

# 클래스

1. 클래스
2. module
3. package

## ■ 클래스

- 객체지향의 가장 기본적 개념
- 관련된 속성(모습)과 동작(기능)을 하나의 범주로 묶어 실세계의 사물을 표현
- 모델링 : 사물을 분석하여 클래스로 작성 (속성, 기능)
- 멤버 : 클래스를 구성하는 변수와 함수
- 메서드 : 클래스에 소속된 함수
- 객체로써 활용되려면 self 사용
- 단순히 기능만 사용할 때는 self 사용하지 않아도 관계없음

```
class 클래스:
```

```
def __init__(self):  
    self.전역변수 = 10
```

생성자

```
def run():  
    지역변수 = 100  
    print('함수 동작 %s' % 지역변수)
```

함수

```
def add(self):  
    self.전역변수 += 10
```

메소드

```
def show(self):  
    return self.전역변수
```

메소드

```
객체1 = 클래스()  
print(객체1.show())
```

객체 사용

```
객체2 = 클래스()  
객체2.add()  
print(객체2.show())
```

```
클래스.run()
```

함수 사용

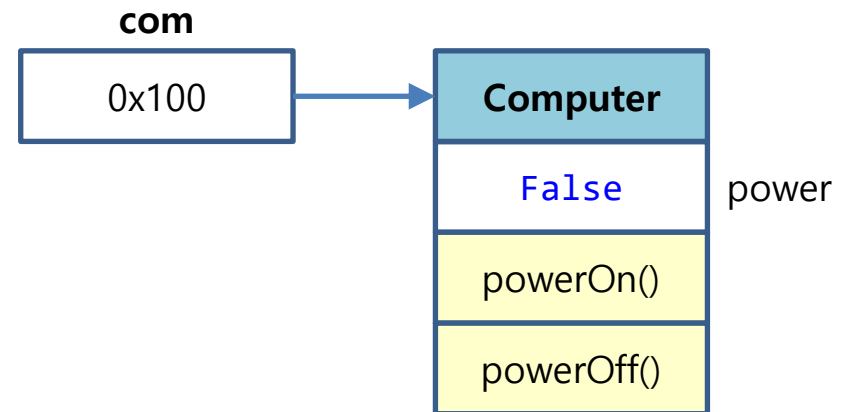
## ■ 클래스

- 클래스(Class) : 설계도면
- 객체(Object) : 클래스(설계도면)에 의해 만들어진 완성물(제품)
- 인스턴스(Instance) : 객체와 같은 의미
  - com은 객체, Computer 클래스의 인스턴스

```
class Computer:
    def powerOn(self):
        self.power = True
        print('전원 ON')

    def powerOff(self):
        self.power = False
        print('전원 OFF')

com = Computer()
com.powerOn()
com.powerOff()
```



## ■ 클래스

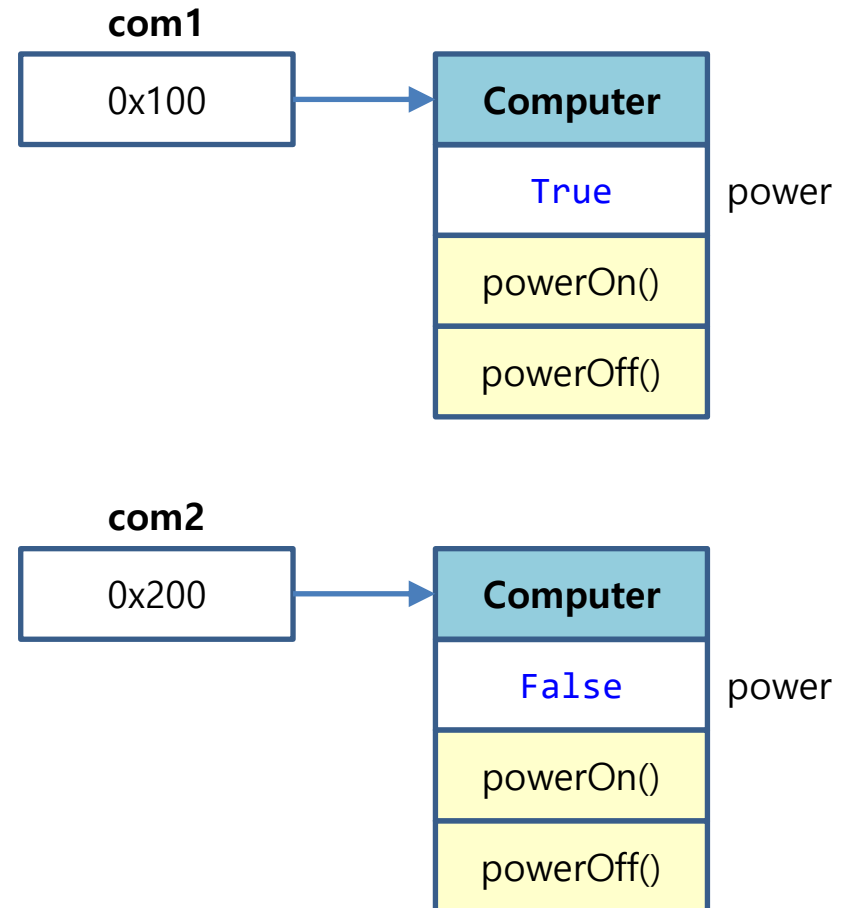
### ● 인스턴스(Instance) 생성

```
class Computer:
    def powerOn(self):
        self.power = True
        print('전원 ON')

    def powerOff(self):
        self.power = False
        print('전원 OFF')

com1 = Computer()
com1.powerOn()

com2 = Computer()
com2.powerOff()
```



## ■ 클래스

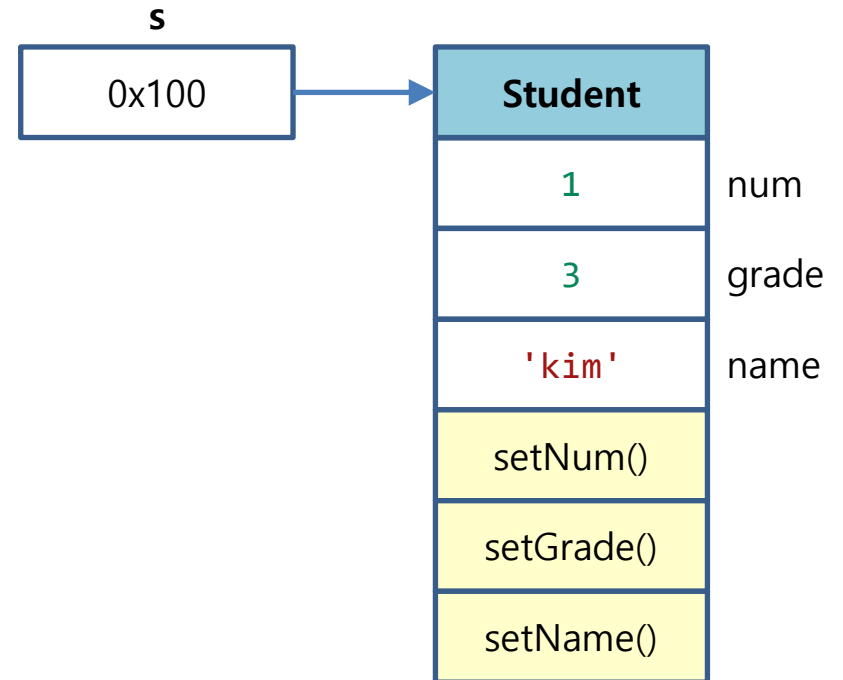
### ● 학생 정보를 저장할 수 있는 클래스

```
class Student:
    def setNum (self, num):
        self.num = num

    def setGrade (self, grade):
        self.grade = grade

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

s = Student()
s.setNum(1)
s.setGrade(3)
s.setName('kim')
```



## ■ 연습문제

- 아래의 코드가 실행되어 결과와 같이 출력될 수 있도록 메소드 작성하기

```
class Book:

    # 코드 작성

book1 = Book()
book1.setTitle('Python')
book1.setPrice(10000)
print(book1.title, book1.price)

book2 = Book()
book2.setTitle('Java')
book2.setPrice(10000)
print(book2.title, book2.price)
```

Python 10000

Java 10000

## ■ 클래스

### ● 생성자를 이용한 초기화

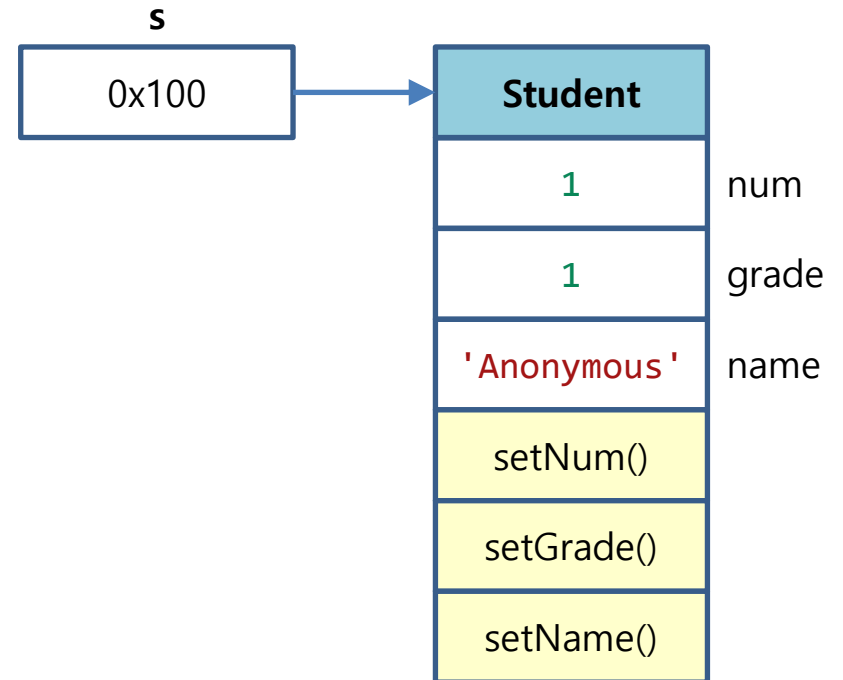
```
class Student:
    def __init__(self):
        self.num = 1
        self.grade = 1
        self.name = 'Anonymous'

    def setNum (self, num):
        self.num = num

    def setGrade (self, grade):
        self.grade = grade

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

s = Student()
print(s.num, s.grade, s.name)
```



## ■ 클래스

### ● 생성자를 이용한 초기화

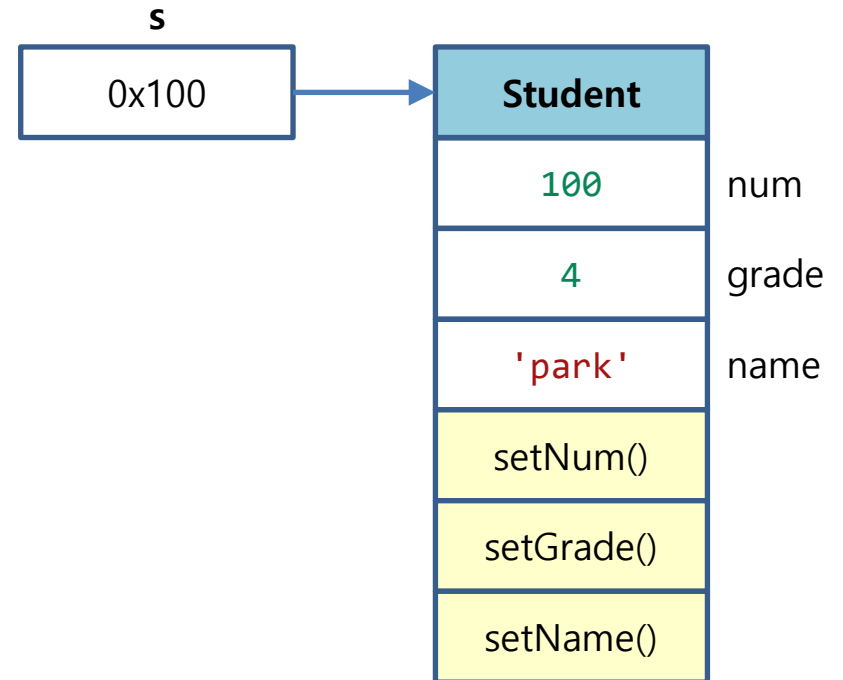
```
class Student:
    def __init__(
        self, num, grade, name):
        self.num = num
        self.grade = grade
        self.name = name

    def setNum (self, num):
        self.num = num

    def setGrade (self, grade):
        self.grade = grade

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

s = Student(100, 4, 'park')
print(s.num, s.grade, s.name)
```





## ■ 연습문제

- 아래의 코드가 실행되어 결과와 같이 출력될 수 있도록 생성자 작성하기

```
class Product:

    # 생성자 작성

    def show_info(self):
        return 'name=%s, code=%s, count=%s' % (
            self.name, self.code, self.count)

p1 = Product('clock', 'c0001', 500)
print(p1.show_info())

p2 = Product('tv', 't0001', 1500)
print(p2.show_info())
```

```
name=clock, code=c0001, count=500
name=tv, code=t0001, count=1500
```

## ■ 연습문제

- 내용을 저장하고 저장된 내용을 읽을 수 있는 Letter 클래스 작성하기

```
class Letter:
    # 생성자 작성, 변수 초기화

    # write 메소드 작성

    # show 메소드 작성

letter1 = Letter()
letter1.write('가나다라')
letter1.write('카타파하')
print(letter1.show())

letter2 = Letter()
letter2.write('ABCD')
letter2.write('WXYZ')
print(letter2.show())
```

가나다라카타파하  
ABCDWXYZ

## ■ 연습문제

- 자동차 옵션을 추가하고 추가된 옵션을 확인할 수 있는 Car 클래스 작성하기

```
class Car:
    # 생성자 작성, 변수 초기화

    # add_option 메소드 작성

    # show_option 메소드 작성

car1 = Car()
car2 = Car()

car1.add_option('전동 트렁크')
car1.add_option('통풍 시트')
print(car1.show_option())

car2.add_option('뒷자리 에어백')
car2.add_option('하이패스')
print(car2.show_option())
```

전동 트렁크, 통풍 시트  
뒷자리 에어백, 하이패스

## ■ 클래스

### ● 상속

#### - 부모 클래스

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

    def speak(self):
        return '%s makes a noise' % self.name
```

#### - 자식 클래스

```
class Dog(Animal):
    pass
```

#### - 사용

```
dog = Dog('ShuShu')
print(dog.speak())
```

```
ShuShu makes a noise
```

## ■ 클래스

### ● 기능 추가

- 부모 클래스에는 없고 자식 클래스만 가지는 메소드 추가

```
class Dog(Animal):  
  
    def run(self):  
        return '%s run' % self.name  
  
dog = Dog('ShuShu')  
print(dog.speak())  
print(dog.run())
```

```
ShuShu makes a noise  
ShuShu run
```

## ■ 클래스

### ● 오버라이드

- 부모 클래스의 메소드를 재정의하여 자식 클래스만의 기능으로 작성

```
class Dog(Animal):  
  
    def speak(self):  
        return '%s 멍멍' % self.name  
  
    def run(self):  
        return '%s run' % self.name  
  
dog = Dog('ShuShu')  
print(dog.speak())  
print(dog.run())
```

ShuShu 멍멍

ShuShu run

## ■ 연습문제

- Shape 클래스를 상속받는 Rectangle, Triangle 클래스 작성하기
  - draw(), getArea() 오버라이드

```
class Shape:
    def __init__(self, width, height, color):
        self.width = width
        self.height = height
        self.color = color

    def draw(self):
        print('draw')

    def getArea(self):
        print(self.width * self.height)
```

# 코드 작성

```
rect = Rectangle(20, 20, 'blue')
rect.draw()
rect.getArea()

tri = Tritangle(20, 20, 'red')
tri.draw()
tri.getArea()
```

```
Draw Rectangle
area => 400

Draw Tritangle
area => 200
```

## ■ 모듈 (module)

- 파이썬 코드를 논리적으로 묶어서 관리하고 사용할 수 있도록 만들어 둔 것
- 보통 py 확장자를 가지는 1개의 파이썬 파일이 하나의 모듈이 됨
- 함수, 클래스, 변수를 정의할 수 있으며, 실행 코드도 포함 가능
- 기본적으로 많은 표준 라이브러리 모듈을 제공하고 있으며,  
필요에 따라서 외부 라이브러리 모듈을 설치하여 사용하는 것도 가능
- import문을 사용하여 1개 이상의 모듈을 불러들임
- 기본 사용
  - import module1, module2, ... , moduleN

```
import random
num = random.randint(1, 10)
print(num)
```



## ■ 모듈 (module)

### ● 모듈 작성 및 사용 (import)

calculator.py

```
def plus(first, second):  
    return first + second  
  
def minus(first, second):  
    return first - second  
  
def multiply(first, second):  
    return first * second  
  
def divide(first, second):  
    return first / second
```

```
import calculator
```

```
result1 = calculator.plus(5, 3)  
print(result1)
```

```
result2 = calculator.minus(5, 3)  
print(result2)
```

```
result3 = calculator.multiply(5, 3)  
print(result3)
```

```
result4 = calculator.divide(5, 3)  
print(result4)
```

```
8  
2  
15  
1.6666666666666667
```

## ■ 모듈 (module)

### ● 모듈 작성 및 사용 (from - import)

calculator.py

```
def plus(first, second):  
    return first + second  
  
def minus(first, second):  
    return first - second  
  
def multiply(first, second):  
    return first * second  
  
def divide(first, second):  
    return first / second
```

```
from calculator \  
    import plus, minus, multiply, divide
```

```
result1 = plus(5, 3)  
print(result1)
```

```
result2 = minus(5, 3)  
print(result2)
```

```
result3 = multiply(5, 3)  
print(result3)
```

```
result4 = divide(5, 3)  
print(result4)
```

## ■ 모듈 (module)

### ● 모듈 작성 및 사용 (alias)

```
from calculator \
    import plus as p, minus as m, \
        multiply as mul, divide as div

result1 = p(5, 3)
print(result1)

result2 = m(5, 3)
print(result2)

result3 = mul(5, 3)
print(result3)

result4 = div(5, 3)
print(result4)
```

```
import calculator as cal

result1 = cal.plus(5, 3)
print(result1)

result2 = cal.minus(5, 3)
print(result2)

result3 = cal.multiply(5, 3)
print(result3)

result4 = cal.divide(5, 3)
print(result4)
```

## ■ 모듈 (module)

- import 될 때 작성된 코드가 실행되므로 현재 실행상태가 main의 역할인지 모듈로써 실행되는지 판단하여 처리
- `__name__` 속성
  - main 실행 : `'__main__'` 문자열 반환
  - module로 import : `'파일명'(모듈명)` 문자열 반환

```
def plus(first, second):  
    return first + second  
  
def minus(first, second):  
    return first - second  
  
def multiply(first, second):  
    return first * second  
  
def divide(first, second):  
    return first / second  
  
print(__name__)  
print(plus(10, 20))
```

메인으로 사용

```
__main__  
30
```

모듈로 사용

```
calculator  
30
```

## ■ 모듈 (module)

### ● `__name__` 속성을 이용한 코드 처리

calculator.py

```
def plus(first, second):  
    return first + second  
  
def minus(first, second):  
    return first - second  
  
def multiply(first, second):  
    return first * second  
  
def divide(first, second):  
    return first / second  
  
if __name__ == '__main__':  
    print(__name__)  
    print(plus(10, 20))
```

## ■ 연습문제

- 결과와 같이 동작될 수 있도록 mymod 모듈 작성하고 사용하기

mymod.py

```
# myfunc 함수 작성
```

08-클래스.ipynb

```
# myfunc 모듈 가져오기
```

```
result = myfunc(1, 2)  
print('myfunc 실행결과 %s' % result)
```

결과

```
myfunc 실행결과 -1
```

## ■ 연습문제

- 결과와 같이 동작될 수 있도록 pw\_encoder 모듈 작성하고 사용하기

pw\_encoder.py

```
# get_pw 함수 작성

# import hashlib
# pw = '1234'
# pw_to_bytes = bytes(pw, 'utf8')
# m = hashlib.sha256()
# m.update( pw_to_bytes )
# print( m.hexdigest() )
```

08-클래스.ipynb

```
# pw_encoder 모듈 가져오기

raw_password = '1234'
encoded_password = get_pw(raw_password)
print('%s => %s' % (raw_password, encoded_password))
```

결과

```
1234 => 03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4
```

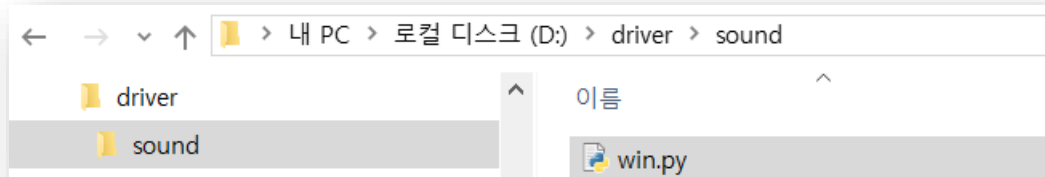
## ■ 패키지 (Package)

- 모듈을 계층(디렉토리) 구조로 관리하는 것

ex) 모듈명이 driver.sound.win 인 경우

패키지 : driver, sound

모듈 : win



- 디렉토리 내에 `__init__.py` 라는 파일을 사용하여  
패키지 내의 모든 모듈을 import 하는 것도 가능



## ■ 패키지 (Package)

### ● driver 패키지 하위 모듈 import

#### 패키지 구조

```
driver/  
  sound/  
    mp3.py  
    wav.py  
  vga/  
    ati.py  
    nvidia.py
```

#### driver/sound/mp3.py

```
print('sound mp3!')  
  
def sound():  
    print('mp3~~')
```

#### driver/sound/wav.py

```
print('sound wav!')  
  
def sound():  
    print('wav~~')
```

#### driver/vga/ati.py

```
print('vga ati!')
```

#### driver/vga/nvidia.py

```
print('vga nvidia!')
```

#### 모듈 사용

```
import driver.sound.mp3  
from driver.sound.wav import *  
import driver.vga.ati  
from driver.vga.nvidia import *  
  
print('----')  
  
driver.sound.mp3.sound()  
sound() # driver.sound.wav
```

```
sound mp3!  
sound wav!  
vga ati!  
vga nvidia!  
----  
mp3~~  
wav~~
```

## ■ 패키지 (Package)

### ● driver 패키지 하위 모듈 import

- `__init__.py` 를 사용하면 패키지 내 모든 모듈 가져오기 가능

#### 패키지 내 모든 모듈 가져오기 불가 (기본)

```
from driver.sound import *  
from driver.vga import *  
  
mp3.sound()  
wav.sound()
```



```
-----  
NameError  
d:\study\python\openeg\08-클래스.ipynb  
   1 from driver.sound import *  
   2 from driver.vga import *  
----> 4 mp3.sound()  
      5 wav.sound()  
  
NameError: name 'mp3' is not defined
```

#### 패키지 내 모든 모듈 가져오기 가능 (`__init__.py` 사용)

```
driver/  
  sound/  
    __init__.py  
  vga/  
    __init__.py
```

```
# sound/__init__.py  
__all__ = ['mp3', 'wav']
```

```
# vga/__init__.py  
__all__ = ['ati', 'nvidia']
```



```
sound mp3!  
sound wav!  
vga ati!  
vga nvidia!  
mp3~~  
wav~~
```