[객체지향 프로그래밍, 클래스]

- 객체지향 프로그래밍
- 클래스 소개
- 생성자(constructor)
- 캡슐화, 접근자, 설정자
- 상속
- 메소드 오버라이딩

프로그래밍 방법

- 절차적 프로그래밍(Procedural Programming) 기법
  - 지금까지의 프로그램 작성 방법
  - 데이터를 다루는 명령어들이 모여 있는 함수들의 조합으로 프로그램을 작성
  - 문제해결 순서에 맞게 호출하여 수행하는 방법
  - C, Fortran, Basic 등의 언어에서 사용
- 객체지향 프로그래밍(Object-Oriented Programming)
  - 우리가 살고 있는 실제 세계의 모든 것들이 여러 객체(object)들로 구성되어 있는 것과 유사하게, 소프트웨어도 객체로 구성되어 있다는 생각에 바탕을 둔 프로그래밍 방법
  - 객체(object)의 속성(변수)과 동작(함수)을 나타내는 클래스를 정의하고
  - 클래스를 이용해 객체를 생성(인스턴스화; Instantiation)하여 작성한 프로그램
  - 파이썬, 자바, C++ 등의 언어에서 사용

객체지향 프로그래밍 방법

- 객체(object) = 속성(attribute) + 기능(method). 또는 상태(state) + 동작(behavior)
- 예: 자동차는 제조사, 모델, 년식, 색상, 속도 등의 속성과 가속, 감속, 좌회전, 우회전 등의 기능
- 객체 지향 프로그래밍은 이해하기가 조금 어렵지만 제대로 프로그래밍하는 방법을 배우면
- 나중에 고급 프로그램을 만들 때 쉽고, 유지 보수하기 쉬운 장점이 있다.
- 파이썬에서는 모든 것이 다 객체이다. 문자열도 객체이고, 리스트도 객체이며, 심지어 정수도 객체이다.
- 셸에서 입력해 각 객체의 형태를 확인해 보자

클래스(class)

- 객체지향 프로그래밍에서 가장 핵심적인 내용은 클래스(class)임
- 객체를 표현하기 위해 클래스를 만들어 사용
- 일반적으로 클래스는 다음과 같은 모양으로 작성
- 생성자, 멤버함수(들), 클래스 변수등은 들여쓰기를 한 블록에 정의
- 클래스 이름의 첫 글자는 대문자로 하는 것이 일반적인 관례

```
class ClassName:
    클래스변수(들)
    메소드(들)
```

가장 간단한 클래스

- 다음과 같이 셀에서 실행

```
>>> class Foo:
...     pass
...
>>> f = Foo()
>>> print(f)
<__main__.Foo object at 0x0000026B4F38C440>```
```

- pass문을 이용해 아무 기능도 없는 클래스 Foo를 생성
- 이 경우 인스턴스 t를 만들고 난 후(빈 클래스로도 인스턴스를 만들 수 있다)
- print문으로 결과를 확인해 보면 이 객체는 main 모듈의 Foo 클래스의 인스턴스임을 알 수 있다.

클래스와 인스턴스, 객체

- 보통 속성은 변수로, 동작은 메소드(또는 멤버함수)로 구현한다.(클래스 안에서 구현되는 함수를 메소드라 한다.)
- 위와 같이 객체에 대한 속성과 동작을 작성하는 설계도를 클래스(class)라고 하며,
- 클래스로부터 만들어지는 각각의 객체를 그 클래스의 인스턴스(instance)라고 한다.
- 객체와 인스턴스의 차이(교재)
  - 클래스로 만든 객체를 `인스턴스` 라고도 한다.
  - 그렇다면 객체와 인스턴스의 차이는 무엇일까? 이렇게 생각해 보자. a = Foo()로 만든 a는 객체이다. 그리고 a 객체는 Foo의 인스턴스이다. 즉, 인스턴스라는 말은 특정 객체(a)가 어떤 클래스(Foo)의 객체인지를 관계 위주로 설명할 때 사용한다.
  - `a는 인스턴스` 보다 `a는 객체` 라는 표현이 어울리며 `a는 Foo의 객체` 보다 `a는 Foo의 인스턴스` 라는 표현이 훨씬 잘 어울린다.

자동차 클래스 표현 _

```
class Car :
    def __init__(self, color, speed) : # 초기화 메소드
        self.color = color          # 인스턴스 변수 정의
        self.speed = speed

    def speedUp(self, v): # 가속 메소드. 1번째 매개변수는 self
        self.speed = self.speed + v
        return self.speed

    def speedDown(self, v): # 감속 메소드
        self.speed = self.speed - v
        return self.speed

c1 = Car('black', 50)
c2 = Car('red', 70)
print('Car c1 : color=%s, speed=%d ' %(c1.color, c1.speed))
print('Car c2 : color=%s, speed=%d ' %(c2.color, c2.speed))
c1.speedDown(10)          # 차 c1의 속도 10만큼 감속
print('Car c1 : speed=%d' %c1.speed)
```

생성자(constructor)

- 인스턴스를 생성할 때 무조건 호출되는 메소드인데, 반드시 이름은 `__init__()`고 붙인다.
- 매개변수에 self만 있는 생성자를 기본 생성자라고 한다.
- 생성자의 첫 매개변수인 self는 클래스 인스턴스이다.
- self를 사용하는 이유는 메소드 안에서 필드에 접근하기 위함
- 메소드 안에서 필드에 접근할 일이 없다면 self 생략 가능. 하지만 거의 모든 메소드의 첫번째 매개변수에 self를 사용.
- 위의 생성자는 self 외에 두 개의 매개변수 color, speed가 있다.
- 생성자를 만들어 사용하면 인스턴스를 생성하면서 필드값을 초기화까지 할 수 있어 편리

People 클래스의 정의와 사용

- 처음 인스턴스를 만들 때 나이와 이름을 갖도록 생성자를 작성

```
class People :
    def __init__(self, age=0, name=None):
        self.__age = age
        self.name = name # `self.__name`으로 바꿔 쓰는 것 권장

p1 = People(20, 'Kim')
print(p1.name)
print(p1.__age)
```

- 위 프로그램을 실행해 보면 이름은 제대로 출력하지만 나이는 출력되지 않고 다음과 같은 에러를 만나게 됨 `AttributeError: 'People' object has no attribute '__age'`
- 차이 발생 이유 : age 변수명 앞에는 `__`가 있고, name 변수명 앞에는 `__`이 없기 때문
- 클래스 안에서 만든 변수의 이름 앞에 `__`를 추가해 사용하면 그 변수의 값을 외부에서 읽거나 변경하지 못함

캡슐화(encapsulation)

- 데이터와 알고리즘을 하나로 묶어 공용 인터페이스만 제공하고 구현의 세부 사항을 감추는 것
- 감춘 인스턴스 변수 값을 알고 싶거나, 바꾸고 싶으면?
  - 클래스 내부에 인스턴스 변수값을 반환하는 접근자(getter)와
  - 인스턴스 변수값을 설정하는 설정자(setter)를 구현하는 것이 좋다.
  - 접근자와 설정자는 각각 get과 set으로 시작하는 이름으로 만드는 것이 관례

접근자(getter)와 설정자(setter)를 사용한 프로그램 _

```
class People :
    def __init__(self, age=0, name=None):
        self.__age = age
        self.__name = name

    def getAge(self):
        return self.__age

    def getName(self):
        return self.__name

    def setAge(self, age):
        self.__age = age

    def setName(self, name):
        self.__name = name

p1 = People(20, "Kim")
print(p1.getName())
print(p1.getAge())

p1.setName('Lee')
p1.setAge(21)
print(p1.getName())
print(p1.getAge())
```

상속(Inheritance)

- 객체 지향 프로그래밍의 장점은 코드를 재사용 할 수 있다는 것인데 이를 위한 방법으로 상속이 사용됨
- 상속은 클래스 간의 형식과 세부 형식을 구현하는 것
- 클래스의 상속은 기존에 존재하는 클래스로부터 코드와 데이터를 이어 받고 자신이 필요한 기능을 추가하는 기법
- 따라서 상속을 받으면 부모클래스보다 자식클래스의 정보가 더 구체화되고 많아진다.
- 상위 클래스 : 부모클래스(parent class) 또는 수퍼 클래스(super class)
  하위 클래스 : 자식클래스(child class) 또는 서브 클래스(sub class)라 함
- 상속을 구현하는 방법은 서브 클래스를 정의할 때 괄호 안에 수퍼클래스의 이름을 넣는다.

```
class subClass이름 (superClass이름) :
```

People 클래스를 상속받아 Teacher 클래스 구현 _

```
class People :
    def __init__(self, age=0, name=None):
        self.__age = age
        self.__name = name

    def introMe(self):
        print("Name :", self.__name, "age :", str(self.__age))

class Teacher(People) :
    def __init__(self, age=0, name=None, school=None) :
        super().__init__(age, name) # 부모클래스 호출. self 매개변수 없음
        self.__school = school      # 자신의 인스턴스변수 추가

    def showSchool(self):
        print("My school : ", self.__school)

p1 = People(29, "Lee")    # People 객체 호출
p1.introMe()              # People.introMe() 호출

t1 = Teacher(48, "Kim", "HighSchool")    # Teacher 객체
t1.introMe()                             # People.introMe() 호출
t1.showSchool()           # Teacher.showSchool() 호출
```

- Teacher 클래스는 People 클래스를 상속받아, 학교의 이름을 추가
- 자기 자신을 소개하는 메소드 introMe()는 부모클래스의 메소드 introMe()를 그대로 상속받아 사용
- 학교의 이름을 알리는 메소드 showSchool()을 추가
- 부모 클래스의 메소드를 그대로 이어 받아 사용하려면 자신의 메소드 이름 앞부분에 super().을 붙이고 매개변수에서는 self를 빼고 사용
- 이와 같이 모든 사람에 공통적인 내용은 부모클래스인 People 클래스에 구현하고, 교사에게 필요한 내용은 자식클래스에 구현하는 것이 상속을 이용한 프로그래밍 방법

메소드 오버라이딩(method overriding)

- 자식클래스에서 부모클래스의 메소드 중 필요한 부분을 수정해 다시 정의해 사용하는 것
- 앞의 프로그램을 보면 Teacher 클래스에 showSchool() 메소드를 추가했는 데, 이 메소드가 하는 일은 학교의 정보를 보여주는 일에 불과하다. 부모클래스에 이미 존재하는 introMe() 메소드를 상속받아 여기에 학교의 정보를 보여주는 것을 추가하는 것이 더 낫다.
- 이렇게 작성된 것이 다음 프로그램이다.
- 이 프로그램은 새로운 클래스인 Student 클래스도 추가로 정의했는 데, Student 클래스는 학교 이름과 학년정보를 포함하는 인스턴스 변수를 갖고 있다.

Student 클래스 _

```
class People :
    def __init__(self, age=0, name=None):
        self.__age = age
        self.__name = name

    def introMe(self):
        print("Name :", self.__name, ", age :", str(self.__age))

class Teacher(People) :
    def __init__(self, age=0, name=None, school=None) :
        super().__init__(age, name) # 부모클래스 호출. self 매개변수 없음
        self.__school = school      # 자신의 인스턴스변수 추가

    def introMe(self) :
        super().introMe()
        print("My school : ", self.__school)

class Student(Teacher) :
    def __init__(self, age=0, name=None, school=None, grade=None) :
        super().__init__(age, name, school) # 매개변수에 self 없음에 주의
        self.__grade = grade

    def introMe(self) :
        super().introMe()
        print("Grade : ", self.__grade)

p1 = People(29, "Lee")   # People 객체 호출
p1.introMe()             # People.introMe() 호출

t1 = Teacher(48, "Kim", "HighSchool")   # Teacher 객체
t1.introMe()                            # People.introMe() 호출

s1 = Student(17, "Park", "HighSchool", 2)
s1.introMe()
```

직사각형을 정의하는 클래스 Rectangle의 구현 _

1. 직사각형을 정의하는 클래스 Rectangle에 대해 생각(어떤 내용이 필요할 지 등등 ..)해 보라
2. 직사각형의 가로와 세로가 주어질 때 다음의 각 메소드를 구현하라.
    - 생성자 `__init__(self, w, h)`
    - 면적을 계산하는 `area(self)`
    - 둘레를 계산하는 `perimeter(self)`
3. 접근자/설정자를 추가로 구현하라

(직사각형 클래스를 상속해) 정사각형 클래스를 추가하고 이를 시험하라. _

- 가로/세로 변수, 접근자/설정자, 면적/둘레길이를 구하는 메소드를 구현해야 함