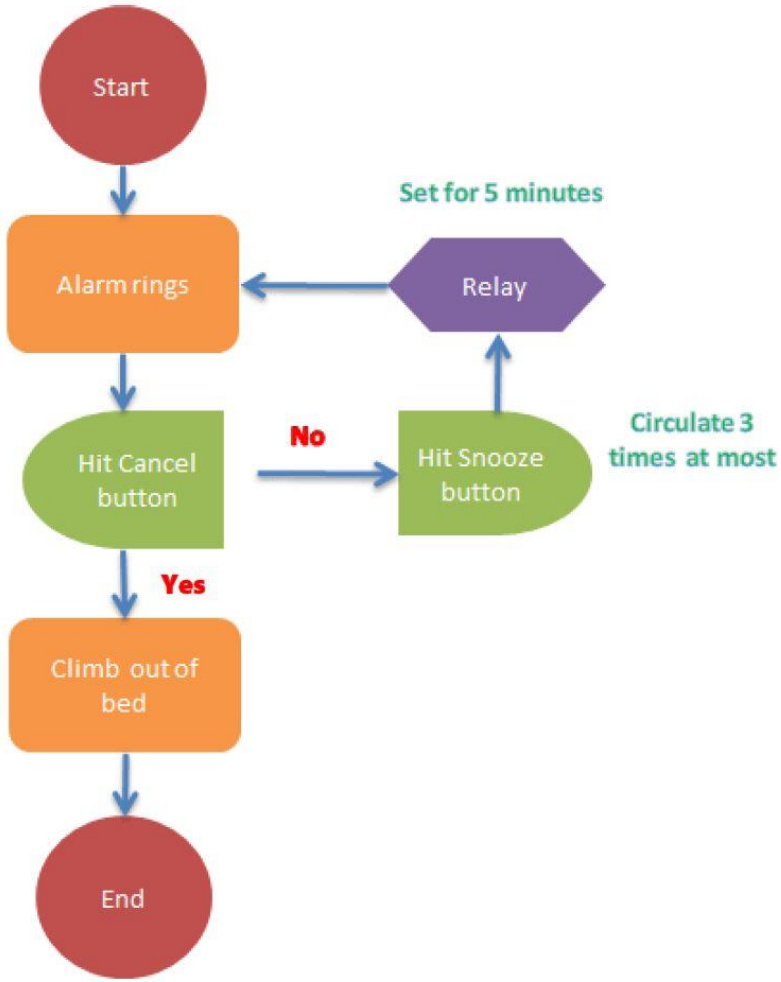
12주차
파이썬 기초 프로그래밍

수업 내용

- 객체지향 프로그래밍이란?
- 클래스, 객체란?
- 함수와 메소드
- 정보 은닉(encapsulation)
- 상속(Inheritance)
- 다양상(polymorphism)

객체지향 프로그래밍 이란?

- 절차적 프로그래밍
  - 문제를 해결하기 위하 작업을 절차적으로 기술하는 프로그래밍 방법
  - 작업의 시작부터 끝까지 하나의 흐름으로 문제를 처리하는 경우에 적합



객체지향 프로그래밍 이란?

- 객체지향 프로그래밍
  - 객체와 객체의 상호작용에 의하여 문제 해결
- 객체(Object)
  - 데이터 + (데이터를 처리하는) 함수(메소드)
  - data
    - attribute, member variable. state, field
  - operation
    - behavior, member function, method
- 객체지향 언어
  - 객체를 쉽게 표현하여 사용할 수 있도록 이와 관련된 기능을 제공하는 언어

객체의 예 : 자동차

- 속성 : 색상, 바퀴 크기, 배기량...
- 기능 : 가속, 감속, 전진, 후진, 좌회전, 우회전...

```
color = 'red'
displacement = 2000

def forward() :  # 전진
        pass
def backward() : # 후진
        pass
def turn_left() : # 좌회전
        pass
def turn_right() : # 우회전
        pass
```

```
class Car :
    def __init__(self) :
        self. color = 'red'
        self.displacement = 2000

    def forward() :  # 전진
            pass
    def backward() : # 후진
            pass
    def turn_left() :   # 좌회전
            pass
    def turn_right() : # 우회전
            pass
```

객체지향 프로그래밍을 사용하는 이유?

- 프로그램의 유지 보수가 편리

- 클래스는 그 내부에 데이터와 데이터를 다르는 함수(메소드)들이 모두 들어 있어, 데이터나 기능이 수정되는 경우 해당 클래스만 수정하면 됨

객체지향의 중요 개념

- 클래스와 객체
- 상속
- 정보 은닉(캡슐화)
- 다양성

클래스(Class)

- 객체를 생성하는 틀(template)
  - 속성
  - 메소드
- 객체(Object)
  - 클래스를 이용하여 만든 변수
  - 인스턴스(instance)라고도 부름

클래스 정의


class 클래스이름 :
변수 초기화
def __init__ (self, ...)
...
def 메서드1 (self, ...)
...
클래스 속성들.
모든 객체 공유!!!
생성자
메서드들

```
s = 'happy coding'
print(s.capitalize())
```

```
class str(object):
    """
    str(object='') -> str
    str(bytes_or_buffer[, encoding[, errors]]) -> str

    Create a new string object from the given object. If encoding or
    errors is specified, then the object must expose a data buffer
    that will be decoded using the given encoding and error handler.
    Otherwise, returns the result of object.__str__() (if defined)
    or repr(object).
    encoding defaults to sys.getdefaultencoding().
    errors defaults to 'strict'.
    """
    def capitalize(self, *args, **kwargs): # real signature unknown
        """
        Return a capitalized version of the string.

        More specifically, make the first character have upper case and th
        case.
        """
        pass
```

Person 클래스 만들기

- 이름, 나이, 성별, 키를 요소로 갖는다.
- self는 자신의 인스턴스를 가르치는 레퍼런스(참조)이다.

```
# 사람 클래스 만들기
class Person() :
    # __init__ 은 클래스가 시작될 때 자동으로 실행되는 메서드
    # 클래스 안에 반드시 선언할 것!
    person_num = 0    # person_num : 클래스 변수
    def __init__ (self, name, age, gender, height) :
        # self.name : 인스턴스 변수
        # name : 인수로 전해진 값
        Person.person_num += 1
        self.name = name
        self.age = age
        self.gender = gender
        self.height = height
    def speak(self, msg) :
        print(msg)
```

Method

객체(Object) 생성

- sjyoun 객체 생성
  - sjyoun의 속성값을 넣어 주어야!

```
sjyoun = person('윤소정', 25, 'f', 166)
mother = person('엄마', 45, 'f', 170)
```

- 객체 사용

```
print(sjyoun.name)
print(mother.age)
print(Person.person_num)
mother.speak('빨리 일어나, 학교가야지~')
```

실습 01

- 다음 코드를 실행하고 결과 값을 알아 봅시다.

```
class Dog :
    attr1 = "mamal"
    attr2 = "dog"

    def fun(self):
        print("I'm a", self.attr1)
        print("I'm a", self.attr2)

Rodger = Dog()
print(Rodger.attr1)
Rodger.fun()
```

실습 02

- 다음 코드를 실행하고 결과 값을 알아 봅시다.

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

    def say_hi(self):
        print('Hello, my name is', self.name)

p = Person('jhkim')
p.say_hi()
```

클래스 속성, 인스턴스 속성

- 클래스 속성은 모든 인스턴스가 그 값의 변화를 공유함.
  - 클래스_이름.속성
  - Person.person_num

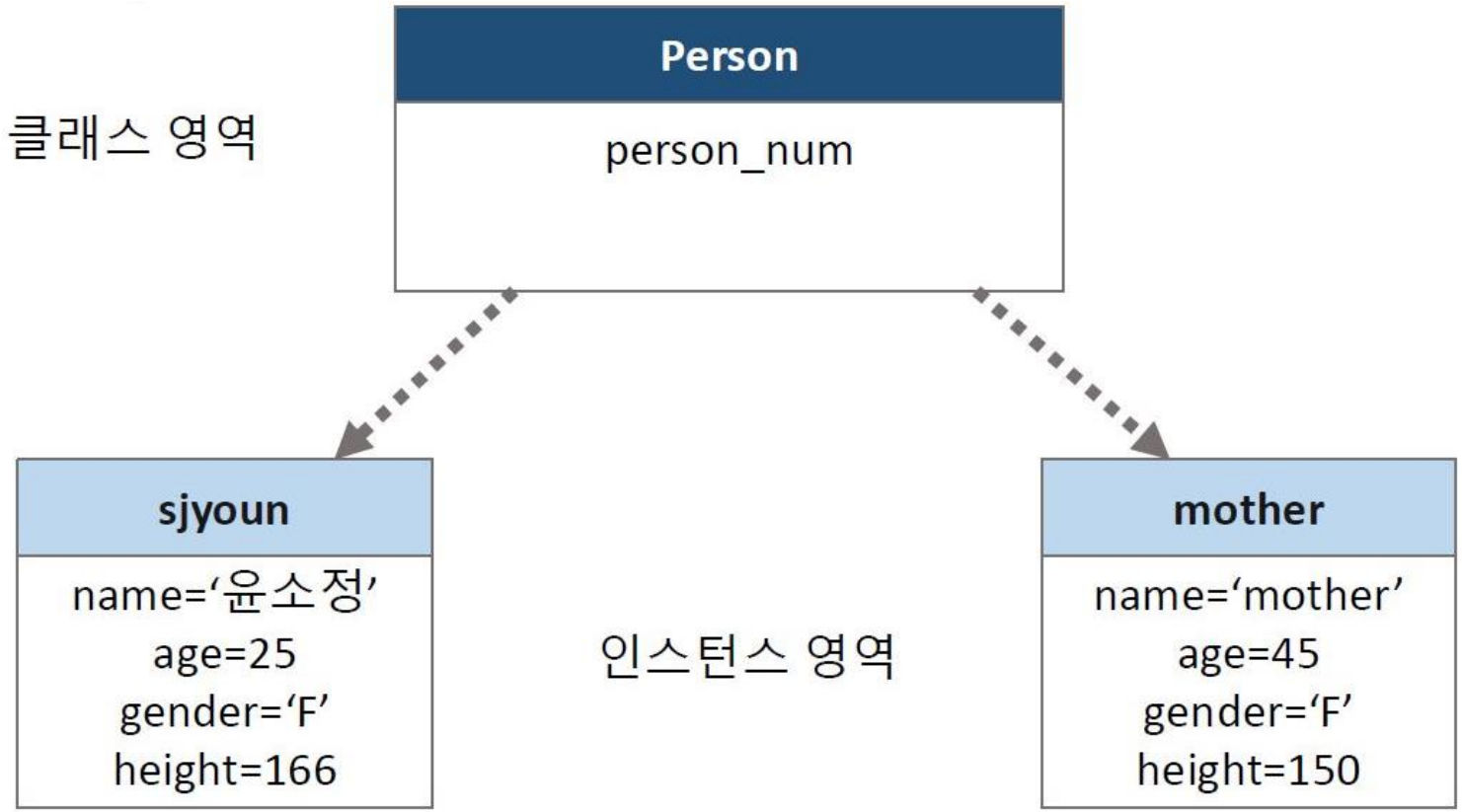
- 인스턴스 속성은 각 인스턴스 별로 속성 관리
  - 인스턴스_이름.속성
  - sjyoun.name

☞ 메서드 안에서 사용할 수 있는 변수

1. 클래스 변수 : Person.person_num
2. 인스턴스 변수 : self.name
3. 로컬 변수 : name

클래스 속성, 인스턴스 속성

- name space



함수와 메소드

- 함수(function)
  - 모든 데이터 타입에 사용할 수 있는 기능
  - 독립적으로 사용
  - `print(), input(), int()`

- 메소드(method)
  - 특정 클래스 안에 정의한 함수
  - 그 클래수로 생성한 객체만 사용 가능
  - **`객체.method()`** 형식으로 호출
    - `friends.append('홍길동')`

실습 03

- 다음 코드를 실행하고 결과 값을 알아 봅시다.

```
class Dog:

    # Class Variable
    animal = 'dog'

    # The init method or constructor
    def __init__(self, breed, color):

        # Instance Variable
        self.breed = breed
        self.color = color
```

```
Rodger = Dog("Pug", "brown")
Buzo = Dog("Bulldog", "black")

print('Rodger details:')
print('Rodger is a', Rodger.animal)
print('Breed: ', Rodger.breed)
print('Color: ', Rodger.color)

print('\nBuzo details:')
print('Buzo is a', Buzo.animal)
print('Breed: ', Buzo.breed)
print('Color: ', Buzo.color)

print("\nAccessing class variable using class name")
print(Dog.animal)
```

실습 04

- 다음 코드를 실행하고 결과 값을 알아 봅시다.

```
class Dog:

    # Class Variable
    animal = 'dog’

    def __init__(self, breed):
        # Instance Variable
        self.breed = breed

    def setColor(self, color):
        self.color = color

    def getColor(self):
        return self.color

Rodger = Dog("pug")
Rodger.setColor("brown")
print(Rodger.getColor())
```

```
class Dog:

    # Class Variable
    animal = 'dog’

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

    def setColor(self, color):
        self.__color = color

    def getColor(self):
        return self.__color

Rodger = Dog("pug")
Rodger.setColor("brown")
print(Rodger.getColor())
```

정보 은닉(캡슐화 : Encapsulation)

- 클래스 안의 변수들을 외부에서 직접 접근하지 못하도록 하는 것
- 이유?
  - 외부에서 부정확한 값으로 수정 가능
  - 유지 보수의 편리성(변수 타입의 변경)
- 방법
  - 클래스 안의 변수명을 __(_ 두개)로 시작하면, 이 변수는 내부에서만 보임

변수값을 가져오거나 수정하는 메서드 필요!
예 : getAge(), setAge()

정보 은닉(캡슐화 : Encapsulation)

```
class Person() :
    person_num = 0    # person_num : 클래스 변수
    def __init__ (self, name, age, gender, height) :
        # self.name : 인스턴스 변수
        # name : 인수로 전해진 값
        Person.person_num += 1
        self.name = name
        self.__age = age
        self.gender = gender
        self.height = height
    def speak(self, msg) :
        print(msg)
    def  getAge(self) :
        return self.__age
    def  setAge(self, age) :
        self.__age = age
```

```
>>> sjyoun = Person('윤소정', 22, 'f', 166)
>>> sjyoun.__age
  → 결과는?
>>> sjyoun.__age = 15
>>> sjyoun.__age
>>> sjyoun.getAge()
>>> sjyoun.setAge(15)
>>> sjyoun.__age
>>> sjyoun.getAge()
```

실습 05

- 다음 코드를 실행하고 결과 값을 알아 봅시다.

```
class Base:
    def __init__(self):
        self.a = ‘Happy Coding!’
        self.__c = ‘Python Coding!’
class Derived(Base):
    def __init__(self):
        Base.__init__(self)
        print("Calling private member of base class: ")
        print(self.__c)

obj1 = Base()
print(obj1.a)
```

- 아래 코드를 결과가 어떻게 될까? 그 이유를 설명하시오.

① `obj2 = Derived()`

상속(Inheritance)

- 기존에 존재하는 클래스로 부터 데이터와 메소드를 이어받는 기법
- 이어 받는 데이터, 메소드에 필요한 기능만 추가
- 장점
  - 소프트웨어의 재사용 가능
  - 확장성
  - 코드의 중복이 줄어듬
  - 개발 및 유지보수 용이

상속 : 정의

상속 : 예

```
class Student (Person) :
    def __init__ (self, name, age, gender, height, st_number)
            self.name = name
            self.age = age
            self.gender = gender
            self.height = height
            self.st_number = st_number
            self.classList = []
    def enrollClasses(self, subject)
            self.classList.append(class)
    def dropClass(self, class)
            self.classList.remove(class)

st1 = Student('김성희', 20, 'm', 180, '2018100001')
st1.enrollClasses('파이썬')
st1.speak('파이썬이 좋아요')
```

person 클래스의 생성자!
super().__init__(name, age, gener, height)
로 대체 가능

실습 06

- 이전 페이지의 상속 : 예를 코딩하고 결과를 출력 하시오.
  - Person -> Student 상속

실습 07

다음 코드를 실행하고 결과 값을 알아 봅시다.

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

       def display(self):
            print(self.name)
            print(self.idnumber)

class Employee( Person ):
        def __init__(self, name, idnumber, salary, post):
            self.salary = salary
            self.post = post
            Person.__init__(self, name, idnumber)

a = Employee('Rahul', 886012, 200000, "Intern")
a.display()
```

다양성(polymorphism)

- 다양성은 하나의 클래스나 메소드가 다양한 방식으로 동작이 가능한 것을 의미함
  - 내장 함수에서 다양성
    - 오버로딩(overloding)
  - 정의한 함수에서 다양성
    - 오버로딩(overloding)
  - 클래스 메소드에서 다양성
  - 상속에서 다양성
    - 메소드 오버라이딩(overriding)
  - 함수, 객체에서의 다양성

좋은 객체지향 프로그래밍

- 클래스 설계를 잘한 프로그램
- 독립적 기능의 객체 정의, 적절한 상속, 캡슐화, 다양성...
- 우리는?
  - 이미 잘 정의된 클래스를 포함하는 프로그램을 읽고 이해
  - 간단한 클래스를 정의하여 사용

실습

- 다음주 GUI 모듈 tkinter를 배운 후 작성해 보기 바랍니다.

- 윈도우를 생성하고 아이디/비밀번호를 입력받아 로그인을 실행하는 Login 객체를 정의하시오

```
from tkinter import messagebox
import tkinter as tk

class Login() :

   def __init__ (self) :
       … 윈도우 생성

   def doLogin(self, *event) :
       … 콜백 함수

if __name__ == '__main__' :
   login = Login()
```