CHAPTER

04

NumPy 라이브러리

NumPy Library

컴퓨터소프트웨어공학과
김대영

개 요

- 배열 지향 프로그래밍
- 파일 입출력
- NumPy 예제 (Random Walk)

- **배열지향 프로그래밍**: sqrt(x^2 + y^2)

```
points = np.arange(-5, 5, 0.01)

# np.meshgrid function takes two 1D arrays and produces two 2D matrices
#   corresponding to all pairs of (x, y)
x, y = np.meshgrid(points, points)

z = np.sqrt(x**2 + y**2)
print(z)

# visualization of the 2d array
plt.imshow(z, cmap=plt.cm.gray)
plt.colorbar()
plt.title("Image plot of $\sqrt{x^2 + y^2}$ for a grid of values")
plt.show()
```

- 배열 연산에 대한 조건부 표현식

```
# case 1
 xarr = np.array([1.1, 1.2, 1.3, 1.4, 1.5])
 yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])
 cond = np.array([True, False, True, True, False])

 # we wanted to take a value from xarr whenever the corresponding value
 #  in cond is True, and otherwise take the value from yarr.
 result = np.where(cond, xarr, yarr)
 print(result)

# case 2
 array = np.random.randn(4, 4)
 print(array)
 print(array > 0)

 # replace all positive values with 2 and all negative values with -2.
 ret = np.where(array > 0, 2, -2)
 print(ret)
```

- **NumPy 수학 및 통계 메소드**

```
arr = np.random.randn(5, 4)
print(arr)
ret = arr.mean()
print(ret)
ret = np.mean(arr)
print(ret)
ret = arr.sum()
print(ret)

ret = arr.mean(axis=1)  # compute mean across the columns (i.e., for row)
print(ret)
ret = arr.sum(axis=0)   # compute sum down the rows (i.e., for column)
print(ret)
```

- **NumPy 수학 및 통계 메소드**

```
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7])
ret = arr.cumsum()
print(ret)

arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
ret = arr.cumsum(axis=0)
print(ret)
ret = arr.cumprod(axis=1)
print(ret)
```

- **NumPy 수학 및 통계 메소드**

sum	배열 원소의 합을 계산
mean	배열 원소의 산술 평균 계산
std, var	표준편차와 분산 계산
min, max	최소값과 최대값 계산
argmin, argmax	최소 원소의 색인값과 최대 원소의 색인값 리턴
cumsum	각 원소의 누적합 계산
cumprod	각 원소의 누적곱 계산

- **불린 배열(Boolean Array)을 위한 메소드**

```
array = np.random.randn(100)
ret = (array > 0).sum()     # Number of positive values
print(ret)
```

  - any(): 배열 원소 중에서 True가 있는지 확인
  - all(): 모든 배열 원소가 True인지 확인

```
bools = np.array([False, False, True, False])
ret = bools.any()
print(ret)
ret = bools.all()
print(ret)
```

- **정렬(Sort)**

```
array = np.random.randn(6)
ret = np.sort(array)
```

  - 2차원 배열의 정렬

```
arr = np.random.randn(5,3)
print(arr)
> [[-0.16614061  1.76038061  2.38428095]
   [-0.5785766   0.75558415 -2.62009326]
   [ 0.82945901  0.45435193 -0.35198793]
   [-0.32622354  1.97977021 -1.6318447 ]
   [ 0.1303902   1.92265247  0.86047697]]
ret = np.sort(arr)
print(ret)
> [[-0.16614061  1.76038061  2.38428095]
   [-2.62009326 -0.5785766   0.75558415]
   [-0.35198793  0.45435193  0.82945901]
   [-1.6318447  -0.32622354  1.97977021]
   [ 0.1303902   0.86047697  1.92265247]]
```

- **집합에서의 Unique()**

```
names = np.array(["Bob", "Joe", "Will", "Bob", "Will", "Joe", "Joel"])
set = np.unique(names)
print(set)
> ['Bob' 'Joe' 'Joel' 'Will’]

ints = np.array([3, 3, 3, 2, 2, 1, 1, 4, 4])
set = np.unique(ints)
print(set)
> [1 2 3 4]
```

  - in1d(): 인자로 주어진 배열의 원소가 포함되어 있는지 확인하고, 그 결과를 불린(Boolean)으로 리턴

```
values = np.array([6, 0, 0, 3, 2, 5, 6])
set = np.in1d(values, [2, 3, 6])
print(set)
> [ True False False  True  True False  True]
```

- **배열 집합 메소드**

unique(x)	배열 x에 대해서 중복되지 않은 원소 리턴
intersect1d(x, y)	배열 x와 y에서 교집합에 해당하는 원소 리턴
union1d(x, y)	배열 x와 y에서 합집합에 해당하는 원소 리턴
in1d(x, y)	배열 x의 원소가 배열 y의 원소에 포함하는지 확인
setdiff1d(x, y)	배열 x와 y의 차집합 원소 리턴
setxor1d(x, y)	하나의 배열에 포함되지만 두 배열 모두 포함되지 않는 원소 리턴

- **배열 데이터의 파일 입출력**
  - NumPy는 텍스트 및 바이너리 포맷의 파일로 데이터를 저장하거나 읽어올 수 있는 기능을 지원함
  - load/save 메소드를 사용하여 쉽게 파일 입출력 처리 가능
  - NumPy 배열은 압축되지 않은 바이너리 포맷의 파일로 저장되고, npy 형식의 확장자를 가짐

```
arr = np.arange(10)
np.save("some_array", arr)

ld = np.load("some_array.npy")
print(ld)
> [0 1 2 3 4 5 6 7 8 9]
```

- **배열 데이터의 파일 입출력**
  - *np.savez*를 사용하여 압축되지않은 묶음 형식으로 다중 배열을 저장할 수 있음
  - *.npz* 파일을 로드하게 되면 배열 데이터는 사전 형식의 객체로 불러옴

```
arr = np.arange(10)
np.savez("array_archive.npz", a=arr, b=arr)

arch = np.load("array_archive.npz")
print(arch['b’])
> [0 1 2 3 4 5 6 7 8 9]
```

- **선형대수**(Linear Algebra)
  - 배열 기반 프로그래밍에서 선형대수는 매우 중요함

```
x = np.array([[1., 2., 3.], [4., 5., 6.]])
print(x)
> [[1. 2. 3.]
   [4. 5. 6.]]

y = np.array([[6., 23.], [-1., 7.], [8., 9.]])
print(y)
> [[ 6. 23.]
   [-1.  7.]
   [ 8.  9.]]

ret = x.dot(y)
print(ret)
> [[ 28.  64.]
   [ 67. 181.]]
```

- **선형대수**(Linear Algebra)
  - *x.dot(y)*는 *np.dot(x, y)*와 동일

```
x = np.array([[1., 2., 3.], [4., 5., 6.]])
y = np.array([[6., 23.], [-1., 7.], [8., 9.]])
ret = np.dot(x,y)
print(ret)
> [[ 28.  64.]
   [ 67. 181.]]
```

  - 파이썬 v3.5 이후부터 @ 사용가능

```
x = np.array([[1., 2., 3.], [4., 5., 6.]])
ret = x @ np.ones(3)    #np.dot(x, np.ones(3))
print(ret)
> [ 6. 15.]
```

- **선형대수**(Linear Algebra)

diag	1차원 배열을 대각선 원소로 하고 나머지는 0으로 채운 단위행렬 반환
dot	행렬 곱셈
trace	행렬의 대각선 원소 합 계산
det	행렬식 계산
eig	정사각 행렬의 고유값(eigenvalue)과 고유벡터(eigenvector) 계산
inv	정사각 행렬의 역행렬 계산
pinv	정사각 행렬의 Moor-Penrose pseudo-inverse 계산
qr	QR 분해 계산
svd	특이값 분해(singular value decomposition)
solve	Ax = b의 선형 시스템 계산
lstsq	Ax = b를 만족하는 최소제곱해 계산

- **선형대수**(Linear Algebra)
  - numpy.linalg 는 행렬의 분할과 역행렬과 같은 행렬 분해(matrix decomposition)의 표준 집합을 지원함

```
X = np.random.randn(3, 3)
mat = X.T.dot(X)
print(mat)
> [[ 0.56972829 -0.15284807  0.35255448]
   [-0.15284807  0.28775704 -0.21737376]
   [ 0.35255448 -0.21737376  2.72457081]]

imat = inv(mat)
print(imat)
> [[ 2.14323231  0.98849998 -0.1984651 ]
   [ 0.98849998  4.15394365  0.20350296]
   [-0.1984651   0.20350296  0.40894733]]

dimat = mat.dot(imat)
print(dimat)
> [[ 1.00000000e+00  3.34186014e-18 -4.12395594e-17]
   [-7.64960762e-17  1.00000000e+00  1.37698489e-17]
   [ 7.64723374e-18  1.91840532e-16  1.00000000e+00]]
```

- **난수생성**(Random Number Generation)
  - *numpy.random* 모듈은 확률 분포의 다양한 종류로 부터 표본값을 생성하기 위해 파이썬 내장 함수를 지원함

```
# normal distribution
samples = np.random.normal(size=(4,4))
print(samples)
> [[-0.6144581  -0.97760137  1.74676881  0.60687775]
   [-0.3006372   0.14749422 -0.13586784 -1.83516092]
   [ 2.01188768  0.35534939  1.07828769 -0.65055445]
   [ 0.58183459 -0.89083568  1.34206662  0.7345279 ]]
```

  - 파이썬 내장 random 모듈은 한 번에 하나의 샘플값만 생성
  - numpy.random 모듈은 큰 표본을 생성하는데 파이썬 내장 random 모듈보다 우수함

- **난수생성**(Random Number Generation)
  - *np.random.seed*를 사용하여 난수 생성 변경

```
np.random.seed(1234)
```

  - *numpy.random*은 전역 난수 시드(global random seed) 사용
    - 전역 상태를 피하기 위해, *numpy.random.RandomState* 를 사용하여 다른 난수 생성기와 분리할 수 있음

```
rng = np.random.RandomState(1234)
ret = rng.randn(10)
print(ret)
> [ 0.47143516 -1.19097569  1.43270697 -0.3126519  -0.72058873
    0.88716294  0.85958841 -0.6365235   0.01569637 -2.24268495]
```

- **numpy.random 함수**

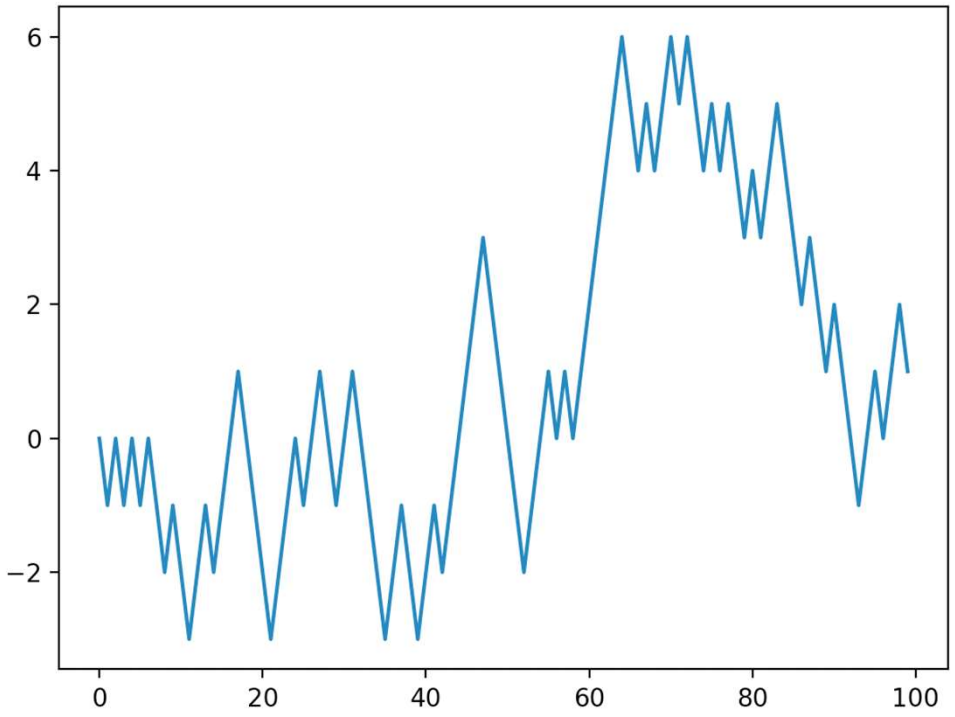
seed	난수 생성을 위한 시드 지정
permutation	순열 반환
shuffle	리스트나 배열의 순서 섞음
rand	균등 분포에서 표본 추출
randint	주어진 범위 내에서 임의의 난수 추출
randn	표준편차가 1이고, 평균이 0인 정규분포에서 표본 추출
binomial	이항분포에서 표본 추출
normal	가우시안 정규분포에서 표본 추출
beta	베타분포에서 표본 추출
chisqare	카이베곱분포에서 표본 추출
gamma	감마분포에서 표본 추출
uniform	균등[0, 1)분포에서 표본 추출

- **Random Points**
  - 0 에서 시작하여 1,000회 반복
  - 0 ~ 1 사이의 난수를 생성하여 0보다 크면 1, 그렇지 않으면 -1 로 포인트를 더함

```
import random
position = 0
walk = [position]
steps = 1000

for i in range(steps):
        if random.randint(0,1):
                step = 1
        else:
                step = -1
        position += step
        walk.append(position)

plt.plot(walk[:100])
plt.show()
```



- **Random Points**
  - 포인트에 대한 누적합을 배열식으로 표현하기

```
# use numpy
nsteps = 1000
draws = np.random.randint(0, 2, size=nsteps)
steps = np.where(draws > 0, 1, -1)
walk = steps.cumsum()

plt.plot(walk[:100])
plt.show()

# statistics
ret = walk.min()
print(ret)
ret = walk.max()
print(ret)
val = np.abs(walk) >= 10
ret = val.argmax()
print(ret)
```