객체지향의 핵심개념(추상화, 상속)

작성일시	@2022년 7월 27일 오후 2:25
내용	OOP 파이썬
주차	2주차
자료	

객체 지향의 핵심 4가지

- 추상화
- 상속
- 당향성
- 캡슐화

추상화

- 현실 세계를 프로그램 설계에 반영
  - 복잡한 것은 숨기고, 필요한 것만 드러내기

상속

- 상속이란
  - 두 클래스 사이 부모 = 자식 관계를 정립하는 것
- 클래스는 상속이 가능함
  - 모든 파이썬 클래스는 object를 상속 받음

```
class ChildClass(ParentClass):
    pass
```

- 하위 클래스는 상위 클래스에 정의된 속성, 행동, 관계 및 제약 조건을 모두 상속 받음
- 부모 클래스의 속성, 메서드가 자식 클래스에 상속되므로, 코드 재사용성이 높아짐

- 상속 없이 구현하는 경우 1
  - 학생 교수 정보를 나타내기 어렵다.

```
class Person:

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

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

s1 = Person('김학생', 23) # 반갑습니다. 김학생입니다.
s1.talk()

p1 = Person('박교수', 49) # 반갑습니다. 박교수입니다.
p1.talk()

s1.gpa = 4.5
p1.department = '컴퓨터공학과'
```

- 상속 없이 구현하는 경우 2
  - 메서드 중복 정의

```
class Professor:

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

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


class Student:

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

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


s1 = Student('김학생', 23, 4.5)
p1 = Professor('박교수', 49, '컴퓨터공학과')
```

- 상속을 통해 구현
  - 상속을 통한 메서드 재사용

```
 class Person:

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

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

class Professor(Person):

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

class Student(Person):

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

s1 = Student('김학생', 23, 4.5)
p1 = Professor('박교수', 49, '컴퓨터공학과')

# 부모 Person 클래스의 talk 메서드를 활용
s1.talk() # 반갑습니다. 김학생입니다.
# 부모 Person 클래스의 talk 메서드를 활용
p1.talk() # 반갑습니다. 박교수입니다.
```

상속 관련 함수와 메서드

- `.isinstance(object, classinfo)`
  - classinfo의 instance거나 **subclass***인 경우 True
- `.issubclass(class, classinfo)`
  - class가 classinfo의 subclass면 True
- `super().`

- 자식클래스에서 부모 클래스를 사용하고 싶은 경우

```
class Person:

    def __init__(self, name, age, number, email):
        self.name = name
        self.age = age
        self.number = number
        self.email = email

class Student(Person):
    def __init__(self, name, age, number, email, student_id):
        # Person 클래스
        super().__init__(name, age, number, email)
        self.student_id = student_id
```

상속 정리

- 파이썬의 모든 클래스는 object로부터 상속됨
- 부모 클래스의 모든 요소(속성, 메서드)가 상속됨
- super()를 통해 부모 클래스의 요소를 호출할 수 있음
- 메서드 오버라이딩을 통해 자식 클래스에서 재정의 가능함
- 상속관계에서의 이름공간은 인스턴스, 자식 클래스, 부모 클래스 순으로 탐색

다중 상속

- 두 개 이상의 클래스를 상속 받는 경우
- 상속 받은 모든 클래스의 요소를 활용 가능함
- 중복된 속성이나 메서드가 있는 경우 **상속 순서에 의해 결정**됨

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

    def greeting(self):
        return f'안녕, {self.name}'

class Mom(Person):
    gene = 'XX'

    def swim(self):
        return '엄마가 수영'

class Dad(Person):
    gene = 'XY'

    def walk(self):
        return '아빠가 걷기'

class FirstChild(Dad, Mom):
    def swim(self):
        return '첫째가 수영'

    def cry(self):
        return '첫째가 응애'

baby1 = FirstChild('아가')
print(baby1.cry()) # 첫째가 응애
print(baby1.swim()) # 첫째가 수영
print(baby1.walk()) # 아빠가 걷기
print(baby1.gene) # XY

class SecondChild(Mom, Dad):
    def walk(self):
        return '둘째가 걷기'

    def cry(self):
        return '둘째가 응애'
baby2 = SecondChild('아가')
print(baby2.cry()) # 둘째가 응애
print(baby2.swim()) # 엄마가 수영
print(baby2.walk()) # 둘째가 걷기
print(baby2.gene) # XX
```

- mro메서드 (Method Resolution Order)
  - 해당 인스턴스의 클래스가 어떤 부모 클래스를 가지는지 확인하는 메서드
  - 기존의 인스턴스 → 클래스 순으로 이름 공간을 탐색하는 과정에서 상속 관계에 있으면 인스턴스 → 자식클래스 → 부모클래스로 확장

```
print(FirstChild.mro()) # [<class '__main__.FirstChild'>, <class '__main__.Dad'>, <class '__main__.Mom'>, <class '__main__.Person'
print(SecondChild.mro()) # [<class '__main__.SecondChild'>, <class '__main__.Mom'>, <class '__main__.Dad'>, <class '__main__.Perso
```