```
from numpy.random import randn
import numpy as np
```

np.random

> - random.randint 와 np.random.randint : 모두 (시작, n-1) 사이의 랜덤숫자 1개 뽑아내기
> - np.random.rand(m, n) : 0~1의 균일분포 표준정규분포 난수를 matrix array(m, n) 생성
> - np.random.randn(m, n) : 평균0, 표준편차1의 가우시안 표준정규분포 난수를 matrix array(m, n) 생성

```
arr = np.arange(10)
arr
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.sqrt(arr)   # 제곱근 연산
```

```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

```
np.exp(arr)    #밑(base)이 자연상수 e 인 지수함수 로 변환
```

```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

```
x = randn(8)
x
```

```
array([-1.33604152, -1.43098019,  1.5243914 , -1.86356175, -0.6680757 ,
       -1.73475534,  0.94898067,  0.39838582])
```

```
y = randn(8)
y
```

```
array([ 0.43957606, -0.40174392, -0.88425385, -0.24423469,  0.23363954,
        2.12845638,  0.69213179, -0.25266014])
```

```
np.maximum(x,y)
```

```
array([ 0.43957606, -0.40174392,  1.5243914 , -0.24423469,  0.23363954,
        2.12845638,  0.94898067,  0.39838582])
```

```
arr = randn(7)
arr
```

```
array([ 2.30361379,  0.5859839 , -0.99470659,  0.6088872 ,  1.19567929,
        1.11810103,  0.25355478])
```

그 외 난수 생성 방식

np.random.rand(m,n)

0~1의 균일분포 (표준정규분포) 난수를 matrix array(m, n)으로 생성한다.

np.random.randn(m, n)

평균은 0. 표준편차가 1인 가우시안 표준정규분포 난수를 matrix array(m, n)으로 생성한다.

```
arr*5
```

```
array([11.51806897,  2.92991952, -4.97353293,  3.04443599,  5.97839645,
        5.59050513,  1.2677739 ])
```

```
np.modf(arr)   #실수와 정수 함께 반환
```

```
(array([ 0.30361379,  0.5859839 , -0.99470659,  0.6088872 ,  0.19567929,
         0.11810103,  0.25355478]), array([ 2.,  0., -0.,  0.,  1.,  1.,  0.]))
```

```
points = np.arange(-5, 5, 0.01) # -5부터 5까지 0.01 단위의 값 적용
points
```

```
xs, ys = np.meshgrid(points,points) # grid 형태로 뿌려주기
```

```
xs
```

```
array([[-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       ...,
       [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99],
       [-5.  , -4.99, -4.98, ...,  4.97,  4.98,  4.99]])
```

```
ys
```

```
array([[-5.  , -5.  , -5.  , ..., -5.  , -5.  , -5.  ],
       [-4.99, -4.99, -4.99, ..., -4.99, -4.99, -4.99],
       [-4.98, -4.98, -4.98, ..., -4.98, -4.98, -4.98],
       ...,
       [ 4.97,  4.97,  4.97, ...,  4.97,  4.97,  4.97],
       [ 4.98,  4.98,  4.98, ...,  4.98,  4.98,  4.98],
       [ 4.99,  4.99,  4.99, ...,  4.99,  4.99,  4.99]])
```

```
z = np.sqrt(xs **2 + ys ** 2)
```

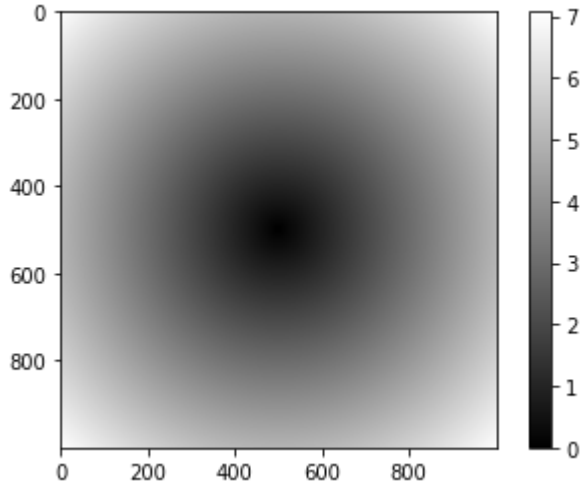
```
z
```

```
array([[7.07106781, 7.06400028, 7.05693985, ..., 7.04988652, 7.05693985,
        7.06400028],
       [7.06400028, 7.05692568, 7.04985815, ..., 7.04279774, 7.04985815,
        7.05692568],
       [7.05693985, 7.04985815, 7.04278354, ..., 7.03571603, 7.04278354,
        7.04985815],
       ...,
       [7.04988652, 7.04279774, 7.03571603, ..., 7.0286414 , 7.03571603,
        7.04279774],
       [7.05693985, 7.04985815, 7.04278354, ..., 7.03571603, 7.04278354,
        7.04985815],
       [7.06400028, 7.05692568, 7.04985815, ..., 7.04279774, 7.04985815,
        7.05692568]])
```

```
from matplotlib.pyplot import imshow, title
import matplotlib.pyplot as plt
```

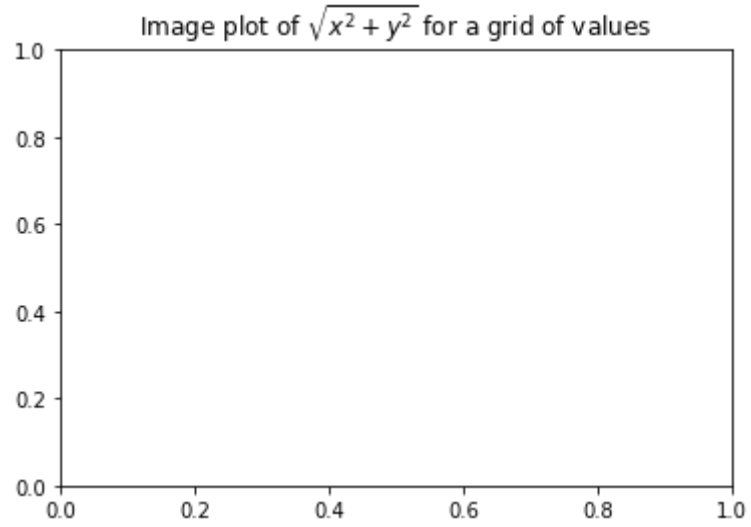
```
plt.imshow(z, cmap = plt.cm.gray)
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x1ce6be9c400>
```



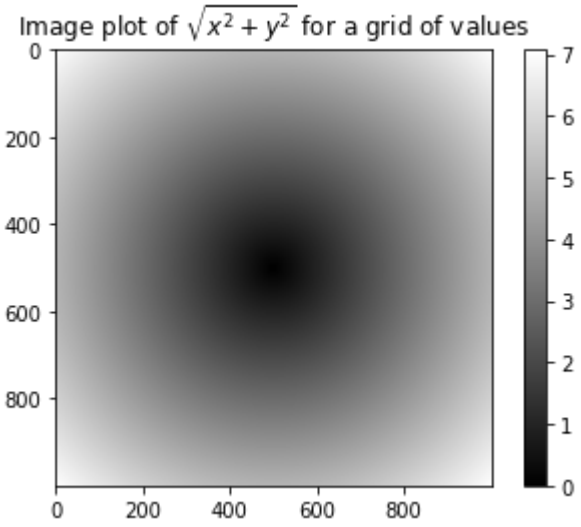

```
plt.title("Image plot of $Wsqrt{x^2 + y^2}$ for a grid of values")

## $W를 붙여 수식형태로 나타냄
```

```
Text(0.5, 1.0, 'Image plot of $WWsqrt{x^2 + y^2}$ for a grid of values')
```




```
plt.imshow(z, cmap = plt.cm.gray)
plt.colorbar()
plt.title("Image plot of $Wsqrt{x^2 + y^2}$ for a grid of values")
plt.draw
```

```
<function matplotlib.pyplot.draw()>
```




```
from IPython.display import Image
```

```
Image('./data/images/1.png')
```

Table 4-3. Unary ufuncs

Function	Description
`abs, fabs`	Compute the absolute value element-wise for integer, float Use `fabs` as a faster alternative for non-complex-valued d
`sqrt`	Compute the square root of each element. Equivalent to ar
`square`	Compute the square of each element. Equivalent to `arr *`
`exp`	Compute the exponent e^x of each element
`log, log10, log2, log1p`	Natural logarithm (base *e*), log base 10, log base 2, and log
`sign`	Compute the sign of each element: 1 (positive), 0 (zero), or
`ceil`	Compute the ceiling of each element, i.e. the smallest integ each element
`floor`	Compute the floor of each element, i.e. the largest integer element
`rint`	Round elements to the nearest integer, preserving the dty
`modf`	Return fractional and integral parts of array as separate arr
`isnan`	Return boolean array indicating whether each value is NaN
`isfinite, isinf`	Return boolean array indicating whether each element is fi infinite, respectively
`cos, cosh, sin, sinh,` `tan, tanh`	Regular and hyperbolic trigonometric functions
`arccos, arccosh, arcsin,` `arcsinh, arctan, arctanh`	Inverse trigonometric functions
`logical_not`	Compute truth value of `not x` element-wise. Equivalent t

배열 연산으로 조건절 표현하기

```
xarr = np.array([1.4,1.6,1.3,1.2,1.8])
yarr = np.array([2.5,2.1,2.7,2.9,2.3])
cond = np.array([True, True, False, True, False])
```

```
result = [(x if c else y)
          for x,y,c in zip(xarr,yarr,cond)]
result
```

```
[1.4, 1.6, 2.7, 1.2, 2.3]
```

```
list(zip(xarr,yarr,cond))
```

```
[(1.4, 2.5, True),
 (1.6, 2.1, True),
 (1.3, 2.7, False),
 (1.2, 2.9, True),
 (1.8, 2.3, False)]
```

```
result = np.where(cond,xarr,yarr)   # cond가 T이면 xarr을, F면 yarr를 반환함
result
```

```
array([1.4, 1.6, 2.7, 1.2, 2.3])
```

```
arr = randn(4,4)
arr
```

```
array([[ 0.71549416, -0.33381476, -1.49285991, -0.34710302],
       [-0.15244375,  1.5896166 ,  1.1433679 , -0.25381219],
       [-0.27148567, -1.98014051, -0.74293165,  0.26185595],
       [-0.1387702 , -0.83582674,  1.2648559 ,  0.16237678]])
```

```
np.where(arr > 0, 2, -2)
```

```
array([[ 2, -2, -2, -2],
       [-2,  2,  2, -2],
       [-2, -2, -2,  2],
       [-2, -2,  2,  2]])
```

```
np.where(arr > 0, 2, arr)
```

```
array([[ 2.        , -0.33381476, -1.49285991, -0.34710302],
       [-0.15244375,  2.        ,  2.        , -0.25381219],
       [-0.27148567, -1.98014051, -0.74293165,  2.        ],
       [-0.1387702 , -0.83582674,  2.        ,  2.        ]])
```

- where에 넘긴 배열은 같은 크기의 배열이거나 스칼라 값일 수 있다.
- where를 사용하면 좀더 복잡한 연산을 수행할 수 있다.

```
arr = np.random.randn(5,4)
```

```
arr
```

```
array([[ 0.74651815, -0.36074977,  0.08308348, -2.00991455],
       [ 0.26324886, -0.93289266,  0.57367506,  0.61422071],
       [ 0.48869654, -0.36147666,  0.25267069,  0.21150471],
       [-0.53023694,  0.82253253, -0.00561634,  0.47410166],
       [ 0.60919032,  2.04995332,  0.28141199, -2.58672351]])
```

```
arr.mean()
```

0.034159879406837

```
np.mean(arr)
```

0.034159879406837

```
arr.sum()  #전체
```

0.68319758813674

```
arr.mean(axis = 1)
```

array([-0.38526567, 0.12956299, 0.14784882, 0.19019523, 0.08845803])

```
arr.sum(0)
```

array([1.57741693, 1.21736675, 1.18522488, -3.29681098])

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

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

```
arr.cumsum(0)   #각 원소의 누적 합
```

```
array([[ 0,  1,  2],
       [ 3,  5,  7],
       [ 9, 12, 15]], dtype=int32)
```

```
arr.cumprod(1)  #각 원소의 누적 곱
```

```
array([[  0,   0,   0],
       [  3,  12,  60],
       [  6,  42, 336]], dtype=int32)
```

불리언 타입 배열을 위한 메소드

- any() : 하나 이상의 True값이 있는지 검사
- all() : 모든 원소가 True인지 검사

```
arr = randn(100)
```

```
(arr>0).sum()
```

43

```
bools = np.array([False,False,True,False])
bools
```

array([False, False, True, False])

```
bools.any()
```

True

```
bools.all()
```

False

정렬. Sorting

```
arr = randn(8)
arr
```

array([-1.49621834, -0.85870564, 0.08561622, -0.15862364, -0.86861063,
       -0.61472611, 0.51700543, -0.33425993])

```
arr.sort()
arr
```

array([-1.49621834, -0.86861063, -0.85870564, -0.61472611, -0.33425993,
       -0.15862364, 0.08561622, 0.51700543])

```
arr = randn(5,3)
arr
```

array([[-1.77145921, -1.71355482, 1.4656097],
       [0.8724959 , 1.66264347, 0.12960429],
       [1.86647271, 1.66098921, -0.26772335],
       [1.24323217, 1.60394781, 1.37331434],
       [-0.46389388, 0.23748021, -1.47376623]])

```
arr.sort(1)
```

```
arr
```

```
array([[-1.77145921, -1.71355482,  1.4656097 ],
       [ 0.12960429,  0.8724959 ,  1.66264347],
       [-0.26772335,  1.66098921,  1.86647271],
       [ 1.24323217,  1.37331434,  1.60394781],
       [-1.47376623, -0.46389388,  0.23748021]])
```

```
large_arr = randn(1000)
large_arr
```

```
large_arr.sort()
large_arr
```

```
large_arr[int(0.05 * len(large_arr))]   # 5%구간
```

-1.7015619050297095

집합함수

```
names = np.array(['Bob','Joe','Will','Bob','Will','Joe','Joe'])
names
```

array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')

```
np.unique(names)
```

array(['Bob', 'Joe', 'Will'], dtype='<U4')

```
ints = np.array([3,3,3,2,2,1,1,4,4,])
```

```
np.unique(ints)
```

array([1, 2, 3, 4])

```
values= np.array([6,0,0,3,2,5,6])
```

```
values
```

array([6, 0, 0, 3, 2, 5, 6])

```
np.in1d(values, [2,3,6])   #in : 불리언 형태로 반환
```

array([True, False, False, True, True, False, True])