Python 객체지향 프로그래밍

클래스란?

나타내고자 하는 개념의 설계도

SNS 게시물

저장해야 하는 데이터

작성자
내용
작성한 날짜와 시간
(선택) 이미지
(선택) 외부 URL 링크

SNS 게시물

할 수 있는 조작

리액션(좋아요)
댓글 달기
내용 수정하기
공유하기
사진 추가하기

클래스와 인스턴스

클래스

어떤 데이터가 있는지
어떤 조작을 할 수 있는지
어떤 제약조건들이 있는지
명시한 추상적인 설계도

인스턴스

그 클래스로 만든 실제 예시

클래스와 인스턴스

게시물 클래스

게시물 하나에 최대 20장의 사진
댓글을 달 수 있음
작성자가 공유 여부를 설정 가능

게시물 인스턴스

elice의 10월 08일 게시물
김XX 님의 03월 12일 게시물
samxxxkxxxx 계정의 10월 05일 게시물
...

클래스 선언

```
class Post :
    author = None
    likes = 0
    content = "What are you doing?"
```

속성

클래스 선언

```
class Post :
  ...

  def like(self, user) :
    self.likes += 1
    user.liked_posts.append(self)
```

메소드

생성자

모든 클래스의 가장 기본이 되는 메소드

인스턴스가 처음 만들어질 때, 어떻게 세팅할 것인지 결정

생성자

```
class Post :

    def __init__ (self, author, content) :
        self.author = author
        self.content = content
```

생성자

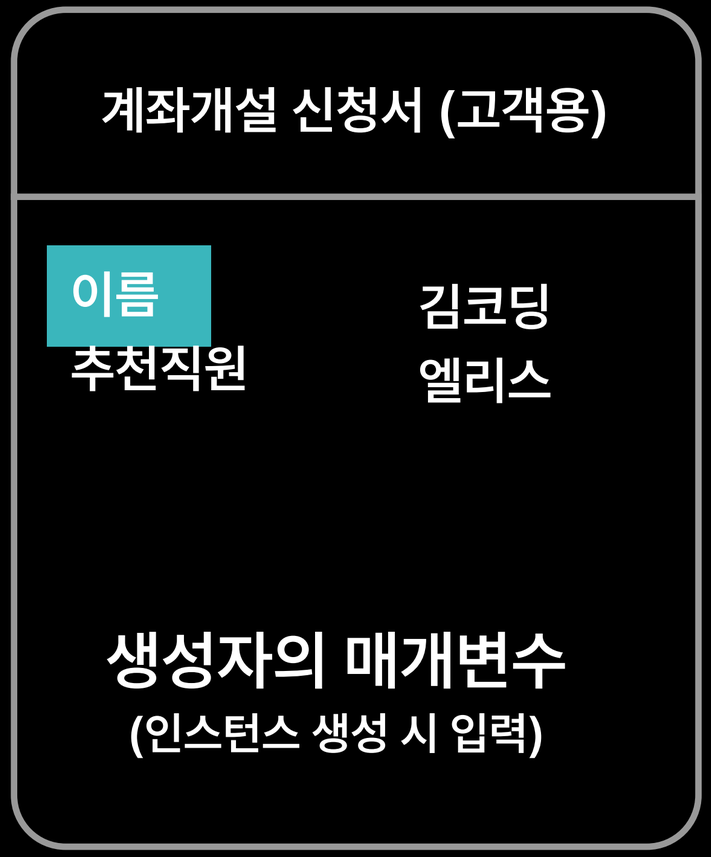
객체가 만들어질 때,
자동으로 호출되는 메서드

생성자의 매개변수 vs 클래스의 속성

계좌개설 신청서 (고객용)	
이름	김코딩
추천직원	엘리스

계좌개설 확인서 (내부용)	
고객명	김코딩
추천직원번호	2435

생성자의 매개변수 vs 클래스의 속성

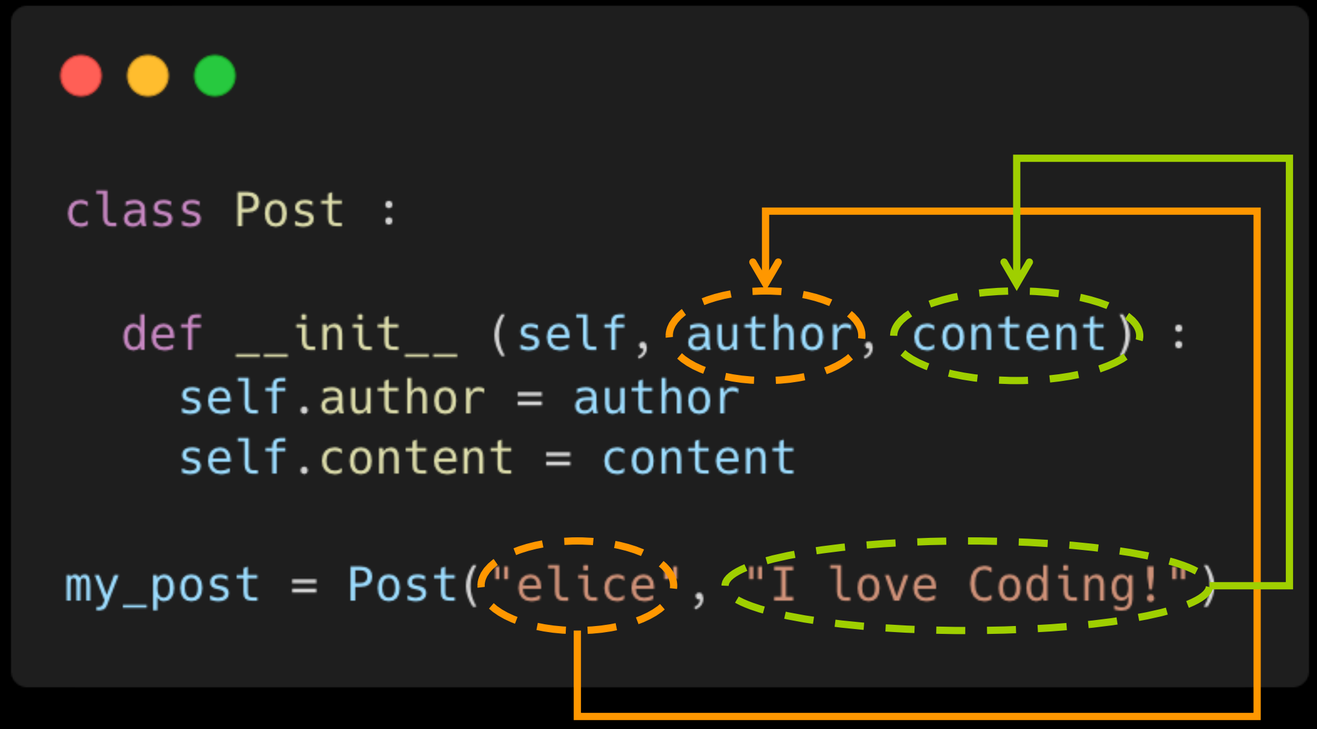


생성자

```
class Post :

  def __init__ (self, author, content) :
    self.author = author
    self.content = content
```

클래스 내부의 속성/메소드에 접근할 때

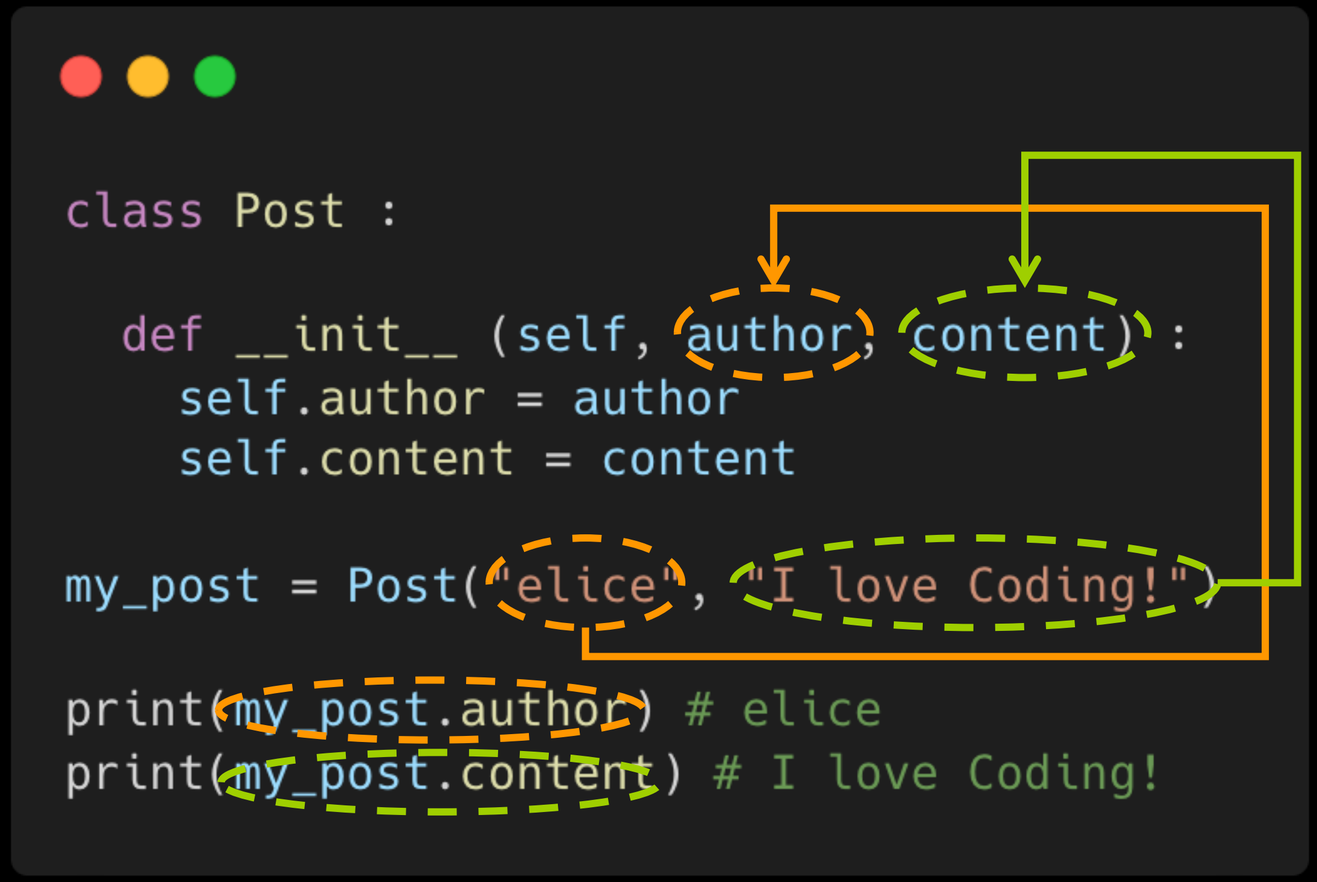
생성자



생성자

객체가 만들어질 때,
자동으로 호출되는 메서드

생성자



```
class Post :

  def __init__ (self, author, content) :
    self.author = author
    self.content = content

my_post = Post("elice", "I love Coding!")

print(my_post.author) # elice
print(my_post.content) # I love Coding!
```

속성을 만들 때 주의할 점

```
class Post :
  def __init__(self, author, content) :
    self.likes = 0
    self.liked_users = []
```

같은 내용을 나타내는 속성이 2개

- likes와 like_count가 사실상 같은 정보를 나타냄
- 코드 작성자나 협업자가 헷갈릴 수 있으며
- 이후 값이 따로따로 변경되면 데이터 불일치 발생할 수 있음

속성을 만들 때 주의할 점

```
class Post :
  def __init__ (self, author, content) :
    ...


my_post = Post("elice", "I love Coding!")
my_post.likes+=1
```

메소드 만들기

```
class Post :
  def num_likes(self) :
    return len(self.liked_users)
```

메소드를 속성처럼 사용

메소드 만들기

```
class Post:
  def __init__(self):
    self.liked_users = []

  def num_likes(self):
    return len(self.liked_users)

my_post = Post()
my_post.liked_users.append("elice")

print(my_post.num_likes()) # 1
```

원하지 않는 값 배제하기

```
class Post :
  def __init__ (self, author, content):
    ...

my_post = Post("elice",1457)
my_post.like(["Hello", "World"])
```

잘못된 값

원하지 않는 값 배제하기 (1)

```
class User :
  def __init__ (self, year_of_birth) :
    if type(year_of_birth) is not int :
      return
```

year_of_birth 의 데이터 타입을 검사

- 숫자(int) 가 아니라면 return 명령어 실행 → **초기화 중단**

원하지 않는 값 배제하기 (1)

```python
class User:
  def __init__(self, year_of_birth):
      if type(year_of_birth) is not int:
        return

      self.year_of_birth = year_of_birth

user1 = User(1994)
user2 = User("1994")

print(user1.year_of_birth) # 1994
print(user2.year_of_birth)
# AttributeError: 'User' object has no attribute 'year_of_birth'
```

예상하지 못한 타입의 데이터가
들어오는 것을 사전에 방지

원하지 않는 값 배제하기 (2)

```python
class User:
  def __init__(self, name):
    self.name = name


class Post:
  def __init__(self, author, content):
    if not isinstance(author, User):
      return
    if type(content) is not str:
      return
```

조건 1
author는 반드시 User 클래스의 인스턴스여야 함

조건 2
content는 반드시 문자열이어야 함

isinstance()는 객체가 특정 클래스(또는 그 하위 클래스)의 인스턴스인지 확인해 True 또는 False를 반환하는 함수

[ref] isinstance??

```
isinstance(객체, 클래스 or 타입)
```

객체의 자료형(타입)을 검사할 때 사용하는 Python 내장 함수

- 첫 번째 인자 (객체) : 검사할 객체
- 두 번째 인자 (클래스 or 타입) : 확인할 클래스 또는 타입
- 반환값 : True or False

```
print(isinstance(5, int))         # True
print(isinstance("hello", str))   # True
print(isinstance(3.14, int))      # False
```

원하지 않는 값 배제하기 (2)

```python
class User:
    def __init__(self, name):
        self.name = name

class Post:
    def __init__(self, author, content):
        if not isinstance(author, User):
            return
        if type(content) is not str:
            return

        self.author = author
        self.content = content
```

```python
author = User("elice")
my_post = Post(author, "I love coding!")

print(my_post.author.name)  # elice
print(my_post.content)      # I love coding!
```

조건 1 : 만족 & 조건 2 : 만족

```python
your_post = Post("hellobit", "Hello World!")

print(your_post.author.name)
# AttributeError: 'Post' object has no attribute 'author'
print(your_post.content)
```

조건 1 : 불만족 & 조건 2 : 만족

원하지 않는 값 배제하기 (3)

```python
class User :
  def __init__ (self, year_of_birth) :
    if year_of_birth > 2005 :
      raise Exception("Too young")
```

- *year_of_birthday* 가 2005년보다 큰 경우 예외구문(raise) 발생

```python
class User :
  def __init__ (self, year_of_birth) :
    if year_of_birth > 2005 :
      raise Exception("Too young")


user1 = User(2000)
user2 = User(2006) # Exception: Too young
```

- 단순히 return 하는 것보다 예외를 발생하는 것이 문제를 명확히 드러나서 유지보수에 유리

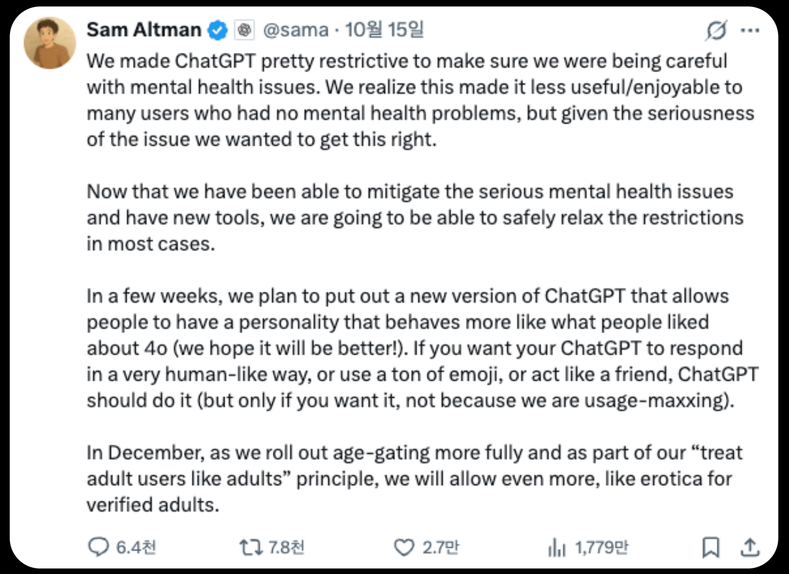
상속이 필요한 이유

여러 클래스가 비슷한 속성과 메소드를 공유해야 할 때

상속이 필요한 이유

서로 다른 클래스 간의 계층 구조가 확실할 때

SNS 게시글의 종류



글만 있는 게시글

링크를 포함한 게시글

사진을 포함한 게시글

동영상을 포함한 게시글

SNS 게시글의 종류

게시글

Sam Altman @sama · 10월 15일

We made ChatGPT pretty restrictive to make sure we were being careful with mental health issues. We realize this made it less useful/enjoyable to many users who had no mental health problems, but given the seriousness of the issue we wanted to get this right.

Now that we have been able to mitigate the serious mental health issues and have new tools, we are going to be able to safely relax the restrictions in most cases.

In a few weeks, we plan to put out a new version of ChatGPT that allows people to have a personality that behaves more like what people liked about 4o (we hope it will be better!). If you want your ChatGPT to respond in a very human-like way, or use a ton of emoji, or act like a friend, ChatGPT should do it (but only if you want it, not because we are usage-maxxing).

In December, as we roll out age-gating more fully and as part of our "treat adult users like adults" principle, we will allow even more, like erotica for verified adults.

6.4천 7.8천 2.7만 1,779만

Bill Gates @BillGates · 2024년 5월 9일

Around the world, unexpected partnerships are driving progress and helping the next generation—like nine-year-old Omphile—reach their fullest potential: gatesnot.es/3UP5f8u

1천 2.4천 60만

Elon Musk @elonmusk · 10월 10일

Grok Imagine prompt:

Black t-shirt with a graphic of a fierce warrior and the text "Off Meta" on the back, laid flat on a textured surface.

The Druid emerges from the shirt, transforming from black and white to full color photorealism and casts a cataclysmic thunderstorm with massive lightning and tornadoes.

Grok 웹, iOS, Android에서 나만의 버전을 만들어 보세요.

5.8천 7.5천 3.3만 1,537만

NVIDIA AI @NVIDIAAI · 10월 3일

Read the science and technology in action: nvda.ws/4nTvSok

1 2 16 3.8천

게시글

```
class Post :
  def __init__ (self, content) :
    self.content = content
```

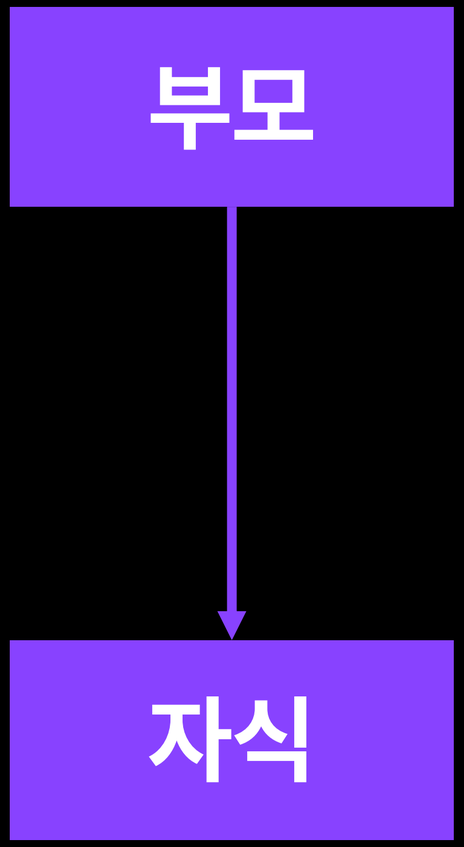
이미지가 있는 게시글

```
class ImagePost :
  def __init__ (self, content, images) :
    self.content = content
    self.images = images
```

이미지가 있는 게시글

```
class ImagePost :
  def __init__ (self, content, images) :
    self.content = content
    self.images = images

  ...

  def num_images(self) :
    return len(self.images)
```

클래스의 상속



부모의 속성, 메소드

부모의 속성, 메소드
+ 자식의 속성, 메소드

청출어람의 법칙

부모 클래스보다 자식 클래스가

더 많은 데이터와 기능을 가지고 있다!

생성자

```python
class Post :
  def __init__ (self, content) :
    self.content = content
```

```python
class ImagePost (Post) :
  def __init__ (self, content, images) :
    super().__init__(content)
    self.images = images
```

super

```python
class Post :
  def __init__ (self, content) :
    self.content = content
```

부모 클래스에
접근할 때

```python
class ImagePost (Post) :
  def __init__ (self, content, images) :
    super().__init__(content)
    self.images = images
```

super

```python
class Post :
  def __init__ (self, content) :
    self.content = content
```

```python
class ImagePost (Post) :
  def __init__ (self, content, images) :
    super().__init__(content)
    self.images = images
```

자식 인스턴스를 생성하면
부모 인스턴스도 생성됨

super 를 안 쓴다면?

```python
class Post :
  def __init__ (self, content) :
    self.content = content
```

```python
class ImagePost(Post):
  def __init__(self, content, images):
    self.content = content
    self.images = images
```

- 부모 클래스의 코드를
  자식 클래스에서 직접 다시 작성해야 함
- 코드 중복이 생기면서
  부모 클래스를 수정하면 자식 클래스도
  일일이 고쳐야 함
  → 유지보수가 어려워짐

상속된 클래스 사용하기

```
class Post :
    def __init__ (self, content) :
        self.content = content

class ImagePost (Post) :
    def __init__ (self, content, images) :
        super().__init__(content)
        self.images = images

image_post = ImagePost(
  "Let's Coding!!!", "home/elice/image.png"
)

print(image_post.content)   # Let's Coding!!!
print(image_post.images)    # home/elice/image.png
```

속성 상속

```
class Post :
  def __init__ (self, content) :
    self.content = content
    self.likers = []
```

속성 상속

- 자식 클래스가 부모 클래스에 정의한 인스턴스 변수(속성)를 물려받아 사용할 수 있는 것

```
class ImagePost(Post) :
  def __init__(self, content, images) :
    super().__init__(content)
    self.images = images

my_post = ImagePost(
  "elice 근처 맛집!!",
  "home/elice/image.png"
)

print(my_post.likers)
```

- ImagePost는 Post를 상속받고 있음
- ImagePost 객체를 생성할 때, 부모 생성자가 호출됨
  - → 부모 클래스에서 정의할 *self.content*, *self.likers* 가 my_post 에 생성됨

속성 상속

```
class Post :
  def __init__ (self, content) :
    self.content = content
    self.likers = []
```

```
class ImagePost(Post) :
  pass

my_post = ImagePost("elice 근처 맛집!!")

print(my_post.likers) # []
```

- ImagePost 안에 __init__ 생성자를 따로 정의하지 않았기 때문에, 자동으로 부모 클래스의 __init__이 호출
- my_post 를 만들 때, content 속성을 생성하고 likers 속성도 빈 리스트로 초기화

속성 상속

```python
class Post :
  def __init__ (self, content) :
    self.content = content
    self.likers = []
```

```python
class ImagePost(Post):
  def __init__(self, content, images):
    self.images = images

my_post = ImagePost(
  "elice 근처 맛집!!",
  "home/elice/image.png"
)

print(my_post.likers)
# AttributeError: 'ImagePost' object has
no attribute 'likers'
```

- 자식 클래스에 __init__을 새로 정의하고
  super()를 호출하지 않으면, 부모 속성은 초기화되지 않음

- 자식 클래스에서 부모 속성을 쓰고 싶다면
  super().__init__()로 부모 생성자를 호출해야 함

메소드 상속

```python
class Post :
  def __init__ (self, content) :
    self.content = content
    self.likers = []

  def like (self, user) :
    self.likers.append(user)
```

메소드 상속

- 자식 클래스가 부모 클래스에서 정의된 메서드를 별도로 다시 작성하지 않고도 사용할 수 있게 하는 기능

```python
class ImagePost(Post):
  def __init__(self, content, images):
    super().__init__(content)
    self.images = images

my_post = ImagePost(
  "elice 근처 맛집!!",
  "home/elice/image.png"
)

my_post.like("Bob")

print(my_post.likers) # ['Bob']
```

- ImagePost는 Post를 상속받고 있음
- ImagePost에는 like() 메소드가 정의되어 있지 않음
- Post에서 정의된 like() 메소드를 자동으로 물려받음

- 자식 클래스는 부모의 기능을 그대로 사용함으로써 코드 중복을 줄이고 유지보수를 쉽게 만듦

좋아요, 슬퍼요

```python
class Like :
  def __init__ (self, post, user) :
    self.post = post
    self.user = user
```

```python
class Sad :
  def __init__ (self, post, user) :
    self.post = post
    self.user = user
```

추상적인 부모 클래스

```
class Reaction :
  def __init__ (self, _type, post, user) :
    self.type = _type
    self.post = post
    self.user = user
```

추상적인 부모 클래스

```
class Like (Reaction) :
  def __init__ (self, post, user) :
    super().__init__("LIKE", post, user)
```

```
class Sad (Reaction) :
  def __init__ (self, post, user) :
    super().__init__("SAD", post, user)
```

추상적인 부모 클래스

여러 자식 클래스가
공통적으로 가져야 할 속성이나 행동의 틀을 정의하지만

스스로는
구체적인 기능을 가지지 않거나 제한된 기능만 가지는 클래스

추상적인 부모 클래스

추상적인 부모 클래스

```
class Reaction :
  def __init__ (self, _type, post, user) :
    self.type = _type
    self.post = post
    self.user = user
```

```
class Like (Reaction) :
  def __init__ (self, post, user) :
    super().__init__("LIKE", post, user)
```

자식 클래스 1

```
class Sad (Reaction) :
  def __init__ (self, post, user) :
    super().__init__("SAD", post, user)
```

자식 클래스 2

추상적인 부모 클래스 사용 이유

```
class Reaction :
  def __init__ (self, _type, post, user) :
    self.type = _type
    self.post = post
    self.user = user
```

코드 중복 제거

자식 클래스마다 post, user 속성을
따로 작성하지 않아도 됨

구조 통일

모든 자식 클래스는
type, post, user을 가짐

확장 용이

새로운 반응(e.g. Angry 등)을
추가할 때 부모 로직 재사용 가능

유지보수 편리

공통 로직을 한 곳 (부모) 에만
수정하면 됨

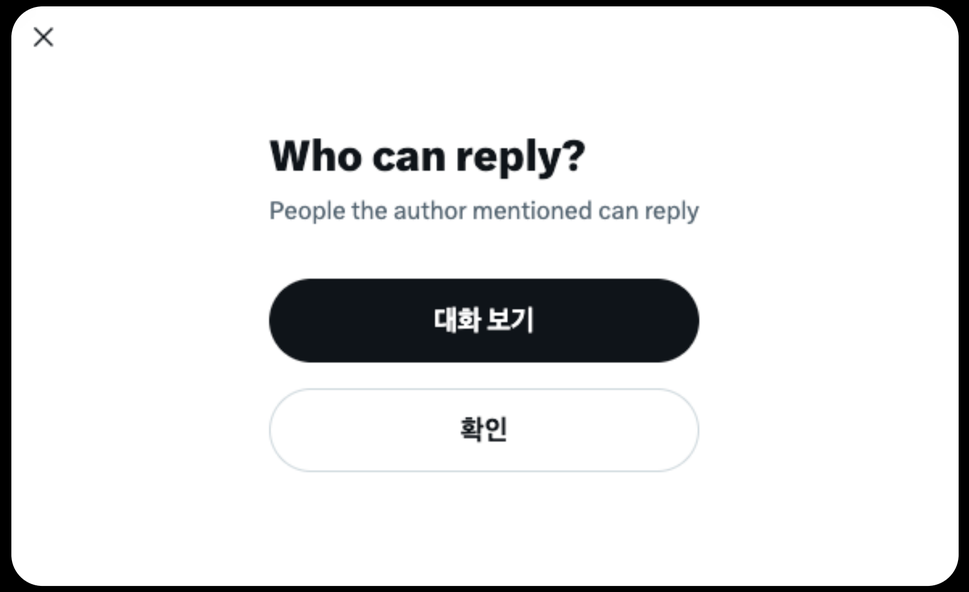
추상적인 부모 클래스

```
# 이렇게 쓰진 않는다!
reaction = Reaction("vsadjkl", post, me)

# 반드시 구체적인 자식 클래스로 쓴다!
like = Like(post, me)
```

추상적인 부모 클래스는
직접 인스턴스화하지 않음

클래스 설계 상황

모든 게시글에는 댓글을 달 수 있지만
보호 설정을 한 게시물에는 댓글을 달 수 없다



오버라이딩

```python
class Post:
  def __init__ (self, content) :
    self.content = content
    self.comments = {}

  def comment (self, user, comment):
    self.comments[user] = comment


normal_post = Post("오공완(오늘도 공부 완료)!!!")

normal_post.comment("Sam", "너무 멋지다!!")
normal_post.comment("Queen", "나도 질 수 없지")

print(normal_post.comments)
# {'Sam': '너무 멋지다!!', 'Queen': '나도 질 수 없지'}
```

오버라이딩

```python
class Post:
  def __init__ (self, content) :
    self.content = content
    self.comments = {}

  def comment (self, user, comment):
    self.comments[user] = comment


normal_post = Post("오공완(오늘도 공부 완료)!!!")

normal_post.comment("Sam", "너무 멋지다!!")
normal_post.comment("Queen", "나도 질 수 없지")

print(normal_post.comments)
# {'Sam': '너무 멋지다!!', 'Queen': '나도 질 수 없지'}
```

```python
class ProtectedPost (Post) :
  def comment (self, user, content) :
    print("Who can reply? People the author mentioned can reply")

protected_post = ProtectedPost("보호된 게시물")

protected_post.comment("jojo", "OH!! NO!!")
# "Who can reply? People the author mentioned can reply"
```

- Post 클래스에는 원래 댓글을 저장하는 comment() 메서드가 존재
- ProtectedPost는 comment()를 오버라이딩(덮어쓰기)해서 댓글을 저장하지 않고 안내 메세지를 출력하도록 동작을 변경
- 같은 메서드 이름이지만 객체의 종류에 따라 결과가 달라짐

모듈이란?

다른 코드에서 사용할 수 있도록 열어 놓은 코드

모듈 불러오기

```
import string

print(string.digits) #0123456789
```

모듈 불러오기

```
from string import digits

print(string) #0123456789
```

모듈이란?

다른 코드에서 사용할 수 있도록 열어 놓은 코드

모든 코드는 모듈이다


posts.py
class Post :
...
class ImagePost (Post) :
...


# timeline.py
from posts import Post, ImagePost
timeline.add(Post("Hello!"))
timeline.add(IamgePost("#food", []))

패키지란?

모듈을 모아 놓은 폴더

SNS

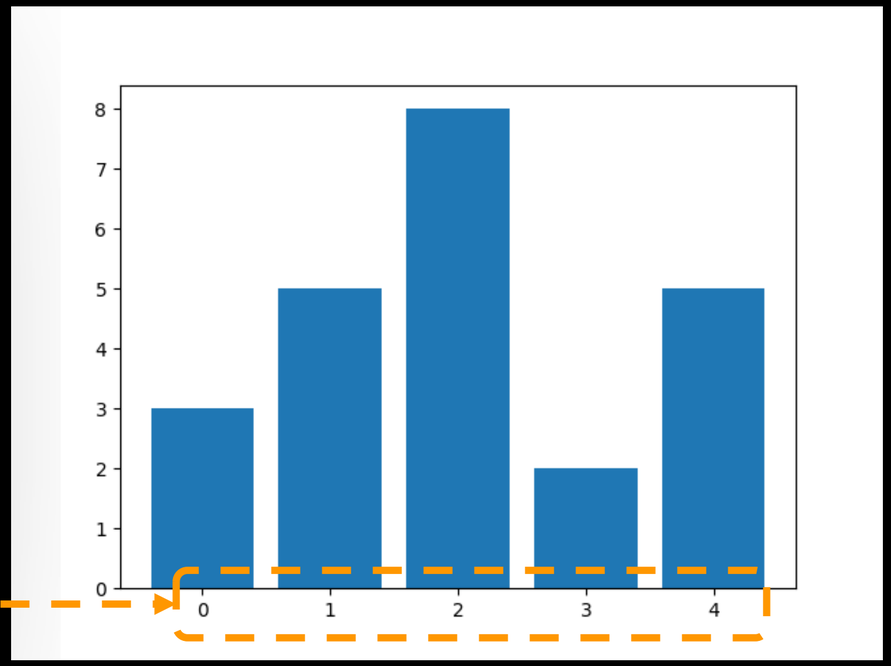
- posts.py
- users.py
- media.py
- ads.py

패키지 안의 모듈

```
import SNS.users
from SNS import media
from SNS.ports import Post, ImagePost

...
```

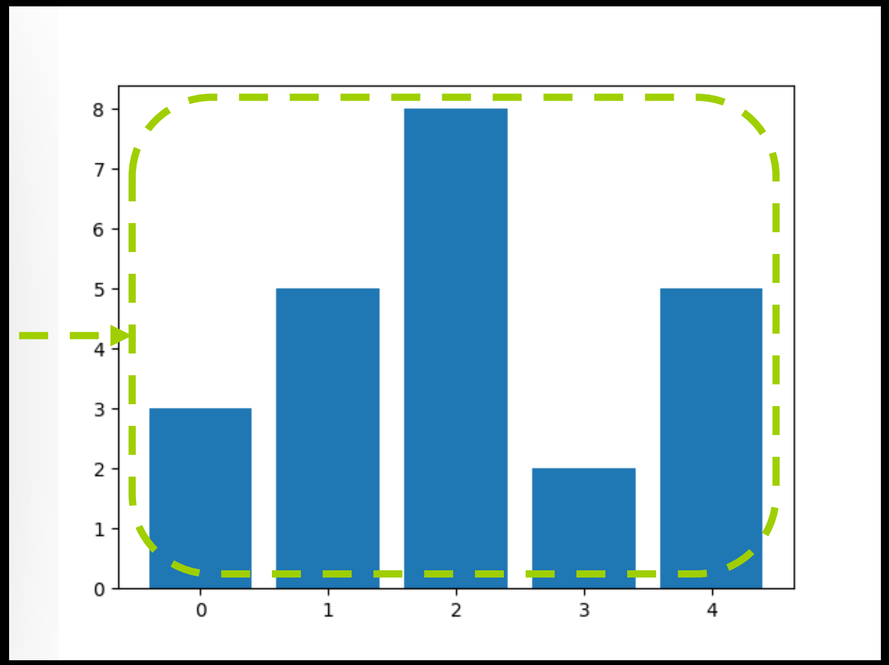
matplotlib

```python
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4]
y = [3, 5, 8, 2, 5]

plt.bar(x, y, align="center")
plt.show()
```



matplotlib

```python
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4]
y = [3, 5, 8, 2, 5]

plt.bar(x, y, align="center")
plt.show()
```

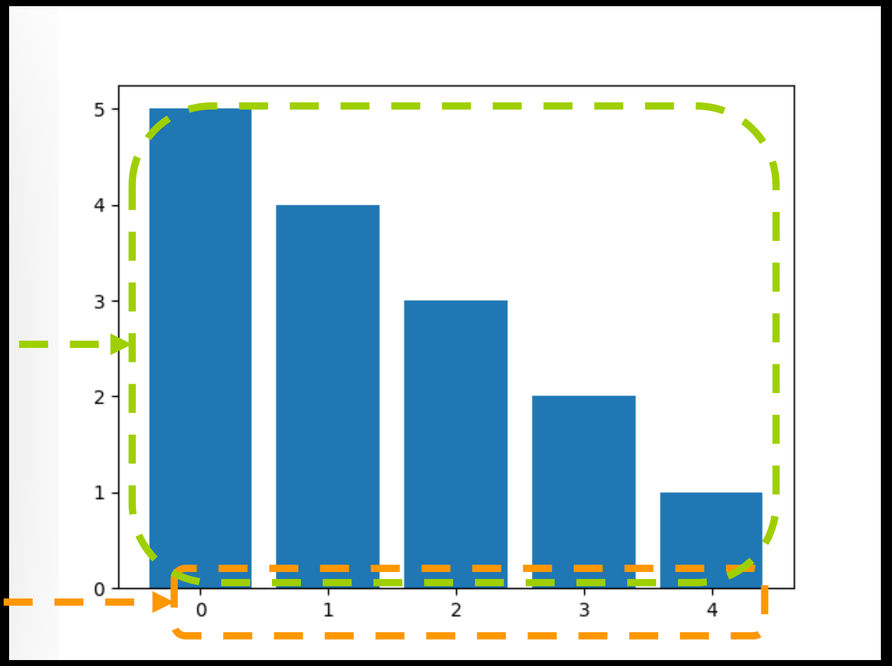


bar

```
import matplotlib.pyplot as plt

plt.bar(위치, 높이)
plt.show( )
```

bar

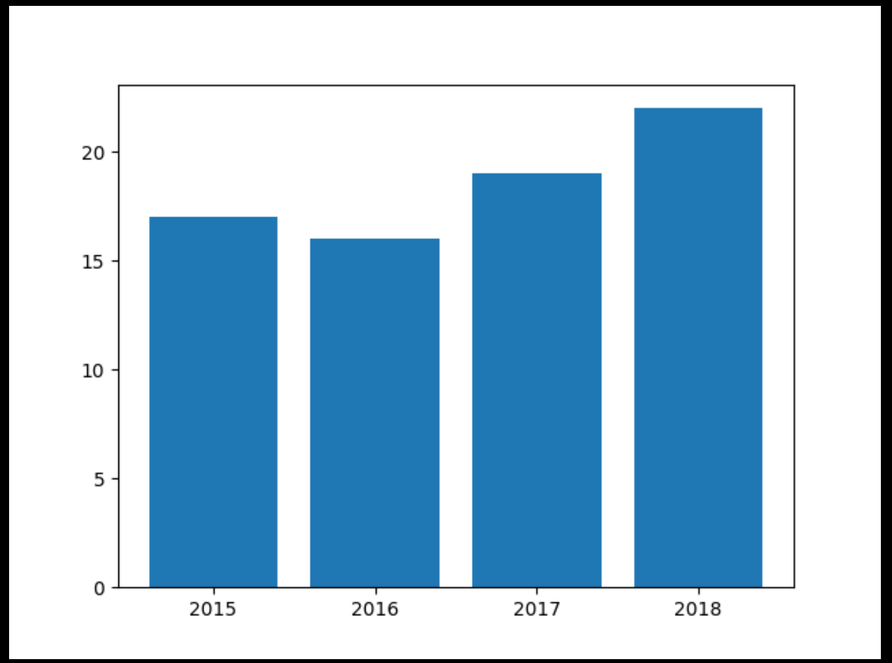
```python
import matplotlib.pyplot as plt

x = range(5)
y = [5, 4, 3, 2, 1]

plt.bar(x, y)
plt.show()
```



xticks()

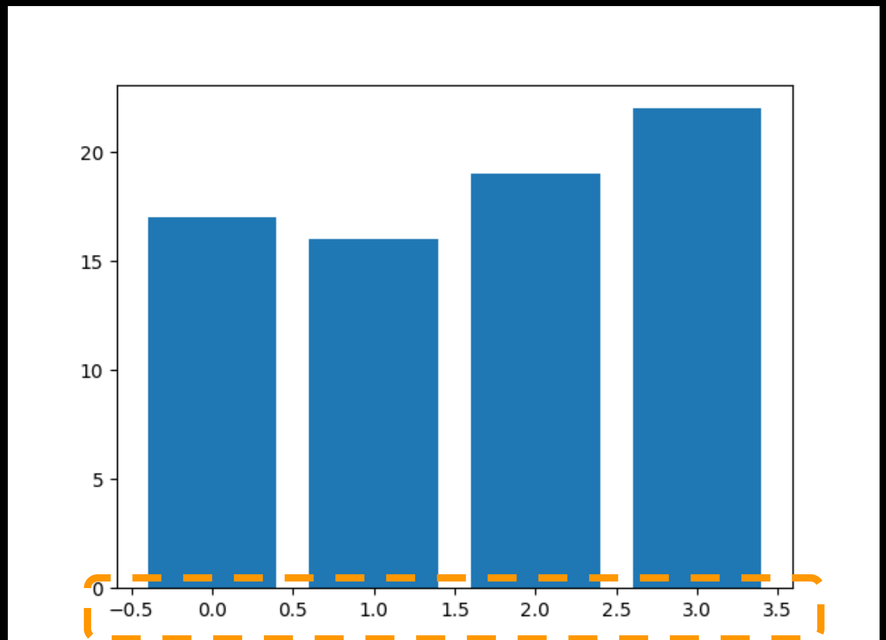
```python
import matplotlib.pyplot as plt

pos = range(4)
years = [2015, 2016, 2017, 2018]
temp = [17, 16, 19, 22]

plt.bar(pos, temp)
plt.xticks(pos, years)
```

xticks() → x축 눈금 위치와 라벨(표시되는 값)을 설정하는 함수



xticks()

```python
import matplotlib.pyplot as plt

pos = range(4)
years = [2015, 2016, 2017, 2018]
temp = [17, 16, 19, 22]

plt.bar(pos, temp)
```




```python
import matplotlib.pyplot as plt

pos = range(4)
years = [2015, 2016, 2017, 2018]
temp = [17, 16, 19, 22]

plt.bar(pos, temp)
plt.xticks(pos, years)
```