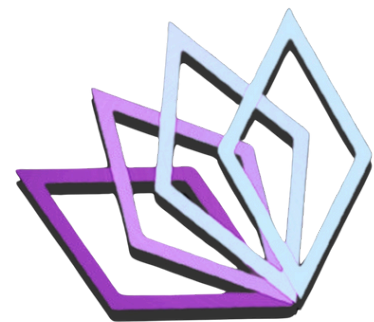




데이터 분석 스터디




1주차


목 차


- 1 코랩 사용법
- 2.1 넘파이 기초
- 2.2 넘파이 인덱싱, 슬라이싱, 반복
- 2.3 결합과 분리
- 2.4 넘파이의 특별한 행렬과 벡터


Colab 사용법

DATA ANALYSIS

 **드라이브**

 새 폴더

 파일 업로드

 폴더 업로드


Google 문서


Google 스프레드시트


Google 프레젠테이션

Google 설문지

더보기

 스팸

 휴지통

 저장용량(81% 사용 중)

15GB 중 12.29GB 사용

추가 저장용량 구매


단축키 첫 글자를 탐색할 수 있도록 Drive 단축키가


드라이브에서 검색


유형


파일


폴더


 Google 드로잉


 Google 내 지도

 Google 사이트 도구

 Google Apps Script

 Google Colaboratory

 Google Jamboard

 연결할 앱 더보기

내 드라이브

 G Suite Marketplace

Colaboratory

검색결과: Colaboratory



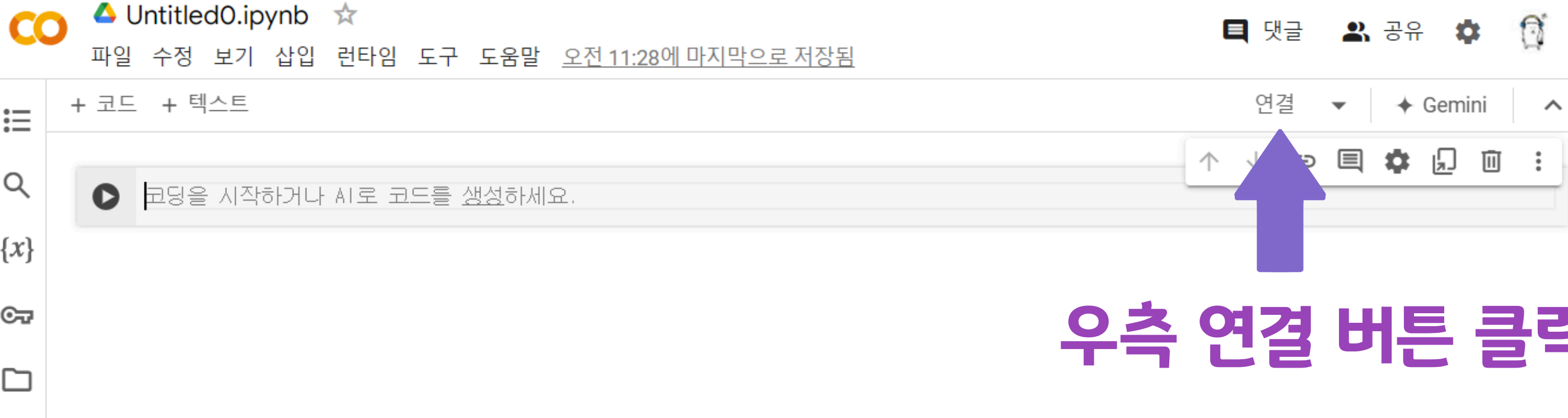
Colaboratory

4.7 ★★★★★ (2390)

1,551,498

Colab 설치!

DATA ANALYSIS



The screenshot shows the Google Colab interface for a notebook titled 'Untitled0.ipynb'. The top bar includes the Colab logo, the notebook title, and a star icon. Below this is a menu bar with options like '파일' (File), '수정' (Edit), '보기' (View), '삽입' (Insert), '런타임' (Runtime), '도구' (Tools), and '도움말' (Help). A status bar indicates the notebook was last saved on '오전 11:28에' (at 11:28 AM). On the right side of the top bar, there are icons for '댓글' (Comments), '공유' (Share), '설정' (Settings), and a user profile icon. Below the top bar, there is a toolbar with '+ 코드' (Add Code) and '+ 텍스트' (Add Text) buttons. The main area of the notebook shows a code cell with the text '코딩을 시작하거나 AI로 코드를 생성하세요.' (Start coding or generate code with AI). To the right of the code cell, there is a '연결' (Connect) button, which is highlighted with a large purple arrow. Below the '연결' button, there is a dropdown menu showing 'Gemini' as the selected option. At the bottom of the interface, there is a '노트 설정' (Note Settings) section. It includes a '런타임 유형' (Runtime Type) dropdown set to 'Python 3'. Below that, there is a '하드웨어 가속기' (Hardware Accelerator) section with radio buttons for 'CPU', 'T4 GPU', 'A100 GPU', 'L4 GPU', and 'TPU v2'. The 'CPU' option is currently selected. Below the hardware accelerator section, there is a message: '프리미엄 GPU를 이용하실까요? [추가 컴퓨팅 단위 구매](#)' (Would you like to use premium GPU? [Purchase additional computing units](#)). At the bottom of the settings section, there are two checkboxes: '모든 실행에서 첫 번째 셀 또는 섹션 자동 실행' (Automatically run the first cell or section in all executions) and '이 노트를 저장할 때 코드 셀 출력 생략' (Omit code cell output when saving this notebook). At the very bottom of the interface, there are two buttons: '취소' (Cancel) and '저장' (Save).

우측 연결 버튼 클릭

수정-노트설정-gpu인스턴스 설정이 가능함

취소 저장

▶ 코딩을 시작하거나 AI로 코드를 생성하세요.

파일 이름 변경 가능

+코드, +텍스트로 새로운 셀 삽입

*단축키

현재 셀 위에 새로운 셀 추가: **컨트롤+M A**

현재 셀 아래 새로운 셀 추가: **컨트롤+M B**




현재 셀 삭제: **컨트롤+M D**

현재 셀 실행, 커서는 해당 셀: **컨트롤+엔터**






현재 셀 실행, 커서는 다음 셀: **쉬프트+엔터**

현재 셀 실행, 커서는 다음 셀 삽입: **알트+엔터**



DATA ANALYSIS

  Untitled0.ipynb 


파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨



+ 코드 + 텍스트

  0초

```
print('100+200')  
print(100+200)
```

 100+200
300

셀

실행 결과

**셀에 원하는 코드를 입력하고
컨트롤+엔터로 실행**

DATA ANALYSIS

✓
0초 [3] a=3

✓
0초 [4] a=a+1

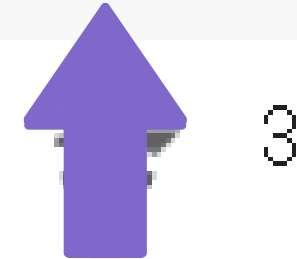
✓
0초 [5] a



✓
0초 [6] a=3

✓
0초 [4] a=a+1

✓
0초 [7] a



**셀 앞 쪽 숫자들은 실행 순서를 나타내고
순서에 따라 결과도 달라진다**

2.1

넘파이 기초

Numpy와 List의 비교

```
[ ] lst1 = [1,2,3,4,5]  
    lst = lst1*2  
    lst
```

```
[ ] import numpy as np  
    np1=np.array(lst1)  
    np1  
    np1*2
```

lst1 리스트를 numpy배열로 바꿈

numpy.array(data)

data = list, tuple, dict 자료형 가능

각 셀을 실행시켜 봅시다

Numpy와 List의 비교

```
[ ] lst1 = [1,2,3,4,5]
    lst = lst1*2
    lst
```

```
[ ] import numpy as np
    np1=np.array(lst1)
    np1
    np1*2
```

각 셀을 실행시켜 봅시다

✓
0초

```
[8] lst1 = [1,2,3,4,5]
    lst = lst1*2
    lst
```

⇒ [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

✓
0초

```
[10] import numpy as np
    np1=np.array(lst1)
    np1
    np1*2
```

⇒ array([2, 4, 6, 8, 10])

다르다!

넘파이는 리스트와 다르게 연산이 가능하다!

`[1,2,3,4,5]` + `[5,4,3,2,1]` = `[1,2,3,4,5,5,4,3,2,1]`

[그림 2-1] 리스트의 덧셈

```
lst2 = [5,4,3,2,1]
sum_lst = lst1 + lst2
sum_lst
```

결과	[1, 2, 3, 4, 5, 5, 4, 3, 2, 1] #sum_lst
----	---

```
np2 = np.array(lst2)
sum_np = np1 + np2
sum_np
```

결과	[6 6 6 6 6]
----	-------------

DATA ANALYSIS

```
lst3 = ['a','b','c','d','e']
lst4 = [1,'a',2,'b',3,'c']
1 print(lst3)
2 print(lst4)
```

결과	1	['a', 'b', 'c', 'd', 'e']
	2	[1, 'a', 2, 'b', 3, 'c']

리스트(연산 목적X)
문자열, 정수, 실수
저장가능

```
np3 = np.array(lst3)
np4 = np.array(lst4)
1 print(np3)
2 print(np4)
```

결과	1	['a' 'b' 'c' 'd' 'e']	#np3
	2	['1' 'a' '2' 'b' '3' 'c']	#np4

넘파이(연산 목적O)
강제로 하나의 자료형 통일

[1,2,3,4,5]

Numpy Attributes		[[1,2,3], [4,5,6]]
ndim	배열의 차원	2
shape	배열의 모양 (튜플)	(2, 3)
size	원소의 개수	6
itemsize	원소의 크기(byte)	4
dtype	데이터 유형	int32

[그림 2-3] Numpy 속성들

1	<code>print(n.ndim)</code>
2	<code>print(n.shape)</code>
3	<code>print(n.size)</code>
4	<code>print(n.itemsize)</code>
5	<code>print(n.dtype)</code>

결과	1	1	# 1차원
	2	(5,)	#1행, 5개
	3	5	#5개
	4	4	#byte
	5	int32	

```
1 np_t1 = np.array([[1,2,3],[4,5,6]], dtype = 'int8')
2 print(np_t1.itemsize)
3 print(np_t1.dtype)
```

```
1 np_t1 = np.array([[1,2,3],[4,5,6]])
2 np_t1 = np_t1.astype('int8')
3 print(np_t1.itemsize)
4 print(np_t1.dtype)
```

결과	1	1	#1byte = 8bit
	2	int8	#8bit

배열 생성과 동시에
자료형 지정

astype()함수를 통해
자료형 변경

numpy.arange(start, end-1, step)

```
np_range = np.arange(1,10,2)  # 10은 포함되지 않는다.  
print(np_range)
```

결과	[1 3 5 7 9]
----	-------------

```
np_array = np.arange(0.1, 1.0, 0.1)  
print(np_array)
```

실수도 가능!

결과	[0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
----	---------------------------------------

numpy.arange(start, end-1, step).reshape(row, colum)

```
np_array1 = np.arange(1, 10).reshape(3, 3)    #3행 3열로 변경
np_array2 = np.arange(1, 10).reshape(3, -1)    #행은 3행으로, 열은 자동으로 변경
print(np_array1)
print(np_array2)
```

결과	[[1 2 3] [4 5 6] [7 8 9]]
	[[1 2 3] [4 5 6] [7 8 9]]

row자리에 오류가 없을 시
colum자리에 -1을 사용해도 된다

Numpy 통계 관련 메서드

Function	Description
np.sum()	배열의 합을 구함
np.mean()	배열의 평균을 구함
np.max()	배열 원소들 중 최댓값 구함
np.min()	배열 원소들 중 최솟값 구함
np.median()	배열 원소들 중 중앙값 구함
np.std()	배열의 표준편차를 구함
np.var()	배열의 분산을 구함
np.argmin()	배열 원소들 중 최솟값의 index
np.argmax()	배열 원소들 중 최댓값의 index
np.percentile()	배열 원소들 중 백분위수를 구함

2.2

넘파이 인덱싱, 슬라이싱, 반복

넘파이 인덱싱

```
np1 = np.arange(1,10) # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
print(np1[[1,3,5]]) # 1,3,5의 인덱스를 한 번에 인덱싱
```

한 번에 여러 개의 인덱스 지정 가능

```
indexing1 = [1,3,5]
```

```
print(np1[indexing1])
```

여러 인덱스 지정할 때는 리스트나

넘파이 배열로 지정해야 함

```
[2 4 6]
```

```
[2 4 6]
```

넘파이 슬라이싱

```
print(np1[3:6]) *마지막 인덱스에서 -1
```

```
# np1[3] ~ np1[5] 슬라이싱
```

```
[4 5 6]
```

논리적 인덱싱

조건을 통해 배열에서 원하는 값만 선택

예제1

어떤 사람이 한 주간 수면시간을 기록했다.

이 사람이 6.5시간보다 많이 잔 날은 며칠?

DAY	1	2	3	4	5	6	7
sleep_time	6.6	7.2	6.0	7.2	5.5	9.5	9

방법1 : for문 사용

```
sleep_time = [6.6, 7.2, 6.0, 7.2, 5.5, 9.5, 9]  
count = 0  
for i in sleep_time:  
    if i > 6.5:  
        count += 1  
print(count)
```

5

방법2 : 넘파이 배열의 연산 이용

```
# numpy 연산 -> 배열 속 원소에 초점을 둔 연산
np_time = np.array(sleep_time)
days = np_time > 6.5 배열의 각 원소와 비교 연산 진행하며  

그 원소 자리에 불리언 값을 넣은 배열 반환
print(days)
print(days.sum()) # True = 1, False = 0 으로 계산

[ True  True False  True False  True  True]
5
```


예제2

10월 온도 데이터 중 평균 온도보다 높은 날은 며칠?

```
list_oct = [20.1, 19.9, 19.7, 19.6, 19.5, 19.1, 18.8, 18.6, 18.4, 18.2, 18.2,
            18.2, 18.2, 18.1, 17.9, 17.7, 17.3, 17.0, 16.8, 16.6, 16.3, 16.1,
            15.9, 15.5, 15.3, 15.2, 15.3, 15.5, 15.8, 15.7, 15.6]

np_oct = np.array(list_oct)
days = np_oct > np_oct.mean()
print(days)
print(days.sum())
```

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

16

예제3

평균온도보다 높은 온도들은 몇 도?

방법1 : 불리언 배열 인덱싱 사용

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

```
print(np_oct[days])
```

```
[20.1 19.9 19.7 19.6 19.5 19.1 18.8 18.6 18.4 18.2 18.2 18.2 18.2 18.1
 17.9 17.7]
```

불리언 배열 인덱싱

: 불리언 값으로 이루어진 배열을 사용해 인덱싱 하는 것
넘파이 배열에서 지원하는 고유한 기능

조건 : 인덱싱 하려는 배열과 불리언 배열의 길이가 일치해야 함

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

```
print(np_oct[days])
```

np_oct[days]를 실행하면 넘파이에서 days를 순회하며
원소가 True인 곳의 인덱스를 수집하고 이 인덱스들을 사용해 인덱싱

방법2 : np.where() 사용

```
# np.where() 참고
np_bool = np.array([True,True,False,False,False])
np.where(np_bool)

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

np.where() : 불리언 배열을 입력하면 배열에서 True값이 있는 곳의 인덱스가 담긴 넘파이 배열을 튜플에 담아 반환

```
print(np_oct[np.where(days)[0]])

[20.1 19.9 19.7 19.6 19.5 19.1 18.8 18.6 18.4 18.2 18.2 18.2 18.2 18.1
 17.9 17.7]
```

문제4

휴대폰 매장에서 올해 매월 팔린 아이폰의 개수

1월 2월 3월 4월 5월 6월 7월 8월 9월 10월 11월 12월

```
iphone_sold = [92, 71, 83, 79, 63, 42, 53, 77, 91, 104, 121, 101]
```

평균보다 작게 팔린 달은 몇 월이 있나

방법1 : 불리언 배열 인덱싱

```
iphone_sold = [92, 71, 83, 79, 63, 42, 53, 77, 91, 104, 121, 101]
month = np.arange(1,13) # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
np_sold = np.array(iphone_sold)
less_avg = np_sold < np_sold.mean()
print(month[less_avg])
```

```
[2 4 5 6 7 8]
```

방법2 : np.where()

```
a = np.where(less_avg)[0]  
print(a+1)
```

```
[2 4 5 6 7 8]
```

불리언 배열을 입력하면 원소가 True인 곳의
인덱스가 담긴 배열 반환 -> 인덱스에 +1하면 월

2차원 배열의 인덱싱과 슬라이싱

```
# 2차원 넘파이 배열 생성
```

```
np_2d = np.arange(1,10).reshape(3,-1)
```

```
print(np_2d)
```

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

```
# 1행 1열 인덱싱
```

```
print(np_2d[1][1])
```

```
print(np_2d[1,1])
```

```
# 리스트에서는 지원되지 않는 방식
```

```
5
5
```


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

`np_2d[1][1]`과 `np_2d[1,1]`의 차이

`np_2d[1][1]` : 순차적으로 진행

`np_2d[1] = [4 5 6]` (인덱스 : 0 1 2)

`np_2d[1][1] = [4 5 6]`에서 1번 인덱싱

`np_2d[1,1]` : 동시에 진행

`np_2d[1,1] = 1행 1열 인덱싱`

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

np_2d[1][1]과 np_2d[1,1]의 차이

np_2d[1][1] : 순차적으로 진행

np_2d[1] = [4 5 6] (인덱스 : 0 1 2)

np_2d[1][1] = [4 5 6]에서 1번 인덱싱

np_2d[1,1] : 동시에 진행

np_2d[1,1] = 1행 1열 인덱싱

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

`np_2d[1][1]`과 `np_2d[1,1]`의 차이

`np_2d[1][1]` : 순차적으로 진행

`np_2d[1] = [4 5 6]` (인덱스 : 0 1 2)

`np_2d[1][1] = [4 5 6]`에서 1번 인덱싱

`np_2d[1,1]` : 동시에 진행

`np_2d[1,1] = 1행 1열 인덱싱`

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

`np_2d[1][1]`과 `np_2d[1,1]`의 차이

`np_2d[1][1]` : 순차적으로 진행

`np_2d[1] = [4 5 6]` (인덱스 : 0 1 2)

`np_2d[1][1] = [4 5 6]`에서 1번 인덱싱

`np_2d[1,1]` : 동시에 진행

`np_2d[1,1] = 1행 1열 인덱싱`

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

`np_2d[:,2]`과 `np_2d[:,2]` 비교

`np_2d[:,2]` = 전체 행 인덱싱

`np_2d[:,2]` = 2행 인덱싱

`np_2d[:,2]` = 전체행 2열 인덱싱

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

`np_2d[:,2]`과 `np_2d[:,2]` 비교

`np_2d[:,2]` = 전체 행 인덱싱

`np_2d[:,2]` = 2행 인덱싱

`np_2d[:,2]` = 전체행 2열 인덱싱

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

`np_2d[:,2]`과 `np_2d[:,2]` 비교

`np_2d[:,2]` = 전체 행 인덱싱

`np_2d[:,2]` = 2행 인덱싱

`np_2d[:,2]` = 전체행 2열 인덱싱

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

`np_2d[:,2]`과 `np_2d[:,2]` 비교

`np_2d[:,2]` = 전체 행 인덱싱

`np_2d[:,2]` = 2행 인덱싱

`np_2d[:,2]` = 전체행 2열 인덱싱

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

`np_2d[[0,2]][0]`과 `np_2d[[0,2],0]` 비교

`np_2d[[0,2]]` = 0과2행 인덱싱

`np_2d[[0,2]][0]` = 0행 인덱싱

`np_2d[[0,2],0]` = 0과2행 0열 인덱싱

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

`np_2d[[0,2]][0]`과 `np_2d[[0,2],0]` 비교

`np_2d[[0,2]]` = 0과 2행 인덱싱

`np_2d[[0,2]][0]` = 0행 인덱싱

`np_2d[[0,2],0]` = 0과 2행 0열 인덱싱

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

`np_2d[[0,2]][0]`과 `np_2d[[0,2],0]` 비교

`np_2d[[0,2]]` = 0과2행 인덱싱

`np_2d[[0,2]][0]` = 0행 인덱싱

`np_2d[[0,2],0]` = 0과2행 0열 인덱싱

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

`np_2d[[0,2]][0]`과 `np_2d[[0,2],0]` 비교

`np_2d[[0,2]]` = 0과2행 인덱싱

`np_2d[[0,2]][0]` = 0행 인덱싱

`np_2d[[0,2],0]` = 0과2행 0열 인덱싱

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

문제5

`np_2d[0:2][0:2]`와 `np_2d[0:2,0:2]` 비교

`np_2d[0:2]` = 0~1행 인덱싱

`np_2d[0:2][0:2]` = 0~1행 인덱싱

`np_2d[0:2,0:2]` = 0~1행 0~1열 인덱싱

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

문제5

`np_2d[0:2][0:2]`와 `np_2d[0:2,0:2]` 비교

`np_2d[0:2]` = 0~1행 인덱싱

`np_2d[0:2][0:2]` = 0~1행 인덱싱

`np_2d[0:2,0:2]` = 0~1행 0~1열 인덱싱

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

문제5

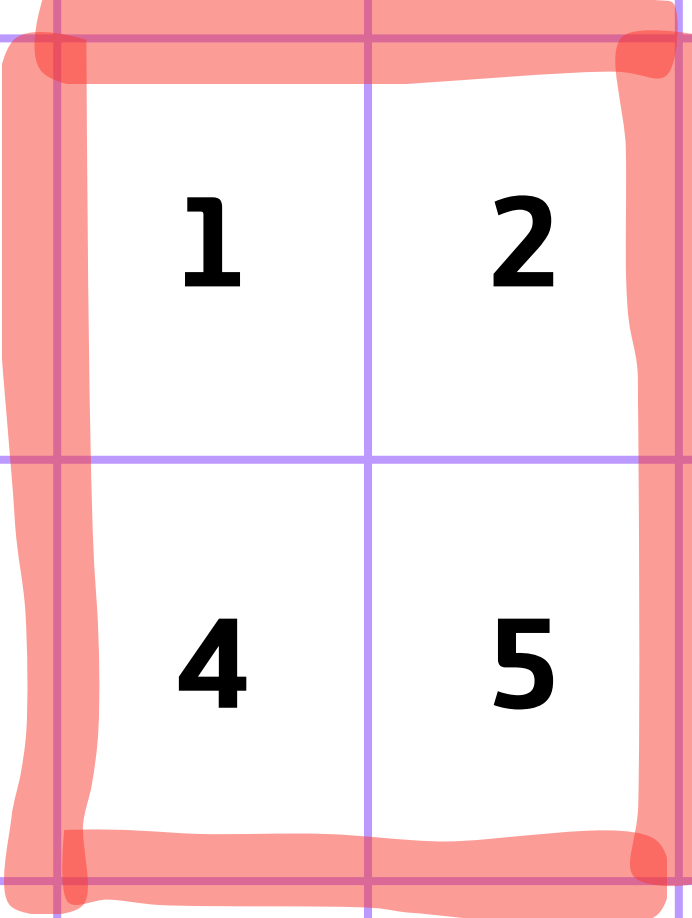
`np_2d[0:2][0:2]`와 `np_2d[0:2,0:2]` 비교

`np_2d[0:2]` = 0~1행 인덱싱

`np_2d[0:2][0:2]` = 0~1행 인덱싱

`np_2d[0:2,0:2]` = 0~1행 0~1열 인덱싱

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



문제5

`np_2d[0:2][0:2]`와 `np_2d[0:2,0:2]` 비교

`np_2d[0:2]` = 0~1행 인덱싱

`np_2d[0:2][0:2]` = 0~1행 인덱싱

`np_2d[0:2,0:2]` = 0~1행 0~1열 인덱싱

2.3

결합과 분리

np.concatenate()

1차원 배열의 결합

```
n1 = np.array([1,2,3])
n2 = np.array([4,5,6])
n3 = np.array([7,8,9])

print(np.concatenate((n1,n2,n3)))
# 리스트나 numpy배열 또는 튜플에 결합할 배열들을 넣어서 입력
```

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

2차원 배열의 결합

```
n1 = np.arange(1,7).reshape(2,-1)
n2 = np.arange(7,13).reshape(2,-1)
con1 = np.concatenate([n1, n2], axis = 0)
con2 = np.concatenate([n1, n2], axis = 1)
```

행 방향 결합
열 방향 결합

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

```
[[ 1  2  3  7  8  9]
 [ 4  5  6 10 11 12]]
```

행 방향 결합

0	1	2	3
1	4	5	6



2	7	8	9
3	10	11	12

조건 : 열의 길이 일치

열 방향 결합

조건 : 행의 길이 일치

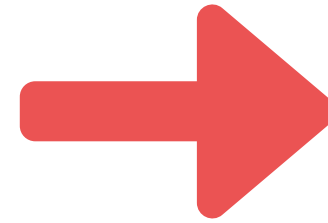
0	1	2
1	2	3
4	5	6



3	4	5
7	8	9
10	11	12

np.transpose()

	0	1
0	10	20
1	30	40



	0	1
0	10	30
1	20	40

n1

행 -> 열
열 -> 행

**np.transpose(n1)
= n1.T**

np.split()

사용1 : `np.split(array, [기준 행 번호], axis = 0)`

```
np_array = np.arange(1,17).reshape(4,-1)
print(np_array)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
```

```
a,b = np.split(np_array, [2], axis = 0)
# 2행 전까지 분리 -> 두 개의 배열을 리스트에 담아 반환

print(a) # np.split(np_array, [2], axis = 0)[0]
print(b) # np.split(np_array, [2], axis = 0)[1]
```

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

```
[[ 9 10 11 12]
 [13 14 15 16]]
```

사용2 : np.split(array, 몇 개로 분리할 지, axis = 0)

조건 : 분리된 배열끼리 크기가 동일해야 함

```
a,b = np.split(np_array, 2, axis = 0)  
# 행 방향, 2개로 분리
```

```
print(a)  
print(b)
```

```
[[ 1  2  3  4]  
 [ 5  6  7  8]  
 [ 9 10 11 12]  
 [13 14 15 16]]
```

```
[[1 2 3 4]  
 [5 6 7 8]]  
  
[[ 9 10 11 12]  
 [13 14 15 16]]
```



```
a,b,c,d = np.split(np_array, 4, axis = 0)  
# 행 방향, 4개로 분리
```

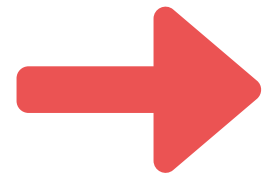
```
print(a)  
print(b)  
print(c)  
print(d)
```

```
[[ 1  2  3  4]  
 [ 5  6  7  8]  
 [ 9 10 11 12]  
 [13 14 15 16]]
```

```
[[1 2 3 4]]  
[[5 6 7 8]]  
[[ 9 10 11 12]]  
[[13 14 15 16]]
```

문제6

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



```
[[5 8]
 [6 9]]
```

방법1 : np.split + np.transpose

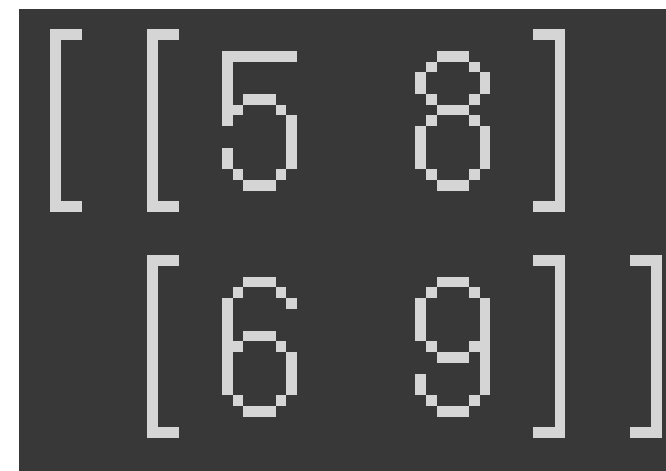
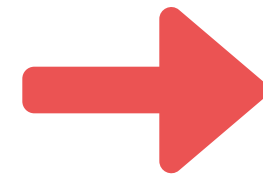
```
array = np.array([[1,2,3],[4,5,6],[7,8,9]])
n1 = np.split(array, [1], axis=0)[1]
n2 = np.split(n1, [1], axis=1)[1]
print(n2.T)
```

방법2 : 슬라이싱 + np.transpose

```
print(array[1:,1:].T)
```



$\begin{bmatrix} [1 & 2 & 3] \\ [4 & 5 & 6] \\ [7 & 8 & 9] \end{bmatrix}$



$\begin{bmatrix} [5 & 8] \\ [6 & 9] \end{bmatrix}$

2.4

넘파이의 특별한 행렬과 벡터

np.zeros()와 np.ones()

영행렬

- 모든 원소가 0인 행렬

`np.zeros(shape, dtype)`

✓
0s



```
nz1 = np.zeros(4)
nz2 = np.zeros(4).reshape(2,2)
nz3 = np.zeros([2,2])
print(nz1)
print(nz2)
print(nz3)
```



```
[0. 0. 0. 0.]
[[0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]
```

np.zeros()와 np.ones()

np.ones()

- 모든 원소가 1인 행렬

np.ones(shape, dtype)

✓
0s



```
nz1 = np.ones(4)
nz2 = np.ones(4).reshape(2,2)
nz3 = np.ones([2,2])
nz4 = np.ones([2,2], dtype = np.int8)
print(nz1)
print(nz2)
print(nz3)
print(nz4)
```



```
[1.  1.  1.  1.]
[[1.  1.]
 [1.  1.]]
[[1 1]
 [1 1]]
```

np.full()와 np.eye()

np.full()

- 동일한 수로 가득 찬 행렬

`np.full(shape, fill_value, dtype)`

✓
0s



```
nf1 = np.full(10,1)
nf2 = np.full([3,3], 5)
nf3 = np.full([3,3], 5, dtype = np.int8)
print(nf1)
print(nf2, "데이터 타입 :", nf2.dtype)
print(nf3, "데이터 타입 :", nf3.dtype)
```



```
[1 1 1 1 1 1 1 1 1 1]
[[5 5 5]
 [5 5 5]
 [5 5 5]] 데이터 타입 : int64
[[5 5 5]
 [5 5 5]
 [5 5 5]] 데이터 타입 : int8
```

np.full()와 np.eye()

np.eye()

- 단위 행렬

`np.eye(row, columns, dtype)`

✓
0s



```
ne1 = np.eye(5,5)
ne2 = np.eye(5,5, dtype = np.int8)
print(ne1, "데이터 타입 :", ne1.dtype)
print(ne2, "데이터 타입 :", ne2.dtype)
```



```
[[1.  0.  0.  0.  0.]
 [0.  1.  0.  0.  0.]
 [0.  0.  1.  0.  0.]
 [0.  0.  0.  1.  0.]
 [0.  0.  0.  0.  1.]] 데이터 타입 : float64
[[1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]] 데이터 타입 : int8
```


np.random.rand()

np.random.rand()

- 0~1의 무작위 수 추출

```
np.random.rand(num)
```

np.random.rand()

✓
0s



```
print(np.random.rand())  
print(np.random.rand(10))  
print(np.random.rand(2,5))
```



```
0.44700002101090264  
[0.7932679  0.2911354  0.10424812 0.36533796 0.43175277 0.824145  
 0.41760231 0.10353346 0.47229606 0.54897158]  
[[0.86431967 0.73810024 0.8327543  0.41723692 0.25750028]  
 [0.17345986 0.22313732 0.76010376 0.65323614 0.17545137]]
```

np.random.random()

행렬을 만들기 위해 차원을 입력할 때,
튜플이나 리스트로 차원을 입력해야 함

✓
0s



```
print(np.random.rand(2,5))  
print(np.random.random([2,5]))
```



```
[[0.25778231 0.69476559 0.03219785 0.34088249 0.86494939]  
 [0.91363795 0.47529987 0.59746422 0.96881206 0.25598015]]  
[[0.73564512 0.79653332 0.47003203 0.08516661 0.61801313]  
 [0.7623624  0.05477853 0.10502939 0.65790099 0.62722019]]
```

np.random.randint()

np.random.randint()

- 정해진 구간에서 주어진 차원의 벡터나 행렬을 만들 때 사용

```
np.random.randint(minimum, maximum-1, size)
```

np.random.randint()

✓
0s



```
print(np.random.randint(1,10,3))  
print(np.random.randint(1,10,[3,3]))
```



```
[7 6 9]  
[[3 8 1]  
 [9 4 7]  
 [7 1 1]]
```

np.random.seed()

np.random.seed()

- 난수들에 패턴 부여, 반복적이고 예측 가능한 난수 발생

```
np.random.seed(num)
```

np.random.seed()

같은 **seed**를 전달했을 때 같은 패턴의 결과가 나온다.



0s



```
np.random.seed(5)  
print(np.random.randint(1,10,3))  
np.random.seed(5)  
print(np.random.randint(1,10,3))
```



```
[4 7 7]  
[4 7 7]
```

질문의 응답

특별 문예 풀이/해설

문계 교환

마무리

과제 설명

발표 준비 : 3팀!

ch.3 판다스 시리즈

9월 23일에 봐요!



1주차 마치겠습니다
구고하셨습니다!