CHAPTER

04

NumPy 라이브러리

NumPy Library

컴퓨터소프트웨어공학과
김대영

개 요

- NumPy 배열
- NumPy 유니버셜 함수

- **NumPy**
  - ndarray: 배열 중심의 빠른 산술연산을 지원하는 다차원 배열 지원
  - 반복되는 루프를 가지지 않고 전체 배열에서의 빠른 연산을 지원하는 수학 함수 지원
  - 디스크에 배열 데이터를 읽고 쓰기 위한 도구 지원
  - 선형대수, 랜덤 넘버 생성, 푸리에 연산 등 고급 수학 연산 지원
  - C, C++로 작성된 라이브러리를 C API를 사용하여 NumPy와 연동

- **데이터 분석을 위한 NumPy**
  - 빠른 벡터화된 배열 연산 제공
    - Data merging, clearning, subsetting & filtering, transformation, etc.
  - 일반적인 배열 알고리즘 지원
    - Sorting, unique, and set operation
  - 효율적인 데이터 통계 및 결합/요약 기능 제공
  - 반복(루프)를 사용하지 않고 배열의 조건부 제어 기능 제공
  - 그룹 단위의 데이터 처리 지원
    - Aggregation, transformation, function application, etc.

- **ndarray** (N-dimensional array)

```
import numpy as np # Generate some random data
data = np.random.randn(2, 3)
print(data)
> [[-0.2047, 0.4789, -0.5194],
   [-0.5557, 1.9658, 1.3934]]
print(data * 10)
> [[-2.047, 4.789, -5.194],
   [-5.557, 19.658, 13.934]]
print(data + data)
> [[-0.4094, 0.9579, -1.0389],
   [-1.1115, 3.9316, 2.7868]]
```

  - 모든 배열 요소는 같은 자료 구조를 가짐
  - shape: 배열 차원의 크기 ➔ data.shape → (2, 3)
  - dtype: 자료형/구조 ➔ data.dtype → float64

- **ndarray 생성**
  - ndarray 배열을 생성하는 가장 쉬운 방법은 배열 함수의 사용

```
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
print(arr1)
> [6. 7.5 8. 0. 1.]
```

  - 다차원 배열

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)
print(arr2)
> [[1 2 3 4]
   [5 6 7 8]]
```

    - ndim: 배열의 차원을 리턴 → attr2.ndim → 2
    - attr2.shape → (2, 4)

• ndarray 생성

- 초기화

```
arr3 = np.zeros((2,3))
print(arr3)
> [[0. 0. 0.]
   [0. 0. 0.]]
arr4 = np.ones((2,3))
print(arr4)
> [[1. 1. 1.]
   [1. 1. 1.]]
```

- empty: 초기화 없이 배열 생성

```
arr5 = np.empty((2,3))
print(arr5)
> [[0.53380283 0.2359652  0.10965788]
   [0.04085236 0.59465308 0.47377718]]
```

- **ndarray 생성**
  - arange :
    - 리스트 대신 배열을 사용하여 값을 리턴
    - 주어진 인자에 대하여 정렬된 배열 요소로 리턴

```
arr6 = np.arange(5)
print(arr6)
> [0 1 2 3 4]
```

- **산술 연산**(Arithmetic calculation)
  - NumPy 배열은 모든 데이터를 행렬(matrix) 연산으로 처리
  - 루프 연산을 사용하지 않음 (for 구문을 사용하지 않음)

```
arr = np.array([[1. 2. 3.] [4. 5. 6.]])
print(arr)
> [[1. 2. 3.]
   [4. 5. 6.]]
arr2 = arr * arr
print(arr2)
> [[ 1.  4.  9.]
   [16. 25. 36.]]
arr2 = 1 / arr
> [[1.         0.5        0.33333333]
   [0.25       0.2        0.16666667]]
arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])
val = arr2 > arr
print(val)
> [[False  True False]
   [True False  True]]
```

- **인덱싱**(indexing)과 **슬라이싱**(slicing)
  - 배열 데이터의 부분 집합 또는 부분 요소를 다루는 방법

```
arr = np.arange(10)
print(arr)
> [0 1 2 3 4 5 6 7 8 9]

print(arr[5]) # indexing
> 5

print(arr[5:8])# slicing
> [5 6 7]

arr[5:8] = 12
print(arr)
> 0  1  2  3  4 12 12 12  8  9]

arr[:] = 10
print(arr)
> [10 10 10 10 10 10 10 10 10 10]
```

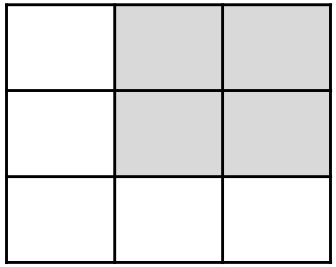
- **인덱싱**(indexing)과 **슬라이싱**(slicing)
  - 다차원 배열 다루기

```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr2d[2])
> [7 8 9]
print(arr2d[0][2])
> 3
print(arr2d[0, 2])
> 3
arr3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr3d)
> [[[ 1  2  3]
    [ 4  5  6]]
   [[ 7  8  9]
    [10 11 12]]]

print(arr3d[0])
> [[ 1  2  3]
   [ 4  5  6]]
```

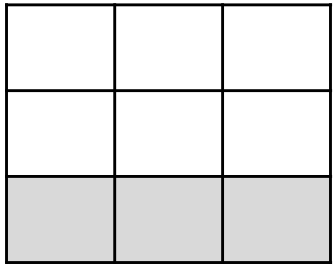
- **인덱싱**(indexing)과 **슬라이싱**(slicing)
  - 다차원 배열 다루기

```
old = arr3d[0].copy()   # need to explicitly copy
print(old)
> [[1 2 3]
   [4 5 6]]

print(arr3d[1, 0])
> [7 8 9]
```

- **인덱싱**(indexing)과 **슬라이싱**(slicing)
  - 다차원 배열에서의 슬라이싱

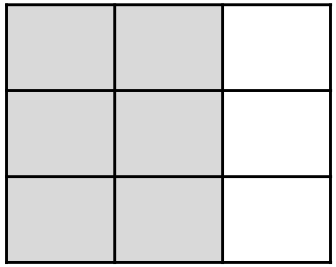


Array[:2, 1:] (2, 2)

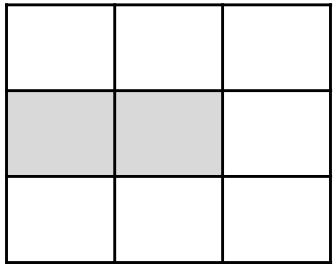


Array[2] (3,)
Array[2, :] (3,)
Array[2:, :] (1, 3)

- **인덱싱**(indexing)과 **슬라이싱**(slicing)
  - 다차원 배열에서의 슬라이싱



Array[:, :2] (3, 2)



Array[1, :2] (2,)
Array[1:2, :2] (1, 2)

- **불린 인덱싱**(Boolean indexing)

```
names = np.array([‘Bob’, ‘Joe’, ‘Will’, ‘Bob’, ‘Will’, ‘Joe’, ‘Joe’])
data = np.random.randn(7, 4)    #generate normal distribution
print(names)
print(data)
> ['Bob' 'Joe' 'Will' 'Bob' 'Will' 'Joe' 'Joe’]
  [[ 1.28408865  0.30948768 -0.26844903  1.92379021]
   [-1.71017394 -0.86549954  0.69822624 -0.26146633]
   [ 0.6519768   1.25469167 -0.58159377 -0.32087822]
   [-0.55638558 -2.30579501  0.16068881 -1.20631817]
   [-0.9625077   0.43034502  0.33365622  1.39249142]
   [ 0.33815464 -0.15305047  0.11825156  0.03494903]
   [ 0.29543837  0.60930417  1.9496912   0.66342351]]

print(names == “Bob”)   # Comparison operation
> [ True False False  True False False False]    # return Boolean array

print(data[names == “Bob”])
> [[ 1.28408865  0.30948768 -0.26844903  1.92379021]
   [-0.55638558 -2.30579501  0.16068881 -1.20631817]]
```

- **불린 인덱싱**(Boolean indexing)

```
mask = (names == “Bob”) | (names == “Will”)
print(mask)
> [ True False  True  True  True False False]
print(data[mask])
>  [[ 1.28408865  0.30948768 -0.26844903  1.92379021]
   [ 0.6519768   1.25469167 -0.58159377 -0.32087822]
   [-0.55638558 -2.30579501  0.16068881 -1.20631817]
   [-0.9625077   0.43034502  0.33365622  1.39249142]]
```

- **팬시 인덱싱**(Fancy indexing)
  - 정수형 배열을 사용하여 인덱스를 나타냄

```
array = np.empty((8, 4))
for i in range(8):
        array[i] = i
print(array)
> [[0. 0. 0. 0.]
   [1. 1. 1. 1.]
   [2. 2. 2. 2.]
   [3. 3. 3. 3.]
   [4. 4. 4. 4.]
   [5. 5. 5. 5.]
   [6. 6. 6. 6.]
   [7. 7. 7. 7.]]
print(array[[4,3,0,6]])
> [[4. 4. 4. 4.]
   [3. 3. 3. 3.]
   [0. 0. 0. 0.]
   [6. 6. 6. 6.]]
```

```
 array = np.arange(32).reshape((8, 4
))
 > [[ 0  1  2  3]
    [ 4  5  6  7]
    [ 8  9 10 11]
    [12 13 14 15]
    [16 17 18 19]
    [20 21 22 23]
    [24 25 26 27]
    [28 29 30 31]]

 # (1,0), (5, 3), (7, 1), (2, 2)
 print(array[[1,5,7,2],[0,3,1,2]])
 > [ 4 23 29 10]
```

- **Transpose**
  - 데이터의 복사를 사용하지 않고 데이터의 모습(shape)을 변환

```
arr = np.arange(15).reshape((3, 5))
print(arr)
> [[ 0  1  2  3  4]
   [ 5  6  7  8  9]
   [10 11 12 13 14]]

print(arr.T)
> [[ 0  5 10]
   [ 1  6 11]
   [ 2  7 12]
   [ 3  8 13]
   [ 4  9 14]]
```

```
arr = np.random.randn(6, 3)
print(arr)
> [[-1.42524468 -0.29374192  1.21137079]
   [ 0.191385   -0.73704859 -0.48644958]
   [ 0.96950639  0.51076365 -0.79444846]
   [ 0.6582691   0.22102956 -0.12977945]
   [-0.60538587 -1.5828205  -0.49924476]
   [ 0.51346503 -0.07847534 -0.06590076]]

val = np.dot(arr.T, arr)
print(val)
> [[ 9.97853723  2.16853372 -2.92548706]
   [ 2.16853372  2.15373915 -0.24251436]
   [-2.92548706 -0.24251436 10.42156009]]
```

유니버셜 함수(Universal funtion: ufunc)

- **유니버셜 함수**(Universal function)
  - NumPy배열에서 배열 요소에 대한 연산을 수행하는 내장 함수
  - 빠른 벡터화된 래퍼제공

```
array = np.arange(10)
print(array)
> [0 1 2 3 4 5 6 7 8 9]

print(np.sqrt(array))
> [0.         1.         1.41421356 1.73205081 2.         2.23606798
2.44948974 2.64575131 2.82842712 3.        ]

print(np.exp(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]
```

유니버셜 함수(Universal funtion: ufunc)

- **유니버셜 함수**(Universal function)

```
 x = np.random.randn(8)
 y = np.random.randn(8)
 print(x)
 > [-0.71194614  1.13253163 -0.66416687 -1.37516265 -0.19726776
-0.43890344 -0.47503553 -1.13566608]
 print(y)
 > [-2.03348403 -0.17120473  0.91755927  1.08231464 -0.02494011  0.15142
868 0.65399922 -0.73413958]

 val = np.maximum(x, y)
 print(val)
 > [-0.71194614  1.13253163  0.91755927  1.08231464 -0.02494011  0.15142
868 0.65399922 -0.73413958]
```

유니버셜 함수(Universal funtion: ufunc)

- **유니버셜 함수**(Universal function)

```
 array = np.random.randn(7) * 5
 print(array)
 > [0.28901038  3.82055857 -5.92584602  2.48201981  0.01938164  3.232281
11 -8.90810665]

 remainder, whole_part = np.modf(arr)

 print(remainder)
 > [0.28901038  0.82055857 -0.92584602  0.48201981  0.01938164  0.232281
11 -0.90810665]

 print(whole_part)
 > [0.  3. -5.  2.  0.  3. -8.]
```

유니버셜 함수(Universal funtion: ufunc)

- **단항 함수**

abs, fabs	각 원소들에 대한 절대값 계산
sqrt	각 원소의 제곱근 계산 (array ** 0.5 와 동일)
square	각 원소의 제곱 계산 (array ** 2 와 동일)
exp	각 원소의 지수 e^x 계산
log, log10, log2, log1p	자연로그, 로그10, 로그2, 로그(1+x) 계산
sign	각 원소의 부호 계산: 1(양수), 0, -1(음수)
ceil	각 원소에 대하여 ceiling 계산 (소수점 값을 올림)
floor	각 원소에 대하여 floor 계산 (소수점 값을 내림)

유니버셜 함수(Universal funtion: ufunc)

- **단항 함수**

rint	각 원소의 소수자리 반올림
modf	각 원소의 몫과 나머지를 배열로 반환
isnan	각 원소가 숫자가 이닌지 확인
isfinite, isinf	각 원소가 유한한지, 무한한지 확인
cos, cosh, sin, sinh, tan, tanh	삼각함수 계산
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	역삼각함수 계산
logical_not	논리 Not 계산

유니버셜 함수(Universal funtion: ufunc)

- **이항 함수**

add	두 배열에서 각각의 원소를 더한다
subtract	배열 A에서 배열 B의 원소를 뺀다
Multiply	배열 원소끼리 곱한다
divide, floor_divide	배열 A의 원소에서 배열 B의 원소로 나눈다 floor_divide: 몫만 리턴
power	배열 A의 원소를 배열B의 원소만큼 제곱한다 (a^b)
maximum, fmax	각 배열의 두 원소 중 큰 값을 반환한다
minimum, fmin	각 배열의 두 원소 중 작은 값을 반환한다

유니버셜 함수(Universal funtion: ufunc)

- **이항 함수**

mod	배열 A의 원소를 배열 B의 원소로 나눈 나머지
copysign	배열 A의 원소 기호를 배열 B의 원소 기호로 변경
greater, greater_equal, less, less_equal, equal, not_equal	비교 연산
logical_and, logical_or, logical_xor	논리 연산