이승준 fb.com/plusjune

Numpy

Numerical Python, "넘피"라고 읽는다.

- http://numpy.org
- 공식 레퍼런스 http://docs.scipy.org/doc/numpy/reference/
- Pandas가 NumPy 기반

특징

수치 데이터를 다루는데 효율적이고 높은 성능을 제공

- 대규모 데이터를 빠르게
- 스칼라 연산과 비슷하게 연산을 수행
- 선형대수 linear algebra, 푸리에 변환 Fourier transform, 랜덤 기능들
- 배열을 특별히 조작하지 않고도 원소별 연산이 가능 – 브로드캐스팅(broadcasting)

ndarray

- ndarray (n-dimensional array object, 다차원 배열 객체)
- NumPy의 핵심
- 모든 요소가 동일한 datatype (기본은 float64)

임포트

```
from numpy import *   # 내장 객체처럼 사용
import numpy          # numpy.obj 형식으로 사용
import numpy as np    # np.obj 형식으로 사용
```

도움말

- np?
- np.array?

NumPy 성능

```
ls = range(1000)
%timeit [i**2 for i in ls]


a = np.arange(1000)
%timeit a**2


# 200배
```

ndarray 생성

np.array() # 리스트, 튜플, 배열로 부터 ndarray를 생성

```
ar = np.array((10, 20, 30)) # 1차원 배열 생성
print (ar.ndim)
print (ar.shape)
```

0에서 시작

- 항상 0부터 시작한다는 점을 기억
- 0으로 시작하는 로우 row, 컬럼 col 으로 읽기를 추천

np.array()로 생성

```
a1 = np.array([0, 1, 2, 3])
a2 = np.array([[0, 1, 2], [3, 4, 5]])    # 2 x 3 array


a2.ndim
a2.shape
```

생성과 초기화

- np.zeros() # ndarray를 생성하고 0으로 초기화
- np.empty() # ndarray을 생성하지만 초기화 하지 않는다.

다양한 생성 방법

- np.arange() # range와 비슷
- np.asarray() # 기존의 array로 부터 ndarray를 생성
  (객체를 따로 생성하지 않는다. 즉, 요소들을 복사하지 않는다)

다차원 배열 생성이 가능하지만, 주로 2차원 배열을 사용

np.arrange()

- ar = np.arange(10) *# 0 .. n-1*
- ar = np.arange(1, 9, 2) *# [start], (end), step*
- ar = np.linspace(0, 10, 15) *# 균일한 간격으로 생성 start, end, num*

ar.reshape()

- ar = np.arange(1, 10).reshape((3,3))

요소 지정

```
ar = np.array( [[10, 20, 30], [60, 70, 80], [90, 91, 92]] )
ar[1][2] # 80
ar[1,2]  # 80 동일
ar[1,1]   # 요소 한개 ar[1][1] 동일
```

데이터 타입

- ndarray을 생성하면서 데이터 타입을 지정할 수 있다.
- 주로 np.int (np.int64), np.float (np.float64) 사용

```
ar = np.array([10, 20, 30, 40])
ar.dtype #dtype('int64')

ar = np.array([10, 20, 30., 40.]) #30., 40. float
ar.dtype #dtype('float64')
```

데이터 타입 바꾸기

ndarray.astype()로 타입을 변경할 수 있다
- 데이터가 새로 복사 된다.
- str 과 float, int 사이에 변환도 가능

```
ar = np.array([10, 20, 30, 40])
af = np.ndarray.astype(ar, np.float)
af.dtype #dtype('float64')
astr = np.ndarray.astype(af, np.str)
astr # array(['10.0', '20.0', '30.0', '40.0'], dtype='<U32')
```

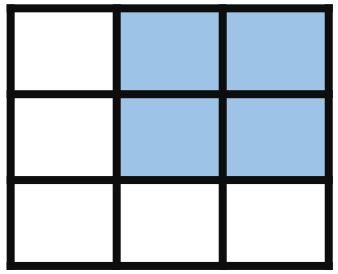
다양한 슬라이싱

- ar = np.arange(0,16).reshape((4,4))
- ar[2,:2] = 888 # 대체해 가면서 테스트 확인

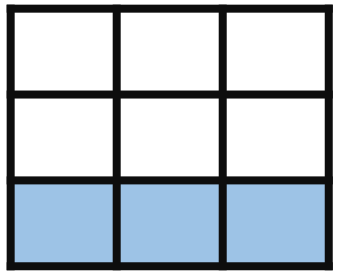
```
ar[:]     # ar의 모든 row (= ar)
ar[0]     # row 0
ar[1]     # row 1
ar[:,1]   # col 1
ar[1:]    # row 1 이후, ar[1:,] ar[1:,:]
ar[:2]    # row 0~1
```

```
ar[:2,1:] # row 0~1, col 1 이후
ar[:2,:2] # row 0~2, col 0~1
ar[2,:2]  # row 2, col 0~1
ar[:,2]   # col 2
ar[:,1:3] # col 1~2
```

슬라이싱 slicing



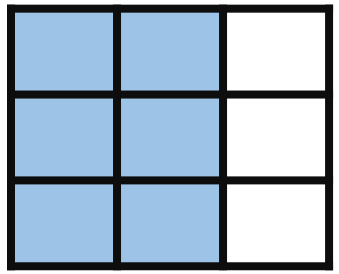
`ar[:2, 1:]`



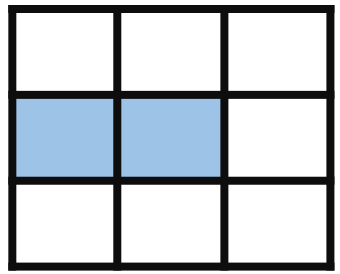
`ar[2]`

`ar[2, :]`

`ar[2:, :]`



`Ar[:, :2]`



`ar[1, :2]`

`Ar[1:2, :2]`

마이너스 인덱스

- ar[-2] # 끝에서 2번째 행
- ar[-2:] # 마지막 2개 행
- ar[:-2] # 마지막 2개 행을 제외한 모든 행

ndarry 연산

ndarray 간에 혹은 ndarray와 단일값 사이에 사칙연산(+, −, , /, *) 이 가능

- ndarray op ndarray
- value op ndarray

```
ar = np.array([[0, 1, 2], [10, 11, 12], [20, 21, 22], [30, 31, 32]])
ar * 10
```

크기가 다른 ndarray 간의 연산을 브로드캐스팅 이라고 한다

연산 (브로드캐스팅)

0	0	0
10	10	10
20	20	20
30	30	30

\+ 10 =

10	10	10
20	20	20
30	30	30
40	40	40

0	0	0
10	10	10
20	20	20
30	30	30

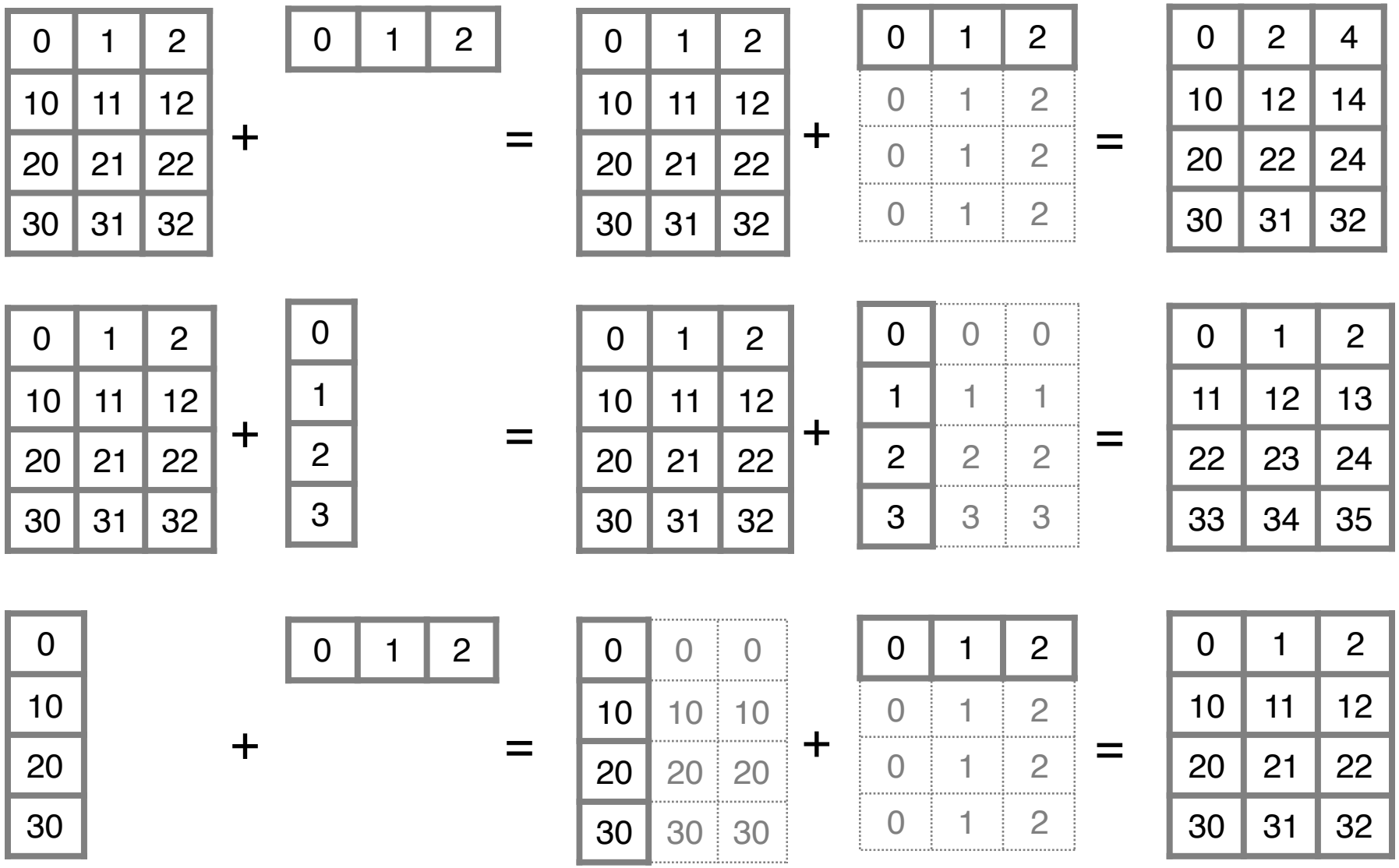
\+

0	0	0
10	10	10
20	20	20
30	30	30

=

0	0	0
20	20	20
40	40	40
60	60	60

연산 (브로드캐스팅)


0 1 2
10 11 12
20 21 22
30 31 32
+
0 1 2
=
0 1 2
10 11 12
20 21 22
30 31 32
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 2 4
10 12 14
20 22 24
30 31 32
0 1 2
10 11 12
20 21 22
30 31 32
+
0
1
2
3
=
0 1 2
10 11 12
20 21 22
30 31 32
+
0 0 0
1 1 1
2 2 2
3 3 3
=
0 1 2
11 12 13
22 23 24
33 34 35
0
10
20
30
+
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 1 2
10 11 12
20 21 22
30 31 32

(계속)

```
ar = np.array([[0, 1, 2], [10, 11, 12], [20, 21, 22], [30, 31, 32]])


ar + np.array([[0, 1, 2]])
ar + np.array([[0], [1], [2], [3]])
np.array([[0], [10], [20], [30]]) + np.array([[0, 1, 2]])
```

슬라이싱과 연산

- ar[n:m] = 20 선택 영역에 브로드캐스팅
- 리스트의 슬라이싱과 차이점은 원본 ndarray에 대한 뷰라는 점
- 뷰에 대한 조작은 그대로 원본에 반영 (별도의 ndarray가 생성되지 않는다)

슬라이싱과 뷰

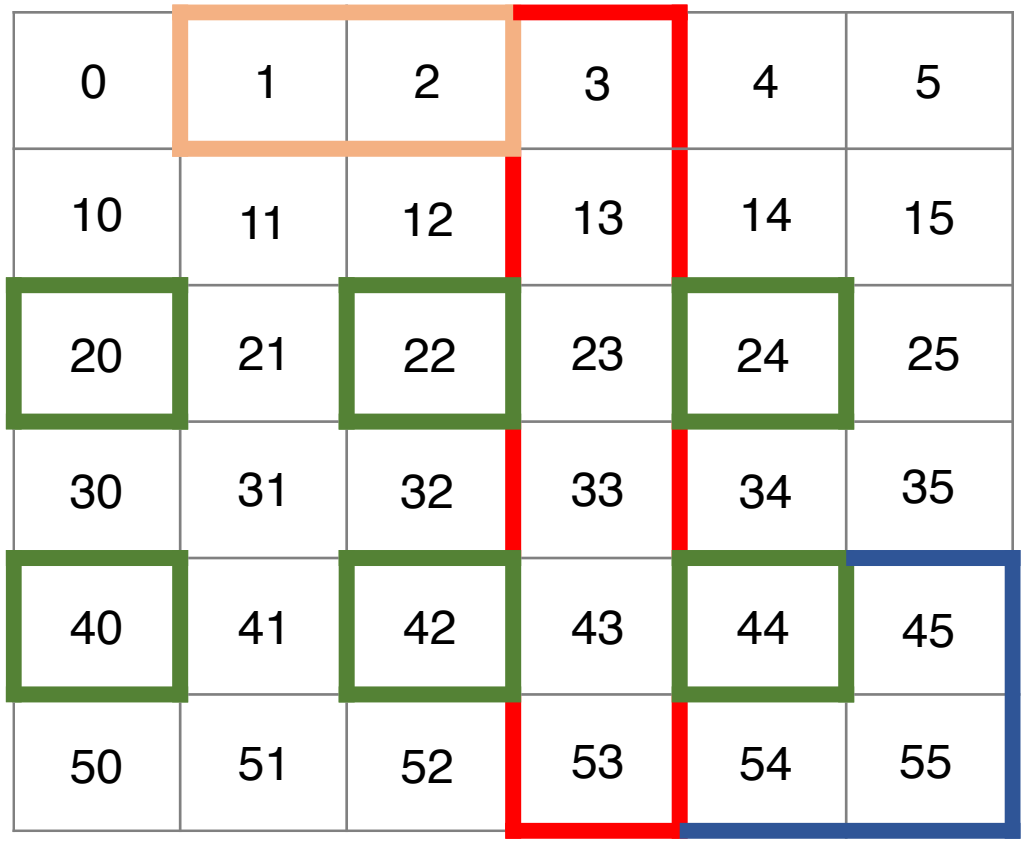
```
ar = np.array([[10, 20, 30], [60, 70, 80], [90, 91, 92]])
v = ar[1]
v[:] = 50 # v 배열 전체 요소에 20을 할당
```

슬라이싱 예제

- ar = np.zeros((4,5))
- v = ar[1:-1,1:-1]
- v[:] = 1

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

다양한 슬라이싱




```
ar[0, 1:2]
```

```
ar[:, 3]
```

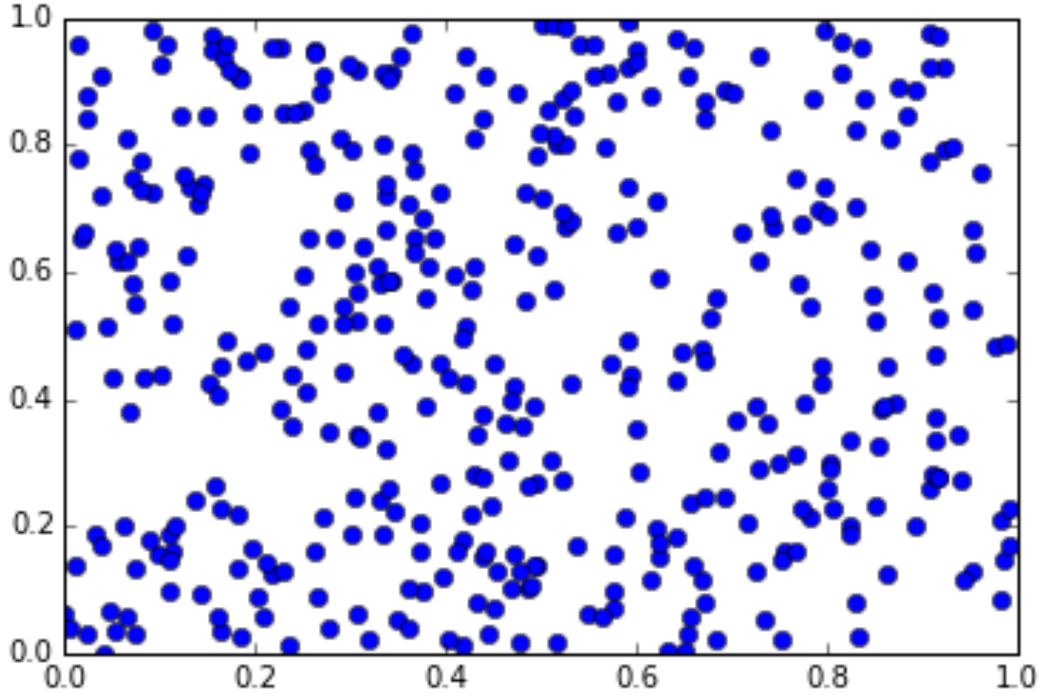
```
ar[4:, 4:]
```

```
ar[2::2, ::2]
```

랜덤 배열 생성

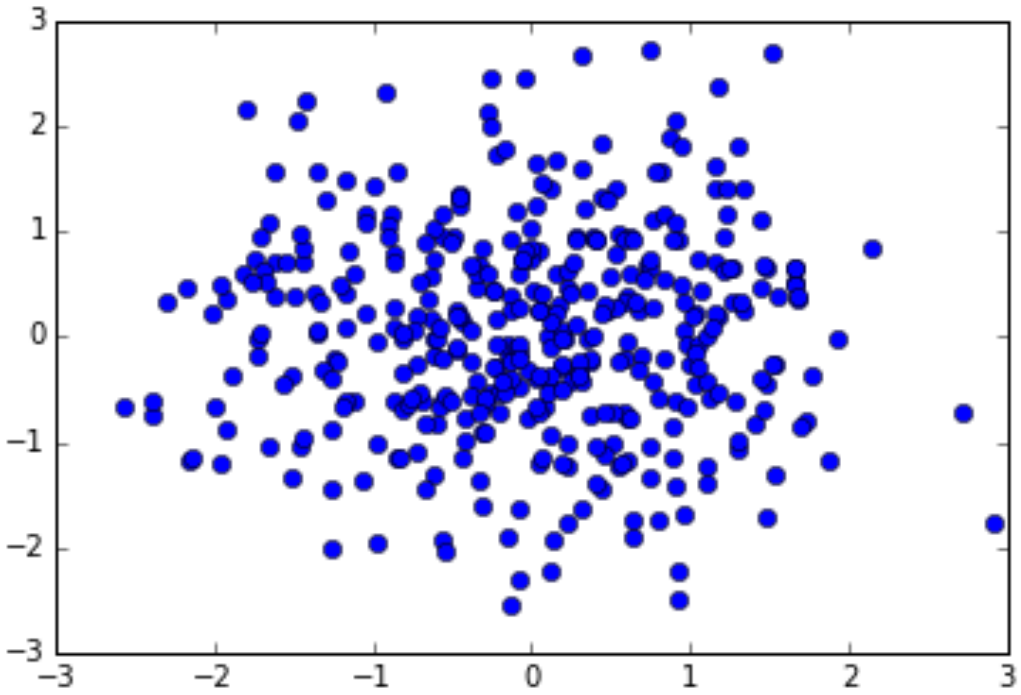
랜덤 숫자로 채워진 ndarray를 생성

```
ar_x = np.random.rand(400)
ar_y = np.random.rand(400)

plt.plot(ar_x, ar_y, 'o')
```



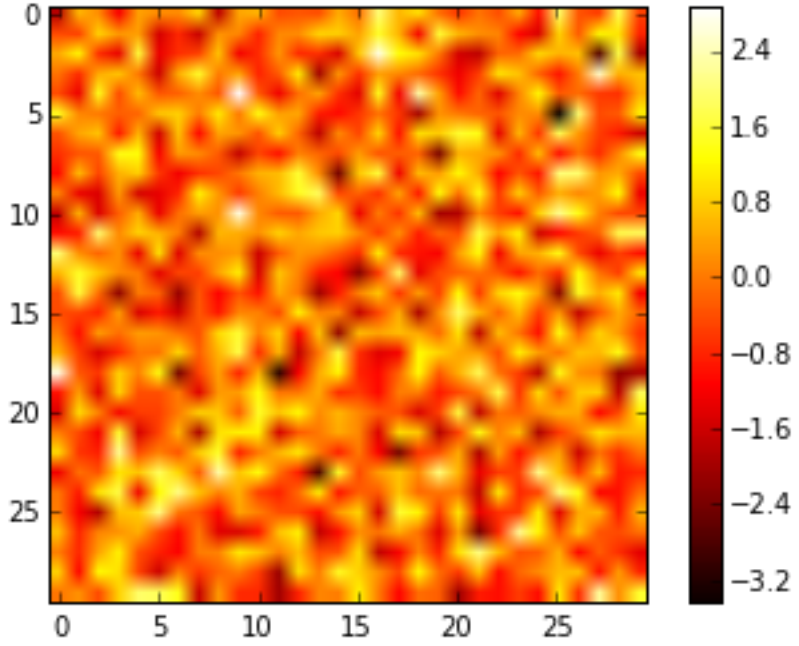
랜덤 배열 생성 (정규분포)

```
ar_x = np.random.randn(400)
ar_y = np.random.randn(400)

plt.plot(ar_x, ar_y, 'o')
```



랜덤 배열 생성 (정규분포)

```
mat = np.random.randn(30, 30)
plt.imshow(mat, cmap=plt.cm.hot)
plt.colorbar()
```



np.getfromtxt()

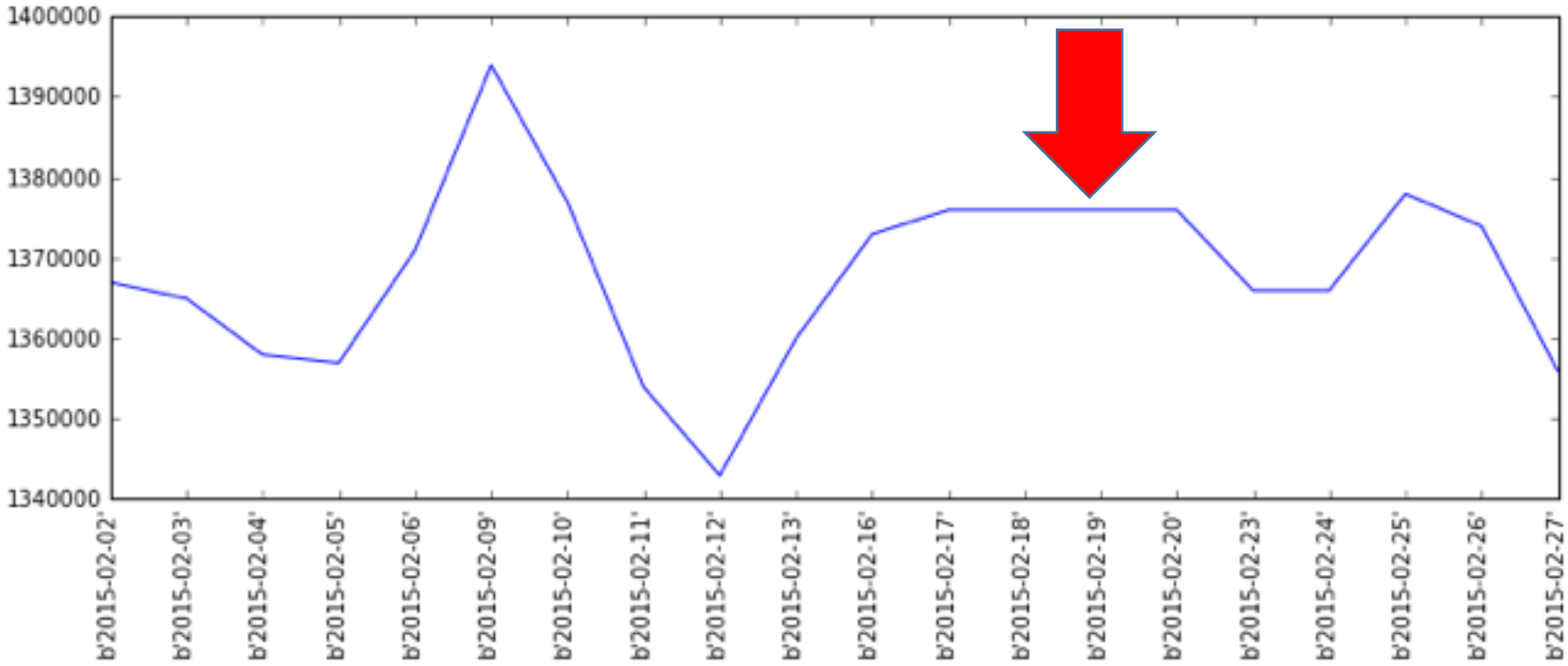
```
import requests
from io import BytesIO

url = "http://real-chart.finance.yahoo.com/table.csv " \
      "?s=005930.KS&a=1&b=1&c=2015&d=1&e=28&f=2015&g=d&ignore=.csv"
dtype = [('Date', 'S10'),('Open', float),('High', float),('Low', float),('Close', float),
('Volume', float), ('Adj_Close', float)]

r = requests.get(url)
xdata = np.genfromtxt(BytesIO(r.content), delimiter=',', skip_header=1,
dtype=dtype)
```

뒤집기 reverse

- data = xdata[::-1]

array([(b'2015-02-02', 1365000.0, 1377000.0, 1356000.0, 1368000.0, 210400.0, 1366929.55),
(b'2015-02-03', 1380000.0, 1380000.0, 1359000.0, 1366000.0, 113000.0, 1364931.12),
(b'2015-02-04', 1375000.0, 1381000.0, 1359000.0, 1359000.0, 186500.0, 1357936.6),
(중략)
(b'2015-02-16', 1368000.0, 1374000.0, 1361000.0, 1374000.0, 124500.0, 1372924.86),
(b'2015-02-17', 1374000.0, 1377000.0, 1364000.0, 1377000.0, 114900.0, 1375922.51),
(b'2015-02-18', 1377000.0, 1377000.0, 1377000.0, 1377000.0, 0.0, 1375922.51),
(b'2015-02-19', 1377000.0, 1377000.0, 1377000.0, 1377000.0, 0.0, 1375922.51),
(b'2015-02-20', 1377000.0, 1377000.0, 1377000.0, 1377000.0, 0.0, 1375922.51),
(b'2015-02-23', 1378000.0, 1390000.0, 1366000.0, 1367000.0, 306000.0, 1365930.34),

```
y = data['Adj_Close']
plt.figure(figsize=(12,4))
ticks = range(0, len(y))
plt.xticks(ticks, data['Date'], rotation='vertical')
plt.plot(ticks, y)
```



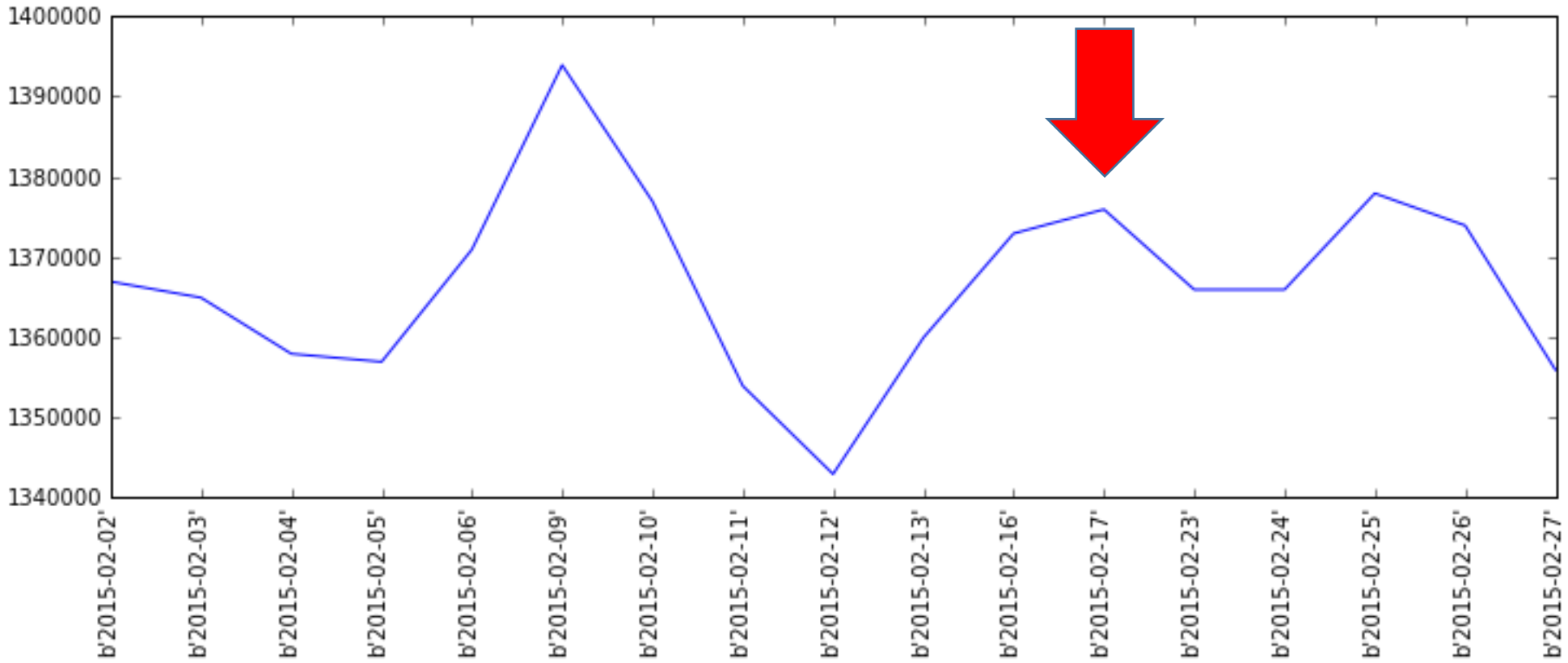
불리언 인덱싱

```
city = np.array(['newyork', 'seoul', 'shanghai', 'tokyo', 'london'])
val = np.array([10, 20, 30, 40, 50])

city == 'seoul'          # array([False,  True, False, False, False],
dtype=bool)
val[city == 'seoul']   # array([20])
city[val >= 30]          # array(['shanghai', 'tokyo', 'london'], dtype='<U8')
city[city != 'tokyo']   # array(['newyork', 'seoul', 'shanghai', 'london']

mask = (city == 'shanghai') | (city == 'london')
val[mask]                  # array([30, 50])
```

불리언 인덱싱 (필터링)

```
# 0 이하를 모두 0으로 만들기
x = np.array([-2, 0, 4, 6, -8, 5, 9, 2])
x[x < 0] = 0
```

```
mask = data['Volume'] > 0

masked_data = data[mask]

y = masked_data['Adj_Close']
```



팬시 인덱싱 (fancy indexing)

정수 리스트(혹은 배열)를 이용하여 여러 개를 동시에 선택

- 팬시 인덱싱은 복사가 일어난다.
- ar[[3,4,5,2,0]] # 지정된 행을 순서대로 추출

```
ar = np.random.randint(1,11,size=(5,5))


ar[[2,4,1,0,3]]
ar[[-2,-3,1]]
```

소트

```
ar = np.random.randint(0,10,5)
ar.sort()
```

소트 – 분위수 구하기

```
# 상위 25%가 되려면 몇 점 이상이어야 할까?


a = random.randn(100) * 100 #임의로 100개의 점수
a.sort()
a[int(0.75 * len(a))]
```

통계량

- np.mean(ar), ar.mean()
- ar.mean(axis=1): column 단위 산술평균
- ar.sum(0): row 단위 합계

2d 배열

- ar.cumsum(0)
- ar.cumprod(1)

기본통계

- sum, mean, std, var, min, max, avgmin avgmax, cumsum, cumprod

통계와 axis

- r = np.arange(20).reshape(4,5)
- r.mean(axis=**None**) *# flat values 9.5*
- r.mean(axis=0) *# array([7.5,   8.5,   9.5, 10.5, 11.5])*
- r.mean(axis=1) *# array([2.,   7., 12., 17.])*

시뮬레이션 – 베르누이 실행

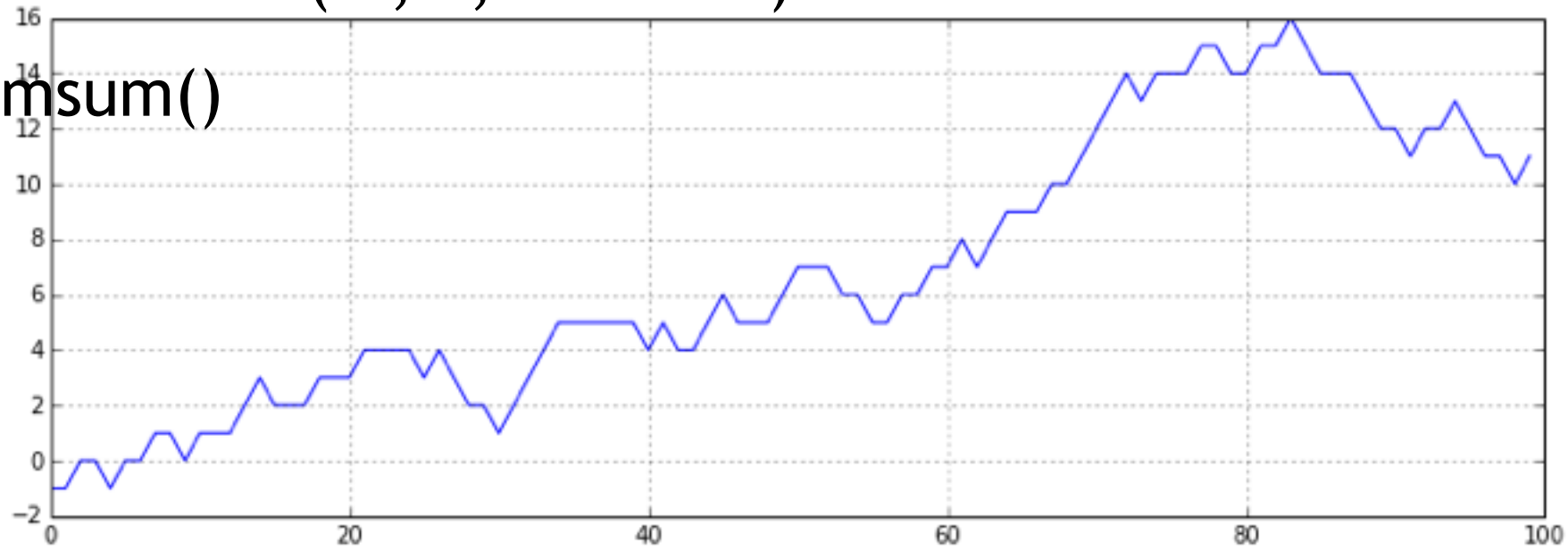
```
import numpy as np
from numpy.random import randint


steps = 100
movements = randint(0, 2, size=steps)
print (movements)
```

```
[0 1 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0
 1 1 1 0 1 0 1 0 0 1 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 0 0 0 1 0 1 1 1 1 0 0
 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 1 0 0 1 1 1]
```

랜덤워크

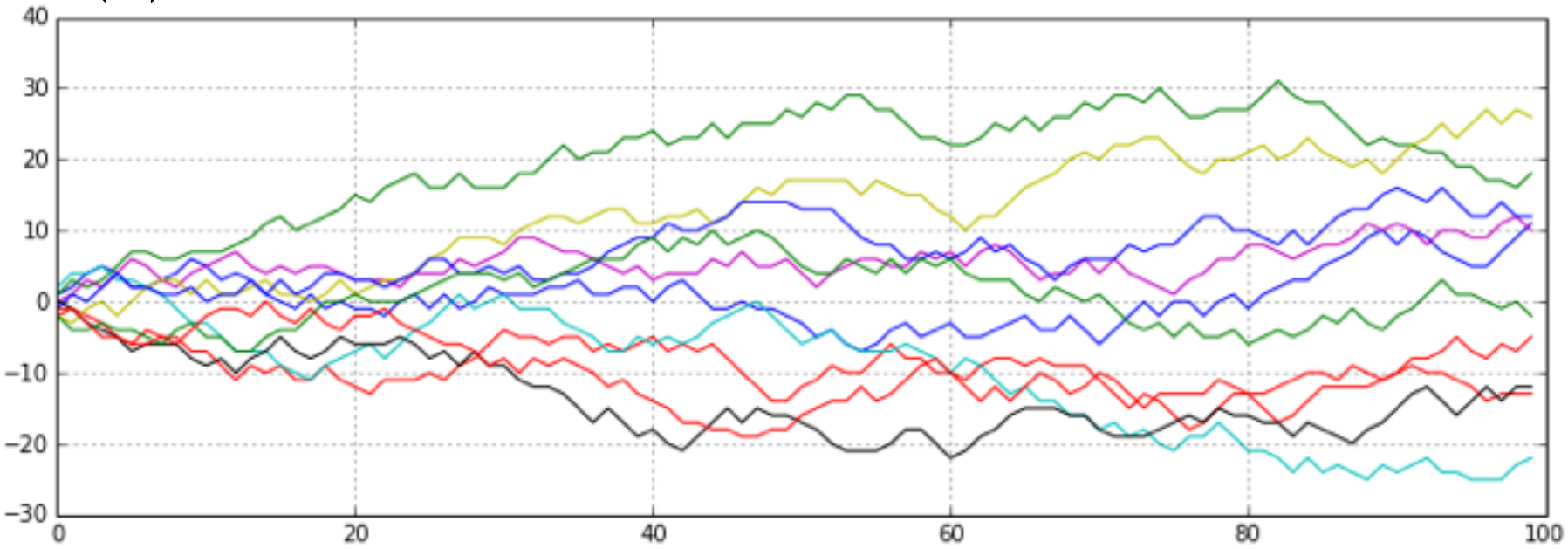
이전 가격대비 −1, 0, 1 사이에서 움직인다고 하자.
(가격 상승, 동일, 하락)

```
steps = np.random.randint(-1, 2, size=100)

walks = steps.cumsum()

plt.plot(walks)
```

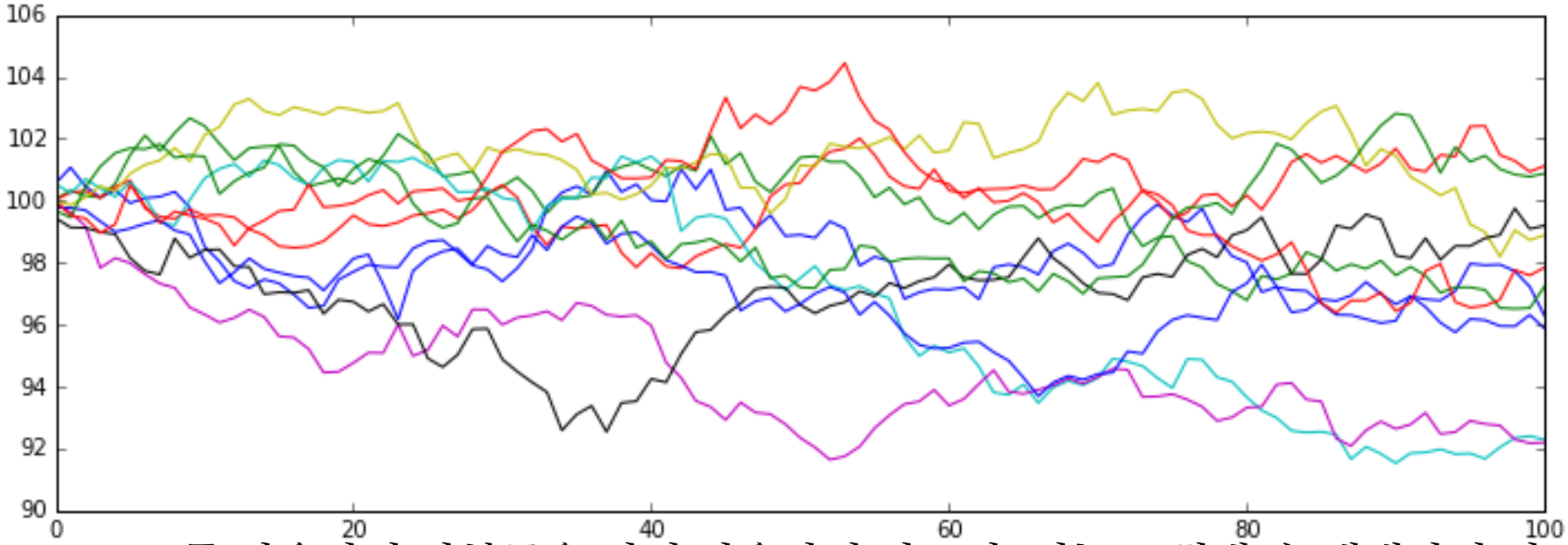


랜덤워크 (다수 종목)

```
# 100 walk를 10개 종목에 대해 실행
t = np.random.randint(-2, 3, size=(100,10))
walks = t.cumsum(0)
plt.plot(walks)
```



```
def random_walk(initial_value = 100, n = 10, tries = 100, volatility = 0.005):
    r = np.random.normal(size=(tries+1, n)) * volatility
    return initial_value * exp(np.cumsum(r, axis=0))


plt.plot(random_walk(n=10))
```



Numpy를 사용하면 반복문을 거의 사용하지 않으며, 성능도 최대 수 백배까지 빠르다.

정규분포 random.normal()

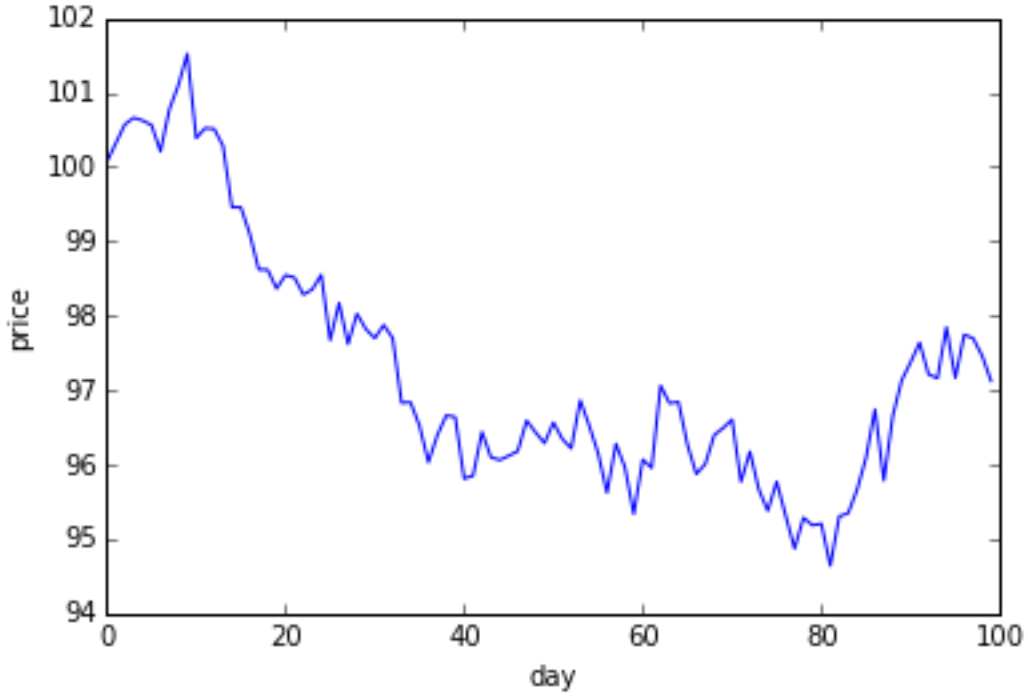
```
initial_value = 100.0
days = 100
random_numbers = np.random.normal(size=days) * 0.005
multipliers = 1 + random_numbers
values = initial_value * np.cumprod(multipliers)
```

```
plt.plot(values)
```
가격

```
plt.plot(random_numbers + 1)
```
수익률

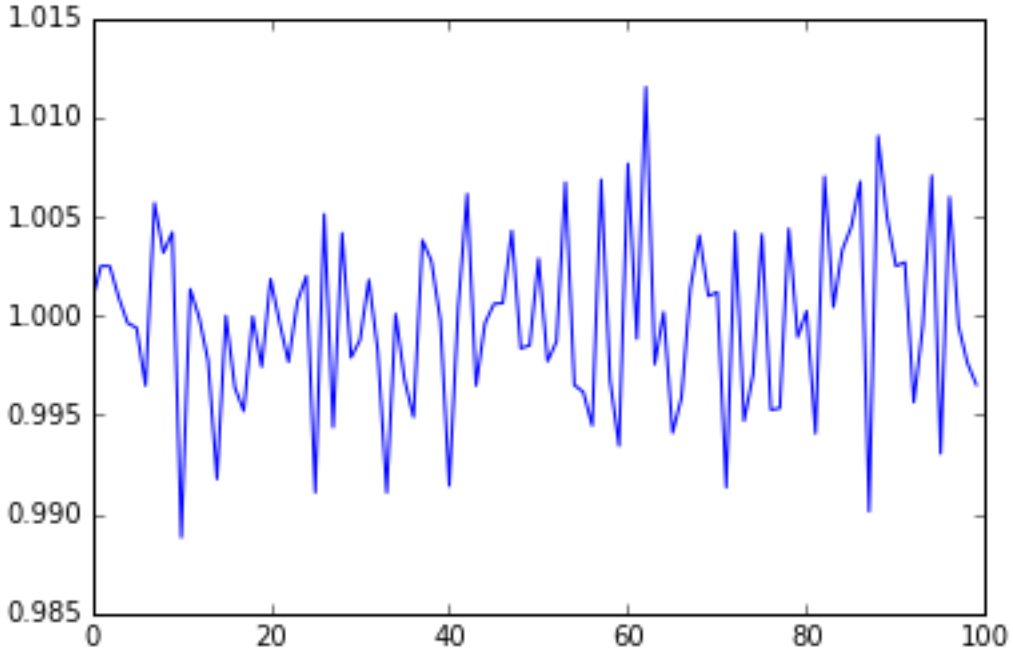
plt.plot(values)

가격



plt.plot(random_numbers + 1)

수익률



로그 수익률

```
from numpy import diff, log


prices = values
log_returns = diff(log(prices))
```

리뷰

- Ndarray
- 슬라이싱 slicing
- 불리언 인덱싱 (필터링)
- 팬시 인덱싱
- 소트
- 랜덤 넘버 생성
- Price Simulation, Random walk, 정규분포