Basic Python

Lecture 10. 객체와 클래스

BAE SEKANG(배세강)

Contents

Object

♣ 객체

- 객체(Object): 데이터와 기능을 함께 가진 하나의 단위
- Python의 모든 값은 객체이다. ➔ id()
- 객체는 어떤 class에 속해있다. ➔ type()

```
'''
1. hello라는 문자 데이터
2. .lower(), .upper(), .find() 등의 기능
'''


s = "hello"
```

Class

♣ 클래스

- 새로운 객체를 만들기 위해 속성(attribute)과 메소드(method)를 묶어 정의한 사용자 정의 자료형
- 같은 종류의 객체들이 공통적으로 가지는 속성과 동작을 정의

```
class Student:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def print_info(self):
        print(f"{self.name}의 점수는 {self.score}점입니다.")
```

Class

♣ 클래스

```
class Student:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def print_info(self):
        print(f"{self.name}의 점수는 {self.score}점입니다.")

s1 = Student("Alice", 90)
s2 = Student("Bob", 75)
s1.print_info()
s2.print_info()
```

Class

♣ 클래스

- class를 이용해 새로운 자료형을 직접 만들 수 있음

```
a = 10
b = "hello"
c = [1, 2, 3]

print(type(a))  # <class 'int'>
print(type(b))  # <class 'str'>
print(type(c))  # <class 'list'>
```

Class

♣ Instance

- 인스턴스: class를 바탕으로 실제 만들어진 객체

```
class Student:
    def __init__(self, name, score):
        self.name = name
        self.score = score

s1 = Student("Alice", 90)
s2 = Student("Bob", 75)
```

Class

♣ Attribute

- 속성: 객체가 가지고 있는 데이터

```
class Student:
    def __init__(self, name, score):
        self.name = name
        self.score = score

s1 = Student("Alice", 90) # Alice가 name, 90이 score로 전달됨
s2 = Student("Bob", 75) # Bob이 name, 75가 score로 전달됨
print(s1.name)
print(s2.score)
```

Class

♣ Method

- 메소드: 객체가 수행할 수 있는 기능

```
class Student:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def print_info(self):
            print(f"{self.name}의 점수는 {self.score}점입니다.")

s1 = Student("Alice", 90) # Alice가 name, 90이 score로 전달됨
s1.print_info()
```

Class

♣ __init__()

- 객체가 생성될 때 자동으로 실행되는 특별한 메소드
- 객체가 처음 만들어질 때 필요한 값을 초기화하는 역할
- ※ 참고: 내부적으로 실제 객체 생성은 __new__()가 담당함

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

dog1 = Dog("초코", 3)
dog2 = Dog("보리", 5)

print(dog1.name, dog1.age)
print(dog2.name, dog2.age)
```

Class

♣ self

- 현재 메소드를 호출한 객체 자신을 의미하는 변수(관례적으로 self로 사용)
- dog1.bark()를 호출하면 self가 dog1, dog2.bark()를 호출하면 self가 dog2

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

    def bark(self):
        print(f"{self.name}: 멍멍!")

dog1 = Dog("초코")
dog2 = Dog("보리")

dog1.bark()
dog2.bark()
```

Class

♣ Instance Variable

- 각 객체마다 따로 가지는 변수
- s1.score와 s2.score는 서로 다른 값이다.

```
class Student:
    def __init__(self, name, score):
        self.name = name
        self.score = score

s1 = Student("Alice", 90)
s2 = Student("Bob", 75)

s1.score = 95

print(s1.score)
print(s2.score)
```

Class

[Example]: Rectangle 클래스

1. 가로 width, 세로 height를 인스턴스 변수로 가진다.
2. area() 메소드는 넓이를 **반환**한다.
3. perimeter() 메소드는 둘레를 **출력**한다.

Class

[Example]: Rectangle 클래스

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        print(2 * (self.width + self.height))

r1 = Rectangle(3, 5)
print(r1.area())
r1.perimeter()
```

Class

♣ Class Variable

- 같은 class로 만들어진 객체들이 공유하는 변수

```
class Student:
    school = "Python High School"

    def __init__(self, name, score):
        self.name = name
        self.score = score

s1 = Student("Alice", 90)
s2 = Student("Bob", 75)

print(s1.school)
print(s2.school)
print(Student.school)
```

Class

♣ Class Variable

- 클래스 변수는 공유된다.

```
class Team:
    members = []                                # mutable한 객체는 공유 시 변경될 수 있다는 점을 유의한다!

    def __init__(self, name):
        self.name = name

    def add_member(self, member):
        self.members.append(member)

team1, team2 = Team("A"), Team("B")
team1.add_member("Alice")
team2.add_member("Bob")

print(team1.members)      # ['Alice', 'Bob']
print(team2.members)      # ['Alice', 'Bob']
```

Class

♣ Class Variable

- team별로 멤버를 두고 싶다면 아래와 같이 수정하여 instance variable로 둔다.

```
class Team:
    def __init__(self, name):
        self.name = name
        self.members = []

    def add_member(self, member):
        self.members.append(member)

team1, team2 = Team("A"), Team("B")
team1.add_member("Alice")
team2.add_member("Bob")

print(team1.members)      # ['Alice']
print(team2.members)      # ['Bob']
```

Special Method

♣ __str__()

- 객체를 출력했을 때 원하는 출력 형태로 만드는 특별한 메소드

```
# 기존

class Student:
    def __init__(self, name, score):
        self.name = name
        self.score = score

s = Student("Alice", 90)

print(s)     # <__main__.Student object at 0x...>
```

```
class Student:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def __str__(self):
        return f"{self.name}: {self.score}점"

s = Student("Alice", 90)

print(s)     # Alice: 90점
```

Special Method

♣ __repr__()

- 개발자가 객체 정보를 확인하기 좋은 문자열로 출력하도록 설정

```
# 기존

class Student:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def __str__(self):
        return f"{self.name}: {self.score}점"

s = Student("Alice", 90)

print(s)     # Alice: 90점
print([s])  # [<__main__.Student object at 0x...>]
```

```
class Student:
    def __init__(self, name, score):
        self.name = name
        self.score = score
    def __str__(self):
        return f"{self.name}: {self.score}점“

    def __repr__(self):
        return f"Student(name={self.name},
score={self.score})"

s = Student("Alice", 90)
print(s)    # Alice: 90점
print([s]) # [Student(name='Alice', score=90)]
```

Special Method

♣ __repr__()

- __str__과 __repr__가 둘 다 있다면 __str__를 우선 사용
- 객체가 리스트 안에 들어 있으면 __repr__이 사용된다.
  단, __str__()이 정의되어 있지 않으면 print(객체)에서도 __repr__()이 사용된다.

```
class Student:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def __repr__(self):
        return f"Student(name='{self.name}', score={self.score})"


s = Student("Alice", 90)

print(s)    # Student(name='Alice', score=90)
```

Inheritance

♣ 상속

- 서로 다른 class이지만, name과 eat()이라는 공통점이 존재

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

    def eat(self):
        print(f"{self.name}이/가 먹습니다.")
    def bark(self):
        print(f"{self.name}: 멍멍!")

class Cat:
    def __init__(self, name):
        self.name = name

    def eat(self):
        print(f"{self.name}이/가 먹습니다.")
    def meow(self):
        print(f"{self.name}: 야옹!")
```

Inheritance

♣ 상속

- 기존 class의 속성과 메소드를 물려 받아 새로운 class를 만드는 기능

```
# 상속 문법

class Parent:
    pass

class Child(Parent):
    pass

# 예시
class Animal:
    pass
class Dog(Animal):
    pass
```

Inheritance

♣ 상속 – Example1

- 부모 class: Animal, 자식 class: Dog

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

    def eat(self):
        print(f"{self.name}이/가 먹습니다.")

class Dog(Animal):
    def bark(self):
        print(f"{self.name}: 멍멍!")

dog = Dog("초코")
dog.eat()     # eat은 Animal의 메소드이지만 Dog도 물려 받은 메소드
dog.bark()
```

Inheritance

♣ 상속 – Example1

- ※ Dog("초코")를 실행하면 Animal의 __init__()에 의해 self.name이 생성된다.

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

    def eat(self):
        print(f"{self.name}이/가 먹습니다.")

class Dog(Animal):
    def bark(self):
        print(f"{self.name}: 멍멍!")

dog = Dog("초코")
dog.eat()     # eat은 Animal의 메소드이지만 Dog도 물려 받은 메소드
dog.bark()
```

Inheritance

♣ 상속 – Example2

- 여러 개의 자식 class를 생성

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

    def eat(self):
        print(f"{self.name}이/가 먹습니다.")


class Dog(Animal):
    def bark(self):
        print(f"{self.name}: 멍멍!")
```

```
class Cat(Animal):
    def meow(self):
        print(f"{self.name}: 야옹!")


dog = Dog("초코")
cat = Cat("나비")

dog.eat()
cat.eat()

dog.bark()
cat.meow()
```

Inheritance

♣ Method Overriding

- 부모 class에 정의된 메소드를 자식 class에서 같은 이름으로 재정의하는 것
- 같은 이름의 메소드가 있다면 자식 class의 메소드가 우선 실행된다.

```
class Animal:
    def speak(self):
        print("동물이 소리를 냅니다.")

class Dog(Animal):
    def speak(self):
        print("멍멍!")

class Cat(Animal):
    def speak(self):
        print("야옹!")
```

```
animal = Animal()
dog = Dog()
cat = Cat()

animal.speak() # 동물이 소리를 냅니다.
dog.speak()    # 멍멍!
cat.speak()    # 야옹!
```

Inheritance

♣ super()

- 자식 class에서 부모 class의 메소드를 호출할 때 활용
- 특히 부모 class의 __init__()을 실행할 때 자주 사용

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age


class Student(Person):
    def __init__(self, name, age, score):
        super().__init__(name, age)
        self.score = score
```

```
s = Student("Alice", 17, 90)

print(s.name)  # Alice
print(s.age)     # 17
print(s.score)  # 90
```

Inheritance

♣ super()

- super를 사용하지 않는다면? ➔ self.name이 만들어지지 않았기 때문에 s.name에 접근하면 오류 발생

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

class Student(Person):
    def __init__(self, name, score):
        self.score = score

s = Student("Alice", 90)

print(s.score) # 90
print(s.name) # AttributeError
```

```
# Student를 다음과 같이 수정

class Student(Person):
    def __init__(self, name, score):
        super().__init__(name)
        self.score = score
```

Inheritance

♣ is-a relationship

- A is a B: A는 B의 한 종류이다 ➔ 이 관계가 성립하면 상속으로 표현하기 좋다.

Dog is an Animal.
강아지는 동물의 한 종류이다.
 	
Cat is an Animal.
고양이는 동물의 한 종류이다.
 	
Student is a Person.
학생은 사람의 한 종류이다.
 	
Rectangle is a Shape.
직사각형은 도형의 한 종류이다.
	class Animal:
	  pass
	
	class Dog(Animal):
	  pass

Inheritance

♣ has-a relationship

- A has a B: A는 B를 가지고 있다. ➔ 이 관계가 성립하면 객체를 속성으로 가지는 방식이 적절하다.

Car has an Engine.
자동차는 엔진을 가지고 있다.
 	
Computer has a CPU.
컴퓨터는 CPU를 가지고 있다.
 	
Student has a Backpack.
학생은 가방을 가지고 있다.
 	
Team has Players.
팀은 선수들을 가지고 있다.
	class Engine:
	  pass
	
	class Car:
	  def __init__(self):
	  self.engine = Engine()

Inheritance

♣ 상속 - Summary

- 단순히 코드를 재사용하기 위한 문법은 아니다.
- “A는 B의 한 종류이다.”라는 관계를 코드로 표현하는 방법
- A가 B의 한 종류이면 is-a relationship ➔ 상속으로 표현

```
class Parent:
    pass


class Child(Parent):
    pass
```

Exercise

[Problem]

1. Person class
   - name, age를 Instance Variable로 가진다.

2. Student class
   - Person을 상속 받고, score를 Instance Variable로 가진다.
   - print_info() 메소드는 이름, 나이, 점수를 출력한다.

```
s = Student(“Alice”, 17, 90)
s.print_info()
```

[출력]
이름: Alice
나이: 17
점수: 90

Exercise

[Problem] - Answer

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Student(Person):
    def __init__(self, name, age, score):
        super().__init__(name, age)
        self.score = score

    def print_info(self):
        print(f"이름: {self.name}")
        print(f"나이: {self.age}")
        print(f"점수: {self.score}")

s = Student("Alice", 17, 90)
s.print_info()
```

THANK YOU!

Any Question?