Python 과학 프로그래밍 기초
7. NumPy (1)

한림대학교 소프트웨어융합대학
박섭형

2021년 1학기

NumPy 개요

- 수학, 과학, 공학, 데이터 과학 분야 응용 분야에서 Python으로 사용할 수 있는 open source 라이브러리
- NumPy Homepage
- C/C++와 Fortran과 통합하기 쉬움
- 데이터 구조
  - ndarray 클래스: array 클래스라고 부르기도 함
  - 동일한 데이터형 원소들(주로 숫자형 데이터)의 다차원 배열
  - indexing: 정수들의 tuple로 인덱스를 표현
  - broadcasting
- 다양한 수치 연산 도구
  - 다차원 배열에 적용할 수 있는 벡터화 함수
  - 선형 대수, 푸리에 변환, 콘볼루션, 통계 라이브러리 등

NumPy의 장점

- 다차원 배열 또는 행렬 연산의 편리함
- 우수한 메모리 효율
- 빠른 처리 속도

[1]:
```
import numpy as np
np.__version__
```

[1]:
```
'1.19.2'
```

[2]:
```
lst = [1, 3, 4]
lst2 = [[1, 3, 4], [3, 2, 1]]
a = np.array(lst)
a2 = np.array(lst2)
```

[3]:
```
a
```

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

[4]:
```
a2
```

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

[5]:
```
a[0]
```

[5]:
```
1
```

[6]:
```
a[-1]
```

[6]:
```
4
```

[7]:
```
a[1:3]
```

[7]:
```
array([3, 4])
```

[8]:
```
a2[1][2]
```

[8]:
```
1
```

[9]:
```
a2[1,2]
```

[9]:
```
1
```

NumPy ndarray와 리스트의 행렬곱셈 연산비교

[10]:
```
a_row = 10
a_col = 10
b_row = 10
b_col = 10

# numpy ndarray
na = np.random.rand(a_row, a_col)
```

```
nb = np.random.rand(a_row, a_col)

# list
la = na.tolist()
lb = nb.tolist()

# numpy matrix
ma = np.matrix(na)
mb = np.matrix(nb)
```

[12]:
```
%timeit na.dot(nb)
%timeit na @ nb
%timeit [[sum([r*c for r,c in zip(row, col)]) for col in zip(*lb)] for row in␣
↪la]
%timeit ma.dot(mb)
%timeit ma @ mb
```

```
1.02 µs ± 58.8 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
1.7 µs ± 30.7 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
144 µs ± 3.16 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
1.47 µs ± 42.2 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
3.71 µs ± 260 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

소요 시간 비교 na.dot(nb) < ma.dot(mb) < na @ nb < ma @ mb < list 곱셈

NumPy의 다차원 배열과 리스트 객체의 구조 비교

- 리스트
  - 이질(heterogeneous) 데이터형 원소 가능
  - 동적 배열
- NumPy 다차원 배열
  - 동질(homogeneous) 데이터형 원소만 가능
  - 고정 크기 배열

[13]:
```
from sys import getsizeof
```

[14]:
```
lst = []
for n in range(10):
    if n > 0:
        lst.append(n)
    print(f"Addr. of lst: {id(lst)}", end=", ")
    print(f"No. of item(s): {len(lst)}", end=", ")
    print(f"Size of lst: {getsizeof(lst)} bytes", end=", ")
    print(f"Size of items: {sum([getsizeof(item) for item in lst])} bytes")
```

```
Addr. of lst: 1978016582144, No. of item(s): 0, Size of lst: 56 bytes, Size of
items: 0 bytes
Addr. of lst: 1978016582144, No. of item(s): 1, Size of lst: 88 bytes, Size of
items: 28 bytes
Addr. of lst: 1978016582144, No. of item(s): 2, Size of lst: 88 bytes, Size of
items: 56 bytes
Addr. of lst: 1978016582144, No. of item(s): 3, Size of lst: 88 bytes, Size of
items: 84 bytes
Addr. of lst: 1978016582144, No. of item(s): 4, Size of lst: 88 bytes, Size of
items: 112 bytes
Addr. of lst: 1978016582144, No. of item(s): 5, Size of lst: 120 bytes, Size of
items: 140 bytes
Addr. of lst: 1978016582144, No. of item(s): 6, Size of lst: 120 bytes, Size of
items: 168 bytes
Addr. of lst: 1978016582144, No. of item(s): 7, Size of lst: 120 bytes, Size of
items: 196 bytes
Addr. of lst: 1978016582144, No. of item(s): 8, Size of lst: 120 bytes, Size of
items: 224 bytes
Addr. of lst: 1978016582144, No. of item(s): 9, Size of lst: 184 bytes, Size of
items: 252 bytes
```

[15]:
```
import numpy as np
x = np.array([])
print(f"Size of numpy array x: {getsizeof(x)} bytes")
```

```
Size of numpy array x: 96 bytes
```

[16]:
```
x = np.array([], np.int32)
print(x.dtype)
for n in range(10):
    if n > 0:
        x = np.append(x, [n])
    print(f"Addr. of x: {id(x)}", end=", ")
    print(f"No. of item(s): {len(x)}", end=", ")
    print(f"Size of x: {getsizeof(x)} bytes")
```

```
int32
Addr. of x: 1978016544688, No. of item(s): 0, Size of x: 96 bytes
Addr. of x: 1978016662672, No. of item(s): 1, Size of x: 100 bytes
Addr. of x: 1978016662912, No. of item(s): 2, Size of x: 104 bytes
Addr. of x: 1978016663232, No. of item(s): 3, Size of x: 108 bytes
Addr. of x: 1978016663472, No. of item(s): 4, Size of x: 112 bytes
Addr. of x: 1978016663712, No. of item(s): 5, Size of x: 116 bytes
Addr. of x: 1978016663952, No. of item(s): 6, Size of x: 120 bytes
Addr. of x: 1978016664192, No. of item(s): 7, Size of x: 124 bytes
Addr. of x: 1978016664432, No. of item(s): 8, Size of x: 128 bytes
Addr. of x: 1978016664672, No. of item(s): 9, Size of x: 132 bytes
```

[17]:
```
x = np.array([], np.int64)
print(x.dtype)
for n in range(10):
    if n > 0:
        x = np.append(x, [n])
    print(f"Addr. of x: {id(x)}", end=", ")
    print(f"No. of item(s): {len(x)}", end=", ")
    print(f"Size of x: {getsizeof(x)} bytes")
```

```
int64
Addr. of x: 1978016628048, No. of item(s): 0, Size of x: 96 bytes
Addr. of x: 1978016664592, No. of item(s): 1, Size of x: 104 bytes
Addr. of x: 1978016661552, No. of item(s): 2, Size of x: 112 bytes
Addr. of x: 1978016661872, No. of item(s): 3, Size of x: 120 bytes
Addr. of x: 1978016662112, No. of item(s): 4, Size of x: 128 bytes
```

```
Addr. of x: 1978016661712, No. of item(s): 5, Size of x: 136 bytes
Addr. of x: 1978016665152, No. of item(s): 6, Size of x: 144 bytes
Addr. of x: 1978016665392, No. of item(s): 7, Size of x: 152 bytes
Addr. of x: 1978016627248, No. of item(s): 8, Size of x: 160 bytes
Addr. of x: 1978016440848, No. of item(s): 9, Size of x: 168 bytes
```

[18]:
```
x.dtype
```

[18]:
```
dtype('int64')
```

NumPy의 다차원 배열을 이용한 1차원 벡터의 표현과 연산

[24]:
```
import numpy as np
a = np.array([1, 3, 5, 3, 1])
b = np.array([3, 5, 7, 1, 2])
a + b
```

[24]:
```
array([ 4,  8, 12,  4,  3])
```

[25]:
```
2 * a
```

[25]:
```
array([ 2,  6, 10,  6,  2])
```

NumPy 다차원 배열 클래스 ndarray의 주요 attributes

- ndarray.ndim: axes (dimensions)의 수
- ndarray.shape: axes별 원소의 개수를 튜플로 표현
- ndarray.size: 원소의 개수
- ndarray.dtype: numpy.int32, numpy.int16, numpy.float64 등
- ndarray.itemsize: 각 원소의 크기를 byte 단위
- ndarray.nbytes: 전체 데이터의 크기를 byte 단위로 표시

[26]:
```
na = np.random.randint(1, 10, (3, 4))
na
```

[26]:
```
array([[5, 1, 7, 2],
       [5, 1, 6, 6],
```

```
       [7, 2, 5, 6]])
```

```
[27]: na.ndim
```

```
[27]: 2
```

```
[28]: na.shape
```

```
[28]: (3, 4)
```

```
[29]: na.dtype
```

```
[29]: dtype('int32')
```

```
[30]: na.size
```

```
[30]: 12
```

```
[31]: na.itemsize
```

```
[31]: 4
```

```
[32]: na.nbytes
```

```
[32]: 48
```

NumPy ndarray 객체의 데이터 형

dtype	문자 코드	설명	호환되는 C의 data type
ndarray.int8	'b' or 'i1'	8-bit signed integer	char
ndarray.int16	'h' or 'i2'	16-bit signed integer	short
ndarray.int32	'i' or 'i4'	32-bit signed integer	int
ndarray.int64	'q' or 'i8'	64-bit signed integer	long
ndarray.uint8	'B' or 'u1'	8-bit unsigned integer	unsigned char
ndarray.uint16	'H' or 'u2'	16-bit unsigned integer	unsigned short
ndarray.uint32	'I' or 'u4'	32-bit unsigned integer	unsigned int
ndarray.uint64	'Q' or 'u8	64-bit unsigned integer	unsigned long

정수 데이터 형

[33]:
```
int_dt = [np.int8, np.uint8,
      np.int16, np.uint16,
      np.int32, np.uint32,
      np.int64, np.uint64, 'i1', 'i2', 'i4', 'i8', 'u1', 'u2', 'u4', 'u8']

for dt in int_dt:
    x = np.array([1, 2, 3], dtype=dt)
    print(x.dtype, x.itemsize)
```

```
int8 1
uint8 1
int16 2
uint16 2
int32 4
uint32 4
int64 8
uint64 8
int8 1
int16 2
int32 4
int64 8
uint8 1
uint16 2
uint32 4
uint64 8
```

[34]:
```
int_dt = ['<b', '<B',
          '<h', '<H',
          '<i', '<I',
          '<q', '<Q']

for dt in int_dt:
    x = np.array([1, 2, 3], dtype=dt)
    print(x.dtype, x.itemsize)
```

```
int8 1
uint8 1
int16 2
uint16 2
int32 4
uint32 4
int64 8
uint64 8
```

dtype	문자 코드	설명 (sign, exponent, mantissa)	호환되는 C의 data type
ndarray.float16	'e' or 'f2'	16-bit floating point (1, 5, 10)	
ndarray.float32	'f' or 'f4'	32-bit floating point (1, 8, 23)	float
ndarray.float64	'd' or 'f8'	64-bit floating point (1, 11, 52)	double

실수 데이터 형

[35]:
```
float_dt = [np.float16, np.float32, np.float64, 'f2', 'f4', 'f8']

for dt in float_dt:
    x = np.array([1, 2, 3], dtype=dt)
    print(x.dtype, x.itemsize)
```

```
float16 2
float32 4
float64 8
float16 2
float32 4
float64 8
```

[36]:
```
float_dt = ['<e', '<f', '<d']

for dt in float_dt:
    x = np.array([1, 2, 3], dtype=dt)
    print(x.dtype, x.itemsize)
```

```
float16 2
float32 4
float64 8
```

dtype	문자 코드	설명
ndarray.complex64	'F' or 'c8'	32-bit floating point x 2
ndarray.complex128	'G' or 'c16'	64-bit floating point x 2

복소수 데이터 형

[37]:
```
complex_dt = [np.csingle, np.cdouble]

for dt in complex_dt:
    x = np.array([1, 2, 3], dtype=dt)
    print(x.dtype, x.itemsize)
```

```
complex64 8
complex128 16
```

[38]:
```
complex_dt = ['<F', '<G']

for dt in complex_dt:
    x = np.array([1, 2, 3], dtype=dt)
    print(x.dtype, x.itemsize)
```

```
complex64 8
complex128 16
```

NumPy 정수형의 overflow와 underflow

[39]:
```
import numpy as np
a = np.array([100, 200], np.int8)
a
```

[39]:
```
array([100, -56], dtype=int8)
```

[40]:
```
np.iinfo(np.int8)
```

[40]:
```
iinfo(min=-128, max=127, dtype=int8)
```

[41]:
```
a + a
```

[41]:
```
array([ -56, -112], dtype=int8)
```

[42]:
```
3 * a
```

[42]:
```
array([44, 88], dtype=int8)
```

[43]:
```
b = np.array([100, 200], np.uint8)
b
```

[43]:
```
array([100, 200], dtype=uint8)
```

[44]:
```
np.iinfo(np.uint8)
```

[44]:
```
iinfo(min=0, max=255, dtype=uint8)
```

[45]:
```
2 * b
```

[45]:
```
array([200, 144], dtype=uint8)
```

0.1 Ndarray 생성

리스트와 튜플을 이용한 ndarray 생성

[49]:
```
data1 = (1, 2, 3, 4, 5) # tuple의 모든 원소의 데이터 유형이 같아야 한다
arr1 = np.array(data1) # 1d array
```

[50]:
```
arr1
```

[50]:
```
array([1, 2, 3, 4, 5])
```

[51]:
```
arr1.dtype
```

[51]:
```
dtype('int32')
```

[52]:
```
arr1.ndim
```

[52]:
```
1
```

[53]:
```
arr1.size
```

[53]:
```
5
```

[54]:
```
data2 = [[1, 2, 3., 4], [5, 6, 7, 8.]] # nested list
arr2 = np.array(data2) # 2d array
```

[55]:
```
data2
```

[55]:
```
[[1, 2, 3.0, 4], [5, 6, 7, 8.0]]
```

[56]:
```
arr2
```

[56]:
```
array([[1., 2., 3., 4.],
       [5., 6., 7., 8.]])
```

[57]:
```
arr2.dtype
```

[57]:
```
dtype('float64')
```

[58]:
```
arr2.ndim
```

[58]:
```
2
```

[59]:
```
arr2.size
```

[59]:
```
8
```

[60]:
```
data3 = [[1, 2, 3., 4], [5+1j, 6, 7, 8.]] # nested list
arr3 = np.array(data3) # 2d array
```

[61]:
```
data3
```

[61]:
```
[[1, 2, 3.0, 4], [(5+1j), 6, 7, 8.0]]
```

[62]:
```
arr3
```

[62]:
```
array([[1.+0.j, 2.+0.j, 3.+0.j, 4.+0.j],
       [5.+1.j, 6.+0.j, 7.+0.j, 8.+0.j]])
```

[63]:
```
arr3.dtype
```

[63]:
```
dtype('complex128')
```

[64]:
```
arr3.ndim
```

[64]:
```
2
```

[65]:
```
arr3.size
```

[65]:
```
8
```

[66]:
```
data4 = ['a', 'b']
arr4 = np.array(data4)
```

[67]:
```
arr4.dtype
```

[67]:
```
dtype('<U1')
```

[68]:
```
arr4.itemsize
```

[68]:
```
4
```

[69]:
```
data5 = ['ab', 'b']
arr5 = np.array(data5)
```

[70]:
```
arr5.dtype
```

[70]:
```
dtype('<U2')
```

[71]:
```
arr5.itemsize
```

[71]:
```
8
```

[72]:
```
data6 = ['abc', 'b']
arr6 = np.array(data6)
```

[73]:
```
arr6.dtype
```

[73]:
```
dtype('<U3')
```

[74]:
```
arr6.itemsize
```

[74]:
```
12
```

[75]:
```
getsizeof(np.array([])), getsizeof(arr6)
```

[75]:
```
(96, 120)
```

[76]:
```
for n in range(0, 53):
    data_s = 'abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz'[:n]
    arr_s = np.array(data_s)
    print(arr_s, arr_s.dtype, arr_s.itemsize, getsizeof(arr_s))
```

```
 <U1 4 84
a <U1 4 84
ab <U2 8 88
abc <U3 12 92
abcd <U4 16 96
abcde <U5 20 100
abcdef <U6 24 104
abcdefg <U7 28 108
abcdefgh <U8 32 112
abcdefghi <U9 36 116
abcdefghij <U10 40 120
abcdefghijk <U11 44 124
abcdefghijkl <U12 48 128
abcdefghijklm <U13 52 132
abcdefghijklmn <U14 56 136
abcdefghijklmno <U15 60 140
abcdefghijklmnop <U16 64 144
abcdefghijklmnopq <U17 68 148
abcdefghijklmnopqr <U18 72 152
abcdefghijklmnopqrs <U19 76 156
abcdefghijklmnopqrst <U20 80 160
abcdefghijklmnopqrstu <U21 84 164
abcdefghijklmnopqrstuv <U22 88 168
abcdefghijklmnopqrstuvw <U23 92 172
abcdefghijklmnopqrstuvwx <U24 96 176
abcdefghijklmnopqrstuvwxy <U25 100 180
abcdefghijklmnopqrstuvwxyz <U26 104 184
abcdefghijklmnopqrstuvwxyza <U27 108 188
abcdefghijklmnopqrstuvwxyzab <U28 112 192
abcdefghijklmnopqrstuvwxyzabc <U29 116 196
abcdefghijklmnopqrstuvwxyzabcd <U30 120 200
abcdefghijklmnopqrstuvwxyzabcde <U31 124 204
abcdefghijklmnopqrstuvwxyzabcdef <U32 128 208
abcdefghijklmnopqrstuvwxyzabcdefg <U33 132 212
abcdefghijklmnopqrstuvwxyzabcdefgh <U34 136 216
abcdefghijklmnopqrstuvwxyzabcdefghi <U35 140 220
```

abcdefghijklmnopqrstuvwxyzabcdefghij <U36 144 224

abcdefghijklmnopqrstuvwxyzabcdefghijk <U37 148 228

abcdefghijklmnopqrstuvwxyzabcdefghijkl <U38 152 232

abcdefghijklmnopqrstuvwxyzabcdefghijklm <U39 156 236

abcdefghijklmnopqrstuvwxyzabcdefghijklmn <U40 160 240

abcdefghijklmnopqrstuvwxyzabcdefghijklmno <U41 164 244

abcdefghijklmnopqrstuvwxyzabcdefghijklmnop <U42 168 248

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopq <U43 172 252

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqr <U44 176 256

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrs <U45 180 260

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrst <U46 184 264

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstu <U47 188 268

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuv <U48 192 272

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvw <U49 196 276

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwx <U50 200 280

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxy <U51 204 284

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz <U52 208 288