 원소를 선택하여 배열을 생성 만약 비교대상이 없으면 원소가 있는 항목을 사용

실수 (real number) 관련 유니버설 함수

◆ Real number universal function

실수 연산 메소드	설 명
isreal(x)	만약 x가 실수이면 True를 반환
iscomplex(x)	만약 x가 복소수이면 True를 반환
isfinite(x[, out])	만약 x가 np.inf이나 np.nan이 아니면 True를 반환
isinf(x[, out])	만약 x가 np.inf이나 np.nan이면 True를 반환
isnan(x[, out])	만약 x가 np.nan이면 True를 반환
signbit(x[, out])	만약 부호비트가 설정되어 있으면 (x <0), True를 반환
copysign(x1, x2[, out])	x1의 부호를 x2의 부호로 복사
modf(x[, out1, out2])	배열 x의 각 원소의 정수값 (integral part)와 소수값 (fractional part)을 배열로 반환 만약 배열 원소가 음수이면 정수값과 소수값이 음수가 됨
ldexp(x1, x2[, out])	x1 * 2**x2
frexp(x[, out1, out2])	배열 x의 각 원소에 대하여 x=mantissa * 2 ** exponent관계의 (가수부, 지수부)인 (mantissa, exponent) 튜플을 반환
fmod(x1, x2[, out])	x1 % x2의 나머지를 반환하며, 부호는 x1의 부호로 설정
floor(x[, out])	x보다 같거나 작으면서 가장 큰 정수
ceil(x[, out])	x보다 같거나 크면서 가장 작은 정수
trunc(x[, out])	x의 원소값에 소수점 이하 값을 버린 정수를 반환

집합 (set) 관련 유니버설 함수

◆ NumPy set function

집합 연산 메소드	설명
unique(ar, return_index=False, return_inverse=False, return_counts=False)	배열 ar의 중복된 요소를 제거하고, 정렬하여 반환. return_index가 True이면 반환된 배열의 원소가 위치한 곳의 인덱스를 반환 return_inverse가 True이면 역순으로 정렬 return_counts가 True이면 중복되지 않는 원소의 개수를 반환
isin(ar1, ar2, assume_unique=False, invert=False)	배열 ar1의 원소가 배열 ar2에 존재하는지를 True와 False로 표시하여 반환. 순서는 ar1의 순서에 따름.
intersect1d(ar1, ar2, assume_unique=False)	배열 ar1과 ar2의 교집합 (intersection)을 구하고, 중복된 원소들을 제거한 후 정렬하여 반환.
setdiff1d(ar1, ar2, assume_unique=False)	배열 ar1과 ar2의 차집합 (difference)을 구하여 반환. 배열 ar1에는 존재하지만 배열 ar2에는 존재하지 않는 원소들을 반환.
union1d(ar1, ar2)[source]	배열 ar1과 ar2의 합집합 (union)을 구하여 중복 원소를 제거하고, 정렬하여 반환
setxor1d(ar1, ar2, assume_unique=False)	배열 ar1과 ar2의 배타적 OR 연산을 수행하며, 두 배열 중에 한 쪽에만 포함되는 원소를 정렬하여 반환.

set - unique()

◆ set – unique()

```
# set () – unique()

import numpy as np
A = np.array([1, 6, 4, 4, 3, 3, 5, 5, 2])
print("A : ", A)
B = np.unique(A)
print("B : ", B)
u, indx = np.unique(A, return_index=True)
print("u : ", u)
print("indx : ", indx)
print("A[indx] : ", A[indx])
```

```
A :  [1 6 4 4 3 3 5 5 2]
B :  [1 2 3 4 5 6]
u :  [1 2 3 4 5 6]
indx :  [0 8 4 2 6 1]
A[indx] :  [1 2 3 4 5 6]
```

in1d(), intersect1d(), setdiff1d(), union1d(), setxor1d()

```
# set () - in, intersect, setdiff, union, setxor

import numpy as np
A = np.array([1, 2, 3, 4])
B = np.array([0, 2, 3])
print("A : ", A)
print("B : ", B)
mask = np.in1d(A, B)
print("mask : ", mask)
C = np.intersect1d(A, B)
print("C : ", C)
D = np.setdiff1d(A, B)
print("D : ", D)
E = np.union1d(A, B)
print("E : ", E)
F = np.setxor1d(A, B)
print("F : ", F)


from functools import reduce
G = reduce(np.union1d, (A, B, [0, 5, 6]))
print("G : ", G)
```

```
A :  [1 2 3 4]
B :  [0 2 3]
mask :  [False  True  True False]
C :  [2 3]
D :  [1 4]
E :  [0 1 2 3 4]
F :  [0 1 4]
G :  [0 1 2 3 4 5 6]
```

배열의 파일 입력 및 출력 관련 유니버설 함수

◆ 배열의 파일 입출력 관련 유니버설 함수

배열 입출력 메소드	설명
load(file, mmap_mode=None, allow_pickle=True, fix_imports=True, encoding='ASCII')	.npy, .npz 또는 pickle 파일로 부터 배열이나 pickle 객체를 읽어 설치.
save(file, arr, allow_pickle=True, fix_imports=True)	배열 arr을 NumPy .npy 포멧의 이진 파일 (binary file)로 저장
savez(file, *args, **kwds)	인수로 전달된 여러 개의 배열을 압축되지 않은 하나의 .npz 포멧의 파일에 저장. 만약 키워드 인수가 전달되지 않으면 파일 이름은 'arr_0', 'arr_1', 등으로 설정됨. 만약 키워드가 지정되는 인수가 전달되면 파일 이름은 주어진 키워드로 설정.
savez_compressed(file, *args, **kwds)	인수로 전달된 여러 개의 배열을 압축된 하나의 .npz 포멧의 파일에 저장. 만약 키워드 인수가 전달되지 않으면 저장되는 파일 이름은 'arr_0', 'arr_1', 등으로 설정됨. 만약 키워드가 지정되는 인수가 전달되면 파일 이름은 주어진 키워드로 설정.
loadtxt(fname, dtype=<type 'float'>, comments='#', delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0)	텍스트 파일로 부터 데이터를 읽어 설치. 텍스트 파일의 각 행의 데이터 개수는 동일하여야 함.
savetxt(fname, X, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='#')	배열을 텍스트 파일에 저장.

savetxt(), loadtxt()

```
# ndarray - text file input and output

import numpy as np

A = np.arange(12).reshape(3,4)
print("np array A ; \n", A)
print("Saving np array A ...")
np.savetxt("npArray_A.txt", A)
B = np.loadtxt("npArray_A.txt")
print("Loaded B : \n", B)
```

```
np array A ;
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
Saving np array A ...
Loaded B :
 [[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]]
```

정렬 (sorting) 관련 유니버설 함수

◆ Sorting and Searching

array/ndarray 관련 메소드	설 명
sort(a, axis=-1, kind='quicksort', order=None)	배열을 별도의 공간을 사용하지 않고 정렬 시켜 줌. 축 (axis)은 정렬을 수행하는 기준 축을 의미하며, axis=-1로 설정하는 경우 다차원 배열의 마지막 축을 기준으로 함. kind는 정렬 알고리즘을 정의하며, 별도로 지정되지 않는 경우 quicksort을 사용함. order는 배열의 원소가 다수의 세부 항목을 가질 경우 어떤 항목을 기준으로 정렬할 것인지를 지정.
ndarray.sort(axis=-1, kind='quicksort', order=None)	ndarray 배열을 별도의 공간을 사용하지 않고 정렬 시켜 줌. 축 (axis)은 정렬을 수행하는 기준 축을 의미하며, axis=-1로 설정하는 경우 다차원 배열의 마지막 축을 기준으로 함. kind는 정렬 알고리즘을 정의하며, 별도로 지정되지 않는 경우 quicksort을 사용함. order는 배열의 원소가 다수의 세부 항목을 가질 경우 어떤 항목을 기준으로 정렬할 것인지를 지정.
searchsorted(A, v, side='left', sorter=None)[source]	배열 A에서 원소 v를 찾아 그 위치 인덱스를 반환. 만약 배열 A에 원소 v가 포함되어 있지 않으면, 반환되는 인덱스 앞에 원소 v를 삽입할 때 배열 A가 정렬된 상태를 유지할 수 있는 인덱스를 찾아 반환.
where(condition[, x, y])	조건 condition에 따라 배열 x 또는 y의 원소를 반환. 만약 조건이 만족되면 x로부터, 조건이 만족되지 않으면 y로부터 원소를 반환. 만약 조건만 지정된 경우 condition.nonzero()를 반환.
extract(condition, arr)[source]	배열 arr로부터 지정된 조건 (condition)을 만족하는 원소들을 골라 반환. np.compress(ravel(condition), ravel(arr))과 동일한 결과 생성. 만약 조건 (*condition*)이 부울 대수이면 np.extract()는 arr[condition]과 동일한 결과 생성.

난수 (random number) 관련 유니버설 함수

난수 생성 메소드	설명
seed(seed=None)	난수발생기의 seed를 초기화
rand(d0, d1, ..., dn)	[0, 1) 구간에서 균일 분포의 난수를 발생하여 주어진 크기의 (다차원) 배열로 생성하여 반환. [0, 1) 구간은 0을 포함하나 1은 포함하지 않음.
random(size=None)	[0.0, 1.0) 구간에서 균일 분포를 가지는 실수형 난수를 지정된 size 개수만큼 1차원 배열로 생성하여 반환
uniform(low=0.0, high=1.0, size=None)	[low, high) 구간에서 균일 분포의 난수를 size개수 만큼 발생하여 반환. [low, high) 구간은 low는 포함하나 high는 포함하지 않음.
randint(low, high=None, size=None)	[low, high] 구간의 np.int 자료형의 정수형 난수 size개수를 생성하여 반환. 만약 high가 설정되어 있지 않으면 [1, low] 구간에서 난수 발생. [low, high] 구간은 low와 high를 모두 포함.
randn(d0, d1, ..., dn)	평균 0, 표준편차 1인 표준 정규 분포 ("standard normal" distribution)의 실수형 (float) 난수를 지정된 크기의 배열로 생성하여 반환. 만약 인수가 전달되지 않으면 실수형 난수 1개가 생성되어 반환.
normal(loc=0.0, scale=1.0, size=None)	평균값 loc, 분산 scale로 설정된 정규 (가우시안) 분포를 가지는 size개의 난수를 생성하여 반환.
multivariate_normal(mean, cov[, size])	평균 mean, 공분산 (covariance) cov인 다변량 정규분포 (multivariate normal distribution)를 가지는 난수를 size개 발생하여 반환.
shuffle(x)	배열 x의 원소들의 위치를 뒤섞어 줌. 다차원 배열의 경우 첫 번째 축 (axis)에 따라 위치를 뒤섞어 줌.
permutation(x)	배열 x의 순서를 뒤섞어 줌. 다차원 배열의 경우 첫 번째 축 (axis)에 따라 순서를 뒤섞어 줌.

통계 분석 관련 유니버설 함수 (1)

통계처리 연산 메소드	설 명
amin(a, axis=None, out=None, keepdims=False)	배열 a의 최솟값을 찾아 반환
amax(a, axis=None, out=None, keepdims=False)	배열 a의 최댓값을 찾아 반환
argmin(a, axis=None, out=None)	배열 a에서 주어진 축 (axis)에 따라 최솟값을 찾고 그 인덱스를 반환
argmax(a, axis=None, out=None)[source]	배열 a에서 주어진 축 (axis)에 따라 최댓값을 찾고 그 인덱스를 반환
sum(a, axis=None, out=None, keepdims=False)	배열 a의 원소들을 주어진 축 (axis)에 따라 합산하여 반환
cumsum(a, axis=None, out=None, keepdims=False)	배열 a의 원소들을 주어진 축 (axis)에 따라 원소들의 누적 합산(cumulative sum)을 계산하여 반환
cumprod(a, axis=None, out=None, keepdims=False)	배열 a의 원소들을 주어진 축 (axis)에 따라 원소들의 누적 곱셈(cumulative product)을 계산하여 반환
median(a, axis=None, out=None, keepdims=False)	배열 a에서 지정된 축 (axis)을 기준으로 중간 (median) 원소를 찾아 반환.
average(a, axis=None, out=None, keepdims=False)	배열 a에서 지정된 축 (axis)을 기준으로 가중치가 고려된 평균 (weighted average)를 계산하여 반환
mean(a, axis=None, dtype=None, out=None, keepdims=False) [source]	배열 a에서 지정된 축 (axis)을 기준으로 산술 평균 (arithmetic mean)을 계산하여 반환.

통계 분석 관련 유니버설 함수 (2)

통계처리 연산 메소드	설 명
var(a, axis=None, dtype=None, out=None, ddof=0, keepdims=False)	배열 a에서 지정된 축 (axis)을 기준으로 분산 (variance)를 계산하여 반환.
std(a, axis=None,dtype=None, out=None, ddof=0, keepdims=False)	배열 a에서 지정된 축 (axis)을 기준으로 표준편차 (standard deviation)를 계산하여 반환.
cov(m, y=None, rowvar=1, bias=0, ddof=None, fweights =None, aweights=None) [source]	주어진 배열 m의 공분산 (covariance)을 계산하여 반환.
histogram(a, bins=10, range=None, normed=False, weights=None, density=None)	주어진 배열 a에서 데이터 집합 (bins)의 도수 분포도 (histogram)을 계산하여 반환.
histogram2d(x, y, bins=10, range=None, normed=False, weights=None, density=None)	주어진 배열 x와 y로부터 2차원 도수 분포도를 계산하여 반환.
histogramdd(sample, bins=10, range=None, normed=False, weights=None)	주어진 샘플 데이터 sample에 대한 다차원 도수 분포도를 계산하여 반환.

```
# Benchmark Test of myQuickSort and NumPy sort (default quick sort) (1)
import time, sys, random
from array import *
import numpy as np

def printArraySample(arr, per_line = 10, sample_lines = 2):
    . . . .
def _partition(arr, left, right, pivot):
    . . . .
def quickSort(arr):
    . . . .

def main():
    while True :
        array_size = int(input("array_size = "))
        if array_size == 0:
            break

        # np.sort() for numpy array
        A = np.arange(array_size)
        print("\nArray created by numpy : ")
        np.random.shuffle(A)
        print("After shuffling by np.random.shuffle, before numpy.sort() : ")
        printArraySample(A, 10, 2)
        t1 = time.time()
        A_Sorted = np.sort(A)
        t2 = time.time()
        elapsedtime = t2 - t1
        print("\nAfter numpy.sort() : ")
        printArraySample(A_Sorted, array_size, 10, 2)
        print('total elapsed time of numpy.sort() for {} integers : {} [sec] '.format(len(A), elapsedtime))
```

```
# Benchmark Test of myQuickSort and NumPy sort (default quick sort)  (2)

        # myQuickSorting for numpy array
        np.random.shuffle(A)
        print("\nAfter shuffling & before myQuickSorting of array A (NumPy ndarray): ")
        printArraySample(A, 10, 2)
        print('my Quick sorting random integer Array (NumPy ndarray) ....')

        t1 = time.time()
        quickSort(A)
        t2 = time.time()
        elapsedtime = t2 - t1
        print("\nAfter QuickSorting : ")
        printArraySample(A, 10, 2)
        print('total elapsed time of my Quick Sort to sort {} integers : {} [sec] '.\
            format(len(A), elapsedtime))


if __name__ == "__main__":
    main()
```

```
array_size = 1000000

Array created by numpy :
After shuffling by np.random.shuffle, before numpy.sort() :
 597987  497397   94057  659317  795071  633310  737875  515234  158830  548427
 601177  309466  749792  423508  691874  494978  555646  639091  209298  184636
. . . . . .
 194519  618820  199711  874974  284410  658217  631367  742864  379930   54677
 118978  824871  471374  283334  387172  533472  572545  957545  993324  933614

After numpy.sort() :
      0       1       2       3       4       5       6       7       8       9
     10      11      12      13      14      15      16      17      18      19
. . . . . .
 999980  999981  999982  999983  999984  999985  999986  999987  999988  999989
 999990  999991  999992  999993  999994  999995  999996  999997  999998  999999
total elapsed time of numpy.sort() for 1000000 integers : 0.052886247634887695 [sec]

After shuffling & before myQuickSorting of array B (NumPy ndarray):
 276539  587370  367834  498564  453748  632310  178746  569326  782764  344142
 598868  296567  787376  890258  912781  607961  861690  136584  684227  209682
. . . . . .
  86760  256265  955272  202581  265350  709862  681706  940827  487230  845220
 559817  796685   94432  996058  881252  925060  233136  950046   44730   96078
my Quick sorting random integer Array (NumPy ndarray) ....

After QuickSorting :
      0       1       2       3       4       5       6       7       8       9
     10      11      12      13      14      15      16      17      18      19
. . . . . .
 999980  999981  999982  999983  999984  999985  999986  999987  999988  999989
 999990  999991  999992  999993  999994  999995  999996  999997  999998  999999
total elapsed time of my Quick Sort to sort 1000000 integers : 10.287463426589966 [sec]
```

선형대수 (linear algebra) 관련
유니버설 함수 (Universal Function, ufunc)

선형대수 관련 유니버설 함수

선형대수 연산 메소드	설명
dot(a, b, out=None)	• 배열 a와 b의 곱셈 (dot product) 결과를 반환 • 만약 a와 b가 1차원 배열이면 벡터의 내적에 해당함 • 만약 a와 b가 2차원 배열이면 행렬 곱셈 matmul() 함수 (a @ b)의 사용을 권장함 • 만약 a와 b 중 하나가 배열이 아닌 스칼라 값이면 numpy.multiply(a, b) 함수 (a * b)의 사용을 권장함 • 만약 a가 N-차원 배열이며, *b*가 1-차원 배열이면 a의 마지막 축 (axis)을 기준으로 배열 a와 b의 원소끼리 곱셈을 한 후 합산 • 만약 a가 N-차원 배열이며 *b* 가 M-차원 배열 (단, M ≥ 2) 인 경우, 배열 a의 마지막 축과 배열 b의 2~마지막 축을 기준으로 곱셈을 한 후 합산: dot(a, b)[i, j, k, m] = sum(a[i, j,:] * b[k, :,m])
vdot(a, b)	• 두 벡터 a, b의 곱셈 (dot product)결과를 반환. • vdot(a, b) 함수는 dot(a, b) 함수와 달리 복소수 계산을 제공함. • 만약 첫 번째 인수 a가 복소수인 경우 곱셈 계산에서 a의 켤레복소수를 사용함. • vdot은 행렬의 곱셈 기능을 제공하지 않으며, 다차원 배열의 인수를 1차원으로 변환한 후 곱셈을 수행함 • 따라서 vdot() 함수는 벡터에 대해서만 사용하여야 함.
inner(a, b)	• 두 배열 a, b의 내적 (inner product)을 계산 및 반환 • 1차원 배열에 대한 곱셈을 수행하며, 복소수 계산을 하지 않음 • 다차원 배열의 경우 마지막 축을 기준으로 곱셈-합산 (sum product)를 수행함
outer(a, b)	• 두 벡터의 외적 (outer product)을 수행

선형대수 관련 유니버설 함수

선형대수 연산 메소드	설명
matmul(a, b, out=None)	• 두 배열의 행렬 곱셈을 수행 • 만약 a, b가 모두 2-차원 배열인 경우, 행렬 곱셈을 계산 • 만약 a, b 중 어느 하나라도 N-차원 배열 (N >2)인 경우, 마지막 두 인덱스를 기반으로 한 행렬이 쌓여 있는 것으로 간주함 • 만약 첫 번째 인수가 1차원 배열이면 차원에 1을 더하여 행렬 (2차원)으로 만들어 곱셈을 계산한 후 추가된 1을 제거함 • 만약 두 번째 인수가 1차원 배열이면 그 차원에 1을 더하여 행렬 (2차원)으로 만들어 곱셈을 계산한 후 추가된 1을 제거함 • 스칼라에 대한 곱셈을 허용하지 않으며, 대신 * 연산자를 사용하여야 함
cholesky(a)	• 정방행렬 a의 숄레스키 분해 (Cholesky decomposition) 연산 결과를 반환
qr(A, mode= 'reduced')	• 행렬 A의 qr 벡터화를 수행하며, orthonormal인 q 와 upper-triangular인 r을 반환
svd(A, full_matrices=1, compute_uv=1)	• 특이값 (singular value) 분해
eig(A)	• 정방행렬 A의 eigenvalues와 right eigenvectors를 계산
det(A)	• 행렬 A의 행렬식 (determinant)을 계산
inv(A)	• 행렬 A의 역행렬 (inverse)을 계산 • 계산된 역행렬 ainv는 dot(A, ainv) = dot(ainv, A) = eye(A.shape[0])의 특성을 만족함
pinv(A, rcond=le-15)	• 행렬 A의 (Moore-Penrose) pseudo-inverse을 계산
solve(A, B)	• 선형 방정식 AX = B의 해 X를 산출함
lstsq(A, B, recond=-1)	• 선형 행렬 공식 AX = B 에 대한 최소 자승해 (least-squares solution) X 를 산출. 즉, ‖ B - AX ‖^2 의 값이 최소가 되는 해 X를 산출

dot(), vdot()

```
# universal function for linear algebra - dot(), vdot()

import numpy as np

A = np.array([1, 2, 3])
B = np.array([4, 5, 6])
C = np.dot(A, B)
print("A = ", A)
print("B = ", B)
print("C = np.dot(A, B) = ", C)

X = np.array([1+2j, 3+4j])
Y = np.array([5+6j, 7+8j])
XY = np.vdot(X, Y) # in vdot, X's conjugate is used
YX = np.vdot(Y, X) # in vdot Y's conjugate is used
print("X = ", X)
print("Y = ", Y)
print("np.vdot(X, Y) = ", XY)
print("np.vdot(Y, X) = ", YX)
```

```
A =  [1 2 3]
B =  [4 5 6]
C = np.dot(A, B) =  32
X =  [1.+2.j 3.+4.j]
Y =  [5.+6.j 7.+8.j]
np.vdot(X, Y) =  (70-8j)
np.vdot(Y, X) =  (70+8j)
```

inner(), outer(), matmul()

```
# universal function for linear algebra - inner(), outer()

import numpy as np

A = np.array([1, 2, 3])
B = np.array([4, 5, 6])
C = np.inner(A, B)
print("A = ", A)
print("B = ", B)
print("np.inner(A, B) = ", C)

D = np.outer(A, B)
print("np.outer(A, B) = \n", D)

E= np.matmul(A, B)
print("np.matmul(A, B) = ", E)

X = np.array([[1, 2], [3, 4]])
Y = np.array([[5, 6], [7, 8]])
W = np.inner(X, Y)
print("X = \n", X)
print("Y = \n", Y)
print("np.inner(X, Y) = \n", W)

Z = np.outer(X, Y)
print("np.outer(X, Y) = \n", Z)

M= np.matmul(X, Y)
print("np.matmul(X, Y) = \n", M)
```

```
A =  [1 2 3]
B =  [4 5 6]
np.inner(A, B) =  32
np.outer(A, B) =
 [[ 4  5  6]
 [ 8 10 12]
 [12 15 18]]
np.matmul(A, B) =  32
X =
 [[1 2]
 [3 4]]
Y =
 [[5 6]
 [7 8]]
np.inner(X, Y) =
 [[17 23]
 [39 53]]
np.outer(X, Y) =
 [[ 5  6  7  8]
 [10 12 14 16]
 [15 18 21 24]
 [20 24 28 32]]
np.matmul(X, Y) =
 [[19 22]
 [43 50]]
```

```
1x4 + 2x5 + 3x6 = 32
```

```
[[1x4, 1x5, 1x6],
 [2x4, 2x5, 2x6],
 [3x4, 3x5, 3x6]]
```

```
1x4 + 2x5 + 3x6 = 32
```

```
[[(1x5+2x6), (1x7+2x8)],
 [(3x5+4x6), (3x7+4x8)]]
```

```
[[1x5, 1x6, 1x7, 1x8],
 [2x5, 2x6, 2x7, 2x8],
 [3x5, 3x6, 3x7, 3x8],
 [4x5, 4x6, 4x7, 4x8]]
```

```
[[1x5+2x7, 1x6+2x8],
 [3x5+4x7, 3x6+4x8]]
```

inv(), det()

```
# linear algebra - inv(), det()

import numpy as np
A = np.array([[1,4, 1], [1, 6, -1], [2, -1, 2]])
print("A : \n", A)
d = np.linalg.det(A)
print("d (determinent of A) : ", d)

inv_A = np.linalg.inv(A)
print("inv_A : \n", inv_A)

B = np.matmul(A, inv_A)
print("B : \n", B)
```

```
A :
 [[ 1  4  1]
 [ 1  6 -1]
 [ 2 -1  2]]
d (determinent of A) :  -17.999999999999996
inv_A :
 [[-0.61111111  0.5         0.55555556]
 [ 0.22222222  0.         -0.11111111]
 [ 0.72222222 -0.5        -0.11111111]]
B :
 [[ 1.00000000e+00  0.00000000e+00 -1.38777878e-17]
 [ 2.22044605e-16  1.00000000e+00 -4.16333634e-17]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

선형 시스템 (Linear System)의 해(Solution) 산출

◆ 선형시스템과 연립방정식의 예

- ***A* × *X* = *B***

$$\begin{aligned} x_1 &+ x_2 &+ x_3 &= 4 \\ 2x_1 &+ 3x_2 &+ x_3 &= 9 \\ x_1 &- x_2 &- x_3 &= -2 \end{aligned}$$

계수행렬 (coefficient matrix)

미지수: Solutions to be calculated

- $A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 1 \\ 1 & -1 & -1 \end{bmatrix}$, $X = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$, $B = \begin{bmatrix} 4 \\ 9 \\ -2 \end{bmatrix}$

◆ 선형 시스템의 해 (solution) 산출

- 첨가행렬 (augmented matrix)과 Gauss-Jordan 소거법
- 계수 행렬 *A*의 역행렬 (A^{-1}) 계산 => $X = A^{-1} \times B$

◆ Linear algebra – solve()

```
# universal function for linear algebra - solve()

import numpy as np

A = np.array([[1, 1, 1], [2, 3, 1], [1, -1, -1]])
B = np.array([4, 9, -2])
X = np.linalg.solve(A, B)
B1 = np.matmul(A, X)

print("A = \n", A)
print("B = ", B)
print("X = np.linalg.solve(A)")
print("X = ", X)
print("B1 = np.matmul(A, X) = ", B1)
```

$A \times X = B$

$$\begin{aligned} x_1 &+ x_2 &+ x_3 &= 4 \\ 2x_1 &+ 3x_2 &+ x_3 &= 9 \\ x_1 &- x_2 &- x_3 &= -2 \end{aligned}$$

계수행렬 (coefficient matrix)

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 1 \\ 1 & -1 & -1 \end{bmatrix}$$

미지수: Solutions to be calculated

$$X = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \quad B = \begin{bmatrix} 4 \\ 9 \\ -2 \end{bmatrix}$$

```
A =
 [[ 1  1  1]
 [ 2  3  1]
 [ 1 -1 -1]]
B =  [ 4  9 -2]
X = np.linalg.solve(A)
X =  [1. 2. 1.]
B1 = np.matmul(A, X) =  [ 4.  9. -2.]
```

Homework 10

Homework 10.1

10.1 NumPy normal()

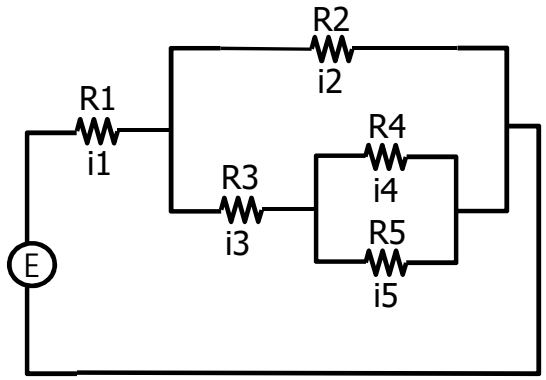
- NumPy의 normal(mu, sigma, size) 함수를 사용하여 정규분포 (normal distribution)의 난수를 10000개 생성하여 ndarray G저장하라. 이 배열 G에 포함된 배열 원소들의 통계 분석을 위하여 유니버설 함수인 mean(), var(), std()를 사용하고, 그 결과를 normal() 함수의 난수 배열 생성에서 사용한 mu과 sigma의 값과 어떤 관계가 있는지 확인하라.
- (실행결과)

```
mu and sigma(in float): 0.0 3.0
size of array : 10000
mu = 0.0, sigma = 3.0
Sample of G = np.random.normal(0.0, 3.0, 10000) =
  -0.25   -5.80    0.54   -8.62    1.75    1.49   -2.07   -4.32    3.64   -0.77
  -2.96   -2.75   -0.01   -0.58   -1.24   -5.64   -2.96    2.18   -1.86    0.28
   0.05   -8.22    2.53    0.35    4.62   -8.01   -1.21   -1.22   -1.19    1.99
     . . . . . .
  -1.24   -0.82    5.70   -4.19   -2.19   -1.30    1.42    0.28    1.79   -1.24
   4.32   -0.33   -1.70   -2.93   -4.38    1.69   -1.20   -4.14    1.25   -2.27
  -0.14   -1.13    1.40   -5.19    0.36    1.60    2.95   -4.78   -1.95   -0.39
mean of G =  0.0038243740303382325
var of G =  9.175296027158913
std of G =  3.029075110848015

mu and sigma(in float): 10.0 5.0
size of array : 100000
mu = 10.0, sigma = 5.0
Sample of G = np.random.normal(10.0, 5.0, 100000) =
   9.80    5.54    2.40   13.44   12.35   13.34   14.39   11.55   14.69    6.00
   9.81   13.73    2.65   21.89    5.35    8.47    3.01   14.33    8.88   15.17
   8.71   10.72   14.00   19.58    3.90    5.15   14.19   15.11    8.69   19.69
     . . . . . .
   4.57   13.53    8.16   13.20   21.56    3.39   15.09    6.57    7.86   11.75
   1.48    9.34    7.93   10.10    4.61   12.05   11.16   15.97   -0.08   10.37
   9.65   10.82   -7.00   13.66   14.15   17.85    8.23   13.58   11.36   17.63
mean of G =  10.004056512923496
var of G =  24.94104174510298
std of G =  4.994100694329559
```

Homework 10.2

10.2 NumPy 기반 선형시스템 해 산출

- 직병렬 전자회로의 선형시스템 해 (solution)를 NumPy 선형대수 유니버설함수 solve()를 사용하여 구하라.
- 아래 그림에서 보는 것과 같은 5개 저항의 직•병렬 전자회로에서 각 저항을 통하여 흐르는 전류의 양을 계산하는 선형 연립방정식을 정리하고, 정리된 연립방정식의 계수 행렬 A을 NumPy 2차원 배열로 구성하라.
- 선형방정식의 A•X = B에서 B를 NumPy 1차원 배열로 구성하라.
- NumPy 패키지의 선형대수관련 유니버설 함수인 solve()를 사용하여 X를 계산하고, A, B와 X를 출력하라.
- NumPy의 행렬 곱셈 함수인 matmul(A, X)를 사용하여 A•X = B 인 것을 증명하라.
- 또한 계수 행렬 A의 역행렬 A^{-1}을 구하고, A^{-1}•B가 X인 것을 증명하라.



E = 100 V
R1 = 10 Ω
R2 = 10 Ω
R3 = 5 Ω
R4 = 10 Ω
R5 = 10 Ω

```
A =
 [[ 10  10   0   0   0]
 [  1  -1  -1   0   0]
 [  0   0   1  -1  -1]
 [  0  10  -5 -10   0]
 [  0   0   0  10 -10]]
B =  [100   0   0   0   0]
X =  [6.66666667 3.33333333 3.33333333 1.66666667 1.66666667]
np.matmul(A, X) =  [ 1.00000000e+02  8.88178420e-16 -4.44089210e-16 -3.55271368e-15
  0.00000000e+00]
```