NumPy

NumPy

- Numerical Python
  - 다차원 배열을 위한 확장 패키지
  - Scientific computation을 위해 설계됨
  - 배열 위주의 계산에 효율적임

- 다운로드 및 설치하기

-. http://www.lfd.uci.edu/~gohlke/pythonlibs/

-. python –m pip install **numpy-1.9.3+mkl-cp34-non-win_amd64.whl**

기본 배열 선언

- 행렬 만들기

```
import numpy as np
data = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

- 행렬의 크기

```
data.ndim
data.shape
```

```
len(data) # the size of the first dimension
```

- 행렬의 데이터 형
  - 행렬은 같은 데이터 타입 **data.dtype**

```
data = np.array([[1,2,3],[4,5,6],[7,8,9.]])
* np.array 는 모두 같은 데이터형을 가짐
```

기본 배열 선언

- 데이터형 변경하기

```
data = np.array([[1,2,3],[4,5,6],[7,8,9]])
data.astype(np.float)
data = np.array(['1','2','3'])
data.astype(np.int)
```

- 데이터형 지정하기

```
data = np.array([[1,2,3],[4,5,6],[7,8,9]], dtype=float)
data = np.array([1,2,3], dtype=complex)
```

기본 배열 선언

- 1로 채워진 행렬

```
np.ones((3,2))
```

- 0으로 채워진 행렬

```
np.zeros((3,2)
```

- 단위 행렬 & 대각행렬

```
np.eye((3))
```

```
np.diag(np.array([1,2,3,4]))
```

- 전치 행렬

```
data = np.array([[1,2,3],[4,5,6],[7,8,9]])
data.T
```

기본 배열 선언

- 범위 지정하여 등간격 배열 만들기

```
np.arange(10) # 0, . . . n-1
np.arange(10,1,-1) #start, end(exclusive), step
np.arange(10,1,-1)[:, np.newaxis] #행 증가
np.arange(2, 8, dtype=np.float)
np.arange(35).reshape(5,7)
```

- 지정된 수의 배열

```
np.linspace(1., 4., 6) #start, end, num-points
np.linspace(1., 4., 6, endpoint=False)
```

기본 배열 선언

- 난수 생성
  - Uniform in [0,1]

```
data = np.random.rand(4)
```

  - Gaussian

```
data = np.random.randn(4)
```

인덱스 & 슬라이싱

- 배열의 인덱스는 0을 기반으로 함
  - -1은 끝에서 부터의 인덱스

```
data = np.array([[1,2,3,4],[4,5,6,7],[7,8,9,10]])

data[0]
data[-1]
data[0:2]
data[:2]
data[1:4:2] #start:end:step
data[::-1]
```

```
data[2][0]
data[2,0]
data[1:4:2, ::3]
```

인덱스 & 슬라이싱

- 인덱스 배열

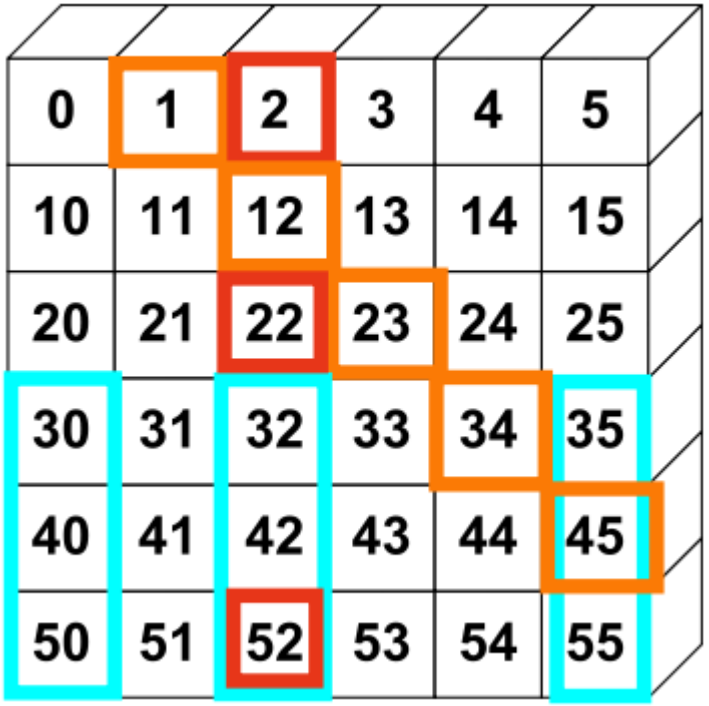
```
x = np.arange(10,1,-1)

x[np.array([3, 3, 1, 8])]
x[np.array([[1,1],[2,3]])]

y=np.arange(35).reshape(5,7)
y[np.array([0,2,4])]
b = y>20
y[b]
```

인덱스 & 슬라이싱

***boolean masks**

```
mask=np.array(np.array([1,0,1,0,0,1],dtype=bool))
data[mask, 2]
```

```
mask1=np.array([0,1,2,3,4])
mask2=np.array([1,2,3,4,5])
data[mask1,mask2]
```



행렬의 연산

- 연산은 **각 요소별 연산을** 수행함

```
data = np.array([[1,2,3],[4,5,6],[7,8,9]])

print(data+2)
print(data-2)
print(data*data)   #not matrix multiplication
print(data/3)
print(data**2)
print(data**0.5)
data.dot(data)   #matrix multiplication
```

행렬의 연산

- 비교연산

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

- Array 비교연산

```
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)
np.array_equal(a, c)
```

행렬의 연산

- 논리연산

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

- 초월함수(**Transcendental functions)**

```
a = np.arange(5)
np.sin(a)
np.log(a)
np.exp(a)
```

행렬의 연산

- 전이행렬

```
a = np.triu(np.ones((3, 3)), 1)
a.T
```

  – triu : 상삼각행렬

행렬의 연산

- Sum

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

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

행렬의 연산

- 최대,최소

```
x = np.array([1, 3, 2])
x.min()
x.max()
x.argmin()  # index of minimum
x.argmax()  # index of maximum
```

행렬의 연산

- 논리 연산

```
np.all([True, True, False])
np.any([True, True, False])

* 배열 비교할 때 주로 사용
a = np.zeros((100,100))
np.any(a!=0)
np.all(a==a)
```

행렬의 연산

- 통계

```
x = np.array([1, 2, 3, 1])
y = np.array([[1, 2, 3], [5, 6, 1]])
x.mean()
np.median(x)
np.median(y, axis=-1) # last axis
x.std()           # full population standard dev.
```

행렬의 연산

- 응용

```
import numpy as np
from matplotlib import pyplot as plt

data = np.loadtxt('data.txt')
year, hares, lynxes, carrots = data.T
plt.plot(year, hares, year, lynxes, year, carrots)
plt.show()
```

  - 년도별 평균 ?
  - 년도별 표준편차 ?
  - 매년 가장 높은 인구를 갖는 종은?