파이썬 프로그래밍

3주차: 객체지향 프로그래밍

목차

1. 객체지향 프로그래밍 기초

클래스와 객체

클래스(Class): 객체를 만들기 위한 설계도

객체(Object): 클래스로부터 만들어진 실체

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

    def bark(self):  # 메서드(method)
        return f"{self.name}가 짖습니다!"

# 객체 생성
my_dog = Dog("멍이", 3)
print(my_dog.bark())  # 멍이가 짖습니다!
```

self의 의미

`self`는 인스턴스 자기 자신을 가리키는 참조

```
class Person:
    def __init__(self, name):
        self.name = name  # self.name은 인스턴스 변수

    def introduce(self):
        return f"안녕하세요, {self.name}입니다."

person1 = Person("김철수")
person2 = Person("이영희")

print(person1.introduce())  # 안녕하세요, 김철수입니다.
print(person2.introduce())  # 안녕하세요, 이영희입니다.
```

캡슐화와 접근 제어

파이썬의 접근 제어 규칙:

- `public` : 어디서나 접근 가능
- `_protected` : 관례상 내부용 (실제로는 접근 가능)
- `__private` : 네임 맹글링 적용

```
class BankAccount:
    def __init__(self, balance):
        self.public = "공개"
        self._protected = "보호됨"
        self.__private = balance  # 실제로는 _BankAccount__private로 저장

    def get_balance(self):
        return self.__private
```

@property 데코레이터

getter/setter를 더 파이썬스럽게!

```python
class Temperature:
    def __init__(self, celsius=0):
        self._celsius = celsius

    @property
    def celsius(self):
        return self._celsius

    @celsius.setter
    def celsius(self, value):
        if value < -273.15:
            raise ValueError("절대영도보다 낮을 수 없습니다")
        self._celsius = value

temp = Temperature()
temp.celsius = 25  # setter 호출
print(temp.celsius)  # getter 호출
```

2. 상속과 다형성

상속의 개념

기존 클래스의 속성과 메서드를 물려받아 새로운 클래스 생성

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

    def speak(self):
        pass

class Dog(Animal):  # Animal 클래스 상속
    def speak(self):
        return f"{self.name}: 멍멍!"

class Cat(Animal):
    def speak(self):
        return f"{self.name}: 야옹~"
```

super() 함수

부모 클래스의 메서드를 호출

```
class Vehicle:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

class Car(Vehicle):
    def __init__(self, brand, model, doors):
        super().__init__(brand, model)  # 부모 클래스의 __init__ 호출
        self.doors = doors

    def info(self):
        return f"{self.brand} {self.model}, {self.doors}도어"

my_car = Car("현대", "소나타", 4)
print(my_car.info())  # 현대 소나타, 4도어
```

다형성 (Polymorphism)

같은 인터페이스로 다른 동작을 수행

```
def animal_sound(animal):
    return animal.speak()

dog = Dog("바둑이")
cat = Cat("나비")

print(animal_sound(dog))  # 바둑이: 멍멍!
print(animal_sound(cat))  # 나비: 야옹~

# 덕 타이핑 (Duck Typing)
# "오리처럼 걷고 오리처럼 소리내면 오리다"
```

3. 객체지향 프로그래밍 심화

특별 메서드 (Magic Methods)

객체의 특별한 동작을 정의

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

    def __str__(self):  # print()할 때
        return f"Point({self.x}, {self.y})"

    def __add__(self, other):  # + 연산자
        return Point(self.x + other.x, self.y + other.y)

    def __eq__(self, other):  # == 연산자
        return self.x == other.x and self.y == other.y

p1 = Point(1, 2)
p2 = Point(3, 4)
print(p1 + p2)  # Point(4, 6)
```

클래스 메서드와 정적 메서드

```
class MathUtils:
    pi = 3.14159

    @classmethod
    def circle_area(cls, radius):
        return cls.pi * radius ** 2

    @staticmethod
    def add(x, y):
        return x + y

# 클래스 메서드: 클래스 변수에 접근
print(MathUtils.circle_area(5))

# 정적 메서드: 클래스와 무관한 유틸리티
print(MathUtils.add(3, 4))
```

4. 실습: 은행 계좌 관리 시스템

은행 계좌 클래스 설계

```python
class Account:
    def __init__(self, account_number, owner, balance=0):
        self.account_number = account_number
        self.owner = owner
        self.__balance = balance
        self.transaction_history = []

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            self.transaction_history.append(f"입금: {amount}원")
            return True
        return False

    def withdraw(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount
            self.transaction_history.append(f"출금: {amount}원")
            return True
        return False
```

상속을 활용한 특수 계좌

```
class SavingsAccount(Account):
    def __init__(self, account_number, owner, interest_rate):
        super().__init__(account_number, owner)
        self.interest_rate = interest_rate

    def add_interest(self):
        interest = self.get_balance() * (self.interest_rate / 100)
        self.deposit(interest)
        return interest

# 사용 예시
savings = SavingsAccount("123-456", "홍길동", 2.5)
savings.deposit(1000000)
interest = savings.add_interest()
print(f"이자: {interest}원")
```

마무리

학습한 내용

- **3주차**: 객체지향 프로그래밍의 핵심 개념
  - 클래스와 객체, 상속, 다형성
  - 특별 메서드와 데코레이터

Q&A

질문 있으신가요?