

파이썬, 어떻게 코딩할 것인가?

10기 장유영

Executive Summary

- ✓ Class는 객체(Object)를 생성하는 “틀”의 역할을 수행함
- ✓ 이전에 만든 Class의 성질을 사용하기 위해 상속, Override, Super 등의 개념이 도입됨
- ✓ 깔끔하고 가독성이 좋은 코딩을 위해서는 Class 사용이 필수적임
- ✓ Method Type에는 Instance Method, Class Method, Static Method가 있음

CONTENTS

객체(Object) 지향 프로그래밍

Class 선언하기

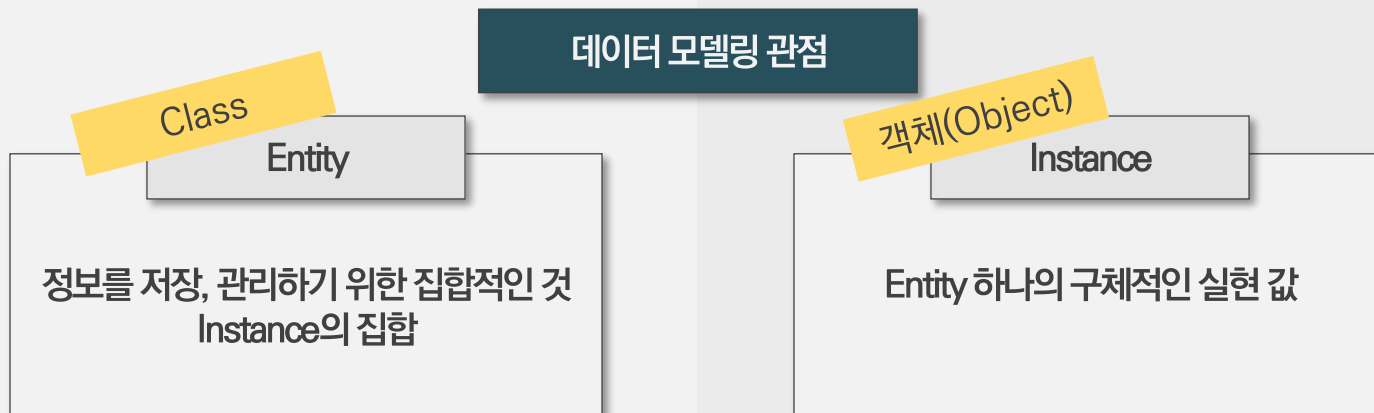
상속과 Override

Method Type

맷음말

데이터 모델링 관점에서 Entity는 Instance의 집합이며, Instance는 Entity의 구체적인 실현 값임

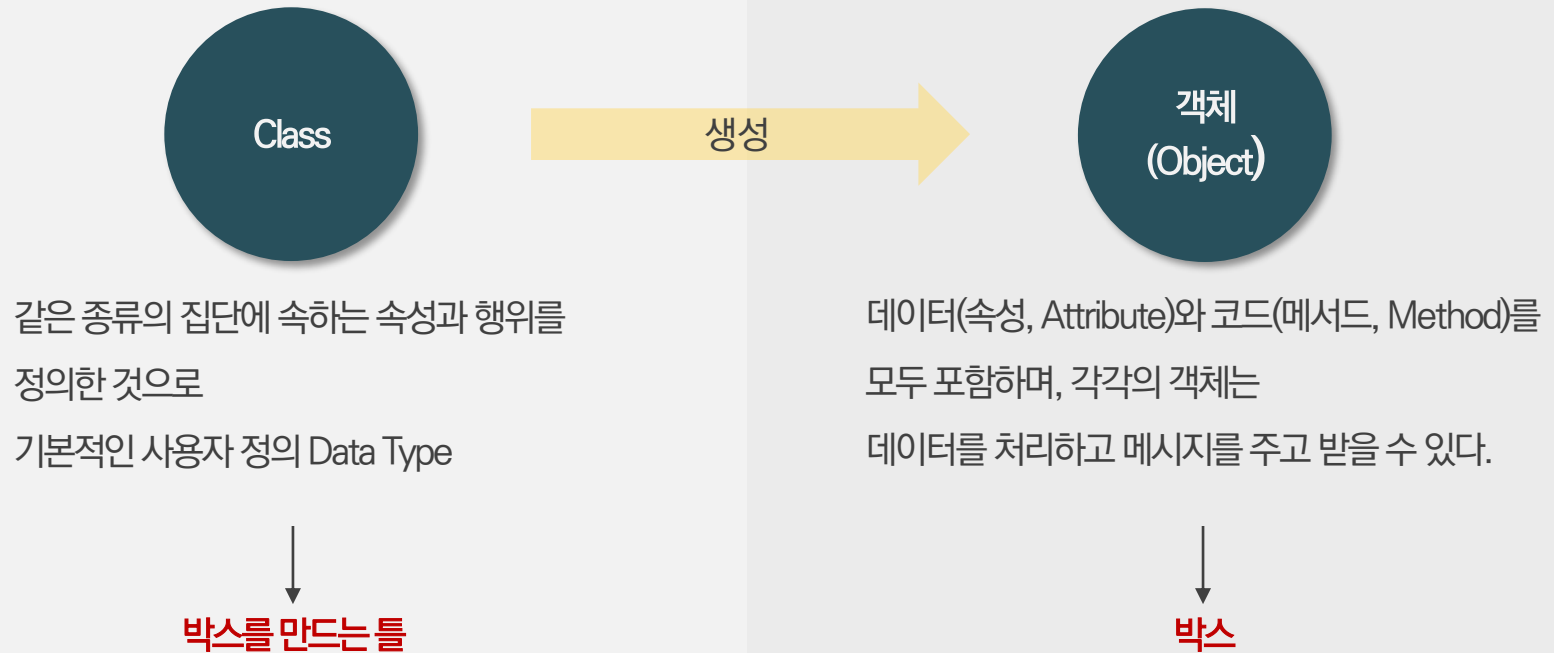
Ch1. 객체 지향 프로그래밍



Entity	Instance
과목	수학 영어
사람	장유영 귀도 반 로섬

Class는 객체를 생성하는 “틀”의 역할을 수행하며, Class의 Instance (메모리에 할당된 값)인 객체는 Method를 통해 서로 통신함

Ch1. 객체 지향 프로그래밍



```

31 def __init__(self, path):
32     self.file = None
33     self.fingerprints = set()
34     self.logdups = True
35     self.debug = debug
36     self.logger = logging.getLogger(__name__)
37     if path:
38         self.file = open(os.path.join(path, 'requests.txt'), 'w')
39         self.file.seek(0)
40         self.fingerprints.update(self.retrieve())
41
42 @classmethod
43 def from_settings(cls, settings):
44     debug = settings.getbool('SUPERSLIM_DEBUG')
45     return cls(job_dir(settings), debug)
46
47 def request_seen(self, request):
48     fp = self.request_fingerprint(request)
49     if fp in self.fingerprints:
50         return True
51     self.fingerprints.add(fp)
52     if self.file:
53         self.file.write(fp + os.linesep)
54
55 def request_fingerprint(self, request):
56     return request_fingerprint(request)

```

Class 선언하기

간단한 Class 선언 시연

생성자, 초기화(Initialization) 메서드

객체 자기 자신

```
class Cocktail:
    def __init__(self):
        self.whiskey_price = 7000
        self.jin_price = 5000
```

다른 Method에서도 이용 가능하게 함

```
def make_cocktail(self, num_of_glasses, base):
    whiskey_price = self.whiskey_price
    jin_price = self.jin_price

    if base == "whiskey":
        print("Here is your Jackcoke, {} glasses, right? It's {} won".
              format(num_of_glasses, num_of_glasses*whiskey_price))
    if base == "Jin":
        print("Here is your Jin Tonic, {} glasses, right? It's {} won".
              format(num_of_glasses, num_of_glasses*jin_price))
```

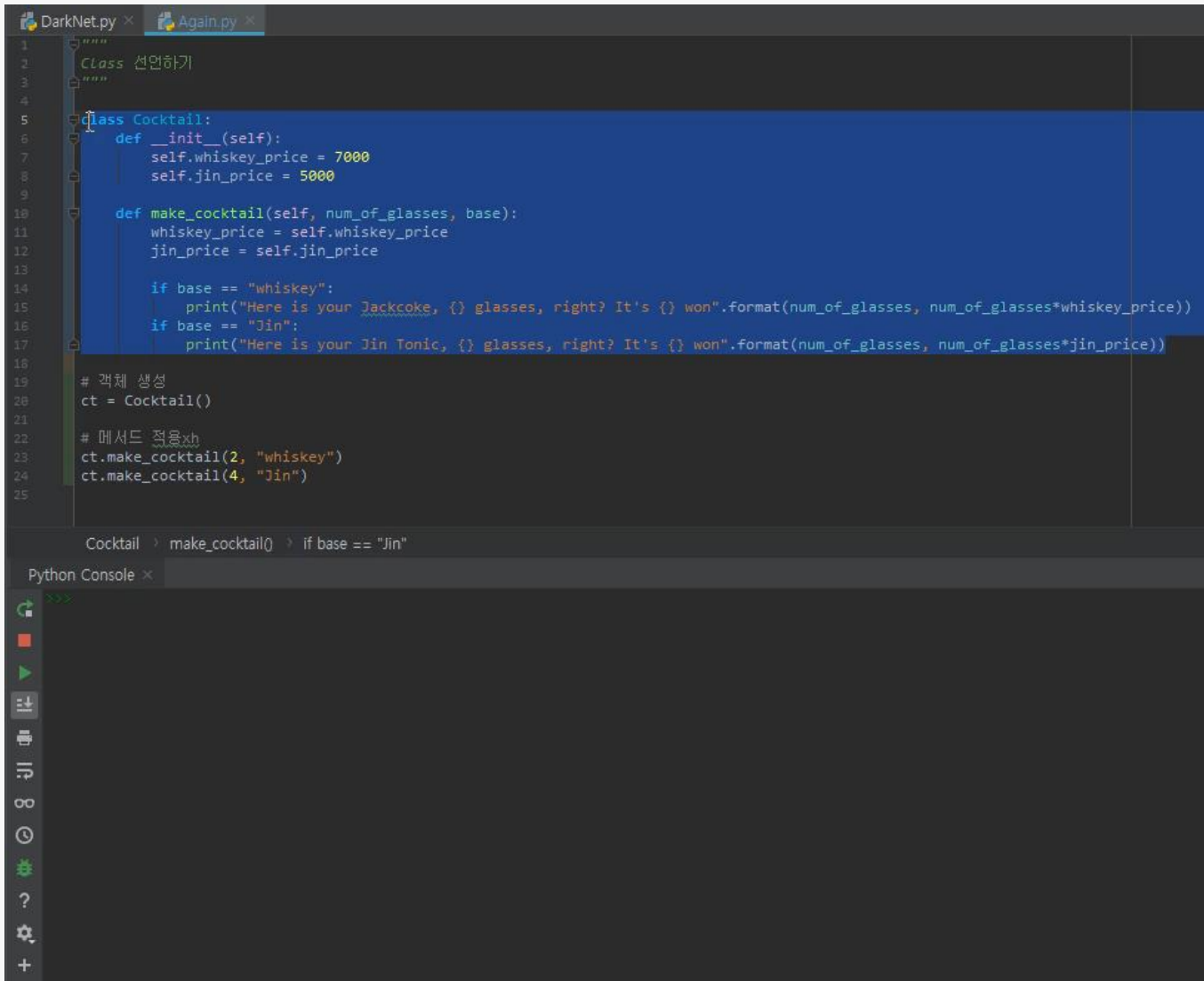
whiskey_price라는 속성의 값은 7000, self인자를 통해 저장

매개변수

1번째 매개변수는 self 객체임을 나타내야 하는 것

저장해야 하는 값이
이렇게 이용 가능

간단한 Class 선언 시연



The screenshot shows a Python IDE with two tabs: 'DarkNet.py' and 'Again.py'. The 'Again.py' tab is active and contains the following code:

```
1  """
2  Class 선언하기
3  """
4
5  class Cocktail:
6      def __init__(self):
7          self.whiskey_price = 7000
8          self.jin_price = 5000
9
10     def make_cocktail(self, num_of_glasses, base):
11         whiskey_price = self.whiskey_price
12         jin_price = self.jin_price
13
14         if base == "whiskey":
15             print("Here is your Jackcoke, {} glasses, right? It's {} won".format(num_of_glasses, num_of_glasses*whiskey_price))
16         if base == "Jin":
17             print("Here is your Jin Tonic, {} glasses, right? It's {} won".format(num_of_glasses, num_of_glasses*jin_price))
18
19     # 객체 생성
20     ct = Cocktail()
21
22     # 메서드 적용
23     ct.make_cocktail(2, "whiskey")
24     ct.make_cocktail(4, "Jin")
25
```

Below the code editor, the 'Python Console' tab is open, showing the execution path: 'Cocktail > make_cocktail() > if base == "Jin"'. The console is currently empty, with a green prompt '>>>' visible.

간단한 Class 선언 시연

매개변수의 수가 차이는 이유는?

```
# def make_cocktail(self, num_of_glasses, base):  
  
ct = Cocktail() 강연!  
  
ct.make_cocktail(2, "whiskey")  
>>> Here is your Jackcoke, 2 glasses, right? It's 14000 won  
  
ct.make_cocktail(4, "Jin")  
>>> Here is your Jin Tonic, 4 glasses, right? It's 20000 won
```

- ✓ Python Method의 첫 매개변수명은 관례적으로 자기 자신을 의미하는 self를 사용함
- ✓ 호출 시 호출한 객체 자신이 전달됨
- ✓ 첫 번째 매개변수 self를 명시적으로 구현하는 것은 Python의 독특한 특징임

간단한 Class 선언 시연

객체 생성

```
ct = Cocktail()  
print(ct.whiskey_price)  
7000  
print(ct.jin_price)  
5000
```

객체의 속성(Attribute)

Cocktail 클래스의 정의를 찾음

새 객체를 메모리에 초기화(생성)함

객체의 `__init__` Method를 호출함

새롭게 생성된 객체를 `self`에 전달함

새로운 객체를 반환함

반환된 객체를 `ct`에 연결함

Class 사용 예시: 딥러닝 네트워크 구현

괄호 등장?

```
class DarkNet(nn.Module):
    def __init__(self, config):
        super(DarkNet, self).__init__()
```

— : 예약된 Method

— : 클래스 네임공간으로 사용하기 위한 것

```
self._config = config
self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
self.conv1 = self._block_1x(in_channels=3, out_channels=32, kernel_size=3)
...
self.conv23 = nn.Conv2d(in_channels=1024, out_channels=config.NUM_BOX * (1 + 4 + config.NUM_CLASS),
                        kernel_size=1, stride=1, padding=1, bias=False)
```

(mnist) (항상 요구사항 - 사항)

decorator

@...??

언더바 하나?

```
@classmethod
def _block_1x(cls, in_channels, out_channels, kernel_size):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=1, bias=False),
        nn.BatchNorm2d(out_channels),
        nn.LeakyReLU(negative_slope=0.1)
    )
```

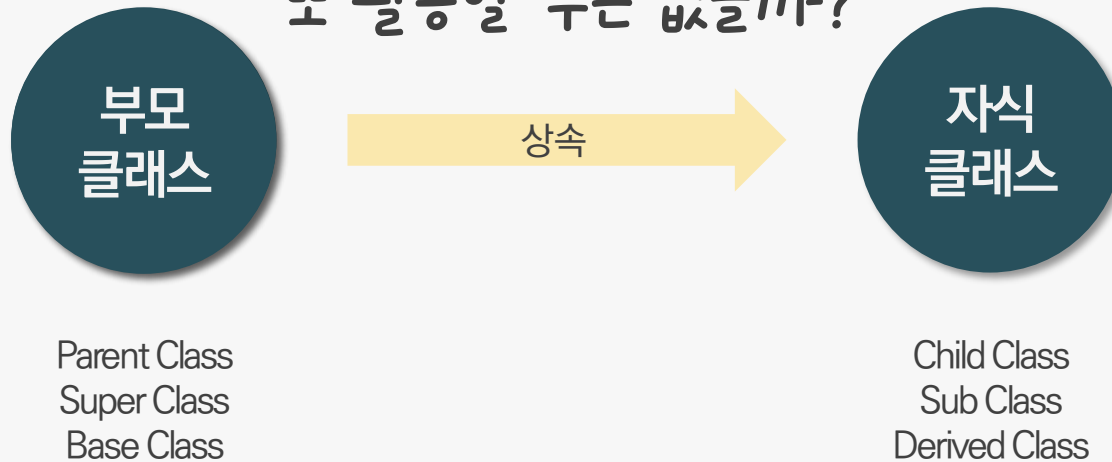
```
def forward(self, x):
    # Layer 1 ~ 2
    out = self.max_pool2d(self.conv1(x))
    out = self.max_pool2d(self.conv2(out))
    ...
    # Layer 23
    out = self.conv23(out)
    out = out.reshape(shape=(self._config.GRID_H,
                             self._config.GRID_W,
                             self._config.NUM_BOX,
                             1 + 4 + self._config.NUM_CLASS))
    return out
```

darkNet = DarkNet(config)

← 객체 생성

상속 - Inheritance

방금 만든 Class를
또 활용할 수는 없을까?



- ✓ 기존 Class에서 일부를 추가하거나 변경하여 새로운 Class를 생성함
- ✓ 코드를 재사용(reuse)하는 방법임

Method Override

자식 Class 선언

Cocktail 클래스 상속

Override



부모 클래스 method
중복해서 이용하게!

```
class MyCocktail(Cocktail):
    def make_cocktail(self, num_of_glasses, base):
        whiskey_price = self.whiskey_price
        jin_price = self.jin_price
        vodka_price = 6000

        if base == "vodka":
            print("This is Appletini, {} glasses, right? It's {} won".
                  format(num_of_glasses, num_of_glasses*vodka_price))
        else:
            print("I only drink Vodka")
```

```
mct = MyCocktail()
mct.make_cocktail(2, "vodka")
```

```
>>> This is Appletini, 2 glasses, right? It's 12000 won
```

Super: 자식 Class에서 부모 Class의 Method 호출

두 코드의 차이는?

```
class MyCocktail(Cocktail):
    def __init__(self):
        self.alcoholicity = "30 proof"

    def ask_strength(self):
        alcoholicity = self.alcoholicity
        print("This is {}".format(alcoholicity))
```

```
mct = MyCocktail()
mct.ask_strength()
mct.jin_price
```

```
This is 30 proof
Traceback (most recent call last):
  File "<input>", line 3, in <module>
AttributeError: 'MyCocktail' object has no attribute 'jin_price'
```

여야 부모클래스까지
각각한 속성이
있어야!

```
class MyCocktail(Cocktail):
    def __init__(self):
        super().__init__()
        self.alcoholicity = "30 proof"

    def ask_strength(self):
        alcoholicity = self.alcoholicity
        print("This is {}".format(alcoholicity))
```

```
This is 30 proof
5000
```

- ✓ 자식 Class에서 `__init__` Method를 새로 정의하면 부모 Class의 `__init__` Method를 대체함
- ✓ 따라서 부모 Class의 Method를 호출하기 위해서는 명시적으로 호출해야 함

Method Type은 아래와 같이 3가지로 분류할 수 있음

Ch4. Method Type

```
class CrazyManiac():
    count = 0

    def __init__(self):
        CrazyManiac.count += 1

    def ask_nicely(self):
        print("Do you even speak?")

    @classmethod
    def count_numbers(cls):
        print("There are {} CrazyManiacs".
              format(cls.count))

    @staticmethod
    def ask_furiously():
        print("DO YOU THINK I'm CRAZY?")

Trump, XiJinping, Putin =
    CrazyManiac(), CrazyManiac(), CrazyManiac()

CrazyManiac.count_numbers()
CrazyManiac.ask_furiously()

>>> There are 3 CrazyManiacs
>>> DO YOU THINK I'm CRAZY?
```

Instance Method

- ✓ 첫 번째 인자가 self임
- ✓ 일반적인 Class를 생성할 때의 Method Type

Class Method

- ✓ Class 전체에 영향을 미침
 - ✓ 이 Method의 첫 번째 매개변수(cls)는 Class 자신임
- class는 클래스의 예약어라 cls.*

Static Method

- ✓ Class나 객체에 영향을 미치지 못함
- ✓ 단지 편의를 위해 존재함
- ✓ 이 Method에 접근하기 위해서 객체를 생성할 필요가 없음

코딩 매너는 지키도록 합시다.

Ch5. 맷음말

- ✓ 조금 복잡한 Method는 쌍 따옴표 3개를 이용하여 부연 설명을 반드시 써준다.
- ✓ 한 줄에 과하게 길게 입력하지 않는다. (Maximum Line Length 지키기)
- ✓ 연산자의 앞에서 줄 바꿈을 한다.
- ✓ Import는 행을 분리해서 선언한다.
- ✓ Import는 다음 순서로 Grouping 한다.
standard library imports, related third party imports, local application/library specific imports
- ✓ Import할 패키지가 많을 경우 Configuration 파일을 따로 만든다.
- ✓ 대입 연산자(=)의 앞 뒤에는 공백을 주도록 한다.
- ✓ 키워드 인자에 사용되는 대입 연산자에는 공백을 사용하지 않는다.
- ✓ 내부 용 함수는 Under Bar(_)를 하나 사용해준다.
- ✓ Class Name은 CapWords 방식을 따른다.
- ✓ Method Name은 소문자를 사용하며 Under Bar로 단어를 구분한다.
- ✓ 상수는 대문자로 표시한다. (MAX_LENGTH)

@Decorator의 정의와 사용법 알아보기

- ✓ 알아서 하세요 😊
- ✓ 과제게시판에 업로드 안 해도 됩니다.

End of Document

Have a Good Day