



PYTHON - 데이터분석

00

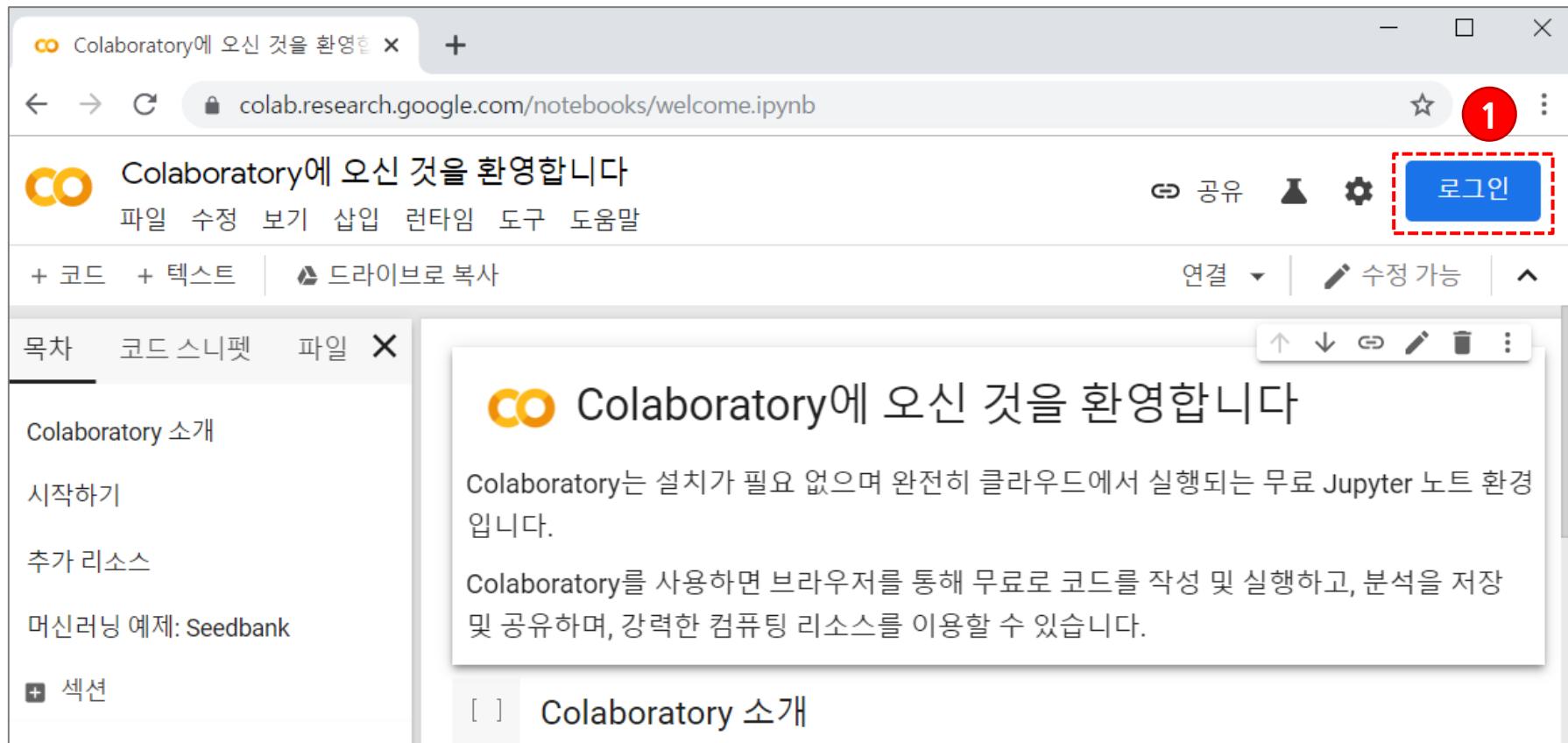
환경 설정 (Colab 사용)

colab 접속



▣ 다음 사이트를 사용한다

▣ <https://colab.research.google.com/> → 접속 후 [로그인] // google 계정



The screenshot shows the Google Colaboratory landing page. At the top, there is a navigation bar with a back arrow, forward arrow, refresh button, and a search bar containing the URL "colab.research.google.com/notebooks/welcome.ipynb". To the right of the search bar is a star icon and a red circular badge with the number "1". Below the navigation bar, the main content area has a title "Colaboratory에 오신 것을 환영합니다" (Welcome to Colaboratory) and a sub-section "Colaboratory는 설치가 필요 없으며 완전히 클라우드에서 실행되는 무료 Jupyter 노트 환경입니다." (Colaboratory does not require installation and is a free Jupyter notebook environment running entirely in the cloud). On the left side, there is a sidebar with links: "Colaboratory 소개", "시작하기", "추가 리소스", "머신러닝 예제: Seedbank", and "[+] 섹션". At the bottom of the main content area, there is a button labeled "Colaboratory 소개". The "로그인" (Login) button in the top right corner is highlighted with a red dashed box and a red circle with the number "1" above it.

파일 업로드



- 로그인 한 뒤 표시되는 창에서 [업로드] -[파일 선택]을 차례로 클릭한다



- 파일 선택 창에서 파일을 선택하고 [열기]를 클릭한다
- Colab에서 해당 파일이 열린다
- 수행 후 기본적으로 Google Drive의 '/Colab Notebooks' 폴더에 저장 된다

새로운 ‘노트’ 작성



☞ 팝업 창에서 [새 PYTHON 3 노트]를 클릭한다

The screenshot shows a window titled 'New Note'. At the top, there are tabs: 예 (Example), 최근 사용 (Recent Use), Google 드라이브 (Google Drive), GitHub, and 업로드 (Upload). A dashed box highlights the first five steps listed below the tabs:

1. 새 노트 생성
2. 노트의 제목 (클릭하여 수정 가능)
3. 왼쪽 도구모음 (각 ICON을 클릭해 볼 것)
4. 왼쪽 창 포함 내용
5. 셀 제거 아이콘

On the right side, there is a blue button labeled '1 새 노트' with a dashed box around it, and a link labeled '취소'.

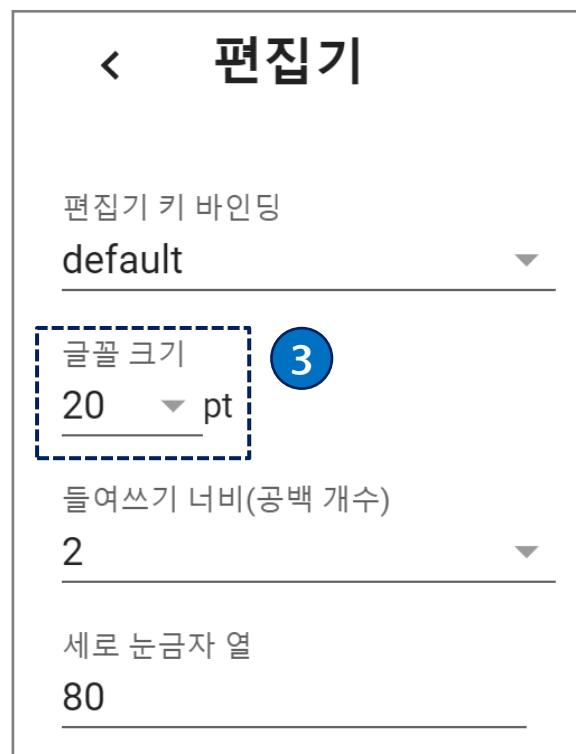
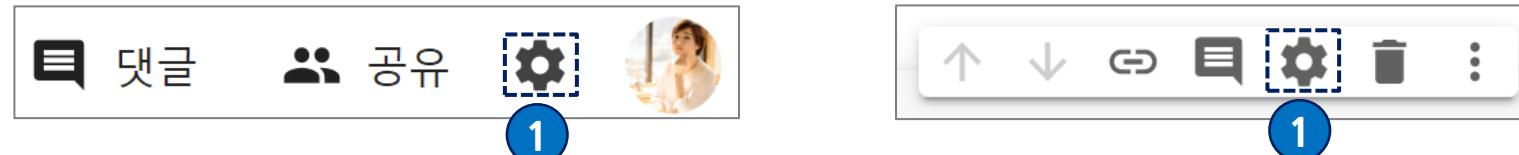
The screenshot shows the Jupyter Notebook interface. At the top, there is a toolbar with icons for CO, Untitled2.ipynb, star, 댓글 (Comment), 공유 (Share), settings, and profile. Below the toolbar, there is a menu bar with 파일 (File), 수정 (Edit), 보기 (View), 삽입 (Insert), 런타임 (Runtime), 도구 (Tools), 도움말 (Help), and 모든 변경... (All Changes...). On the left, there is a sidebar with icons for + 코드 (Code), + 텍스트 (Text), and a trash bin. The main area shows a play button icon. At the bottom, there is a toolbar with icons for up, down, link, comment, settings, and a trash bin, with the number '5' in a blue circle next to the trash bin icon. A dashed box highlights the fourth step:

- 1 새 노트 생성
- 2 노트의 제목 (클릭하여 수정 가능)
- 3 왼쪽 도구모음 (각 ICON을 클릭해 볼 것)
- 4 목차 (Table of Contents) [dashed box]
- 5 셀 제거 아이콘 (Delete Cell icon) [dashed box]

설정



메뉴 옆 도구모음 또는 셀의 도구모음에서 '설정' 아이콘 클릭



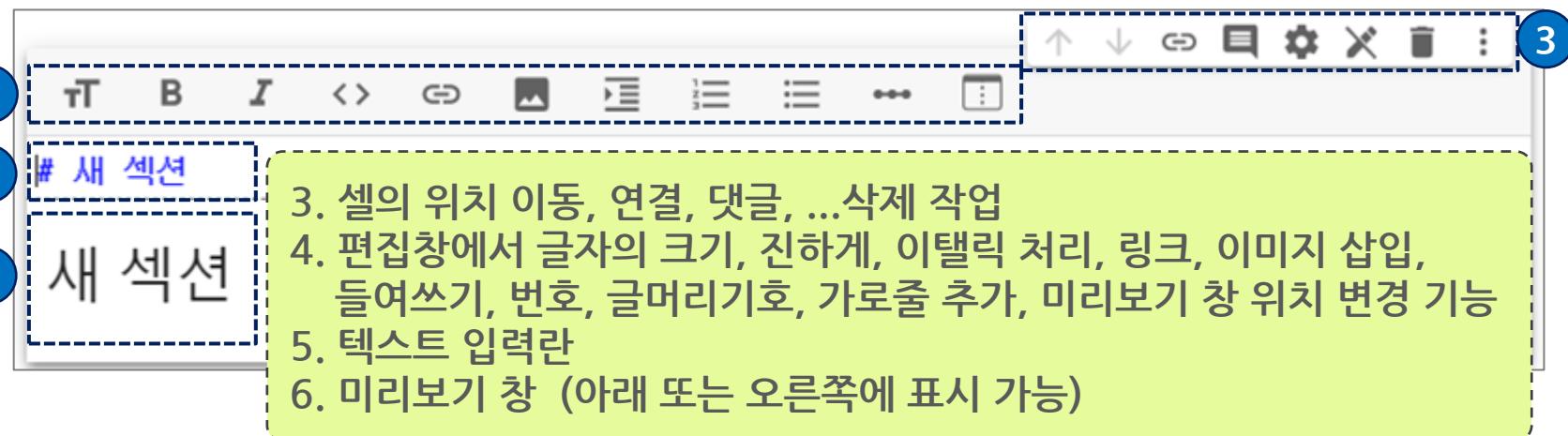
‘섹션’ 추가



- ▶ 왼쪽 도구모음에서 ‘목차’ 아이콘 - [+섹션]을 클릭하여 ‘섹션’을 추가한다
- ▶ 섹션은 여러 개의 ‘텍스트’셀과 ‘코드’ 셀의 묶음이다



- ▶ 추가된 ‘새 섹션’의 모습은 다음과 같다 ('텍스트' 셀이 기본으로 추가된다)



'코드' 셀 추가

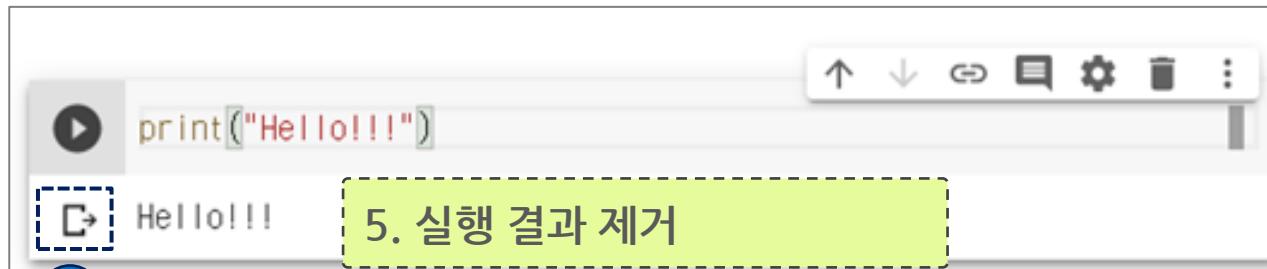


창의 상단에 [+ 코드]를 클릭하여 '코드' 셀을 추가한다

실제 실행할 python 코드를 입력 하는 용도로 사용한다



1. 새로운 '코드' 셀 추가 버튼
2. 코드 입력 부분
3. '코드' 실행 (단축키 : Ctrl + Enter)
4. 위치 이동, 연결, 댓글, ... 삭제 작업



```
print("Hello!!!")
```

5. 실행 결과 제거

‘텍스트’ 셀 추가



창의 상단에 [+ 텍스트]를 클릭하여 ‘텍스트’ 셀을 추가한다

코드에 대한 설명을 할 수 있으며 다양한 요소를 넣어 작성할 수 있다



1. 새로운 ‘텍스트’ 셀 추가 버튼
2. ‘텍스트’ 셀을 더블클릭 하여 편집전환
3. 편집 가능한 텍스트 셀의 모습
4. 위치 이동, 연결, 댓글, ... 삭제 작업



‘섹션’ 접기, 펼치기



▶ 섹션 (텍스트 + 코드)의 하위 셀 접기, 펼치기 기능의 사용은 다음과 같다



1. 섹션 제목의 하위 셀 접기
2. 섹션 제목의 하위 셀 펼치기
3. 코드 실행 횟수 (실행 결과를 제거해도 기억하고 있음)

새로운 노트 작성 연습



새로운 노트를 생성하여 파일이름을 수정하고 코드 작성 후 실행해 본다

The screenshot shows a Jupyter Notebook interface with the following elements:

- 1.** File tab: Shows the file name "pandas_00.ipynb".
- 2.** Run button: Located in the toolbar.
- 3.** Code cell: Contains Python code to create a DataFrame from a dictionary. The code is:

```
import pandas as pd
mydata = {'ID' : (20030101, 20030102, 20040103),
          'name' : ('Julie Yoon', 'Sam Park', 'Rose Lee') }
data = pd.DataFrame(mydata, index=[0, 1, 2])
print(data)
print(data['name'])
```
- 4.** Output cell: Shows the resulting DataFrame and its columns.
- 5.** Refresh button: Located in the toolbar.

	ID	name
0	20030101	Julie Yoon
1	20030102	Sam Park
2	20040103	Rose Lee

Output details:
0 Julie Yoon
1 Sam Park
2 Rose Lee
Name: name, dtype: object

A green callout box on the right lists the steps:

- 파일이름 변경
- 코드 입력 부분
- 코드 실행 아이콘 (Ctrl + Enter)
- 코드 실행 결과 출력
- 실행 결과 삭제
- 파일 저장은 '파일' - '저장' 메뉴 또는 (Ctrl + S) 사용

주요 메뉴



☞ Colab의 메뉴를 이용한 기본 작업을 살펴 보도록 한다

☞ 파일 저장 및 불러오기 관련 메뉴

새 노트	파일 - 새 노트	
노트 열기 (기존)	파일 - 노트 열기	Ctrl + O
노트 저장	파일 - 저장	Ctrl + S
노트 업로드	파일 - 노트 업로드	
휴지통으로 이동	파일 - 휴지통으로 이동	삭제 전에 휴지통으로 넣은 것
.ipynb 파일 저장	파일 - .ipynb 저장	사용자의 '다운로드' 폴더에 저장됨
.py 파일 저장	파일 - .py 저장	사용자의 '다운로드' 폴더에 저장됨

☞ 이외 수정, 보기, 삽입, 런타임, 도구, 도움말 메뉴를 한 번씩 살펴 보도록 하자

노트 공유



- 노트 상단 오른쪽 ‘공유’ 아이콘을 클릭하여 노트를 공유할 수 있다
- 링크 또는 메일로 보내기가 가능하며 ‘IE’에서는 실행되지 않는다 (Chrome 권장)

The screenshot shows the sharing interface for a note. Step 1 highlights the 'Share' icon (a person icon) in the top right corner of the note card. Step 2 highlights the 'Get shareable link' button (a link icon) in the 'Sharing' section. Step 3 highlights the input field for entering a name or email address.

다른 사용자와 공유

2 공유 가능한 링크 가져오기

3 이름 또는 이메일 주소 입력...

사용자

완료

- 공유 실행을 위한 아이콘
- ‘링크’를 만들어 노트 공유
 - 링크가 클립보드에 복사되므로 바로 붙여 넣기 가능
- ‘이메일’로 노트 링크 전송
 - 메일주소를 입력하여 링크를 공유함

Google Drive에서 노트가 저장된 디렉터리가 ‘공유’ 상태여야 함

Google Drive 보기



▶ 노트 화면 상단 왼쪽의 아이콘을 클릭하여 Google Drive를 확인할 수 있다



2

3

4

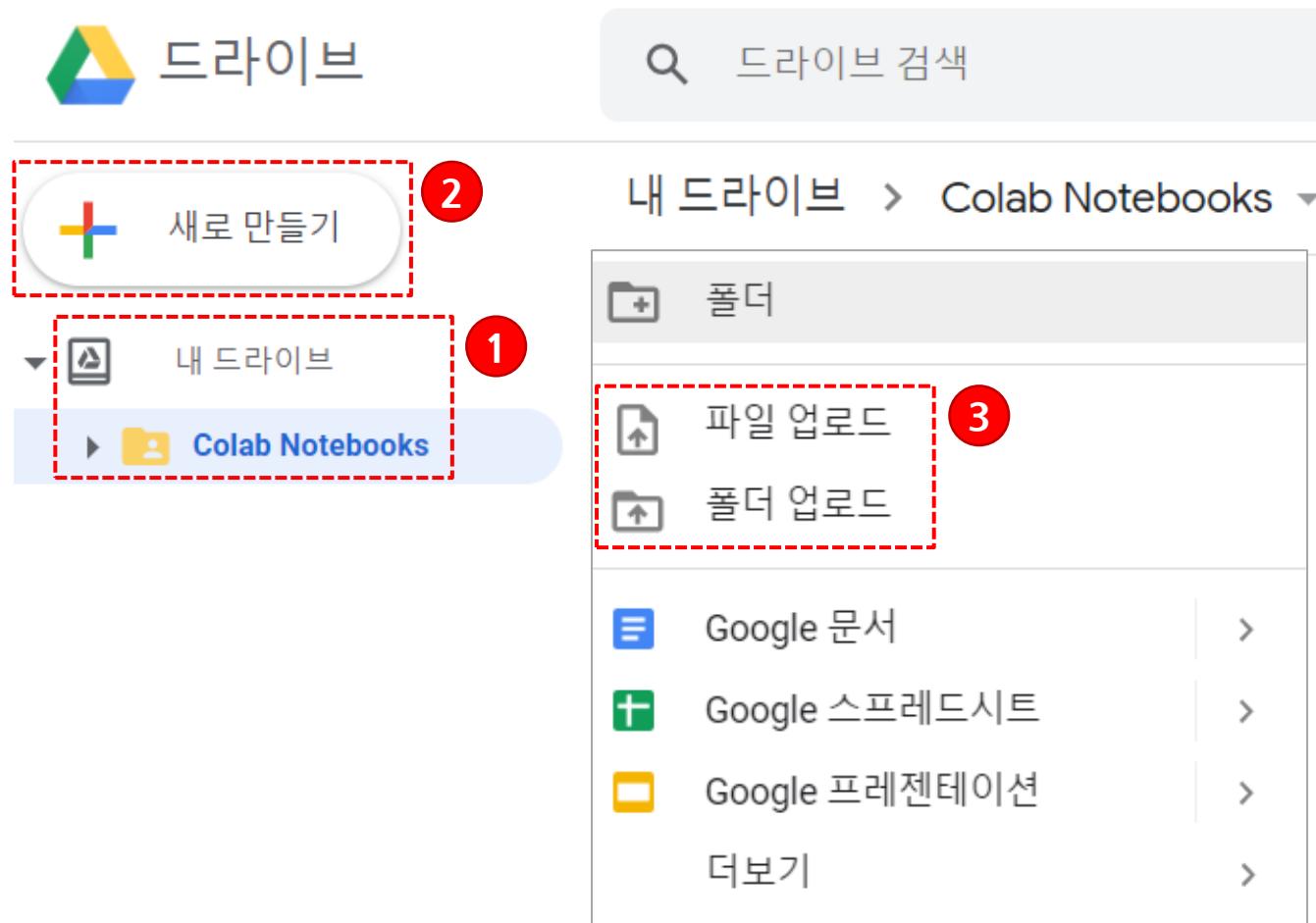
이름	소유자	마지막으로 수정...
pandas_000.ipynb	나	오후 3:06 나
python_000.ipynb	나	오후 2:55 나

1. Google Drive 표시 아이콘
2. 내 드라이브의 폴더를 확인할 수 있음
3. 목록 보기 방식을 변경하는 아이콘 (목록 보기, 바둑판 보기)
4. 문서를 더블 클릭하여 Colab 으로 실행 가능

Google Drive에 파일/폴더 업로드



- 업로드 위치 선택 후 '새로 만들기'를 클릭하여 '파일' / '폴더'를 업로드 한다



Google 드라이브 폴더에 공유 설정



The screenshot shows the Google Drive interface. On the left, there's a sidebar with a '새로 만들기' button, a '내 드라이브' section containing folders like '.ipynb_checkpoints', '00_SKI' (which is highlighted with a red dashed box and has a red circle with '1'), 'Day1', 'AI_Basic', 'Classroom', 'Colab Notebooks', and 'newphotos', and a progress bar indicating '15GB 중 426MB 사용'. At the bottom of the sidebar is a '저장용량 구매' link. The main area shows a search bar and a list of options for the selected folder '00_SKI': '연결 앱', '새 폴더', '공유' (highlighted with a red dashed box and has a red circle with '2'), '공유 가능한 링크 가져오기', '드라이브에 바로가기 추가', '이동', '중요 문서함에 추가', '이름 바꾸기', '색상 변경', '00_SKI 내에서 검색', '다운로드', and '삭제'. A tooltip window titled '링크 보기' is open over the '공유' option, containing the text '제한됨 추가된 사용자만 이 링크로 항목을 열 수 있음' and '링크가 있는 모든 사용자로 변경' (highlighted with a red dashed box and has a red circle with '3').

드라이브

새로 만들기

내 드라이브

.ipynb_checkpoints

00_SKI 1

.ipynb_checkpoint

Day1

AI_Basic

Classroom

Colab Notebooks

newphotos

15GB 중 426MB 사용

저장용량 구매

드라이브에서 검색

연결 앱

새 폴더

2

공유 가능한 링크 가져오기

드라이브에 바로가기 추가

이동

중요 문서함에 추가

이름 바꾸기

색상 변경

00_SKI 내에서 검색

다운로드

삭제

링크 보기

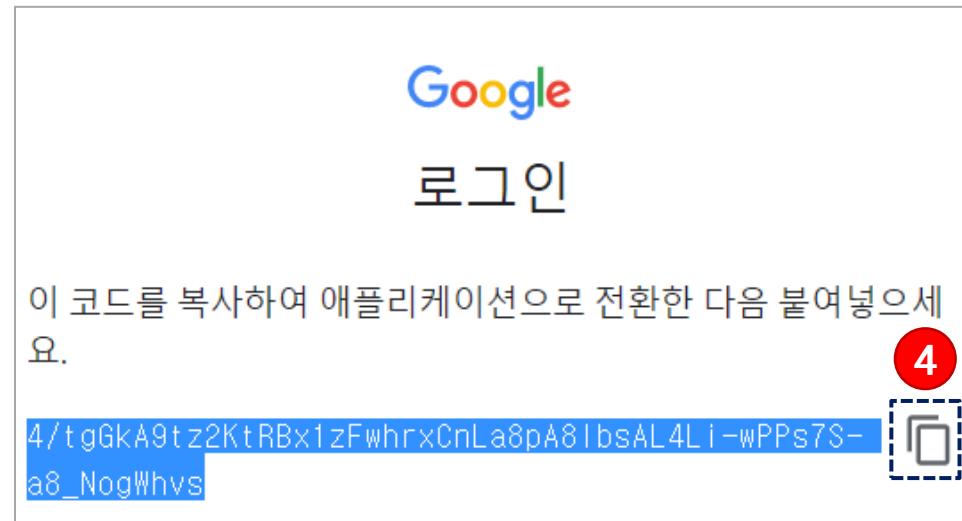
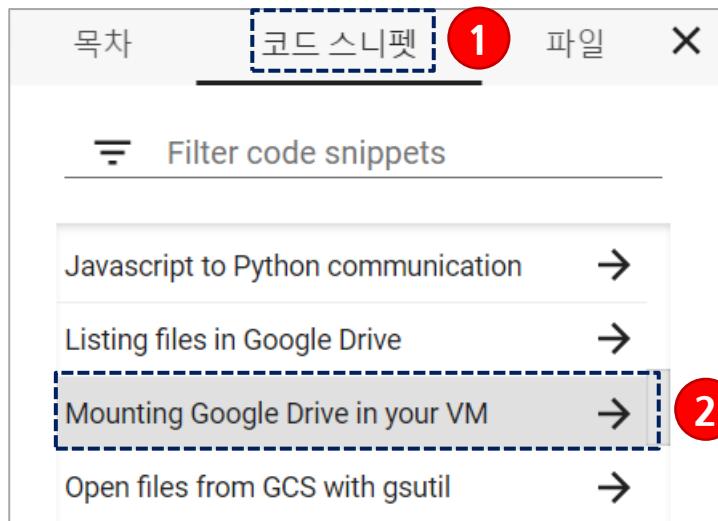
제한됨 추가된 사용자만 이 링크로 항목을 열 수 있음

링크가 있는 모든 사용자로 변경 3

Google Drive 연결 - 1



- DB, 사용자가 만든 모듈/패키지 등을 연동해서 사용하고자 할 때 사용함
- Google Drive에 파일을 업로드하고 colab의 ipynb 파일을 열고 다음을 처리한다



```
from google.colab import drive
drive.mount('/gdrive')
```

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=417416335701.apps.googleusercontent.com&redirect_uri=https://colab.research.google.com/notebooks/api/oauthcallback&response_type=code&scope=https://www.googleapis.com/auth/drive ③

Enter your authorization code: ⑤

- 왼쪽 창에서 ‘코드 스니펫’ 클릭
- 하단의 목록을 스크롤하여 ‘Mounting Google ...’ 선택
- 코드가 삽입되면 링크 클릭
- 새 탭에서 Google 계정 선택 후, 코드 복사
- colab 화면에 복사한 코드 붙여넣기

Google Drive 연결 - 2



- 특정 폴더 연결은 % cd '경로명' 으로 한다
- 우선 앞에서 마운트 하는 코드를 실행하고, % cd '경로명'을 입력 후 실행한다

```
[6] from google.colab import drive  
drive.mount('/gdrive')
```

1

↳ Mounted at /gdrive

```
[8] % cd '/gdrive/My Drive/Colab Notebooks'
```

2

↳ /gdrive/My Drive/Colab Notebooks

1. Google Drive의 인증을 위한 절차를 끝낸 뒤 코드를 '실행' 한다
2. 새로운 '코드' 셀 삽입 후 작업 경로를 변경하는 코드를 작성 후, 코드를 '실행' 한다
작업 경로는 다른 경로일 수 있으며, 예시는 colab에서 기본으로 사용하는 경로이다



01

numpy의 ndarray랑 친해져요!



numpy의 이해



- 통계, 분석, AI 분야의 라이브러리 내부에 다양한 수치연산이 필요함
 - 수치연산을 얼마나 효율적으로 처리하는가에 따라 성능에 많은 영향을 줌
 - numpy는 ndarray라는 자료를 바탕으로 강력한 연산 기능을 제공함
-
- numpy와 다른 python 패키지와의 관계



numpy는 다양한 python 패키지에서 데이터 표현과 연산 기능 구현에 사용됨

numpy 설치



- ▶ numpy는 외부 라이브러리이다
- ▶ 표준 파이썬에 포함되지 않으므로 numpy는 import 하여 사용한다
 - np라는 이름으로 numpy 라이브러리를 가져오라는 명령

```
import numpy as np
```

- ▶ numpy 모듈이 설치 되지 않았다면 다음 명령으로 설치한다
 - pip install numpy
 - Windows 의 경우 명령 프롬프트에서 위의 명령을 실행하면 설치된다
- ▶ 자세한 메서드와 속성에 대한 설명은 다음 링크 참조한다
 - <https://docs.scipy.org/doc/numpy/reference/index.html>
 - <https://numpy.org/devdocs/genindex.html>
 - 자세한 parameter 사용에 대한 종류 및 설명을 확인할 수 있다

Vectorizing 연산



▶ numpy는 Vectorizing 연산을 사용한다

▶ vectorizing 연산의 특징 및 장점

- Loop을 사용하지 않고 배열(ndarray)의 각 요소를 계산 함
- 코드의 높은 가독성, 처리 속도 향상
- 대용량 데이터를 다룰 때 사용하면 성능에 큰 도움이 됨

▶ numpy Vectorizing 연산 원리

Python with Loop

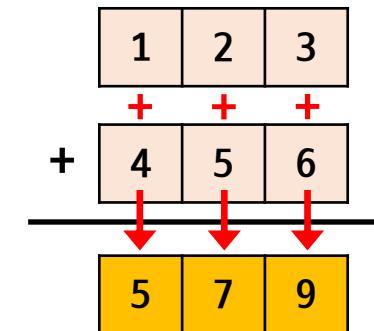
```
A = [ 1, 2, 3]
B = [ 4, 5, 6]

C = []
for a, b in zip(A, B):
    C.append(a + b)
```

python with NumPy

```
import numpy as np
A = np.array([1, 2, 3])
B = np.array([4, 5, 6])
C = A + B
```

numpy 연산 과정
(element by element)



numpy의 성능 예제



- ▶ [예제1] 다음을 실행하여 numpy와 일반 list의 성능 차이를 확인하라
- ▶ 다음은 열의 위치를 변경하는 코드이다

```
a = [[ 1, 2, 3, 4 ] for _ in range(1000000)]  
b = np.array(a)  
print(a[:4], b[:4], sep='\n')
```

데이터 범위를 변경해 가며
차이를 확인해 본다

0:00:00.376637

```
start = datetime.datetime.now()  
c = []  
for x in a:  
    x[2], x[3], x[0], x[1] = x  
    c.append(x)  
end = datetime.datetime.now()  
print(end-start)
```

0:00:00.011031

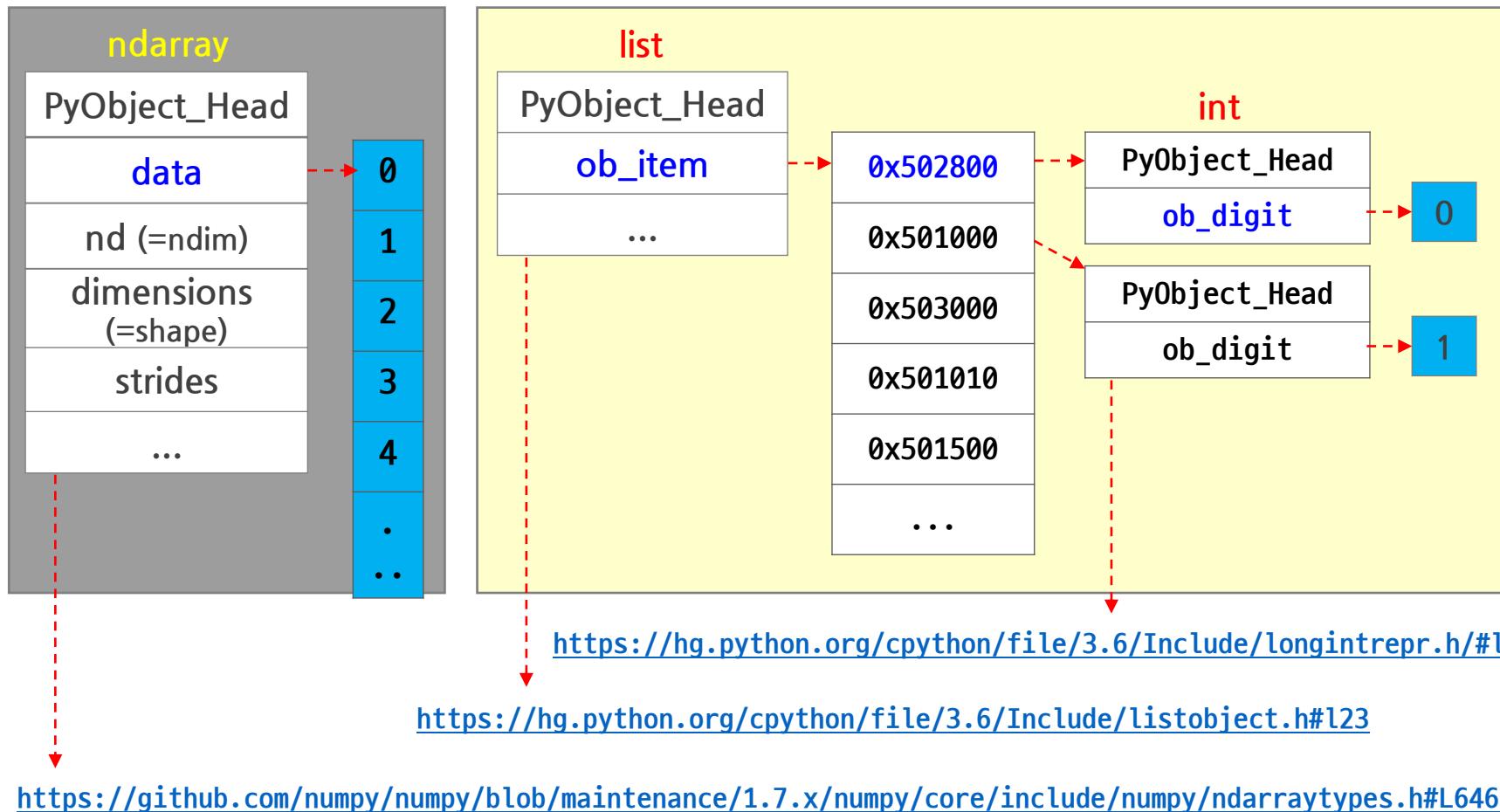
```
start = datetime.datetime.now()  
d = b[ :, [2, 3, 0, 1]]  
end = datetime.datetime.now()  
print(end-start)
```

numpy 를 사용한 코드가 더 간결하며, 빠른 성능인 것을 확인 할 수 있다

ndarray 가 list 보다 빠른 이유는?



▶ numpy의 ndarray와 built-in list의 데이터 접근 방법 차이를 살펴보자



ndarray의 주요 속성



▶ [예제2] ndarray의 주요 속성을 살펴보자

속성	설명
a.ndim	▪ 배열의 깊이(차원)
a.shape	▪ 생성된 배열의 구조에 대한 정보 ▪ tuple 형태, 각 차원 별 요소 개수 정보를 가지고 있음
a.size	▪ 배열의 item 개수
a.dtype	▪ 생성된 배열의 데이터 타입(자료형)에 대한 정보
a.itemsize	▪ item의 메모리 사용 byte 수
a.strides	▪ 메모리에서 데이터의 간격에 대한 정보, shape 및 dtype과 관련이 있음 ▪ tuple 형태, 각 차원 별 데이터 간격 정보를 가지고 있음

```
a = np.array([[0,1,2],[3,4,5]], dtype=np.int32)
```

0	1	2
3	4	5

```
a.ndim      2  
a.shape     (2, 3)  
a.size      6  
a.dtype      int32  
a.itemsize   4  
a.strides   (12, 4)
```

numpy의 자료형



▶ numpy의 자료형은 다음과 같이 분류할 수 있다

자료형	종류
부호 있는 정수형 (int)	int8, int16, int32, int64
부호 없는 정수형 (uint)	uint8, uint16, uint32, uint64
실수형 (float)	float16, float32, float64, float128
복소수(complex)	complex64, complex128, complex256
부울형(boolean)	bool (8bit)
문자열	bytes_(byte string), str_ (unicode string)
객체	object, Container 자료형들은 object로 취급함

numpy 자료형 클래스 사용방법

1. np.int8, np.float32 와 같이 ‘np.자료형’으로 사용
2. 자료형 문자 코드사용 (다음 페이지 참조)
3. 각 자료형 뒤의 8, 16, 32 등의 숫자는 bit 수를 의미하며 8 bit == 1Byte 이다

ndarray 학습 개요



본 과정에서 다루는 numpy의 ndarray 관련 내용은 다음과 같다

ndarray 생성 함수	array-like, 범위(start, stop), random 값 및 0, 1, 특정 값을 사용하여 ndarray를 생성하는 방법
ndarray 변환 함수	ndarray의 shape 또는 dtype을 변경하는 방법
ndarray 연산	산술연산, 비교연산, element-wise, broadcasting
ndarray indexing	다양한 indexer의 사용 방법
numpy의 유용한 함수	unique, where, sum, mean, std, any, all 함수 사용법
ndarray 파일 입출력	ndarray를 파일로 저장하고 불러오는 방법
np.inf, np.nan	np.inf, np.nan에 대한 특징, 의미 학습

ndarray 생성 함수



▶ [예제3] np.array는 array_like를 사용하여 ndarray를 생성하는 함수이다

np.array(array_like, [dtype])

array_like 객체를 인수로 받아 배열 생성

[[1,2,3],[4,5,6]] → array_like, [[1,2,3], [4, 5]] → array_like 아님

Structured Array

- np.array(array_like, dtype) 을 사용하며, array_like를 여러 데이터의 tuple 리스트로 만듦
- 이 경우 Structured Datatype을 작성하여 dtype으로 지정함
- Structured Datatype은 데이터 tuple의 각 필드에 대해 이름과 자료형을 지정하여 작성함
(다양한 작성 방법이 있으며, <https://numpy.org/devdocs/user/basics.rec.html> 를 참조함)
- arr[필드이름] 또는 arr[index] 를 사용하여 indexing 할 수 있음

```
sdtype = [('name', 'U10'), ('height', '<i4'), ('weight', np.float32)]
value = [('Tom', 178, 98.5), ('Jim', 183, 79.5), ('Adam', 175, 82.8)]
a = np.array(value, dtype=sdtype)
```

10글자 문자열(1글자 4byte)

Little Endian, int, 4byte 의미 ('>' → Big Endian)

name	height	weight
Tom	178	98.5
Jim	183	79.5
Adam	175	82.8

ndarray 생성 예 1



[예제3]

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

```
[[1 2 3]  
 [4 5 6]]
```

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

```
object [list([1, 2, 3]) list([4, 5])]
```

```
sdtype = [('name', 'U10'), ('height', 'i4'), ('weight', np.float32)]  
value = [('Tom', 178, 98.5), ('Jim', 183, 79.5), ('Adam', 175, 82.8)]  
a = np.array(value, dtype=sdtype)
```

Structured Array

```
a
```

```
[('Tom', 178, 98.5) ('Jim', 183, 79.5) ('Adam', 175, 82.8)]
```

```
a.dtype
```

```
[('name', 'U10'), ('height', 'i4'), ('weight', 'f4')]
```

```
a['name']
```

```
['Tom' 'Jim' 'Adam']
```

```
a[1]
```

```
('Jim', 183, 79.5)
```

name	height	weight
Tom	178	98.5
Jim	183	79.5
Adam	175	82.8

ndarray 생성 함수 2/4



▶ [예제4] 값 범위(start, stop)을 사용하여 ndarray를 생성하는 함수들이다

- `np.arange([start,] stop [, step,] dtype=None)`

- `range`와 같은 원리로 배열 생성

- `np.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)`

- start부터 stop까지 동일 step의 데이터를 num개 갖는 배열 생성

- endpoint : stop까지 포함할 경우 True 포함하지 않을 경우 False

- retstep : True인 경우 ndarray, step의 tuple을 반환

```
a = np.arange(5)      [0 1 2 3 4]  
b = np.arange(1, 10, 2) [1 3 5 7 9]
```

```
a = np.linspace(1, 5, 9)  
b = np.linspace(1, 5, 10, endpoint=False)  
c = np.linspace(1, 5, 9, retstep=True)
```

```
[1. 1.5 2. 2.5 3. 3.5 4. 4.5 5. ]  
[1. 1.4 1.8 2.2 2.6 3. 3.4 3.8 4.2 4.6]  
(array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ]), 0.5)
```

linear space

ndarray 생성 함수 3/4



▶ [예제5] random을 사용하여 ndarray를 생성하는 함수들이다

함수	기능
<code>np.random.rand(d0, d1, ...dn)</code>	<ul style="list-style-type: none">▪ 0 ~ 1 사이의 균일 분포로 실수 난수 배열 생성▪ d0 ... dn 은 배열의 shape를 의미하는 정수
<code>np.random.randint(e, size=n)</code> <code>np.random.randint(s, e, size=n)</code>	<ul style="list-style-type: none">▪ 0 ~ e-1 또는 s ~ e-1 값을 갖는 균일 분포의 정수 난수 생성▪ size : 정수(1차원) 또는 tuple(2차원 이상)로 shape 지정
<code>np.random.randn(d0, d1, ...dn)</code>	<ul style="list-style-type: none">▪ 평균 0, 표준편차 1을 갖는 가우시안 표준 정규 분포로 난수 배열 생성 (음수도 포함됨)
<code>np.random.normal(평균, 표준편차, size)</code>	<ul style="list-style-type: none">▪ 평균과 표준편차를 지정하여 size 개의 난수로 배열 생성▪ randn()과 normal()은 정규 분포를 갖음
<code>np.random.permutation(e)</code> <code>np.random.permutation(c)</code>	<ul style="list-style-type: none">▪ e 는 정수, c 는 array_like 객체▪ 0 ~ e-1 값을 갖는 정수의 무작위 순서 1차원 배열 생성▪ c 요소들에 대해 무작위 순서로 변경된 1차원 배열 생성▪ c 가 다차원인 경우 가장 상위 차원만 무작위 순서로 변경 됨

random 참조 : <https://docs.scipy.org/doc/numpy/reference/routines.random.html>

ndarray 생성 예 2



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

```
[0.9088 0.8697 0.536 0.7954 0.2754]
```

```
np.random.randint(50, 100, (2,4))
```

```
[[95 74 65 93]
 [53 70 74 61]]
```

```
np.random.randn(200, 50)
```

```
[[ -0.9049 -1.11 ... -0.168 -0.6898]
 ...
 [-1.1395 0.0986 ... 0.9346 -1.0711]]
 mean : 0.013, std : 0.993
```

```
np.random.normal(2, 3, (100, 20))
```

```
[[ 1.6929 2.1374 ... -0.6581 0.5421]
 ...
 [ 2.7684 1.4454 ... 2.5391 -1.1127]]
 mean : 2.030, std : 3.040
```

```
np.random.permutation(10)
```

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

```
data = [1,3,4,5,3,1,3]
np.random.permutation(data)
```

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

np.set_printoptions 를 사용한 배열 출력 옵션 설정 (보기 좋은 배열 출력)

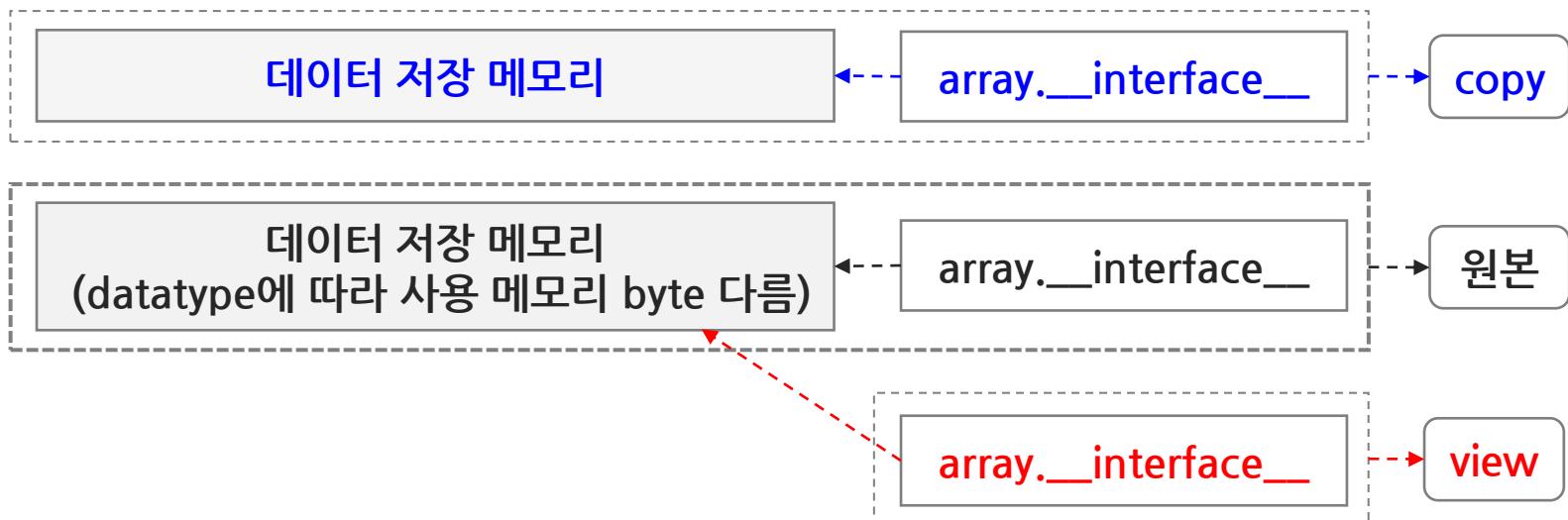
https://numpy.org/doc/stable/reference/generated/numpy.set_printoptions.html

copy & view 객체



▶ [예제6] ndarray의 copy 객체 또는 view 객체의 특징을 살펴보다

copy 객체	view 객체
<ul style="list-style-type: none">▪ arr.copy()를 사용하여 생성▪ 원본과 다른 메모리에 데이터 복사	<ul style="list-style-type: none">▪ arr.view()를 사용하여 생성▪ 원본 배열과 같은 데이터 참조



ndarray 배열 변환/조작 함수 -1/3



▶ [예제7] ndarray 를 다른 속성의 ndarray로 변환/조작하는 함수들이다

변환 함수	동작
<code>a.reshape(shape, order='C')</code> <code>np.reshape(a, shape, order='C')</code>	<ul style="list-style-type: none">▪ 배열의 ‘shape’을 변환하여 반환함▪ arr와 shape의 size는 동일해야 함▪ shape의 요소 중 1개를 -1로 하여 자동 계산 가능▪ order='F'로 지정하면 copy 객체가 반환 됨
<code>a.astype(dtype, order='K')</code>	<ul style="list-style-type: none">▪ 배열의 데이터 ‘타입’을 변환하여 반환함

```
a = np.array([[1,2,3],[4,5,6]], dtype=np.int32)
b = a.astype(np.int64)
c = a.flatten()
for n, x in zip("abc", (a,b,c)):
    npinfo(n, a, x)

name      : a
share     : True
data      : [[1, 2, 3], [4, 5, 6]]
shape     : (2, 3)
stride   : (12, 4)

name      : b
share     : False
data      : [[1, 2, 3], [4, 5, 6]]
shape     : (2, 3)
stride   : (24, 8)

name      : c
share     : False
data      : [1, 2, 3, 4, 5, 6]
shape     : (6,)
stride   : (4,)
```

배열의 shape 변경 - reshape



[예제8] reshape을 사용하면 배열의 shape을 변경할 수 있다

```
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(1, -1)
c = a.reshape(-1)
d = a.reshape(-1, 1)
e = a.reshape(-1, 1, order='F')
```

```
np.may_share_memory(a, x)
→ 두 배열이 view 관계인 경우 True 반환
x.tolist()
→ ndarray x를 list 객체로 변환하여 반환
```

```
name : b
share : True
data : [[1, 2, 3, 4, 5, 6]]
shape : (1, 6)
```

```
name : d
share : True
data : [[1], [2], [3], [4], [5], [6]]
shape : (6, 1)
```

```
name : c
share : True
data : [1, 2, 3, 4, 5, 6]
shape : (6, )
```

```
name : e
share : False
data : [[1], [4], [2], [5], [3], [6]]
shape : (6, 1)
```

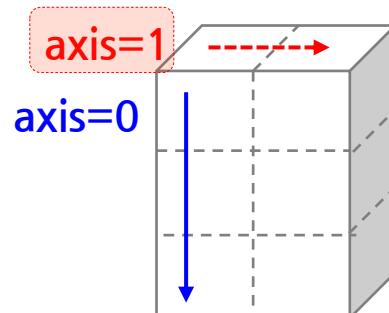
axes, axis의 이해



- ndarray에 사용되는 axes 또는 axis 는 ‘축’을 의미한다
- shape 표기 법에 따라 앞에 표기되는 축이 0번 그 뒤로 1씩 증가한 번호가 된다
 - 축 번호를 음수로 사용할 수 있으며, 뒤에서부터 -1, 그 앞으로 1씩 감소한 번호가 된다

0	1
2	3
4	5

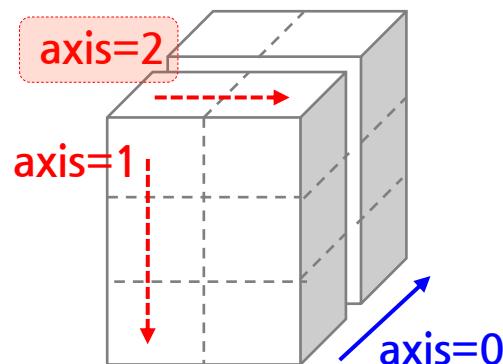
ndim : 2, shape : (3, 2)



axis = -1 과 동일

0	1	6	7
2	3	8	9
4	5	10	11

ndim : 3, shape : (2, 3, 2)



numpy의 산술 연산



▶ [예제9] numpy의 산술 연산을 살펴보자

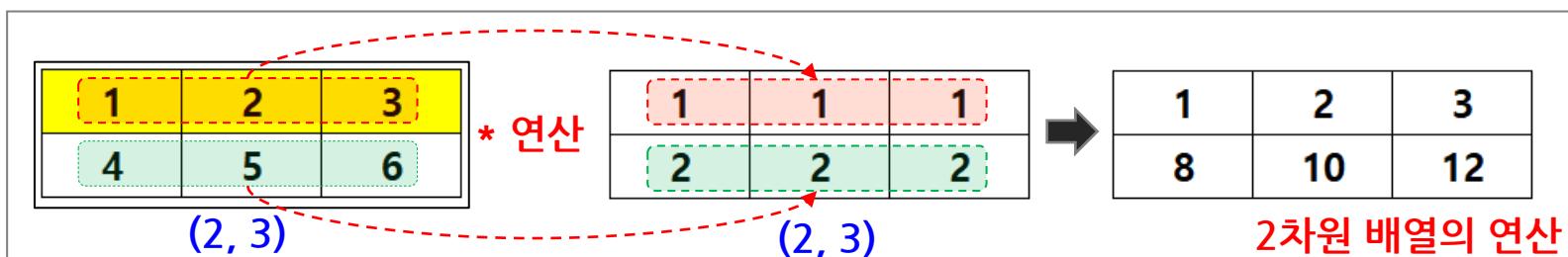
연산자	연산	결과
+	a + b	[3 9 17]
-	b - a	[1 3 7]
*	a * b	[2 18 60]
/	b / a	[2. 2. 2.4]

- a = np.array([1, 3, 5])
- b = np.array([2, 6, 12])

**, //, % 연산도 가능하다

element-wise

- 같은 shape을 갖는 두 배열 간의 연산을 말하며, numpy 연산의 기본이다
- 같은 차원의 배열에서 각 차원을 구성하는 요소 수가 다르면 오류가 발생한다



broadcasting의 이해 -1/2



- ▶ numpy의 배열 연산은 element-wise 방식으로 이루어진다
- ▶ 다른 차원의 두 배열 연산 시 broadcasting을 통해 배열 전환 후 연산이 수행된다
- ▶ broadcasting의 배열 전환 원리는 다음과 같다
 - 한 배열 shape이 다른 배열 shape의 부분 shape이면 차원이 높은 배열 shape을 따른다
 - $(2, 3, 4) + (3, 4) \rightarrow (2, 3, 4)$ $(4, 2) + (2,) \rightarrow (4, 2)$ $(2, 4) + (3, 1, 2, 4) \rightarrow (3, 1, 2, 4)$
 - 요소 개수가 1인 차원에 대해서는 다른 배열의 동일 차원을 요소 수를 따른다
 - $(5, 1) + (3,) \rightarrow (5, 3)$ $(5, 1, 4) + (3, 1) \rightarrow (5, 3, 4)$ $(4, 2, 1) + (1, 3) \rightarrow (4, 2, 3)$
 - 스칼라 값은 상대 배열의 shape과 같은 shape, 동일 값으로 구성된 배열이 된다
 - $(2, 3) + 2 \rightarrow (2, 3)$

1	2	3
4	5	6

A 배열 : (2, 3)

2



스칼라 값 : 2

broadcasting

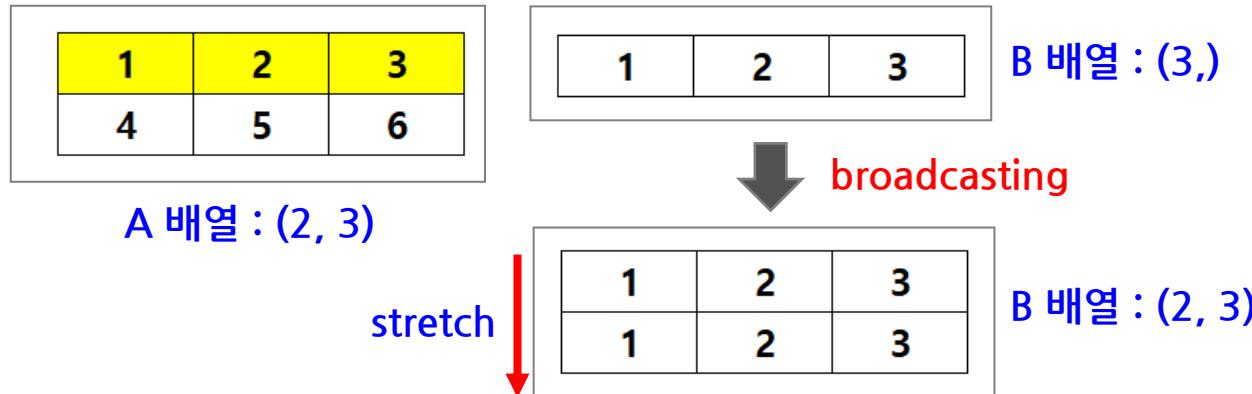
2	2	2
2	2	2

B 배열 : (2, 3)

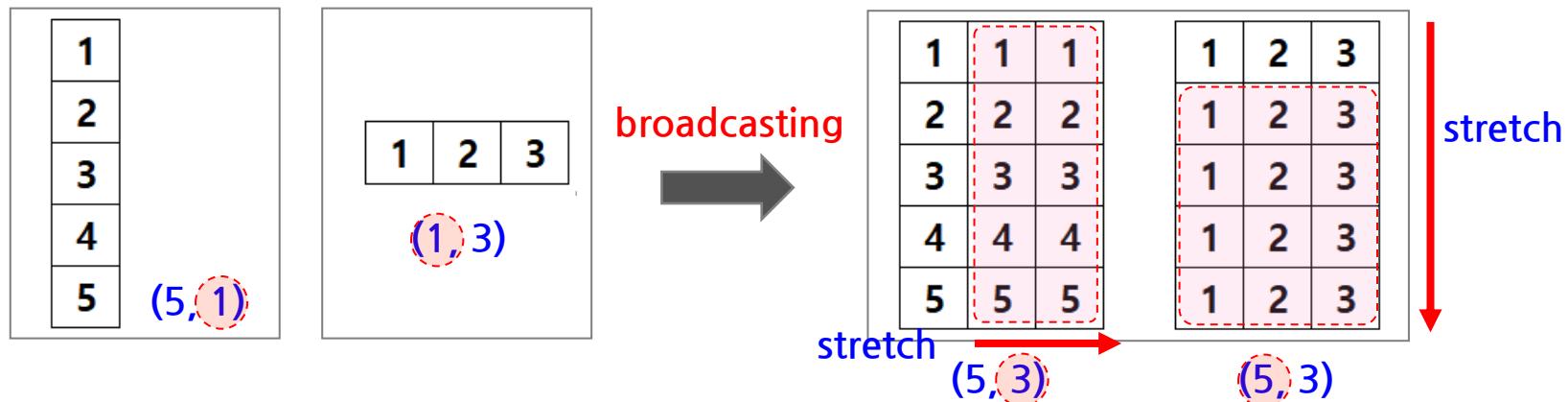
broadcasting의 이해 -2/2



- 두 개 배열의 shape이 같도록 broadcast 하여 연산한다
- 부분 배열인 경우는 큰 쪽에 맞추어 확장 된다



- 요소 개수가 1인 axis는 상대 배열의 동일 axis의 요소 개수만큼 확장 된다



numpy의 broadcasting 연산



▶ [예제10] 배열 연산 결과를 분석하여 broadcasting을 이해하여 보자

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

종류	차원 계산	연산	결과
element-wise	(2, 3) + (2, 3) : (2, 3)	a + b	[[2 2 2] [3 3 3]]
broadcasting	(2, 3) + (3,) : (2, 3)	a + d	[[4 4 4] [4 4 4]]
	(2, 3) + 스칼라 : (2, 3)	b + 4	[[5 5 5] [6 6 6]]
	(3, 1) + (3,) : (3, 3)	c + d	[[4 4 4] [5 5 5] [6 6 6]]

1	2	3
---	---	---

4	5
---	---

A 배열 : (3,)

B 배열 : (2,)

ndarray의 비교 연산



▶ [예제11] 코드를 실행하여 비교 연산을 이해하여 보자

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

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

```
c=np.array([4, 3, 3])
```

차원 계산	연산	결과
$(2, 3) == (2, 3) : (2, 3)$	$a == b$	$\begin{bmatrix} \text{[[True False False]} \\ \text{[False False True]]} \end{bmatrix}$
$(2, 3) != (2, 3) : (2, 3)$	$a != b$	$\begin{bmatrix} \text{[[False True True]} \\ \text{[True True False]]} \end{bmatrix}$
$(2, 3) > (2, 3) : (2, 3)$	$a > b$	$\begin{bmatrix} \text{[[False False False]} \\ \text{[True True False]]} \end{bmatrix}$
$(2, 3) < (2, 3) : (2, 3)$	$a < b$	$\begin{bmatrix} \text{[[False True True]} \\ \text{[False False False]]} \end{bmatrix}$
$(2, 3) \geq (3,) : (2, 3)$	$a \geq c$	$\begin{bmatrix} \text{[[False False True]} \\ \text{[True True True]]} \end{bmatrix}$
$(2, 3) \leq \text{스칼라} : (2, 3)$	$b \leq 3$	$\begin{bmatrix} \text{[[True True False]} \\ \text{[True False False]]} \end{bmatrix}$

indexing



▶ [예제12] indexing을 사용하여 배열에서 원하는 데이터를 참조/변경한다

▶ 배열 이름 뒤에 [] 를 사용하여 참조/변경 할 데이터를 표기한다

- 결과는 원본의 view이며, view에 대입을 통해 데이터 변경이 가능함
- ndim >= 2 인 경우 [] 내부에 콤마(,)를 사용하여 차원(축, axes, axis)을 구분함
- 콤마(,) 사이에는 참조/변경하기 원하는 데이터에 대한 표기(indexer)가 포함되어야 함
- 다양한 indexer가 존재함
- axis=0을 제외한 차원에 대한 표기 생략가능, 생략은 “모두 선택”의 의미를 갖음

```
a = np.arange(9).reshape(3,3)  
  
print(a)  
print(a[0, 2])  
print(a[1])  
print(a[2][1])
```

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

- <https://docs.scipy.org/doc/numpy/reference/arrays.indexing.html>
- <https://www.numpy.org/devdocs/user/basics.indexing.html>

indexer의 종류



▶ [예제13] ndarray에서 사용할 수 있는 indexer의 종류는 다음과 같다

indexer	설명
single element index	▪ integer object
slice and stride	▪ slice object ➔ start:stop:step
index arrays	▪ sequence-like, numpy object
Boolean arrays	▪ boolean array : the same shape with original array
structural indexing tools	▪ ..., np.newaxis
index + slice	▪ combine index and slice

```
a = np.arange(5)

print(a[1])
print(a[1:4:2])
print(a[[0,1,-1]])
print(a[[True, False, False, True, False]])
print(a[:, np.newaxis])
```

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

Indexing의 반환 값



- ▶ [예제14] 아래의 코드 실행 결과를 예측한 뒤 실행하여 결과를 확인하라
- ▶ ndarray의 indexing의 반환 값은 원본을 참조하는 view이다
 - 새로운 ndarray로 처리하려면 copy 함수나 copy 메소드를 사용해야 한다

```
# list of python
x = [[1, 2, 3], [6, 7, 8]]
z = x[:2]
z[0] = [10, 20, 30]
print(x, z, sep='\n')
print("-"*25)

# ndarray of numpy
x = np.array([[1, 2, 3], [6, 7, 8]])
z = x[:2]
z[0] = [10, 20, 30]
print(x, z, sep='\n')
print(np.may_share_memory(x,z))
```

```
[[1, 2, 3], [6, 7, 8]]
[[10, 20, 30], [6, 7, 8]]
-----
[[10 20 30]
 [ 6  7  8]]
[[10 20 30]
 [ 6  7  8]]
True
```

x, z의 원본 데이터가 같은 객체인지 반환
True인 경우 원본 데이터가 같은 객체임

Indexing - single element index



▶ [예제15] single element index를 사용한 indexing을 살펴보자

방법	설명
a[n] (1D array)	<ul style="list-style-type: none">n번 index 요소를 반환 (index는 0부터 시작하는 일련번호, 음수가 능)
a[n, m, ...] (N D array)	<ul style="list-style-type: none">axis=0부터 뒤쪽으로 콤마(,)를 기준으로 index를 나열함index 표기되지 않은 차원은 “모두 선택” 예) 2차원 배열에서 a[n]은 n번 행 전체를 반환 함indexing을 연속으로 진행한 것과 콤마(,) 나열은 같은 결과 예) $a[n][m] == a[n, m]$

<table border="1"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td></tr></table>	0	1	2	3	4	5	<table border="1"><tr><td>6</td><td>7</td></tr><tr><td>8</td><td>9</td></tr><tr><td>10</td><td>11</td></tr></table>	6	7	8	9	10	11
0	1												
2	3												
4	5												
6	7												
8	9												
10	11												
$c : (2, 3, 2)$													

<table border="1"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td></tr></table>	0	1	2	3	4	5	<table border="1"><tr><td>8</td><td>9</td></tr></table>	8	9	7	$c[1, 1]$	$c[1, 0, 1]$
0	1											
2	3											
4	5											
8	9											
$\cancel{\times} \ 3, 2) \rightarrow (3, 2) \quad (\cancel{\times} \cancel{\times} \ 2) \rightarrow (2,) \quad (\cancel{\times} \cancel{\times} \cancel{\times}) \rightarrow ()$												

single element index의 사용이 1개 증가 할 때마다 배열의 차원(ndim)은 1씩 감소한다

Indexing - slice & stride



▶ [예제16] slice & stride를 사용한 indexing을 살펴보자

n, m, k 는 정수이며, 생략 가능, 음수 일 수 있으며, -1은 마지막 요소의 번호

방법	설명
a[n:m:k] (1D array)	<ul style="list-style-type: none">n 부터 m-1 까지, k 씩 건너 뛴 위치 요소로 구성된 배열 반환
a[n1:m1:k1, n2:m2:k2, ...] (N D array)	<ul style="list-style-type: none">axis=0부터 뒤쪽으로 콤마(,)를 기준으로 index를 나열함index 표기되지 않은 차원은 “모두 선택”모든 정수가 생략된 index에 대해서는 “모두 선택”을 적용함<ul style="list-style-type: none">→ 특정 차원에 대해 모든 범위를 의미함→ 나머지 모든 차원에 대해 모든 범위를 의미함

<table border="1"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td></tr></table>	0	1	2	3	4	5	<table border="1"><tr><td>6</td><td>7</td></tr><tr><td>8</td><td>9</td></tr><tr><td>10</td><td>11</td></tr></table>	6	7	8	9	10	11
0	1												
2	3												
4	5												
6	7												
8	9												
10	11												
$c : (2, 3, 2)$													

차원(ndim) 변경 없음

<table border="1"><tr><td>6</td><td>7</td></tr><tr><td>8</td><td>9</td></tr><tr><td>10</td><td>11</td></tr></table>	6	7	8	9	10	11	<table border="1"><tr><td>7</td><td>6</td></tr><tr><td>1</td><td>0</td></tr><tr><td>3</td><td>2</td></tr><tr><td>5</td><td>4</td></tr></table>	7	6	1	0	3	2	5	4	<table border="1"><tr><td>10</td><td>11</td></tr><tr><td>4</td><td>5</td></tr><tr><td>2</td><td>3</td></tr><tr><td>0</td><td>1</td></tr></table>	10	11	4	5	2	3	0	1
6	7																							
8	9																							
10	11																							
7	6																							
1	0																							
3	2																							
5	4																							
10	11																							
4	5																							
2	3																							
0	1																							
$c[1:] (1, 3, 2)$	$c[...,-1] (2, 3, 2)$	$c[:, -1] (2, 3, 2)$																						

Indexing - index_array



▶ [예제17] index_array를 사용하여 원하는 항목을 정교하게 표현할 수 있다

방법	설명
a[index_array] (1D array)	<ul style="list-style-type: none">index_array에 포함된 index 항목들로 구성된 배열 반환a[[0, 2, 4]] 는 a에서 0,2,4 번 index 항목으로 구성된 배열 반환index_array 사용시 주의 사항<ul style="list-style-type: none">- index는 중복해서 사용할 수 있음- 2차원 이상의 index_array 는 ndarray를 사용([[[➔ 사용 불가능)
a[idx_a1, idx_a2, ...] (N D array)	<ul style="list-style-type: none">idx_a1, idx_a2의 같은 위치 항들이 묶어 위치 정보가 됨다차원이 되면 콤마(,)로 나열하며 동일 길이로 구성되어야 함a가 2D array인 경우<ul style="list-style-type: none">- a[[0,0,1],[2,3,1]] 은 [0,2], [0,3], [1,1] 항을 배열로 반환- a[[0, 1], 0] : a[[0,1], [0,0]] (broadcasting) [0, 0], [1, 0] 항 반환

x

1	2	3
---	---	---

 x = np.array([1, 2, 3])
single element index ← print(x[1], x[[1]])
print(x[np.array([[0,0], [2,1]])])
print(x[[2, 0, 1, 1]])

2 [2]
[[1 1]]
[3 2]]
[3 1 2 2]

index_array의 형태에 따라 ndim이 동일하거나, 증가, 감소 될 수 있다

Indexing - boolean index 1/2



▶ [예제18] Boolean 배열을 사용한 indexing을 살펴보자

방법	설명
<code>a[boolean_index]</code>	<ul style="list-style-type: none">boolean_index는 배열과 동일한 shape의 Boolean 배열임boolean_index는 ndarray, list 등으로 작성, 연산 결과 일 수 있음주로 결과가 True/False인 비교연산식을 사용하여 작성함복잡한 조건은 np.logical_and(), np.logical_or(), np.logical_not() 활용Boolean 배열과 &, , ~ 연산자를 사용하여 조건작성 가능 (괄호 사용 중요)

a	1	2	3	4	5	6
b	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
a[b]	1	3	4	6		

Boolean 배열

a	1	2	3	4	5	6
<code>a%2==0</code>	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
<code>a[a%2==0]</code>	2	4	6			

대상의 차원에 상관없이
결과 배열은 1차원이다

Indexing - boolean index 2/2



▶ [예제21] 논리형 배열을 기반으로 인덱싱을 할 수 있다

boolean indexing	의미
<code>a[a%2==0]</code>	2의 배수인 값
<code>a[a>=4]</code>	4보다 크거나 같은 값
<code>a[np.logical_or(a%3==0, a%2 == 0)]</code>	3의 배수 또는 2의 배수 인 것
<code>a[np.logical_and(a>=6, a<=9)]</code>	6~9 사이의 값
<code>a[np.logical_not(a%3 == 0)]</code>	3의 배수가 아닌 것
<code>a[a%3==0] = -1</code>	

마지막 동작의 의미를 적어 보도록 한다

Indexing 활용



▶ [예제22] 다음은 indexing 활용에 대한 예시이다

aN3 은 (N,3) shape, aN 은 (N,) shape, a 는 임의 shape의 배열이다

indexing	의미
1 aN3[:, [1, 0, 2]]	0번과 1번 열을 바꿈
2 a3N[[0, 2, 1], :]	1번과 2번 행을 바꿈
3 aN[:, None] == aN[:, np.newaxis]	aN.reshape(-1, 1) 과 같은 결과
4 aN3[aN3[:, 0]<0]	0번째 열의 값이 0보다 작은 행
5 aN3[aN3[:,0]<0, 1:3]	0번째 열의 값이 0보다 작은 행의 1, 2번 열

<code>[[0 1 2] [0 1 2] [0 1 2] [0 1 2]]</code>	<code>[[0 0 0] [1 1 1] [2 2 2]]</code>	<code>[0 1 2]</code>	<code>[[-1 0 0] [1 1 1] [-5 2 2] [5 3 3]]</code>	<code>[[-1 0 0] [-5 2 2]]</code>
<code>[[1 0 2] [1 0 2] [1 0 2] [1 0 2]]</code>	<code>[[0 0 0] [2 2 2] [1 1 1]]</code>	<code>[[0] [1] [2]]</code>		<code>[[0 0] [2 2]]</code>

numpy의 유용한 함수 - 1



▶ [예제23] numpy의 유용한 함수들은 다음과 같다 (a는 ndarray 객체)

함수	기능
<code>np.mean(a, axis=None)</code>	▪ 배열내 모든 원소의 평균을 스칼라 값으로 반환 (float64)
<code>np.sum(a, axis=None)</code>	▪ 배열내 모든 원소의 합을 스칼라 값으로 반환
<code>np.std(a, axis=None)</code>	▪ 배열내 모든 원소의 표준 편차를 스칼라 값으로 반환
<code>np.any(a, axis=None)</code>	▪ 배열의 원소들 중 참이 하나라도 있으면 True 반환 ▪ 0, False, None → 거짓
<code>np.all(a, axis=None)</code>	▪ 배열의 모든 원소가 참인 경우 True 반환

- `axis=0`은 열, `axis=1`은 행에 대한 연산 결과를 배열로 반환
- ndarray에도 동일 동작을 하는 메서드가 있음 (`arr.mean()`, `arr.sum()` 등)

`np.sum(a, axis = 0)`

1	2	3
4	5	6

[5 7 9]

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

1	2	3
4	5	6

[6 15]

numpy의 유용한 함수 - 2



▶ [예제24] 조건에 따른 배열 반환 함수

• np.where(조건, [,c, c])

- 조건 : 참, 거짓(=0, False, None)으로 이루어진 배열 또는 스칼라
- 조건 배열이 참일 때 c, 거짓일 때 d를 취한 배열 반환 (c, d→스칼라 가능)
- c, d 생략 시 조건이 참인 것에 대한 index 배열(ndarray) tuple 반환
(ndarray, ndarray, ...)의 형태, 조건 배열의 ndim == len(tuple)
- 복잡한 조건은 ~, &, |, 및 np.logical_and(), np.logical_or() 사용
→ boolean index 참조

```
a = np.array([4, 8, 2, 5])
b = np.array([3, 9, 1, 7])
print(np.where(True, a, b))
print(np.where(False, a, b))
print(np.where(a<5, a, 10*b))
print(np.where([[True, False], [False, True]], 
              [[3, 4], [5, 7]],
              [[1, 2], [9, 2]]))
```

```
[4 8 2 5]
[3 9 1 7]
[ 4 90  2 70]
[[3 2]
 [9 7]]
```

배열 파일 입출력



▶ numpy의 배열을 파일로 저장하고 불러오는 함수는 다음과 같다

함수	기능
np.save()	<ul style="list-style-type: none">▪ 1개 배열을 npy 파일로 저장 (NumPy format, binary file)
np.savez()	<ul style="list-style-type: none">▪ 여러 개 배열을 압축 안 된 npz 파일로 저장▪ 각 배열에 이름을 부여하여 저장
np.save_compressed()	<ul style="list-style-type: none">▪ 압축 된 npz 파일로 저장, 각 배열에 이름을 부여하여 저장
np.load()	<ul style="list-style-type: none">▪ save, savez, save_compressed로 저장된 npy, npz 파일 불러오기▪ npz를 읽어온 경우, 이름을 이용하여 indexing▪ npz type : numpy.lib.npyio.NpzFile, close() 할것
np.savetxt()	<ul style="list-style-type: none">▪ 1D/2D array_like를 텍스트 파일로 저장▪ delimiter, newline, encoding, fmt 등 지정 가능 (fmt='%.1.2f')
np.loadtxt()	<ul style="list-style-type: none">▪ Text 파일 불러오기, delimiter, dtype 등 지정 가능

- 각 함수의 자세한 사용 법은 필요시 Numpy API를 참조하며, 예제를 사용하여 알아보자

배열 파일 입출력 예제



▶ [예제25] 다음 배열 파일 입출력 예시를 실행하고 파일을 확인해 보자

```
np.save(path+"a1.npy", a1)
r = np.load(path+"a1.npy")
printary(r)
```

```
np.savez(path+"an.npz", A1=a1, A2=a2, A3=a3)
r = np.load(path+"an.npz")
printary(r['A1'], r['A2'], r['A3'])
```

```
np.savez_compressed(path+"c_an.npz", A1=a1, A2=a2, A3=a3)
r = np.load(path+"c_an.npz")
printary(r['A1'], r['A2'], r['A3'])
```

```
np.savetxt(path+"a1.txt", a1, fmt='%d')
r = np.loadtxt(path+"a1.txt", dtype=np.int32)
printary(r)
```

	a1.npy	79KB
	a1.txt	126KB
	an.npz	107KB
	c_an.npz	40KB

np.nan, np.inf



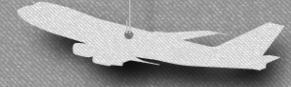
▶ [예제26] 다음 numpy의 version, nan, inf에 대한 속성과 함수를 살펴보자

속성, 함수	설명
np.__version__	▪ numpy의 버전
np.nan	▪ not a number, ‘nan’으로 출력됨
np.inf	▪ infinite, ‘inf’으로 출력됨
np.isnan()	▪ np.nan 값을 갖는 항을 True, 아닌 항을 False로 하여 반환
np.isinf()	▪ np.inf 값을 갖는 항을 True, 아닌 항을 False로 하여 반환

```
print(np.__version__)
print(np.nan, type(np.nan))
print(np.inf, type(np.inf))

a = np.array([np.nan, 1, 0, -1, np.inf], dtype=np.float16)
print(np.isnan(np.nan), np.isnan(a))
print(np.isinf(np.inf), np.isinf(a))
```

- nan, inf가 포함된 경우 연산 시 원치 않는 상황이 발생할 수 있으므로 주의 해야함



dot, matmul 은
Deep Learning에서 사용 (선택학습)



numpy의 유용한 함수 - 2



▶ [예제1] 배열의 곱과 관련한 numpy의 함수이다

함수	기능
np.matmul(a, b) a @ b	▪ a와 b의 'Matrix product'을 구해 반환, a, b : 스칼라 사용할 수 없음 ▪ a가 ndarray인 경우 @ 연산자를 사용할 수 있음
np.dot(a, b) a.dot(b)	▪ a와 b의 'Dot(Inner) product'을 구해 반환, a, b : 스칼라 사용할 수 있음 ▪ 많은 경우 matmul과 동일한 동작을 함

matmul, dot 같은 동작

matmul, dot 다른 동작

a, b 모두 1-D

a 1-D, b N-D

a, b 모두 2-D

a N-D, b 1-D

a, b 모두 N-D ($N > 2$)
단, axis = -1, -2 이외 axis는
같은 요소 개수

a N-D, b M-D ($M > 2$)

a, b 중 한 쪽이 스칼라

- 곱 연산에서 a는 앞, b는 뒤에 위치한 배열
- a의 axis = -1과 b의 axis = -2의 각 요소 곱의 합을 구하는 것
- a의 axis = -1과 b의 axis = -2의 요소 수가 같아야 함
- b가 1-D 인 경우는 b의 axis = -1의 요소가 곱의 대상이 됨

np.matmul, np.dot 함수의 연산 규칙



matmul dot
같은 동작

a, b 모두 1-D

▪ sum($a * b$) \rightarrow c 는 scalar # sum($a[:] * b[:]$)

a, b 모두 2-D

▪ sum($a[i, :] * b[:, n]$) \rightarrow c[i, n]

a 1-D, b N-D

▪ sum($a[:] * b[n, :, m]$) \rightarrow c[n, m]

a N-D, b 1-D

▪ sum($a[i, j, :] * b[:]$) \rightarrow c[i, j]

a, b 모두 N-D
(N>2)

▪ axis -1, -2 이외의 같은 axis는 서로 같은 요소 개수

sum($a[i, j, :] * b[i, :, m]$) \rightarrow c[i, j, m]

sum($a[i, j, k, :] * b[i, j, :, m]$) \rightarrow c[i, j, k, m]

matmul dot
다른 동작

a N-D, b M-D(M>=2)

▪ dot : sum($a[i, j, :] * b[k, :, m]$) \rightarrow c[i, j, k, m]

▪ dot : sum($a[k, :] * b[i, :, m]$) \rightarrow c[k, i, m]

▪ matmul : sum($a[k, :] * b[i, :, m]$) \rightarrow c[i, k, m]

a, b 중 한 쪽이 scalar

▪ matmul : 제공하지 않음, dot : multiply를 구함

▪ np.multiply() 및 * 사용이 권장됨

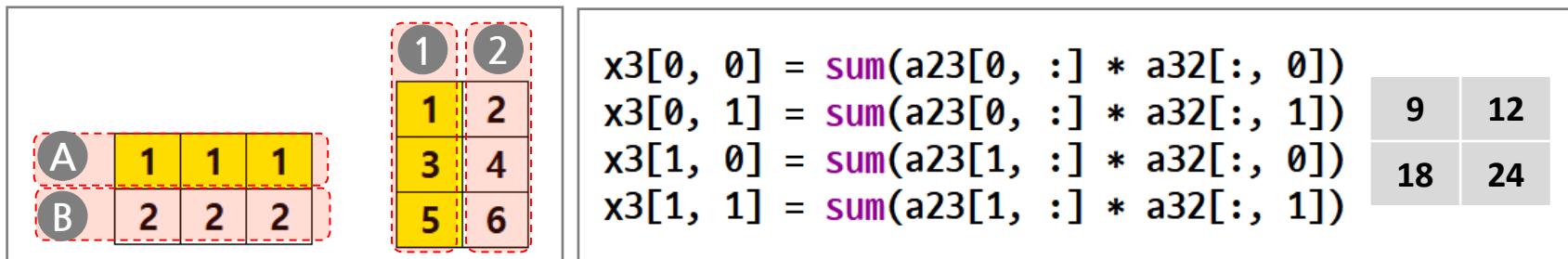
numpy의 유용한 함수 - 2



☞ dot와 matmul의 공통 연산 방법을 알아보자

☞ dot와 matmul은 행렬 연산이므로 “앞 배열의 열, 뒤 배열의 행”수가 같아야 한다

- (2, 3) 앞 배열과 (3, 2) 뒤 배열 연산 결과 (2, 2) shape의 배열이 생성된다



다음 배열 연산에 대한 그림을 그리고 연산 결과를 예측하여 보자

- A : (2,3), B : (3,2)일 때 np.dot(B, A) 및 np.matmul(B, A)가 가능한가?
- 가능하다면 결과의 shape은 무엇인가?

np.matmul, np.dot 공통



a, b 모두 1-D

[1 2 3]

[1 2 3]

14

a, b 모두 2-D

[[1 2]
[3 4]
[5 6]]

[[1 1]
[2 2]]

[[5 5]
[11 11]
[17 17]]

a 1-D, b N-D

[1 2 3]

[[1]
[2]
[3]]

[14]

[1 2 3]

[[[1 2]
[3 4]
[5 6]]]

[[22 28]
[9 11]]

[[1 1]
[1 2]
[2 2]]]

a N-D, b 1-D

[[1 1 1]
[2 2 2]]

[1 2 3]

[6 12]

[[[1 2 3]
[4 5 6]]
[[1 1 1]
[2 2 2]]]

[1 2 3]

[[14 32]
[6 12]]

matmul 연산의 이해 (3차원이상 포함)



▶ [예제2] 3차원 이상에서 matmul 연산 원리는 다음과 같다

● 대응하는 열/행을 제외한 나머지는 배열에서의 원래 index 위치를 유지한다

↖ 뒤 배열은 앞 배열과 ‘행’ 전까지의 요소 개수 및 ‘열’의 개수가 같아야 한다

↖ 단, 뒤 배열이 1차원인 경우 배열 크기를 행의 수로 취급한다

✗ (2, 2, 3)과 (2, 3, 3) ➔ (2, 2, 3)

✗ (2, 2, 3)과 (3, 3, 2) ➔ 불가능 (axis = 0 의 크기가 다름)

✗ (2, 2, 3)과 (3, 3) ➔ dot 와 동일

✗ (2, 2, 3)과 (3, 1) ➔

✗ (2, 2, 3)과 (3,) ➔

✗ (2, 3)과 (2, 3, 2) ➔ dot 와 shape은 동일하지만 결과는 다름

✗ (2, 3)과 (3, 2) ➔ dot 와 동일

✗ (2, 3)과 (3,) ➔

✗ (3,)과 (3, 1) ➔

배열에서의 원래 index 를 유지한다

(2, 2, 3) (2, 3, 3) (2, 3) (2, 3, 2)
(면, 행) (면, 열) (행) (면 열)

- A : (2, 4, 2, 3), B : (4, 3, 2)일 때 np.dot(A, B) 및 np.matmul(A, B)가 가능한가?
- 가능하다면 결과의 shape은 무엇인가?

matmul 연산의 예



[예제3] matmul의 연산에서의 제약을 살펴보자

- matmul은 스칼라 값과 연산 불가능 (dot은 가능)
- matmul은 axis -1, -2 이외의 같은 axis는 서로 같은 요소 개수 이어야 함

A	1	2	3
B	4	5	6
C	1	1	1
D	2	2	2

shape : [2, 2, 3]

면, 행 위치

shape : [2, 2, 2]

1	2
3	4
5	6
3	1
1	2
2	2

shape : [2, 3, 2]

면 위치 열 위치

A*1	A*2	C*3	C*4
B*1	B*2	D*3	D*4

A	0,0
B	0,1
C	1,0
D	1,1

1	0,0
2	0,1
3	1,0
4	1,1

- A*1 → 0, 0, 0
- B*2 → 0, 1, 1

dot 연산의 이해



- ▶ [예제4] 3차원 이상에서 dot(=inner product) 연산 원리는 다음과 같다
- ▶ 대응하는 열/행을 삭제하고 앞 배열에서 뒤 배열 쪽으로 남은 index를 나열한다

- 앞 배열의 열과 뒤 배열의 행의 수가 같은 경우 연산이 가능하다
- 단, 뒤 배열이 1차원인 경우 배열 크기를 행의 수로 취급한다

- (2, 2, 3)과 (2, 3, 3) → (2, 2, 2, 3)
- (2, 2, 3)과 (3, 3, 2) →
- (2, 2, 3)과 (3, 3) →
- (2, 2, 3)과 (3, 1) →
- (2, 2, 3)과 (3,) →
- (2, 3)과 (2, 3, 2) →
- (2, 3)과 (3, 2) →
- (2, 3)과 (3,) →
- (3,)과 (3, 1) →

대응하는 열/행을 제외한 나머지 index는
앞에서 뒤쪽으로 axis=0, 1, ... 로 사용된다

- 배열과 스칼라 값의 연산은 broadcast 연산이 된다
- (3, 2)와 2 → (3, 2)

dot 연산의 예



▶ [예제5] dot은 matmul과 다음과 같은 차이가 있다

A	1	2	3
B	4	5	6
C	1	1	1
D	2	2	2

shape : [2, 2, 3]

공간, 면 위치

1	2
3	4
5	6
3	4
1	1
1	2
2	2

shape : [2, 3, 2]

행, 열 위치

A*1	A*2	B*1	B*2
A*3	A*4	B*3	B*4

C*1	C*2	D*1	D*2
C*3	C*4	D*3	D*4

A	0,0
B	0,1
C	1,0
D	1,1

1	0,0
2	0,1
3	1,0
4	1,1

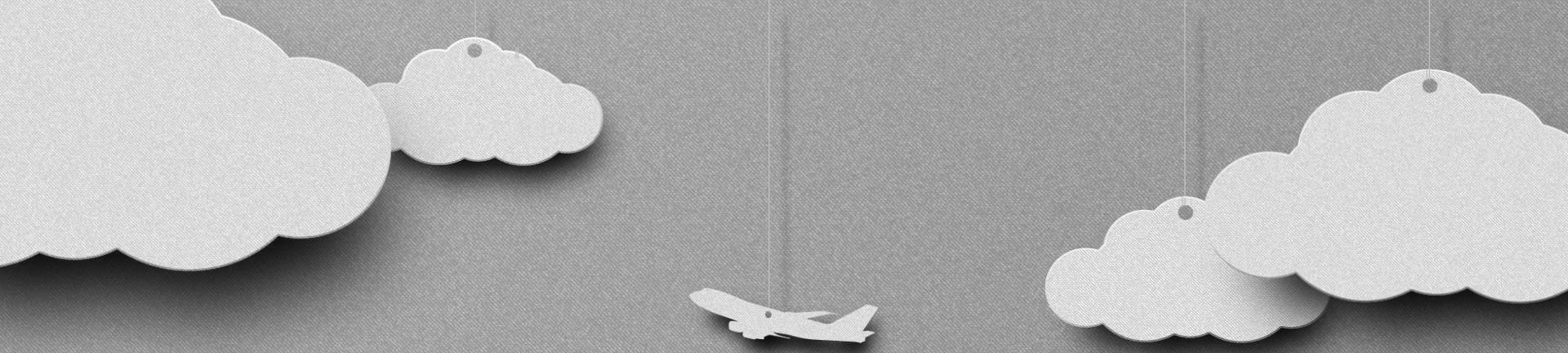
- A*1 → 0, 0, 0, 0
- B*2 → 0, 1, 0, 1

dot, matmul 연산 이해



- ▣ 다음 두 배열에 대해 dot과 matmul 연산 결과의 shape를 적어보자
- ▣ 만일, 연산이 불가능하다면 이유를 적어보도록 한다

dot 연산	배열 A	배열 B	matmul 연산
	(3,4)	(4,2)	
	(3,4)	(4,)	
	(3,4)	2	
	(4, 3, 3)	(4, 3, 2)	
	(4, 2, 3)	(2, 3, 2)	
	(5, 4, 2, 3)	(4, 3, 2)	
	(2, 4, 3, 3)	(3, 4)	
	(3, 3)	(2, 3, 4)	
	(2, 3, 3)	(4, 2, 3, 4)	



Don't worry Be happy!

조금씩 알아 가다 보면 언젠가 많이 알 수 있다!

