

# Lecture 1

Hello Python,  
Hello PyTorch!



Seoul National University



Human Interface Laboratory

# Contents

- **Intro**
  - 딥러닝 코드의 대략적 구조
  - Python이란?
  - PyTorch란?
- **실습환경 구축**
- **Python 입문**
- **PyTorch 입문**
- **실습코드 실행**



# 강의자료 다운로드

- 다음의 링크를 주소창에 입력:

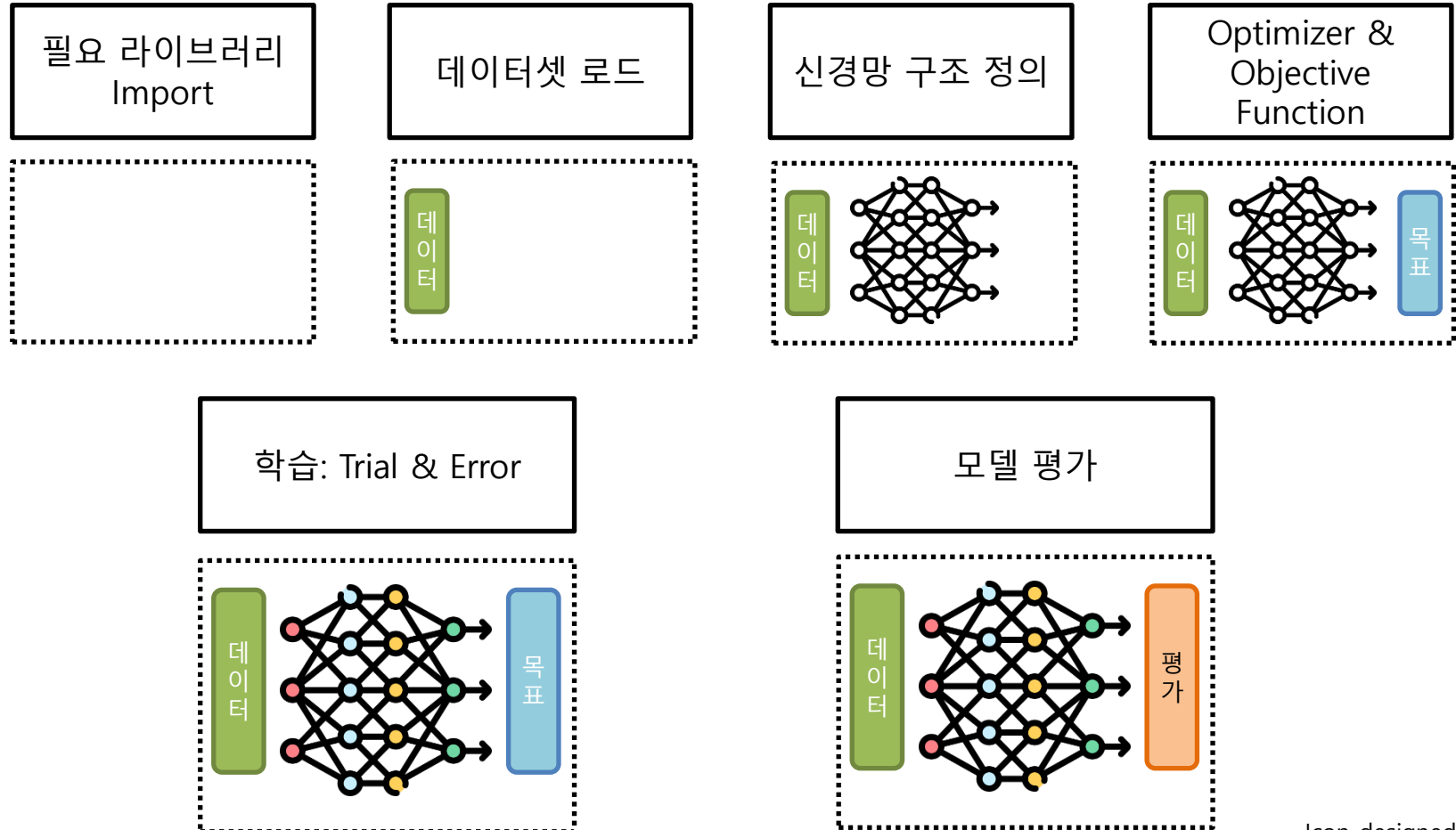
<https://bit.ly/3BZxZon>

- lecture1.pptx를 다운로드할 것



# Intro

- 딥러닝 코드의 대략적 구조



Icon designed by freepik

# Intro

- **Python**

- 컴퓨터에게 사용자가 의도한 연산을 수행시키는 프로그래밍 언어
- 다음과 같은 장점으로 AI 개발 커뮤니티의 필수 언어로 자리잡음
  - ‘비교적’ 쉽고 간단함
  - 다양한 라이브러리와 모듈
    - 사용자가 일일이 구현해야 하는 수고를 덜 수 있음
    - 데이터 분석, AI 개발 등을 위한 프레임워크도 이미 구현돼있음
  - 상당량의 온라인 오픈소스 코드

PYTHON

```
>>> print("Hello World!")
Hello World!
```

C++

```
#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

JAVA

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

ASSEMBLY

```
...
mov     eax, 0
call    _printf@14
...

```

# Current Works

- **PyTorch**

- Facebook에서 개발한 딥러닝 프레임워크
- 구현의 편의성과 자유도가 높음
- GPU 사용이 편리함
- 대부분의 AI Research Community에서 사용 중



## TOP 10 DEEP LEARNING FRAMEWORKS

10. You can't
9. Rank them
8. Because each has
7. Their own merits
6. That make them better
5. Tools for certain tasks
4. Than the others
3. Just appreciate that they're
2. All used to build great stuff
1. Pytorch

# Intro

- 코드 예제
  - 간단한 딥러닝 코드 구현

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# 데이터 전처리
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

# 데이터셋 다운로드 및 로드
train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = datasets.MNIST(root='./data', train=False, transform=transform, download=True)

# DataLoader 설정
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

# 간단한 신경망 모델
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28) # Flatten the image
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = SimpleNN().to(device)

# 손실 함수 및 옵티마이저 설정
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
# 모델 훈련
epochs = 5
for epoch in range(epochs):
    model.train()
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")

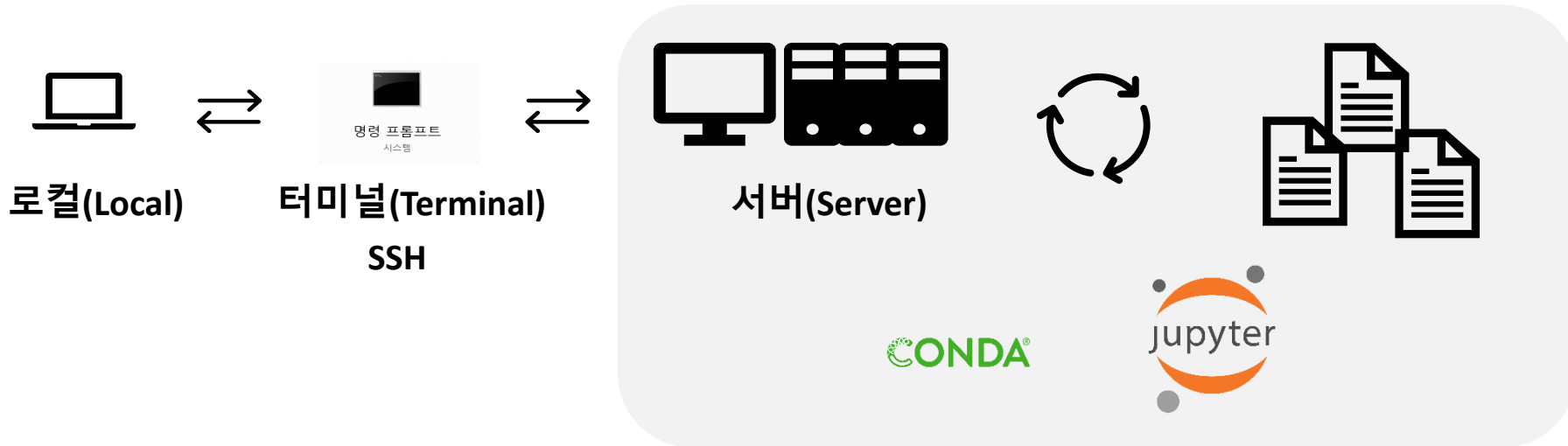
model.eval()
correct = 0
total = 0

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f'Accuracy: {accuracy:.2f}%')
```

# 실습환경

## • 실습환경 도식



- **로컬(local)**: 현재 사용중인 노트북으로, 하드웨어 성능 한계로 AI 모델 직접 동작은 어려움
- **서버(server)**: 코드가 실제로 저장되고 실행될 고성능 컴퓨터
- **터미널(terminal)**: 네트워크 연결(**SSH**)을 통해 로컬에서 서버로 접속하는 창구
- **콘다(conda)**: 프로젝트에 맞게 서버 내 환경을 설정하는 도구
- **주피터 랩(jupyter lab)**: 코드 편집 도구



# 실습환경

- 로컬에서 서버 접속
  - 본인 계정 확인하기

ID	PW
aiwinter01	a4JHG5TlvW
aiwinter02	fyT)*DKZ7B
aiwinter03	C6kO0iAkFl
aiwinter04	&HTs&zPK^c
aiwinter05	SS(lA**oG2
aiwinter06	mCi1s&XiqV
aiwinter07	Z1o%wAW3\$T
aiwinter08	OS7OdXGV#
aiwinter09	S7AYUTXast
aiwinter10	YClps4sl^!

ID	PW
aiwinter11	4l%xKxc\$^l
aiwinter12	IGQ2XXq7NS
aiwinter13	%9!lyuV!AG
aiwinter14	)n5Dajipyv
aiwinter15	PsWWiMiNAw
aiwinter16	xqGnLx\$4aK
aiwinter17	@ls3*\$Nn7l
aiwinter18	l!%KLbl%wJ
aiwinter19	RYKWQjf3tO
aiwinter20	b7qSjOL!pb

ID	PW
aiwinter21	bP1*F*oDV%
aiwinter22	px4%0!ZMaX
aiwinter23	yZA0xqtIR9
aiwinter24	Oo3QYopRzA
aiwinter25	RxL9kSzndx
aiwinter26	a!j!36lTTq
aiwinter27	\$uBmWempgl
aiwinter28	fXLeqP2Wxo
aiwinter29	DoxmwVKtM\$
aiwinter30	RVq4BTwg*w

# 실습환경

- 로컬에서 서버 접속

- 터미널에서 SSH 연결

- 본인 계정 ID 확인하기
    - 명령 프롬프트 열기(cmd)
    - 터미널에서 다음을 입력:

ssh {본인계정ID}@147.46.121.38 (ex. ssh aiwinter96@147.46.121.38)

- 본인 계정 비밀번호 입력

(만약)Are you sure you want to continue connecting (yes/no/[fingerprint])?

>> yes

```
Last login: Fri Jan  3 15:32:17 2025 from 147.46.243.160
(base) aiwinter01@ECE-util1:~$
```

다음으로 넘어가지 말고 대기

# 실습환경

- 실습코드를 서버에 다운로드

- 터미널에서 다음을 한 줄마다 입력:

```
git clone https://github.com/WiFiHan/ssai_winter_24w
```

```
ls
```

```
(base) aiwinter01@ECE-util1:~$ git clone https://github.com/WiFiHan/ssai_winter_24w
'ssai_winter_24w'에 복제합니다...
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 15 (delta 1), reused 15 (delta 1), pack-reused 0 (from 0)
오브젝트 묶음 푸는 중: 100% (15/15), 완료.
(base) aiwinter01@ECE-util1:~$ ls
examples.desktop  lab  ssai_winter_24w
(base) aiwinter01@ECE-util1:~$
```

# 실습환경

- 코드 편집기 실행을 위한 환경 설정

- GPU 할당

- 터미널에서 다음을 입력:

```
ssai_agpu -g=1
```

- Conda를 활용하여 필요한 라이브러리 설치

- 터미널에서 다음을 한 줄마다 입력:

```
conda init
```

```
source ~/.bashrc
```

```
conda create -n aiwinter
```

```
conda activate aiwinter
```

```
conda install pytorch torchvision torchaudio torchtext matplotlib pytorch-cuda=12.1 jupyterlab -c pytorch -c nvidia -c defaults
```

**다음으로 넘어가지 말고 대기**

# 실습환경

- 코드 편집기 실행

- Jupyter lab 시동

- 터미널에서 다음을 입력:

`ssai_agpu -g=1`

- 서버명 확인(ex. ece-agpu10) `(test) aiwinter01@ece-agpu4:~$`

- 보안조치된 Jupyter lab 접속하기

- 새 터미널(명령프롬프트) 띄우기
    - 새 터미널에서 다음을 한 줄마다 입력:

`ssh -L 12345:localhost:9999 {본인ID}@147.46.121.38`

(ex. `ssh -L 12345:localhost:9999 aiwinter01@147.46.121.38`)

`ssh -L 9999:localhost:8888 {본인ID}@{앞에서 확인한 서버명}`

(ex. `ssh -L 9999:localhost:8888 aiwinter01@ece-agpu4`)

**다음으로 넘어가지 말고 대기**

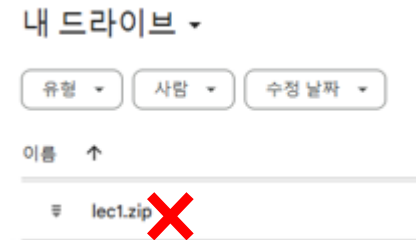
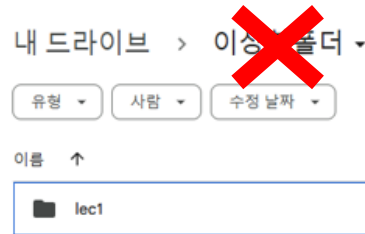
# 실습환경

- **(안 되는 사람 한정) 대안: 구글코랩 사용하기**

- 실습코드 다운로드(lec1.zip) 후 압축해제

<https://bit.ly/4gBVkvk>

- 압축해제된 폴더를 본인 구글드라이브 '내 드라이브' 첫 화면에 업로드
  - 다른 폴더에 넣거나, zip 폴더를 업로드하면 안 됨!



- 아래 두 셀을 실행:

✓ 구글 코랩을 사용할 경우에만 아래 두 셀을 실행

```
[1] from google.colab import drive
    drive.mount('/content/drive')
```

Mounted at /content/drive

```
[5] #반드시 "내 드라이브" 첫 화면에 lec1 폴더가 있는지 확인(다른 경로에 넣을 경우 확인 못함)
```

```
%cd /content/drive/MyDrive/lec1
```

# Python 입문

- **Contents**
  - 출력과 입력
  - 변수를 저장하는 자료형
  - 반복문과 조건문
  - 함수
  - 클래스와 상속
  - 모듈과 패키지



# Python 입문

- 출력과 입력

- print: 출력

- 화면에 사용자가 원하는 텍스트나 변수의 내용을 출력할 때 사용



```
print("Hello, World!")  
# 결과: Hello, World!
```



```
name = "홍길동"  
age = 25  
print(f"이름: {name}, 나이: {age}")  
# 결과: 이름: 홍길동, 나이: 25
```



# Python 입문

- 출력과 입력

- input: 입력

- 프로그램에 사용자가 원하는 텍스트나 변수의 내용을 입력할 때 사용



```
name = input("이름을 입력하세요: ")  
print(f"안녕하세요, {name}님!")
```

# Python 입문

- 자료형: 변수의 '품사'
  - 숫자형
    - int(정수), float(실수)

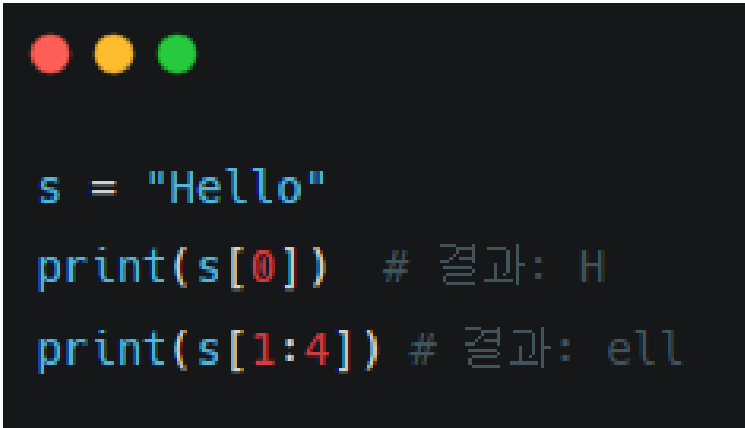
```
x = 10      # int
y = 3.14    # float
```

# Python 입문

- 자료형: 변수의 '품사'

- 문자열

- 큰따옴표(“”) 또는 작은따옴표(”)로 묶음
    - s[0]: 문자열의 첫 번째 글자, s[1:4]: 문자열의 두 번째~다섯 번째 글자



```
s = "Hello"  
print(s[0]) # 결과: H  
print(s[1:4]) # 결과: ell
```

# Python 입문

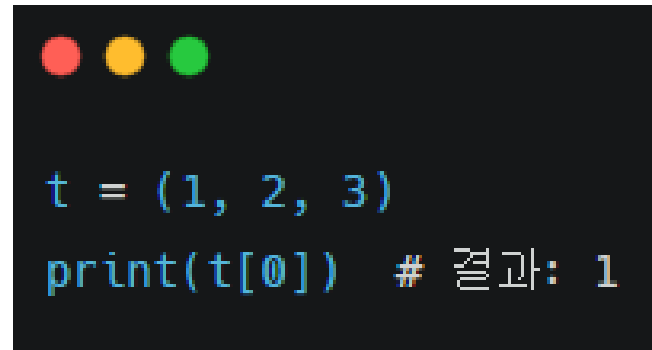
- 자료형: 변수의 '품사'
  - 리스트: 여러 데이터를 저장하는 '유연한' 상자과 같음
    - 개별 데이터를 넣었다 빼는 것이 자유로움(mutable)
    - 대신 처리 속도가 튜플(tuple) 자료형에 비해 느림

```
numbers = [1, 2, 3, 4]
numbers.append(5) # 리스트에 요소 추가
print(numbers)   # 결과: [1, 2, 3, 4, 5]
print(numbers[3]) # 결과: 4
```

# Python 입문

- 자료형: 변수의 '품사'

- 튜플: 여러 데이터를 저장하는 '고정된' 상자과 같음
  - 개별 데이터를 넣었다 빼는 것이 어려움(immutable)
  - 대신 처리 속도가 리스트(list) 자료형에 비해 빠름

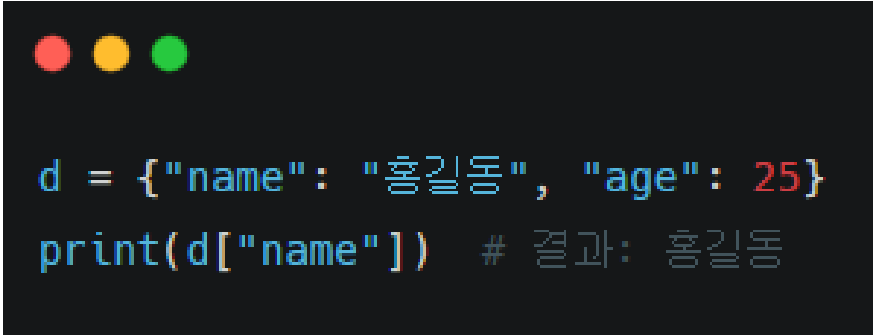


```
t = (1, 2, 3)
print(t[0]) # 결과: 1
```

# Python 입문

- 자료형: 변수의 '품사'

- 딕셔너리: 여러 데이터를 '라벨을 붙여' 저장하는 상자와 같음
  - Key-value 쌍으로 데이터를 저장함

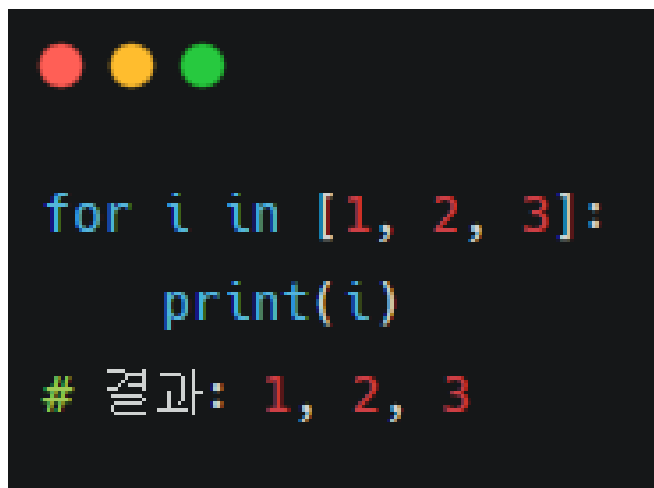


```
d = {"name": "홍길동", "age": 25}
print(d["name"]) # 결과: 홍길동
```

# Python 입문

- 반복문

- For문: 각 요소에 대하여 들여쓰기된 행위를 반복 수행



```
for i in [1, 2, 3]:  
    print(i)  
# 결과: 1, 2, 3
```

# Python 입문

- 반복문

- While문: 어떤 조건의 평가값이 True일 때까지 들여쓰기된 행위를 반복 수행

```
x = 0
while x < 3:
    print(x)
    x += 1
# 결과: 0, 1, 2

while True:
    print("논문을 투고합니다")
# 무한루프
```



# Python 입문

## • 조건문

- 조건에 따라 수행되는 코드를 다르게 할 수 있음(Branch)
- 참고) Python에서 '='과 '=='는 다름
  - '=': 좌변의 변수에, 우변을 계산한 값을 **할당**하는 기호
    - Ex. `x = x+10`: 변수 x에 기존 x값에 10을 더한 새로운 값을 할당하겠다
  - '==': 좌변과 우변을 각각 계산한 값이 같은지 다른지 True/False로 **판단**하는 기호

```
x = 10
if x > 5:
    print("크다")
elif x == 5:
    print("같다")
else:
    print("작다")
# 결과: 크다
```

# Python 입문

## • 함수

- 특정 행위들을 수행하고 경우에 따라 반환하는 코드 블록
- 함수의 동작 원리와 반환값을 앞에서 정의하면(definition),
- 나중에 input argument를 제공하여 호출함으로써(call) 해당 동작을 실행 가능

```
def greet(name):  
    return f"안녕하세요, {name}님!"
```

```
print(greet("홍길동"))  
# 결과: 안녕하세요, 홍길동님!
```

```
def add(x, y):  
    return x + y
```

```
result = add(3, 5)  
print(result) # 결과: 8
```

# Python 입문

## • 클래스와 상속

- 동일한 특징과 기능을 공유하는 대상을 하나로 묶어서 정의하는 설계도
- Ex) Human Class는
  - Attribute(Human이 공유하는 속성)으로 name, age 등이 있으며
  - Method(Human이 공유하는 기능)으로 introduce, celebrate\_birthday 등이 있다

```
class Person: # 클래스 정의
    # 생성자
    def __init__(self, name, age):
        self.name = name # 속성
        self.age = age

    # 메서드
    def introduce(self):
        print(f"안녕하세요, 제 이름은 {self.name}이고 나이는 {self.age}살입니다.")

    def celebrate_birthday(self):
        self.age += 1
        print(f"생일 축하합니다! 이제 {self.age}살이에요.")
```

```
p1 = Person("홍길동", 25) # 객체 생성
p2 = Person("김영희", 30)

p1.introduce() # 메서드 호출
# 출력: 안녕하세요, 제 이름은 홍길동이고 나이는 25살입니다.

p2.celebrate_birthday()
# 출력: 생일 축하합니다! 이제 31살이에요.
```

# Python 입문

## • 클래스와 상속

- 어떤 클래스는 다른 클래스를 상속할 수 있음(Inheritance)
- Ex) Dog Class가 Animal Class를 상속하도록 정의하면
  - Dog Class는 Animal Class의 Attribute, Method를 물려받으며
  - 물려받은 Attribute, Method를 재정의하거나 새 Attribute, Method를 가질 수 있음

```
# 부모 클래스
class Animal:
    def __init__(self, name, sound):
        # Animal의 속성 (Attributes)
        self.name = name
        self.sound = sound

    # Animal의 메서드 (Methods)
    def speak(self):
        print(f"{self.name}는 '{self.sound}' 소리를 냅니다.")

    def eat(self):
        print(f"{self.name}가 음식을 먹습니다.")
```

```
# 자식 클래스
class Dog(Animal):
    def __init__(self, name, sound, breed):
        # 부모 클래스의 __init__ 메서드 호출 (상속)
        super().__init__(name, sound)
        # Dog만의 새로운 속성 추가
        self.breed = breed

    # 부모 메서드 'speak' 재정의 (오버라이딩)
    def speak(self):
        print(f"{self.name}(품종: {self.breed})가 '{self.sound}' 소리를 냅니다.")

    # Dog만의 새로운 메서드 추가
    def fetch(self, item):
        print(f"{self.name}가 {item}을 물어옵니다.")
```

# Python 입문

- 모듈과 패키지

- 모듈: 특정 함수, 변수, 클래스들의 정의가 담긴 개별 파일
- 코드 중복을 줄이고, 기능별로 프로그램을 구조화하기 위해 사용

```
# module1.py
def add(x, y):
    return x + y
```

=

```
#main.py
import module1
print(add(2, 3))
# 결과: 5
```

```
# main.py
def add(x, y):
    return x + y

print(add(2, 3))
# 결과: 5
```

# Python 입문

## • 모듈과 패키지

- 패키지: 여러 모듈(파일)이 담긴 폴더
- 모듈이 책 한 권이라면, 패키지는 책장과 같음
- 모듈과 패키지의 집합이 라이브러리(PyTorch 또한 라이브러리임)

```
math_package/  
  __init__.py  
  calculator.py  
  geometry.py  
  main.py
```

```
# math_package/calculator.py  
  
def add(x, y):  
    return x + y  
  
def subtract(x, y):  
    return x - y  
  
def multiply(x, y):  
    return x * y  
  
def divide(x, y):  
    if y == 0:  
        return "0으로 나눌 수 없습니다."  
    return x / y  
  
# math_package/geometry.py  
import math  
  
def area_circle(radius):  
    return math.pi * radius ** 2  
  
def perimeter_circle(radius):  
    return 2 * math.pi * radius  
  
def area_rectangle(width, height):  
    return width * height  
  
def perimeter_rectangle(width, height):  
    return 2 * (width + height)
```

```
# main.py  
  
from math_package.calculator import add, subtract, multiply, divide  
from math_package.geometry import area_circle, perimeter_circle, area_rectangle, perimeter_rectangle  
  
# 계산기 기능 테스트  
print(add(3, 4))           # 결과: 7  
print(subtract(10, 5))     # 결과: 5  
print(multiply(3, 4))      # 결과: 12  
print(divide(10, 2))       # 결과: 5.0  
print(divide(10, 0))       # 결과: 0으로 나눌 수 없습니다.  
  
# 도형 계산 기능 테스트  
print(area_circle(5))      # 결과: 78.53981633974483  
print(perimeter_circle(5)) # 결과: 31.41592653589793  
print(area_rectangle(3, 4)) # 결과: 12  
print(perimeter_rectangle(3, 4)) # 결과: 14
```

# Python 입문

## • 모듈과 패키지

- 패키지: 여러 모듈(파일)이 담긴 폴더
- 모듈이 책 한 권이라면, 패키지는 책장과 같음
- 모듈과 패키지의 집합이 라이브러리(PyTorch 또한 라이브러리임)

```
math_package/  
  __init__.py  
  calculator.py  
  geometry.py  
  main.py
```

```
# math_package/calculator.py  
  
def add(x, y):  
    return x + y  
  
def subtract(x, y):  
    return x - y  
  
def multiply(x, y):  
    return x * y  
  
def divide(x, y):  
    if y == 0:  
        return "0으로 나눌 수 없습니다."  
    return x / y  
  
# math_package/geometry.py  
import math  
  
def area_circle(radius):  
    return math.pi * radius ** 2  
  
def perimeter_circle(radius):  
    return 2 * math.pi * radius  
  
def area_rectangle(width, height):  
    return width * height  
  
def perimeter_rectangle(width, height):  
    return 2 * (width + height)
```

```
# main.py  
  
from math_package.calculator import add, subtract, multiply, divide  
from math_package.geometry import area_circle, perimeter_circle, area_rectangle, perimeter_rectangle  
  
# 계산기 기능 테스트  
print(add(3, 4))           # 결과: 7  
print(subtract(10, 5))     # 결과: 5  
print(multiply(3, 4))      # 결과: 12  
print(divide(10, 2))       # 결과: 5.0  
print(divide(10, 0))       # 결과: 0으로 나눌 수 없습니다.  
  
# 도형 계산 기능 테스트  
print(area_circle(5))      # 결과: 78.53981633974483  
print(perimeter_circle(5)) # 결과: 31.41592653589793  
print(area_rectangle(3, 4)) # 결과: 12  
print(perimeter_rectangle(3, 4)) # 결과: 14
```

# PyTorch 입문

- **Contents**
  - Tensor
  - 데이터 로드하기
  - 신경망 모델 구조 정의하기
  - 손실함수 및 옵티마이저 정의하기
  - 모델 훈련 및 평가하기





# PyTorch 입문

- **Tensor**

- PyTorch에서 사용하는 다차원 배열(배열의 일반화)의 자료형
- 1D: 벡터, 2D: 행렬, 3D: 행렬의 묶음, 4D...
- CPU에 저장된 Tensor를 GPU로 이동시킬 수 있음
  - 병렬처리가 가능해져 연산속도가 가속됨

```
import torch

# 1D Tensor 생성
x = torch.tensor([1, 2, 3])
print(x)

# 2D Tensor 생성
y = torch.tensor([[1, 2], [3, 4]])
print(y)

z = torch.tensor([[[1,2], [3,4]], [[5,6], [7,8]]])

# 랜덤 Tensor 생성
w = torch.rand(3, 3) # 3x3 행렬
print(w)
```

```
# 1. 디바이스 설정 (GPU 사용 가능 여부 확인)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# 2. 텐서 생성 및 디바이스로 전송
x = torch.tensor([1, 2, 3], dtype=torch.float32).to(device)

# 3. 텐서 정보 출력
print("Shape:", x.shape)      # 크기 출력
print("Dtype:", x.dtype)     # 자료형 출력
print("Device:", x.device)    # 사용 중인 장치 출력 (CPU/GPU)
```

# PyTorch 입문

- Dataset & Dataloader

- Dataset: 데이터를 정의하고 저장함
  - 아래 예제에선 MNIST 손글씨 데이터셋(0~9 손글씨)을 사용
- Dataloader: Dataset에서 batch 단위로 데이터를 로드함
  - Batch Size: 한 번에 학습하는 데이터의 수. 클수록 학습 속도 & 메모리 사용량 증가
  - 아래 예제에서는 Batch Size가 64; 즉 한 번에 64개의 손글씨 이미지를 처리함



```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# 데이터 변환 정의
transform = transforms.Compose([
    transforms.ToTensor(),           # 이미지를 Tensor로 변환
    transforms.Normalize((0.5,), (0.5,)) # 정규화 수행
])

# 훈련 및 테스트 데이터셋 로드
train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

# DataLoader 설정
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

# PyTorch 입문

## • 신경망 모델 구조 정의

- nn.Module 클래스를 상속하여 신경망 정의
- 모델 구조 정의: 레이어의 크기 및 순서 설정



```
import torch.nn as nn
import torch.nn.functional as F

class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128) # 입력층 → 은닉층
        self.fc2 = nn.Linear(128, 64)      # 은닉층 → 은닉층
        self.fc3 = nn.Linear(64, 10)       # 은닉층 → 출력층

    def forward(self, x):
        x = x.view(-1, 28 * 28) # 이미지 데이터 Flatten
        x = F.relu(self.fc1(x)) # 활성화 함수 적용
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

# PyTorch 입문

- 손실함수 및 옵티마이저 정의

- 손실함수: 모델의 예측 오차를 정량화
  - 아래 예제에선 CrossEntropyLoss라는 손실함수를 사용
- 옵티마이저: 손실함수값을 바탕으로 가중치 갱신 규칙을 정함
  - 아래 예제에선 Adam이라는 옵티마이저를 사용

```
import torch.optim as optim

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = SimpleNN().to(device)

criterion = nn.CrossEntropyLoss() # 손실 함수
optimizer = optim.Adam(model.parameters(), lr=0.001) # 옵티마이저
```

# PyTorch 입문

## • 모델 훈련 및 평가

- 훈련(Train): 데이터셋을 바탕으로 가중치를 갱신하는 과정
  - 데이터셋을 바탕으로 손실함수값을 계산한 뒤, 옵티마이저를 작동시킴
- 평가(Evaluation): 훈련된 모델의 성능을 평가하는 과정
  - 아래 예제에서는 Accuracy를 계산함

```
epochs = 5
for epoch in range(epochs):
    model.train()
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")
```

```
model.eval()
correct = 0
total = 0

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy: {100 * correct / total:.2f}%')
```

# PyTorch 입문

- 예측 결과 시각화

- Matplotlib: Python에서 사용되는 시각화 라이브러리

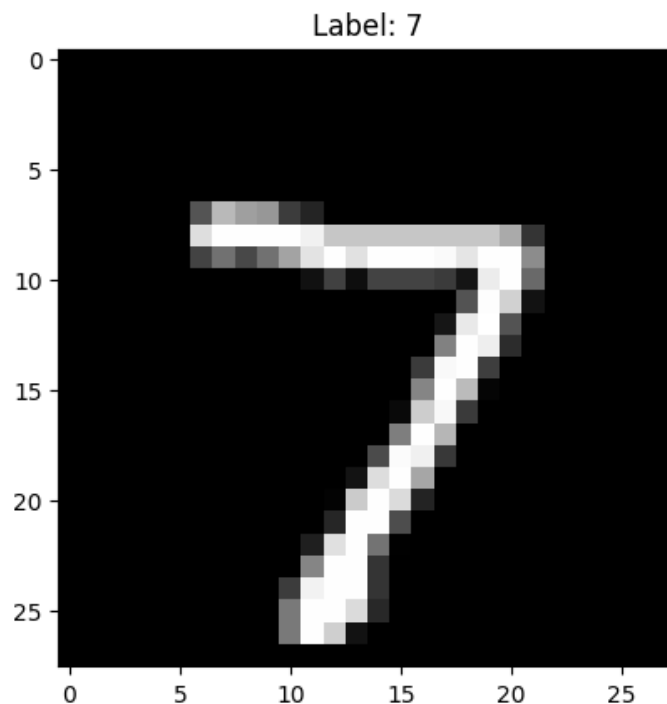
```
import matplotlib.pyplot as plt

sample_image, sample_label = test_dataset[0]

plt.imshow(sample_image.squeeze(), cmap='gray')
plt.title(f"Label: {sample_label}")
plt.show()

sample_image = sample_image.to(device)
model.eval()
with torch.no_grad():
    prediction = model(sample_image.unsqueeze(0))
    predicted_label = torch.argmax(prediction, 1).item()

print(f"Predicted Label: {predicted_label}")
```



# Closing

## • 코드 예제 해석

- 라이브러리 Import, 데이터셋 로드, 신경망 모델구조 정의, 손실함수 및 옵티마이저 정의
- 모델 훈련 및 평가

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# 데이터 전처리
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

# 데이터셋 다운로드 및 로드
train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = datasets.MNIST(root='./data', train=False, transform=transform, download=True)

# DataLoader 설정
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

# 간단한 신경망 모델
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28) # Flatten the image
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = SimpleNN().to(device)

# 손실 함수 및 옵티마이저 설정
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
# 모델 훈련
epochs = 5
for epoch in range(epochs):
    model.train()
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")

model.eval()
correct = 0
total = 0

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f'Accuracy: {accuracy:.2f}%')
```