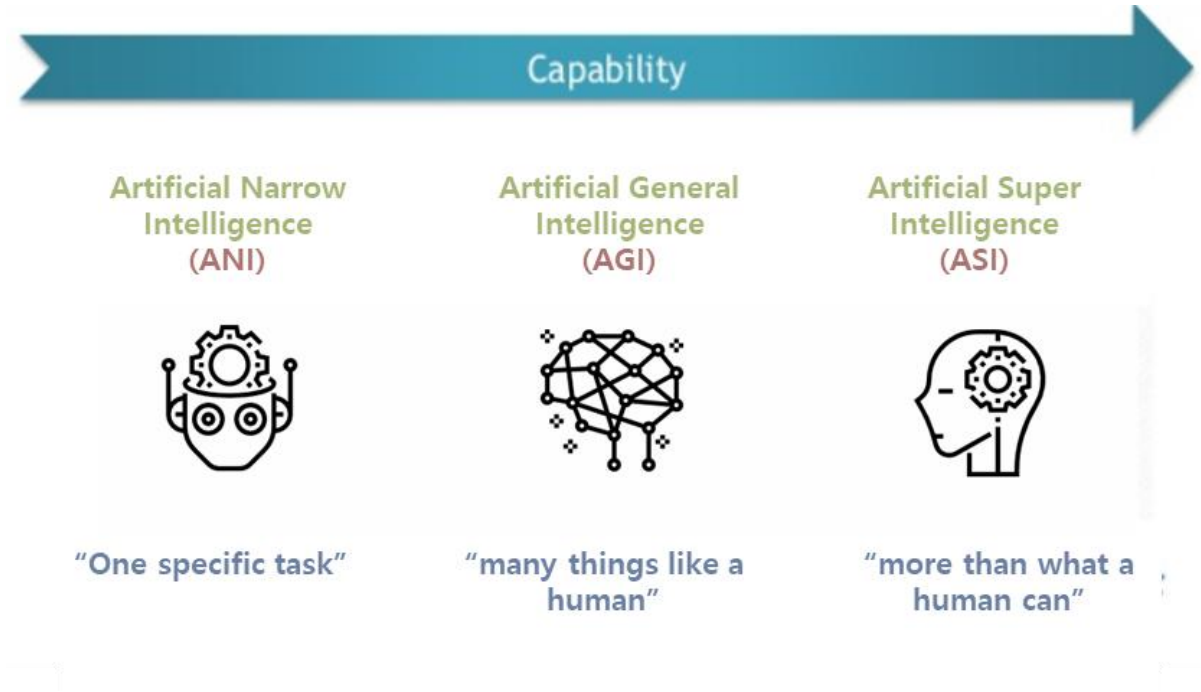
[기술공통]

데이터 엔지니어링

한화시스템

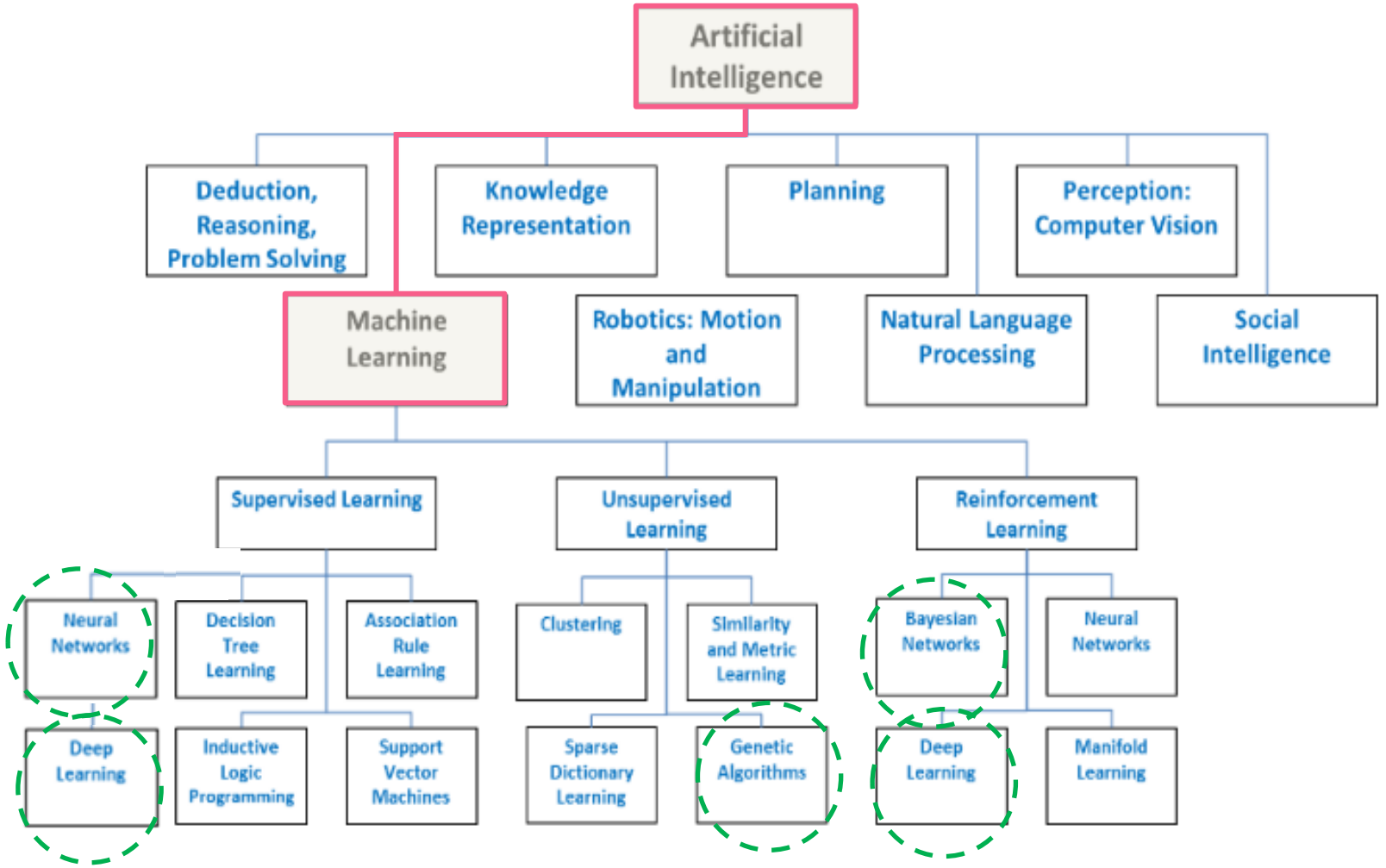
Table of Contents

- AI



- '약인공지능(ANI)', '범용인공지능(AGI)', '초인공지능(ASI)'
- ANI: 주어진 데이터 내에서 주어진 문제를 해결하는 AI
- AGI로의 진입 가속화 중

- AI


Artificial Intelligence
Deduction, Reasoning, Problem Solving
Knowledge Representation
Planning
Perception: Computer Vision
Machine Learning
Robotics: Motion and Manipulation
Natural Language Processing
Social Intelligence
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Neural Networks
Decision Tree Learning
Association Rule Learning
Clustering
Similarity and Metric Learning
Bayesian Networks
Neural Networks
Deep Learning
Inductive Logic Programming
Support Vector Machines
Sparse Dictionary Learning
Genetic Algorithms
Deep Learning
Manifold Learning

- **Scaling Laws (Open AI, 2020)**
  - 컴퓨팅 리소스, 데이터, 모형 크기를 늘릴 수록 성능 개선
  - 새로운 능력이 생겨남 (Emergent Abilities)

Scaling Laws for Neural Language Models

Jared Kaplan *
Johns Hopkins University, OpenAI
jaredk@jhu.edu

Sam McCandlish*
OpenAI
sam@openai.com

Tom Henighan
OpenAI
henighan@openai.com

Tom B. Brown
OpenAI
tom@openai.com

Benjamin Chess
OpenAI
bchess@openai.com

Rewon Child
OpenAI
rewon@openai.com

Scott Gray
OpenAI
scott@openai.com

Alec Radford
OpenAI
alec@openai.com

Jeffrey Wu
OpenAI
jeffwu@openai.com

Dario Amodei
OpenAI
damodei@openai.com

Abstract

We study empirical scaling laws for language model performance on the cross-entropy loss. The loss scales as a power-law with model size, dataset size, and the amount of compute used for training, with some trends spanning more than seven orders of magnitude. Other architectural details such as network width or depth have minimal effects within a wide range. Simple equations govern the dependence of overfitting on model/dataset size and the dependence of training speed on model size. These relationships allow us to determine the optimal allocation of a fixed compute budget. Larger models are significantly more sample-efficient, such that optimally compute-efficient training involves training very large models on a relatively modest amount of data and stopping significantly before convergence.

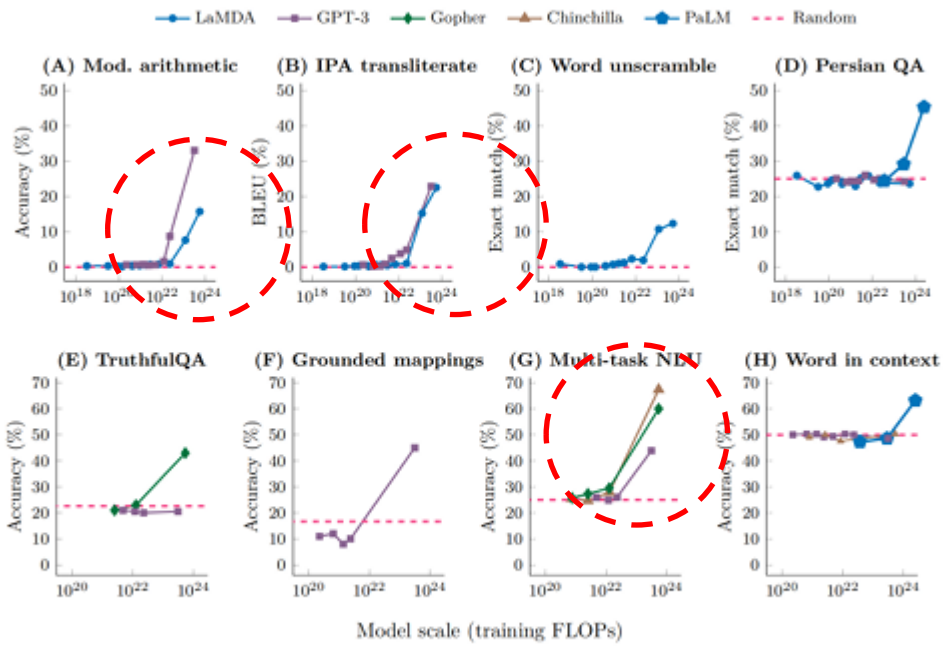


Figure 1: **Emergent abilities of large language models**. Model families display *sharp* and *unpredictable* increases in performance at specific tasks as scale increases. Source: Fig. 2 from [33].

- **DX to AX: AI Transformation**

  - AI도입과 관련한 기업 의견 및 기술 활용도 인식

(단위: %; N=1,000)

구분	비중	구분	비중
기업 수요에 맞는 AI 기술 및 솔루션 부족	35.8	데이터 활용(개인정보 및 데이터 접근)	15.6
		성과 창출의 불확실성	11.2
AI 기술 및 솔루션 개발 비용	20.6	기타	1.0
전문인력 및 역량 부족	15.7	모름/무응답	0.1

* 자료: KDI(2020)

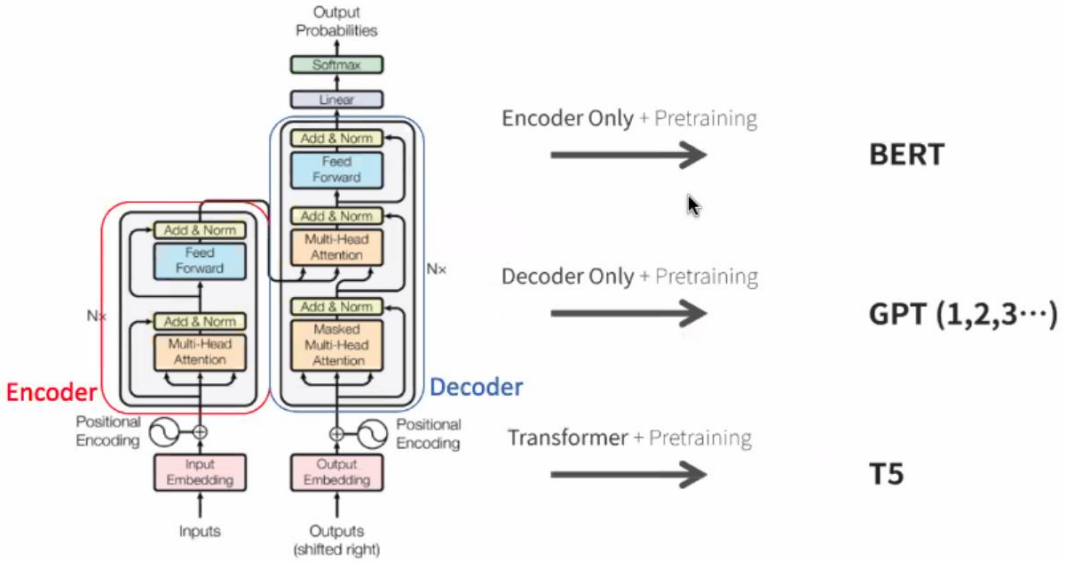
(단위: %; N=738)

ID	세부 기술(토픽)	기술 활용도		수용도	기술 유용성		개발 시급성 정도	
		현재 활용도	미래 전망	수용 의사	성과 도움	경쟁력 제고	R&D 시급	국고 지원
1	자연어 이해 및 인식 처리 기술	54.7	64.8	51.9	60.3	64.6	58.1	56.5
2	인간 감정 분석 기술	49.2	58.8	45.5	54.2	57.9	51.4	50.1
3	지식 추론 기술	56.9	67.2	54.3	60.8	60.6	55.6	58.7
4	생성형 인공지능 기술	70.1	76.0	62.2	74.9	75.6	70.5	73.8
5	인공지능 신뢰성 기술	64.2	76.2	62.1	70.7	76.3	67.8	71.5
6	경로 탐색 및 모델 최적화	68.6	73.2	64.9	71.5	73.6	72.8	72.2
7	객체 감지 및 추적을 위한 비전 딥러닝 기술	66.8	77.5	65.9	70.7	71.7	73.4	74.1
8	그래프 분석 기반 진단 및 예측 기술	66.0	74.3	62.7	63.7	68.6	67.3	69.8
9	강화학습 기술	66.0	74.7	65.9	68.0	71.3	68.6	70.6
10	머신러닝 기반 데이터 보안 및 보호 기술	68.7	76.8	68.4	75.1	76.3	72.9	72.6
11	딥러닝 기반 이미지 분석 및 처리 기술	73.3	78.3	70.1	74.4	76.8	75.2	73.2
12	딥러닝 모델 알고리즘 및 성능 최적화	71.1	82.8	66.0	76.4	78.0	73.3	75.9

* 주: 각 토픽별 최고값과 최저값의 항목을 각각 파란색, 주황색으로 표시함

Source: 우리나라 및 주요국 인공지능(AI) 기술수준의 최근 변화 추이 (SPRI,2023)

- **간단하게 이해하는 LLM**
  - 언어모형: 주어진 문장을 구성하는 단어에 대한 확률을 모델링
  - 주어진 문장/단어의 다음 단어를 예측, 문장 내 Masking한 단어를 예측
  - 대용량 텍스트를 학습한 모형이 좋은 Representation(Embedding)을 보유

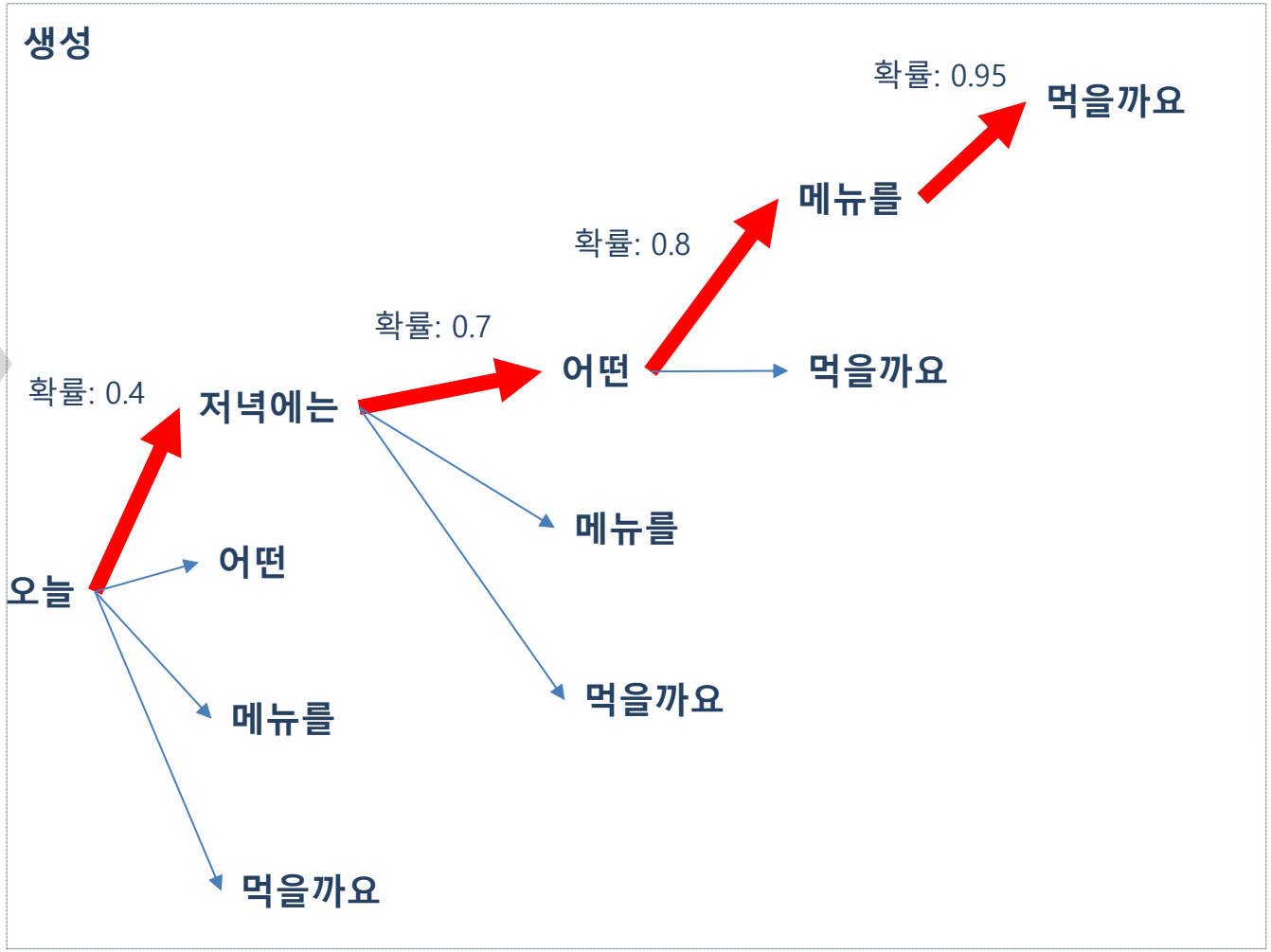


- 간단하게 이해하는 LLM

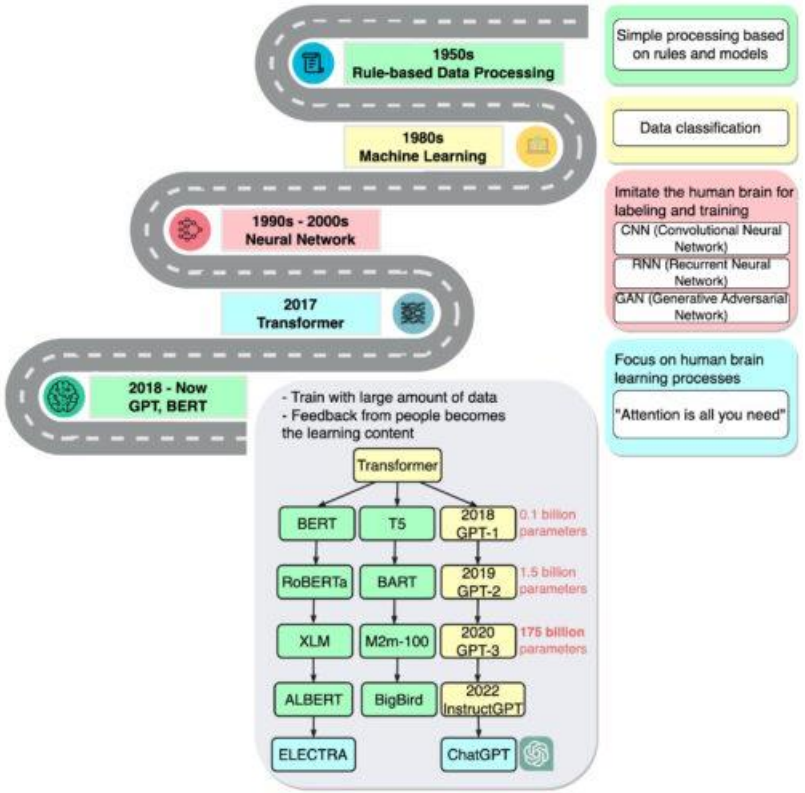
"오늘 저녁에는 어떤 메뉴를 먹을까요?"

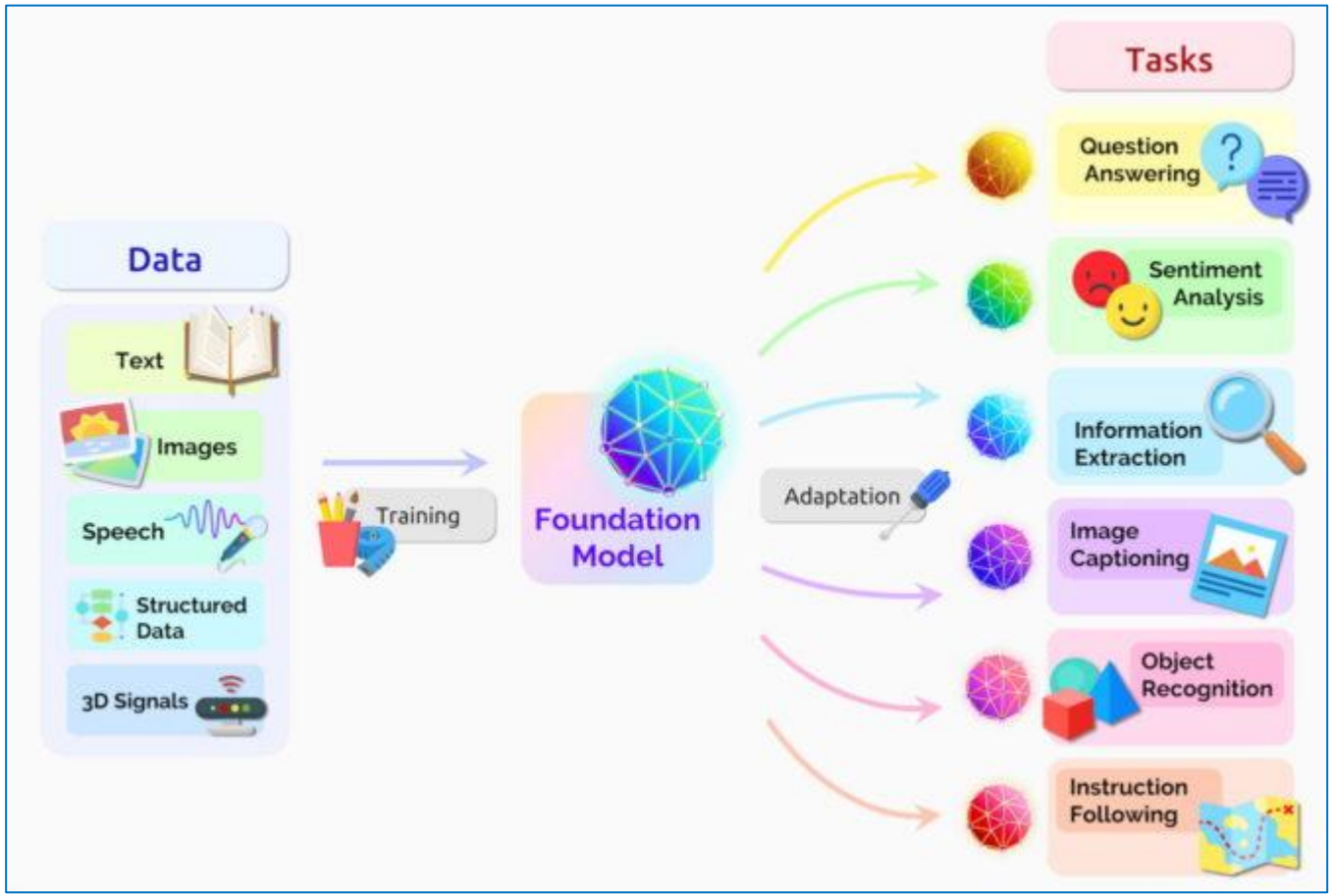
오늘 : 0.5, 1.2, 0.3, ..., 0.9
저녁에는 : 1.1, 0.2, 1.3, ..., 0.7
어떤 : 1.5, 0.2, 0.9, ..., 0.5
메뉴를 : 1.1, 1.7, 1.3, ..., 1.5
먹을까요 : 1.12, 1.3, 1.7, ..., 1.1

Representation (Embedding)



- **Foundation Model**
  - 비지도학습을 통해 학습된 AI신경망
  - 다양한 작업에 응용
  - Open source LLM



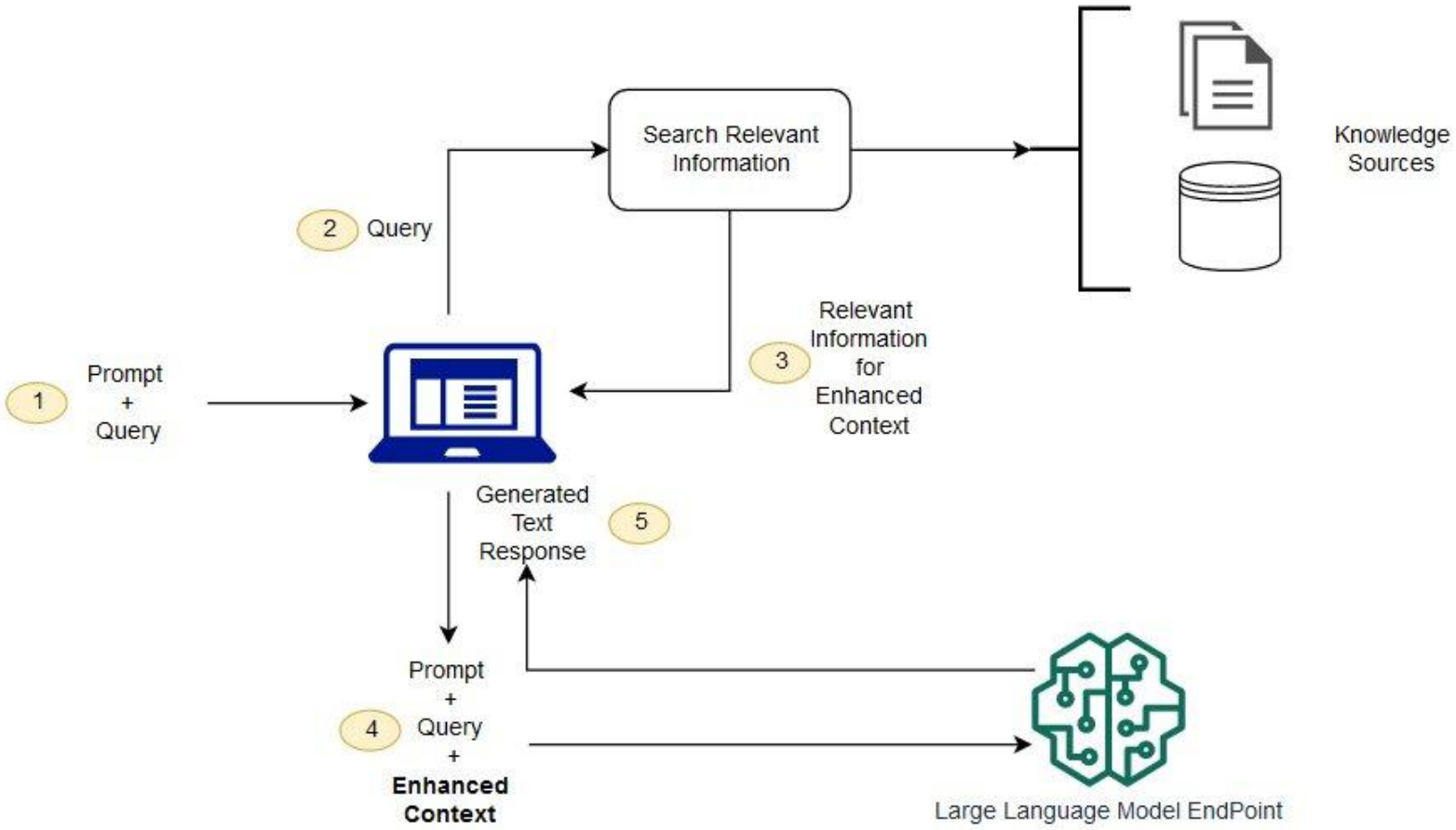


Source: nvidia blog

- **RAG: 검색 증강 생성**
  - Retrieval-Augmented Generation
  - Hallucination을 줄이는 비용 효율적인 접근 방식
    - LLM의 기능을 특정 도메인이나 특정 기관의 내부 지식 기반으로 확장
    - LLM 외부의 데이터를 활용해서 보다 정확한 답변을 찾는 방안
    - 모형 학습을 다시 할 필요는 없음

- **RAG: 검색 증강 생성**

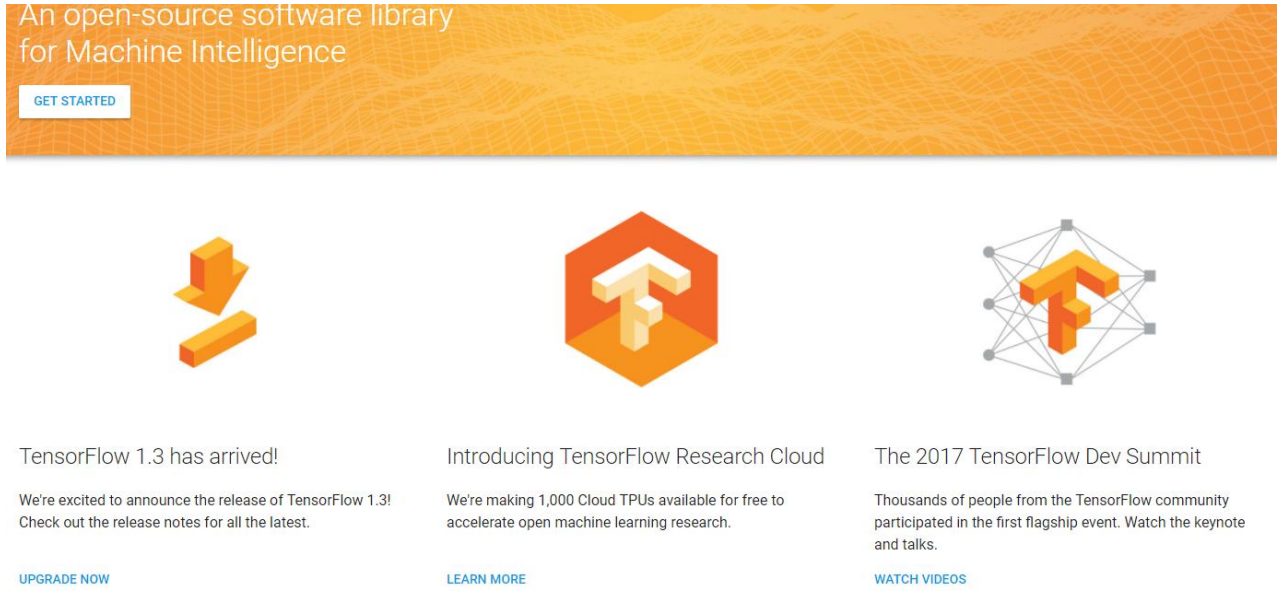
- LLM 학습 데이터 외의 데이터인 외부 데이터 생성 후 임베딩을 통해 벡터DB저장
- 질문에 대한 관련 정보 검색
- 외부 데이터 업데이트: 실시간 및 배치 방식



II. 자료구조와 파이썬 문법

Python

- 1990년대 후반 Guido van Rossum이 개발 (네덜란드)
- 오픈소스 platform: General Public License (GPL)라이선스)
- Python의 특징: High-level / Interactive / 객체지향 / Scripting 언어
- Easy-to-learn / Easy-to-read / Easy-to-maintain



- **데이터 구조?**
  - 데이터를 일정한 기준에 의해 모아 놓은 것 (collection of data elements, structured in some way)
- **List**
  - 여러 같은 type의 값을 하나의 객체로 모음
  - 관련 함수: methods (object에 관련된 함수): len, max, min
  - object.method(arguments) 방식으로 사용: lst.append(값), lst.count(값), lst.insert(위치, 값), lst.pop(), lst.reverse(), lst.remove(값), lst.sort()

- **List**
  - 숫치형 값 또는 문자열 등 여러 값의 모임
    - indexing: Sequence 내의 element에 대해 순서 별 index로 접근
    - slicing: 일정 범위 내의 element를 access하는 것
  - 값의 추가: 같은 type의 값을 추가

- **Set**
  - list와 같은 값의 모임
  - 값의 순서가 없음, 집합으로 이해
  - { }로 생성

- **Tuple**
  - 다른 형태의 sequence, list와 유사, 0 index에서 시작하는 값
  - (, ,)로 표현
  - Immutable sequence: 일단 생성 후에는 값을 변경할 수 없음. 문자열과 유사
  - sort, append, reverse 등의 method는 사용 불가

- **Dictionary**
  - 각 값마다 이름이 있는 값의 모음(A “bag” of values, each with its own label)
  - 파이썬의 유용한 기능으로, key-value 구조를 지원
  - Dictionary는 { }를 이용하여 생성 / 값으로는 key:value 쌍을 , 로 나열
  - 빈 dictionary는 {}를 통해 생성

➢ 조건문과 반복문

- 조건문: 조건의 참/거짓여부에 따라 문장을 각각 실행, if else, if elif else를 사용
- 반복문: 파라미터 값을 변경하면서 동일한 작업을 여러 번 실행, for 사용
- 조건문-bool을 사용하여 조건 충족 여부 판단
- *block은 { }로 표현하지 않고 indentation으로 표현(*참고: 한 레벨당 공백 4개)

➢ 조건문과 반복문

```
var = 100
if var < 200:
    print("Expression value is less than 200")
    if var == 150:
        print("Which is 150")
    elif var == 50:
        print("Which is 50")
    elif var < 50:
        print("Expression value is less than 50")
    else:
        print("Could not find true expression")
```

if에 대한 block

하위 block

- **조건문과 반복문**
  - 반복문-while과 for를 사용
  - for: set/sequence(또는 iterable object(iterate가능 객체)의 각 값에 대해 Code block 수행

```
x =1
while x <=100:
    print(x)
    x+=1
words = ['this','is','a','wonder']
for word in words:
    print(word)
```

➢ 조건문과 반복문의 들여쓰기

- ➢ 같은 level의 statement들은 같은 들여쓰기(indentation)값을 갖아야 함

```
if a==1:
        print(1)
        print(0)
```

X

```
if a==1:
    print(1)
    print(0)
```

O

분기(Branch):

일방향적인 프로그램의 흐름이 어느 지점에서 2개 이상으로 나뉘게 됨

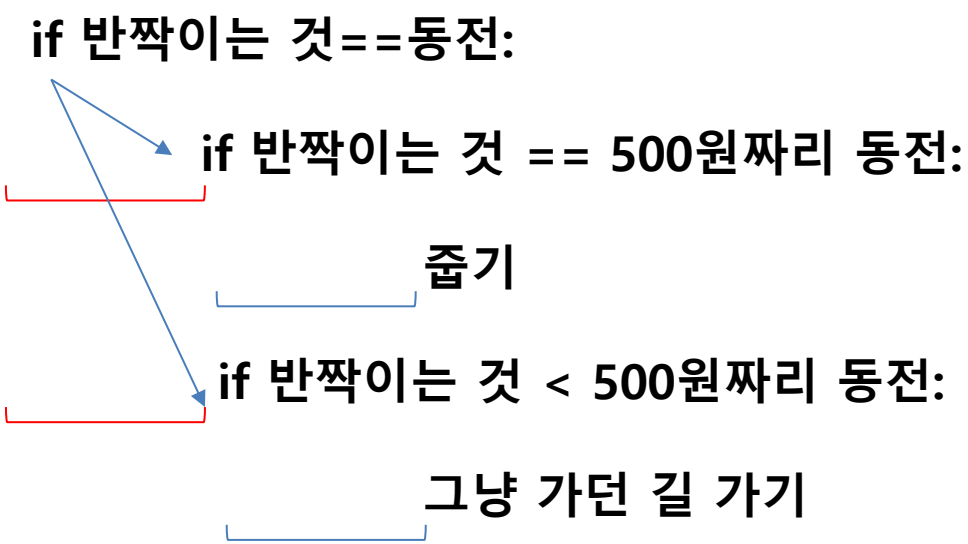
if : 분기(branch)를 위한 명령어

사용 방법:

```
if 조건식:
    코드
    코드
    코드
```

들여쓰기!

조건문의 중첩



else 사용 방법:

```
if 조건식:
    코드1
else:
    코드2
```

True & True는 True
True & False는 False
False & True는 False
False & False는 False

True | True는 True
True | False는 True
False | True는 True
False | False는 False

여러 개의 조건식

```
if 조건식1 &(또는 |) 조건식2:
    코드
if 조건식1 &(또는 |) 조건식2:
    코드1
else
    코드2
```

```
if 국어>90점 & 수학>90점:
    피자
Else:
    식빵
```

for + if 사용

2 1에서 선택된 값이 임시 변수에 임시로 할당

1 시퀀스 자료구조에서 값이 하나씩 선택됨

시퀀스 자료구조의 모든 값이 다 사용될때까지 이 과정이 반복

for 임시변수 in 시퀀스자료구조:

실행할 코드

3 임시변수에 조건문을 적용

```
for i in sales:
    if i > 50:
        print(i)
```

들여쓰기!

또 들여쓰기!

while 반복문

while 반복문은 조건식으로만 동작하며 반복할 코드 안에 조건식에 영향을 주는 변화식

사용방법
초기식
while 조건식:
    반복할 코드
    변화식

```
i = 0                    # 초기식
while i < 10:             # while 조건식
    print('펭하 ' )      # 반복할 코드₩
    i = i +1               # 변화식
```

For+리스트

```
a = [1,2,3,4]
result = []
for num in a:
    result.append(num*3)

print(result)
```

사용방법:
[표현식 for 항목 in 반복가능객체 if 조건]

```
result = [num * 3 for num in a]
print(result)
```

```
result = [num * 3 for num in a if num % 2 == 0]
print(result)
```

함수(Function)

반복적으로 계속 사용되는 일정한 작업을 수행하는 코드들의 모음

함수: 함수이름( ) 의 형식으로 사용

예: 함수 객체 확인하는 함수
globals()

함수의 입력과 출력

입력값없는 경우: 함수명()

입력값있는 경우: 함수명(입력값, 입력값2, ...)

출력값 있는 경우: 함수 내 return을 통해 반환

결과값 받는 변수 = 함수명()

- **함수 정의**
  - def 라는 예약어와 함께 정의
  - 함수의 사용은 함수명(입력값)의 형태

```
def Times(a,b):
        return a*b
```

  - 함수 객체 확인: globals()
  - return 없으면 None 반환

```
def setValue( newValue):
        x=newValue
retval = setValue(10)
print(retval)
```

모듈, 패키지, 라이브러리

모듈: 특정 기능을 .py 파일 단위로 작성

패키지: 특정 기능과 관련된 여러 모듈을 묶음. 패키지는 모듈에 네임스페이스(namespace, 이름공간)를 제공

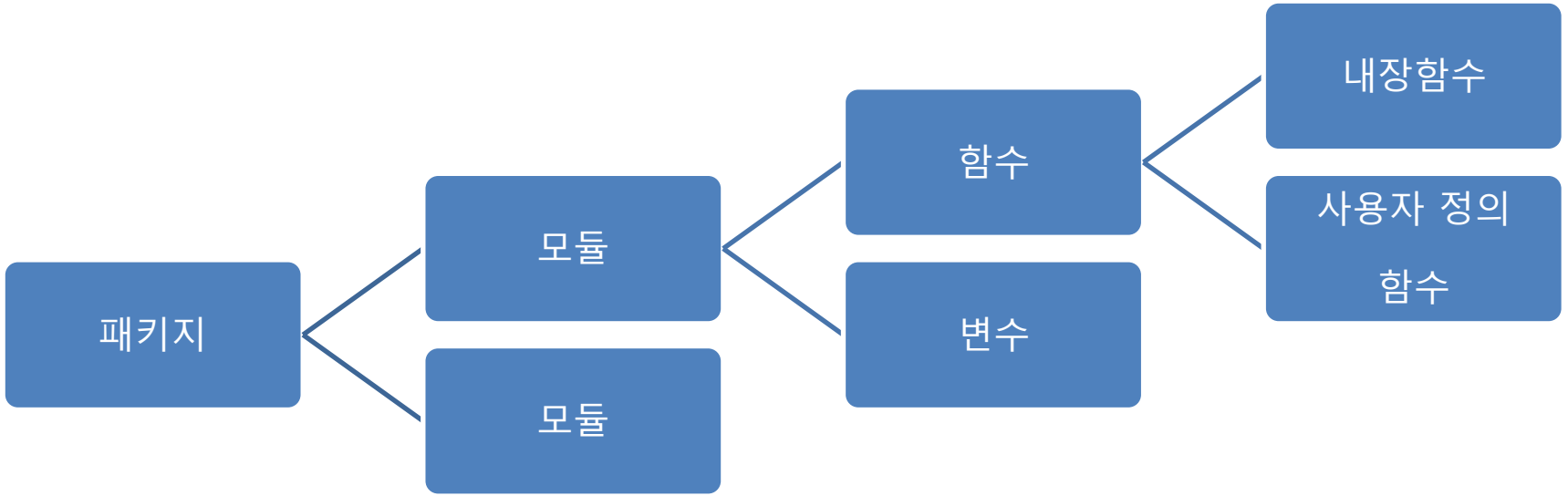
네임스페이스?

패키지1: 모듈1
패키지2: 모듈1

패키지1의 모듈1을 쓰고 싶다면!

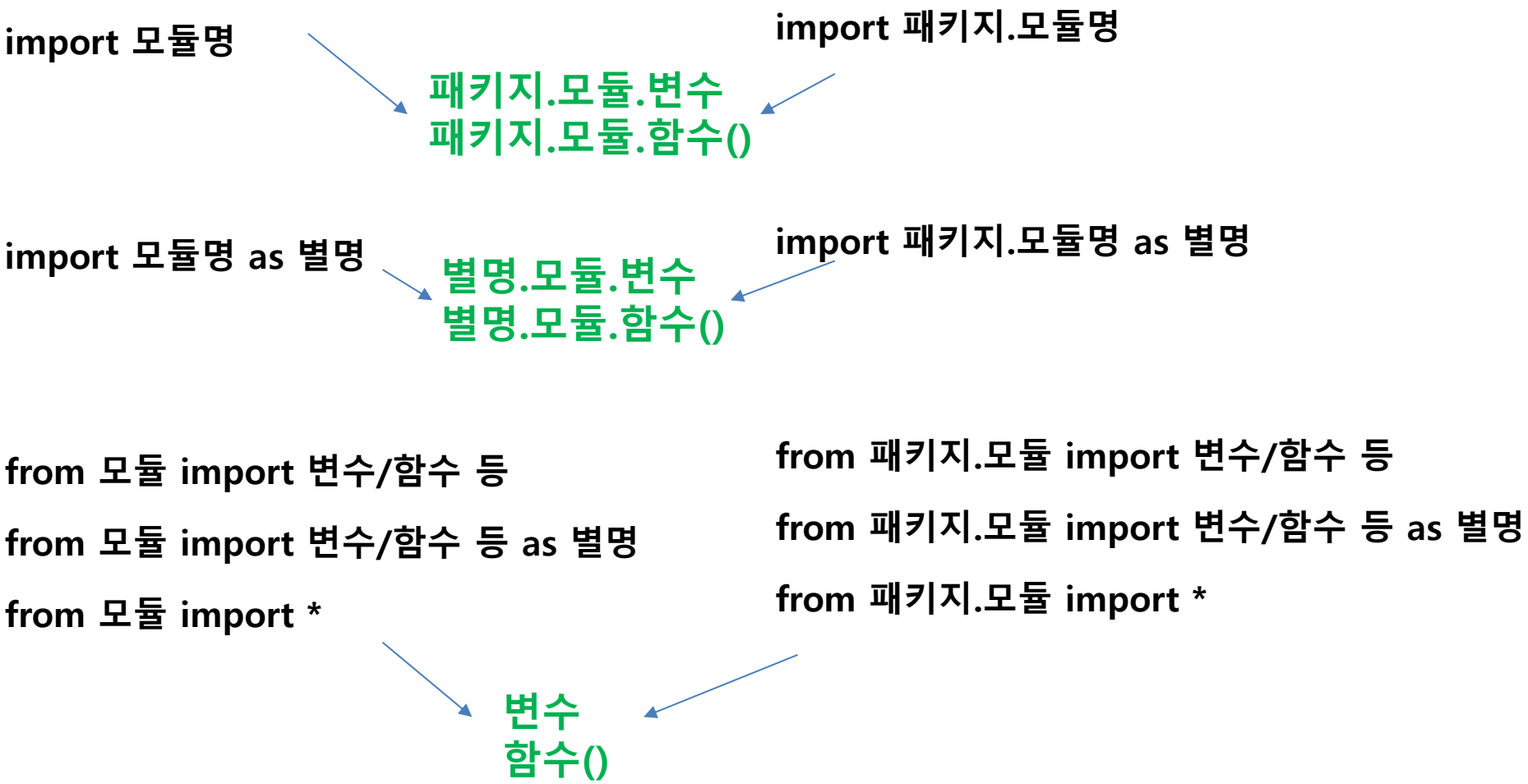
패키지1.모듈1

모듈, 패키지, 라이브러리



파이썬 표준 라이브러리: 파이썬에 기본으로 설치된 모듈과 패키지, 내장 함수를 통칭하여 파이썬 표준 라이브러리

모듈이나 패키지를 불러오는 3가지 방법

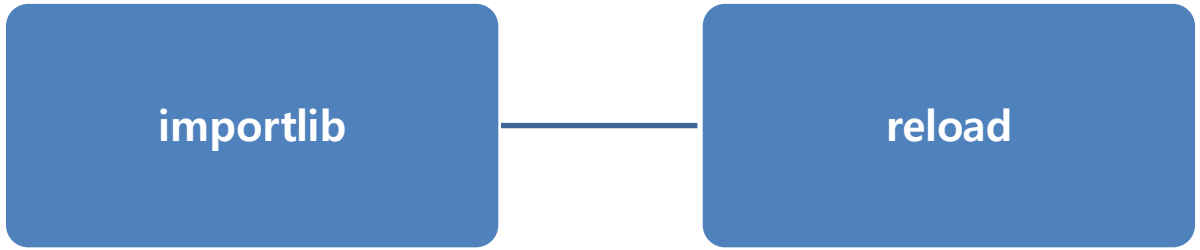


모듈 해제와 다시 불러오기

del 모듈

import importlib

importlib.reload(모듈)



모듈 설치하기

pip ?

파이썬 패키지 인덱스(Python Package Index, PyPI)의 패키지 관리 명령어

패키지 설치/삭제/업데이트 지원

쥬피터노트북에서 터미널-> pip install 패키지 (또는 모듈) 이름

예: pip install mlxtend

문자열(String)

따옴표, 또는 홑따옴표로 둘러싼 문자들의 집합

아무리 수치값이어도 문자열로 표현되면 더 이상 숫자가 아님!

"문자열은 어렵나요?"

'문자열은 사용하면 안되요'

'I can speak English'

'123'

파일 열고 값 출력

내장함수 open 사용

이름=open(파일명, 파일열기모드)

*이름은 Open한 파일의 실제 값을 나타내는 역할을 함(객체)

파일열기모드

1) r: 읽기
2) w: 쓰기(기존 내용을 무시!)
3) a: 추가하기(기존 내용에 추가!)

파일 열고 출력의 3단계

1. 파일 열고 값 출력

```
file1=open( "file1.txt", 'r')
file1=open( "file1.txt", 'w')    #해당 파일명이 없다면 파일이 생성됨
file1=open( "C"₩file1.txt", 'r')
```

2. 열린 파일에 값 쓰기

```
f.write(값)
```

3. 파일 열고 작업 후에는 파일을 닫기

```
file1.close()
```

파일 입력: 제공된 파일을 읽기

파일을 열은 후,

파일 객체.readline()

또는 파일 객체. readlines()를 이용

또는 파일 객체. read()를 이용

파일 읽기 3단계

1. **파일 열기 : r 모드**
2. **파일 내용 읽어서 할당**
3. **파일 닫기**

with 사용

```
with open( " 파일명 " ,  " w " ) as file1:
    file1.write( " 파일 닫기, 거참 번거롭구먼 허허")
```

예외 처리: 오류 발생 시 적용

try except 사용

```
try:
        실행할 구문(들)
except:
        실행할 구문
```

```
try:
        실행할 구문(들)
except 발생 오류 코드1:
        실행할 구문(들)
except 발생 오류 코드2:
        실행할 구문(들)
except 발생 오류 코드3:
        실행할 구문(들)
```

오류 그냥 넘어가기: pass

array

- 1차원, 2차원, 3차원, … 가능
- +,-,*,/ 계산 지원

참고. arange 함수: 일정한 범위내 일련의 정수를 갖는 array 생성

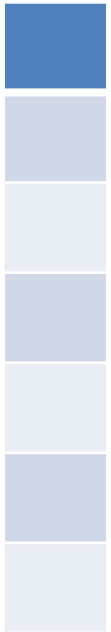
```
import numpy as np
arr4 = np.arange(10)          #0부터 9까지 값을 갖는 1차원 array
arr5 = np.arange(3,10)        #3부터 9까지 값을 갖는 1차원 array
```

array의 다차원!

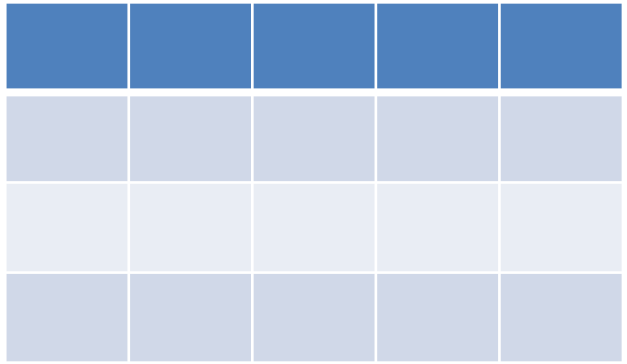
1차원 / 2차원 / 3차원

```
import numpy as np
arr4 = np.array( [1,2,3] )
arr5 = np.array( [ [1,2,3], [4,5,6] ] )
```

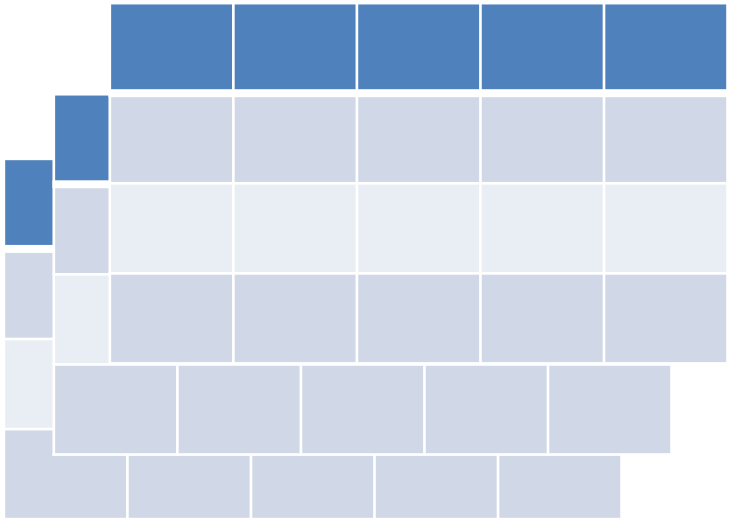
1차원



2차원



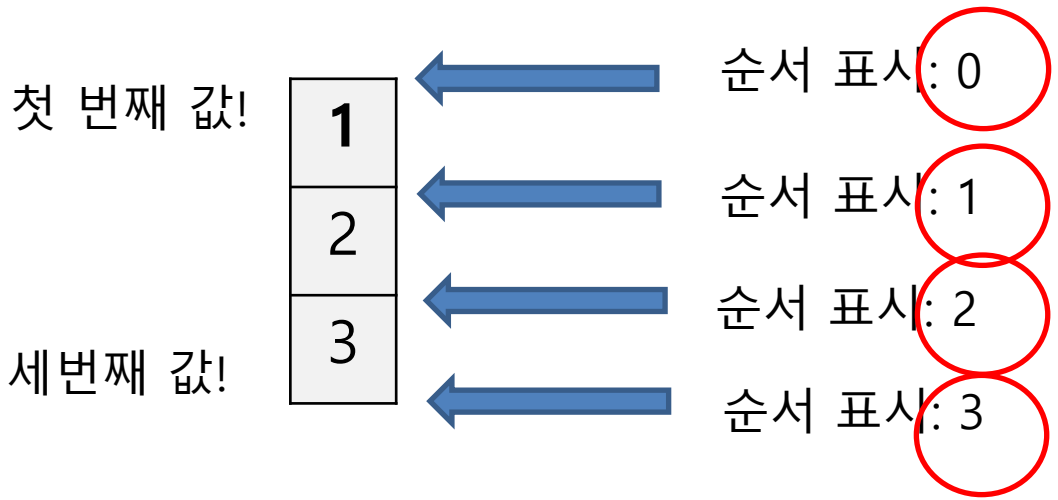
3차원



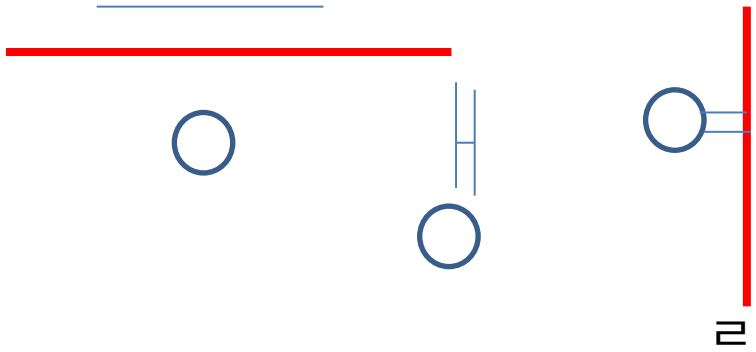
array 값들의 순서!

```
import numpy as np
arr4 = np.array( [1,2,3] )
arr4.tolist()    #array의 list 변환 함수!
```

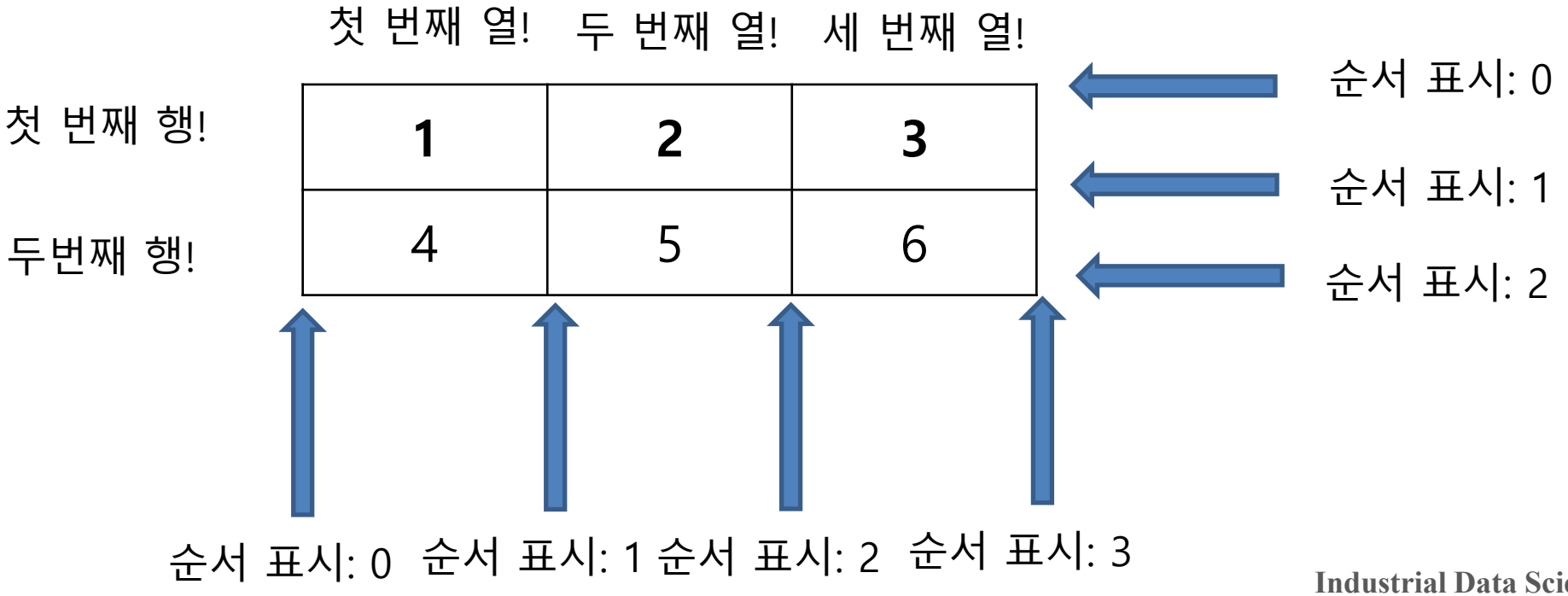
값의 순서가 있다!



array 값들의 순서!

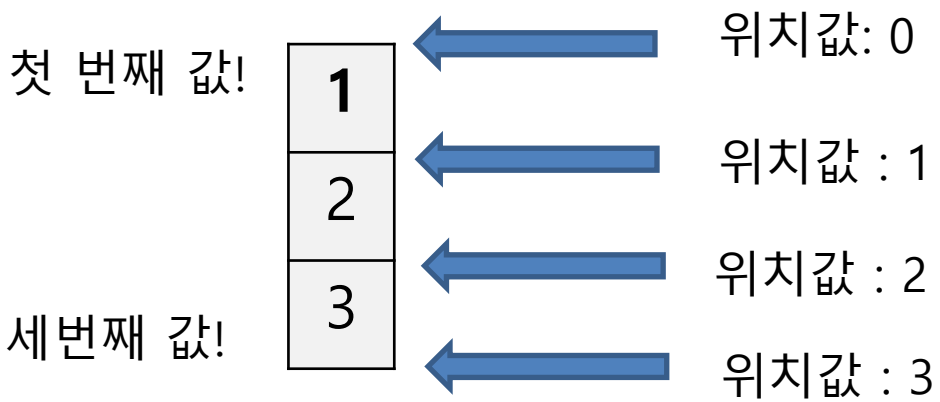


```
import numpy as np
arr5 = np.array( [ [1,2,3], [4,5,6] ] )
```



1차원 array의 Indexing

```
import numpy as np
arr4 = np.array( [1,2,3] )
```



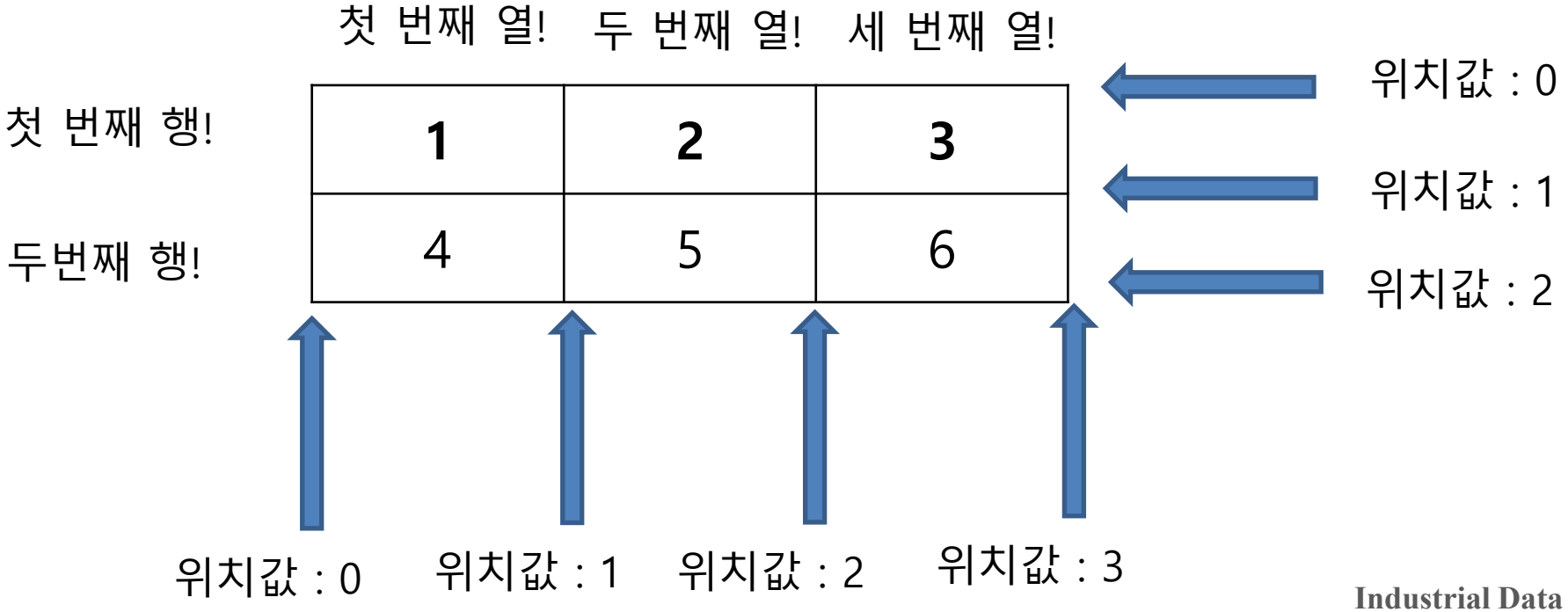
[ ]

arr4의 2번째 값! arr4[1]
1번째, 3번째 값! Arr4[[0,2]]

2차원 array의 Indexing

```
import numpy as np
arr5 = np.array( [ [1,2,3], [4,5,6] ] )
```

```
arr5[ 0 , :]   #첫번째 행
arr5[ : , 1] #두번째 열
arr5[ 0 , 1] #첫번째 행, 두번째 열
```



다차원 array에서의 Slicing

Slicing에 사용되는 연산자!

[ ]

Slicing에 : 의 사용!

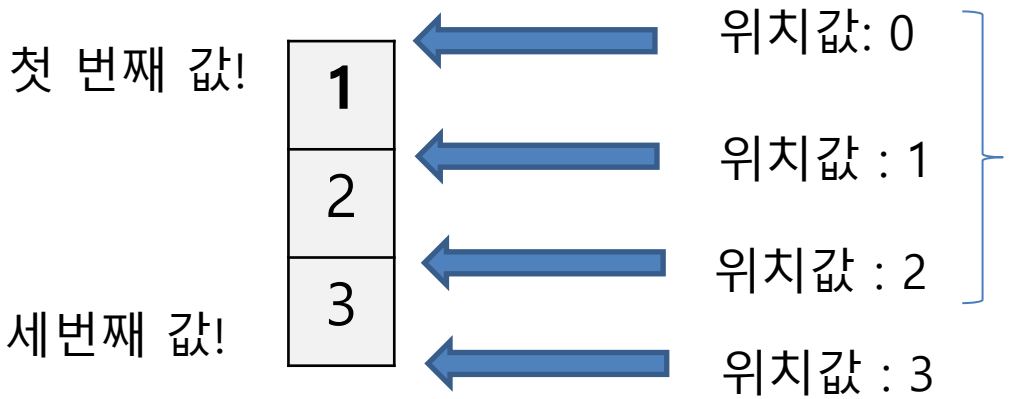
시작되는 위치:끝나는 위치

Slicing 사용 방식!

array이름[시작되는 위치값:끝나는 위치값]

1차원 array의 slicing

```
import numpy as np
arr4 = np.array( [1,2,3] )
```

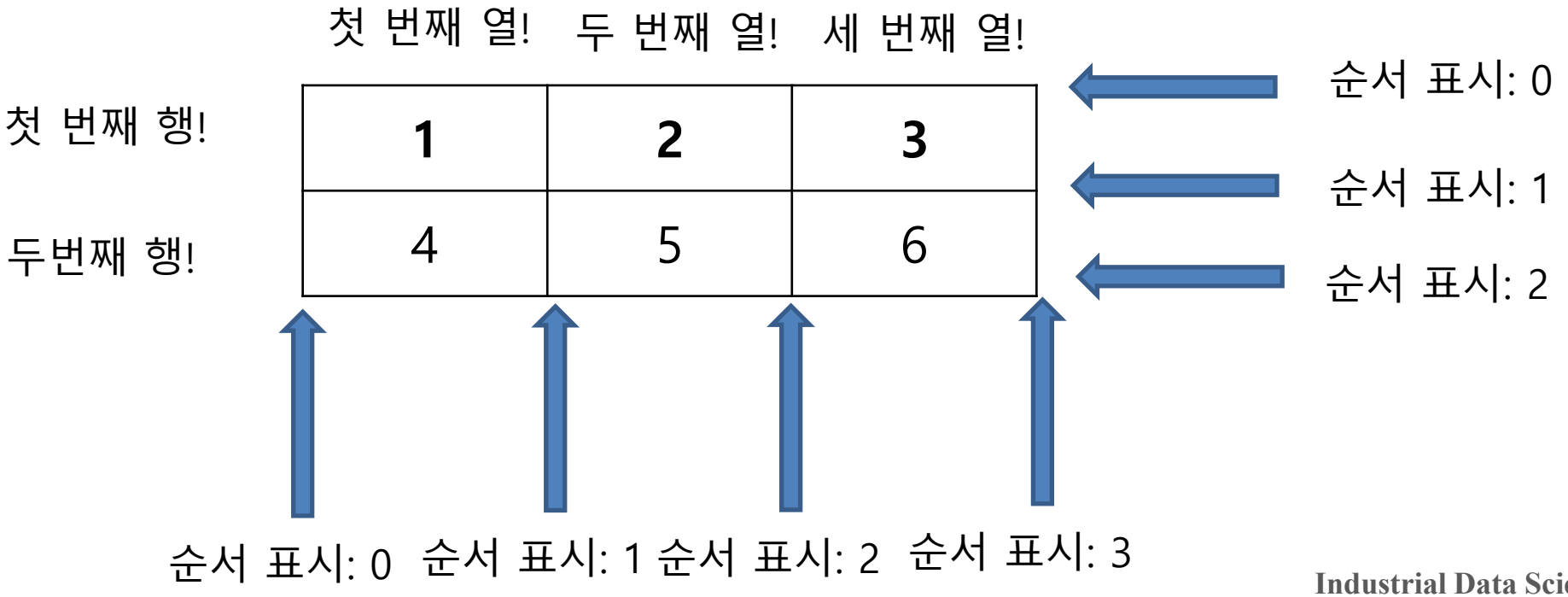


arr4의 1~2번째 값! arr4[0:2]
arr4의 1~2번째 값! arr4[0:3]

2차원 array의 Slicing

```
import numpy as np
arr5 = np.array( [ [1,2,3], [4,5,6] ] )
```

```
arr5[ 0:1 , 1:3]      #1행, 2,3열
arr5[ 0:2 , 1:2]      #1,2행, 2열
```



array

- array에 Scalar를 사칙연산이 가능

```
arr4 * 3
arr5 + 1
arr4 -3
arr5 / 2
```

계산 결과를 새로운 array에 할당(assignment)

```
arr6 = arr4*3
```

array간 연산(1차원)

- array 간 덧셈

arr8+arr9

- array 간 뺄셈

arr8-arr9

- array 간 곱셈

arr8*arr9

- array 간 나눗셈

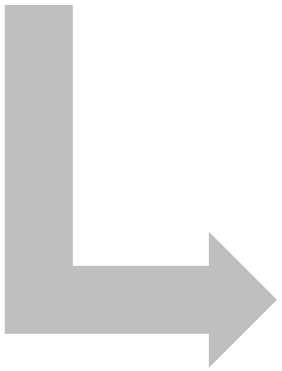
arr8/arr9

```
arr8.size
arr8.shape
arr8.ndim
```

주요 통계량 함수를 이용한 array 계산

통계량? 주어진 자료의 특성을 나타내는 요약값

```
import numpy as np
arr12 = np.array( [3,5,2,4] )
arr13 = np.array( [3.1,5.2,4.5] )
arr14 = np.array( [-2,3,1,10] )
```



예: 평균, 편차, 최대, 최소값 등

평균: np.mean(arr12)

표준편차: np.std(arr12)

최대: np.max(arr13)

최소: np.min(arr13)

주요 통계량 함수를 이용한 array 계산

그 외에도 다양한 수학함수를 지원

sum, prod, abs, sqrt, exp, log 등등...

절대값: np.abs(arr14)

제곱근: np.sqrt(arr12)

지수: np.exp(arr13)

로그: np.log(arr13)

지수? Exponential!

로그? Log!

e = 2.71828182846

pi와 e를 다음처럼 미리 정의
pi = 3.14159265359
e = 2.71828182846

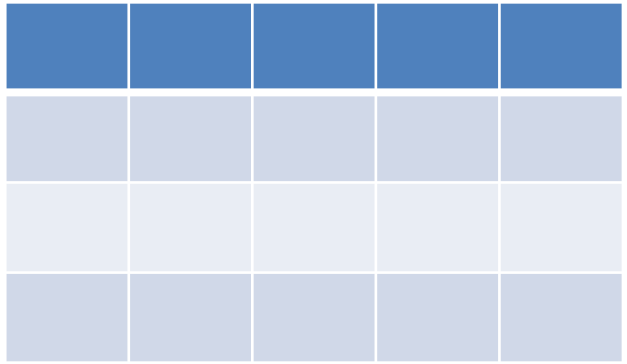
- **pandas**
  - 데이터 처리를 위한 최고의 패키지!
  - 데이터 처리 기능을 풍부하게 제공
  - 빠르고 직관적인 자료 구조!

- **특징**
  - DataFrame 제공
  - 엑셀, CSV 등을 포함한 다양한 포맷 지원
  - 데이터 정렬과 결측치 처리, 피벗 등 기능!
  - 고급 인덱싱과 슬라이싱
  - 데이터 합치기와 형태 변경 지원!

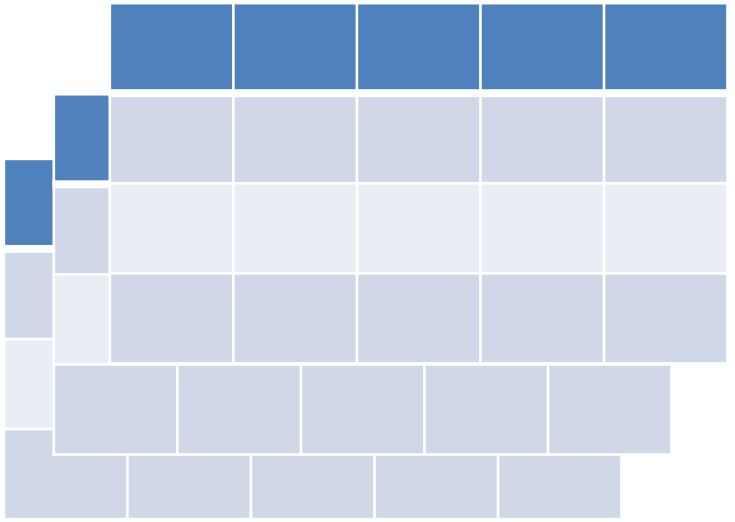
- **Pandas의 데이터 구조**
  - 데이터 구조
    - Series (1D)
    - DataFrame (2D)
    - Panel (3D)

Series

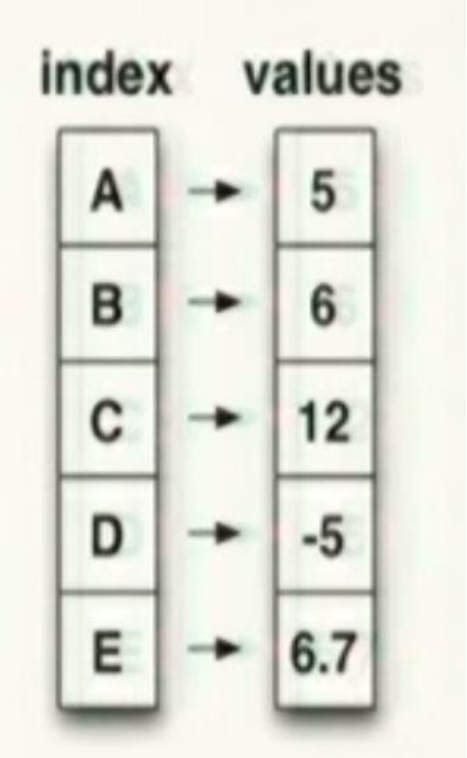
DataFrame



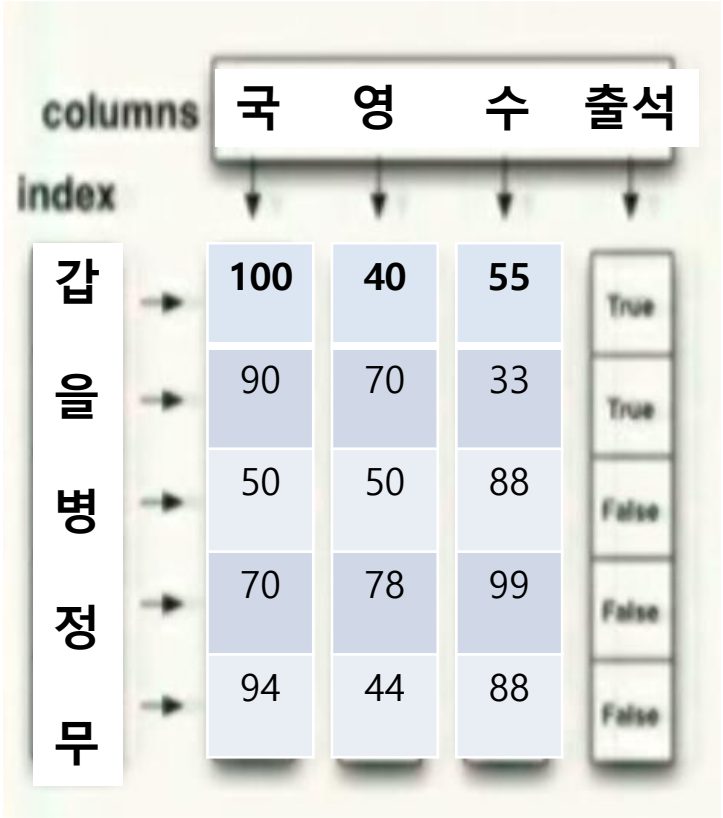
Panel



- **Series**
  - 1차원
  - Subclass of numpy.ndarray
  - Index, Values를 갖음
  - Index는 원하는 값을 넣을 수 있음-시계열



- **DataFrame**
  - 행과 열이 있는 2차원
  - 열마다 값의 타입이 다를 수 있음!
  - 행, 열 모두 인덱스
  - 크기가 바뀔 수 있음-행, 열 추가 및 삭제!

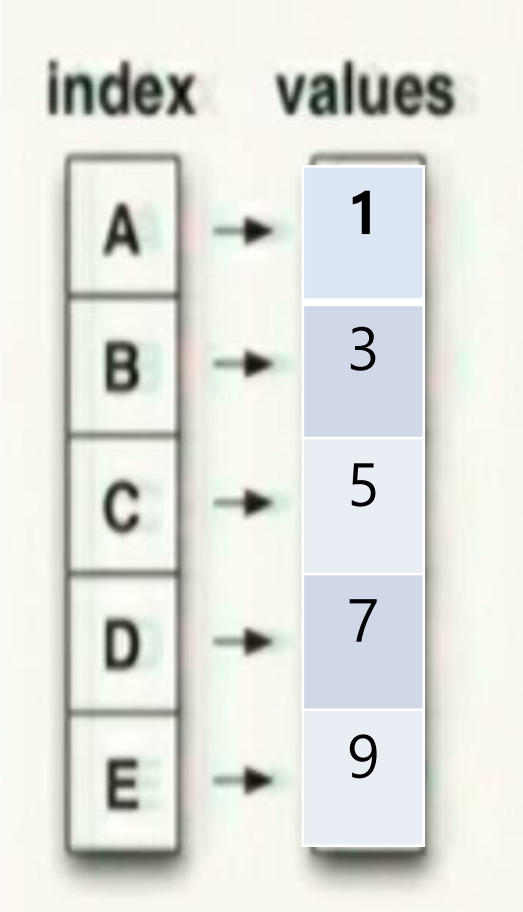


- **Series**
  - pandas에서 제공하는 1차원 자료 구조 , 연산, 인덱싱, 슬라이싱을 포함해 다양한 기능을 지원

- **Series의 인덱싱과 슬라이싱**
  - Series는 Index와 Value를 나누어 활용할 수 있으며, 이때 Index Label이나 Array를 이용한 인덱싱, 슬라이싱을 지원

- **Series 연산**
  - 수치 계산 및 벡터화 계산을 지원하며, Series 간 연산 시 같은 Index 값이 계산

- **Series**
  - 1차원
  - Index, Values를 갖음



- **Series 생성하기**
  - 1차원 시퀀스 형태의 자료구조(예: 리스트 array)로 부터 생성 가능
  - **Series = Value + Index**
  - Index Label: Index에 원하는 값을 지정할 수 있음

```
import numpy as np
import pandas as pd

S1 = pd.Series( [1,2,3] )

S2 = pd.Series( np.array([1,2,3]))

S3 = pd.Series( [80,90,75], index=["가","나","다"] )
```

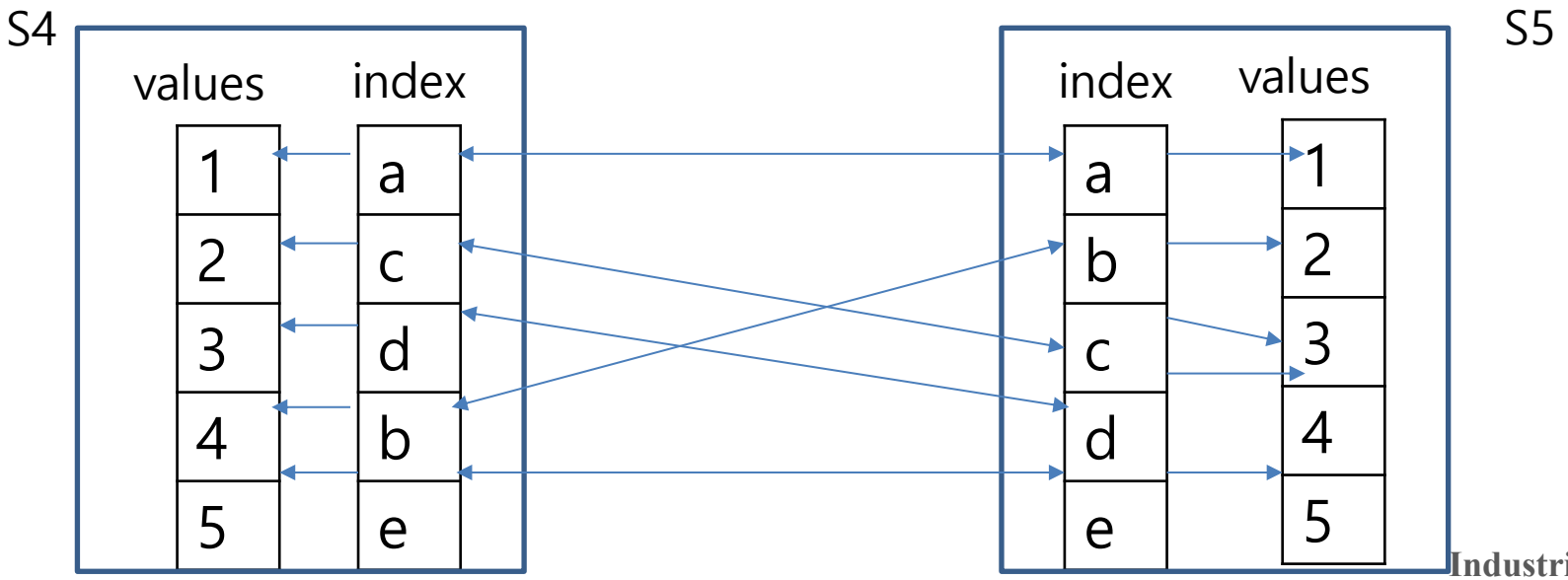
  - **Series명.values, 예를 들어 S1.values**
  - **Series명.index, 예를 들어 S3.index**

- **Series 인덱싱과 슬라이싱**
  - Index Label을 활용해서 원하는 범위의 값을 선택할 수 있음
    - Index가 지정 안 된 경우: 첫번째값이 0부터 시작하는 정수 index를 갖음
    - Index가 지정된 경우: 별도로 지정한 Index Label을 사용할 수 있으며, 첫번째값이 0부터 시작하는 정수 index도 사용할 수 있음

```
S4 = pd.Series( [1,2,3,4,5] )

S5 = pd.Series( [1,2,3,4,5], index=["a","b","c","d","e"] )

S4[0] # S4의 첫번째 값
S4["a"] # 오류, S4에는 별도로 지정된 index 없음

S5[0] # S5의 첫번째 값
S5["a"] # S5의 첫번째 값

S4[ 0:3 ] #S4의 1~4번째 값
S5[ ["a", "c", "e"] ] #S5의 1,3,5번째 값 출력
```

- **Series 연산**
  - Series는 수치 계산을 지원
    - +,-,*,/
  - Series 간 연산은 같은 Index 값끼리 계산

```
S4 = pd.Series( [1,2,3,4,5], index=["a","c","d","b","e"] )

S5 = pd.Series( [1,2,3,4,5], index=["a","b","c","d","e"] )

S4+S5
```

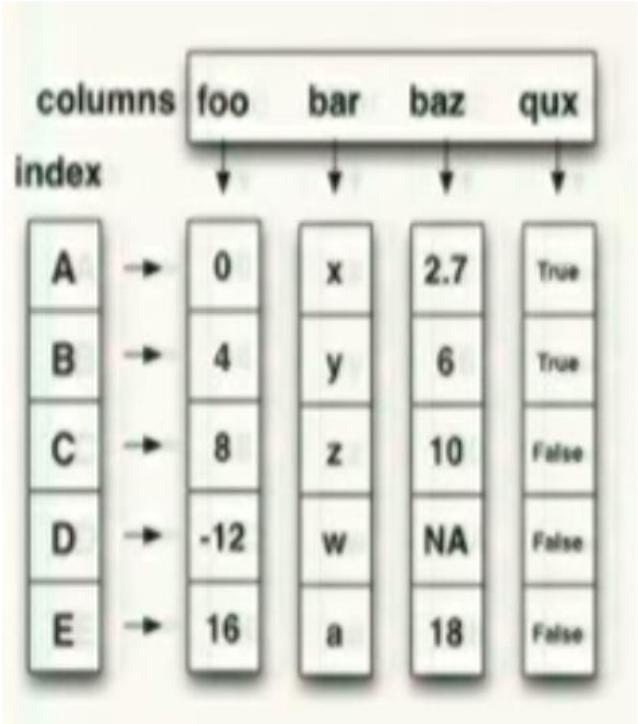


- **DataFrame**
  - pandas에서 제공하는 2차원 자료 구조, 연산, 인덱싱, 슬라이싱을 포함해 다양한 기능을 지원

- **DataFrame의 인덱싱과 슬라이싱**
  - DataFrame은 행은 슬라이싱을, 열은 인덱싱을 지원할 수 있으며, 추가적인 함수를 통해 인덱싱과 슬라이싱을 자유롭게 사용 가능.

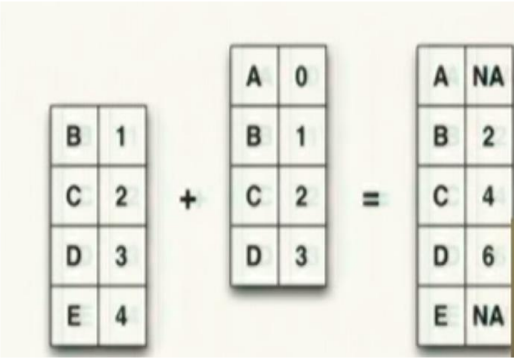
- **DataFrame 활용**
  - 수치 계산 및 벡터화 계산을 지원하며, 자료처리를 위한 다양한 함수를 활용할 수 있음

- **DataFrame의 특징!**
  - 2D table with rows and column labels
  - column 별로 서로 다른 dtype을 가질 수 있음
  - 행과 열 Index Label을 지원



- **DataFrame의 더 많은 특징!**
  - 데이터의 reshaping 및 pivoting
  - Label활용 슬라이싱과 인덱싱, 부분집합 구하기가 쉬움
  - Group by 엔진 지원: Data Aggregation/ Data Transformation

- **Data Alignment**
  - "Outer join"을 지원: 두 데이터프레임의 행 또는 열 Index를 기준으로 계산!



- **DataFrame 생성하기**
  - 2차원 형태의 자료구조(예: 리스트의 리스트, 2차원 array)와 Dictionary(Value가 1차원 자료구조인 경우)로 부터 생성 가능
  - **DataFrame = Series + Series + Series ….**
  - 행과 열의 Index Label: Index에 원하는 값을 지정할 수 있음!

```
import numpy as np
import pandas as pd

df1 = pd.DataFrame( [[1,2,3], [1,2,3]], columns=['a','b'], index=['r1','r2','r3'] )
df2 = pd.Series( np.array([[1,2,3], [1,2,3]]) , columns=['a','b'],
index=['r1','r2','r3'])
raw_data = {'col0': [1, 2, 3, 4],
            'col1': [10, 20, 30, 40],
            'col2': [100, 200, 300, 400]}
df3 = pd.Series( raw_data )
```

  - **DF명.values, 예를 들어 df1.values**
  - **DF명.index, 예를 들어 df1.index**
  - **DF명.columns, 예를 들어 df1.columns**

- **DataFrame의 인덱싱과 슬라이싱**
  - 인덱싱: 열 이름을 지정, 복수인 경우 리스트로 묶어서 지정
  - 슬라이싱: 선택을 원하는 행을 슬라이싱 / 단 행 단위에 인덱싱 사용 불가

```
#행
df1[ 0 ]  #오류
df1[ 0:1 ]

#열
df1['a']
df1[['a','c']]
```

- **DataFrame 활용**
  - .head(), .describe() 등의 유용한 기능 활용
  - 두 데이터프레임의 수치 계산은 같은 인덱스의 해당되는 컬럼끼리 계산!
    - +,-,*,/

```
df3 = pd.DataFrame( [[1,2,3], [1,2,3]], columns=['a','b','c'], index=['r2','r3'] )
df1+df3
```