■ 클래스

- 객체지향의 가장 기본적 개념
- 관련된 속성(모습)과 동작(기능)을 하나의 범주로 묶어 실세계의 사물을 표현
- 모델링 : 사물을 분석하여 필요한 속성과 동작 추출
- 캡슐화 : 모델 결과를 클래스로 포장(표현)
- 멤버 : 클래스를 구성하는 변수와 함수
- 메서드 : 클래스에 소속된 함수
- 클래스의 변수를 활용하려면 반드시 self 선언
- 메서드 자체의 코드만으로 동작할 때는
  self 인수의 정보가 없어도 상관없음

```
class 클래스:
    변수 = 10

    def 함수1():
        함수내변수 = 20
        print('함수1', 함수내변수)

    def 함수2(self):
        print('함수2', self.변수)

객체 = 클래스()
print(객체.변수)
클래스.함수1()
객체.함수2()
```

인수를 명시하지 않음
자동으로 메소드로 전달

```
10
함수1 20
함수2 10
```

■ 클래스

● self 사용

```
class 클래스:
    def 함수(self):
        print('실행')

객체 = 클래스()
객체.함수()
```

실행

● self 미사용

```
class 클래스:
    def 함수():
        print('실행')

클래스.함수()
```

실행

● 사용불가

```
class 클래스:
    def 함수():
        print('실행')

객체 = 클래스()
객체.함수()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback
<ipython-input-27-c5a031d0cc14> in <moc
      4
      5 객체 = 클래스()
----> 6 객체.함수()
```

```
class 클래스:
    def 함수(self):
        print('실행')

클래스.함수()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback
<ipython-input-26-259df5fd5ca7> in <moc
      3         print('실행')
      4
----> 5 클래스.함수()
```

■ 클래스

● 함수를 이용한 계산 기능

- 결과값 누적

```
result = 0  # 결과를 저장할 전역변수

def adder(num):
    global result  # 전역변수 사용 설정
    result += num
    return result  # 결과값 반환

print(adder(3))
print(adder(10))
```

```
3
13
```

■ 클래스

● 함수를 이용한 계산 기능

- 2개의 결과값 누적

```
result1 = 0  # 결과를 저장할 전역변수
result2 = 0  # 결과를 저장할 전역변수

def adder1(num):
    global result1  # 전역변수 사용 설정
    result1 += num
    return result1  # 결과값 반환

def adder2(num):
    global result2  # 전역변수 사용 설정
    result2 += num
    return result2  # 결과값 반환

print(adder1(3))
print(adder1(10))

print(adder2(13))
print(adder2(100))
```

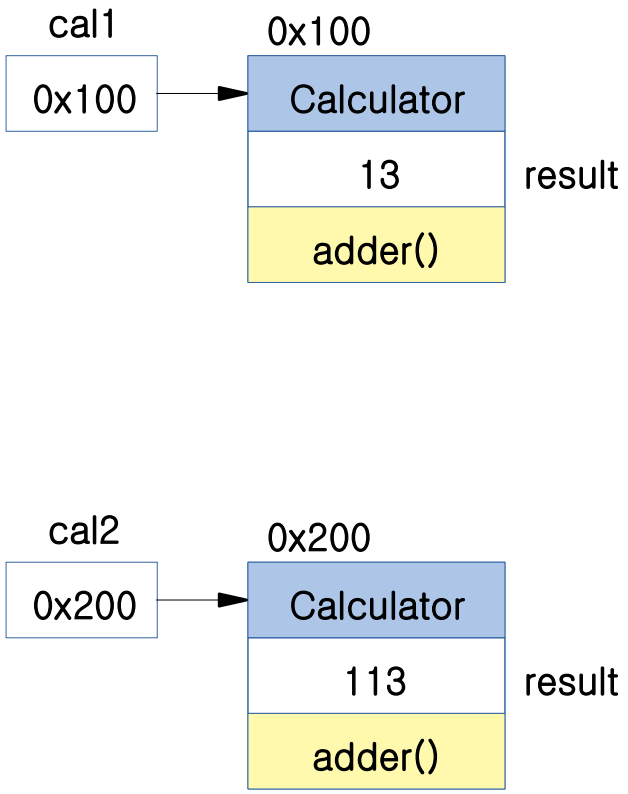
```
3
13

13
113
```

■ 클래스

● 클래스를 이용한 계산 기능

```
class Calculator:
    result = 0

    def adder(self, num):
        self.result += num
        return self.result

cal1 = Calculator()  # 객체 생성
cal2 = Calculator()  # 객체 생성

result1 = cal1.adder(3)  # 3
result1 = cal1.adder(10)  # 13

result2 = cal2.adder(13)  # 13
result2 = cal2.adder(100)  # 113

print(result1)

print(result2)
```

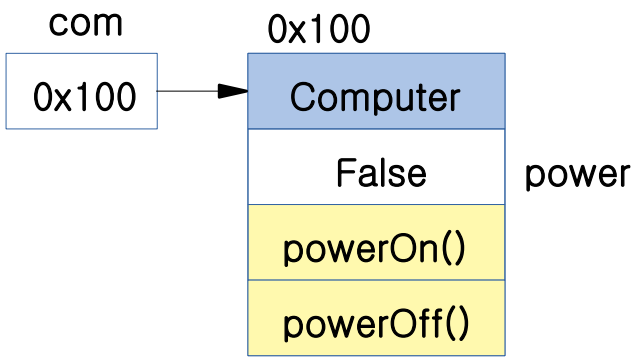
```
13
113
```



■ 클래스

- 클래스(Class) : 설계도면
- 객체(Object) : 클래스(설계도면)에 의해 만들어진 완성물(제품)
- 인스턴스(Instance) : 객체와 같은 의미
  - com은 객체
  - com은 Computer 클래스의 인스턴스 (관계를 나타내는 의미로 사용할 때)

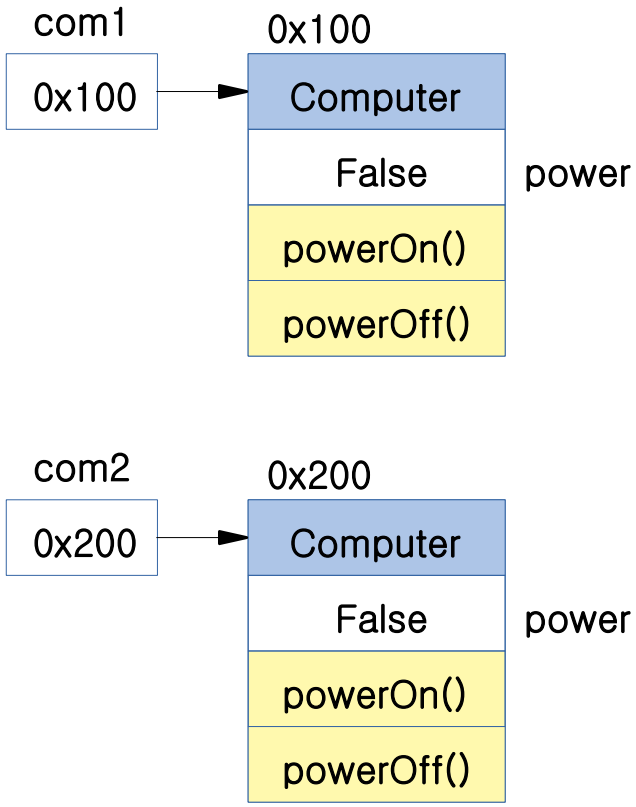
```
class Computer:
    power = False

    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

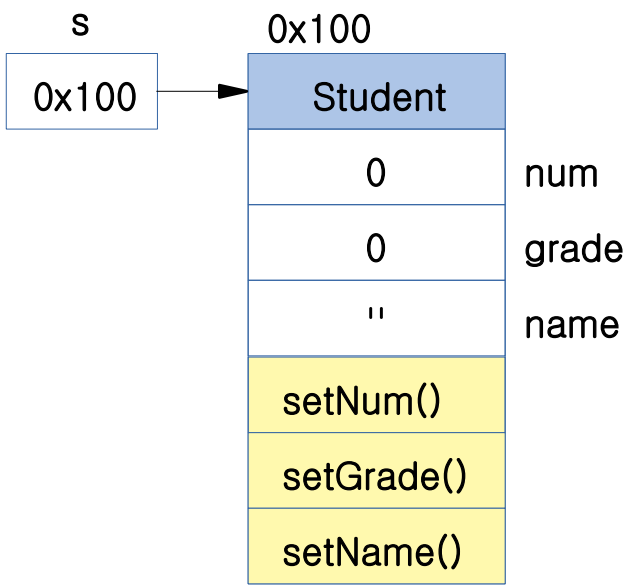
```
class Computer:
    power = False

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

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

com1 = Computer()
com2 = Computer()
```



■ 클래스

● 학생 정보를 저장할 수 있는 클래스

```
class Student:
    num = 0  # 번호, 생략가능
    grade = 0  # 학년, 생략가능
    name = ''  #이름, 생략가능

    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')
```




```
s = Student()
s.setNum(1)
```

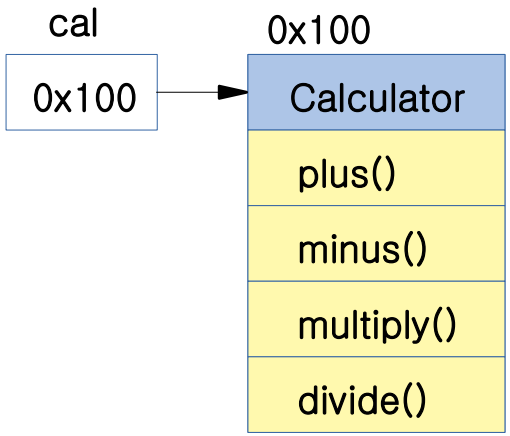
```
def setNum(self, num):
    self.num = num
```

■ 클래스

● 계산기 사칙연산 기능 추가

```
class Calculator:
    def plus(self, first, second):
        return first + second

    def minus(self, first, second):
        return first - second

    def multiply(self, first, second):
        return first * second

    def divide(self, first, second):
        return first / second

cal = Calculator()
print(cal.plus(5, 3))
print(cal.minus(5, 3))
print(cal.multiply(5, 3))
print(cal.divide(5, 3))
```

```
8
2
15
1.6666666666666667
```

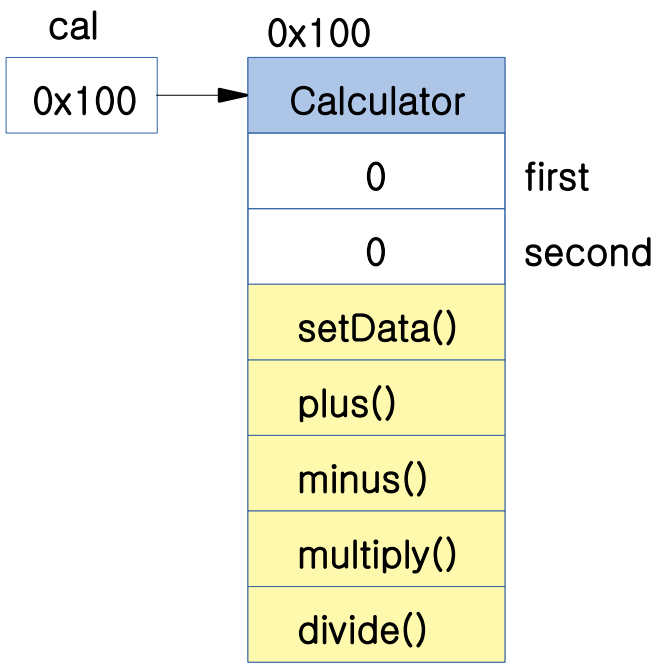


■ 클래스

● 계산할 값을 미리 저장할 수 있도록 수정

```
class Calculator:
    def setData(self, first, second):
        self.first = first
        self.second = second

    def plus(self):
        return self.first + self.second

    def minus(self):
        return self.first - self.second

    def multiply(self):
        return self.first * self.second

    def divide(self):
        return self.first / self.second

cal = Calculator()
cal.setData(5, 3)
print(cal.plus())
print(cal.minus())
print(cal.multiply())
print(cal.divide())
```

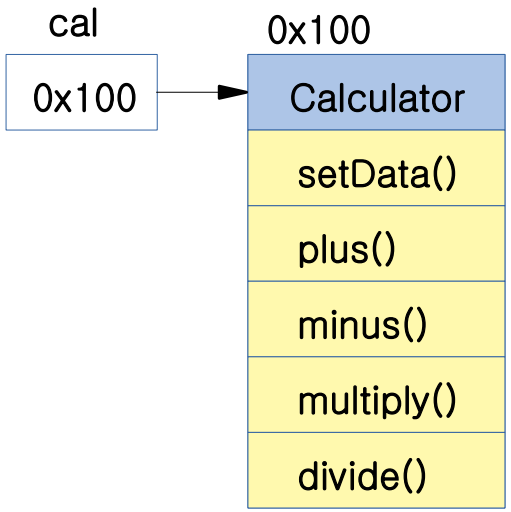
```
8
2
15
1.6666666666666667
```



■ 클래스

● 계산할 값을 미리 저장할 수 있도록 수정

```
class Calculator:
    def setData(self, first, second):
        self.first = first
        self.second = second

    def plus(self):
        return self.first + self.second

    def minus(self):
        return self.first - self.second

    def multiply(self):
        return self.first * self.second

    def divide(self):
        return self.first / self.second

cal = Calculator()
# cal.setData(5, 3)  # 미호출 시 오류
print(cal.plus())
print(cal.minus())
print(cal.multiply())
print(cal.divide())
```

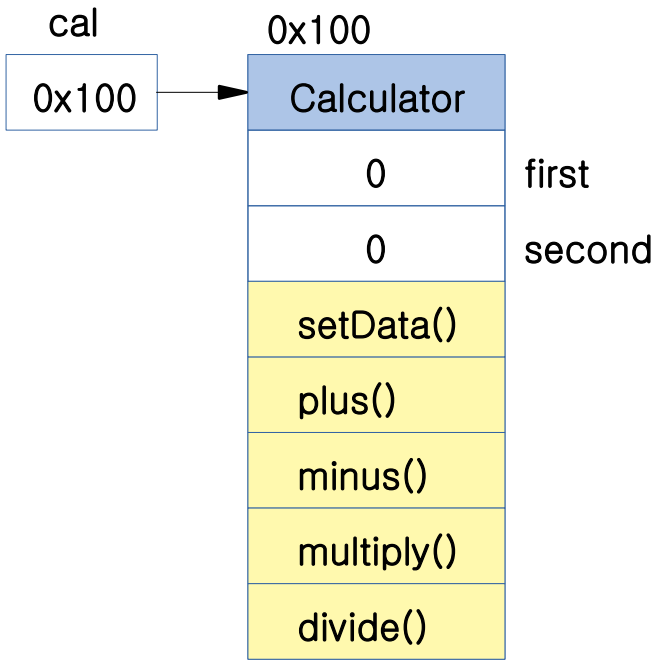


setData 메소드 미호출 시
first, second 변수가 생성되지 않음

■ 클래스

● 생성자를 이용한 값 초기화

```
class Calculator:
    def __init__(self, first, second):
        self.first = first
        self.second = second

    def plus(self):
        return self.first + self.second

    def minus(self):
        return self.first - self.second

    def multiply(self):
        return self.first * self.second

    def divide(self):
        return self.first / self.second

cal = Calculator(5, 3)

print(cal.plus())
print(cal.minus())
print(cal.multiply())
print(cal.divide())
```



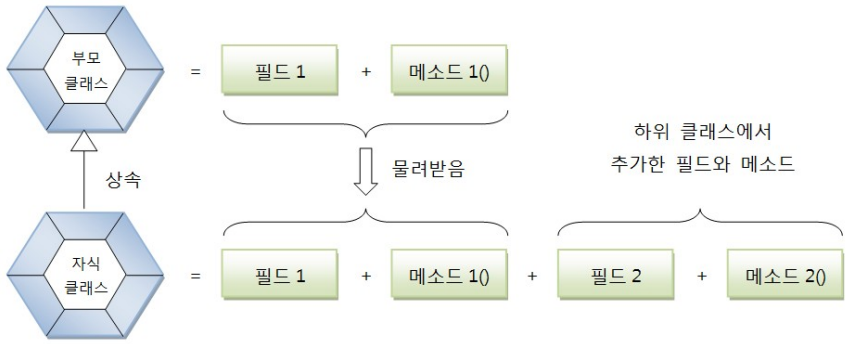
setData 메소드 사용 대신 생성자를 이용하여 first, second 변수를 생성

■ 클래스

● 상속

```
class Calculator:
    def __init__(self, first, second):
        self.first = first
        self.second = second

    def plus(self):
        return self.first + self.second

    def minus(self):
        return self.first - self.second

    def multiply(self):
        return self.first * self.second

    def divide(self):
        return self.first / self.second
```

```
class EngineeringCalculator(Calculator):
    pass
```

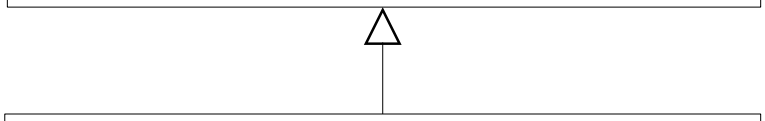


first, second 변수
plus(), minus(), multiply(), divide() 메소드

■ 클래스

● 상속

```
class Calculator:
    def __init__(self, first, second):
        self.first = first
        self.second = second

    def plus(self):
        return self.first + self.second

    def minus(self):
        return self.first - self.second

    def multiply(self):
        return self.first * self.second

    def divide(self):
        return self.first / self.second
```



```
class EngineeringCalculator(Calculator):
    pass
```

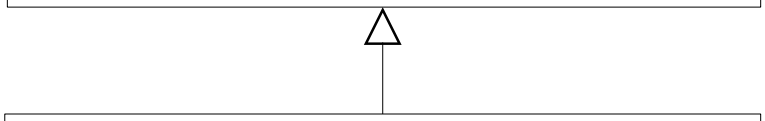
```
cal = EngineeringCalculator(5, 3)
print(cal.plus())
print(cal.minus())
print(cal.multiply())
print(cal.divide())
```

```
8
2
15
1.6666666666666667
```

■ 클래스

● 상속 후 기능(메소드) 추가

```
class Calculator:
    def __init__(self, first, second):
        self.first = first
        self.second = second

    def plus(self):
        return self.first + self.second

    def minus(self):
        return self.first - self.second

    def multiply(self):
        return self.first * self.second

    def divide(self):
        return self.first / self.second
```



```
class EngineeringCalculator(Calculator):
    def pow(self):
        return self.first ** self.second
```

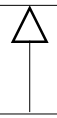
```
cal = EngineeringCalculator(5, 3)
print(cal.plus())
print(cal.minus())
print(cal.multiply())
print(cal.divide())
print(cal.pow())
```

```
8
2
15
1.6666666666666667
125
```

■ 클래스

● 오버라이딩 (Overriding)

```
class Calculator:
    def __init__(self, first, second):
        self.first = first
        self.second = second

    def plus(self):
        return self.first + self.second

    def minus(self):
        return self.first - self.second

    def multiply(self):
        return self.first * self.second

    def divide(self):
        return self.first / self.second
```



```
class EngineeringCalculator(Calculator):
    def pow(self):
        return self.first ** self.second

    def divide(self):
        return '%0.2f' % float(self.first / self.second)
```

```
cal = EngineeringCalculator(5, 3)
print(cal.plus())
print(cal.minus())
print(cal.multiply())
print(cal.divide())
print(cal.pow())
```

```
8
2
15
1.67
125
```

■ 클래스

● 클래스 활용

순번	이름	나이	성적
1	김길동	20	A
2	홍길동	25	B

- 위와 같은 데이터에서 각 학생별 나이를 출력
- 각 데이터를 특정 마크업 문자를 이용하여 표현 후

  string 관련 함수 split()을 사용하여 분리

```
data1 = '1|김길동|20|A'
words1 = data1.split('|')
age1 = words1[2]

data2 = '2|홍길동|25|B'
words2 = data2.split('|')
age2 = words2[2]
```

■ 클래스

● 클래스 활용

```
class Data:
    def __init__(self, data):
        words = data.split('|')
        self.idx = words[0]
        self.name = words[1]
        self.age = words[2]
        self.grade = words[3]

    def print_age(self):
        print('%s의 나이는 %s' % (self.name, self.age))

data1 = Data('1|김길동|20|A')
age1 = data1.age;
print(age1)

data2 = Data('2|홍길동|25|B')
Data2.print_age()
```

```
20
홍길동의 나이는 25
```

■ 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)

● Calculator 모듈 작성 – 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
```

● Calculator 모듈 활용 – ModuleTest.py

```
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)

● Calculator 모듈 활용 – 함수 명시

```
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)
```

```
from Calculator import *

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)

● Calculator 모듈 활용 – 함수 명시, alias 지정

```
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)
```

■ 모듈 (module)

● Calculator.py 코드 추가

```
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))
```

```
Calculator
30
8
2
15
1.6666666666666667
```

● ModuleTest.py 실행

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

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

...
```

■ 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))
```

Calculator 에서 실행

```
__main__
30
```

ModuleTest 에서 실행

```
Calculator
30
```

■ 모듈 (module)

● __name__ 속성을 이용한 코드 처리

```
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))
```

■ 모듈 (module)

● 모듈 내부에서 클래스 사용 – CalculatorClass.py

```
PI = 3.141

class Calculator:
    PI = 3.142

    def plus(self, first, second):
        return first + second

    def minus(self, first, second):
        return first - second

    def multiply(self, first, second):
        return first * second

    def divide(self, first, second):
        return first / second

def average():
    print('평균', 100)
```

```
import CalculatorClass  # module

cal = CalculatorClass.Calculator()  # module 클래스 생성
print(cal.PI)  # 클래스 변수
print(cal.plus(23, 24))  # 클래스 메소드

CalculatorClass.average()  # module 함수

print(CalculatorClass.PI)  # module 변수
```

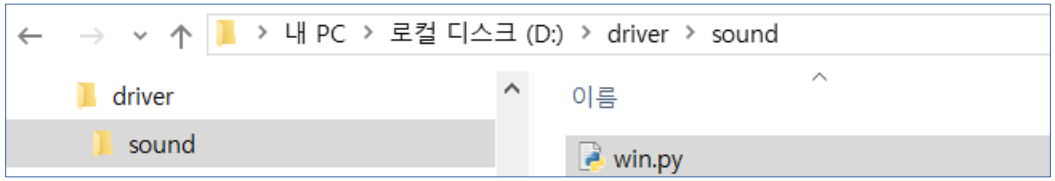
■ 패키지 (Package)

● 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

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

driver/sound/mp3.py

```
print('sound mp3!')
```

driver/sound/wav.py

```
print('sound wav!')
```

driver/vga/ati.py

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

driver/vga/nvidia.py

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

package.py

```
import driver.sound.mp3
from driver.sound.wav import *


import driver.vga.ati
from driver.vga.nvidia import *
```

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

■ 패키지 (Package)

● 패키지를 이용한 import 시 주의사항

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

상위 패키지 지정

```
import driver  # (X)
```

패키지 내 모든 모듈

```
import driver.sound  # (X)
import driver.sound.*  # (X)
```

__init_.py 사용 시 패키지 내 모든 모듈 import 가능

```
__all__ = ['mp3', 'wav']
```

```
__all__ = ['ati', 'nvidia']
```

```
from driver.sound import *

from driver.vga import *
```

```
sound mp3!
sound wav!

vga ati!
vga nvidia!
```