SE213 2016

Lecture 12: numpy

오늘 다룰 내용

- numpy
  - numpy 사용 (import)
  - ndarray 생성
    - 생성함수: array(), ones(), zeros(), empty(), ones_like(), …
    - 난수 발생
  - 데이터 타입(dtype)
  - 연산
  - 색인/슬라이싱
  - 유니버셜 함수
- matplotlib에 응용

numpy 소개

- 기능
  - 고성능(빠른 실행 속도)의 다차원 배열(ndarray) 제공
  - 다양한 함수
  - C/C++ 혹은 Fortran 등 다른 언어와 쉽게 사용 가능
  - 선형대수, 푸리에 변환, 난수 발생기 제공
- 응용
  - 과학/공학의 계산
  - 일반적인 다차원 배열 → 각종 database 등에도 응용됨

numpy 사용

- numpy import 방법

```
import numpy as np
```

- numpy에 정의된 상수

```
np.pi
```

```
3.141592653589793
```

```
np.e
```

```
2.718281828459045
```

numpy array 생성: list에서 생성

```
arr = np.array([42, 23, 6])

print(arr)
print(type(arr))
print(arr.ndim)    # 차원
print(arr.shape)   # 차원 당 아이템 수
```

```
[42 23  6]
<class 'numpy.ndarray'>
1
(3,)
```

```
arr2d = np.array([[42, 23, 6],
                  [0, 1, 1024]])
print(arr2d)
print(type(arr2d))
print(arr2d.ndim)    # 차원
print(arr2d.shape)   # 차원 당 아이템의 수
```

```
[[   42    23     6]
 [    0     1  1024]]
<class 'numpy.ndarray'>
2
(2, 3)
```

numpy array 생성: 함수를 이용 (1/3)

```
np.arange(5)
```

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

```
np.arange(1, 6, 2) # start, stop, step
```

```
array([1, 3, 5])
```

```
np.linspace(0, 2, 5)
```

```
array([ 0. ,  0.5,  1. ,  1.5,  2. ])
```

```
np.linspace(0, 2, 5, endpoint=False)
```

```
array([ 0. ,  0.4,  0.8,  1.2,  1.6])
```

numpy array 생성: 함수를 이용 (2/3)

```
np.zeros((2, 3)) # tuple을 사용!
```

```
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

```
t = [42, 23, 6]
np.zeros_like(t)
```

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

```
np.ones((2, 3)) # tuple을 사용!
```

```
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

numpy array 생성: 함수를 이용 (3/3)

```
# 메모리만 할당받 초기화를 하지 않음
# 메모리에 0, 1 혹은 어떤 값이 할당되어 있을 수 있음
np.empty((2, 3)) # tuple을 사용!
```

```
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

```
np.eye(3) # 단위 행렬 생성
```

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

numpy array 생성: 난수 발생 (1/2)

- numpy.random
  - 정규분포 등 다양한 형태의 난수를 만들어 줌
  - ndarray를 반환하는 함수들이 많아 초기화에 편리함
- 참고: random seed를 설정하면 같은 순서로 난수가 발생됨
  → 디버깅할 때 유용

```
# seeds the random generator (optional)
np.random.seed(1234)
```

```
# generate random numbers in [0, 1)
np.random.rand(2, 3)
```

```
array([[ 0.95080985,  0.32570741,  0.19361869],
       [ 0.45781165,  0.92040257,  0.87906916]])
```

```
# generate random numbers
# from the standard normal distribution
np.random.randn(2, 3)
```

```
array([[ 3.21999894,  0.22348686,  1.1428329 ],
       [ 0.72485469,  0.92849521,  0.65091101]])
```

numpy array 생성: 난수 발생 (2/2)

- 정수인 난수 발생, 아래 두 함수 중 하나를 이용
  - np.random.randint()
  - np.random.random_integers()

```
# generate random integers in [0, 3)
np.random.randint(3, size=(2,3))
```

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

```
# generate random integers in [1, 3]
np.random.random_integers(3, size=(2,3))
```

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

```
# generate random integers in [5, 7)
np.random.randint(5, 7, size=(2,3))
```

```
array([[6, 6, 5],
       [5, 5, 6]])
```

```
# generate random integers in [5, 7]
np.random.random_integers(5, 7, size=(2,3))
```

```
array([[7, 7, 7],
       [7, 6, 5]])
```

Reference: numpy array 생성 함수

함수	설명
array()	입력데이터인 시퀀스(리스트, 튜플, 배열 등)를 복사하여 ndarray 생성
asarray()	입력 데이터를 ndarray로 변환, 입력 데이터가 ndarray인 경우, 복사가 되지 않는다
arange()	range() 함수와 유사한 형태로 ndarray 반환
ones(), ones_like()	모든 값을 1로 채운 ndarray 생성. ones_like()는 주어진 입력과 같은 크기의 ndarray생성
zeros(), zeros_like()	위와 유사, 단 1이 아닌 0으로 초기화
empty(), empty_like()	위와 유사, 단 메모리를 초기화하지 않는다 (garbage값, 즉 의미없는 값들이 채워져 있다)
eye(), identity()	N x N 단위 행렬을 반환
linspace()	선형 벡터 생성 (특정 범위를 등분으로 나눈 숫자들을 생성)
random.rand()	[0, 1) 사이의 난수 발생
random.randn()	정규 분포를 따르는 난수 발생
random.randint(), random.random_integers()	정수형의 난수 발생

데이터 타입(dtype) 예제 (1/2)

- ndarray의 모든 아이템은 값은 데이터 타입(dtype)을 가짐
  - 같은 데이터 타입을 가지기 때문에 성능 상의 이점이 생김
    (메모리 상에서 각 아이템이 동일한 크기를 가짐)
  - `float64` 혹은 `int64` (플랫폼마다 다름), `object` 등이 주로 사용됨
- 생성함수에서 키워드 인자 dtype을 이용하여, 원하는 데이터 타입을 지정할 수 있음
- 참고
  - bit: 2진수 한 자리, 즉 0 혹은 1
  - byte: 8 bit

```
a_d = np.ones(3, dtype=np.int64)
print(a_d)
print(a_d.dtype)
a_f = np.ones(3, dtype=np.float64)
print(a_f)
print(a_f.dtype)
```

```
[1 1 1]
int64
[ 1.  1.  1.]
float64
```

데이터 타입(dtype) 예제 (2/2)

- python의 모든 데이터를 object
  → object 데이터 형의 경우,
  리스트처럼 어떤 데이터도 사용 가능
- 그러나 주로 동일한 클래스의
  인스턴스를 저장하기 위하여 사용됨
  - 같은 리스트 혹은 ndarray에 여러 데이터형을 저장할 필요가 있는 경우가 많지 않음

```
a = np.array([1, 2, 3], dtype='O')
```

```
a[0] = 'abc'
a[1] = 3
a[2] = 3.5
```

```
print(a)
```

```
['abc' 3 3.5]
```

Reference: 데이터 타입 (data type)

종류	type code	설명
int8, uint8	i1, u1	부호가 있는/없는 8비트 정수형
int16, uint16	i2, u2	부호가 있는/없는 16비트 정수형
int32, uint32	i4, u4	부호가 있는/없는 32비트 정수형
int64, uint64	i8, u8	부호가 있는/없는 64비트 정수형
float16	f2	반정밀도 부동소수점
float32	f4 또는 f	단정밀도 부동소수점. C언어의 float*에 해당
float64	f8 또는 d	배정밀도 부동소수점. C언의 double*, python의 float에 해당
float128	f16 또는 g	확정정밀도 부동소수점
complex64, complex128, complex256	c8, c16, c32	각각 2개의 32비트, 64비트, 128비트 부동소수점을 가지는 복소수
bool	?	불리언 형
object	O	python 객체형
string_	S	고정일기 문자열. 길이가 10인 문자열의 dtype은 S10
unicode_	U	고정 길이 유니코드형. 길이가 10인 유니코드 문자열은 U10

* 플래폼에 따라 차이를 보일 수 있음

연산

- 대부분의 연산자가 ndarray의 각 아이템에 대해서 연산이 됨
- ndarray 사이의 연산은, 각각의 아이템끼리의 연산

```
a = np.array([42, 23, 6])
b = np.array([1, 2, 3])
print(a + b)
print(a + 2*b)
print(b ** 2)
print(1 / b)
```

```
[43 25  9]
[44 27 12]
[1 4 9]
[ 1.          0.5         0.33333333]
```

색인/슬라이싱

- 리스트의 색인/슬라이싱과 유사

열 (column)

행 (row)	0	1	2	3
0	[0, 0]	[0, 1]	[0, 2]	[0, 3]
1	[1, 0]	[1, 1]	[1, 2]	[1, 3]
2	[2, 0]	[2, 1]	[2, 2]	[2, 3]

- 2차원에서 특정 행/열을 인덱싱을 하는 경우에는 차원이 줄어듦

```
a = np.random.randint(100, size=(3,4))
print(a)
print(a[1][2])
print(a[0:2, :])
print(a[0:2, 2:3])
print(a[0][2:3])
print(a[0, 2:3])
```

```
[[76 69 81 80]
 [ 8 75 15 20]
 [16 64 61 96]]
15
[[76 69 81 80]
 [ 8 75 15 20]]
[[81]
 [15]]
[81]
[81]
```

리스트와 ndarray 슬라이싱의 다른 점

- 리스트에서 슬라이싱
  - 새로운 리스트를 생성하고, 필요한 부분의 아이템을 복사
- ndarray에서 슬라이싱
  - 기존 ndarray의 일부 아이템만 접근 가능하는 기능의 제공
  - 데이터의 복사는 없음 → 경우에 따라 훨씬 빠른 실행이 가능함

```
t = [42, 23, 6]
a = np.array(t)
t2 = t[:2] # 새로운 리스트를 생성
a2 = a[:2] # 동일한 ndarray의 일부만 접근 (view)
print(t2)
print(a2)
t2[0] = 7
a2[0] = 7
print(t)
print(a)    # a의 값은 바뀜
```

```
[42, 23]
[42 23]
[42, 23, 6]
[ 7 23  6]
```

불리언 색인

- 비교 연산자도 산술 연산과 마찬가지로 아이템별로 값을 계산
- ndarray와 같은 길이의 (같은 수의 아이템을 가지는) 불리언 배열은 인덱스처럼 사용 가능
  → True인 위치의 값만 가지는 별도의 ndarray 생성
- 복합적인 조건을 이용한 불리언 색인:and, or, not 대신 다음 연산자 사용 가능
  - &: and를 의미
  - |: or를 의미
  - ~: not을 의미

```
a = np.array([1, 7, 3, 9, 6])
print(a)
bool_index = a >= 5
print(bool_index)
a2 = a[bool_index]
print(a2)
print(a[a >= 5]) # 축약형
print(a[(a >= 5) & (a <= 7)])
```

```
[1 7 3 9 6]
[False  True False  True  True]
[7 9 6]
[7 9 6]
[7 6]
```

전치 (transpose)

- nparray.T 를 통하여,
  전치행렬을 접근할 수 있음
  (2차원의 경우)

```
a = np.random.randint(10, size=(2,3))
print(a)
b = a.T # b는 a의 전치 (데이터 복사 안 함)
print(b)
b[2][1] = -1
print(a)
```

```
[[0 7 0]
 [2 8 4]]
[[0 2]
 [7 8]
 [0 4]]
[[ 0  7  0]
 [ 2  8 -1]]
```

유니버설 함수

- 유니버설 함수: ndarray 아이템 각각에 대해 연산을 수행하는 함수
  - http://docs.scipy.org/doc/numpy-1.10.1/reference/ufuncs.html
- 예
  - 산술연산: `add()`, `substract()`, `multiply()`, `divide()`, `log()`, `sqrt()`, …
  - 삼각함수: `sin()`, `cos()`, `tan()`, …
  - 비트연산: `bitwise_and()`, `bitwise_or()`, …
  - 비교: `greater()`, `greater_equal()`, …

유니버설 함수 예제

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

```
np.sqrt(arr)
```

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

```
np.exp(arr)
```

```
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 = np.random.randn(4)
y = np.random.randn(4)
print(x)
print(y)
print(np.maximum(x, y))
```

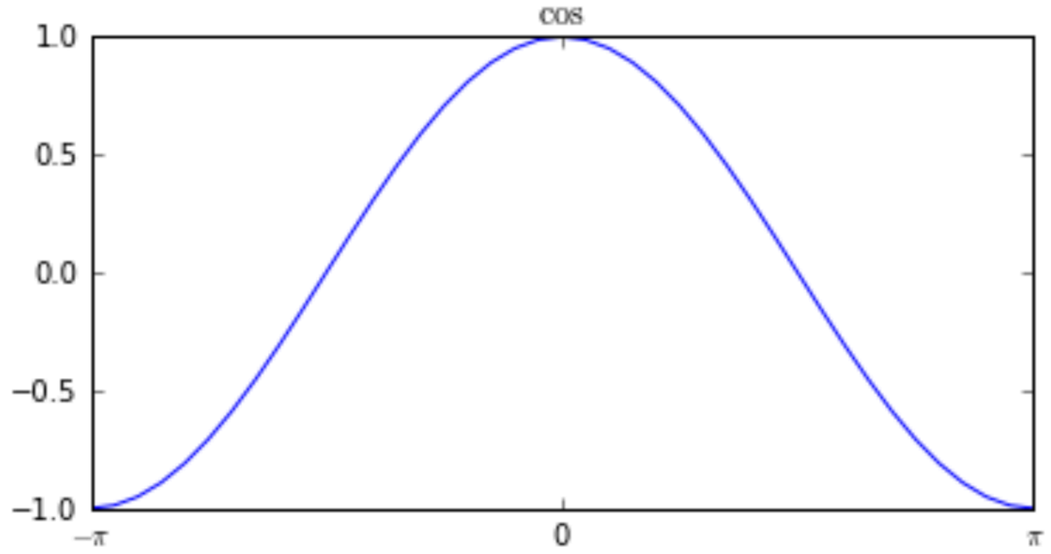
```
[ 1.48871612 -0.82520803 -0.46571086  0.23246849]
[-0.3068199  -0.73555063 -0.1138759  -0.5072815 ]
[ 1.48871612 -0.73555063 -0.1138759   0.23246849]
```

matplotlib에 적용

- matplotlib 내부적으로 numpy를 사용
  - 함수 인자 등에 리스트 대신 ndarray를 사용 가능
  - 많은 경우에 더 편리하게 사용할 수 있음

```
x = np.linspace(-np.pi, np.pi, 41)
plt.figure(figsize=(6,3))
plt.plot(x, np.cos(x))
plt.xlim(-np.pi, np.pi)
plt.xticks([-np.pi, 0, np.pi],
           ['$-\pi$', '0', '$\pi$'])
plt.title('$\cos$')
```

```
<matplotlib.text.Text at 0x1145e3b38>
```



참고 자료

- 공식 사이트: http://www.numpy.org
  - User's guide: http://docs.scipy.org/doc/numpy-dev/user/
  - Reference: http://docs.scipy.org/doc/numpy-dev/reference/
- 튜토리얼
  - http://www.python-course.eu/numpy.php
  - http://www.scipy-lectures.org
- 책 (영문): 파이썬 라이브러리를 활용한 데이터 분석 (4장)
- Matlab과 비교
  - https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

ANY QUESTIONS?