

LG 전자 DX (AI + BigData) 교류

1주차 – Numpy 활용



KNU KYUNGPOOK
NATIONAL UNIVERSITY



LG전자

소 속 : 경북대학교/인공지능학과

이 름 : 김현철 교수

E-Mail : hyunchul_kim@knu.ac.kr

Timetable

시간표		학습 내용
1차시	09:00 – 09:50	Numpy를 이용한 데이터 조작 Pandas를 이용한 데이터 조작 / 데이터 전처리
2차시	10:00 – 10:50	
3차시	11:00 – 11:50	
Lunch Break (11:50 – 13:30)		
5차시	13:30 - 14:20	실습
6차시	14:30 – 15:20	
7차시	15:30 – 16:20	
8차시	16:20 – 17:30	

NumPy 배열

Array

1. Packages

- <https://numpy.org/>



- <https://pandas.pydata.org/>

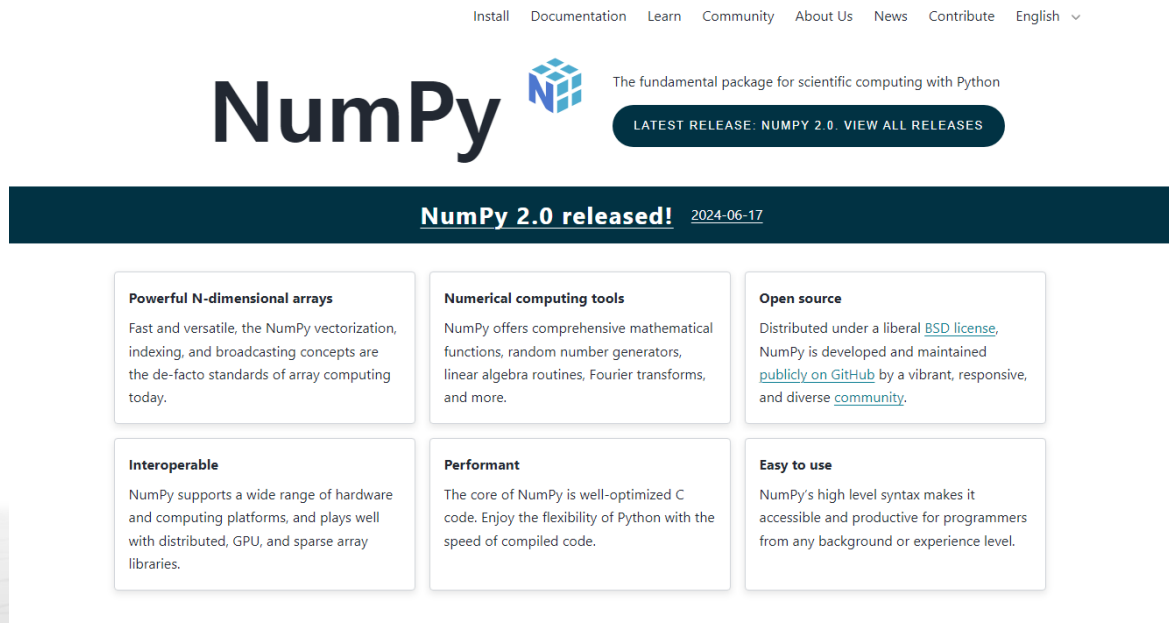
1. NumPy 소개

- Numerical Python

- 과학 계산용 패키지

→ numpy.org

- 고성능 N차원 배열 객체 사용



The image is a screenshot of the NumPy website. At the top, there is a navigation bar with links: Install, Documentation, Learn, Community, About Us, News, Contribute, and English. Below the navigation bar is the NumPy logo, which consists of the word "NumPy" in a large, bold, sans-serif font, followed by a small blue cube icon. To the right of the logo is the text "The fundamental package for scientific computing with Python". Below this text is a dark blue button with white text that says "LATEST RELEASE: NUMPY 2.0. VIEW ALL RELEASES". Below the navigation bar and logo is a dark blue banner with white text that says "NumPy 2.0 released!" followed by the date "2024-06-17". Below the banner is a grid of six white boxes, each with a title and a description. The boxes are arranged in two rows of three. The titles are: "Powerful N-dimensional arrays", "Numerical computing tools", "Open source", "Interoperable", "Performant", and "Easy to use". The descriptions provide details about each feature.

Install Documentation Learn Community About Us News Contribute English

NumPy

The fundamental package for scientific computing with Python

LATEST RELEASE: NUMPY 2.0. VIEW ALL RELEASES

NumPy 2.0 released! 2024-06-17

Powerful N-dimensional arrays

Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.

Numerical computing tools

NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

Open source

Distributed under a liberal [BSD license](#), NumPy is developed and maintained [publicly on GitHub](#) by a vibrant, responsive, and diverse [community](#).

Interoperable

NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries.

Performant

The core of NumPy is well-optimized C code. Enjoy the flexibility of Python with the speed of compiled code.

Easy to use

NumPy's high level syntax makes it accessible and productive for programmers from any background or experience level.

4

1. NumPy 특징

- 강력한 N차원 배열
 - 빠르고 다양한 배열 연산 지원
 - 벡터화, 인덱싱, 브로드캐스팅 개념으로 효율적인 배열 연산
 - 수치 계산 도구
- 포괄적인 수학 함수
 - 난수 생성기
 - 선형 대수 루틴
 - 푸리에 변환 등 다양한 수치 계산 기능
- 오픈 소스
 - 관대한 BSD 라이선스로 배포
 - 활발하고 다양한 커뮤니티에 의해 개발 및 유지보수
 - GitHub를 통한 공개적인 개발 환경

1. NumPy 특징

- 상호 운용성
 - 다양한 하드웨어 및 컴퓨팅 플랫폼과의 호환성
 - 분산 컴퓨팅, GPU, 희소 배열 라이브러리와 함께 사용 가능
- 성능 최적화
 - C로 최적화된 핵심 코드
 - Python의 유연성과 컴파일된 코드의 속도 결합
- 사용 편의성
 - 높은 수준의 문법으로 쉽게 사용 가능
 - 배경이나 경험 수준에 상관없이 접근성과 생산성 제공

1. NumPy 활용 영역

- 데이터 분석
- 기계 학습
- 과학 연구
- 신호 및 이미지 처리
- 그 외, 다양한 영역에서 활용

1. NumPy 설치 및 사용

- Installation

```
pip install numpy
```

- Colab에 설치되어 있음.

- Import 및 버전 확인

```
import numpy as np  
  
print(np.version.full_version)  
  
# output : 설치된 버전 정보가 표시됨  
1.24.3
```

NumPy 배열

Array

2 배열(array)

- NumPy의 핵심 데이터 구조
- 동일한 데이터 타입의 요소들로 구성된 다차원 그리드
- 인덱스(index)에 의해 식별되는 요소들(elements)로 구성
- Python 리스트에 비해 메모리 사용과 연산 속도 측면에서 효율적



2 리스트로 배열 생성하기

- **np.array()** 함수 사용
 - numpy를 np 이름으로 import 했을 경우
- 1차원 배열

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

1	2	3	4	5
---	---	---	---	---

2 내장함수로 배열 만들기

- 주어진 구간에서 연속된 간격의 숫자 배열 생성
- `np.arange([start,] stop[, step,])`
 - 시작/끝/스텝 인자에 따라 배열 크기가 달라짐 (계산 필요)
- `np.linspace(start, stop, num=50)`
 - 시작/끝 값을 포함하도록 배열 크기를 설정할 수 있음

2

np.arange()

```
a = np.arange(5)  
print(a)
```

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

```
b = np.arange(3, 7)  
print(b)
```

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

```
c = np.arange(1, 11, 2)  
print(c)
```

```
array([1, 3, 5, 7, 9])
```

- `np.ones()`
 - 모든 성분이 1인 배열 생성
- `np.zeros()`
 - 모든 성분이 0인 배열 생성
- `np.full()`
 - 0,1이 아닌 다른 값으로 초기화 할 수 있음

2

배열 생성/초기화 함수

```
a = np.ones(4)  
print(a)
```

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

```
b = np.zeros(4)  
print(b)
```

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

```
c = np.full(4, 7)  
print(c)
```

```
array([7, 7, 7, 7])
```


2

이름이 "_like"로 끝나는 배열 생성 함수

- shape, dtype이 동일한 배열 생성

```
a = np.array([1,2,3])
```

```
np.ones_like(a)          → array([1., 1., 1.])
```

```
np.zeros_like(a)         → array([0., 0., 0.])
```

```
np.full_like(a, 100)     → array([100., 100., 100.])
```

2

랜덤 값을 가지는 배열 생성

- 구간 [0.0, 1.0) 사이의 랜덤 실수 값을 size개 반환
 - `np.random.random(size)`

```
np.random.random(5)
```

```
array([0.417 , 0.7203, 0.0001, 0.3023, 0.1468])
```

- 구간을 10~15로 변경하고 싶다면?

```
np.random.random(5) * 5 + 10
```

```
array([13.884 , 14.3765, 14.3453, 10.3699, 12.748 ])
```

2

2차원 배열 생성

- 리스트로 2차원 배열 생성 (2개의 축)

```
a1 = np.array([[1, 2, 3], [4, 5, 6]])
```

- 1차원 배열을 reshape 해서 생성

```
a2 = np.arange(1, 7).reshape(2, 3)
```

2

3차원 이상의 배열 (tensor) 생성

- 1차원 배열을 만들어 reshape 으로 변형

```
np.arange(36).reshape(3, 2, 6)
```

```
array([[[ 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, 32, 33, 34, 35]]])
```

2

랜덤 3차원 배열 (정규분포)

```
np.random.randn(24).reshape(4, 2, -1) # -1: 자동 계산
```

```
array([[[ 0.0641,  0.7999,  0.5685],  
        [-0.9565, -0.9306,  1.3785]],  
       [[ 0.1329, -0.88  ,  0.3783],  
        [-0.7978, -1.3658, -0.3849]],  
       [[-0.0283,  0.678 , -0.076 ],  
        [-1.6788,  0.7791, -2.1975]],  
       [[ 1.365 , -0.2176, -0.3248],  
        [ 1.2052,  2.3149,  1.2082]])
```

배열의 속성 확인

- **shape**
 - Tuple of array dimensions (각 차원의 크기)
- **size**
 - Number of elements in the array (총 요소 개수)
- **ndim**
 - Number of array dimensions (몇 차원?)
- **dtype**
 - Data type of the array's elements (자료형)

```
a = np.array([1, 2, 3, 4, 5, 6])  
b = np.array([[1, 2, 3], [4, 5, 6]])  
  
print(a.ndim) # ndim is 1  
print(b.ndim) # ndim is 2
```


3 크기 속성 (shape, size)

```
a = np.arange(4)
```

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

```
a.shape          # → (4,)
```

```
a.size           # → 4
```

```
print(a.shape, a.size)
```

```
(4,), 4
```

```
b = a.reshape(2, 2)
```

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

```
b.shape          # → (2, 2)
```

```
b.size           # → 4
```

```
print(b.shape, b.size)
```

```
(2,2), 4
```

3 N-Dimensional array

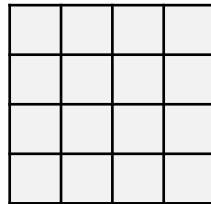
1D



`[1,2,3,4]`

vector

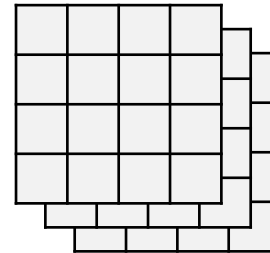
2D



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

matrix

3D



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

tensor

...

...

3

Dimension of 1d and 2d arrays

```
v = np.array([1, 2, 3])
```

```
v.shape # (3,)
```

```
v.ndim # 1
```

```
v2 = np.array([[1],
                [2],
                [3]])
```

```
v2.shape # (3, 1)
```

```
v2.ndim # 2
```

Matrix

- rows and columns

Row 1	a_{11}	a_{12}	a_{13}
Row 2	a_{21}	a_{22}	a_{23}
Row 3	a_{31}	a_{32}	a_{33}

Column 1 Column 1 Column 1

a_{11}	a_{12}	a_{13}
a_{21}	a_{22}	a_{23}
a_{31}	a_{32}	a_{33}

3 Data-type 속성

Data-type	Default	Aliases
Bool 불리언	<code>np.bool_</code>	
(u)int (unsigned) 정수	<code>np.int_</code>	<code>np.int[8, 16, 32, 64]</code> <code>np.uint[8, 16, 32, 64]</code>
float 실수	<code>np.float_</code>	<code>np.float[16, 32, 64]</code>
complex 복소수	<code>np.complex_</code>	<code>np.complex[64, 128]</code>

3 배열 dtype 확인

```
a = np.array([0, 1, 2, 3])  
print(a.dtype)
```

int32

```
b = np.array([0, 1/2, 2/3, 3/4])  
print(b.dtype)
```

float64

3

특정 dtype 으로 배열 생성

```
a = np.array([1, 2, 3, 4], dtype=np.float32)
print(a.dtype)
```

float32

```
b = np.array([0, 1/2, 2/3, 3/4], dtype=np.int32)
print(b)
print(b.dtype)
```

[0 0 0 0]

int32

- `np.astype()`

```
a = np.arange(1, 6)
b = a.astype(np.float32)
print(a)
pinrt(b)
```

```
array([1, 2, 3, 4, 5])
array([1., 2., 3., 4., 5.], dtype=float32))
```

배열의 연산

operations

- Vectorization (벡터화)
 - for 문 없이 배열 데이터 일괄처리
- 같은 크기 배열 연산
 - 대응 되는 원소 단위로 각각 처리
- 스칼라 연산
 - 배열 내 모든 원소에 스칼라 인자 적용
- 다른 크기 배열 연산
 - 크기가 작은 배열 요소가 복제되어 큰 배열과 호환되도록 처리

4

같은 크기의 배열 연산

```
a = np.arange(16).reshape(4,4)
b = a.T # transpose of a
c = a * b
d = a == b
```

[[0 4 16 36]	[[True False False False]
[4 25 54 91]	[False True False False]
[16 54 100 154]	[False False True False]
[36 91 154 225]]	[False False False True]]

```
a = np.arange(16).reshape(4,4)
b = a + 10
c = a > 8
print(b)
print(c)
```

```
[[10 11 12 13]
 [14 15 16 17]
 [18 19 20 21]
 [22 23 24 25]]
```

```
[[False False False False]
 [False False False False]
 [False  True  True  True]
 [ True  True  True  True]]
```

- **브로드캐스팅 (broadcasting)**
 - 차원이 다른 배열 연산에서 자동으로 차원을 확장하는 기능
 - 배열의 크기를 일치시키지 않고 효율적 연산 수행 가능
 - 서로 다른 임의의 크기 배열 연산에 모두 허용되는 것은 아님

```
a = np.arange(24).reshape(4, 6) + 1  
b = np.arange(6) + 1  
a + b
```

$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 \\ 19 & 20 & 21 & 22 & 23 & 24 \end{bmatrix}$	+	$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$	=	$\begin{bmatrix} 2 & 4 & 6 & 8 & 10 & 12 \\ 8 & 10 & 12 & 14 & 16 & 18 \\ 14 & 16 & 18 & 20 & 22 & 24 \\ 20 & 22 & 24 & 26 & 28 & 30 \end{bmatrix}$
---	---	--	---	---

```
a = np.arange(4).reshape(1,4)
b = np.arange(4).reshape(4,1)
a+b
```

```
a = np.arange(12).reshape(4, 3)
b = np.arange(4)
a + b
```

- Python and, or, not \rightarrow &, |, ~

```
a = np.arange(16).reshape(4,4)
```

```
(a > 4) & (a < 10)
```

```
[[False False False False]
 [False  True  True  True]
 [ True  True False False]
 [False False False False]]
```


- **Dot product**

```
v1 = np.arange(1, 4)           # array([1, 2, 3])  
v2 = np.arange(4, 7)          # array([4, 5, 6])  
v1 @ v2, v1.dot(v2)
```

(32, 32)

- 행렬 곱

```
m1 = np.arange(4).reshape(2,2) + 1
```

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

```
m1 @ m1
```

```
array([[ 7, 10], [15, 22]])
```

- 역행렬

```
np.linalg.inv(m1)
```

```
array([[ -2. ,  1. ], [  1.5, -0.5]])
```

배열 인덱싱/슬라이싱

Indexing and slicing

- 인덱스(index)

- 배열의 특정 요소에 접근하거나 선택하기 위한 방법
- 인덱스는 0부터 시작
- 배열이름[index]

배열 a :

1	2	3	4	5
---	---	---	---	---

index :

0	1	2	3	4
---	---	---	---	---

$a[2] \rightarrow 3$

$a[-1] \rightarrow 5$

5

Slicing을 통한 부분 배열 추출

- 배열이름[start: stop: step]

```
a = np.arange(9) + 1  
a[3:]          # [4, 5, 6, 7, 8, 9]  
a[:4]          # [1, 2, 3, 4]  
a[3:6]         # [4, 5, 6]  
a[2::2]        # [3, 5, 7, 9]
```

- 배열 슬라이싱의 결과는 원본 배열의 뷰(view)
→ 변경 결과가 원본에 반영됨!
- 데이터 복사가 필요한 경우 `copy()`를 명시해야 함

- 기존 배열의 데이터에 대한 **다른 표현 방식**을 제공하는 개념
 - 배열의 view는 원본 배열의 일부 데이터에 대한 포인터
 - 뷰를 통해 배열의 shape을 변경하면 데이터 복사가 발생하지 않음
 - 메모리 사용을 줄이고 배열 조작의 효율성을 높이는 데 도움이 됨

- Slicing 한 결과를 수정하면 원본 데이터에 영향을 줌

```
arr = np.arange(10)
arr_slice = arr[5:8]
arr_slice
arr_slice[1] = 123
arr
```

- 배열의 복사를 위해서는 아래와 같이 copy()를 호출

```
arr_slice = arr[5:8].copy()
```


5

Indexing and slicing : 2d array

```
A = np.array( 0 [[ 1,  2,  3,  4,  5,  6],  
1 [ 7,  8,  9, 10, 11, 12],  
2 [13, 14, 15, 16, 17, 18],  
3 [19, 20, 21, 22, 23, 24] )
```

Diagram illustrating indexing and slicing on a 2D array `A`.

The array `A` is defined as:

```
A = np.array([ [1, 2, 3, 4, 5, 6],  
               [7, 8, 9, 10, 11, 12],  
               [13, 14, 15, 16, 17, 18],  
               [19, 20, 21, 22, 23, 24] ])
```

Indices are shown above the columns (0 to 5) and to the left of the rows (0 to 3).

Two specific indexing operations are highlighted:

- `A[: , :]` (Red arrow) indicates slicing all rows and all columns.
- `A[0, 4]` (Orange arrow) indicates accessing the element at row 0, column 4 (the value 5).

- Boolean array
 - 배열 요소가 True, False로 구성 된 배열
- Boolean indexing
 - 배열 각 요소의 선택 여부를 True/False 로 지정
 - 요소가 True 인 인덱스의 값만 추출
- Python 연산자 and, or는 Boolean 배열에 쓸 수 없음
 - &, |, ~(not) 사용

5 Boolean indexing

- 랜덤한 숫자가 0.5보다 큰 인덱스에 대해서 a 요소 선택

```
a = np.arange(10)
b = np.random.random(10) > 0.5

b, a[b]
```

5 Fancy (advanced) indexing

- 정수 배열을 사용한 인덱싱

```
a = np.arange(10) + 10
```

```
[0 1 2 3 4 5 6 7 8 9]
```

```
idx1 = np.array([0, 5, 3])
```

```
a[idx1]
```

```
[10 15 13]
```

```
idx2 = np.array([[0, 5], [1, 2]])
```

```
a[idx2]
```

```
[[10 15]
```

```
 [11 12]]
```

배열 조작하기

6

축(axis)의 개념

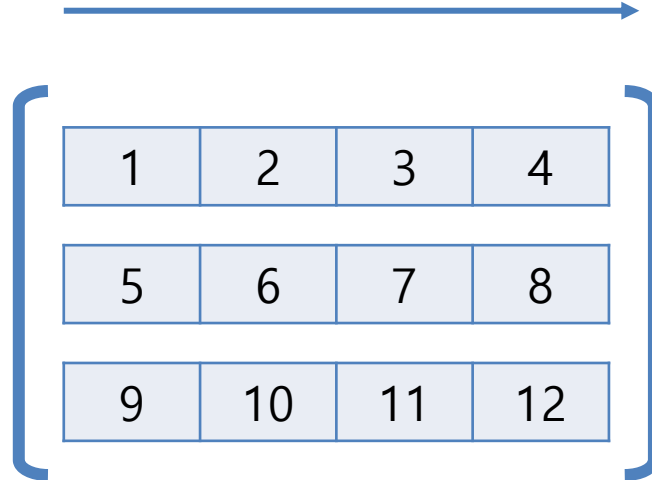
```
a = np.arange(1, 13).reshape(3, 4)
print(a)
```

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

```
a.shape
```

```
(3, 4)
```

axis 1 (n차원인 경우,
n-1번 축이 됨)



axis 0

6

배열 transpose

- 배열의 차원을 바꾸는 것
- 배열의 축(axis)을 서로 교환하여 전치(transpose)된 뷰를 반환
- `transpose()` or `T`
- `swapaxes()` method

```
a = np.arange(10).reshape(2, -1)
print(a.transpose())
print(a.T)
print(a.swapaxes(0, 1))
```

6 transpose 주의

- 1차원 배열은 transpose 할 수 없다

```
a = np.arange(3)
print('a:', a)
print('a.T:', a.T)
```

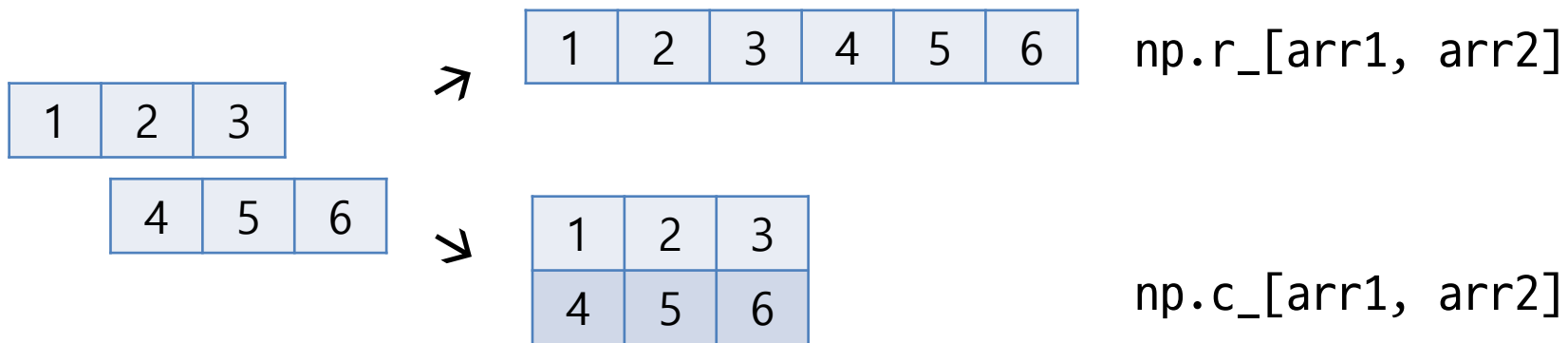
```
a: array([0, 1, 2])
a.T: array([0, 1, 2])
```


6

배열 합치기

- 가로, 세로로 연결

```
arr1 = np.array([1, 2, 3])  
arr2 = np.array([4, 5, 6])
```



- `np.concatenate()`

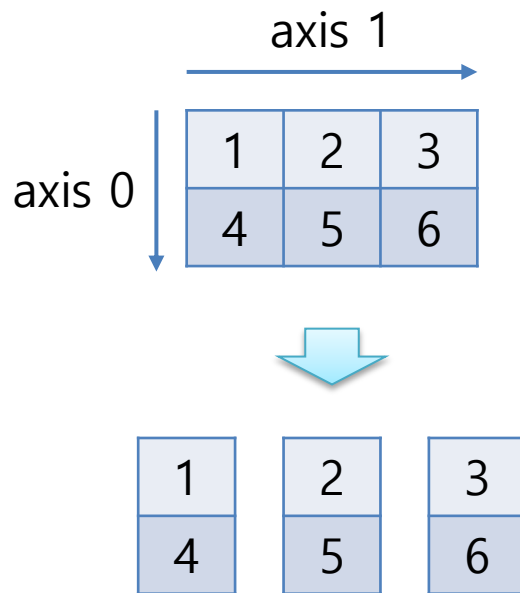
```
arr1 = np.array([1, 2, 3])  
arr2 = np.array([4, 5, 6])  
np.concatenate([arr1, arr2], axis=0)
```

6

배열 분리하기

- 축을 따라 n개로 분리

```
a = np.arange(1, 7).reshape(2, 3)  
np.split(a, 3, axis=1)
```



6

배열 반복하기

- 배열의 요소 반복

```
np.repeat(arr1, 2, axis=0)  
[1 2 3] → [1 1 2 2 3 3]
```

- 배열 전체 반복

```
np.tile(arr1, (2, 2))  
[1 2 3] → [[1 2 3 1 2 3]  
            [1 2 3 1 2 3]]
```

6

다차원을 1차원으로 만들기

- `flatten()`

```
a = np.arange(9).reshape(3,-1)  
print(a)
```

```
[[0 1 2]  
 [3 4 5]  
 [6 7 8]]
```

```
b = a.flatten()  
print(b)
```

```
[0 1 2 3 4 5 6 7 8]
```

- `np.where(조건, 참인 경우, 거짓인 경우)`
 - `x = 3 if y == 3 else -3` 의 배열 버전

```
x = np.array([1.1, 1.2, 1.3, 1.4, 1.5])  
y = np.array([2.1, 2.2, 2.3, 2.4, 2.5])  
cond = np.array([True, False, True, True, False])  
np.where(cond, x, y)
```

- `sum`, `mean`, `std`, `var`
 - 배열의 합, 평균, 표준편차, 분산을 각각 계산
 - 축을 지정하면 지정된 축에 대해서 연산
- `min`, `max`, `argmin`, `argmax`
 - 최대값, 최솟값, 최대값의 index, 최소값의 index 각각 계산
- `cumsum`, `cumprod`
 - 누적 합, 누적 곱 계산

6

1차원 축 따라 배열 합 계산

```
d = np.random.randn(25).reshape(5,-1)  
d[3, :] = 0  
d[4, :] = np.arange(5)
```

```
[[ 1.6243 -0.6118 -0.5282 -1.073   0.8654]  
 [-2.3015  1.7448 -0.7612  0.319  -0.2494]  
 [ 1.4621 -2.0601 -0.3224 -0.3841  1.1338]  
 [ 0.      0.      0.      0.      0.    ]  
 [ 0.      1.      2.      3.      4.    ]]
```

```
np.sum(d)
```

```
8.857856
```

```
np.sum(d, axis=1)
```

```
[ 0.2769, -1.2483, -0.1707, 0. , 10. ]
```


- 불리언 배열
 - `any` : 요소 전체 or 연산 (하나 이상 True 인가?)
 - `all` : 요소 전체 and 연산 (모두 True 인가?)

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

6 sort, unique, in1d

- **sort**: 배열 요소 정렬

```
arr = np.random.randn(10)
arr.sort()
arr
```

- **unique**: 중복 제거

```
s = np.array([1,1,1,2,3,3,4])
np.unique(s)
```

- **in1d**: 요소 확인

```
values = np.array([6, 0, 0, 3, 2, 5, 6])
np.in1d(values, [3, 6])
```

파일 입/출력

7 np.save, np.load

- 배열 데이터 파일에 저장하기/ 불러오기
 - np.save(파일이름, 배열변수)
 - np.load(파일이름)

```
arr = np.arange(100)
```

```
np.save("some_array", arr)
```

```
# 파일 확장자 npy가 자동으로 붙음, 바이너리 파일
```

```
np.load("some_array.npy")
```

Thank you

Q & A

