

# DEEP LEARNING WITH PYTORCH



# ABOUT PYTORCH

# ABOUT PYTORCH

## ◆ 오픈 프레임워크 비교

	장점	단점
텐서플로 (Tensorflow)	<ul style="list-style-type: none"><li>• 텐서보드 통해 파라미터 변화 양상 및 구조 확인 가능</li></ul>	<ul style="list-style-type: none"><li>• 메모리 효율적으로 사용하지 못함</li></ul>
케라스 (Keras)	<ul style="list-style-type: none"><li>• 배우기 쉽고 모델 구축 쉬움</li></ul>	<ul style="list-style-type: none"><li>• 오류 발생할 경우 케라스 문제인지 텐서플로우 문제인지 알 수 없음</li></ul>
파이토치 (Pytorch)	<ul style="list-style-type: none"><li>• 간단하고 직관적으로 학습 가능</li><li>• 속도 대비 빠른 최적화 가능</li></ul>	<ul style="list-style-type: none"><li>• 텐서플로우 비해 사용자층이 얇음</li><li>• 예제 및 자료 구하기 힘들</li></ul>

# ABOUT PYTORCH

---

## ◆ Pytorch

- ❖ 페이스북 AI 리서치 랩 (FAIR)에서 2016년 9월 최초 출시 루아(Lua)언어로 개발
- ❖ 파이썬의 넘파이(NumPy) 라이브러리처럼 **과학 연산 위한 라이브러리로 공개**
- ❖ 이후 발전을 거듭하면서 **딥러닝 프레임워크로 발전**
- ❖ **Numpy와 유사한 구조**
- ❖ 동적계산그래프(Define by Run)
- ❖ GPU사용 가능해서 속도가 상당히 빠름

# ABOUT PYTORCH

---

## ◆ Pytorch 개발환경 구축

### ❖ 버전

**Stable 버전** : 테스트 및 지원되고 있는 가장 최근 PyTorch 버전

**Preview 버전** : 아직 완전히 테스트/지원 되지 않는 최신 버전. 매일 밤 업데이트

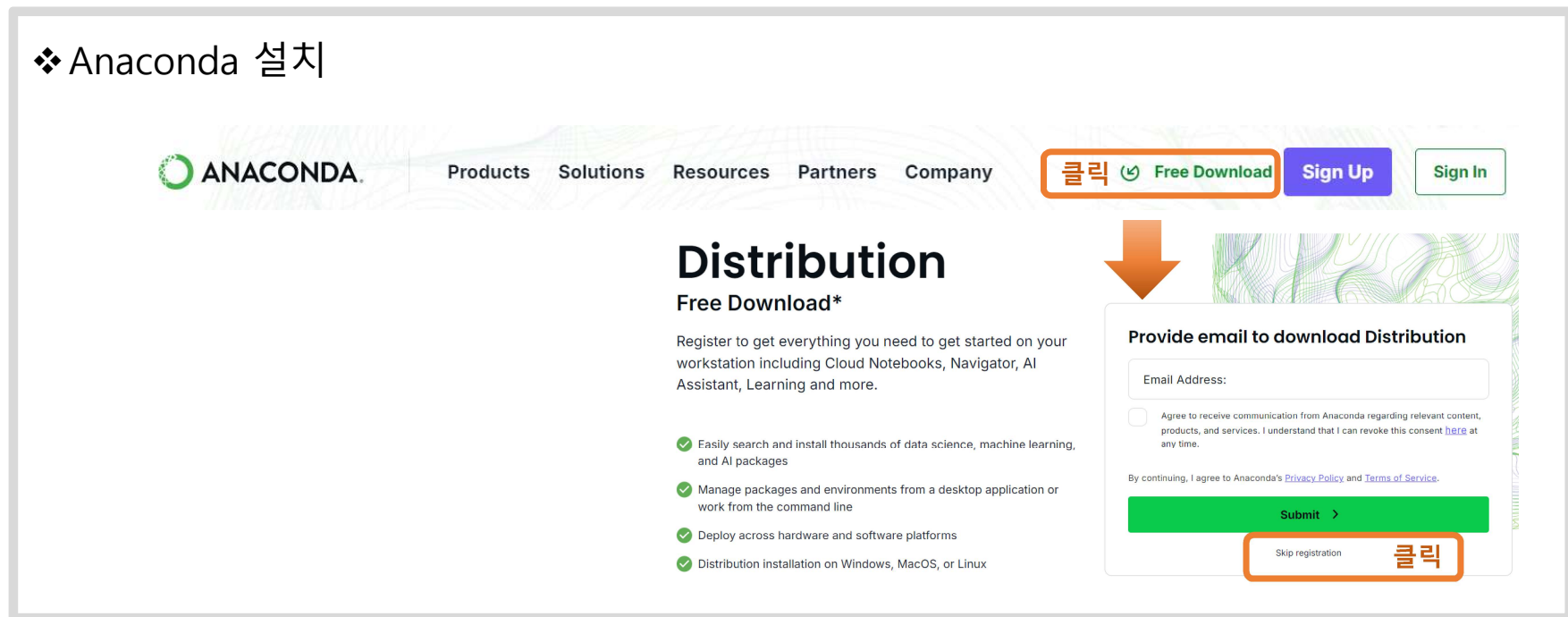
### ❖ 의존성 패키지 버전

**Python** : 3.8 이상

# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

### ❖ Anaconda 설치



The screenshot shows the Anaconda website's 'Distribution' page. At the top, there is a navigation bar with the Anaconda logo and links for Products, Solutions, Resources, Partners, and Company. A 'Free Download' button is highlighted with an orange box and the Korean text '클릭' (Click). Below the navigation bar, the 'Distribution' section is titled 'Free Download\*'. It includes a paragraph about registering to get everything needed to get started on a workstation. A list of benefits is provided, each with a green checkmark. To the right, a registration form titled 'Provide email to download Distribution' is shown. It has an 'Email Address' input field, a checkbox for agreeing to communication, and a 'Submit' button. The 'Submit' button is highlighted with an orange box and the Korean text '클릭' (Click). Below the 'Submit' button is a 'Skip registration' link.

ANACONDA

Products Solutions Resources Partners Company

클릭 Free Download Sign Up Sign In

## Distribution

Free Download\*

Register to get everything you need to get started on your workstation including Cloud Notebooks, Navigator, AI Assistant, Learning and more.

- ✓ Easily search and install thousands of data science, machine learning, and AI packages
- ✓ Manage packages and environments from a desktop application or work from the command line
- ✓ Deploy across hardware and software platforms
- ✓ Distribution installation on Windows, MacOS, or Linux

Provide email to download Distribution

Email Address:

☐ Agree to receive communication from Anaconda regarding relevant content, products, and services. I understand that I can revoke this consent [here](#) at any time.

By continuing, I agree to Anaconda's [Privacy Policy](#) and [Terms of Service](#).

Submit >

Skip registration

클릭

# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

❖ Anaconda 설치

**Download Now**  
For installation assistance, refer to [Troubleshooting](#).

Download Distribution by choosing the proper installer for your machine.

**Download** 클릭

**Anaconda Installers**

Windows	Mac	Linux
<b>Python 3.12</b> 64-Bit Graphical Installer (912.3M)	<b>Python 3.12</b> 64-Bit (Apple silicon) Graphical Installer (704.7M) 64-Bit (Apple silicon) Command Line Installer (707.3M) 64-Bit (Intel chip) Graphical Installer (734.7M) 64-Bit (Intel chip) Command Line Installer (731.2M)	<b>Python 3.12</b> 64-Bit (x86) Installer (1007.9M) 64-Bit (AWS Graviton2 / ARM64) Installer (800.6M) 64-bit (Linux on IBM Z & LinuxONE) Installer (425.8M)

↓

최근 다운로드 기록

Anaconda3-2024.06-1-Windows-x86\_64.exe  
912MB • 완료

클릭

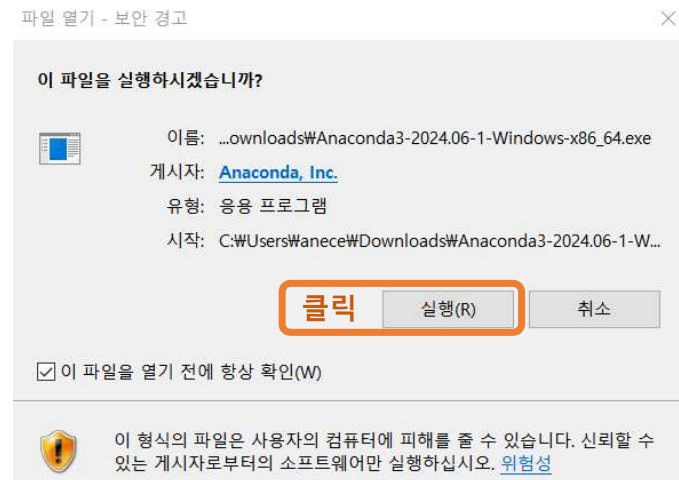
클릭

# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

### ❖ Anaconda 설치

Anaconda3-2024.06-1-Windows-x86\_64.exe **더블클릭**

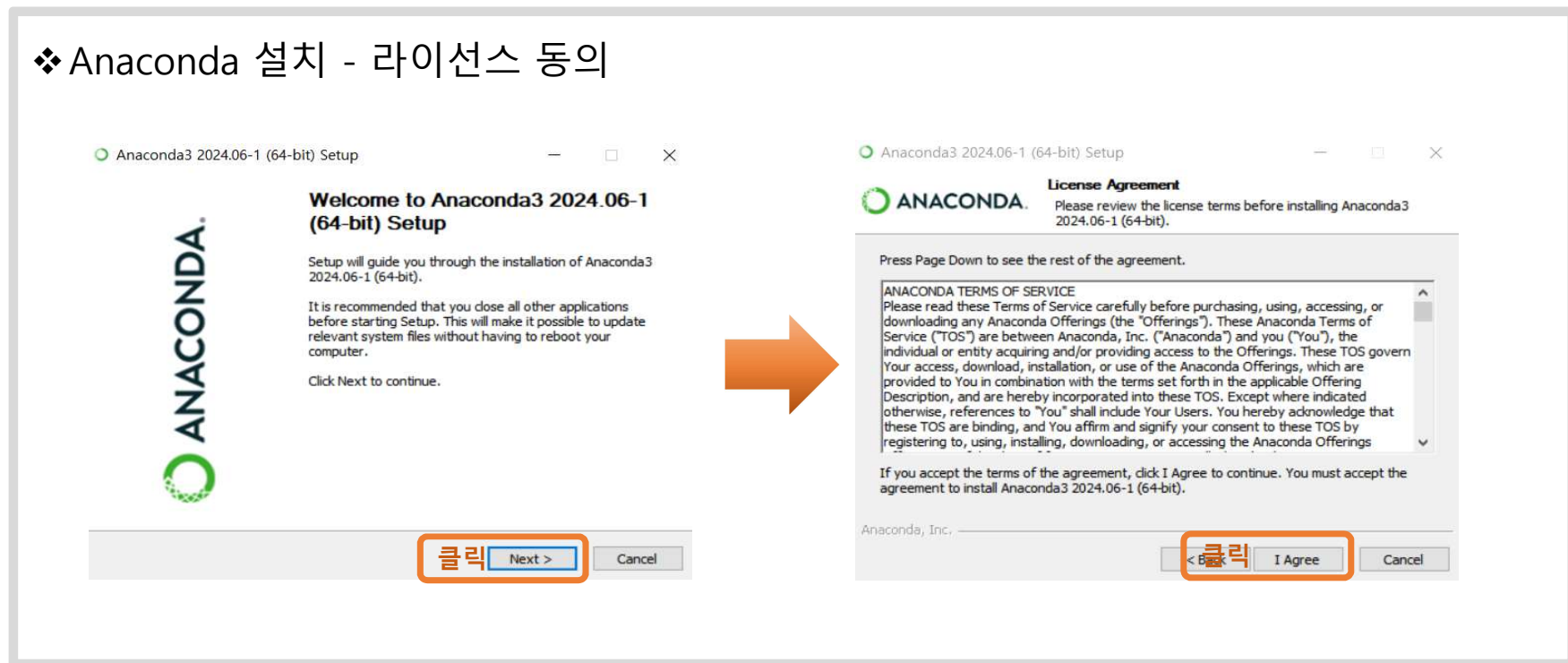




# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

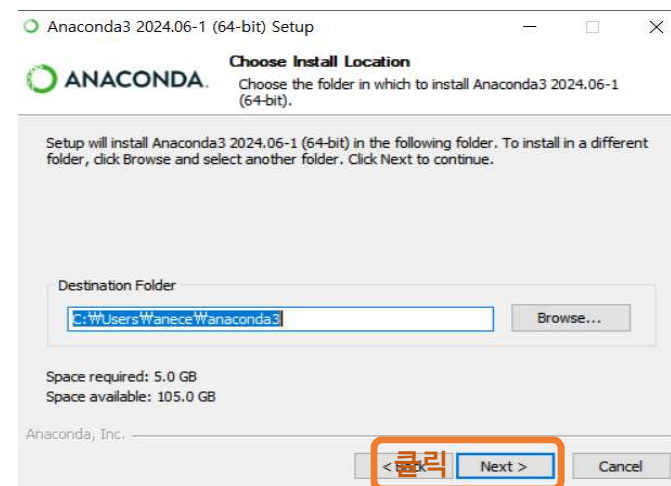
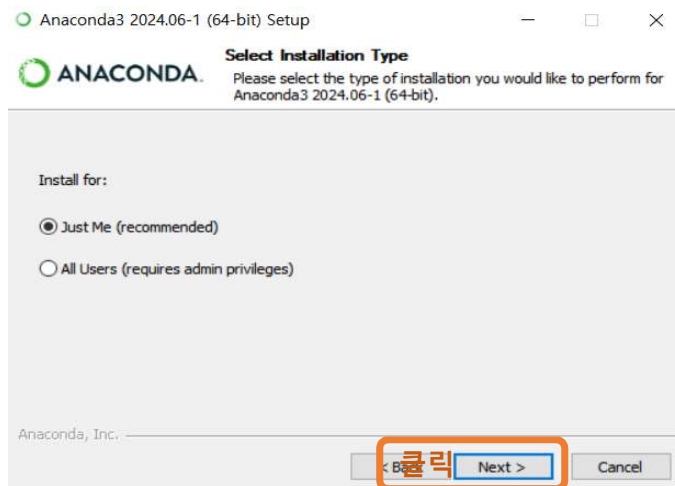
### ❖ Anaconda 설치 - 라이선스 동의



# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

❖ Anaconda 설치 - 설치 타입 및 위치 설정 → **Just Me(recommended)**



# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

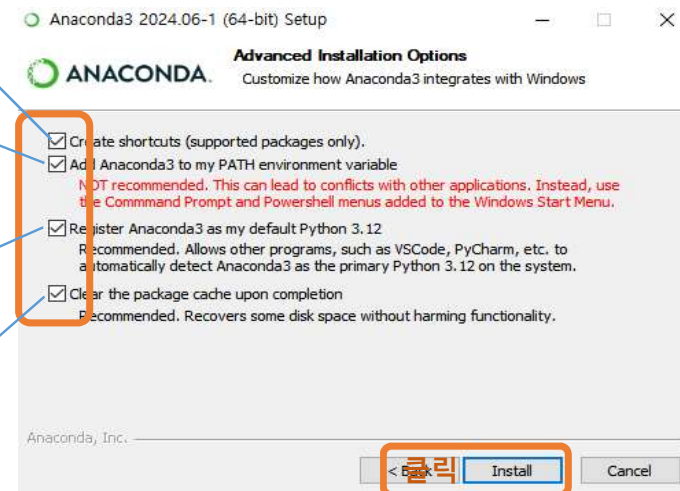
### ❖ Anaconda 설치 - 옵션 설정 및 설치

시작 메뉴에 바로가기를 추가

- Anaconda를 시스템의 PATH 환경 변수에 추가
- 터미널, 명령 프롬프트에서 'conda', 'python' 명령어 사용

Anaconda를 기본 파이썬 해석기로 등록

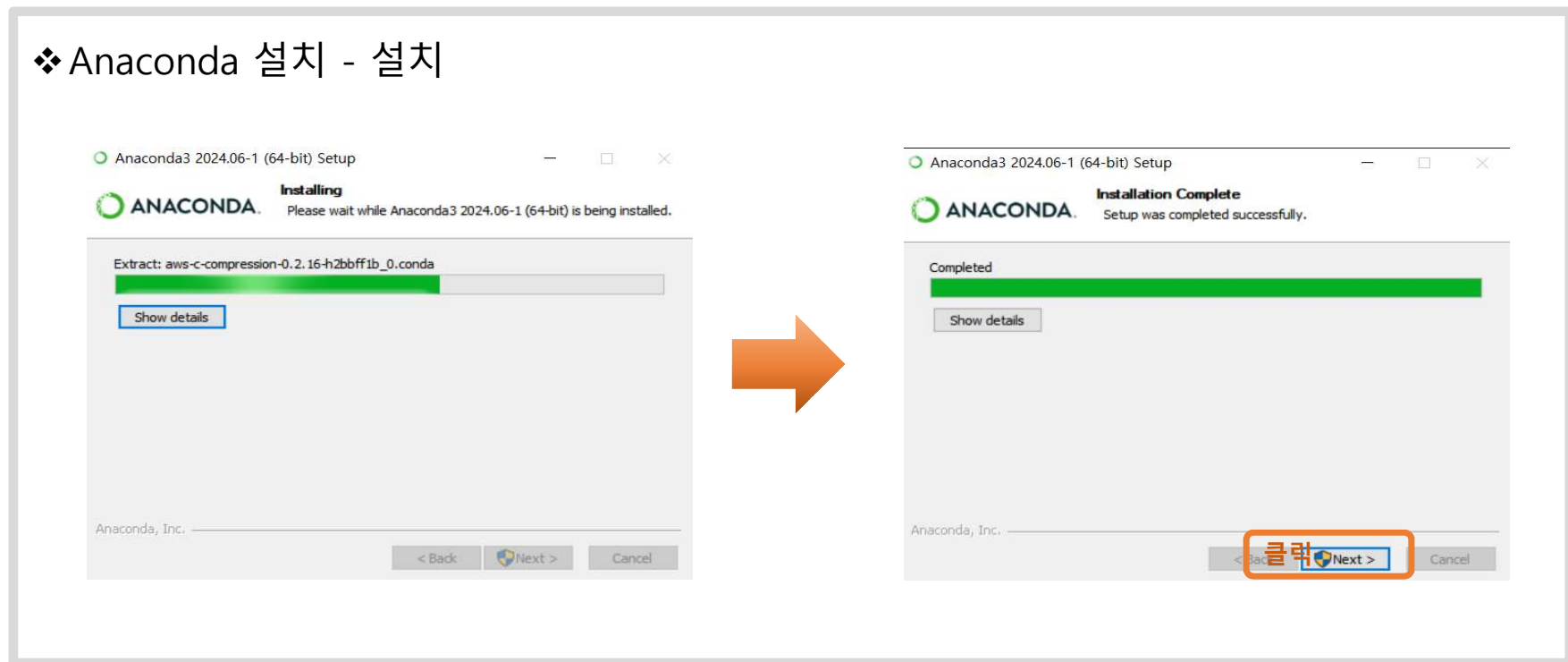
- 설치 완료 시 패키지 캐시를 삭제
- 이미 파이썬이 설치되어 있는 경우 체크



# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

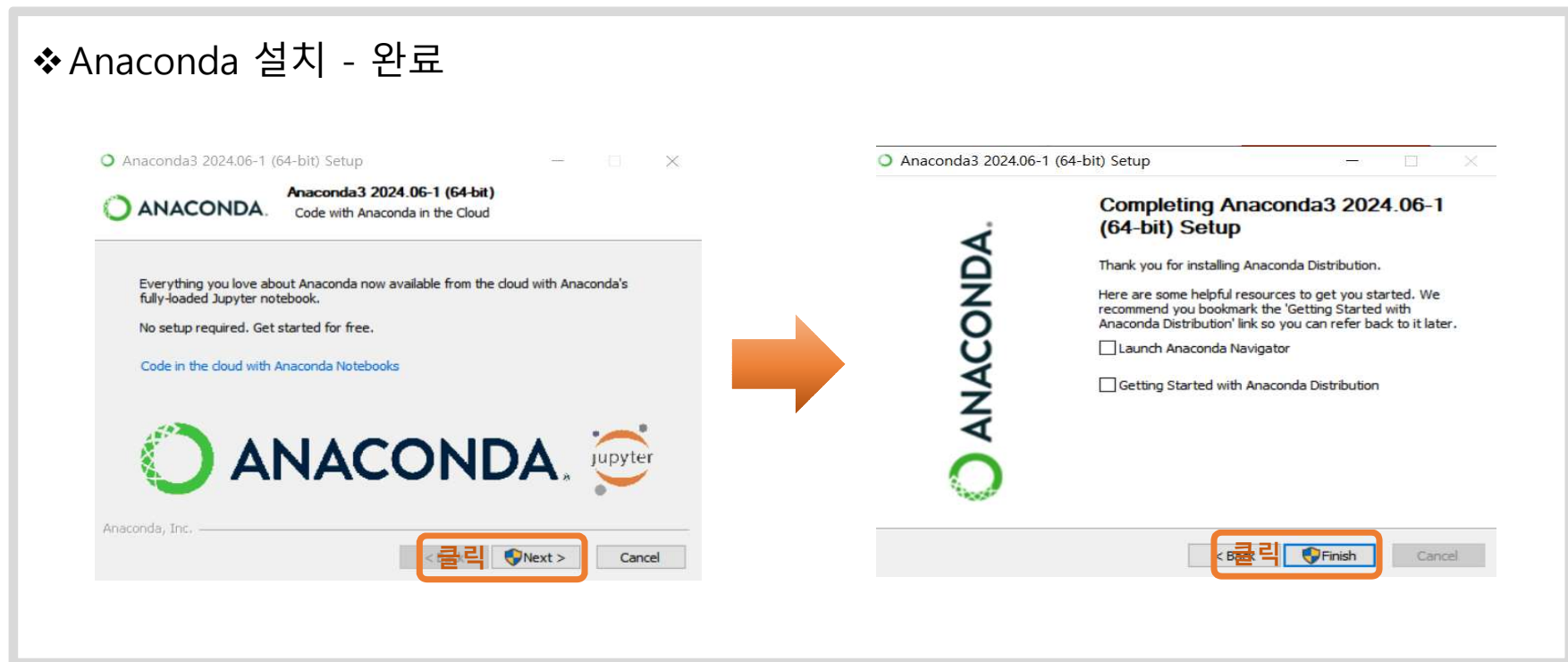
### ❖ Anaconda 설치 - 설치



# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

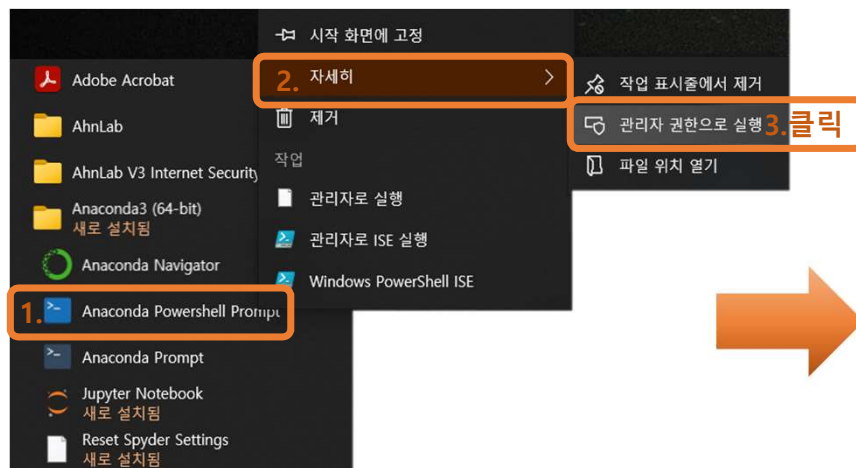
### ❖ Anaconda 설치 - 완료



# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

### ❖ Anaconda 가상환경 구축



현재 존재하는 가상환경 확인  
> conda env list

```
관리자: Anaconda Powershell Prompt
(base) PS C:\Users\Wanece> conda env list
# conda environments:
#
base                    * C:\Users\Wanece\anaconda3

(base) PS C:\Users\Wanece>
```

# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

❖ Anaconda 가상환경 구축 → 파이토치용 가상환경 생성

생성 명령어 > `conda create -n TORCH38 python=3.8`

관리자: Anaconda Powershell Prompt

```
(base) PS C:\Users\Wanece> conda env list
# conda environments:
#
base                  * C:\Users\Wanece\anaconda3

(base) PS C:\Users\Wanece> conda create -n TORCH38 python=3.8
```

# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

❖ Anaconda 가상환경 구축 → 파이토치용 가상환경 생성

관리자: Anaconda Powershell Prompt

The following NEW packages will be INSTALLED:

ca-certificates	pkgs/main/win-64::ca-certificates-2024.7.2-haa95532_0
libffi	pkgs/main/win-64::libffi-3.4.4-hd77b12b_1
openssl	pkgs/main/win-64::openssl-3.0.14-h827c3e9_0
pip	pkgs/main/win-64::pip-24.0-py38haa95532_0
python	pkgs/main/win-64::python-3.8.19-h1aa4202_0
setuptools	pkgs/main/win-64::setuptools-69.5.1-py38haa95532_0
sqlite	pkgs/main/win-64::sqlite-3.45.3-h2bbff1b_0
vc	pkgs/main/win-64::vc-14.2-h2eaa2aa_4
vs2015_runtime	pkgs/main/win-64::vs2015_runtime-14.29.30133-h43f2093_4
wheel	pkgs/main/win-64::wheel-0.43.0-py38haa95532_0

의존성 패키지 설치

Proceed ([y]/n) **y**



# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

❖ Anaconda 가상환경 구축 → 파이토치용 가상환경 생성

생성확인 : conda env list

실 행 : conda activate TORCH38

```
관리자: Anaconda Powershell Prompt
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate TORCH38
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) PS C:\Users\Wanece> conda env list
# conda environments:
#
base                    * C:\Users\Wanece\Anaconda3
TORCH38                 C:\Users\Wanece\Anaconda3\envs\TORCH38

(base) PS C:\Users\Wanece> conda activate TORCH38
(TORCH38) PS C:\Users\Wanece>
```

# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

❖ Anaconda 가상환경 구축 → CPU 기반 PyTorch 설치

PyTorch Build	Stable (2.4.0)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.1	CUDA 12.4	CPU
Run this Command:	<code>conda install pytorch torchvision torchaudio cpuonly -c pytorch</code>			

# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

❖ Anaconda 가상환경 구축 → GPU 기반 PyTorch 설치

PyTorch Build	Stable (2.4.0)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.1	CUDA 12.4	ROCm 6.1
Run this Command:	<pre>conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c pytorch -c nvidia</pre>			

# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

❖ Anaconda 가상환경 구축 → CPU 기반 PyTorch 설치

관리자: Anaconda Powershell Prompt

```
(TORCH38) PS C:\Users\Wanece>  
(TORCH38) PS C:\Users\Wanece> conda install pytorch torchvision torchaudio cpuonly -c pytorch
```

관리자: Anaconda Powershell Prompt

```
tbb                pkgs/main/win-64::tbb-2021.8.0-h59b6b97_0  
torchaudio         pytorch/win-64::torchaudio-2.4.0-py38_cpu  
torchvision        pytorch/win-64::torchvision-0.19.0-py38_cpu  
typing_extensions  pkgs/main/win-64::typing_extensions-4.11.0-py38haa95532_0  
urllib3            pkgs/main/win-64::urllib3-2.2.2-py38haa95532_0  
win_inet_pton       pkgs/main/win-64::win_inet_pton-1.1.0-py38haa95532_0  
xz                 pkgs/main/win-64::xz-5.4.6-h8cc25b3_1  
yaml               pkgs/main/win-64::yaml-0.2.5-he774522_0  
zlib               pkgs/main/win-64::zlib-1.2.13-h8cc25b3_1  
zstd               pkgs/main/win-64::zstd-1.5.5-hd43e919_2
```

의존성 패키지 설치

Proceed ([y]/n)? y

# ABOUT PYTORCH

## ◆ Pytorch 개발환경 구축

❖ Anaconda 가상환경 구축 → CPU 기반 PyTorch 설치

설치 완료

관리자: Anaconda Powershell Prompt

```
done  
(TORCH38) PS C:\Users\Wanece>  
(TORCH38) PS C:\Users\Wanece>
```

설치 확인

관리자: Anaconda Powershell Prompt

```
(TORCH38) PS C:\Users\Wanece> python  
Python 3.8.19 (default, Mar 20 2024, 19:55:45) [MSC v.1916 64 bit (AMD64)]  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import torch  
>>> torch.__version__  
'2.4.0'  
>>>
```

# ABOUT PYTORCH

---

## ◆ Pytorch 패키지 구성

- `torch` : 신경망 구성 기본 모듈, 텐서 생성, 과학 수치 계산 기능
- `torch.autograd` : 자동미분 기능 제공 모듈
- `torch.nn` : 인공신경망 관련 모듈
- `torch.multiprocessing` : 병렬처리 기능 제공 모듈
- `torch.utils` : 데이터 조작 및 유틸 기능 제공 모듈
- `torch.legacy` : `./nn/.optim` 포팅해온 코드 관련 모듈
- `torch.onnx` : Open Neural Network Exchange 다른 프레임워크간 모델 공유

# ABOUT PYTORCH

---

## ◆ Pytorch GPU 가용 여부 체크

❖ Pytorch의 GPU 활용 가능 체크

```
import torch

if torch.cuda.is_available():
    print(f'Current Device      : {torch.cuda.current_device()}')
    print(f'Device Name        : {torch.cuda.get_device_name()}')
    print(f'Device Capability    : {torch.cuda.get_device_capability()}')
else:
    print("No GPU")
```



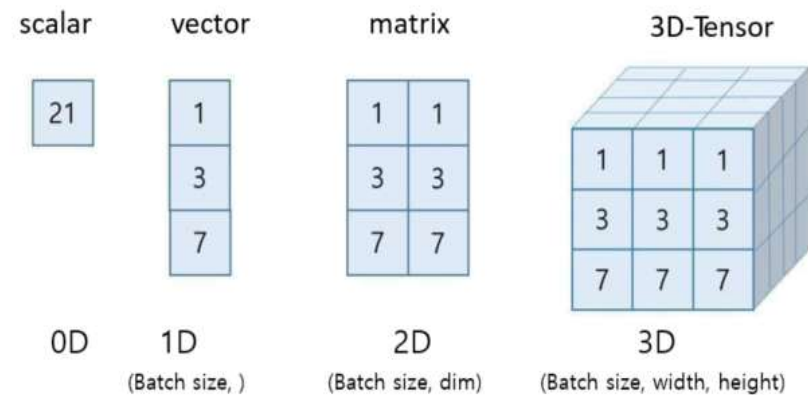
# ABOUT TENSOR



# ABOUT TENSOR

## ◆ 텐서(Tensor) 살펴보기

- 다차원 데이터를 담는 수학 객체
- 모델 입력(Input)과 출력(Output) 사용



# ABOUT TENSOR

---

## ◆ 텐서(Tensor) 살펴보기

### ❖ 학습데이터와 텐서 형태

→ 시계열 데이터	: 3D Tensor	shape(samples, timesteps, feature)
→ 이미지 데이터	: 3D Tensor	shape(samples, height, width)
→ 이미지 데이터	: 4D Tensor	shape(samples, height, width, channels)
→ 비디오 데이터	: 5D Tensor	shape(samples, frames, height, width, channels)

# ABOUT TENSOR

---

## ◆ 텐서(Tensor) 살펴보기

### ❖ Tensor 객체

- 속성/필드/attribute/property
- 기능/역할/function/method

### ❖ 속성 및 메서드 사용법

- Tensor객체변수명.속성
- Tensor객체변수명.메서드( )

# ABOUT TENSOR

---

## ◆ 텐서(Tensor) 살펴보기

### ❖ 기본 속성

- 모양 : Tensor.shape
- 자료형 : Tensor.dtype
- 차원 : Tensor.ndim
- 저장장치 : Tensor.device → CPU, GPU

# ABOUT TENSOR

## ◆ 텐서(Tensor) 살펴보기

### ❖ 자료형 - 정수형(Integer)

Data Type	dtype	CPU TENSOR	GPU TENSOR
	표기 : torch.XXX	표기 : torch.XXX	표기 : torch.cuda.XXX
8bit unsigned integer	uint8	ByteTensor	ByteTensor
8bit integer	int8	CharTensor	CharTensor
16bit integer	int16 OR short	ShortTensor	ShortTensor
32bit integer	int32 OR int	IntTensor	IntTensor
64bit integer	int64 OR long	LongTensor	LongTensor

# ABOUT TENSOR

## ◆ 텐서(Tensor) 살펴보기

### ❖ 자료형 - 실수형(Floating Point)

Data Type	dtype	CPU TENSOR	GPU TENSOR
	표기 : torch.XXX	표기 : torch.XXX	표기 : torch.cuda.XXX
16bit Floating Point	float16 OR half	HalfTensor	HalfTensor
	bfloat16	BFloat16Tensor	BFloat16Tensor
32bit Floating Point	float32 OR float	FloatTensor	FloatTensor
64bit Floating Point	float64 OR double	DoubleTensor	DoubleTensor

# ABOUT TENSOR

## ◆ 텐서(Tensor) 살펴보기

❖ 자료형 - 논리형(Boolean : True, False만 값으로 가지는 타입)

Data Type	dtype	CPU TENSOR	GPU TENSOR
	표기 : torch.XXX	표기 : torch.XXX	표기 : torch.cuda.XXX
Boolean	bool	BoolTensor	BoolTensor

The image features a central dark blue horizontal bar. Above it is a light blue bar that starts from the left edge and ends about two-thirds of the way across. Below the dark blue bar is another light blue bar that starts about one-third of the way across and extends to the right edge.

**CREATE TENSOR**



# CRATE TENSOR

## ◆ 텐서(Tensor) 생성

❖ 원하는 값으로 채운 텐서 생성 → `torch.tensor( 데이터 )`

```
import torch

# -----
# 함수기능: 텐서 속성 정보 출력 함수
# -----
def printInfo(obj, obj_name):
    print(f'\n[{obj_name}]')
    print(f'shape : {obj.shape}')
    print(f'ndim  : {obj.ndim}차원')
    print(f'dtype  : {obj.dtype}')
    print(f'device : {obj.device}')
    print(f'data   :\n{obj.data}')
```

# CRATE TENSOR

## ◆ 텐서(Tensor) 생성

❖ 원하는 값으로 채운 텐서 생성 → `torch.tensor( 데이터 )`

```
# -----  
# 생성 형식 : torch.tensor(데이터)  
# -----  
  
ten1 = torch.tensor(10)  
ten2 = torch.tensor([22,33,44])  
ten3 = torch.tensor([[90,91],[80,81], [70,71]])  
  
# 텐서 정보 출력  
printInfo(ten1, 'ten1')  
printInfo(ten2, 'ten2')  
printInfo(ten3, 'ten3')
```

# CRATE TENSOR

## ◆ 텐서(Tensor) 생성

❖ 모든 값 0으로 채운 텐서 생성 → `torch.zeros( )`

```
# -----  
# 0으로 채운 텐서 생성 : torch.zeros( 행 [ , 열 ] )  
# -----  
zten1=torch.zeros(2,5)  
zten2=torch.zeros(3)  
  
printInfo(zten1, 'zten1')  
printInfo(zten2, 'zten2')
```

# CRATE TENSOR

---

## ◆ 텐서(Tensor) 생성

❖ 모든 값 1로 채운 텐서 생성 → `torch.ones()`

```
# -----  
# 1로 채운 텐서 생성 : torch.ones( 행 [ , 열 ] )  
# -----  
oten1=torch.ones(2,3)  
oten2=torch.ones(3)  
  
printInfo(oten1, 'oten1')  
printInfo(oten2, 'oten2')
```

# CRATE TENSOR

## ◆ 텐서(Tensor) 생성

❖ 모든 값 특정 값으로 채운 텐서 생성 → `torch.full()`

```
# -----  
# 1로 채운 텐서 생성 : torch.full( (행, 열 ), 데이터 )  
# -----  
ften1=torch.full((3, ), 10)  
ften2=torch.full((3,2), 10)  
  
printInfo(ften1, 'ften1')  
printInfo(ften2, 'ften2')
```

# CRATE TENSOR

## ◆ 텐서(Tensor) 생성

❖ 대각선의 1로 채운 텐서 생성 → `torch.eye()`

```
# -----  
# 1로 채운 텐서 생성 : torch.eye( (행 [, 열 ]) )  
# -----  
eten1=torch.eye(3)  
eten2=torch.eye(3,5)  
  
printInfo(eten1, 'eten1')  
printInfo(eten2, 'eten2')
```

# CRATE TENSOR

## ◆ 텐서(Tensor) 생성

❖ 임의의 랜덤 값으로 채운 텐서 생성 → `torch.rand()`

```
# -----  
# - 값의 범위 :  $0 \leq \sim < 1$   [0, 1)  torch.rand(행, 열)  
# -----  
torch.manual_seed(1)  
  
rten1=torch.rand(2,3)  
  
printInfo(rten1, 'rten1')
```

# CRATE TENSOR

## ◆ 텐서(Tensor) 생성

❖ 임의의 정규분포 범위 랜덤 값으로 채운 텐서 생성 → `torch.randn()`

```
# -----  
# - 값의 범위 : 평균0, 표준편차 1 torch.randn(행, 열)  
# -----  
torch.manual_seed(1)  
  
rten2=torch.randn(2,3)  
  
printInfo(rten1, 'rten2')
```



# CRATE TENSOR

## ◆ 텐서(Tensor) 생성

❖ 임의의 정수 랜덤 값으로 채운 텐서 생성 → `torch.randint( )`

```
# -----  
# - 값의 범위 : low<= ~ <high torch.randint(low, high, (행, 열))  
# -----  
torch.manual_seed(1)  
  
rten3=torch.randint(1,5, (2,4))  
  
printInfo(rten1, 'rten3')
```

# CRATE TENSOR

## ◆ 텐서(Tensor) 생성

❖ 특정 타입으로 텐서 생성 → torch.OOOTensor( )

```
ften = torch.FloatTensor(10)
dten = torch.DoubleTensor([1,2])
```

```
boten1=torch.BoolTensor([1,299])
cten1=torch.CharTensor([1,299])
bten1=torch.ByteTensor([1,299])
```

```
tten1=torch.Tensor([1,2])
iten1=torch.IntTensor([1,2])
lten1=torch.LongTensor([1,2])
```

# CRATE TENSOR

## ◆ 텐서(Tensor) 생성

❖ 값의 범위의 숫자로 채운 텐서 생성 → `torch.arange()`

```
# -----  
# torch.arange(start, stop, step)  
# -----  
t1=torch.arange(2)           # 0<= ~ <1 범위에 숫자들  
t2=torch.arange(1,5)         # 1<= ~ <5 범위에 숫자들  
t3=torch.arange(1, 10, 2)     # 1<= ~ <10 범위에 간격 2가 되는 숫자들  
  
printInfo(t1, 't1')  
printInfo(t2, 't2')  
printInfo(t3, 't3')
```

# CRATE TENSOR

## ◆ 텐서(Tensor) 생성

❖ 값의 범위에 균등한 간격의 숫자로 채운 텐서 생성 → `torch.linspace( )`

```
# -----  
# torch.linspace(start, stop, num) : start<= ~ <=stop  
# -----  
t1=torch.linspace(0, 2, 4)      # 0<= ~ <=2 범위 균등하게 4개  
t2=torch.linspace(1,5, 10)     # 1<= ~ <=5 범위 균등하게 10개  
t3=torch.linspace(1, 10, 2)    # 1<= ~ <=10 범위 균등하게 2개  
  
utils.printInfo(t1, 't1')  
utils.printInfo(t2, 't2')  
utils.printInfo(t3, 't3')
```

# CONVERT TENSOR

# CONVERT TENSOR

## ◆ 텐서(Tensor) 변환

❖ list 객체 → 텐서 변환 : `torch.as_tensor()`

```
# -----  
# 값의 범위 : start<= ~ <= stop 범위에 동일 간격 숫자 list 생성  
# -----  
nums=list(range(1, 1000))  
  
lten=torch.as_tensor(nums)  
  
print(arr1)  
print(arr2)
```

# CONVERT TENSOR

---

## ◆ 텐서(Tensor) 변환

❖ ndarray 객체 → 텐서 변환 : `torch.as_tensor()`

```
# -----  
# ndarray 객체로 tensor 객체 변환 : torch.as_tensor(ndarray)  
# -----  
aten1=torch.as_tensor(arr1)  
aten2=torch.as_tensor(arr2[0])  
  
printInfo(aten1, 'aten1')  
printInfo(aten2, 'aten2')
```

# CONVERT TENSOR

## ◆ 텐서(Tensor) 변환

❖ ndarray 객체 → 텐서 변환 : torch.as\_tensor()

```
# -----  
# - ndarray 객체로 tensor 객체 변환 : torch.as_tensor(ndarray)  
# - ndarray와 tensor View 형태로 공유 → 둘 중 하나 변경 시 모두 변경  
# - 벡터 기준 512 이상의 큰 사이즈 리스트의 tensor 변환에 적합  
# -----  
arr1[0]=3.3  
print(f'arr1 ==> {arr1}\nat1 ==> {aten1}')  
aten1[0]=7777  
print(f'arr1 ==> {arr1}\nat1 ==> {aten1}')
```



# CONVERT TENSOR

## ◆ 텐서(Tensor) 변환

❖ ndarray 객체 → 텐서 변환 : torch.from\_numpy()

```
# -----  
# - ndarray 객체로 tensor 객체 변환 : torch.from_numpy(ndarray)  
# - numpy.ndarray 경우에 적합한 변환 함수  
# -----  
aten1=torch.from_numpy(arr1)  
aten2=torch.from_numpy(arr2[0])  
  
printInfo(aten1, 'aten1')  
printInfo(aten2, 'aten2')
```

# CONVERT TENSOR

## ◆ 텐서(Tensor) 변환

❖ ndarray 객체 → 텐서 변환 : torch.from\_numpy()

```
# -----  
# - ndarray 객체로 tensor 객체 변환 : torch.from_numpy(ndarray)  
# - ndarray와 tensor View 형태로 공유 → 둘 중 하나 변경 시 모두 변경  
# - numpy.ndarray 객체의 tensor 변환에 적합  
# -----  
arr1[0]=3.3  
print(f'arr1 ==> {arr1}\naten1 ==> {aten1}')
```

```
aten1[0]=7777  
print(f'arr1 ==> {arr1}\naten1 ==> {aten1}')
```

# CONVERT TENSOR

## ◆ 텐서(Tensor) 변환

❖ ndarray 객체 → 텐서 변환 : torch.tensor()

```
# -----  
# - ndarray 객체로 tensor 객체 변환 : torch.tensor(ndarray)  
# - ndarray 데이터 복사(copy)하여서 텐서 객체 생성  
# -----  
  
aten1=torch.tensor(arr1)  
aten2=torch.tensor (arr2[0])  
  
printInfo(aten1, 'aten1')  
printInfo(aten2, 'aten2')
```

# CONVERT TENSOR

## ◆ 텐서(Tensor) 변환

❖ ndarray 객체 → 텐서 변환 : torch.tensor()

```
# -----  
# - ndarray 객체로 tensor 객체 변환 : torch.tensor(ndarray)  
# - ndarray 데이터 복사(copy)하여서 텐서 객체 생성 → 변경 시 서로 영향 x  
# -----  
arr1[0]=3.3  
print(f'arr1 ==> {arr1}\nat1 ==> {aten1}')
```

```
aten1[0]=7777  
print(f'arr1 ==> {arr1}\nat1 ==> {aten1}')
```

# CONVERT TENSOR

## ◆ 텐서(Tensor) 변환

❖ ndarray 객체 → 텐서 변환

데이터 공유 Sharing	데이터 복사 Copy
<code>touch.as_tensor()</code> <code>touch.from_numpy()</code>	<code>touch.tensor()</code> <code>touch.Tensor()</code>

# CONVERT TENSOR

---

## ◆ 텐서(Tensor) 자료형 변환

### ❖ 형변환 (Type Casting)

- 데이터의 자료형을 변경하는 것을 의미
- 종류
  - 묵시적 형변환 : 시스템에서 자동 형변환, 작은 데이터 >> 큰 데이터, 데이터 손실 없는 경우 진행
  - 명시적 형변환 : 개발자가 함수 및 메서드 활용하여 진행

# CONVERT TENSOR

## ◆ 텐서(Tensor) 자료형 변환

❖ float32 → int32 형변환 : torch.type(변환데이터타입)

```
import torch
import numpy as np

torch.manual_seed(1)

# 3차원 텐서 생성
ten1=torch.randn((2,2,2))
print(f'ten1.dtype : {ten1.dtype}')

# float32 ==> float64 변환
ten1=ten1.type(torch.float64)
print(f'ten1.dtype : {ten1.dtype}')
```

# CONVERT TENSOR

## ◆ 텐서(Tensor) 자료형 변환

❖ float32 → int32 형변환 : torch.type(변환데이터타입)

```
import torch
import numpy as np

torch.manual_seed(1)

# 3차원 텐서 생성
ten1=torch.randn((2,2,2))
print(f'ten1.dtype : {ten1.dtype}')

# float32 ==> int64 변환
ten2=ten1.type(torch.int64)
print(f'ten2.dtype : {ten2.dtype}')
```



# CONVERT TENSOR

## ◆ 텐서(Tensor) 자료형 변환

❖ float32 → int32 형변환 : torch.타입명( )

```
import torch
import numpy as np

torch.manual_seed(1)

# 3차원 텐서 생성
ten1=torch.randn((2,2,2))
print(f'ten1.dtype : {ten1.dtype}')

# float32 ==> int32 변환
ten1=ten1.int( )
print(f'ten1.dtype : {ten1.dtype}')
```

← tensor.float(), double(), long(), int() 등....

# CONVERT TENSOR

## ◆ 텐서(Tensor) 자료형 변환

❖ float32 → int32 형변환 : torch.to( torch.타입명 )

```
import torch
import numpy as np

torch.manual_seed(1)

# 3차원 텐서 생성
ten1=torch.randn((2,2,2))
print(f'ten1.dtype : {ten1.dtype}')

# float32 ==> int32 변환
ten1=ten1.to( torch.int32 )
print(f'ten1.dtype : {ten1.dtype}')
```

# CONTROL ELEMENTS

# CONTROL ELEMENTS

## ◆ 원소/요소 다루기

### ❖ 인덱싱(INDEXING)

- 원소/요소 : 텐서를 구성하는 하나 하나 데이터
- 인덱싱 : 원소 하나 하나를 식별하기 위한 Zero-base의 정수 번호
- 접근/읽기 : 변수명[ 인덱스 번호 ]

```
torch.tensor( [11, 22, 33] )
```

인덱스	0	1	2
인덱스	-3	-2	-1

# CONTROL ELEMENTS

---

## ◆ 원소/요소 다루기

### ❖ 인덱싱(INDEXING)

```
import torch
import numpy as np

ten1 = torch.tensor( [0., 1., 2., 3.] )

# 원소값 읽기 => 객체변수명[ 인덱스 ]
print( ten1[0], ten1[-1] )
print( ten1[ [1, -1] ] )
```

# CONTROL ELEMENTS

---

## ◆ 원소/요소 다루기

### ❖ 슬라이싱(SLICING)

- 연속된 원소/요소를 읽어오는 방식
- 형 식 : 변수명[ 시작인덱스 : 끝인덱스+1 ]

# CONTROL ELEMENTS

---

## ◆ 원소/요소 다루기

### ❖ 슬라이싱(SLICING)

```
import torch
import numpy as np

ten1 = torch.tensor( [0., 1., 2., 3.] )

# 원소값 읽기 => 객체변수명[ 시작인덱스:끝인덱스+1 ]
print( ten1[0 : 2] )
print( ten1[:2] )
```

# CONTROL ELEMENTS

## ◆ 텐서 연산

### ❖ 벡터화 연산

열(column)벡터  $M \times 1$  행렬

같은 인덱스에 위치한 원소(Element-wise)들끼리

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ 10000 \end{bmatrix} + \begin{bmatrix} 10001 \\ 10002 \\ 10003 \\ \vdots \\ 20000 \end{bmatrix} = \begin{bmatrix} 1 + 10001 \\ 2 + 10002 \\ 3 + 10003 \\ \vdots \\ 10000 + 20000 \end{bmatrix} = \begin{bmatrix} 10002 \\ 10004 \\ 10006 \\ \vdots \\ 30000 \end{bmatrix}$$



# CONTROL ELEMENTS

## ◆ 텐서 연산

### ❖ 브로드캐스팅(BROADCASTING)

- 서로 크기 다른 행렬들이 사칙연산 수행할 수 있도록 **자동 크기 조정 연산 수행**
- 서로 다른 **shape** 가진 **array의 산술 연산이 가능**하도록 하는 것

$$x = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \quad x + 1 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \textcircled{1} = ?$$

자동확장

$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + 1 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

# CONTROL ELEMENTS

## ◆ 텐서 연산

### ❖ 브로드캐스팅(BROADCASTING)

- 서로 크기 다른 행렬들이 사칙연산 수행할 수 있도록 **자동 크기 조정 연산 수행**
- 서로 다른 **shape** 가진 **array의 산술 연산이 가능**하도록 하는 것

**자동확장 행(row) 갯수일치**

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{bmatrix}$$

# CONTROL ELEMENTS

## ◆ 텐서 연산

### ❖ 브로드캐스팅(BROADCASTING)

- 서로 크기 다른 행렬들이 사칙연산 수행할 수 있도록 **자동 크기 조정 연산 수행**
- 서로 다른 **shape** 가진 **array의 산술 연산이 가능**하도록 하는 것

3개 컬럼

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}$$

자동확장 단! 요소수==컬럼수

# CONTROL ELEMENTS

## ◆ 텐서 연산

### ❖ 브로드캐스팅(BROADCASTING)

- 서로 크기 다른 행렬들이 사칙연산 수행할 수 있도록 **자동 크기 조정 연산 수행**
- 서로 다른 **shape** 가진 **array의 산술 연산이 가능**하도록 하는 것

3개 컬럼

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}$$

자동확장 단! 요소수==컬럼수

# CONTROL ELEMENTS

## ◆ 텐서 연산

❖ 기본 연산

행(row)벡터  $1 \times M$  행렬

열(column)벡터  $M \times 1$  행렬

기본

$$\mathbf{x} = [x_1 \quad x_2 \quad \dots \quad x_m]$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

# CONTROL ELEMENTS

## ◆ 텐서 연산

❖ 기본 연산

행(row)벡터 1 x M 행렬

전치

열(column)벡터 M x 1 행렬

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}^T = [x_1 \ x_2 \ \dots \ x_m]$$

# CONTROL ELEMENTS

---

## ◆ 텐서 연산

### ❖ 브로드캐스팅(BROADCASTING)

# Case 1 : 같은 크기 행렬

```
m1 = torch.FloatTensor([[3, 4]])
```

```
m2 = torch.FloatTensor([[2, 5]])
```

```
m3=m1+m2
```

```
print(m3, m3.shape)
```

# CONTROL ELEMENTS

---

## ◆ 텐서 연산

### ❖ 브로드캐스팅(BROADCASTING)

# Case 2 : 벡터와 행렬

```
m1 = torch.FloatTensor([[1, 2]])
```

```
m2 = torch.FloatTensor([3])
```

```
m3 = m1 + m2
```

```
print(m3, m3.shape)
```



# CONTROL ELEMENTS

---

## ◆ 텐서 연산

### ❖ 브로드캐스팅(BROADCASTING)

```
# Case 3 : 크기 다른 두 개 행렬  
m1 = torch.FloatTensor([[1, 2]])  
m2 = torch.FloatTensor([[3], [4]])  
m3 = m1 + m2  
print(m3, m3.shape)
```

# CONTROL ELEMENTS

---

## ◆ 텐서 연산

❖ 원소 곱셈 mul() – element-wise

```
m1 = torch.FloatTensor([[1, 2], [3, 4]])
```

```
m2 = torch.FloatTensor([[1], [2]])
```

```
print('Shape of Matrix 1: ', m1.shape)
```

```
print('Shape of Matrix 2: ', m2.shape)
```

```
print( m1 * m2 )
```

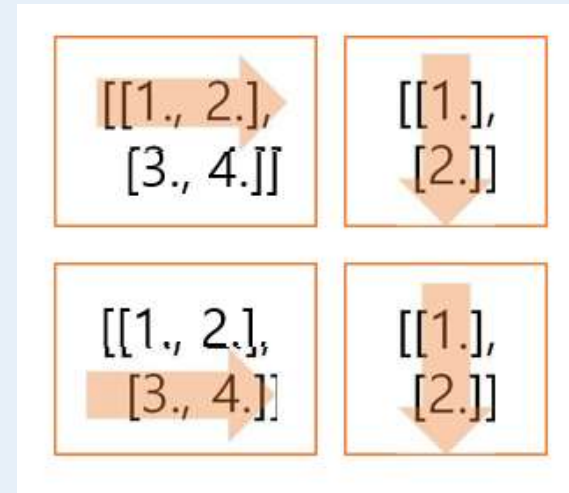
```
print( m1.mul( m2 ) )
```

# CONTROL ELEMENTS

## ◆ Tensor 연산

❖ 행렬 곱셈 matmul()

```
# 필수 조건 ==> ( 행 , 열 ) * ( 행 * 열 )  
m1 = torch.FloatTensor([[1, 2], [3, 4]])  
m2 = torch.FloatTensor([[1], [2]])  
print('Shape of Matrix 1: ', m1.shape)  
print('Shape of Matrix 2: ', m2.shape)  
  
m12= m1.matmul( m2 )  
print( m12.shape)
```



# STORAGE

# STORAGE

## ◆ Tensor 내부 구조

### ❖ STORAGE

- 텐서를 효율적으로 다룰 수 있도록 만들어주는 이 1차원 배열
- 1차원 텐서이든, 2차원 텐서이든, n차원 텐서이든 메모리 안에는 1차원 배열 형태로 저장
- 구성 : offset + stride
- offser
  - 첫 번째 원소가 storage에 저장되어 있는 인덱스
- stride
  - 텐서 안의 어떤 한 값에서 다음 원소를 얻기 위하여 storage에서 뛰어넘어야할 인덱스의 개수
  - 표기 형식 : 다음 행 원소 얻기위해 n만큼, 다음 열 원소 얻기위해 m만큼 → ( n, m )

# STORAGE

## ◆ Tensor 내부 구조

❖ Storge 정보 출력 함수

```
def printStorage(obj, obj_name):  
    print(f"\n==== [{obj_name}] 기 본 정 보 ====")  
    print(f'Shape   : {obj.shape}')  
    print(f'Dim     : {obj.ndim}D')  
    print(f'DType   : {obj.dtype}')  
    print(f'itemsize: {obj.itemsize}')  
  
    print("==== STORAGE ====")  
    print("Offset: ", obj.storage_offset())  
    print("Strides: ", obj.stride())  
    print("=====")  
    print(obj.untyped_storage())
```

# STORAGE

---

## ◆ Tensor 내부 구조

❖ Tsensor 생성

```
import torch

ten1 = torch.tensor([[3,1,8], [0,9,2]], dtype=torch.int8)
ten2 = torch.tensor([[3,1],[8, 0],[9,2]], dtype=torch.int8)

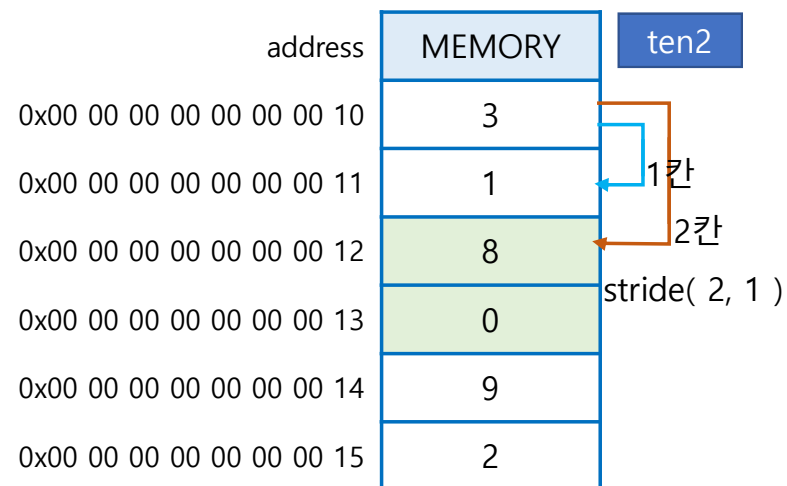
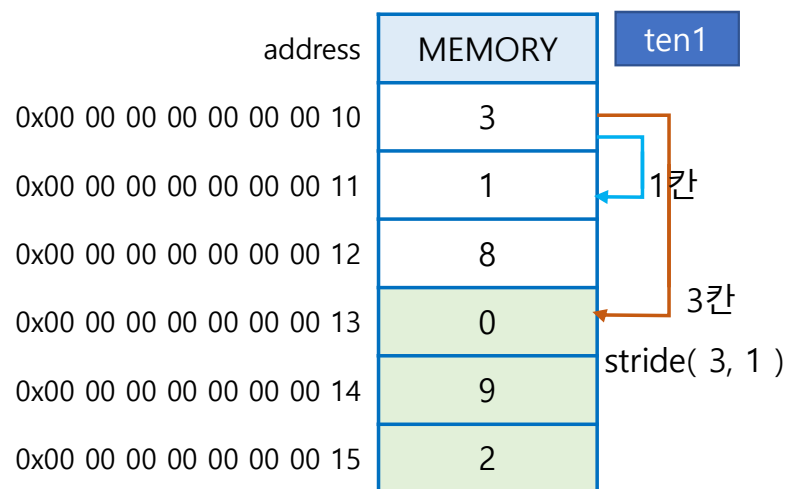
printStorage(ten1,'ten1')
printStorage(ten2,'ten2')
```

# STORAGE

## ◆ Tensor 내부 구조

❖ Tsensor 생성

▪ dtype : torch.int8 즉, 1바이트





# STORAGE

## ◆ Tensor 내부 구조

### ❖ 결과

==== [ten1] 기본 정보 ====

Shape : torch.Size([2, 3])

Dim : 2D

DType : torch.int8

itemsizes: 1

==== STORAGE ====

Offset: 0

Strides: (3, 1)

=====

3

1

8

0

9

2

[torch.storage.UntypedStorage(device=cpu) of size 6]

==== [ten2] 기본 정보 ====

Shape : torch.Size([3, 2])

Dim : 2D

DType : torch.int8

itemsizes: 1

==== STORAGE ====

Offset: 0

Strides: (2, 1)

=====

3

1

8

0

9

2

[torch.storage.UntypedStorage(device=cpu) of size 6]

# CONVERT SHAPE

# CONVERT SHAPE

---

## ◆ Tensor 형태/모양 변경

### ❖ Tensor.reshape( )

- 형태/모양 변경된 텐서의 copy 텐서 또는 view 텐서 반환
  - view Tensor 란? tensor 메모리 주소 그대로 두고 모양만 변경한 Tensor
  - copy Tensor란? 동일한 데이터를 새로루 메모리 주소에 생성한 Tensor
- 형식 : tensor객체.reshape( 행 [, 열] )

# CONVERT SHAPE

## ◆ Tensor 형태/모양 변경

❖ Tensor.reshape( )

```
import torch

ten1=torch.tensor([11,22,33,44,55,66])

ten2=ten1.reshape(-1,1,1)  # -1 의미 : 원소 개수에 맞게 알아서 배정
print(ten2, ten2.ndim)

ten3=ten2.reshape(-1)
print(ten3, ten3.ndim)
```

# CONVERT SHAPE

---

## ◆ Tensor 형태/모양 변경

### ❖ Tensor.view( )

- 형태/모양 변경된 텐서의 view 텐서 반환
  - 기존 Tensor와 memory 공간 공유
  - stride 크기만 변경
- 형식 : tensor객체.view ( 행 [, 열] )

# CONVERT SHAPE

## ◆ Tensor 형태/모양 변경

❖ Tensor.view( )

```
import torch

ten1=torch.tensor([11,22,33,44,55,66])

ten2=ten1.view(-1,1,1)  # -1 의미 : 원소 개수에 맞게 알아서 배정
print(ten2, ten2.ndim)

ten3=ten2.view(-1)
print(ten3, ten3.ndim)
```

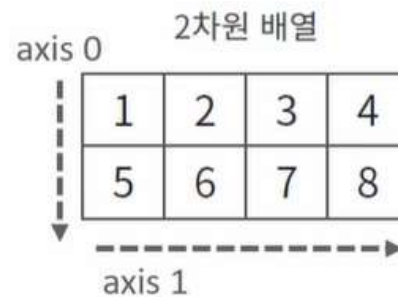
# CONVERT SHAPE

## ◆ Tensor 형태/모양 변경

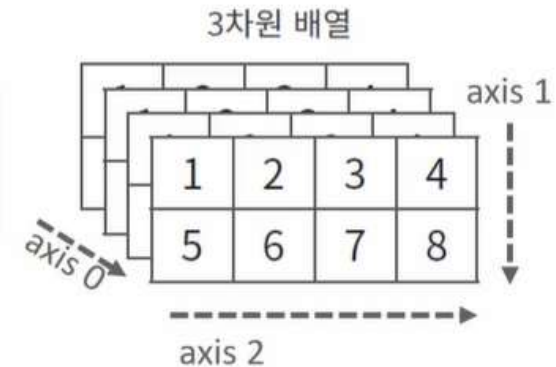
❖ 축 (axis)



shape(수, )



shape(행, 열)  
axis = 0 행 row  
axis = 1 열 column



shape(차원, 행, 열)  
axis = 0 차원  
axis = 1 행 row  
axis = 2 열 column

# CONVERT SHAPE

---

## ◆ Tensor 형태/모양 변경

❖ Tensor.transpose( )

- 2개 축/차원 맞교환하여 형태/모양 변경된 텐서 반환
- 형식 : tensor객체.transpose ( 축0, 축1, 축2 )



# CONVERT SHAPE

---

## ◆ Tensor 형태/모양 변경

❖ Tensor.permute( )

- 모든 축/차원 맞교환하여 형태/모양 변경된 텐서 반환
- 형식 : tensor객체.permute ( 축0, 축1, 축2 )

# CONVERT SHAPE

---

## ◆ Tensor 형태/모양 변경

❖ `torch.squeeze( tensor )`

- 1인 차원을 모두 제거하여 텐서 형태/모양 변경
- 차원 지정하면 해당 **차원이 1인 경우 제거**

❖ `tensor.unsqueeze( dim )`

- **지정된 차원에 1인 차원을 생성**하여 텐서 형태/모양 변경

# CONVERT SHAPE

---

## ◆ Tensor 형태/모양 변경

### ❖ Contiguous

- 메모리 안에 끊어지지 않은 block안에 저장되어 있는 어레이
- 어레이의 다음 값에 접근하기 위해, 메모리의 다음 주소로 이동