

◦ Numpy ?

↳ 벡터 행렬 계산을 효율적으로 하기 위한 모듈.

↳ Numeric 모듈과 Numarray 모듈이 합쳐져

높은 수준의 다차원 배열 계산을 고속으로 효율적으로 할 수 있다.

◦ numpy 배열의 생성.

numpy.array(배열)

↳ numpy는 list와 비슷하지만

요소를 ',', '로 구분하지 않는다.

↳ list 형식으로된 배열을 numpy 형식으로 바꿀 수 있다.

◦ numpy 배열 복제

↳ numpy 배열은 list, tuple 등과 동일하게

복사하여 값을 변경할 경우, 원본의 값도 변경된다.

↳ [numpy 배열].copy()를 통하여 복제할 경우,

원본과 별개의 배열이 생성된다.

o numpy 배열 훑기

a = [1, 2, 3, 4, 5]

b = np.array(a)

c = np.array([1, 3, 5])

print(a[2]) → 3

print(c[-1]) → 5

print(c[0:2]) → [1 3]

o numpy 배열 계산

a = [1, 2, 3, 4, 5]

b = np.array(a)

c = np.array([1, 3, 5])

print(a * 2) → [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

print(b * 2) → [2 4 6 8 10]

print(c + 3) → [4 6 8]

list의 경우 '*' 사용시 해당 배열을 이어붙임

numpy의 경우 수학 기호를 사용한다면

각각의 원소로 계산하여 반환한다.

arr =
[i * 2 for i in
range(size)]

* numpy를 사용할 경우

list comprehension을

사용하지 않아도

원소의 값을 연산할 수 있다

o 1차원 배열

↳ numpy.array(배열, dtype=자료형)을 사용하여 생성.

[int
float
bool
:]

↳ dtype이 생략될 경우

데이터의 자료형을 유추하여 자동적으로 dtype을 할당.

o 2차원 배열

↳ numpy.array(2차원 배열, dtype=자료형)을 사용하여 생성.

o 배열 속성 반환

↳ 배열.ndim 또는 np.ndim(배열) → 배열의 차원

↳ 배열.shape 또는 np.shape(배열) → 배열의 형태

↳ 배열.dtype → 배열의 자료형.

o 모든 값이 1인 배열

↳ `numpy.ones(배열, dtype=자료형)`

을 사용하여 모든 원소 값이 1인 배열을 생성.

↳ `numpy.ones_like(배열, dtype=자료형)`

을 사용하여 배열의 크기와 동일하여

모든 원소의 값이 1인 배열을 생성할 수 있다.

o 모든 값이 0인 배열

↳ `numpy.zeros(배열, dtype=자료형)` 을 사용해

모든 원소 값이 0인 배열을 만들 수 있다.

↳ `numpy.zeros_like(배열, dtype=자료형)` 을 사용해

배열의 크기와 동일하여 모든 원소 값이 0인 배열을 생성.

o 모든 값이 비어 있는 행렬 \rightarrow tutorial doc가 훨씬 더 좋음.

↳ `numpy.empty(배열, dtype=자료형)` 을 사용하여

특정한 값으로 초기화하지 않는 배열을 생성.

↳ `numpy.empty_like(배열, dtype=자료형)` 을 사용하여

배열의 크기와 동일하여 특정한 값으로 초기화하지 않는 배열을 생성.

↳ 난수와 다른 일의의 값이 들어가며 같은 제도리에 저장된 내용에 따라 달라짐

* `empty` 를 이용하여 배열을 생성할 경우, 조금 더 빠른 속도로 생성 가능.

○ 대각의 값이 1인 배열 (단위 행렬)

↳ numpy.identity (N , dtype=자료형) 을 사용하여

$N \times N$ 크기의 단위 행렬을 반환

↳ numpy.eye (N, M , k=0, dtype=자료형) 을 사용하여

$N \times M$ 크기의 k값 만큼 이진진 단위 행렬을 반환

[양수일 경우 : 우상 방향으로
음수일 경우 : 좌하 방향으로]

○ 등간격

이상 ↑ 미만 ↓

↳ numpy.arange (start, end, step=1, dtype=자료형) 을 사용하여

start ~ (end-1) 사이의 값을 간격만큼 띄워 배열로 반환

↳ start 값은 포함되며, end 값은 포함되지 않을 것이다.

간격이 end 값을 초과할 경우, start 값만 포함된다.

e.g) $a = np.arange(0, 1, step=2)$

→ [0]

6 등간격

`↳ num.linspace (start, end, num=7개, endpoint=True, retstep=False, dtype='float64')`

$\frac{0}{2}$ 사용하여 $start \sim end$ 사이의 간격을 개수만큼 생성하여 배열로 반환.

- True : end의 값이 마지막 값으로 사용
- False : end의 값을 마지막 값으로 사용X

- True : 간격을 배열에 포함
 - False : 간격을 배열에 미포함.
- e.g)

`(array([0., 2., 4., 6., 8.], 2.0))`

간격.

0 등간격.

$\frac{1}{2}x^2$ $11x^2$ $6x^2$

값을 개수 만큼 생성하여 배열로 반환

- True : end 의 값이 마지막 값
- False : 마지막 값 사용 X

◦ 슬라이싱.

↳ 배열 $[a:b:c]$ 를 이용하여 배열의 일부를 잘라 표시할 수 있다.

↳ index 는 $0 \sim |en| - 1$ 까지 존재하며,

아무것도 입력하지 않고 $::$ 을 사용한 경우

모든 행 또는 열을 의미한다.

↳ ~ -1 은 입력한 경우, 마지막 index-1 ($|en| - 2$) 을 의미.

↳ 배열 $[a:b:e, c:d:f]$ 를 이용하여 배열의 일부를 잘라 표시할 수 있다.

↳ 배열의 슬라이싱에서 간격만 입력하여 배열은 출력할 수 있다.

e.g) $[\sim 2, \sim 2] \rightarrow$ 행과 열은 각 2칸씩

되며 출력된다.

◦ reshape

↳ 배열. shape = (행, 열) 을 통하여 배열의 형태를 변환할 수 있다.

↳ 행렬의 같은 흐름의 길이의 길(개수) 를 같아야 한다.

↳ 3차원 이상의 배열 또한 형태를 변환할 수 있다.

* 흐름의 개수와 동일 해야 한다.

배열. shape = (페이지, 행, 열)

o 기본연산

↳ 사칙연산과 관련된 연산은 array와 array 사이에

연산기호 (+, -, *, /)를 포함하여 계산할 수 있다.

↳ 곱하기와 나누기를 우선적으로 연산한다.

↳ 배열의 차원구조가 동일하지 않아도

블로드 캐스팅 조건에 포함된다면 연산이 가능.

↳ 모양이 다른 배열들 간의 연산이

어떤 조건을 만족해도 때 가능해지도록.

배열을 자동적으로 변환하는 것

* 조건

1) 두 배열 중 최소한 하나의 배열이 /차원이면

2) 차원에 대해 축의 길이가 동일하면

◦ 배열 연산

↳ np. cross (a, b) → 벡터곱

↳ np. dot (a, b) → 내적

◦ 심화 연산

↳ Math 라이브러리에서 사용할 수 있는

수학 함수들이 있다.

↳ 모든 배열에 일괄로 적용할 수 있으며

특정 함수는 자료형 (dtype) 이 인지해야 사용할 수 있다.

◦ 매트릭스

↳ 행렬의 곱을 반전해야하는 경우

mat 형식으로 변환 후 곱연산 실행

e.g)

$ma = np.mat(a)$

$mb = np.mat(b)$

`print(ma * mb)`

◦ 매트릭스 클래스

↳ $mat(\text{배열})$, T의 경우, 매트릭스의 전치 값을 반환

↳ $mat(\text{배열})$, H의 경우, 매트릭스의 공액부수의 전치 값을 반환

↳ $mat(\text{배열})$, I의 경우, 매트릭스의 곱의 역함수 값을 반환

↳ $mat(\text{배열})$, A의 경우, 매트릭스 형식을 다시 array 형식으로 변환

o 차원 확장 및 축소 - 차원 추가

↳ index \vdash `np.newaxis` \vdash 이용해 차원을 확장할 수 있다.

e.g) `arr = np.array([1, 2, 3, 4])`

`print(arr[np.newaxis])`

↳ 열부분에 입력시

차원을 한 단계 추가

> `[[1, 2, 3, 4]]`

`print(arr[:, np.newaxis])`

↳ 열부분에 입력시

차원은 분해한 후 한 단계 추가

> `[[],
[],
[],
[]]`

↳ 배열 `[:, :, :, ..., np.newaxis]` 를 이용하여 차원을 확장할 수 있다.

차원이 증가할 때 따라 index의 표시법이 같이 증가

주로, 슬라이싱을 통한 연산에 사용된다.

◦ 차원 확장 및 축소

↳ np.expand_dims(배열, 축)을 통해 지정된

축의 차원을 확장할 수 있다.

↳ np.squeeze(배열, 축)을 통해 지정된

축의 차원을 축소할 수 있다.

* 만약, 차원 축소 함수에 축을 입력하지
않으면 1차원 배열로 축소한다.

↳ 만약, 일의의 차원으로 변경하는 경우,

reshape을 통해 예시적으로 차원의 형태를 변경할 수 있다.

↳ 동일 차원을 사용하더라도 reshape 함수의 모수값을 변경할 수 있다.

↳ reshape 함수는 브로드캐스팅의 조건을 만족해야 쓸 수 있다.

◦ 난수 - 무작위 선택.

↳ `numpy.random.seed(n)`을 이용하여 임의의 시드를 생성할 수 있다.

시드값에 따라 난수와 흡사하지만 항상 같은 결과를 반복.

↳ `numpy.random.choice(배열, n, replace=True, p=None)`

을 이용해 배열에서 n개의 값을 선택하여 반복할 수 있다.

[True : 값 중복 허용
False : 값 중복 X]

각 데이터가 선택될 확률.

1) p값들의 합은 항상 1 /

2) replace를 False로

사용할 경우,

값이 중복되지 않기 때문에

n개 이상 0의 값과 달라야 한다.

◦ 난수 발생.

↳ numpy.random.rand(n, m, ...) 을 이용하여

다차원 무작위 배열을 생성할 수 있다.

↳ numpy.random.random(n, m) 을 이용하여

표준 정규분포에서 무작위 배열을 생성할 수 있다.

↳ numpy.random.randint(low, high, (n, m), dtype=None)

을 이용하여 low ~ (high-1) 사이의 무작위 (n, m) 크기의

정수 배열을 반환.

↳ numpy.random.random((n, m)) 라

numpy.random.sample((n, m)) 를 이용하여

0.0 ~ 1.0 사이의 무작위 (n, m) 크기 배열을 반환한다.

◦ 난수 발생.

↳ numpy.random.uniform (low, high, size) 를 이용하여

low ~ high 사이의 균일한 분포의 무작위 배열을 반환.

↳ numpy.random.lognormal (mean, sigma, size) 를 이용하여

평균과 시그마를 대입한 로그 정규 분포의 무작위 배열을 반환한다.

↳ numpy.random.laplace (loc, scale, size)

μ 와 λ 를 대입한 라플라스 무작위 배열을 반환.

◦ 병합

↳ numpy.hstack([배열1, 배열2])를 이용하여

배열1 우측에 배열2를 이어붙일 수 있다.

↳ numpy.vstack([배열1, 배열2])를 이용하여

배열1 하단에 배열2를 이어붙일 수 있다.

↳ numpy.stack([배열1, 배열2], axis=축)를 이용하여

지정한 축으로 배열1과 배열2를 이어붙일 수 있다.

* 축은 이어붙인 차원의 범위를 넘어갈 수 없다.

↳ numpy.dstack([배열1, 배열2])를 이용하여

새로운 축으로 배열1과 배열2를 이어붙일 수 있다.

↳ numpy.c_[배열1, 배열2]를 이용하여

배열1 우측에 배열2를 이어붙일 수 있다.

↳ numpy.r_[배열1, 배열2]를 이용하여

배열1 하단에 배열2를 이어붙일 수 있다.

◦ 분할

↳ numpy.hsplit(배열, 개수 또는 구간)을 이용하여

배열을 수평 방향 또는 열(column) 단위로 분할

↳ numpy.vsplit(배열, 개수 또는 구간)을 이용하여

배열을 수직 방향 또는 행(row) 단위로 분할

↳ 축을 따라 입력한 개수만큼 분할을 실행하며,

구간을 입력할 경우 해당 구간마다 잘라 n+1개의 배열이 생성됨.

↳ numpy.split(배열, 개수 또는 구간, 축)을 이용하여

배열을 축 방향으로 분할한다.

↳ 지정된 축을 따라 입력한 개수만큼 분할을 실행하며

구간을 입력할 경우 해당 구간마다 잘라 n+1 개의 배열이 생성됨.

↳ np.dsplt(배열, 개수 또는 구간)을 이용하여

배열을 축을 따라 분할한다.

* 해당 함수는 3차원 이상의 배열에서만 동작.

① 브로드 캐스팅

↳ `Numpy` 배열에서 차원의 크기가 서로 다른 배열에서
산술 연산을 가능하게 하는 원리.

e.g) 두 배열 간 차원의 크기가 (4, 2), (2,) 일 때,

산술 연산을 실행한다면

(2,)의 배열이 (4, 2) 행렬의 각 행에 대해
요소별 연산을 실행할 수 있다.

↳ 두 배열 간의 차원의 크기가 달라도 차원의 크기가
더 큰 배열에 대해 작은 배열을 여러번 반복하지 않아도 되는 것을 의미.

※ 브로드 캐스팅 허용 규칙

1) 두 배열의 차원(`ndim`)이 같지 않다면

차원이 더 낮은 배열이 차원이 더 높은 배열과
같은 차원의 배열로 인식된다.

2) 반환된 배열의 연산은 연산을 수행한 배열 중

차원의 수(`ndim`)가 가장 큰 배열이 된다.

3) 연산에 사용된 배열과 반환된 배열의 차원의 크기(`shape`)가 같거나 1일 경우 브로드캐스팅이 가능.

4) 브로드 캐스팅이 적용된 배열의 차원 크기(`shape`)는 연산에 사용된 배열들의 차원 크기에 대한 최소 공배수 값으로 사용.

→ 자료형을 비교해 표현 범위가 더 넓은 자료형을 선택.

* 형식 캐스팅(type casting)을 지원해

두 배열의 자료형(`dtype`)이 다르더라도 연산이 가능하다.

◦ 메모리 레이아웃

↳ NumPy 배열의 요소값 정렬 방식.

행 우선 : 첫 번째 차원은 메모리에 넣은 다음
두 번째 차원은 메모리에 넣음.

열 우선 : 첫 번째 열은 메모리에 넣은 다음
두 번째 열은 메모리에 넣음.

행 우선

- 가장 빠르게 변화하는 쪽인 순서로 합침

- C 스타일의 메모리 순서

(0, 0)	(0, 1)	(0, 2)
(1, 0)	(1, 1)	(1, 2)
(2, 0)	(2, 1)	(2, 2)

...	(0, 0)	(0, 1)	(0, 2)	(1, 0)	(1, 1)	(1, 2)	(2, 0)	(2, 1)	(2, 2)	...
-----	--------	--------	--------	--------	--------	--------	--------	--------	--------	-----

열 우선

- 가장 느리게 변화하는 순서로 합침

- Fortran 언어 스타일

(0, 0)	(0, 1)	(0, 2)
(1, 0)	(1, 1)	(1, 2)
(2, 0)	(2, 1)	(2, 2)

...	(0, 0)	(1, 0)	(2, 0)	(0, 1)	(1, 1)	(2, 1)	(0, 2)	(1, 2)	(2, 2)	...
-----	--------	--------	--------	--------	--------	--------	--------	--------	--------	-----

◦ Numpy의 메모리 레이아웃

↳ Numpy 배열에서는 order 매개변수의 인수를 통해

메모리 레이아웃을 설정할 수 있다.

e.g) np.reshape(array, (2, -1), order = 'C')

- C : 레이아웃에 최대한 일치
- A : Fortran에 근접한 경우 'F'
아니면 'C'
- C : C언어 스타일
- F : Fortran 언어 스타일

- 이어붙이기

↳ np.append 는 차원이 같아야 이어붙일 수 있다.

e.g) np.append (배열 1, 배열 2, axis = ?)

- 삭제

↳ np.delete (배열, [시작. 끝], 축) 을 통해

배열을 제거할 수 있다.

o 조건 반환

↳ np.where 는 배열의 요소값이 특정 조건에

만족하는 값을 반환하는 함수

↳ 비교함수 (np.greater, np.greater_equal) or

특정 조건을 만족하는 배열은 반환할 수 있다

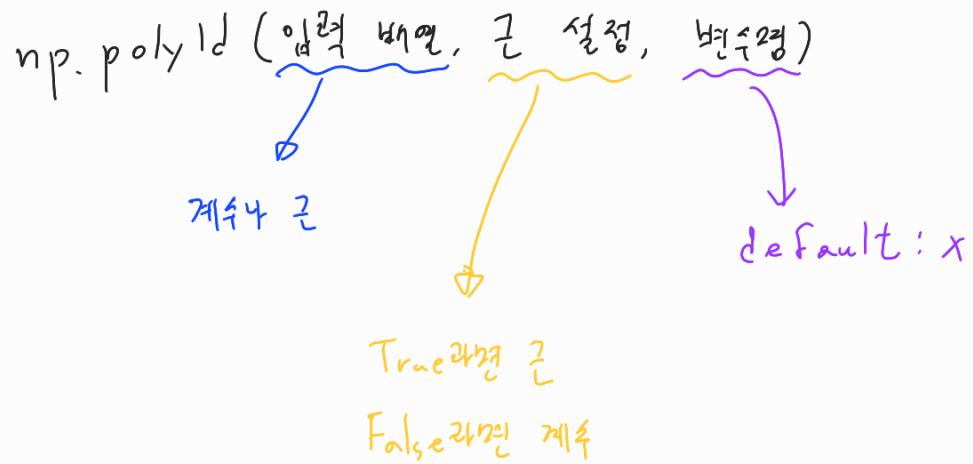
but, np.where() 은 더 세밀한 조건을 사용 가능

e.g) np.where(조건식) → 조건에 만족하는 배열의 색인값을
튜플로 반환

• r
np.where(조건식, 참 값, 거짓 값)

◦ 다항식 생성.(poly1d)

↳ np.poly1d 는 압축 배열을 험으로 간주하여 다항식을 생성.



e.g.)

`np.poly1d([1, 2], True, variable='a')`

$$> (a-1)(a-2)$$

$$= a^2 - 3a + 2$$

◦ 다항식의 속성

1) c, coefficients → 다항식의 계수

2) o, order → 다항식의 최고 차수

3) r, roots → 다항식의 근

4) variable → 다항식의 변수명.

◦ 다항식 근 계산 (roots)

np. roots 는 입력배열을 다항식으로 간주하여 근을 계산한다.

↳ np. roots (배열 혹은 계수 직접 입력)

◦ 다항식의 계수 계산 (poly)

↳ np. poly (배열 혹은 직접 입력)

◦ 다항식 계산 (polyval)

↳ np. polyval (입력 배열, 입력값)

는 입력 배열을 다항식으로 간주하고

입력값을 넣어 계산된 값을 반환.

↳ 숫자 or poly1d 값

◦ 다항식 사칙 연산

↳ list, Numpy, poly, poly1d

↳ np.poly ??? (입력 배열)

polyadd

polysub

polymul

polydiv → 반환 형식은 (list, ndarray)로 투플

◦ 다항식 적분 (Poly Integral)

↳ np.polyint (입력 배열, 반복 횟수, 상수)

로 다항식을 부정적분

적분 상수.

◦ 다항식 미분 (Poly Derivative)

↳ np.polyder (입력 배열, 반복 횟수)

입력 배열에 대해 반복 횟수 (~) 만큼

고계로 할 수을 칠해한다.