
파이썬 기반
빅데이터 처리
및 분석기술
Crayon

이번 시간에는

2 Numpy 심화

- 유니버설함수
- 집계 함수
- 브로드캐스팅 연산
- 비교 연산자
- 마스크로서 불리언 배열
- 팬시 인덱싱
- 정렬

1. 유니버설 함수

단항(unary) 유니버설 함수

매개변수를
하나만 갖는 함수

이항(binary) 유니버설 함수

매개변수를
두 개 갖는 함수

1. 유니버설 함수

1) 단항 유니버설 함수

◆ 배열 산술 연산

```
x=np.arange(4)
print("x =", x)
print("x + 10 =", x + 10)
print("x - 10 =", x - 10)
print("x * 2 =", x * 2)
print("x / 2 =", x / 2)
print("x // 2 =", x // 2)
```

```
x =  [0 1 2 3]
x + 10 =  [10 11 12 13]
x - 10 =  [-10  -9  -8  -7]
x * 2 =  [0 2 4 6]
x / 2 =  [0.  0.5 1.  1.5]
x // 2 =  [0 0 1 1]
```

나머지를 버리고 몫만 산출

1. 유니버설 함수

1) 단항 유니버설 함수

◆ 음수 · 지수 · 나머지 연산

```
x=np.arange(4)
print("x = ", x)
print("-x = ", -x)
print("x ** 2 = ", x ** 2)
print("x % 2 = ", x % 2)
```

```
x =  [0 1 2 3]
-x =  [ 0 -1 -2 -3]
x ** 2 =  [0 1 4 9]
x % 2 =  [0 1 0 1]
```

1. 유니버설 함수

1) 단항 유니버설 함수

◆ 래퍼 함수

• 기존 함수를 한 번 감싸서 원래 동작에 약간의 처리를 추가하는 함수

연산자	래퍼 함수	설명
+	np.add	덧셈
-	np.subtract	뺄셈
-	np.negative	단항 음수
*	np.multiply	곱셈
/	np.divide	나눗셈
//	np.floor_divide	정수 나눗셈
**	np.power	지수 연산
%	np.mod	나머지 연산

1. 유니버설 함수

1) 단항 유니버설 함수

◆ 절대값 함수

- abs(absolute) 함수

```
x = np.array([-2, -1, 0, 1, 2])
np.absolute(x)
```

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

```
x = np.array([-2, -1, 0, 1, 2])
np.abs(x)
```

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

1. 유니버설 함수

1) 단항 유니버설 함수

◆ 절대값 함수

- 복소수 데이터 적용 → 절대값으로 복소수의 크기 반환

```
x = np.array([3 - 4j, 4 - 3j, 2 + 0j, 0 + 1j])
np.abs(x)
```

```
array([5., 5., 2., 1.])
```

1. 유니버설 함수

1) 단항 유니버설 함수

◆ 삼각 함수

• sin, cos, tan 함수 적용

```
theta = np.linspace(0, np.pi, 3)
print("theta = ", theta)
print("sin(theta) = ", np.sin(theta))
print("cos(theta) = ", np.cos(theta))
print("tan(theta) = ", np.tan(theta))
```

```
theta =  [0.         1.57079633 3.14159265]
sin(theta) =  [0.0000000e+00 1.0000000e+00 1.2246468e-16]
cos(theta) =  [ 1.000000e+00  6.123234e-17 -1.000000e+00]
tan(theta) =  [ 0.00000000e+00  1.63312394e+16 -1.22464680e-16]
```

1. 유니버설 함수

1) 단항 유니버설 함수

◆ 역삼각 함수

• arcsin, arccos, arctan 함수 적용

```
x = [-1, 0, 1]
print("x = ", x)
print("arcsin(theta) = ", np.arcsin(x))
print("arccos(theta) = ", np.arccos(x))
print("arctan(theta) = ", np.arctan(x))
```

```
x =  [-1, 0, 1]
arcsin(theta) =  [-1.57079633  0.          1.57079633]
arccos(theta) =  [3.14159265 1.57079633 0.        ]
arctan(theta) =  [-0.78539816  0.          0.78539816]
```

1. 유니버설 함수

1) 단항 유니버설 함수

◆ 지수 함수

- a를 1이 아닌 양의 상수, x를 모든 실숫값을 가지는 변수라고 할 때, $f(x)=a^x$ 로 주어지는 함수

```
x = [1, 2, 3]
print("x =", x)
print("e^x = ", np.exp(x))
print("2^x = ", np.exp2(x))
print("3^x = ", np.power(3,x))
```

```
x = [1, 2, 3]
e^x =  [ 2.71828183  7.3890561  20.08553692]
2^x =  [2. 4. 8.]
3^x =  [ 3  9 27]
```

1. 유니버설 함수

1) 단항 유니버설 함수

◆ 로그 함수

• 지수 함수 a의 역함수로 log X로 표기

```
x = [1, 2, 4, 10]
print("x =", x)
print("ln(x) =", np.log(x))
print("log2(x) =", np.log2(x))
print("log10(x) =", np.log10(x))
```

```
x = [1, 2, 4, 10]
ln(x) =  [0.         0.69314718 1.38629436 2.30258509]
log2(x) =  [0.         1.         2.         3.32192809]
log10(x) =  [0.         0.30103    0.60205999 1.        ]
```

1. 유니버설 함수

1) 단항 유니버설 함수

◆ out 인자

• 지정한 배열에 직접 연산 결과 표기 가능

```
x = np.arange(5)
y = np.empty(5)
np.multiply(x, 10, out=y)
print(y)
```

```
[ 0. 10. 20. 30. 40.]
```

1. 유니버설 함수

1) 단항 유니버설 함수

◆ out 인자

• 배열 슬라이싱 사용

```
x=np.arange(5)
y=np.zeros(10)
np.power(2, x, out=y[::2])
print(y)
```

```
[ 1.  0.  2.  0.  4.  0.  8.  0. 16.  0.]
```

1. 유니버설 함수

1) 단항 유니버설 함수

◆ out 인자

- 소규모 연산
  - 특정 주제에 대한 모델이 많음

- 대규모 연산
  - 임시 배열을 생략하여 연산 시간 절감

2. 집계 함수

1) reduce 함수

```
x=np.arange(1, 5)
np.add.reduce(x)
```

```
10
```

결과가 하나만 남을 때까지
해당 연산을 배열 요소에 반복적으로 적용

2. 집계 함수

2) sum, max, min 함수

◆ 합계(sum) 산출

```
s = np.random.random(1000)
np.sum(s)
```

```
493.48969490332115
```

2. 집계 함수

2) sum, max, min 함수

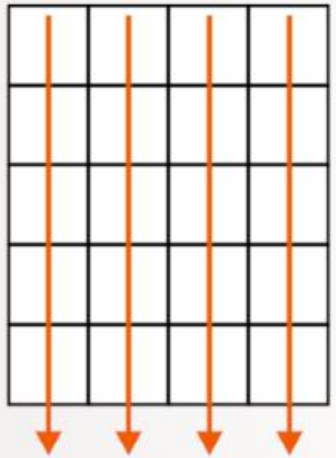
◆ 최솟값(min), 최댓값(max) 산출

```
s = np.random.random(1000)
np.min(s), np.max(s)
```

```
(0.0007634236589856291 0.9997803876745347)
```

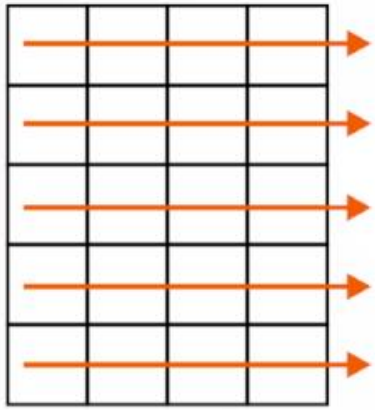
2. 집계 함수

3) axis 인자



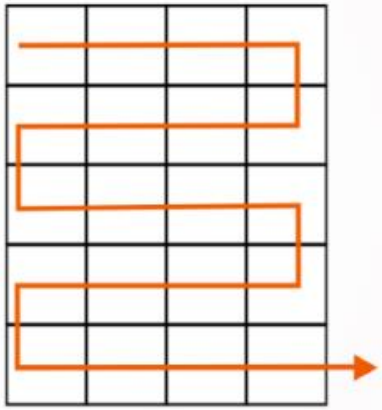
Axis = 0

열기준
집계 연산



Axis = 1

행기준
집계 연산



Axis = None

배열 내
모든 요소의
집계 연산

2. 집계 함수

3) axis 인자

```
m = np.random.random((3, 4))
print(m)
```

```
[[0.66092442 0.09969407 0.28901024 0.85744675]
 [0.66248372 0.684956   0.65281902 0.59031014]
 [0.7721359  0.1262104  0.82254776 0.43690469]]
```

```
m.sum()
```

```
6.655443100008886
```

Axis 값을 지정하지 않은 경우

```
m.max(axis=0)
```

```
array([0.7721359 , 0.684956  , 0.82254776, 0.85744675])
```

Axis 값을 0으로 지정한 경우

```
m.max(axis=1)
```

```
array([0.85744675, 0.684956  , 0.82254776])
```

Axis 값을 1로 지정한 경우

2. 집계 함수

함수명	설명
np.sum	요소의 합 계산
np.prod	요소의 곱 계산
np.mean	요소의 평균 계산
np.std	요소의 편차 계산
np.var	분산 계산
np.min	최솟값 계산
np.max	최댓값 계산
np.argmin	최솟값의 인덱스 찾기
np.argmax	최댓값의 인덱스 찾기
np.median	요소의 중앙값 계산
np.percentile	요소의 순위 기반 백분위 값 계산

3. 브로드캐스팅 연산

서로 다른 크기의 배열에
이항 유니버설 함수를 적용할 수 있는 기능

서로 다른 크기의 리스트끼리도
연산 수행이 가능

브로드캐스팅 연산

3. 브로드캐스팅 연산

1) 크기가 같은 경우

```
x=np.array([0, 1, 2])
y=np.array([3, 3, 3])
x+y
```

```
array([3, 4, 5])
```

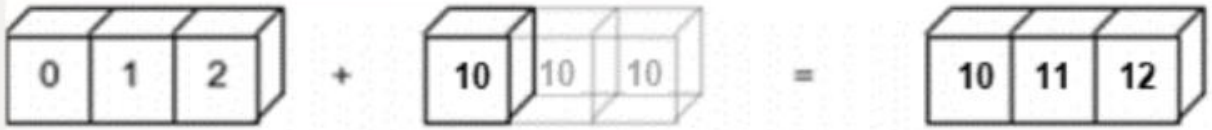
3. 브로드캐스팅 연산

2) 크기가 다른 경우

◆ 스칼라 값을 더해야 할 때

```
x=np.array([0, 1, 2])
x+10
```

```
array([10, 11, 12])
```



3. 브로드캐스팅 연산

3) 브로드캐스팅의 기능

배열을 더 높은 차원으로 확장함

2개 배열의 차원이 다르더라도 배열을 확장함

3. 브로드캐스팅 연산

3) 브로드캐스팅의 기능

◆ 배열 확장의 3가지 규칙

- 두 배열의 차원의 수가 서로 다를 경우,
  더 작은 차원을 가진 배열의 앞(왼쪽 부분)을 1로 채운다.
- 두 배열의 형상이 어떤 차원에서도 일치하지 않는 경우,
  해당 차원의 형상이 1인 배열이 늘어난다.
- 임의의 차원에서 크기가 일치하지 않고 1도 아닌 경우,
  오류가 발생한다.

3. 브로드캐스팅 연산

4) 1차원 배열 + 2차원 배열

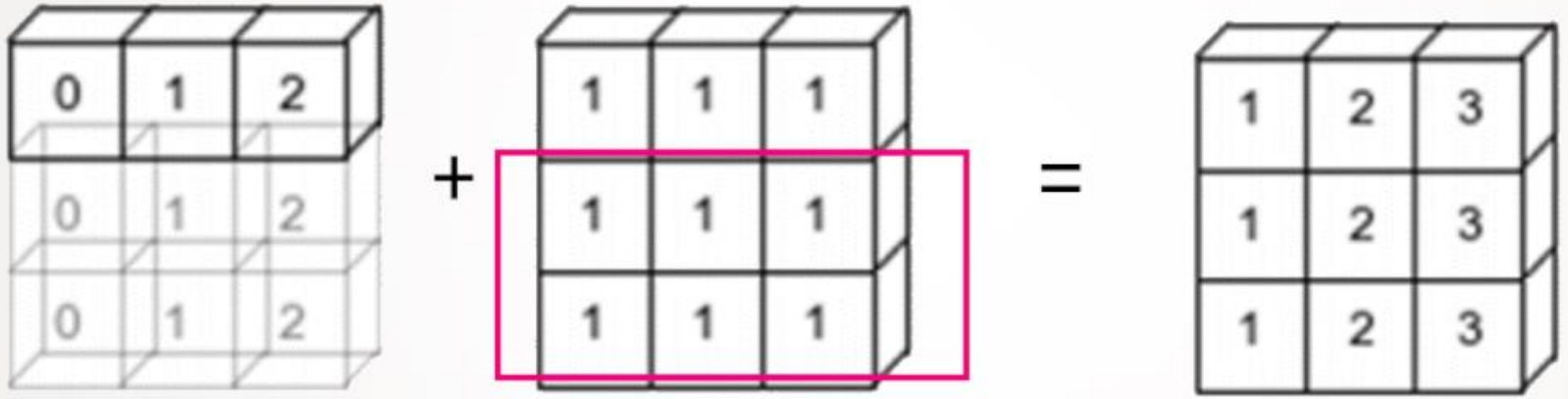
```
x = np.array([0, 1, 2])
M = np.ones((3, 3))
x + M
```

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

1차원 배열 x가
2차원 배열 M의 형상에 맞춤

3. 브로드캐스팅 연산

4) 1차원 배열 + 2차원 배열



3. 브로드캐스팅 연산

4) 두 배열 브로드캐스팅

```
a=np.arange(3).reshape((3, 1))
b=np.arange(3)
a+b
```

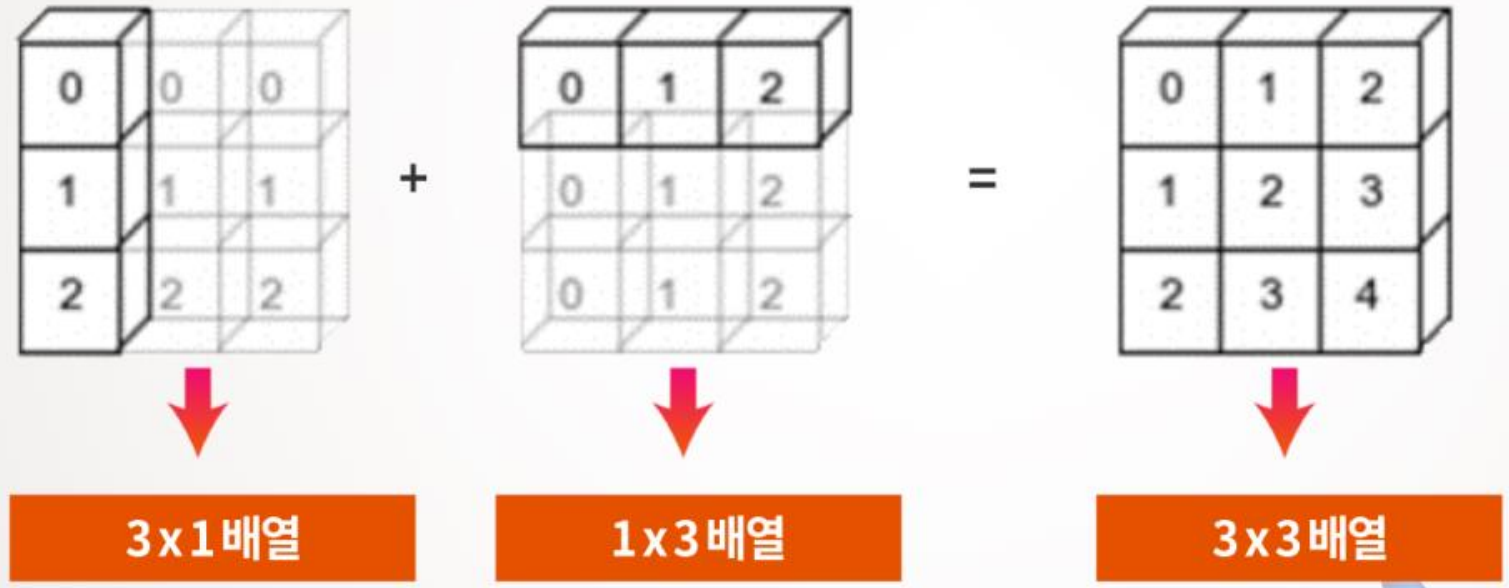
각 차원을 다른 배열의 크기에 일치하도록 늘림

덧셈 연산 수행

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

3. 브로드캐스팅 연산

4) 두 배열 브로드캐스팅



3. 브로드캐스팅 연산

5) 실제 사례

◆ 2차원 함수를 기반으로 이미지를 그릴 때

```
x=np.linspace(0, 5, 50)
y=np.linspace(0, 5, 50)[:, np.newaxis]
z=np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
```

- x는 1 x 50 배열
- y는 50 x 1 배열
- z는 50x50 2차원 배열

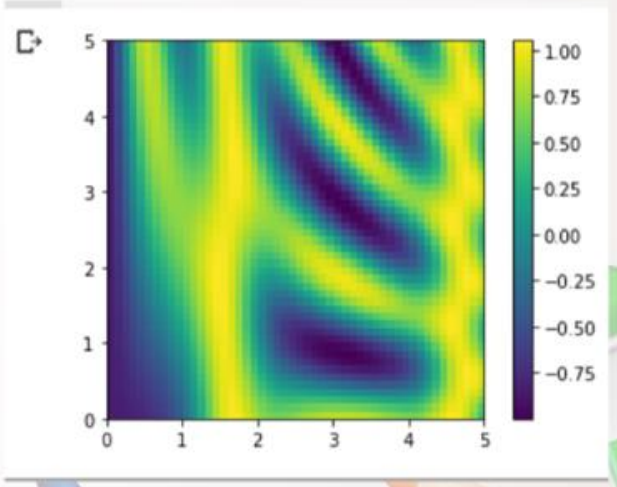
3. 브로드캐스팅 연산

5) 실제 사례

◆ 2차원 함수를 기반으로 이미지를 그릴 때

• 2차원 배열을 플로팅하기 위해 matplotlib 사용

```
import matplotlib.pyplot as plt
plt.imshow(z, origin='lower', extent=[0, 5, 0, 5], cmap='viridis')
plt.colorbar();
```



4. 비교 연산자

1) 행렬 데이터의 조건에 따른 결과 산출

◆ any 함수

• 하나라도 만족하면 true 산출

```
import numpy as np
x = np.arange(10)
np.any(x < 5), np.any(x < 0)
```

```
(True, False)
```

4. 비교 연산자

1) 행렬 데이터의 조건에 따른 결과 산출

◆ all 함수

• 모두 만족해야만 true 산출

```
import numpy as np
x = np.arange(10)
np.all(x > 5), np.all(x < 10)
```

```
(False, True)
```

4. 비교 연산자

2) 비교 연산자와 유니버설 함수

연산자	유니버설 함수
==	np.equal
!=	np.not_equal
<	np.less
<=	np.less_equal
>	np.greater
>=	np.greater_equal

4. 비교 연산자

3) 적용 사례

◆ 1차원 배열일 때

```
a=np.random.randint(1, 5, 5)
b=np.random.randint(1, 5, 5)
print(a, b)
a == b
```

```
[3 1 4 3 2] [3 2 4 1 2]
array([ True, False,  True, False,  True])
```

- 1~5 사이의 구간에서 5개 정수를 무작위로 발생시켜 1차원 배열 구성
- 배열의 각 요소가 서로 같은 지를 비교하고 이에 대한 불리언 값을 결과로 산출

4. 비교 연산자

3) 적용 사례

◆ 2차원 배열일 때

```
x=np.random.randint(10, size=(3, 4))
x
```

```
array([[3, 8, 8, 1],
       [5, 2, 8, 1],
       [4, 5, 2, 7]])
```

무작위 값을 갖는 2차원 배열 생성

```
x<5
```

```
array([[ True, False, False,  True],
       [False,  True, False,  True],
       [ True, False,  True, False]])
```

비교 연산 적용

6. 팬시 인덱싱

인덱스 배열을 전달하고, 인덱스 배열의 값을 인덱스로 사용

복잡한 배열의 하위 집합에 매우 빠르게 접근

**인덱스 배열의 값을 인덱스로 사용하여
배열의 요솟값에 접근하는 방법**

6. 팬시 인덱싱

1) 1차원 배열 적용 사례

```
a=np.arange(10)
b=np.array([3, 6, 9])
a[b]
```

```
array([3, 6, 9])
```

배열 b를 인덱스로 사용

6. 팬시 인덱싱

2) 다차원 배열 적용 사례

```
x=np.arange(12).reshape((3,4))
x
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
row=np.array([0, 1, 2])
col=np.array([2, 1, 3])
x[row, col]
```

```
array([ 2,  5, 11])
```

인덱스 값을 가진 2차원 배열을 전달

7. 정렬

1) sort 함수

◆ np.sort 함수

- 원래 배열 데이터는 그대로 유지한 채 정렬된 배열이 복사본으로 반환

```
import numpy as np
x = np.array([4, 1, 5, 2, 3])
print(x, np.sort(x))
```

```
[4 1 5 2 3] [1 2 3 4 5]
```

7. 정렬

1) sort 함수

◆ x.sort 함수

• 원래 배열 자체를 정렬

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

```
[1 2 3 4 5]
```

7. 정렬

1) sort 함수

◆ np.argsort 함수

- 정렬된 요소의 인덱스를 반환

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

```
[1 3 4 0 2]
```

7. 정렬

1) sort 함수

◆ x_reverse

• 내림차순 정렬

```
x=np.array([4, 1, 5, 2, 3])
x_reverse = np.sort(x)[::-1]
print(x_reverse)
```

```
[5 4 3 2 1]
```

7. 정렬

2) 다차원 배열 정렬

◆ axis 인자

- axis=0일때, 각 열을 기준으로 정렬

```
X = np.array([[2, 0, 6],
              [7, 3, 4],
              [1, 5, 3]])
np.sort(X,axis=0)
```

```
array([[1, 0, 3],
       [2, 3, 4],
       [7, 5, 6]])
```

7. 정렬

2) 다차원 배열 정렬

◆ axis 인자

- axis=1일때, 각 행을 기준으로 정렬

```
X = np.array([[2, 0, 6],
              [7, 3, 4],
              [1, 5, 3]])
np.sort(X,axis=1)
```

```
array([[0, 2, 6],
       [3, 4, 7],
       [1, 3, 5]])
```

이번 시간에는

Numpy 심화

1. 유니버설 함수
2. 집계 함수
3. 브로드캐스팅 연산
4. 비교 연산자
5. 마스크로서 불리언 배열
6. 팬시 인덱싱
7. 정렬

다음 시간에는

pandas 패키지

1 pandas 기초