객체지향의 핵심개념(다형성, 캡슐화)

작성일시	@2022년 7월 27일 오후 3:13
내용	OOP 파이썬
주차	2주차
자료	

다형성(Polymorphism)

- 여러 모형을 뜻하는 그리스어
- 동일한 메서드가 클래스에 따라 다르게 행동할 수 있음을 의미
- 서로 다른 클래스에 속해있는 객체들이 **동일한 메세지에 대해 다른 방식으로 응답할 수 있음**

메서드 오버라이딩

- 상속받은 메서드를 재정의
  - 클래스 상속 시, 부모 클래스에서 정의한 메서드를 자식 클래스에서 변경
  - 부모 클래스의 메서드 이름과 기본 기능은 그대로 사용하지만, 특정 기능을 바꾸고 싶을 때 사용

```
class Person:

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

    def talk(self):
        print(f'반갑습니다. {self.name}입니다.')

# 자식 클래스 - Professor
class Professor(Person):

    def talk(self):
        return (f'{self.name}일세')

class Student(Person):
```

```
    def talk(self):
        super().talk()
        print(f'저는 학생입니다.')

p1 = Professor('김교수')
p1.talk() # 김교수일세.

s1 = Student('이학생')
s1.talk()
# 반갑습니다. 이학생입니다. -> (super().talk())
# 저는 학생입니다. -> override
```

캡슐화

- 객체의 일부 구현 내용에 대해 외부로부터의 직접적인 엑세스를 차단
  - ex) 주민등록번호
- 파이썬에서 암묵적으로 존재하지만, 언어적으로는 존재하지 않음

접근제어자 종류

- Public Access Modifier
- Protected Access Modifier
- Private Access Modifier

Public Member

- 언더바 없이 시작하는 메서드나 속성
- 어디서나 호출이 가능, 하위클래스 override 허용
- 일반적으로 작성되는 메서드와 속성의 대다수를 차지

Protected Member

- 언더바 1 개로 시작하는 메서드나 속성
- 암묵적 규칙에 의해 부모 클래스 내부와 자식 클래스에서만 호출 가능
- 하위 클래스 override 허용

```
class Person:

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

```
        self._age = age

    def get_age(self):
        return self._age

# 인스턴스를 만들고 get_age 메서드를 활용하여 호출할 수 있습니다.
p1 = Person('김철수', 30)
print(p1.get_age()) # 30

# _age에 직접 접근하여도 확인이 가능
# 파이썬에서는 암묵적으로 활용할 뿐입니다
print(p1._age) # 30
```

Private Member

- 언더바 2개로 시작하는 메서드나 속성
- 본 클래스 내부에서만 사용 가능
- 하위 클래스 상속 및 호출 불가능(오류)
- 외부 호출 불가능

```
class Person:

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

    def get_age(self):
        return self.__age

# 인스턴스를 만들고 get_age 메서드를 활용하여 호출할 수 있습니다.
p1 = Person('김철수', 30)
print(p1.get_age()) # 30

# _age에 직접 접근하여도 확인이 가능
# 파이썬에서는 암묵적으로 활용할 뿐입니다
print(p1.__age) # AttributeError: 'Person' object has no attribute '__age'
```

getter 메서드와 setter 메서드

- **변수에 접근할 수 있는 메서드**를 별도로 생성
  - getter 메서드: 변수의 값을 읽는 메서드
    - `@property` 데코레이터 사용
  - setter 메서드: 변수의 값을 설정하는 성격의 메서드
    - `@<변수>.setter`사용

```
class Person:

    def __init__(self, age):
        self._age = age

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, new_age):
        if new_age <= 19:
            raise ValueError('Too Young')
            return

        self._age = new_age

p1 = Person(20)
print(p1.age) # 20

p1.age = 33
print(p1.age) # 33

p1.age = 19
print(p1.age) # ValueError: Too Young
```