

15기 정규세션

ToBig's 14기 민거홍

클래스

Contents

Unit 01 | Intro

Unit 02 | OOP & Class 개념

Unit 03 | Class의 구조와 상속

Unit 04 | Magic Method

+ (Bonus) 모듈, __main__

15기 정규세션

ToBig's 14기 민거홍

01. Intro

Unit 01 | Intro

클래스???

- 클래스가 대체 뭐지??
- 어디에 쓰는 거지??
- 빅데이터 동아리에서 이걸 왜 배워야 하는 거지??



Unit 01 | Intro

파이썬을 쓰고 계신다면, 여러분들은 이미 클래스를 활용하고 계셨습니다!

- 파이썬에서는 C 등 다른 언어들과 달리 원시 자료형 (primitive type)인 int, float 등은 다 클래스입니다!

```
example.py X
C:\> Users > placi > Desktop > example.py > ...
1 anInt = 3
2 print(type(anInt))
```

```
PS C:\Users\placi\Desktop> python example.py
PS C:\Users\placi\Desktop> py example.py
<class 'int'>
```

- 과제에서도 pandas, BeautifulSoup 등을 활용하셨는데...

```
from sklearn.linear_model import LinearRegression
```

```
model = LinearRegression()
```

```
def get_top_news(date):
    # 많이 본 뉴스 페이지 링크
    url = f"https://news.naver.com/main/ranking/popularDay.nhn?rankingType=popular_day&date={date}"
    res = requests.get(url)
    soup = BeautifulSoup(res.text, "html.parser")
```

Unit 01 | Intro

파이썬을 쓰고 계신다면, 여러분들은 이미 클래스를 활용하고 계셨습니다!

- 여러분들이 지난 과제에서 쓰신 BeautifulSoup, Pandas 등도 클래스로 구성되었습니다.

example.py X

C: > Users > placi > Desktop > example.py

```
1 from bs4 import BeautifulSoup
2 html_doc = """<html><head><title>The Dormouse's story</title></head>
3 <body>
4 <p class="title"><b>The Dormouse's story</b></p>
5 </body>
6 """
7 soup = BeautifulSoup(html_doc, 'html.parser')
8 print(type(soup))
```

```
PS C:\Users\placi\Desktop> py example.py
<class 'bs4.BeautifulSoup'>
```

pandas.DataFrame

```
class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False) ¶
```

[\[source\]](#)

Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

Unit 01 | Intro

그럼 클래스는 뭔가요??

클래스는 **데이터**와 **기능**을 함께 묶는 방법을 제공합니다.

데이터 → 값 → 속성 (attributes)
기능 → 행동 → 매소드 (methods)

새 클래스를 만드는 것은 객체의 새 형을 만들어서, 그 형의 새 인스턴스를 만들 수 있도록 합니다. 각 클래스 인스턴스는 상태를 유지하기 위해 그 자신에게 첨부된 어트리뷰트를 가질 수 있습니다. 클래스 인스턴스는 상태를 바꾸기 위한 (클래스에 의해 정의된) 메서드도 가질 수 있습니다.

출처: Python 공식 문서 (<https://docs.python.org/ko/3/tutorial/classes.html>)

Unit 01 | Intro

Problem 0. 나만의 Zoom을 만들고 싶어요!

화상회의 기업 줌, 1년만에 3조 벌었다...1년새 매출 4배·영업익 52배

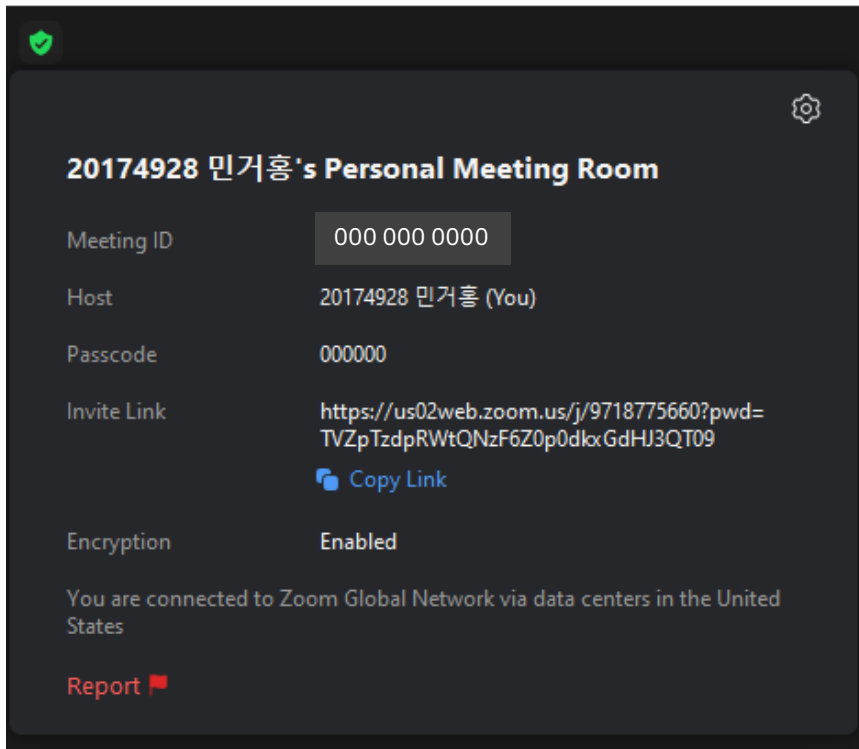
임민철 기자 | 입력 : 2021-03-02 09:46

요즘 Zoom, Google Hangout 등 화상회의 솔루션들이 많이 각광받고 있습니다!

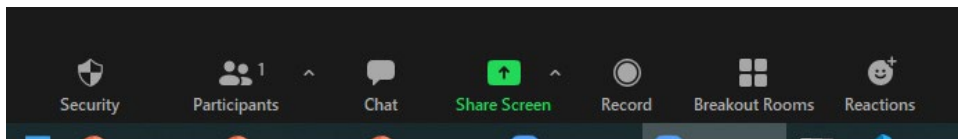
투빅스 구성원들은 이같은 소프트웨어가 돈이 되는 걸 알고, 나만의 Zoom을 만들어보고 싶다고 생각했습니다 :)

줌의 미팅 룸을 아주아주 간단하게나마 클래스를 활용해서 똑똑한 15기가 정의해볼까요?

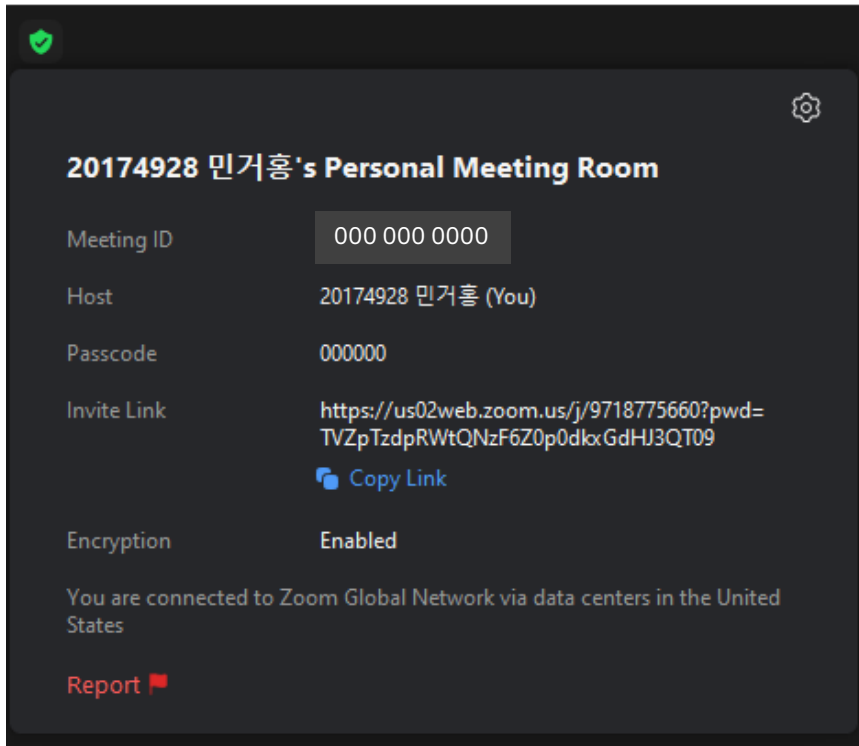
Unit 01 | Intro



일단 단순히 비유하자면
Zoom에서의 미팅 룸과 같은 역할을 하는 게
바로 클래스 (Class) 입니다!



Unit 01 | Intro



Class 미팅룸:

```
self.id = ..
```

```
self.hostname = ..
```

```
self.inviteLink = ..
```

```
def 링크출력(..):
```

```
def 방정보(..):
```

클래스는 크게 2가지 요소로
구성되어 있습니다:

데이터 (속성들)

+

기능 (행동)

Unit 01 | Intro

Class 미팅룸:

`self.id = ..`

`self.hostname = ..`

`self.inviteLink = ..`

`def 링크출력(..):`

`def 방정보(..):`

example.py

C: > Users > placi > Desktop > example.py > ...

```
1 class ZoomMeetingRoom:
2     def __init__(self, id, hostname, inviteLink):
3         self.id = id
4         self.hostname = hostname
5         self.inviteLink = inviteLink
6
7     def print_inviteLink(self):
8         print("초대 링크는:", self.inviteLink)
9
10    def print_roomInfo(self):
11        print(self.hostname, "의 방")
12
```

Unit 01 | Intro

Class 미팅룸:

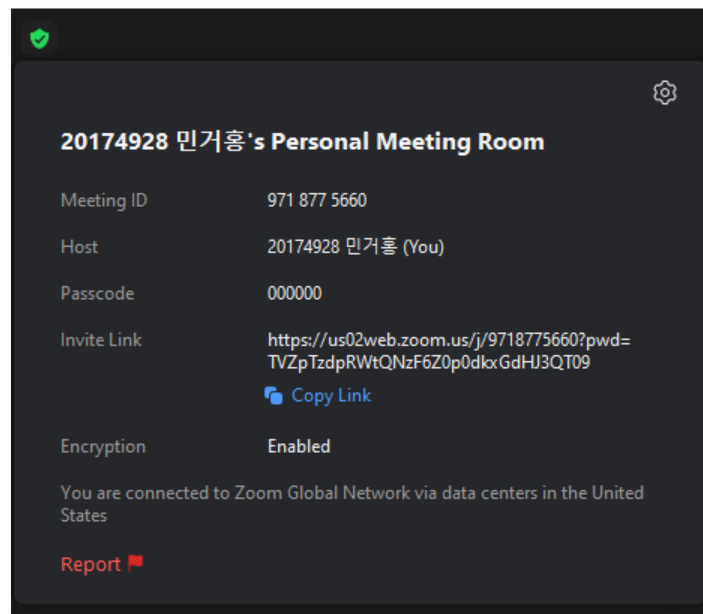
```
self.id = ..
```

```
self.hostname = ..
```

```
self.inviteLink = ..
```

```
def 링크출력(..):
```

```
def 방정보(..):
```



```
tobigs = ZoomMeetingRoom(3221222222, "투빅이", "http://datamarket.kr/x")
naver = ZoomMeetingRoom(1111222222, "네이버", "http://naver.com")
google = ZoomMeetingRoom(2221111111, "구글", "http://google.com")
```

Unit 01 | Intro

Class 미팅룸:

```
self.id = ..
```

```
self.hostname = ..
```

```
self.inviteLink = ..
```

```
def 링크출력(..):
```

```
def 방정보(..):
```

```
tobigs = ZoomMeetingRoom(3221222222, "투빅이", "http://datamarket.kr/xe")
naver = ZoomMeetingRoom(1111222222, "네이버", "http://naver.com")
google = ZoomMeetingRoom(2221111111, "구글", "http://google.com")

tobigs.print_roomInfo()
naver.print_roomInfo()
google.print_roomInfo()
```

```
PS C:\Users\placi\Desktop> py example.py
투빅이 의 방
네이버 의 방
구글 의 방
```

Unit 01 | Intro

클래스는 데이터와 기능을 함께 묶는 방법을 제공합니다.

데이터
(를 담을 수 있는 틀)



기능

```
example.py
C: > Users > placi > Desktop > example.py > ...
1 class ZoomMeetingRoom:
2     def __init__(self, id, hostname, inviteLink):
3         self.id = id
4         self.hostname = hostname
5         self.inviteLink = inviteLink
6
7     def print_inviteLink(self):
8         print("초대 링크는:", self.inviteLink)
9
10    def print_roomInfo(self):
11        print(self.hostname, "의 방")
12
```

Unit 01 | Intro

그래서 왜 배워야 하나요?

1) 구조를 읽은 후에 이해할 수 있어야 쉽게 pandas 등을 활용할 수 있고, 수정할 수 있습니다!

example.py X

```
C: > Users > placi > Desktop > example.py
1 from bs4 import BeautifulSoup
2 html_doc = """<html><head><title>The Dormouse's story</title></head>
3 <body>
4 <p class="title"><b>The Dormouse's story</b></p>
5 </body>
6 """
7 soup = BeautifulSoup(html_doc, 'html.parser')
8 print(type(soup))
```

```
PS C:\Users\placi\Desktop> py example.py
<class 'bs4.BeautifulSoup'>
```

pandas.DataFrame

```
class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False) \[source\]
```

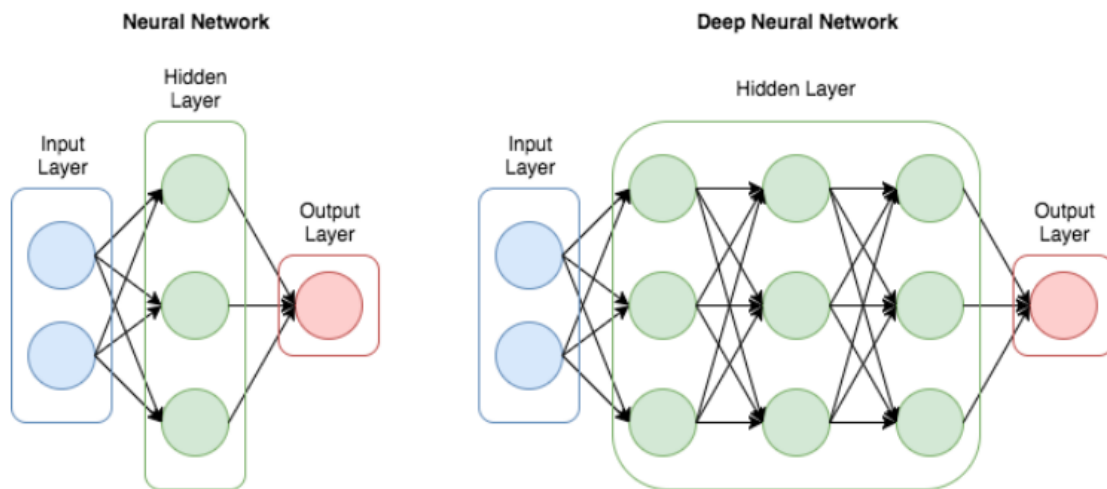
Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

Unit 01 | Intro

그래서 왜 배워야 하나요?

2) 딥러닝 모델을 잘 구축하고 효율적으로 활용과 수정하기 위해서!



복잡한 레이어 구조는 Class를 이용해 구현한다!

Unit 01 | Intro

그래서 왜 배워야 하나요?

2) 딥러닝 모델을 잘 구축하고 효율적으로 활용과 수정하기 위해서!

```
example.py
C: > Users > placi > Desktop > example.py > ...
1 class ZoomMeetingRoom:
2     def __init__(self, id, hostname, inviteLink):
3         self.id = id
4         self.hostname = hostname
5         self.inviteLink = inviteLink
6
7     def print_inviteLink(self):
8         print("초대 링크는:", self.inviteLink)
9
10    def print_roomInfo(self):
11        print(self.hostname, "의 방")
12
```

줌 미팅 룸 틀

```
class MyModel(Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.conv1 = Conv2D(32, 3, activation='relu')
        self.flatten = Flatten()
        self.d1 = Dense(128, activation='relu')
        self.d2 = Dense(10, activation='softmax')

    def call(self, x):
        x = self.conv1(x)
        x = self.flatten(x)
        x = self.d1(x)
        return self.d2(x)

model = MyModel()
```

텐서플로우 mnist 공식 튜토리얼

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 4 * 4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

파이토치 fashion-mnist 공식 튜토리얼

15기 정규세션

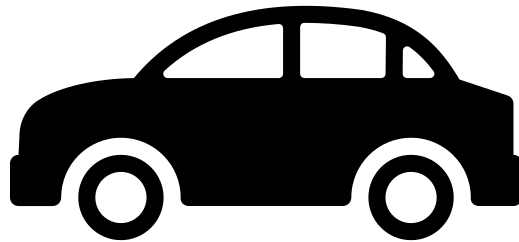
ToBig's 14기 민거홍

02. OOP & Class

Unit 02 | OOP & Class

OOP (파이썬 등) vs. POP (C 등)

- POP (절차 지향 언어) - **절차**에 의해 진행되는 언어/프로그래밍을 실행 **순서**에 의하여 프로그래밍



- 서로 분리되면 안되고, 순서가 틀려서도 안되며 하나가 고장나면 전체기능이 마비되도록 설계 되었을 시
- 핸들을 바꾸고싶을 때 엔진부터 다시 설계해야하며, 바퀴를 만들기전엔 의자도 만들 수 없다.

코드를 활용하려면 엔진부터 바퀴까지 다 바뀌야 되서 재활용성(reusability)이 상대적으로 낮음

Unit 02 | OOP & Class

OOP (파이썬 등) vs. POP (C 등)

객체

지향

프로그래밍

OOP = Object Oriented Programming

각 객체를 중심으로 지향/지시되는 프로그래밍

의의: 프로그램을 명령어의 나열로 보는 시각에서 벗어나, 프로그램도 현실의 대상처럼
각 객체, 즉 여러 대상들의 모임으로 파악하려는 관점

Unit 02 | OOP & Class

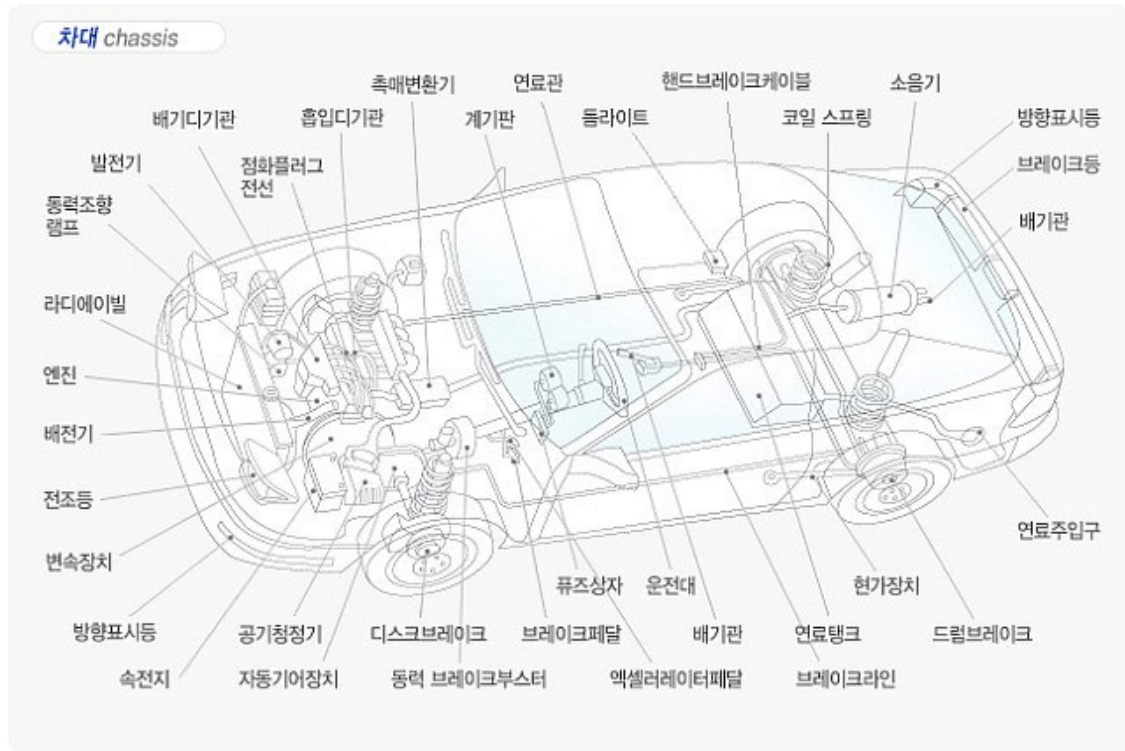
현실의 대상을 볼 때, 어떻게 작동하는 지 (작동 원리)를 중점적으로 보는 것이 아닌, 하나의 커다란 덩어리, 대상을 중심으로 인식.



자동차

Unit 02 | OOP & Class

그 안을 들여다 보면, 바퀴, 차체, 엔진이라는 큰 **속성** + 자동차의 **기능**을 중심으로 보게 됨



자동차

바퀴

차체

달린다!

멈춘다!

차대 chassis

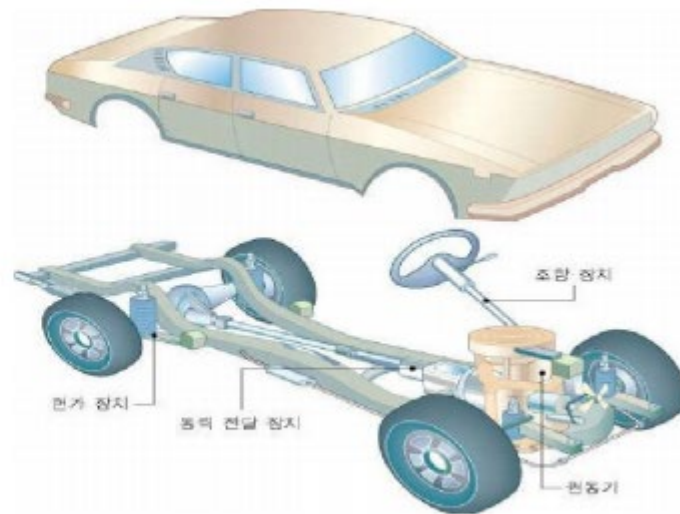
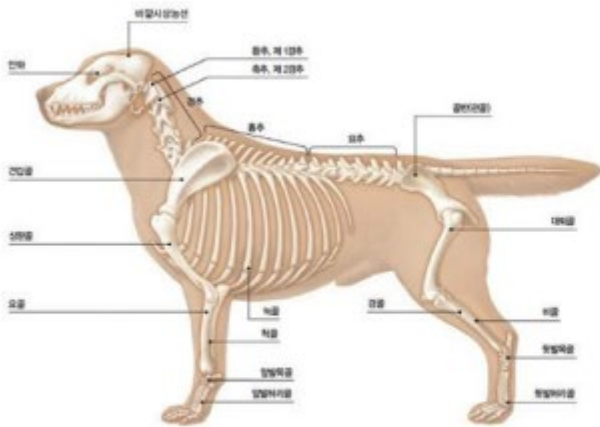


```
def 멈춘다(..):
```

Unit 02 | OOP & Class

OOP관점에서는 어떻게 작동하는지, 구조가 어떻게 구성되었는지는 **크게 상관하지 않습니다.**

큰 덩어리, 대상을 중점적으로 보기 때문에 구조를 생각하지 않아도 자동차를 운전할 수 있고, 강아지와도 놀 수 있습니다.



Unit 02 | OOP & Class

저희가 활용하는 `LogisticRegression()`등도 `class`이기 때문에 직접 복잡한 수식 구현을 할 필요가 없이, 우리가 쉽게 사용할 수 있습니다 😊

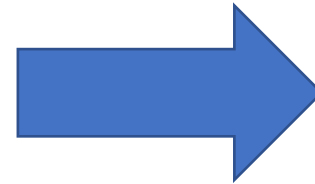
`sklearn.linear_model.LogisticRegression`

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False,
tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None,
random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0,
warm_start=False, n_jobs=None, l1_ratio=None) [source]
```

```
In [14]: # Logistic Regression
classifier = LogisticRegression()
classifier.fit(X, y)

C:\Users\WLG\venv\Wjbeen\lib\site-packages\W
0.22. Specify a solver to silence this war
FutureWarning)

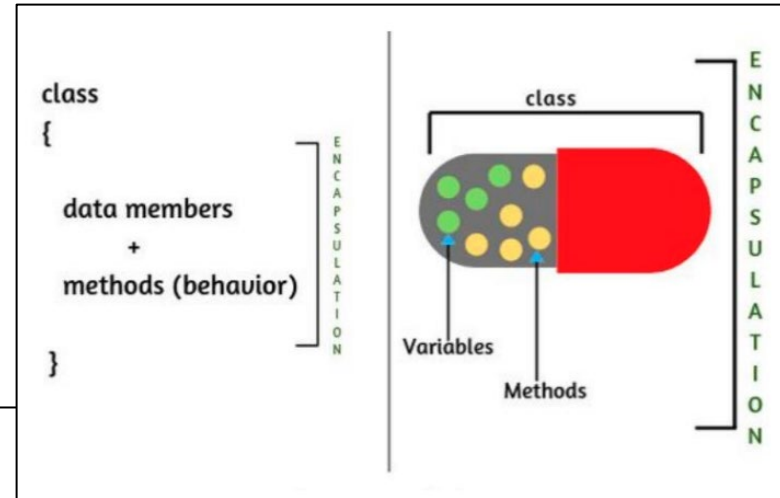
Out[14]: LogisticRegression(C=1.0, class_weight=Nor
intercept_scaling=1, max_iter=100
n_jobs=None, penalty='l2', randk
tol=0.0001, verbose=0, warm_stai
```



Encapsulation
(캡슐화)

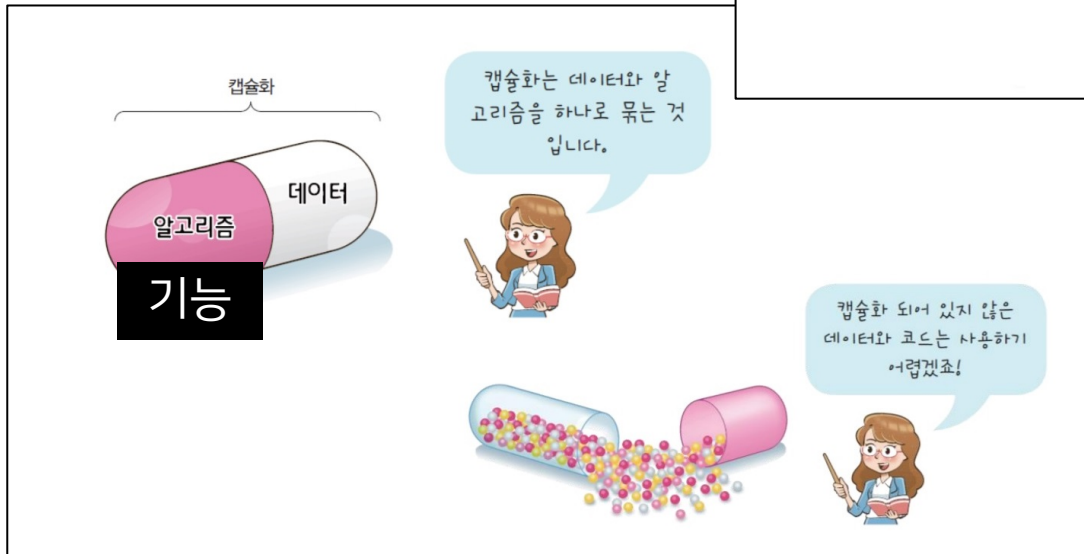
Unit 02 | OOP & Class

캡슐화는..



encapsulation을 통하여 데이터(variable)와 기능(method)를 합칩니다.

이를 통해 **복잡한 내용을 숨기고**, 함수만 있을 때보다 체계적으로 데이터를 구분과 관리할 수 있습니다.



Unit 02 | OOP & Class

붕어빵을 예시를 들면,



이때 이런 틀 자체를 **클래스(class)**,
클래스를 통해 만들어진 객체를 해당 클래스의 **인스턴스(instance)** 라고 부릅니다

Unit 02 | OOP & Class

클래스는 데이터와 기능을 함께 묶는 방법을 제공합니다.

클래스

데이터
(를 담을 수 있는 틀)



기능

실체화된, 만들어진
객체는 인스턴스

```
example.py > ...
1 class ZoomMeetingRoom:
2     def __init__(self, id, hostname, inviteLink):
3         self.id = id
4         self.hostname = hostname
5         self.inviteLink = inviteLink
6
7     def print_inviteLink(self):
8         print("초대 링크는:", self.inviteLink)
9
10    def print_roomInfo(self):
11        print(self.hostname, "의 방")
12
13    tobigs = ZoomMeetingRoom(322122222, "투빅이", "http://datamarket.kr/xe")
```

15기 정규세션

ToBig's 14기 민거홍

Unit 03: Class 구조와 상속

Unit 03 | Class 구조와 상속

자동차를 예시로 진행해보겠습니다.

Class 자동차:

```
def __init__ (...)
```

```
self. 바퀴 크기 = ..
```

```
self. 차체 색깔 = ..
```

```
def 달린다! (..):
```

```
def 멈춘다! (..):
```

Class 자동차:

```
def __init__ (...)
```

```
self. 바퀴 크기 = ..
```

```
self. 차체 색깔 = ..
```

```
def 달린다! (..):
```

```
def 멈춘다! (..):
```

```
class Car:
```

```
def __init__(self, color = "white", wheel_size=14):  
    self.color = color  
    self.wheel_size = wheel_size
```

```
def drive(self):  
    print("driving")
```

```
def brake(self):  
    print("brake")
```

```
def showInfo(self):  
    print("color: ", self.color,  
          ", wheel size: ", self.wheel_size)
```

Unit 03 | Class 구조와 상속

__init__, self는 필수예요!

```
class Car:
    def __init__(self, color, wheel_size):
        self.color = color
        self.wheel_size = wheel_size

    def drive(self):
        print("driving")

    def brake(self):
        print("brake")

    def showInfo(self):
        print("color: ", self.color,
              ", wheel size: ", self.wheel_size)
```

__init__: 생성자, initialize 매직메소드. 인스턴스를 만들 때 자동으로 실행된다. 파라미터에 Default로 들어갈 값을 설정해줄 수도 있다.

```
class Car:
    def __init__(self, color = "white", wheel_size=14):
        self.color = color
        self.wheel_size = wheel_size
```

self: 현재 돌고있는 인스턴스 자기 자신을 의미. 생성자에서 인스턴스 변수를 선언할 때 self.~~의 형태로 선언
인스턴스 메소드(클래스의 함수)의 파라미터로 꼭 들어간다.

Unit 03 | Class 구조와 상속

매개변수 (parameter), 전달인자 (argument)

```
class Car:
    def __init__(self, color, wheel_size):
        self.color = color
        self.wheel_size = wheel_size

    def drive(self):
        print("driving")

    def brake(self):
        print("brake")

    def change_color(self, new_color):
        self.color = new_color

    def showInfo(self):
        print("color: ", self.color,
              ", wheel size: ", self.wheel_size)
```

```
myCar = Car("red", 16)
```

```
myCar.drive()
myCar.showInfo()
```

```
driving
color: red , wheel size: 16
```

매개변수는 함수의 정의 부분에서 볼 수 있으며,

전달인자는 함수를 호출하는 부분에서 볼 수 있다.

여기서 매개변수는 color와 wheel_size이며,

전달인자는 “red”와 16이다.

Class Car에 (“red”, 16) 인자를 넘기면, __init__이 실행되며 class안의 속성, 메소드 정보가 myCar에 담긴다.

Unit 03 | Class 구조와 상속

디폴트 전달인자 (argument), 왜 필요한가?

```
class Car:
    def __init__(self, color, wheel_size):
        self.color = color
        self.wheel_size = wheel_size

    def drive(self):
        print("driving")

    def brake(self):
        print("brake")

    def change_color(self, new_color):
        self.color = new_color

    def showInfo(self):
        print("color: ", self.
              ", wheel size: ")
```

```
myCar = Car()

myCar.drive()
myCar.showInfo()
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-11-9dad57e3e7f3> in <module>
    18
    19
--> 20 myCar = Car()
    21
    22 print(myCar.color)
```

TypeError: __init__() missing 2 required positional arguments: 'color' and 'wheel_size'

디폴트 전달인자가 없을 시,
모든 전달인자가 있어야한다.

없으면 error..

Unit 03 | Class 구조와 상속

디폴트 전달인자 (argument), 왜 필요한가?

```
class Car:
    def __init__(self, color = "white", wheel_size=14):
        self.color = color
        self.wheel_size = wheel_size

    def drive(self):
        print("driving")

    def brake(self):
        print("brake")

    def showInfo(self):
        print("color: ", self.color,
              ", wheel size: ", self.wheel_size)
```

```
car = Car()
myCar = Car("red", 16)
tobigsCar = Car("blue", 20)
```

```
car.showInfo()
myCar.showInfo()
tobigsCar.showInfo()
```

```
color:  white , wheel size:  14
color:  red , wheel size:  16
color:  blue , wheel size:  20
```

Default 값을 설정해두고, 파라미터를 비워두면 default 값이 들어간 인스턴스 생성

Unit 03 | Class 구조와 상속

디폴트 전달인자 (argument), 왜 필요한가?

```
class Car:
    def __init__(self, color = "white", wheel_size=14):
        self.color = color
        self.wheel_size = wheel_size

    def drive(self):
        print("driving")

    def brake(self):
        print("brake")

    def showInfo(self):
        print("color: ", self.color,
              ", wheel size: ", self.wheel_size)
```

```
car = Car()
myCar = Car("red", 16)
tobigsCar = Car("blue", 20)
```

```
car.showInfo()
myCar.showInfo()
tobigsCar.showInfo()
```

```
color:  white , wheel size:  14
color:  red , wheel size:  16
color:  blue , wheel size:  20
```

같은 클래스를 썼지만 각 인스턴스마다 다른 데이터가 들어가 구현됨 => 다른 속성과 메소드를 갖는다
Car라는 같은 클래스를 썼지만, myCar 객체와 tobigsCar 객체가 다르다 ☺

Unit 03 | Class 구조와 상속

데이터 (arguments) 접근 방법

```
class Car:
    def __init__(self, color = "white", wheel_size=16):
        self.color = color
        self.wheel_size = wheel_size

    def drive(self):
        print("driving")

    def brake(self):
        print("brake")

    def change_color(self, new_color):
        self.color = new_color

    def showInfo(self):
        print("color: ", self.color,
              ", wheel size: ", self.wheel_size)
```

```
myCar = Car()
myCar.change_color("red")
myCar.wheel_size = 16

print(myCar.color)
print(myCar.wheel_size)

myCar.drive()
myCar.showInfo()

red
16
driving
color: red , wheel size: 16
```

직접 접근하거나, 메소드를 이용하여
값을 바꿀 수 있음

Unit 01 | Intro

그럼 클래스는 뭔가요??

클래스는 **데이터**와 **기능**을 함께 묶는 방법을 제공합니다.

데이터 → 값 → 속성 (attributes)
기능 → 행동 → 매소드 (methods)

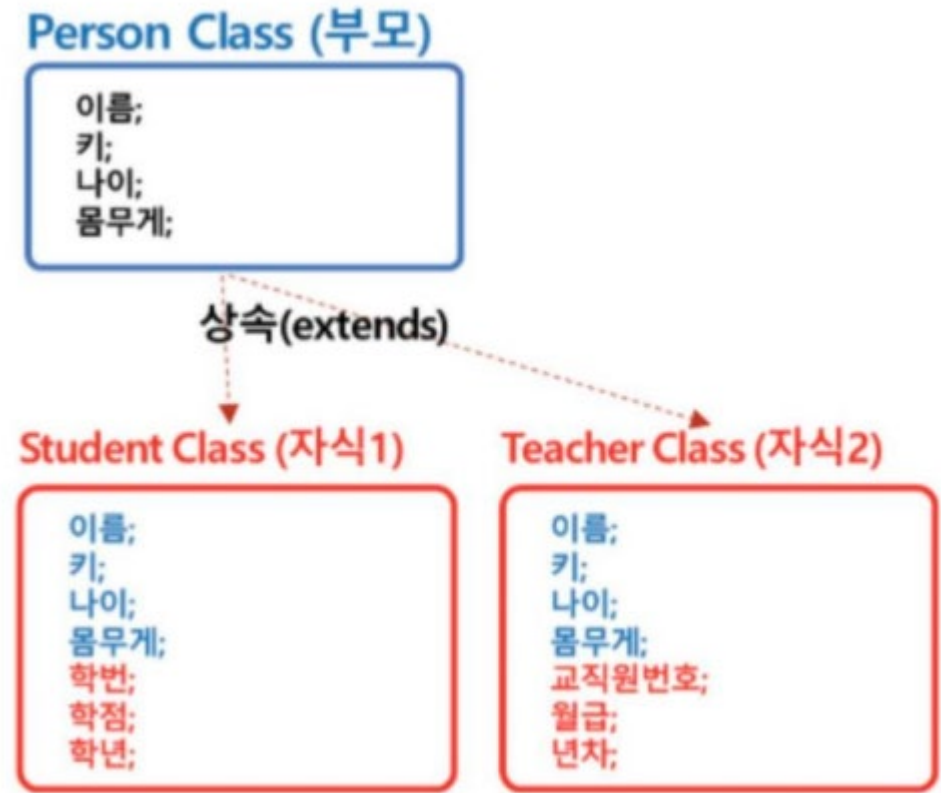
새 클래스를 만드는 것은 **객체**의 새 형을 만들어서, 그 형의 새 인스턴스를 만들 수 있도록 합니다. 각 클래스 **인스턴스**는 상태를 유지하기 위해 그 자신에게 첨부된 **어트리뷰트**를 가질 수 있습니다. 클래스 인스턴스는 상태를 바꾸기 위한 (클래스에 의해 정의된) **메서드**도 가질 수 있습니다.

출처: Python 공식 문서 (<https://docs.python.org/ko/3/tutorial/classes.html>)

Unit 03 | Class 구조와 상속

Inheritance (상속)이란?

부모에게서 유산을 상속(inheritance)받듯이
부모에게서 속성과 메소드를 그대로 가져와서
자식이 똑같은 코드를 다시 작성할 필요 없이
사용할 수 있게 해주는 것
보통 부모가 더 추상적이고 자식이 더 구체적인 class를 가진다.



Unit 03 | Class 구조와 상속

Override (상속)이란?

Translations of override

명사

보수

[compensation](#), [remuneration](#), [reward](#), [fee](#), [conservation](#), [override](#)

장치가 작동하지 않도록 하는 시스템

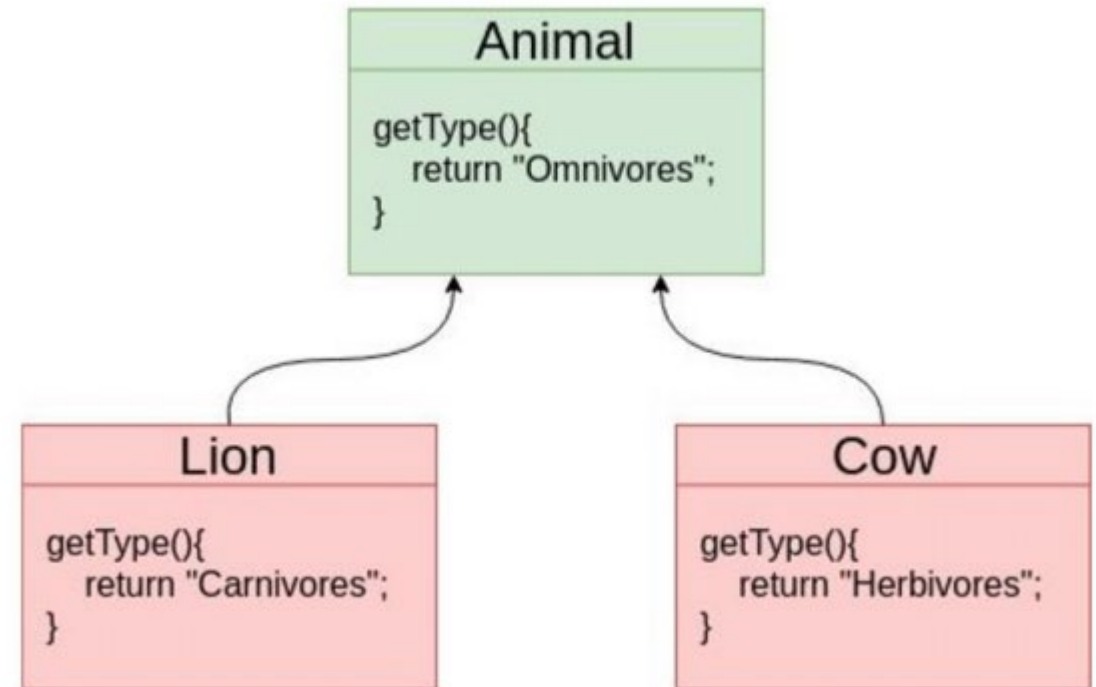
[override](#)

무효로 함

[invalidation](#), [rescission](#), [override](#), [defeat](#)

Show more ▼

부모에게서 받은 유산이 마음에 들지 않는다면
자식단에서 상속 받은 것을 커스텀하는 것, 즉 덮어쓰기
이때 부모의 클래스는 영향을 받지 않음



Unit 03 | Class 구조와 상속

상속 예시

```
class Car:
    def __init__(self, color, wheel_size):
        self.color = color
        self.wheel_size = wheel_size

    def drive(self):
        print("driving")

    def brake(self):
        print("brake")

    def change_color(self, new_color):
        self.color = new_color

    def showInfo(self):
        print("color: ", self.color,
              ", wheel size: ", self.wheel_size)
```

```
class Bus(Car):
    def __init__(self, color="green", wheel_size=24, busstop=" "):
        super().__init__(color, wheel_size)
        self.busstop = busstop

    def transfer(self):
        print("환승입니다")

    def brake(self):
        print("이번 정류장은 ", self.busstop, "입니다.")

bus = Bus()
bus.busstop = "연세대학교 정문"

bus.drive()
bus.brake()
bus.transfer()
bus.showInfo()

driving
이번 정류장은 연세대학교 정문 입니다.
환승입니다
color: green , wheel size: 24
```


Unit 03 | Class 구조와 상속

상속 예시

```
class Bus(Car):
    def __init__(self, color="green", wheel_size=24, busstop=" "):
        super().__init__(color, wheel_size)
        self.busstop = busstop

    def transfer(self):
        print("환승입니다")

    def brake(self):
        print("이번 정류장은 ", self.busstop, "입니다.")

bus=Bus()
bus.busstop="연세대학교 정문"

bus.drive()
bus.brake()
bus.transfer()
bus.showInfo()

driving
이번 정류장은 연세대학교 정문 입니다.
환승입니다
color: green , wheel size: 24
```

Class Bus (상속 받을 부모)

super(): 부모의 class를 의미,
super().__init__()으로 부모의 __init__() 함수 실행
= 부모 class의 메소드와 속성을 사용할 수 있음
부모의 다른 함수 역시 같은 방식으로 실행 가능

drive, showinfo는 부모에게서 상속
brake는 오버라이딩
transfer는 자식이 자체적으로 추가

=> Reusability

Unit 03 | Class 구조와 상속

데이터 (arguments) 접근 방법

```
class Car:
    def __init__(self, color = "white", wheel_size=16):
        self.color = color
        self.wheel_size = wheel_size

    def drive(self):
        print("driving")

    def brake(self):
        print("brake")

    def change_color(self, new_color):
        self.color = new_color

    def showInfo(self):
        print("color: ", self.color,
              ", wheel size: ", self.wheel_size)
```

```
myCar = Car()
myCar.change_color("red")
myCar.wheel_size = 16

print(myCar.color)
print(myCar.wheel_size)

myCar.drive()
myCar.showInfo()

red
16
driving
color: red , wheel size: 16
```

직접 접근하거나, 메소드를 이용하여
값을 바꿀 수 있음

Unit 03 | Class 구조와 상속

Pytorch nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

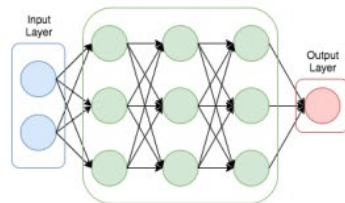
Modules can also contain other Modules, allowing to nest them in a tree structure.

regular attributes:

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

class TwoLayerNet(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)

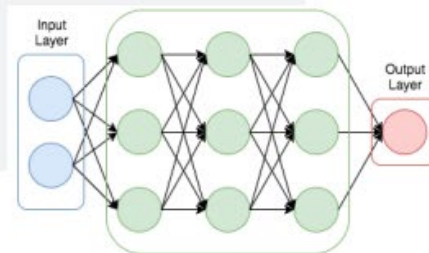
    def forward(self, x):
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred
```



Unit 03 | Class 구조와 상속

Tensorflow tf.keras.Model

```
class MyModel(tf.keras.Model):  
  
    def __init__(self):  
        super(MyModel, self).__init__()  
        self.dense1 = tf.keras.layers.Dense(4, activation=tf.nn.relu)  
        self.dense2 = tf.keras.layers.Dense(5, activation=tf.nn.softmax)  
  
    def call(self, inputs):  
        x = self.dense1(inputs)  
        return self.dense2(x)  
  
model = MyModel()
```



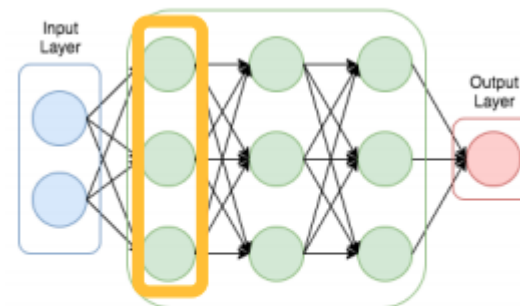
Unit 03 | Class 구조와 상속

Tensorflow keras.layers.Layer

Keras의 중심 추상화 중 하나는 `Layer` 클래스입니다. 레이어는 상태 (레이어의 "가중치")와 입력에서 출력으로의 변환 ("호출", 레이어의 순방향 패스)을 모두 캡슐화합니다.

여기에 조밀하게 연결된 레이어가 있습니다. 상태가 있습니다: 변수 `w` 및 `b`.

```
class Linear(keras.layers.Layer):  
    def __init__(self, units=32, input_dim=32):  
        super(Linear, self).__init__()  
        w_init = tf.random_normal_initializer()  
        self.w = tf.Variable(  
            initial_value=w_init(shape=(input_dim, units), dtype="float32"),  
            trainable=True,  
        )  
        b_init = tf.zeros_initializer()  
        self.b = tf.Variable(  
            initial_value=b_init(shape=(units,), dtype="float32"), trainable=True  
        )  
  
    def call(self, inputs):  
        return tf.matmul(inputs, self.w) + self.b
```



```
linear_layer = Linear(4, 2)  
y = linear_layer(x)
```

Unit 03 | Class 구조와 상속

이처럼, NN 중 다중 입력/출력 모델, layer를 공유하는 모델, 데이터 흐름이 차례대로
진행되지 않는 모델 (ex resnet) 등은 단순히 층을 쌓는 것만으로는 구현하기 어렵다 =>

상속 (자식클래스=subclass) 통해 사용자 정의 모델을 구현해야 함

15기 정규세션

ToBig's 14기 민거홍

Unit 04: Magic Method

Unit 04 | Magic Method

모든 건 class..

```
i = 3  
s = "tobigs"  
  
print(type(i))  
print(type(s))
```

```
<class 'int'>  
<class 'str'>
```

파이썬은 "객체지향언어" 이므로 모든 것이 class로 구현되어있다.
Int, string, list 등도 파이썬에 내장되어 있던 클래스

즉, 우리는 우리도 모르는 사이에
붕어빵 틀에 반죽을 붓듯이
자료형 틀에 데이터를 부어서 사용했던 것!

Unit 04 | Magic Method

Intrinsic methods (내장된 메소드)

```
print(dir(type(i)))
```

 사용하기 쉽게 +에 add가 맵핑되어있음

```
['_abs_', '__add__', '__and__', '__bool__', '__cell__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__', '__format__', '__ge__', '__getattr__', '__getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

```
print(dir(type(s)))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Class int와 Class string이 갖고 있는 메소드!

Unit 04 | Magic Method

메소드 사용법: . 을 쓴다.

```
myCar = Car()
myCar.change_color("red")
myCar.wheel_size = 16

print(myCar.color)
print(myCar.wheel_size)

myCar.drive()
myCar.showInfo()
```

```
red
16
driving
color: red , wheel size: 16
```

```
s = "t o b i g s"
l = s.split()
print(l)

i1 = 1
i2 = 3
result = i1.__add__(i2) # == i1 + i2
print(result)
```

```
['t', 'o', 'b', 'i', 'g', 's']
4
```

. 을 통해 해당 인스턴스가 갖고있는 메소드를 사용할 수 있다.

Unit 04 | Magic Method

매직 메소드 (__init__ 등등)

```
print(dir(type(s)))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

이 중 더블 언더스코어(__)가 있는 메소드들을 Magic Method라고 부른다 (__init__도 이 중 하나)

파이썬이 내부적으로 구현해놓은 메소드!

(ex 사칙연산, len(), int(), str() 등)

Unit 04 | Magic Method

우리만의 class

```
example.py  twoD_vector.py X
twoD_vector.py > ...
1  class TwoDVector:
2      def __init__(self, x, y):
3          self.x = x
4          self.y = y
5
6  v1 = TwoDVector(1, 3)
7  v2 = TwoDVector(2, 0)
8
9  v3 = v1 + v2
10 print(v3)
```

```
SyntaxError: missing parentheses in call to 'print'. Did
PS C:\Users\placi\Desktop> py twoD_vector.py
Traceback (most recent call last):
  File "twoD_vector.py", line 9, in <module>
    v3 = v1 + v2
TypeError: unsupported operand type(s) for +: 'TwoDVector' and 'TwoDVector'
```

```
i1 = 1
i2 = 2
i3 = i1 + i2
print(i3)
```

3

+ 는 `__add__`가 맵핑된 기호

그러나 `int` 클래스와 달리

우리가 만든 클래스에는 없으니

해당 메소드를 직접 정의해야 한다

Unit 04 | Magic Method

우리만의 class

```
class TwoDVector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return TwoDVector(self.x + other.x, self.y + other.y)

    def __repr__(self):
        return "TwoDVector({}, {})".format(self.x, self.y)
```

```
v1 = TwoDVector(1, 3)
v2 = TwoDVector(2, 0)
```

```
v3 = v1 + v2
print(v3)
```

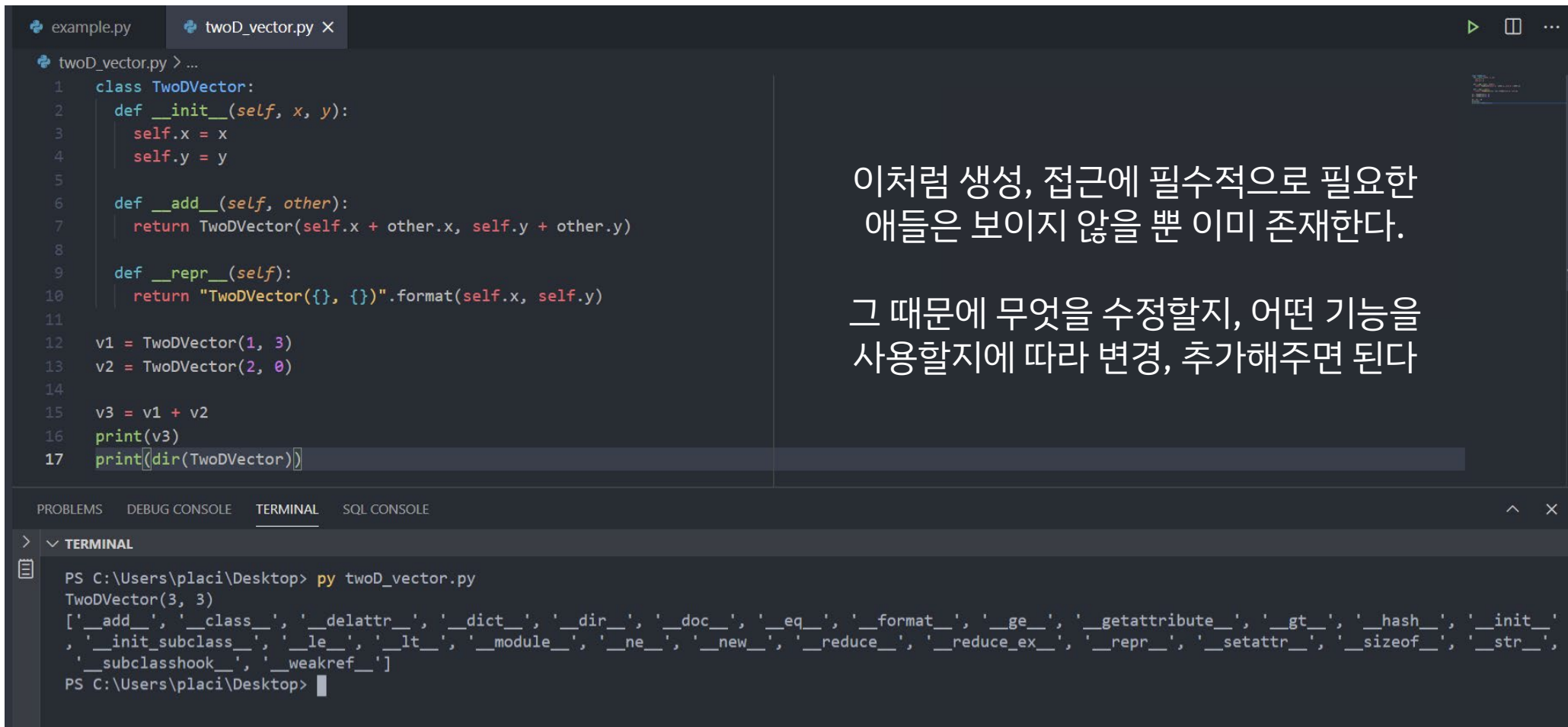
```
PS C:\Users\placi\Desktop> py twoD_vector.py
TwoDVector(3, 3)
```

덧셈 정의

출력 정의

똑같은 이름의 메소드를 정의해주면
기존 자료형 (클래스)에서 사용하던 메소드와
기호를 직접 만든 클래스에서도 사용할 수 있다

Unit 04 | Magic Method



```
example.py twoD_vector.py X
twoD_vector.py > ...
1 class TwoDVector:
2     def __init__(self, x, y):
3         self.x = x
4         self.y = y
5
6     def __add__(self, other):
7         return TwoDVector(self.x + other.x, self.y + other.y)
8
9     def __repr__(self):
10        return "TwoDVector({}, {})".format(self.x, self.y)
11
12 v1 = TwoDVector(1, 3)
13 v2 = TwoDVector(2, 0)
14
15 v3 = v1 + v2
16 print(v3)
17 print(dir(TwoDVector))
```

이처럼 생성, 접근에 필수적으로 필요한
애들은 보이지 않을 뿐 이미 존재한다.

그 때문에 무엇을 수정할지, 어떤 기능을
사용할지에 따라 변경, 추가해주면 된다

PROBLEMS DEBUG CONSOLE TERMINAL SQL CONSOLE

> TERMINAL

```
PS C:\Users\placi\Desktop> py twoD_vector.py
TwoDVector(3, 3)
['__add__', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__',
 '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
 '__subclasshook__', '__weakref__']
PS C:\Users\placi\Desktop>
```

15기 정규세션

ToBig's 14기 민거홍

Bonus

Bonus | 모듈

모듈 (module)

모듈은 데이터, 함수, 클래스 등이 담겨져있는 파일 입니다.

평소에 사용하던 `print()` 와 같은 함수는 내장함수(Built in function)을 이용해서 사용을 했지만, 외부의 라이브러리(모듈)을 사용하기 위해서는 따로 `import` 하는 작업이 필요합니다.

import 모듈명

import 모듈명1, 모듈명2, ...

Bonus | 모듈

예: math module

상단에 math 모듈을 import 했기 때문에 math안에 들어있는 함수나 변수 등을 사용할 수 있습니다.

사용할때는 import 에 선언한 모듈명을 앞에쓰고 그 뒤에 사용할 함수나 변수 등을 사용하시면 됩니다. (math.xxx)

example_math_module.py X

C: > Users > placi > Desktop > example_math_module.py > ...

```
1  import math # 외장 모듈 math 불러 오기
2
3  print(math.pi) # 파이의 값
4  print(math.sqrt(4)) # 제곱근
5  print(math.pow(2,3)) # 제곱 2^3
6  |
```

```
hello world
PS C:\Users\placi\Desktop> py example_math_module.py
3.141592653589793
2.0
8.0
```

Bonus | 모듈

원하는 이름으로 import

만약 내가 원하는 이름으로 모듈을 활용하고 싶을때는 import ~ as ~ 를 사용하시면 됩니다.

import 모듈명 **as** 원하는 이름

import 모듈명1 **as** 원하는 이름, 모듈명2 **as** 원하는 이름, ...

example_math_module.py X

C: > Users > placi > Desktop > example_math_module.py > ...

```
1  import math as mt # 외장 모듈 math 불러오기
2  print(mt.pi) # 파이의 값
3  print(mt.sqrt(4)) # 제곱근
4  print(mt.pow(2,3)) # 제곱 2^3
```

Bonus | if __name__ == "__main__"

if __name__ == "__main__"

지금 파이썬 파일을 직접 실행 한 거라면 (import하지 않았고, 직접 실행했다면), 아래 코드들도 실행하라.

```
example.py  example_math.py  example_import.py X
C: > Users > placi > Desktop > example_import.py > ...
1  import example_math
2
3  print("hello world")
4
PS C:\Users\placi\Desktop> py example_import.py
hello world
PS C:\Users\placi\Desktop>
```

```
example.py  example_math.py X
C: > Users > placi > Desktop > example_math.py > ...
1  def add (a, b):
2      return a + b
3
4  def sub (a, b):
5      return a - b
6
7  if __name__ == "__main__":
8      print(add(3, 5))
9      print(sub(3, 2))
PS C:\Users\placi\Desktop> py example_math.py
8
1
PS C:\Users\placi\Desktop>
```

Unit 01 | Intro

Unit 02 | OOP & Class 개념

Unit 03 | Class의 구조와 상속

Unit 04 | Magic Method

+ (Bonus) 모듈, __main__

참고자료

- 13기 조혜원님의 Class 강의
- 12기 조민호님의 Class 강의
- 11기 이소라님의 Class 강의
- Python, Pytorch, Tensorflow 공식 문서
- 모듈 정리:
- 매직메소드 정리 <https://corikachu.github.io/articles/python/python-magic-method>

Q & A

들어주셔서 감사합니다.