데이터처리프로그래밍

numpy

강원대학교 교육혁신원 송혜정

<hjsong@kangwon.ac.kr>

Numpy

✓ 학습목표

- 파이선 라이브러리 numpy 를 이해한다.

✓ 학습내용

- 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

강의에 앞서서..

- 본 강의자료는 아래의 자료들을 참고하여 만들어 졌음을 알립니다

1. Numpy (https://docs.scipy.org/doc/numpy-1.15.0/index.html)

numpy

- Numpy(Numerical Python)
  - 과학계산에 필요한 파이썬 라이브러리로 빠르고 효율적인 조작이 가능한 다차원 배열객체를 제공한다.
  - NumPy의 배열 ndarray는 파이썬의 배열 list보다 더 큰 배열을 관리하는 데 효율적인 저장과 데이터 작업을 제공한다.
  - NumPy 배열(ndarray) 특징
    - 고정크기 배열
    - NumPy 배열의 요소는 모두 동일한 데이터 유형
    - NumPy 배열은 큰 데이터를 저장하여 고급 처리 작업에 용이
    - 많은 파이썬 기반의 과학 및 수학 패키지는 NumPy 배열을 사용

numpy의 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, float64 …
  - 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.

numpy의 배열 클래스 예

```
# The Basics
a= np.array([2,3,4])  # 1차원 배열 생성
print("**Arrays**")
print("a=", a)
print("a.ndim=", a.ndim)          #a배열의 차원,
print("a.shape=",a.shape)         #a배열의 각 차원의 크기,
print("a.itemsize=", a.itemsize) #배열요소의  바이트수
print("a.size=", a.size)          #배열 요소의 개수
print("type(a)=",type(a))

b = np.array([[1,2,3], [5,6,7]])  # 2차원 배열 생성
print("b.ndim=", b.ndim)
print("b.shape=",b.shape)
```

```
**Arrays**
a= [2 3 4]
a.ndim= 1
a.shape= (3,)
a.itemsize= 4
a.size= 3
type(a)= <class 'numpy.ndarray'>
b.ndim= 2
b.shape= (2, 3)
```

배열 생성

```
#Array creation, 배열 만들기
print("**Array creation**")
a1= np.array([20,30,40])  # 1차원 배열 생성
a2 = np.array([[10,20,30], [50,60,70]])  # 2차원 배열 생성
f1 = np.arange(0, 2, 0.4) #0~2 범위내 0.4 간격으로 실수값 배열 생성
f2 = np.linspace( 0, 2, 6 )  # 0~2 범위내 6개 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
r1 = np.random.randint(2, size=10)  #0~1, 10개 정수난수
r2 = np.random.randint(10, size=(2, 3)) #0~9, 2X3 array
r3 = np.random.randint(1, 7, size=(5, 3)) #1~6, 5X3 array

print("a1=",a1)
print("a2=",a2)
print("f1 = ", f1)
print("f2 = ", f2)
print("zeros=",z)
print("ones=",o)
print("full=",c)
print("eye=",e)
print("random1=",r1)
print("random2=",r2)
print("random3=",r3)
```

```
**Array creation**
a1= [20 30 40]
a2= [[10 20 30]
 [50 60 70]]
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.]]
random1= [1 0 0 0 0 1 1 0 0 0]
random2= [[6 3 4]
 [4 5 3]]
random3= [[2 6 1]
 [2 1 2]
 [4 3 3]
 [5 2 4]
 [4 3 2]]
```

Random number

- numpy.random 서브패키지를 통해 난수를 발생
  - rand(d0,d1, ..,dn): 0 ~ 1사이의 균일 분포(uniform distribution) 난수생성
  - randn(d0,d1, ..,dn): 평균0, 표준편차1인 표준 정규 분포(standard normal distribution)
  - normal(mean, std, d) : 평균 mean, 표준편차 strd인 정규분포 (normal distribution)
  - randint(low, high=None, size=None): low (inclusive) to high (exclusive). 정수 난수 size 개 생성

```
np.random.rand(3)      #0~1사이 난수를 3개를 1차원 배열에 생성하여 반환
```

```
array([0.21276842, 0.30382988, 0.525148  ])
```

```
np.random.rand(3,2)      #0~1사이 난수를 3X2 배열에 생성하여 반환
```

```
array([[0.64596241, 0.92468632],
       [0.09351173, 0.86811593],
       [0.55882584, 0.03807589]])
```

Random number

```
np.random.randn()       #평균0, 표준편차1인 표준정규분포 난수 1개 발생
```

```
-0.21486020524792135
```

```
np.random.randn(5)       #평균0, 표준편차1인 표준정규분포 난수 5개 발생
```

```
array([ 0.85462729, -0.87393037, -0.46179613, -1.7063945 ,  0.68692294])
```

```
np.random.randn(2, 4)   #평균0, 표준편차1인 표준정규분포 난수 2x4 개 발생
```

```
array([[-1.25068198, -0.2655829 ,  1.16732182,  1.65217772],
       [ 0.29722394, -1.40135486, -0.73887499,  0.23767817]])
```

```
mean = 10
std = 5
r = np.random.normal(mean, std, 10) #평균 mean, 표준편차 std인 정규분포
print(r)
```

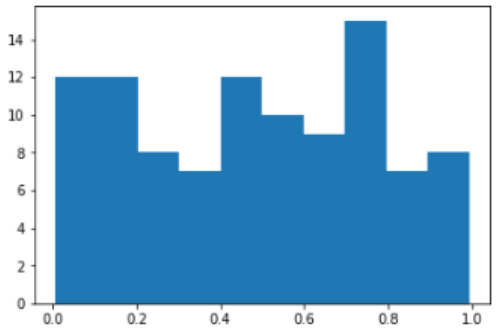
```
[16.59723704 16.21240692  8.41744936 15.82288854  7.02571991  5.65832958
  8.38909924 20.27919934  5.3014978  15.21215778]
```

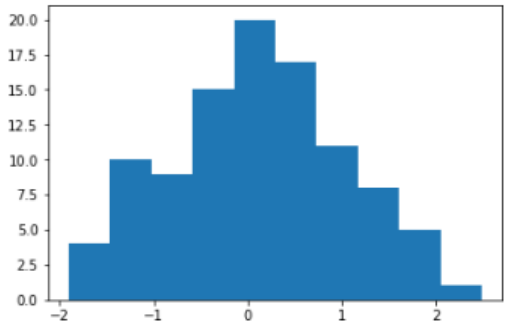
```
np.random.randint(3, size=10)  #0~2, 10개 정수난수
```

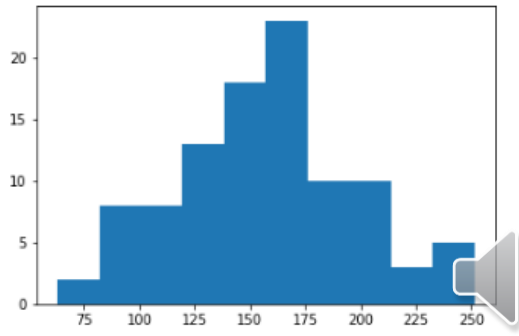
```
array([2, 2, 1, 0, 2, 0, 2, 0, 2, 2])
```

Random number

```
#그래프로 랜덤값 분포 확인
import matplotlib.pyplot as plt

r1 = np.random.rand(100)      #0~1사이 균일분포
plt.hist(r1)  # 히스토그램, 데이터 분포 확인
plt.show()

r2 = np.random.randn(100)  #평균0, 표준편차1인 표준정규분포 (가우시안 분포)
plt.hist(r2)
plt.show()

#평균, 표준편차를 알고 있는 정규 분포에서 표본 추출
mean = 160
std = 40
r3 = np.random.normal(mean, std, 100) #평균 mean, 표준편차 std인 정규분포
plt.hist(r3)
plt.show()
```







배열 연산, 함수

```
#1차원 배열 연산
print("**Array operation**")
a = np.arange(5)
b= np.array([5,6,7,8,9])
print("a = ", a)
print("b = ", b)
print("a + 2 = ", a + 2)        #배열 요소에 2를 더한 결과
print("a ** 2 = ", a ** 2)      #배열요소를 제곱한 결과
print("a < 2 = ", a < 2)        #배열요소가 2보다 작으면 True, 아니면 False
print("a + b = ", a + b)        #a배열 요소와 b 배열요소의 합
print("max(a) = ", max(a))      #a배열 요소의 최대값 구하는 함수호출
print("sum(a) = ", sum(a))      #a배열요소의 합의 구하는 함수 호출
print("a.sum() = ", a.sum())    #a배열요소의 합을 구하는 함수호출
print("b.mean() = ", b.mean())  #b배열요소의 평균을 구하는 함수호출
print("b.std() = ", b.std())    #b배열요소의 표준편차을 구하는 함수호출
```

```
**Array operation**
a =  [0 1 2 3 4]
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
b.mean() =  7.0
b.std() =  1.4142135623730951
```

Indexing, Slicing , Assigning values

- 배열의 인덱싱(Indexing)
  - 배열 요소에 값을 설정하거나 얻기 위한 위치를 접근하는 인덱싱
- 배열의 슬라이싱(Slicing)
  - 큰 배열 안의 서브배열에 접근하여 값을 설정하거나 얻기 위한 인덱싱

```
#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]
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])   #참인 인덱스값만 추출

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

```
**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]
```

2차원 배열 생성 및 연산

```
#2차원 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)
```

```
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]]
```

2차원 배열, Indexing, Slicing , index arrays

```
#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])          #1행, 2열
print("a2[:, 1] = ", a2[:,1])         #1열의 모든행
print("a2[1:3, 2:4] = ", a2[1:3,2:4]) #1~(3-1)행, 2~(4-1)열
print("a2[1:2, :] = ", a2[1:2,:])
print("a2[-1] = ", a2[-1])            #마지막 행
print("a2[1, :] = ", a2[1,:])         #1행의 모든 열
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)

#2차원을 1차원으로 변경
print("** 1차원 변경 iterator **")
#a1 =a2.ravel()
a1 =a2.flat
for e in a1:       #1차원으로 접근
    print(e, end=" ")
```

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

Shape Manipulation

```
#차원 변경, Shape Manipulation

a = np.random.randint(10, size=(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)
b2= a.reshape(-1)               # 2차배열이 펼쳐진 1차배열로 반환
print("a.reshape(-1)=", b2)
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= [[2 7 0 5]
 [5 0 9 5]
 [3 2 6 4]]
a.shape (3, 4)
a.ravel()= [2 7 0 5 5 0 9 5 3 2 6 4]
a.reshape(-1)= [2 7 0 5 5 0 9 5 3 2 6 4]
a.reshape(6,2) =  [[2 7]
 [0 5]
 [5 0]
 [9 5]
 [3 2]
 [6 4]]
a.T =  [[2 5 3]
 [7 0 2]
 [0 9 6]
 [5 5 4]]
a.a.T.shape =  (4, 3)
a.reshape(2,6) =  [[2 7 0 5 5 0]
 [9 5 3 2 6 4]]
a.reshape(2,6) =  None
a.reshape(3,-1) =  [[2 7 0 5]
 [5 0 9 5]
 [3 2 6 4]]
```

View , Shallow Copy / Deep Copy

```python
#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]
```

QUIZ!

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 요소 값을 -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 열 추출, 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

QUIZ!

- Numpy 에 대한 학습활동 결과는 e-루리에 제출바랍니다.
- 제목 : Report11. Numpy 연습문제
- 제출내용 : Numpy 연습문제를 해결하여

  “Report11_성명_numpy.ipynb” 파일을 제출
- 제출기한 : 2021년 6월 8일 오후 11:30