11. 클래스

11.1 객체지향 프로그래밍 이해하기

파이썬을 기능별로 나누어서 프로그래밍하는 방법

- 함수
- 모듈
- 객체지향 프로그래밍

객체지향 프로그래밍이란?

프로그램의 구성요소들을 독립된 객체로 정의

객체는 속성(변수)과 기능(함수)으로 구성

객체지향프로그래밍의 목적

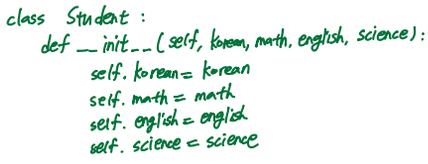
프로그램 내부 요소의 독립성 확보

지속적 수정 및 업그레이드 용이성

분업 용이성

11.2 Class 이해하기

Class란?



파이썬에서 객체를 어떻게 구성할 것인가에 대한 설계도

파이썬의 자료형 중 하나

설계도(클래스)를 바탕으로 실제로 구현된 객체를 인스턴스라고 한다.

```
class Student:
    def __init__ (self, name, korean, math, english, science):
        self.name = name
        self.korean = korean
        self.math = math
        self.english = english
        self.science = science

    def get_sum(self):
        return self.korean + self.math + self.english + self.science
    def get_average(self):
        return self.get_sum / 4
    def get_string(self):
```

```
        return "{}\t {}\t {}}".format(self.name, self.get_sum(),
self.get_average())
```

클래스 정의 시 유의사항

- 대문자로 시작
- 목적과 의미가 분명해야 함
- 영어 대소문자, 숫자, _로 구성됨
- 숫자로 시작 불가

생성자란?

객체 생성 시, 변수 선언 및 초기화를 담당하는 메소드

객체 자기 자신을 의미하는 self를 매개변수로 받음.

```
def __init__(self, name, korean, math, english, science):
    self.name = name
    self.korean = korean
    self.math = math
    self.english = english
    self.science = science
```

메소드 정의하기

일반적인 함수 선언하는 방법과 똑같은데, 메소드마다 첫 번째 매개변수로 self를 사용해야 함.

```
def get_sum(self):
    return self.korean + self.math + self.english + self.science
def get_average(self):
    return self.get_sum()/4
def get_string(self):
    return "{}\t {}\t {}\t".format(self.name, self.get_sum(),
self.get_average())
```

인스턴스 (객체) 생성하기

```
students = Students("James", 98,97,92,94)
```

객체를 배열로 나타낼 수도 있음

```
students = {
    Students("James", 98,97,92,94),
    Students("Kates", 98,97,92,94)
}
```

이렇게 생성된 인스턴스는 클래스의 메소드를 사용할 수 있다.

```
students.get_string()
'''
James, 381, 95.25
'''
```

11.3 클래스의 고급사용

어떤 클래스의 인스턴스인지 확인하기

```
students = Students("James",100,92,94,92,93)
isinstance(students, Students) # True
```

James = Student("james", 97, 96, 93, 87)
print(~~isinstance~~ is instance (James, Student))
>> True

클래스 상속하기

클래스를 만들 때 다른 클래스의 기능을 물려받을 수 있다.

물려주는 클래스를 부모(parent)클래스, 물려받는 클래스를 자식(child)클래스라고 정의한다.

자식클래스는 부모클래스의 모든 변수와 메소드를 사용할 수 있다.

상속의 목적

1. 기존클래스의 큰 변경 없이 기능을 추가하거나 수정할 수 있다.
2. 기존 클래스가 패키지 형태로만 제공하거나 수정허용하지 않는 상황일 때

아래와 같이 부모 클래스인 `Calculator.py`가 있다.

```
class Calculator():
    def __init__(self, a,b):
        self.a = a
        self.b = b
    def add(self):
        return self.a + self.b
    def sub(self):
        return self.a - self.b
    def mult(self):
        return self.a * self.b
    def div(self):
        return self.a/self.b
```

그리고 이를 상속 받는 자식 클래스는 아래와 같이 선언할 수 있다.

```
class MoreCalculator(Calculator):
    def pow(self):
        return self.a**self.b
```

MoreCalculator에서 따로 선언하지 않아도 바로 사용이 가능하다

부모클래스를 가져와 수정하는 오버라이드도 다음과 같이 가능하다.

```
class MoreCalculator(Calculator):
    def div(self):
        if self.b == 0:
            return 0
        else:
            return self.a/self.b
```

super() 를 써서 자식 클래스에서 부모클래스 객체를 사용할 수 있다.

super() 는 부모 클래스를 접근할 수 있게 하는 함수로,

자식클래스에서 부모클래스의 속성에 일부 속성을 추가하거나 기능을 덧댈 수 있다.

```
class MoreCalculator(Calculator):
    def __init__(self, a,b,c):
        super().__init__(a,b)
        self.c = c
    def add(self):
        print("덧셈입니다.")
        return super().add()+self.c
```

프라이빗 변수 사용하기

객체 중 일부 변수를 남이 볼 수 없도록 보호하는 것

- 다른 사람에게 코드 전달 시, 본래 의도대로 프로그램이 작동할 수 있게 제한을 두어 오용을 방지
- 필요없는 정보를 숨김
- 제품판매시 소스의 보호

클래스 외부에서 접근 불가능한 변수

```
import math

class Circle:
    def __init__(self, radius):
        self.__radius = radius

    def get_length(self):
        return 2*math.pi*self.__radius
    def get_area(self):
        return math.pi*self.__radius**2

circle = Circle(10)
print("circle의 둘레는 {}".format(circle.get_length))
print("circle의 넓이는는 {}".format(circle.get_area))
print("circle의 반지름은 {}".format(circle.__radius))
```

getter 와 setter

getter : 프라이빗 변수에 접근할 수 있도록 해주는 함수

```
@property
def radius(self):
    return self.__radius
```

setter : 프라이빗 변수의 값을 설정해주는 함수

```
@radius.setter
def radius(self, value):
    self.__radius = value
```

```
import math

class Circle:
    def __init__(self, radius):
        self.__radius = radius
    def get_length(self):
        return 2*math.pi*self.radius
    def get_area(self):
        return math.pi*self.radius**2

    @property
    def radius(self):
        return self.__radius
    # getter를 통해 프라이빗변수인 __radius에 접근할 수 있게 됨.

    @radius.setter
    def radius(self, value):
        self.__radius = value
```

```
circle = Circle(10)
print("둘레 : {}".format(circle.get_length()))
print("넓이 : {}".format(circle.get_area()))
print("반지름 : {}".format(circle.radius))
circle.radius = 2
print("반지름 : {}".format(circle.get_length()))
```