Numpy advanced

Numpy advanced

- 산술연산
- 유니버셜 함수
- 기본 통계 메소드
- 기타 메소드
- 선형대수

산술연산

산술연산

- 기본
- 브로드캐스팅

산술연산

- 벡터화 : 배열은 for 문을 작성하지 않고 데이터를 일괄 처리 하는 것
  - 배열은 for 문을 작성하지 않고 데이터를 일괄 처리 하는 것
  - 같은 크기의 배열 간의 산술 연산은 배열의 각 원소 단위로 적용됨

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

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

```
[3]  arr + arr
```

```
array([[ 2,  4,  6],
       [ 8, 10, 12]])
```

```
[6]  arr / arr
```

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

산술연산

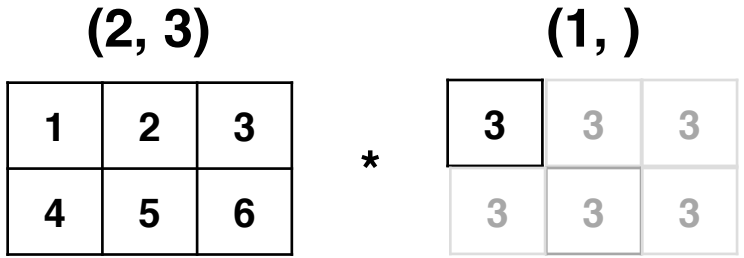
- 브로드캐스팅
  - 다른 모양의 배열 간의 산술 연산을 수행할 수 있도록 해주는 numpy의 기능
  - 브로드캐스팅이 가능하려면 연산을 수행하는 축을 제외한 나머지 축의 shape이 일치하거나 둘 중 하나의 길이가 1이여야 함

산술연산

- 브로드캐스팅

  1) 스칼라 인자 : 모든 원소에 각각 적용

```
[9]  10 - arr

     array([[9, 8, 7],
            [6, 5, 4]])

[10] arr*3

     array([[ 3,  6,  9],
            [12, 15, 18]])

[11] arr/3

     array([[0.33333333, 0.66666667, 1.
            [1.33333333, 1.66666667, 2.
```

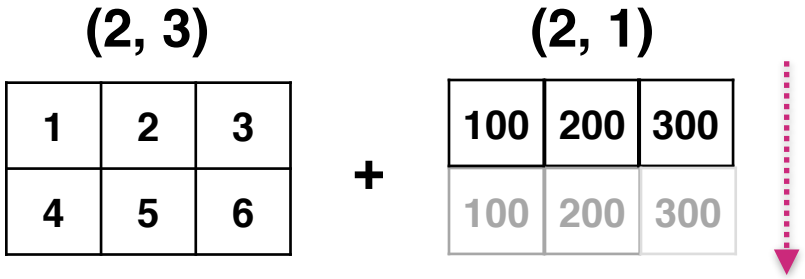


산술연산

- 브로드캐스팅

2) 배열

```
[15] arr

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

[36] arr2

     array([100, 200, 300])

[37] arr + arr2

     array([[101, 202, 303],
            [104, 205, 306]])
```



산술연산

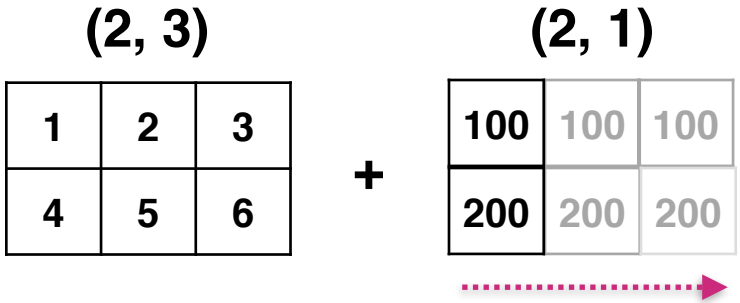
- 브로드캐스팅

2) 배열

```
[15] arr

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

```
[21] arr3

array([[100],
       [200]])
```

```
arr + arr3

array([[101, 102, 103],
       [204, 205, 206]])
```



산술연산

- 기본
- 브로드캐스팅

유니버셜 함수

유니버셜 함수

- 단항 유니버셜 함수
- 이항 유니버셜 함수

유니버셜 함수(ufunc)

- 유니버셜 함수란
  - ndarray 안에 있는 데이터 원소별로 연산을 수행하는 함수
  - 하나 이상의 스칼라 값을 받아서 하나 이상의 스칼라 값을 반환하는 간단한 함수를 고속으로 수행할 수 있는 벡터화 된 함수

유니버셜 함수(ufunc)

- 단항 유니버셜 함수

Function	설명
abs, fabs	각 원소의 절대값. 복소수가 아닌 경우 fabs를 쓰면 빠른 연산이 가능
sqrt	각 원소의 제곱근 계산(arr**0.5)
square	각 원소의 제곱을 계산(arr**2)
exp	각 원소에 지수e^x 계산
log, log10, log2, log1p	자연로그, 밑10인 로그, 밑2인 로그, log(1+x)
sign	각 원소의 부호(양수 1, 영 0, 음수 -1)
ceil	각 원소의 값보다 같거나 큰 정수 중 가장 작은 값
floor	각 원소의 값보다 같거나 작은 정수 중 가장 큰 값
rint	각 원소의 소수자리를 반올림
modf	각 원소의 몫과 나머지를 각각의 배열로 반환
isnan	각 원소가 NaN인지 아닌지. 불리언 배열로 반환
isfinite, isinf	각 원소가 유한한지, 무한한지. 불리언 배열로 반환
cos, cosh, sin, sinh, tan, tanh	일반 삼각함수, 쌍곡삼각함수
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	역삼각함수

유니버셜 함수(ufunc)

- 단항 유니버셜 함수

```
[2] arr = np.arange(-3, 3).reshape(3, -1)
    arr
```

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

```
[3] np.exp(arr)
```

```
array([[0.04978707, 0.13533528],
       [0.36787944, 1.        ],
       [2.71828183, 7.3890561 ]])
```

```
[4] np.floor(arr)
```

```
array([[-3., -2.],
       [-1.,  0.],
       [ 1.,  2.]])
```

```
np.isinf([np.inf, -np.inf, 1, np.nan])
```

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

유니버셜 함수(ufunc)

- 이항 유니버셜 함수

Function	설명
add	두 배열의 같은 위치의 원소끼리 합함
subtract	첫번째 배열의 원소에서 두번째 배열의 원소를 뺌
multiply	같은 위치의 원소끼리 곱함
divide, floor_divide	첫번째 배열의 원소를 두번째 배열의 원소를 나눔. floor는 몫만 취함
power	첫번째 매열의 원소를 두번째 배열의 원소만큼 제곱함
maximum, fmax	각 배열의 두 원소 중 큰 값을 반환. fmax는 NaN 무시
minimum, fmin	각 배열의 두 원소 중 작은 값을 반환. fmax는 NaN 무시
mod	첫번째 배열의 원소를 두번째 배열의 원소로 나눈 나머지
copysign	첫번째 배열의 원소의 기호를 두번째 배열의 원소의 기호로 바꿈
greater, greater_equal, less, less_equal, less, less_equal,	각 두 원소 간의 비교 연산(<, <=, >, >=, ==, !=)를 불리언 배열로 반환

유니버셜 함수(ufunc)

- 이항 유니버셜 함수

```
[6]  arr1 = np.arange(8).reshape(2, -1)
     arr2 = np.arange(-40, 40, 10).reshape(2, -1)
     print(arr1)
     print(arr2)
```

```
[[0 1 2 3]
 [4 5 6 7]]
[[-40 -30 -20 -10]
 [  0  10  20  30]]
```

```
[7]  np.maximum(arr1, arr2)
```

```
array([[ 0,  1,  2,  3],
       [ 4, 10, 20, 30]])
```

```
[9]  np.subtract(arr2, arr1)
```

```
array([[-40, -31, -22, -13],
       [ -4,   5,  14,  23]])
```

```
[10] np.multiply(arr1, arr2)
```

```
array([[  0, -30, -40, -30],
       [  0,  50, 120, 210]])
```

유니버셜 함수

- 단항 유니버셜 함수
- 이항 유니버셜 함수

기본 통계 메소드

기본 통계 메소드

- 기본 통계 메소드

기본 통계 메소드

- 배열 전체 혹은 배열에서 한 축을 따르는 자료에 대한 통계를 계산하는 수학 함수

sum
mean
std, var
min, max
argmin, argmax
cumsum
cumprod

```
[11] arr = np.random.randn(200, 500)
     arr.shape
```

```
(200, 500)
```

```
[12] arr.sum()    # ndarray의 메서드 이용
```

```
-596.3978294521196
```

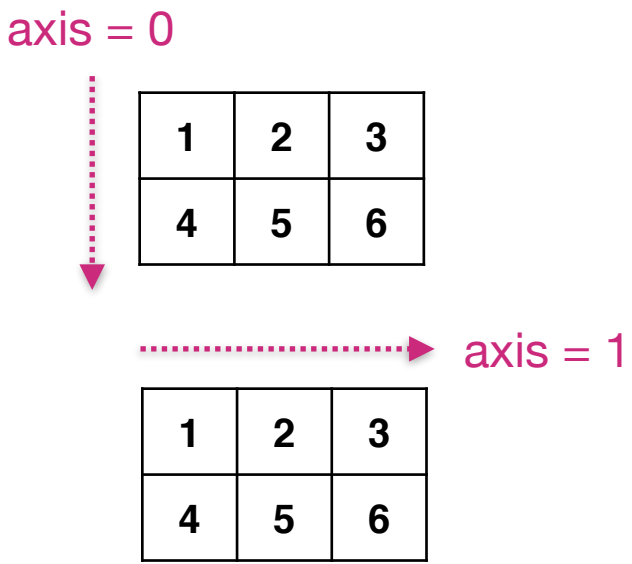
```
[13] np.sum(arr)     # numpy의 최상위 함수를 이용
```

```
-596.3978294521196
```

기본 통계 메소드

- sum, mean과 같은 함수의 경우 axis를 인자로 받아서 해당 axis에 대한 통계를 계산할 수 있음
- 축을 지정하지 않을 경우 배열 전체에 대한 값을 연산

```
[21] arr.shape

     (200, 500)

[14] arr.mean()

     -0.005963978294521196

[22] arr.mean(axis=0).shape

     (500,)

[23] arr.mean(axis=1).shape

     (200,)
```



기본 통계 메소드

- 기본 통계 메소드

연습문제 1

기타 메소드

기타 메소드

- any, all
- where
- sort

any, all

- any
  - 하나 이상의 값이 True이면 True를 반환
- all
  - 모든 값이 True이면 True를 반환

```
[2]  arr = np.array([True, False, True])
     arr.any()
```

```
True
```

```
[3]  arr = np.array([True, False, True])
     arr.all()
```

```
False
```

```
[4]  arr = np.array([0, 0, 3])
     arr.any()
```

```
True
```

```
[5]  arr = np.array([0, 0, 1])
     arr.all()
```

```
False
```

any, all

```
[6]  arr = np.array([1, 1, 1])
     arr.all()
```

```
True
```

```
arr = np.array([5, -1, np.inf])
arr.all()
```

```
True
```

where

- **x if 조건 else y**의 벡터화 버전
- numpy를 사용하여 큰 배열을 빠르게 처리 할 수 있으며, 다차원도 간결하게 표현이 가능

np.where(조건, x, y)

```
[8]  xarr = np.array([100, 200, 300, 400])
     yarr = np.array([1, 2, 3, 4])
     cond = np.array([True, False, True, False])
```

```
[9]  result = np.where(cond, xarr, yarr)
     result
```

```
array([100,   2, 300,   4])
```

→ True일때는 xarr에 있는 값을,
False일때는 yarr에 있는 값을 대입하여 반환

where

- x와 y자리에 들어가는 인자는 배열이 아니여도 가능

```
[8]  xarr = np.array([100, 200, 300, 400])
     yarr = np.array([1, 2, 3, 4])
     cond = np.array([True, False, True, False])
```

```
[10] np.where(xarr>200, max(xarr), 0)
```

```
array([  0,   0, 400, 400])
```

```
np.where(xarr%3==0, 1, 0 )
```

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

sort

- arr.sort( )
  - 주어진 축에 따라 정렬하며, 다양한 정렬방법들을 지원(참고)
  - arr자체를 정렬함(*in-place*)

arr.sort(axis=-1)

```
[23] np.random.seed(10)
     arr = np.random.randint(1, 100, size=10)
     arr
```

```
array([10, 16, 65, 29, 90, 94, 30,  9, 74,  1])
```

```
     arr.sort()
     arr
```

```
array([ 1,  9, 10, 16, 29, 30, 65, 74, 90, 94])
```

sort

- np.sort( )
  - np.sort는 배열을 직접 변경하지 않고 정렬된 결과를 가진 복사본을 반환(참고)

np.sort(arr, axis=-1)

```
[32] np.random.seed(20)
     arr = np.random.randint(1, 100, size=10)
     np.sort(arr)
```

```
array([10, 16, 21, 23, 29, 72, 76, 91, 91, 96])
```

```
[33] arr
```

```
array([91, 16, 96, 29, 91, 10, 21, 76, 23, 72])
```

```
-np.sort(-arr)
```

```
array([96, 91, 91, 76, 72, 29, 23, 21, 16, 10])
```

→ 부호를 이용하여 내림차순으로 정렬

기타 메소드

- any, all
- where
- sort

선형대수

선형대수

- dot
- matmul
- linalg

선형대수

- numpy는 행렬의 곱셈, 분할, 행렬식과 같은 선형대수에 관한 라이브러리를 제공함

Function	설명
dot	내적
dialog	정사각 행렬의 대각/비대각 원소를 1차원 배열로 반환하거나, 1차원 배열을 대각선 원소로 하고 나머지는 0으로 채운 단위행렬 반환
trace	행렬의 대각선 원소의 합을 계산
linalg.det	행렬식을 계산(ad-bc)
linalg.eig	정사각행렬의 고유값, 고유벡터를 계산
linalg.inv	정사각행렬의 역행렬을 계산
linalg.solve	A가 정사각 행렬일 때, Ax = b를 만족하는 x를 구함
linalg.svd	특이값 분해(SVD)를 계산

선형대수

- dot
  - dot product : 벡터의 내적

np.dot(a, b)

a.sort(b)

$$\mathbf{a} = (a_1, a_2, \ldots, a_n)$$

$$\mathbf{b} = (b_1, b_2, \ldots, b_n)$$

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

선형대수

- dot
  - dot product : 벡터의 내적

```
[36] x = np.random.randint(-5, 5, size=10)
     y = np.random.randint(-10, 10, size=10)
     x.shape, y.shape
```

```
((10,), (10,))
```

```
[37] x.dot(y)
```

```
-3
```

```
[38] np.dot(x, y)
```

```
-3
```

선형대수

- dot
  - dot product : 벡터의 내적

```
[39] x = np.array([[1, 2, 3], [4, 5, 6]])
     y = np.array([[2, -3], [1, 6], [-1, 2]])
     x.shape, y.shape
```

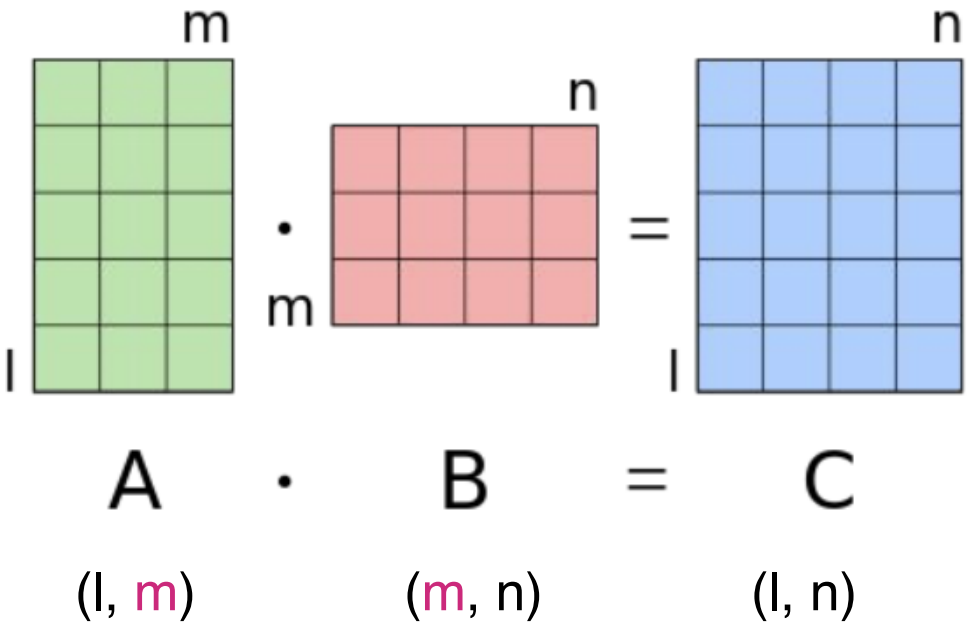
```
((2, 3), (3, 2))
```

```
np.dot(x, y)
```

```
array([[ 1, 15],
       [ 7, 30]])
```

선형대수

- matmul
  - matrix multiplication : 행렬의 곱



선형대수

- matmul
  - matrix multiplication : 행렬의 곱

np.matmul(a, b)

```
[41] a = np.random.randint(-3, 3, 10).reshape(2, 5)
     b = np.random.randint(0, 5, 15).reshape(5, 3)
     a.shape, b.shape
```

```
((2, 5), (5, 3))
```

```
[42] ab = np.matmul(a, b)
     print(ab.shape,'\n')
     print(ab)
```

```
(2, 3)

[[  6   0   6]
 [-14  -6  -4]]
```

선형대수

- matmul
  - matrix multiplication : 행렬의 곱

np.matmul(a, b)

```
[41] a = np.random.randint(-3, 3, 10).reshape(2, 5)
     b = np.random.randint(0, 5, 15).reshape(5, 3)
     a.shape, b.shape
```

```
((2, 5), (5, 3))
```

```
np.matmul(b, a)
```

→ matmul시에는 shape에 유의할 것

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-43-3c9936ad69e2> in <module>()
----> 1 np.matmul(b, a)  # matrix곱할때는 shape이 맞도록

ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0,
            with gufunc signature (n?,k),(k,m?)->(n?,m?) (size 2 is different from 3)
```

선형대수

- matmul
  - 3차원 이상의 경우에는 **마지막 2개 축**으로 이루어진 행렬을 다른 축들에 따라 쌓은 것으로 파악
  - 따라서, **마지막 2개의 차원이 행렬곱을 할 수 있으면 matmul가능**

```
[44] c = np.arange(24).reshape(2, 3, 4)
     d = np.arange(2*4*5).reshape(2, 4, 5)
     c.shape, d.shape
```

```
((2, 3, 4), (2, 4, 5))
```

```
[48] arr = np.matmul(c, d)
     arr.shape
```

```
(2, 3, 5)
```

선형대수

- dot
- matmul
- linalg

연습문제 2