실습2. numpy

- Numpy(https://docs.scipy.org/doc/)
- Numpy 배열클래스
- 배열 생성
- 배열 연산, 함수
- Indexing, Slicing , Assigning values
- Shape Manipulation
- View , Shallow Copy / Deep Copy
- Vector Stacking

numpi의 배열 클래스

- numpi의 배열 클래스 : numpy.ndarray
  - ndarray.ndim : 배열의 축수 (차원), the number of axes (dimensions)
  - ndarray.shape : 배열 크기(m,n), 행 크기 m, 열 크기 n
  - ndarray.size : 배열의 요소의 총수 (m*n)
  - ndarray.dtype : 배열 내의 요소의 형태(유형), numpy.int32, numpy.int16 …
  - ndarray.itemsize : 배열의 각 요소의 바이트 단위의 사이즈, float64의 itemsize는 8 (= 64 / 8)
  - ndarray.data : 배열의 실제의 요소를 포함한 버퍼, 색인화 기능을 사용하여 배열의 요소에 액세스하기 때문에 이 속성을 사용할 필요가 없다.

numpi의 배열 클래스 예

```python
import numpy as np

#The Basics
a = np.arange(15) #0-14값을 배열 a에 저장
print("**Arrays**")
print("a=", a)
print("a.ndim=", a.ndim)  #a배열의 차원, the number of axes (dimensions) of the array.
print("a.shape=",a.shape) #a배열의 각 차원의 크기, the dimensions of the array.
print("a.dtype.name=", a.dtype.name) #a 배열의 요소타입, an object describing the type of the elements in the a
print("a.itemsize=", a.itemsize) #the size in bytes of each element of the array
print("a.size=", a.size)  #the total number of elements of the array.
print("type(a)=",type(a))

b = a.reshape(3, 5)       #a배열을 3행 5열 2차원 배열로 변경
print("a.reshape(3, 5) =",b)
print("b.ndim=", b.ndim)
print("b.shape=",b.shape)
```

```
**Arrays**
a= [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
a.ndim= 1
a.shape= (15,)
a.dtype.name= int32
a.itemsize= 4
a.size= 15
type(a)= <class 'numpy.ndarray'>
a.reshape(3, 5) = [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
b.ndim= 2
b.shape= (3, 5)
```

배열 생성

```
#Array creation, 배열 만들기
print("**Array creation**")
f1 = np.arange(0, 2, 0.4) #범위내 실수값 배열 생성
f2 = np.linspace( 0, 2, 6 )  # 범위내 개수로 배열 생성
z = np.zeros((2,2))  # 모든 값이 0인 배열 생성
o = np.ones((1,2))   # 모든 값이 1인 배열 생성
c = np.full((2,2), 7) # 모든 값이 특정 상수인 배열 생성
e = np.eye(2)        # 2x2 단위행렬 생성
r = np.random.random((2,2)) # 임의의 값으로 채워진  2X2 배열 생성
r2 = np.round(5* np.random.random((3,4)) )#  0~4 사이의 임의의 값으로 채워진
a1= np.array([2,3,4])  # 1차원 배열 생성
a2 = np.array([[1,2,3], [5,6,7]])  # 2차원 배열 생성
v1=[33,44,55,66]       # 벡터화 연산, list
v2 = []
for v in v1:           #배열 요소 반복 접근, terating
    v2.append(2 * v)   #요소 추가
```

```
**Array creation**
f1 =  [0.  0.4 0.8 1.2 1.6]
f2 =  [0.  0.4 0.8 1.2 1.6 2. ]
zeros= [[0. 0.]
 [0. 0.]]
ones= [[1. 1.]]
full= [[7 7]
 [7 7]]
eye= [[1. 0.]
 [0. 1.]]
random= [[0.44709161 0.12805096]
 [0.62246773 0.65971785]]
random2= [[1. 1. 2. 5.]
 [4. 4. 1. 1.]
 [2. 5. 4. 3.]]
a1= [2 3 4]
a2= [[1 2 3]
 [5 6 7]]
v1= [33, 44, 55, 66]
v2= [66, 88, 110, 132]
```

배열 연산, 함수

```
#배열 연산
print("**Array operation**")
a = np.arange(5)
b= np.array([5,6,7,8,9])
print("b = ", b)
print("a + 2 = ", a + 2)
print("a ** 2 = ", a ** 2)
print("a < 2 = ", a < 2)
print("a + b = ", a + b)
print("max(a) = ", max(a))
print("sum(a) = ", sum(a))
print("a.sum() = ", a.sum())

#행렬 생성, 연산
m1=np.array([[1,2], [1,0]])
m2=np.array([[1,2], [3,2]])
print("m1 = ", m1)
print("m2 = ", m2)
print("m1.sum(axis=0) = ", m1.sum(axis=0))  #열의 합
print("m1.sum(axis=1) = ", m1.sum(axis=1))  #행의 합
print("m1 + m2=", m1 + m2)
print("m1 * m2=", m1 * m2)   #배열요소곱
print("m1 @ m2=", m1 @ m2)  # 행렬곱
print("m1.dot(m2)=",m1.dot(m2))  # 행렬내적
print("m1 < 2 = ", m1 < 2)
```

```
#Universal Functions
B = np.arange(3)
print("B=",B)
print("np.exp(B)=",np.exp(B))
print("np.sqrt(B)=",np.sqrt(B))
C = np.array([2., -1., 4.])
print("C=",C)
print("np.add(B,C)=",np.add(B, C))
```

```
**Array operation**
b =  [5 6 7 8 9]
a + 2 =  [2 3 4 5 6]
a ** 2 =  [ 0  1  4  9 16]
a < 2 =  [ True  True False False False]
a + b =  [ 5  7  9 11 13]
max(a) =  4
sum(a) =  10
a.sum() =  10
m1 =  [[1 2]
 [1 0]]
m2 =  [[1 2]
 [3 2]]
m1.sum(axis=0) =  [2 2]
m1.sum(axis=1) =  [3 1]
m1 + m2= [[2 4]
 [4 2]]
m1 * m2= [[1 4]
 [3 0]]
m1 @ m2= [[7 6]
 [1 2]]
m1.dot(m2)= [[7 6]
 [1 2]]
m1 < 2 =  [[ True False]
 [ True  True]]
B= [0 1 2]
np.exp(B)= [1.         2.71828183 7.3890561 ]
np.sqrt(B)= [0.         1.         1.41421356]
C= [ 2. -1.  4.]
np.add(B,C)= [2. 0. 6.]
```

Indexing, Slicing , Assigning values

```
#Indexing, Slicing
print("**Array Indexing, Slicing**")
a= np.arange(5)
print("a = ", a)
print("a[0] = ", a[0])
a1 = a[2:4]                    #2~(4-1)
print("a[2:4] = ", a1)
a2 = a[:2]                     #~(2-1)
print("a[:2] = ", a2)
a3 = a[-1]                     #reverse  index,    ... -2 -1
print("a[-1] = ", a3)
a4 = a[1:4:2]                  #1에서 (4-1)까지 step 2
print("a[1:4:2] = ", a4)

# numpy.ndarray :  index arrays, Fancy indexing
print("** index arrays**")
i = [1,3,4]
a[i]
print("type(a)=",type(a))  #numpy.ndarray
print("i = ", i)
print("a[i] = ", a[i])

# Boolean or  "mask"  index arrays¶
print("** mask index arrays**")
f = [True, False, True, True,False]
print("f = ", f)
print("a[f] = ", a[f])
```

```
#Assigning values
print("** Assigning values **")
a[0]=100
a[1:3] = -2              #1~(3-1)인덱스에 해당하는 요소에 -2 대입
print("a = ", a)
i = a[1:3]               #1~(3-1)인덱스에 해당하는 인덱스를 i에 대입
i[1:3] = 99              # i 값 수정
print("a = ", a)
print("i = ", i)
```

```
**Array Indexing, Slicing**
a =  [0 1 2 3 4]
a[0] =  0
a[2:4] =  [2 3]
a[:2] =  [0 1]
a[-1] =  4
a[1:4:2] =  [1 3]
** index arrays**
type(a)= <class 'numpy.ndarray'>
i =  [1, 3, 4]
a[i] =  [1 3 4]
** mask index arrays**
f =  [True, False, True, True, False]
a[f] =  [0 2 3]
** Assigning values **
a =  [100  -2  -2   3   4]
a =  [100  -2  99   3   4]
i =  [-2 99]
```

Indexing, Slicing(2차원 배열)

```
#2차원 배열, Indexing, Slicing , index arrays
print("** 2차원 Indexing, Slicing , index arrays **")
a2 =  np.array([[1,2,3,4], [10,20,30,40], [6,7,8,9]])
print("a2 = ", a2)
print("a2[1,2] = ", a2[1,2])
print("a2[:, 1] = ", a2[:,1])
print("a2[1:3, 2:3] = ", a2[1:3,2:3])
print("a2[1:2, :] = ", a2[1:2,:])
print("a2[-1] = ", a2[-1])              #마지막 행
print("a2[1, :] = ", a2[1,:])
print("a2[1, ...] = ", a2[1,...])      #1행의 모든 열

m = a2 < 10                     #조건에 맞는 mask 생성
print("m= ", m)
print("a2[m] = ", a2[m])        #조건에 맞는 배열요소만 추출

#iterator
print("** 2차원 iterator **")
for r in a2:            #행단위 접근
    print(r)

for e in a2.flat:       #1차원으로 변경하여 접근
    print(e)
```

```
** 2차원 Indexing, Slicing , index arrays **
a2 =  [[ 1  2  3  4]
 [10 20 30 40]
 [ 6  7  8  9]]
a2[1,2] =  30
a2[:, 1] =  [ 2 20  7]
a2[1:3, 2:3] =  [[30]
 [ 8]]
a2[1:2, :] =  [[10 20 30 40]]
a2[-1] =  [6 7 8 9]
a2[1, :] =  [10 20 30 40]
a2[1, ...] =  [10 20 30 40]
m=  [[ True  True  True  True]
 [False False False False]
 [ True  True  True  True]]
a2[m] =  [1 2 3 4 6 7 8 9]
```

```
** 2차원 iterator **
[1 2 3 4]
[10 20 30 40]
[6 7 8 9]
1
2
3
4
10
20
30
40
6
7
8
9
```

Shape Manipulation

```
#Shape Manipulation

a = np.floor(10*np.random.random((3,4))) # 3X4  배열에 random number (0~9)
print("a=", a)
print("a.shape", a.shape)
b= a.ravel()                  # 2차배열이 펼쳐진 1차배열로 반환
print("a.ravel()=", b)
c=a.reshape(6,2)              # a의 형태가 6행 2열로 수정되어 반환
print("a.reshape(6,2) = ", c)
at = a.T                      # a의 전치행렬 반환 (행, 열이 교환된 행렬 )
print("a.T) = ", at)
print("a.a.T.shape = ", a.T.shape)
d= a.resize((2,6))           #a의 shape을 수정, 반환없음
print("a.reshape(2,6) = ", a)
print("a.reshape(2,6) = ", d)
e = a.reshape(3,-1)          #행크기 3에 맞추어 열크기 자동으로 변경
print("a.reshape(3,-1) = ", e)
```

```
a= [[0. 6. 8. 2.]
 [8. 7. 0. 4.]
 [2. 7. 4. 4.]]
a.shape (3, 4)
a.ravel()= [0. 6. 8. 2. 8. 7. 0. 4. 2. 7. 4. 4.]
a.reshape(6,2) =  [[0. 6.]
 [8. 2.]
 [8. 7.]
 [0. 4.]
 [2. 7.]
 [4. 4.]]
a.T) =  [[0. 8. 2.]
 [6. 7. 7.]
 [8. 0. 4.]
 [2. 4. 4.]]
a.a.T.shape =  (4, 3)
a.reshape(2,6) =  [[0. 6. 8. 2. 8. 7.]
 [0. 4. 2. 7. 4. 4.]]
a.reshape(2,6) =  None
a.reshape(3,-1) =  [[0. 6. 8. 2.]
 [8. 7. 0. 4.]
 [2. 7. 4. 4.]]
```

View , Shallow Copy / Deep Copy

```
#View , Shallow Copy
a = np.arange(12)
b = a            # b는 새로운 객체가 아닌 a를 참조
print("a=", a)
print("b=", b)
print("b is a ", b is a)    # a and b are two names for the same ndarray object
b.shape = 3,4    # changes the shape of a
print("b.shape=", b.shape)
print("a.shape=", a.shape)
print("a=", a)
print("b=", b)

c = a.view()     #view : a와는 다른 객체이지만 데이터는 공유
print("c=", c)
print("c is a ", c is a)
print("c.base is a ", c.base is a)
c.shape = 2,6                    # a's shape doesn't change
c[0,4] = 1234                    # a's data changes
print("c.shape=", c.shape)
print("a.shape=", a.shape)
print("c=", c)
print("a=", a)

#Deep Copy
d = a.copy()     #copy : a와는 다른 객체로 데이터 복제
print("d=", d)
print("d is a ", d is a)
print("d.base is a ", d.base is a)
d[0,0] = 99
print("d=", d)
print("a=", a)
```

```
a= [ 0  1  2  3  4  5  6  7  8  9 10 11]
b= [ 0  1  2  3  4  5  6  7  8  9 10 11]
b is a  True
b.shape= (3, 4)
a.shape= (3, 4)
a= [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
b= [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
c= [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
c is a  False
c.base is a  True
c.shape= (2, 6)
a.shape= (3, 4)
c= [[   0    1    2    3 1234    5]
 [   6    7    8    9   10   11]]
a= [[   0    1    2    3]
 [1234    5    6    7]
 [   8    9   10   11]]
d= [[   0    1    2    3]
 [1234    5    6    7]
 [   8    9   10   11]]
d is a  False
d.base is a  False
d= [[  99    1    2    3]
 [1234    5    6    7]
 [   8    9   10   11]]
a= [[   0    1    2    3]
 [1234    5    6    7]
 [   8    9   10   11]]
```

Vector Stacking, 벡터 결합

```
#Vector Stacking, 벡터 결합
#vstack : 행으로 결합
#hstack : 열로 결합


x = np.arange(0,10,2)                    # x=([0,2,4,6,8])
y = np.arange(5)                         # y=([0,1,2,3,4])
m = np.vstack([x,y])                     # m=([[0,2,4,6,8],
                                         #     [0,1,2,3,4]])
xy = np.hstack([x,y])                    # xy =([0,2,4,6,8,0,1,2,3,4])
```

```
x= [0 2 4 6 8]
y= [0 1 2 3 4]
m= [[0 2 4 6 8]
 [0 1 2 3 4]]
xy= [0 2 4 6 8 0 1 2 3 4]
```

연습문제

#1. 배열 생성, indexing, slicing, Assigning values, reshape, resize 연습
(1)1~25 사이의 2의 배수 12개로 1차원 배열(a) 생성, 출력
(2)a 의 인덱스 2~5의 요소값을 -20으로 수정, 출력
(3)a를 이용하여 3X4 배열(b)로 변경, 출력
(4)a를 2X6 배열로 변경, 출력
(5)a를 1행의 모든 값을 추출하여 a1을 만들고, a1의 모든 값을 0으로 변경, 출력

2. 배열 연산, 함수 연습
(1)1~10 사이의 임의의 값으로 3X3 배열 x, y생성, 출력
(2)x의 1, 2 행의 모든 열추출, 출력
(3)x의 2열의 모든 행 추출, 출력
(4)x의 0,2열의 1,2 행 추출, 출력
(5)x의 각행의 합, 각 열의 합, 출력
(6)x의 각행의 최대값, 각 열의 최대값, 출력
(7)x, y 배열의 합, 곱, 내적을 구하여 출력

3. 배열복사, 결합 연습
(1) 1~3, 10~30, 100~300 3개의 1차원 배열(a1,a2,a3)을 생성하여 행으로 결합하여 y배열 생성
(2) y배열의 0,2 열을 복사하여 y1, 1열을 복사하여 y2 생성
(3) y1, y2를 결합하여 yy배열 생성