파이션 Numpy

충북대학교 소프트웨어학과
류관희

목 차

01

파이션 제어문 및 함수

- 제어문(조건문, 반복문)
- 함수

02

Numpy 설치 및 배열처리

- Numpy 설치 방법
- 수치 데이터 처리 방법

03

Numpy 데이터 생성 및 연산

- 배열 생성 방법
- 배열 모양 변경
- 산술연산 및 관계 연산
- 행렬연산 및 선형 대수함수

학습목표

- 파이션에서 제공하는 다음과 같은 문법을 이해할 수 있다.
  - 제어문(조건문, 반복문)
  - 함수

파이션 제어문

- if, if ~ else, if ~ elif ~ else

```
if 조건문 :                    # 문장은 반드시 단계별로 들여쓰기
  문장                         # pass 사용가능
else :
  문장
```

- while

```
while 조건문 :                 # 조건에 True사용(무한루프)
  문장
```

- for

```
for 변수 in 리스트(투플,문자열 등) :
  문장
for 변수 in range(시작값, 끝값) :          # 끝값은 1 더 큰값
  문장
```

- break, continue

조건문 예제


```
x = int(input("정수 x = "))

if(x % 2 == 1):                # 괄호 생략 가능
    print("홀수입니다.")
else:
    print("짝수입니다.")
```

조건문 예제


```
score = int(input("점수 = "))

if(score >= 90):
    print("A학점")
elif(score >= 80):
    print("B학점")
elif(score >= 70):
    print("C학점")
elif(score >= 60):
    print("D학점")
else: print("F학점")
```

반복문 예제


```
sum = 0
for i in range(1,101):   # range의 끝값 주의
    sum += i
print("1+2+...+100 = ", sum)
print("1+2+...+100 = " + str(sum))
```

range 함수

range(start,stop[,step])

range(stop[,step]) # 0부터 시작

```
score = [23, 45, 67, 43, 12]
sum = 0
for data in reversed(score):
    sum += data
print("자료의 합 =", sum)
```

반복문 예제


```
sum = 0
i = 1
while(i <= 100):
    sum += i
    i += 1
print("1+2+...+100 =", sum)
```

```
sum = 0
i = 1
while(i <= 100):
    sum += i
    i *= 2
print("1+2+4+...+64 =", sum)
```

반복문 예제


```
import random
# 게임을 위한 랜덤 숫자 생성
ans = random.randrange(1, 101, 1)
num = -1
t_cnt = 0 # 시도횟수
print("1~100 숫자 Up & Down 게임을 시작합니다 !!!")
print("-------------------------")
while ( ans != num ):
    num = int(input("1 ~ 100 사이의 숫자를 입력하세요 : "))
    if (num > ans):
        print("Down")
    elif (num < ans):
        print("Up")
    t_cnt += 1
print("-------------------------")
print(t_cnt, "번 만에 정답을 맞추셨습니다.")
```

파이션 함수

함수의 구조

```
def 함수명(입력 인수) :
    수행할 문장1
    수행할 문장2
```

```
def sum(a, b):
    c = a + b
    return c

m = sum(4, 7)
print('m = ',  m)
```

파이션 함수

가변인수

```
def sum_many(*args):
    sum = 0
    for i in args:
        sum += i
    return sum

m = sum_many(1, 2, 3)
print('m = ', m)
m = sum_many(1, 2, 3, 4)
print('m = ', m)
```

문제풀이

- 파이션에서 다음과 같은 제어문을 설명하시오 ?
  - ✓ if, if ~ else, if ~ elif ~ else

- 파이션에서 다음과 같은 반복문을 설명하시오 ?
  - ✓ while 조건문 :
    문장
  - ✓ for 변수 in range(시작값, 끝값) :
    문장

  - ✓ range(1, 100, 2)

요약

- 파이션에서 사용하는 제어문을 공부하였다
  - ✓ if, if ~ else, if ~ elif ~ else

- 파이션에서 사용하는 반복문을 공부하였다.
  - ✓ while 조건문 :
    문장
  - ✓ for 변수 in range(시작값, 끝값) :
    문장
- 파이션에서 사용하는 함수를 공부하였다.

01

파이션 제어문 및 함수

- 제어문(조건문, 반복문)
- 함수

02

Numpy 설치 및 배열처리

- Numpy 설치 방법
- 수치 데이터 처리 방법

03

Numpy 데이터 생성 및 연산

- 배열 생성 방법
- 배열 모양 변경
- 산술연산 및 관계 연산
- 행렬연산 및 선형 대수함수

학습목표

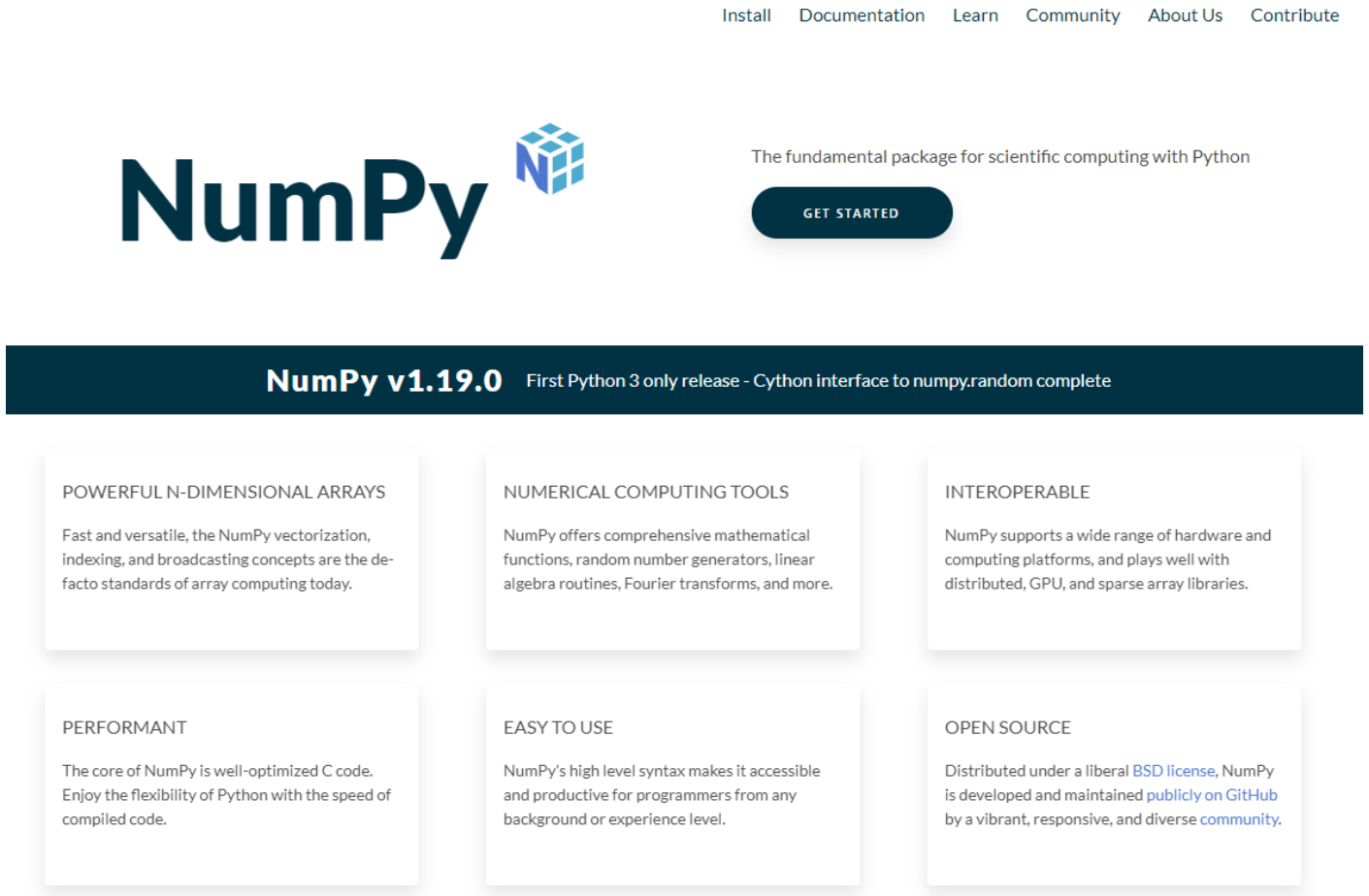
- 파이션에서 제공하는 Numpy 라이브러리를 이해할 수 있다.
  - 설치 방법
  - Numpy 라이브러리를 이용한 수치 데이터 처리 방법

데이터 분석시각화 라이브러리

- Numpy
  - https://numpy.org/
- Scipy
  - https://www.scipy.org/
- Pandas
  - https://pandas.pydata.org/
- Plotly
  - https://plotly.com/
- Matplotlib
  - https://matplotlib.org/

Numpy

- Numpy: Numerical Python
  - https://numpy.org/



Numpy 설치 및 장점

- Conda 환경
  - conda install numpy
- Pip 환경
  - pip install numpy
- 사용방법
  - import numpy as np
- 사용장점
  - C언어로 구현된 저수준 고성능 라이브러리
  - 빠르고 메모리 효율적으로 다차원 배열 ndarray 연산 지원
  - 선형대수, 난수 발생기, 푸리에 변환 등 다양한 연산 기능 지원

Numpy 장점

- 행렬과 벡터를 곱하는 연산(반복문 사용)

In [1]:
```
import numpy as np
import time
```

In [2]:
```
n = 1000
m = 5000
```

In [3]:
```
#X 행렬: 크기 1000 x 5000
#W 행렬: 크기 1000 x 1
#Z = W*X
```

In [4]:
```
X = np.random.rand(n, m)
W = np.random.rand (n,1)
Z = np.zeros((1, m))
```

In [5]:
```
start_time = time.time()
```

In [6]:
```
for i in range (X.shape[1]):
    for j in range (X.shape[0]):
        Z[0][i] += W[j]*X[j][i]
```

In [7]:
```
end_time = time.time()
```

In [8]:
```
print("반복문 이용 실행시간: ", (end_time - start_time)*1000, "ms")
```

```
반복문 이용 실행시간:  16142.804861068726 ms
```

Numpy 장점

- 행렬과 벡터를 곱하는 연산(Numpy 사용)

```
In [8]:  1  print("반복문 이용 실행시간: ", (end_time - start_time)*1000, "ms")
반복문 이용 실행시간:  16142.804861068726 ms

In [9]:  1  start_time = time.time()

In [10]: 1  Z = np.dot(W.T,X)

In [11]: 1  end_time = time.time()

In [12]: 1  print("numpy 이용 실행시간: ", (end_time - start_time)*1000, "ms")
numpy 이용 실행시간:  33.908843994140625 ms
```

- 속도 향상
  - Z = np.dot(W.T, X)
  - Vectorization: 반복문이나 인덱싱 을 사용하지 않고 데이터에 배치 작업 (실제 수행은 C 언어에서 처리)

Numpy 기능

- narray: a (usually fixed-size) multidimensional container of items of the same type and size

속성	설명
ndarray.ndim	배열의 차원(dimension)의 수, 축(axes), 랭크(rank)로도 부름
ndarray.shape	배열의 각 차원에서의 크기를 tuple로 반환
ndarray.size	배열의 요소의 전체 개수
ndarray.dtype	배열 요소의 자료형, 파이썬 기본자료형 또는 numpy.int32, numpy.int16, numpy.float64 등의 자체 자료형
ndarray.itemsize	배열 요소의 바이트 수
ndarray.data	배열 요소를 담고 있는 실제 버퍼, 직접 사용하지 않음

Numpy 사용 예제

In [1]:
```
1 import numpy as np
```

In [2]:
```
1 x = np.float32(1.0)
2 x
```

Out[2]:
```
1.0
```

In [3]:
```
1 y = np.int_([1,2,4])
2 y
```

Out[3]:
```
array([1, 2, 4])
```

In [4]:
```
1 z = np.arange(3, dtype=np.uint8)
2 z
```

Out[4]:
```
array([0, 1, 2], dtype=uint8)
```

In [5]:
```
1 z.dtype
```

Out[5]:
```
dtype('uint8')
```

In [6]:
```
1 w = np.array([1, 2, 3], dtype='f')
2 w
```

Out[6]:
```
array([1., 2., 3.], dtype=float32)
```

Numpy 사용 예제

In [7]:
```
1 x = np.array([2,3,1,0])
2 x
```

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

In [8]:
```
1 x = np.array([[ 1.+0.j, 2.+0.j], [ 0.+0.j, 0.+0.j], [ 1.+1.j, 3.+0.j]])
2 x
```

Out[8]:
```
array([[1.+0.j, 2.+0.j],
       [0.+0.j, 0.+0.j],
       [1.+1.j, 3.+0.j]])
```

In [9]:
```
1 x = np.array([[1,2.0],[0,0],(1+1j,3.)]) # note mix of tuple and lists,and types
2 x
```

Out[9]:
```
array([[1.+0.j, 2.+0.j],
       [0.+0.j, 0.+0.j],
       [1.+1.j, 3.+0.j]])
```

Numpy 사용 예제


```
In [10]:
1 np.zeros((2, 3))
```

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

```
In [11]:
1 np.arange(10)
```

```
Out[11]:
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [12]:
1 np.arange(2, 10, dtype=float)
```

```
Out[12]:
array([2., 3., 4., 5., 6., 7., 8., 9.])
```

```
In [13]:
1 np.arange(2, 3, 0.1)
```

```
Out[13]:
array([2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9])
```

```
In [14]:
1 np.linspace(1., 4., 6)
```

```
Out[14]:
array([1. , 1.6, 2.2, 2.8, 3.4, 4. ])
```

```
In [17]:
1 Z = np.ones(4*1000000, np.float32)
2 len(Z)
```

```
Out[17]:
4000000
```

Numpy 사용 예제

a=vec(4,2,9,3)

```
In [26]:
1 a = np.array([4, 2, 9, 3])
2 a + 1
Out[26]: array([ 5,  3, 10,  4])

In [27]:
1 2**a
Out[27]: array([ 16,   4, 512,   8], dtype=int32)

In [28]:
1 b = np.ones(4) + 5
2 b
Out[28]: array([6., 6., 6., 6.])

In [29]:
1 a - b
Out[29]: array([-2., -4.,  3., -3.])

In [30]:
1 a * b
Out[30]: array([24., 12., 54., 18.])

In [31]:
1 a + b
Out[31]: array([10.,  8., 15.,  9.])

In [32]:
1 j = np.arange(5)
2 j
Out[32]: array([0, 1, 2, 3, 4])

In [33]:
1 2**(j + 1) - j
Out[33]: array([ 2,  3,  6, 13, 28])
```

Numpy 연산자

```
In [42]:
1 c = np.ones((3, 3))
2 c
```

```
Out[42]: array([[1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.]])
```

3x3 행렬 생성(모든 값은 1)

```
In [43]:
1 c*c
```

```
Out[43]: array([[1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.]])
```

C=CxC : multiplication

```
In [44]:
1 c = c.dot(c)
2 c
```

```
Out[44]: array([[3., 3., 3.],
                [3., 3., 3.],
                [3., 3., 3.]])
```

C[i][j]=C[i][j]*C[i][j]

```
In [45]:
1 c*c
```

```
Out[45]: array([[9., 9., 9.],
                [9., 9., 9.],
                [9., 9., 9.]])
```

Numpy 연산자

Comparisons:

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array([4, 2, 2, 4])
>>> a == b
array([False,  True, False,  True])
>>> a > b
array([False, False,  True, False])
```

Array-wise comparisons:

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array([4, 2, 2, 4])
>>> c = np.array([1, 2, 3, 4])
>>> np.array_equal(a, b)
False
>>> np.array_equal(a, c)
True
```

Logical operations:

```
>>> a = np.array([1, 1, 0, 0], dtype=bool)
>>> b = np.array([1, 0, 1, 0], dtype=bool)
>>> np.logical_or(a, b)
array([ True,  True,  True, False])
>>> np.logical_and(a, b)
array([ True, False, False, False])
```

Numpy 연산자


```
In [46]: a = np.arange(5)

In [47]: np.sin(a)
Out[47]: array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ])

In [48]: np.log(a)
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning:
  """Entry point for launching an IPython kernel.
Out[48]: array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436])

In [49]: np.exp(a)
Out[49]: array([ 1.        ,  2.71828183,  7.3890561 , 20.08553692, 54.59815003])
```

Upper Triangulation

```
In [52]: a = np.triu(np.ones((3, 3)), 0)
         a
Out[52]: array([[1., 1., 1.],
                [0., 1., 1.],
                [0., 0., 1.]])
```

```
In [53]: a.T
Out[53]: array([[1., 0., 0.],
                [1., 1., 0.],
                [1., 1., 1.]])
```

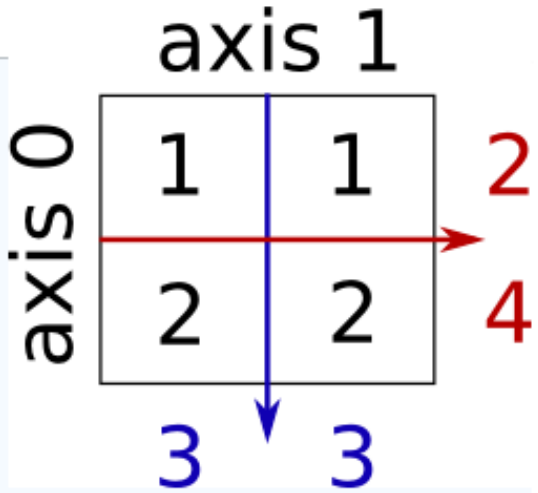
Transpose
전치

Numpy 연산자

```
>>> x = np.array([1, 2, 3, 4])
>>> np.sum(x)
10
>>> x.sum()
10
```

Sum by rows and by columns:

```
>>> x = np.array([[1, 1], [2, 2]])
>>> x
array([[1, 1],
       [2, 2]])
>>> x.sum(axis=0)   # columns (first
    dimension)
array([3, 3])
>>> x[:, 0].sum(), x[:, 1].sum()
(3, 3)
>>> x.sum(axis=1)   # rows (second dimension)
array([2, 4])
>>> x[0, :].sum(), x[1, :].sum()
(2, 4)
```



Same idea in higher dimensions:

```
>>> x = np.random.rand(2, 2, 2)
>>> x.sum(axis=2)[0, 1]
1.14764...
>>> x[0, 1, :].sum()
1.14764...
```

Numpy 함수

```
1 x.min()
```

0.10288979685592559

```
1 x.max()
```

0.6658059906402732

```
1 x.mean()
```

0.39335800898850837

```
1 np.median(x)
```

0.35314737041116734

문제풀이

- Numpy의 full name은 무엇인가?
- Numpy에서 제공하는 다음 함수들의 기능을 설명하시오.
  - arange(2,10)
  - linspace

요약

- 데이터 분석에서 필수적으로 사용되는 Numpy 기본 기능을 소개하였음
- Numpy에서 제공하는 다음과 같은 함수를 소개하였음
  - arange(2,10)
  - linspace

01

파이션 제어문 및 함수

- 제어문(조건문, 반복문)
- 함수

02

Numpy 설치 및 배열처리

- Numpy 설치 방법
- 수치 데이터 처리 방법

03

Numpy 데이터 생성 및 연산

- 배열 생성 방법
- 배열 모양 변경
- 산술연산 및 관계 연산
- 행렬연산 및 선형 대수함수

학습목표

- 파이션에서 제공하는 Numpy 라이브러리를 이해할 수 있다.
  - 배열 생성 방법
  - 배열 모양 변경
  - 산술연산 및 관계연산
  - 행렬연산 및 선형대수함수

배열 생성: eye()


```
# eye(N, M=None, k=0, dtype=<class 'float'>)
>>> E1 = np.eye(3, dtype=int)
>>> E1
array([[1, 0, 0],
[0, 1, 0],
[0, 0, 1]])
>>> E2 = np.eye(3, k=1, dtype=int)
>>> E2
array([[0, 1, 0],
[0, 0, 1],
[0, 0, 0]])
```

```
>>> E3 = np.eye(3, k=-1, dtype=int)
>>> E3
array([[0, 0, 0],
[1, 0, 0],
[0, 1, 0]])
```

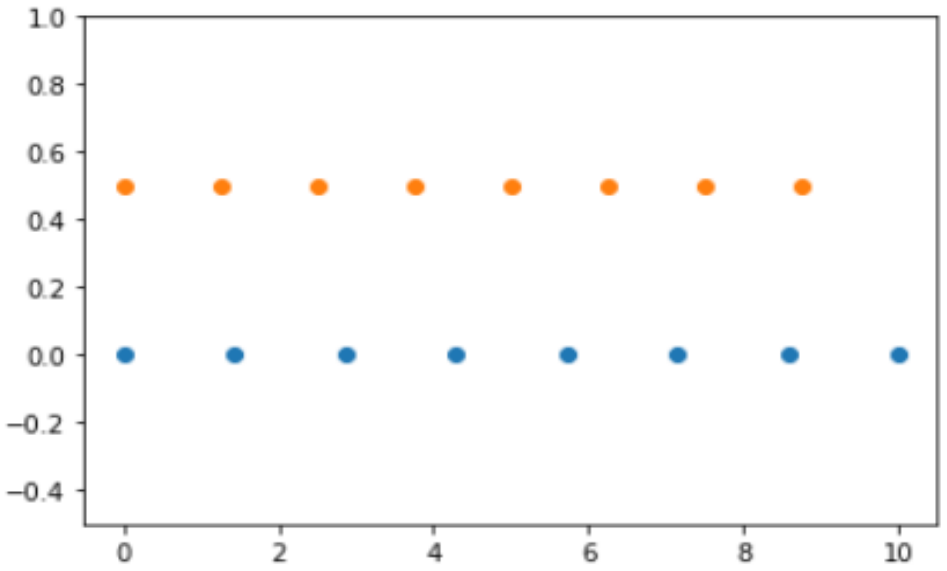
배열생성: linspace()


```
import matplotlib.pyplot as plt
```

```
N = 8
y = np.zeros(N)
```

```
x1 = np.linspace(0, 10, N, endpoint=True)
x2 = np.linspace(0, 10, N, endpoint=False)
```

```
plt.plot(x1, y, 'o')
plt.plot(x2, y + 0.5, 'o')
plt.ylim([-0.5, 1])
plt.show()
```



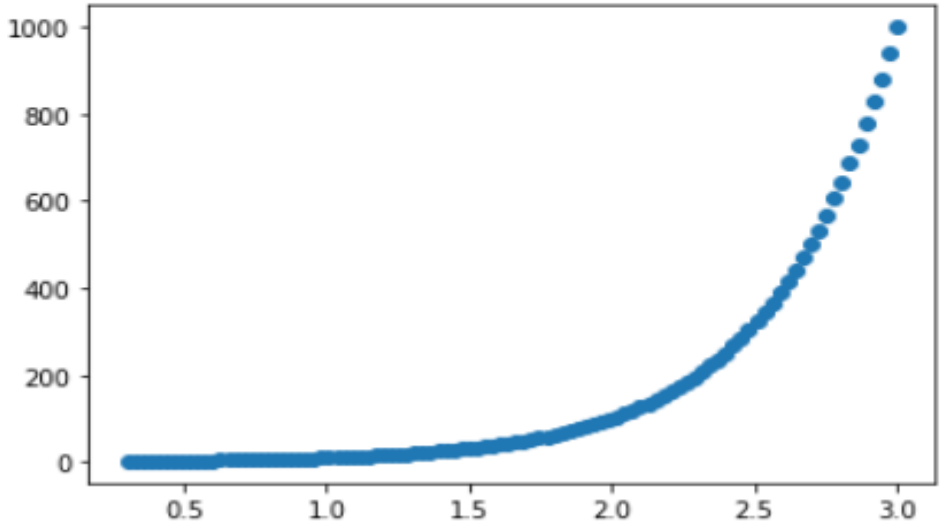
배열생성 : linspace()

>>> import numpy as np

logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)

```
1  x = np.linspace(0.3, 3, 100, endpoint=True)
2  y = np.logspace(0.3, 3, num = 100)
```

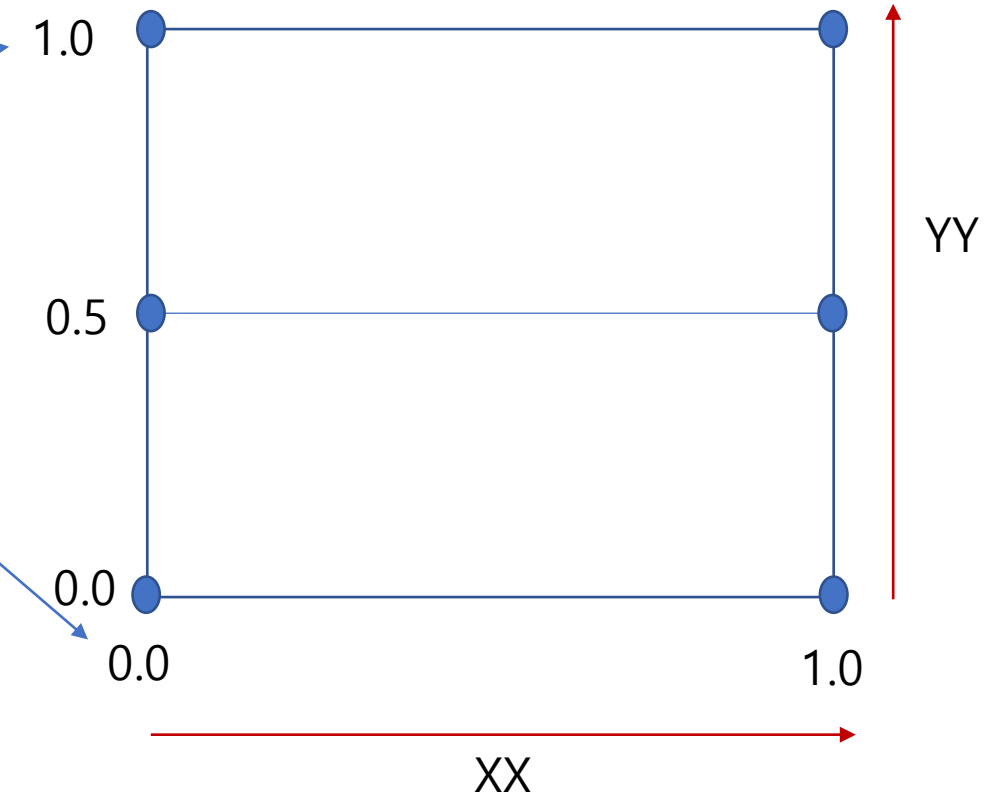
```
1  plt.plot(x, y, 'o')
2  plt.show()
```



배열생성: meshgrid()


```
1  nx, ny = (2, 3)
2  x = np.linspace(0, 1, nx)
3  y = np.linspace(0, 1, ny)
4  xx, yy = np.meshgrid(x, y)
5  print("X:", x, "Y:", y)
6  print("XX:", xx, "₩n YY:", yy)
```

```
X: [0. 1.] Y: [0.  0.5 1. ]
XX: [[0. 1.]
 [0. 1.]
 [0. 1.]]
 YY: [[0.  0. ]
 [0.5 0.5]
 [1.  1. ]]
```

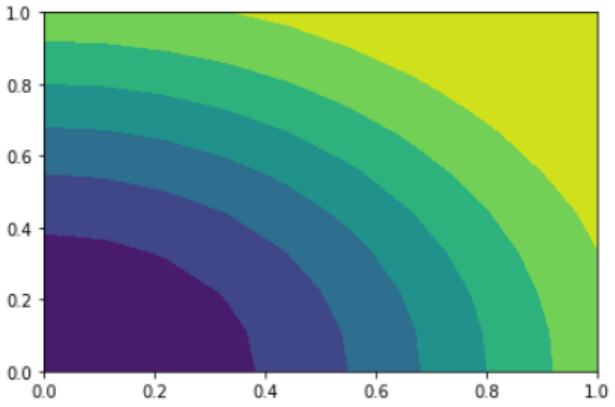


배열생성

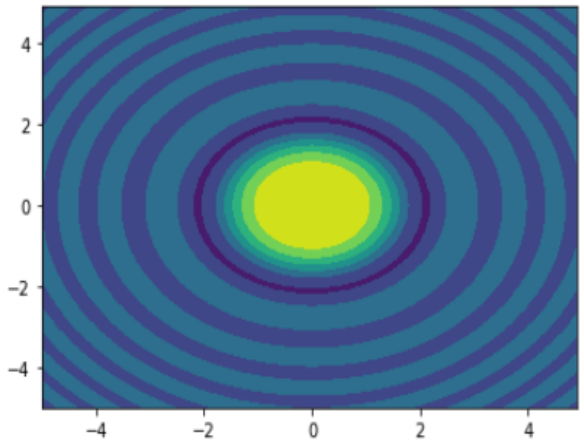
```
nx, ny = (2, 3)
x = np.linspace(0, 1, nx)
y = np.linspace(0, 1, ny)
xx, yy = np.meshgrid(x, y)
print("X:", x, "Y:", y)
print("XX:", xx, "₩n YY:", yy)
```

```
X: [0. 1.] Y: [0.  0.5 1. ]
XX: [[0. 1.]
 [0. 1.]
 [0. 1.]]
 YY: [[0.  0. ]
 [0.5 0.5]
 [1.  1. ]]
```

```
nx, ny = (10, 10)
x = np.linspace(0, 1, nx)
y = np.linspace(0, 1, ny)
xx, yy = np.meshgrid(x, y)
z = np.sin(xx**2 + yy**2)
h = plt.contourf(x,y,z)
plt.show()
```




```
x = np.arange(-5, 5, 0.1)
y = np.arange(-5, 5, 0.1)
xx, yy = np.meshgrid(x, y)
z = np.sin(xx**2 + yy**2) / (xx**2 + yy**2)
h = plt.contourf(x,y,z)
plt.show()
```



배열생성: vstack()


```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
>>> b = np.array([4, 5, 6])
>>> np.vstack((a,b))
array([[1, 2, 3],
       [4, 5, 6]])
>>> A = np.vstack(([1,2,3], [4,5,6], [7, 8, 9], [10, 11, 12]))
>>> A
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

배열생성: vstack()

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

배열생성: hstack()

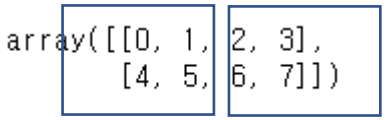
```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
>>> b = np.array([4, 5, 6])
>>> np.hstack((a,b))
array([1, 2, 3, 4, 5, 6])
```

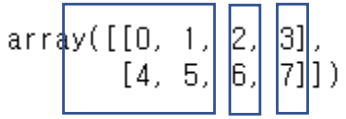
```
>>> a = np.array([[1],[2],[3]])
>>> b = np.array([[4],[5],[6]])
>>> b
array([[4],
       [5],
       [6]])
>>> np.hstack((a,b))
array([[1, 4],
       [2, 5],
       [3, 6]])
```

배열생성: hstack()


```
>>> import numpy as np
>>> A = np.arange(8).reshape(2, 4)
>>> np.hsplit(A, 2)
[array([[0, 1],
       [4, 5]]), array([[2, 3],
       [6, 7]])]
>>> np.hsplit(A, [2, 3])
[array([[0, 1],
       [4, 5]]), array([[2],
       [6]]), array([[3],
       [7]])]
```

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





배열모양변경: reshape()

```
1 A = np.arange(8)
2 A
```

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

```
1 B = A.reshape(2, 4)
```

```
1 A.ndim
```

```
1
```

```
1 A.shape
```

```
(8,)
```

```
1 B.ndim
```

```
2
```

```
1 B
```

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

```
1 C = B.reshape(4,2)
2 C
```

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

```
1 D = B.reshape(2,2, 2)
2 D
```

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

       [[4, 5],
        [6, 7]]])
```

배열모양변경:ravel(), flatten()

```
>>> import numpy as np
>>> A = np.arange(10)
>>> B = A.reshape((2, 5))
>>> B
array([[0, 1, 2, 3, 4],
[5, 6, 7, 8, 9]])
>>> B.ravel()   # a contiguous flattened array.
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.ravel(B) # a contiguous flattened array.
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> B.flatten() # a contiguous flattened array.
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

배열모양변경: 전치행렬

```
>>> import numpy as np
>>> A = np.arange(15).reshape((5,3))
>>> A
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
>>> A.T
array([[ 0,  3,  6,  9, 12],
       [ 1,  4,  7, 10, 13],
       [ 2,  5,  8, 11, 14]])
```

```
>>> A.transpose()
array([[ 0,  3,  6,  9, 12],
       [ 1,  4,  7, 10, 13],
       [ 2,  5,  8, 11, 14]])
```

행렬연산

```
>>> import numpy as np
>>> A = np.arange(10).reshape((2,5))
>>> A
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> A<5
array([[ True,  True,  True,  True,  True],
       [False, False, False, False, False]], dtype=bool)
>>> A>5
array([[False, False, False, False, False],
       [False,  True,  True,  True,  True]], dtype=bool)
```

행렬연산

```
>>> A%2 == 0
array([[ True, False,  True, False,  True],
       [False,  True, False,  True, False]], dtype=bool)
>>> A%2 != 0
array([[False,  True, False,  True, False],
       [ True, False,  True, False,  True]], dtype=bool)
# ndarray.any(axis=None, out=None, keepdims=False)
>>> (A>5).any()
True
# ndarray.all(axis=None, out=None, keepdims=False)
>>> (A>5).all()
False
```

행렬연산

```
>>> np.logical_and(A%2 == 0 , A>5)
array([[False, False, False, False, False],
       [False,  True, False,  True, False]], dtype=bool)
>>> np.logical_or(A%2 == 0 , A>5)
array([[ True, False,  True, False,  True],
       [False,  True,  True,  True,  True]], dtype=bool)
```

Numpy.linalg 선형대수 함수

표준 선행대수 알고리즘에 관련된 함수를 제공하는 라이브러리

Matrix and vector products

dot(a, b[, out])	Dot product of two arrays.
linalg.multi_dot(arrays, *[, out])	Compute the dot product of two or more arrays in a single function call, while automatically selecting the fastest evaluation order.
vdot(a, b)	Return the dot product of two vectors.
inner(a, b)	Inner product of two arrays.
outer(a, b[, out])	Compute the outer product of two vectors.
matmul(x1, x2, /[, out, casting, order, ...])	Matrix product of two arrays.
tensordot(a, b[, axes])	Compute tensor dot product along specified axes.
einsum(subscripts, *operands[, out, dtype, ...])	Evaluates the Einstein summation convention on the operands.
einsum_path(subscripts, *operands[, optimize])	Evaluates the lowest cost contraction order for an einsum expression by considering the creation of intermediate arrays.
linalg.matrix_power(a, n)	Raise a square matrix to the (integer) power *n*.
kron(a, b)	Kronecker product of two arrays.

Numpy.linalg 선형대수 함수

표준 선행대수 알고리즘에 관련된 함수를 제공하는 라이브러리

Decompositions

linalg.cholesky(a)	Cholesky decomposition.
linalg.qr(a[, mode])	Compute the qr factorization of a matrix.
linalg.svd(a[, full_matrices, compute_uv, ...])	Singular Value Decomposition.

Matrix eigenvalues

linalg.eig(a)	Compute the eigenvalues and right eigenvectors of a square array.
linalg.eigh(a[, UPLO])	Return the eigenvalues and eigenvectors of a complex Hermitian (conjugate symmetric) or a real symmetric matrix.
linalg.eigvals(a)	Compute the eigenvalues of a general matrix.
linalg.eigvalsh(a[, UPLO])	Compute the eigenvalues of a complex Hermitian or real symmetric matrix.

Norms and other numbers

linalg.norm(x[, ord, axis, keepdims])	Matrix or vector norm.
linalg.cond(x[, p])	Compute the condition number of a matrix.
linalg.det(a)	Compute the determinant of an array.
linalg.matrix_rank(M[, tol, hermitian])	Return matrix rank of array using SVD method
linalg.slogdet(a)	Compute the sign and (natural) logarithm of the determinant of an array.
trace(a[, offset, axis1, axis2, dtype, out])	Return the sum along diagonals of the array.

Numpy.linalg 선형대수 함수

표준 선행대수 알고리즘에 관련된 함수를 제공하는 라이브러리

Solving equations and inverting matrices

linalg.solve(a, b)	Solve a linear matrix equation, or system of linear scalar equations.
linalg.tensorsolve(a, b[, axes])	Solve the tensor equation `a x = b` for x.
linalg.lstsq(a, b[, rcond])	Return the least-squares solution to a linear matrix equation.
linalg.inv(a)	Compute the (multiplicative) inverse of a matrix.
linalg.pinv(a[, rcond, hermitian])	Compute the (Moore-Penrose) pseudo-inverse of a matrix.
linalg.tensorinv(a[, ind])	Compute the 'inverse' of an N-dimensional array.

Exceptions

linalg.LinAlgError	Generic Python-exception-derived object raised by linalg functions.

선형대수 함수 이용 예제


```
>>> import numpy as np
>>> np.dot(2, 5)
10
>>> np.dot([1, 2], [3, 4])  # np.inner([1, 2], [3, 4])
11
>>> A=np.array([[1,2],[3,4]])
>>> B=np.array([[5,6],[7,8]])
>>> np.dot(A, B)
array([[19, 22],
       [43, 50]])
>>> np.matmul(A, B)
```

$$np.dot(A,\ B) = \begin{bmatrix} 1*5+2*7 & 1*6+2*8 \\ 3*5+4*7 & 3*6+4*8 \end{bmatrix}$$

연립방정식 해

```
>>> import numpy as np
>>> L = np.array([[1, 0, 0], [1, 1, 0], [2, -4.5, 1]])
>>> b = [7, 13, 5]
>>> y = np.linalg.solve(L, b)
>>> y
array([ 7.,  6.,  18.])
```

$$Ly = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & -4.5 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 13 \\ 5 \end{bmatrix} = b$$

문제풀이

- Numpy에서 배열을 생성하는 방법은 무엇인가?
- Numpy에서 제공하는 선형대수 함수를 3개 소개하시오.

요약

- 데이터 분석에서 필수적으로 사용되는 Numpy를 이용한 배열생성 기능을 소개하였음
- Numpy에서 제공하는 행렬 연산 및 선형대수 함수를 소개하였음