



---

Python

# 클래스/모듈

---

참고서적 : Do it! 점프 투 파이썬 | 박응용지음 | 이지스퍼블리싱

김선녕(sykim.lecture@gmail.com)

---

# 클래스 모듈



```
In [1]: # 클래스 변수 : secret
class Service:
    secret = "영구는 배꼽이 두 개다."
```

```
In [2]: pey = Service()
pey.secret
```

Out [2]: '영구는 배꼽이 두 개다.'

```
In [6]: # 클래스 함수 : sum()
class Service:
    secret = "영구는 배꼽이 두 개다."
    def sum(self, a, b):
        result = a + b
        print("%s + %s = %s입니다." % (a, b, result))
```

```
In [9]: pey = Service()
pey.sum(1,1)

1 + 1 = 2입니다.
```

```
In [11]: # pey.sum(pey, 1, 1) : self는 호출시 이용했던 인스턴스(pey)로 바뀐다.
Service.sum(pey, 1,1)

1 + 1 = 2입니다.
```

```
In [11]: # pey.sum(pey, 1, 1) : self는 호출시 이용했던 인스턴스(pey)로 바뀐다.  
Service.sum(pey, 1, 1)
```

1 + 1 = 2입니다.

```
In [13]: # pey라는 아이디와 홍길동이라는 이름을 연결해 주는 것이 바로 self이다.  
class Service:  
    secret = "영구는 배꼽이 두 개다."  
    def setname(self, name):  
        self.name = name  
    def sum(self, a, b):  
        result = a + b  
        print("%s님 %s + %s = %s입니다." % (self.name, a, b, result))
```

```
In [14]: pey = Service()  
pey.setname("홍길동")  
pey.sum(1, 1)
```

홍길동님 1 + 1 = 2입니다.

- 클래스 내에 정의된 self는 클래스 인스턴스
- self 변수는 클래스 함수의 첫 번째 인수로 받는다

```
In [15]: pey = Service()
         pey.sum(1,1)
```

```
AttributeError                                Traceback (most recent call last)
<ipython-input-15-83dafb30d622> in <module>()
      1 pey = Service()
--> 2 pey.sum(1,1)

<ipython-input-13-a738c36ea69e> in sum(self, a, b)
      5     def sum(self, a, b):
      6         result = a + b
--> 7         print("%s님 %s + %s = %s입니다." %(self.name,a, b, result))
```

AttributeError: 'Service' object has no attribute 'name'

```
In [16]: class Service:
         secret = "연구는 배꼽이 두 개다."
         def __init__(self, name):
             self.name = name
         def setname(self, name):
             self.name = name
         def sum(self, a, b):
             result = a + b
             print("%s님 %s + %s = %s입니다." %(self.name,a, b, result))
```

```
In [17]: # __init__ : 인스턴스를 만들때 항상 실행된다.
         pey = Service("홍길동")
         pey.sum(1,1)
```

홍길동님 1 + 1 = 2입니다.

```
class 클래스이름[(상속 클래스명)]:  
    <클래스 변수 1>  
    <클래스 변수 2>  
    ...  
    <클래스 변수 N>  
  
    def 클래스함수1(self[, 인수1, 인수2,...]):  
        <수행할 문장 1>  
        <수행할 문장 2>  
        ...  
    def 클래스함수2(self[, 인수1, 인수2,...]):  
        <수행할 문장1>  
        <수행할 문장2>  
        ...  
    def 클래스함수N(self[, 인수1, 인수2,...]):  
        <수행할 문장1>  
        <수행할 문장2>  
        ...
```

# 사칙연산 클래스 만들기

7

```
1 class FourCal:
2     pass # 아무것도 수행하지 않는 문법. 임시로 코드를 작성할 때 주로 사용
```

```
1 a = FourCal()
2 type(a) # 객체의 타입을 출력
```

\_\_main\_\_.FourCal

```
1 class FourCal:
2     def setdata(self, first, second):
3         self.first = first
4         self.second = second
```

```
1 a = FourCal()
2 a.setdata(4,2)
3 print(a.first)
```

4

```
1 print(a.second)
```

2

```
1 b = FourCal()
2 b.setdata(3,7)
3 print(b.first)
```

3

```
1 print(b.second)
```

7

# 사칙연산 클래스 만들기

8

```
1 class FourCal:
2     def setdata(self, first, second):
3         self.first = first
4         self.second = second
5     def sum(self):
6         result = self.first + self.second
7         return result
8     def mul(self):
9         result = self.first * self.second
10        return result
11    def sub(self):
12        result = self.first - self.second
13        return result
14    def div(self):
15        result = self.first / self.second
16        return result
```

```
1 a = FourCal()
2 b = FourCal()
3 a.setdata(4,2)
4 b.setdata(3,7)
5 a.sum()
```

6

```
1 a.mul()
```

8

```
1 a.sub()
```

2

```
1 a.div()
```

2.0

```
1 b.sum()
```

10

```
1 b.mul()
```

21

```
1 b.sub()
```

-4

```
1 b.div()
```

0.42857142857142855



# HousePark 클래스 만들기

```
In [39]: class HousePark:
         lastname = "박"
```

```
In [40]: pey = HousePark()
         pex = HousePark()
         print(pey.lastname)
```

박

```
In [41]: print(pex.lastname)
```

박

```
In [42]: class HousePark:
         lastname = "박"
         def setname(self, name):
             self.fullname = self.lastname + name
```

```
In [43]: pey = HousePark()
         pey.setname("응용")
         print(pey.fullname)
```

박응용

## HousePark 클래스 만들기

10

```
In [44]: class HousePark:
          lastname = "박"
          def setname(self, name):
              self.fullname = self.lastname + name
          def travel(self, where):
              print("%s, %s여행을 가다." %(self.fullname, where))
```

```
In [45]: pey = HousePark()
          pey.setname("응용")
          pey.travel("부산")
```

박응용, 부산여행을 가다.

```
In [46]: pey = HousePark()
          pey.travel("부산")
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-46-2574e615b80b> in <module>()
      1 pey = HousePark()
----> 2 pey.travel("부산")

<ipython-input-44-04d8a13e6bff> in travel(self, where)
      4     self.fullname = self.lastname + name
      5     def travel(self, where):
----> 6         print("%s, %s여행을 가다." %(self.fullname, where))
```

**AttributeError:** 'HousePark' object has no attribute 'fullname'

```
In [47]: class HousePark:
          lastname = "박"
          def __init__(self, name):
              self.fullname = self.lastname + name
          def travel(self, where):
              print("%s, %s여행을 가다." % (self.fullname, where))
```

```
In [48]: pey = HousePark()
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-48-1fc05c75a068> in <module>()
----> 1 pey = HousePark()
```

```
TypeError: __init__() missing 1 required positional argument: 'name'
```

```
In [49]: pey = HousePark("응용")
          pey.travel("태국")
```

박응용, 태국여행을 가다.

```
In [50]: # 클래스의 상속
class HouseKim(HousePark):
    lastname = "김"
```

```
In [51]: juliet = HouseKim("줄리엣")
juliet.travel("독도")
```

김줄리엣, 독도여행을 가다.

```
In [52]: # 메서드 오버라이딩
class HouseKim(HousePark):
    lastname = "김"
    def travel(self, where, day):
        print("%s, %s여행을 %d일 가다." %(self.fullname, where, day))
```

```
In [53]: juliet = HouseKim("줄리엣")
juliet.travel("독도",3)
```

김줄리엣, 독도여행을 3일 가다.

In [54]: # 연산자 오버로딩 : 연산자(+,-,\*,/..)를 객체끼리 사용할 수 있게 하는 기법

```
class HousePark:
    lastname = "박"
    def __init__(self, name):
        self.fullname = self.lastname + name
    def travel(self, where):
        print("%s, %s여행을 가다." % (self.fullname, where))
    def love(self, other):
        print("%s, %s 사랑에 빠졌네" % (self.fullname, other.fullname))
    def __add__(self, other):
        print("%s, %s 결혼했네" % (self.fullname, other.fullname))

class HouseKim(HousePark):
    lastname = "김"
    def travel(self, where, day):
        print("%s, %s여행을 %d일 가다." % (self.fullname, where, day))

pey = HousePark("응용")
juliet = HouseKim("줄리엣")
pey.love(juliet)
pey + juliet # __add__ 함수가 호출(객체끼리 더한다)
```

박응용, 김줄리엣 사랑에 빠졌네

박응용, 김줄리엣 결혼했네

```
In [57]: class HousePark:
    lastname = "박"
    def __init__(self, name):
        self.fullName = self.lastname + name
    def travel(self, where):
        print("%s, %s여행을 가다." % (self.fullName, where))
    def love(self, other):
        print("%s, %s 사랑에 빠졌네" % (self.fullName, other.fullName))
    def fight(self, other):
        print("%s, %s 싸우네" % (self.fullName, other.fullName))
    def __add__(self, other):
        print("%s, %s 결혼했네" % (self.fullName, other.fullName))
    def __sub__(self, other):
        print("%s, %s 이혼했네" % (self.fullName, other.fullName))

    pey = HousePark("응용")
    juliet = HouseKim("줄리엣")
    pey.travel("부산")
    juliet.travel("부산",3)
    pey.love(juliet)
    pey + juliet
    pey.fight(juliet)
    pey - juliet
```

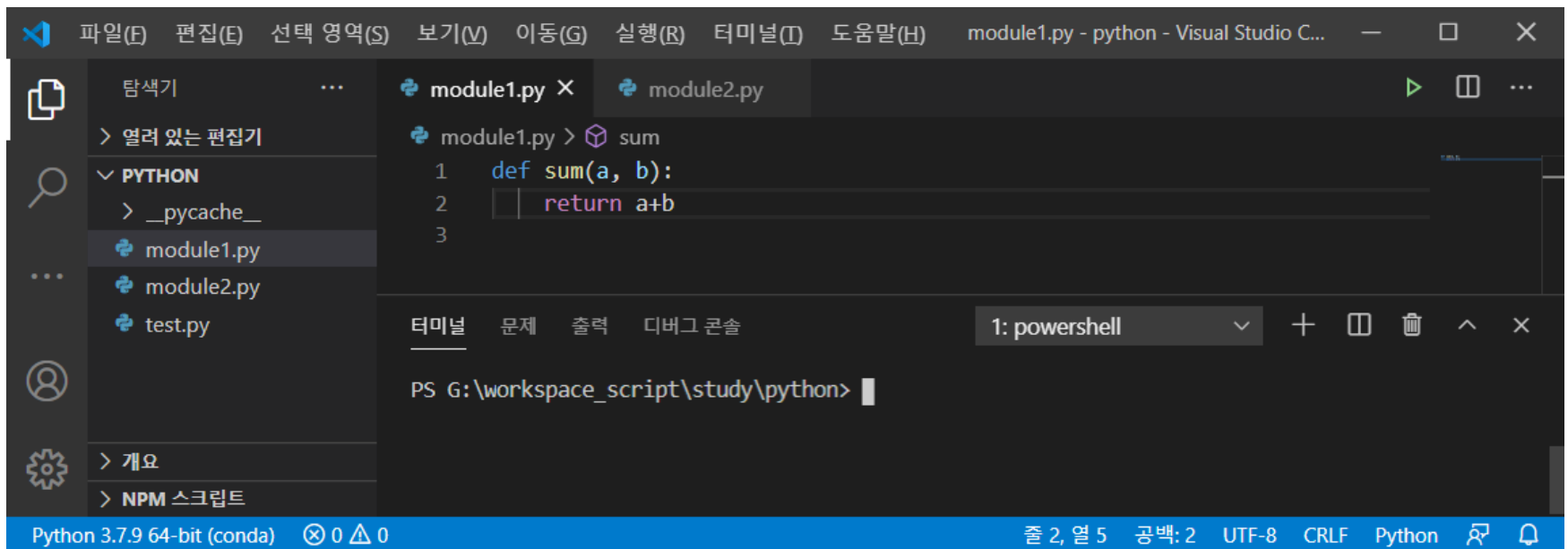
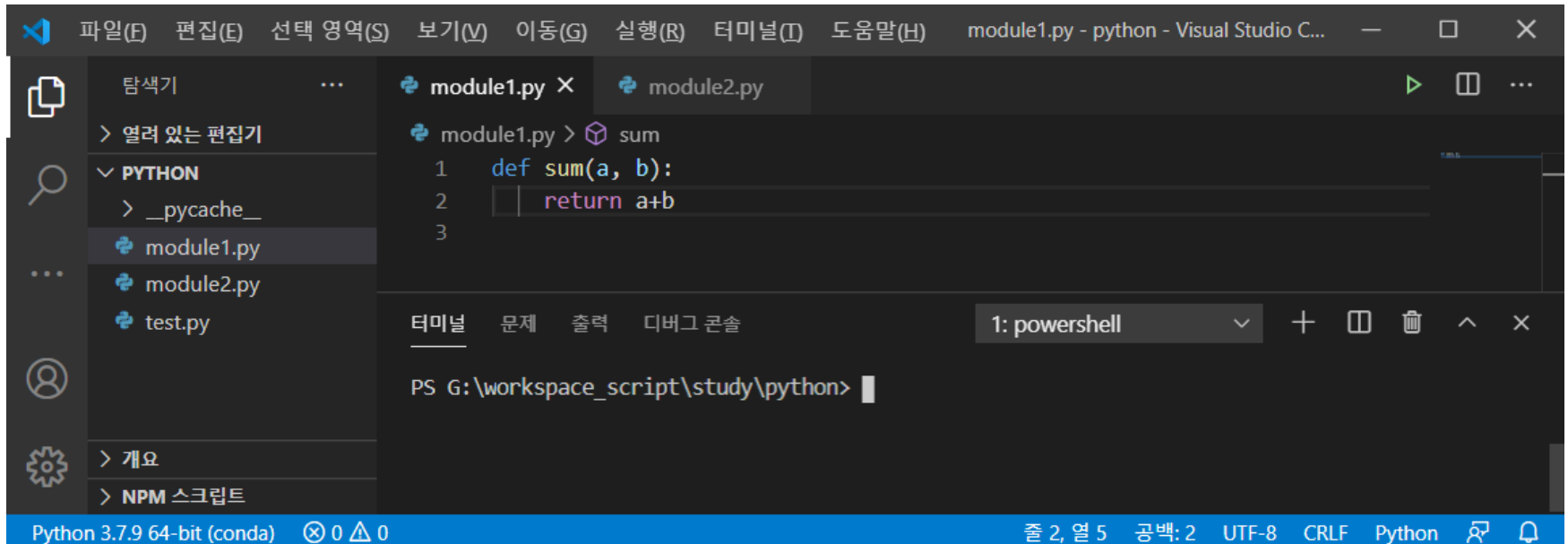
박응용, 부산여행을 가다.  
김줄리엣, 부산여행을 3일 가다.  
박응용, 김줄리엣 사랑에 빠졌네  
박응용, 김줄리엣 결혼했네  
박응용, 김줄리엣 싸우네  
박응용, 김줄리엣 이혼했네

# 클래스 모듈



# 모듈 import

16





# 모듈에 함수추가

17

The image consists of two screenshots of the Visual Studio Code interface, demonstrating how to create and use a Python module.

**Top Screenshot:** The editor shows `module1.py` with the following code:

```
1 def sum(a, b):
2     return a+b
3
4
5 def safe_sum(a, b):
6     if type(a) != type(b):
7         print("더할 수 있는 것이 아닙니다.")
8         return
9     else:
10        result = sum(a, b)
11        return result
12
```

The file explorer on the left shows a project structure with `__pycache__`, `module1.py`, `module2.py`, and `test.py`.

**Bottom Screenshot:** The editor shows `module2.py` with the following code:

```
1 import module1
2 print(module1.sum(3, 4))
3 print(module1.safe_sum(3, 4))
4 print(module1.safe_sum(20, 30))
```

The terminal at the bottom shows the command `python module2.py` being executed, resulting in the output:

```
7
7
50
```

The status bar at the bottom indicates the environment is `Python 3.7.9 64-bit (conda)` and the file encoding is `UTF-8`.

## sum, safe\_sum 함수처럼 쓰고 싶은 경우

18

```
1 from module1 import sum
2 sum(3,4)
```

```
1 from module1 import sum, safe_sum
2 safe_sum(3,4)
```

```
1 from module1 import *
```

# import 수행시 모듈이 실행?

19

The screenshot displays the Visual Studio Code interface with three Python files open: `module1.py`, `module2.py`, and `module3.py`. The `module1.py` file contains the following code:

```
2 return a+b
3
4
5 def safe_sum(a, b):
6     if type(a) != type(b):
7         print("더할 수 있는 것이 아닙니다.")
8         return
9     else:
10        result = sum(a, b)
11        return result
12
13
14 print(safe_sum('a', 4))
15 print(safe_sum(1, 4))
16 print(sum(10, 10.4))
17 print(__name__)
18
```

The terminal output for `python module1.py` is as follows:

```
PS G:\workspace_script\study\python> python module1.py
더할 수 있는 것이 아닙니다.
None
5
20.4
main
PS G:\workspace_script\study\python>
```

The `main` output is highlighted with a yellow dashed box.

The `module3.py` file contains the following code:

```
1 import module1
2
```

The terminal output for `python module3.py` is as follows:

```
PS G:\workspace_script\study\python> python module3.py
더할 수 있는 것이 아닙니다.
None
5
20.4
module1
PS G:\workspace_script\study\python>
```

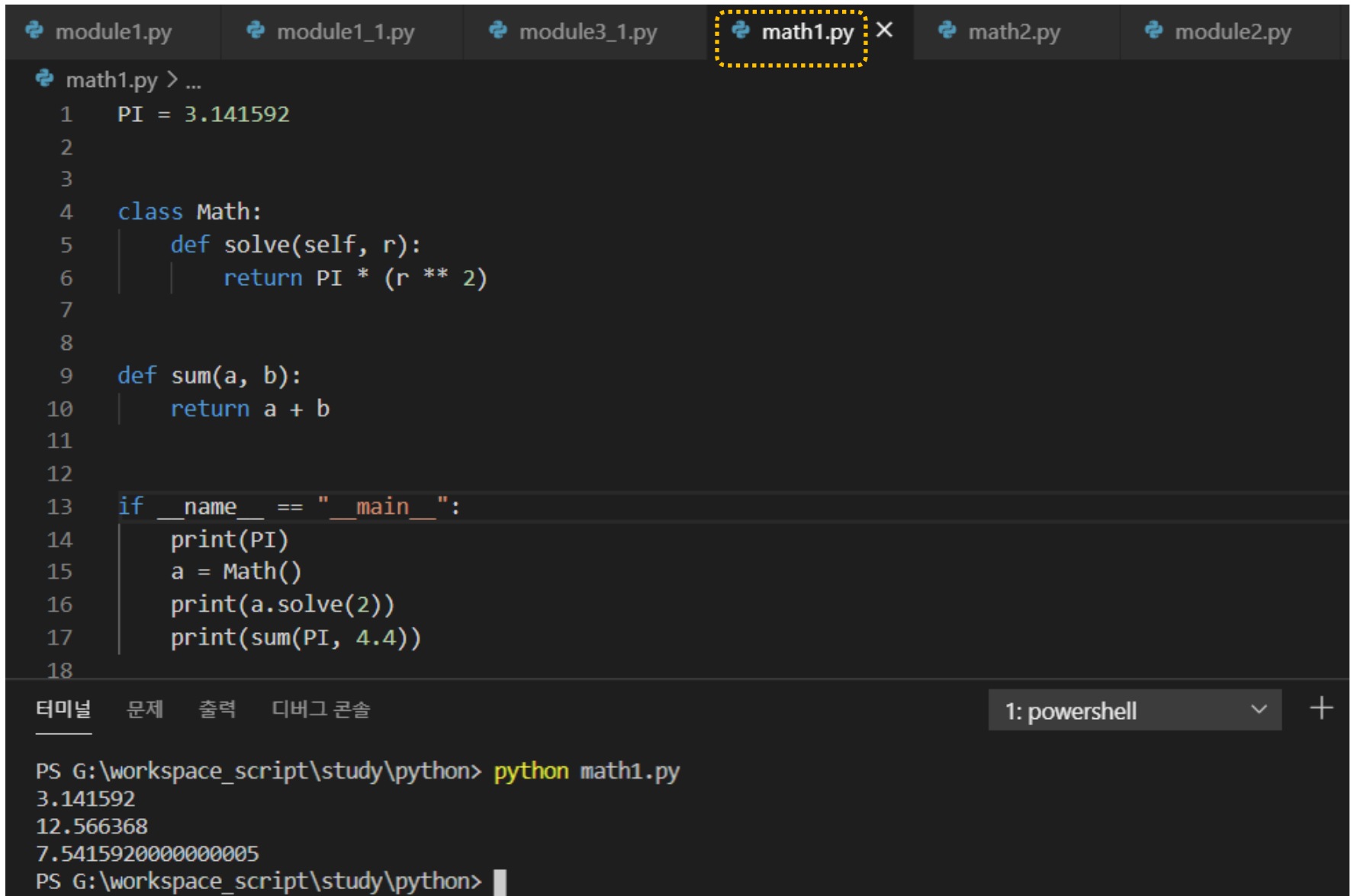
The `module1` output is highlighted with a yellow dashed box.

- 파이썬은 자동으로 실행되는 메인함수가 없다.
- 대신 들여쓰기 하지 않은 모든 코드(level 0 코드)를 실행한다.
- \_\_name\_\_은 현재 모듈의 이름을 담고 있는 내장변수
  - import로 모듈을 가져왔을 때 모듈의 이름으로 설정된다.
  - **모듈을 직접 실행하면 “\_\_name\_\_”은 “\_\_main\_\_”으로 설정된다.**



# 클래스나 변수등을 포함한 모듈

22

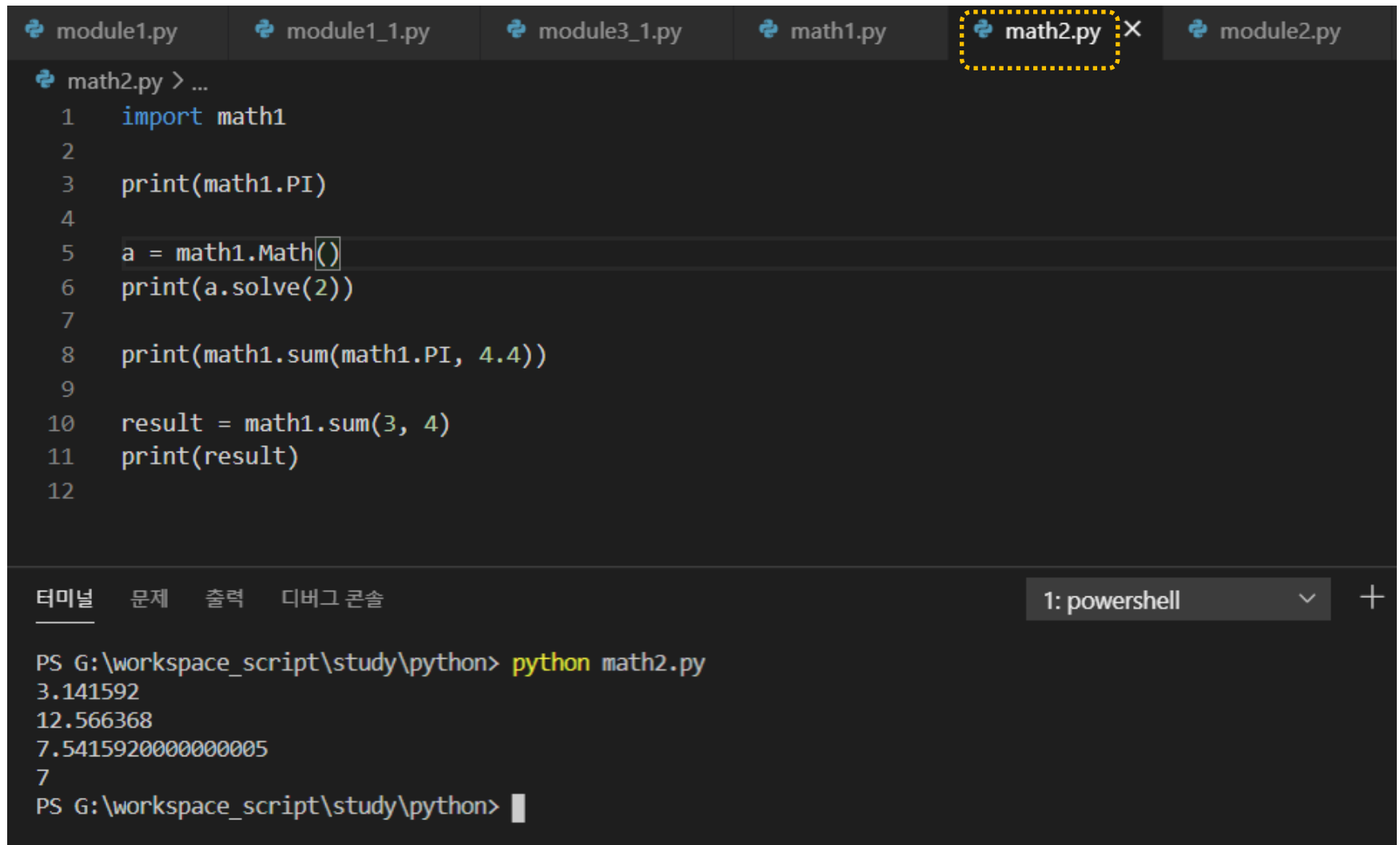


```
module1.py  module1_1.py  module3_1.py  math1.py X  math2.py  module2.py

math1.py > ...
1  PI = 3.141592
2
3
4  class Math:
5      def solve(self, r):
6          return PI * (r ** 2)
7
8
9  def sum(a, b):
10     return a + b
11
12
13  if __name__ == "__main__":
14     print(PI)
15     a = Math()
16     print(a.solve(2))
17     print(sum(PI, 4.4))
18

터미널  문제  출력  디버그 콘솔  1: powershell  +

PS G:\workspace_script\study\python> python math1.py
3.141592
12.566368
7.5415920000000005
PS G:\workspace_script\study\python> 
```



The image shows a code editor with several tabs at the top: module1.py, module1\_1.py, module3\_1.py, math1.py, math2.py (highlighted with a dashed yellow border), and module2.py. The active tab, math2.py, contains the following Python code:

```
math2.py > ...
1  import math1
2
3  print(math1.PI)
4
5  a = math1.Math()
6  print(a.solve(2))
7
8  print(math1.sum(math1.PI, 4.4))
9
10 result = math1.sum(3, 4)
11 print(result)
12
```

Below the code editor is a terminal window. The terminal has tabs for '터미널' (Terminal), '문제' (Problem), '출력' (Output), and '디버그 콘솔' (Debug Console). The '터미널' tab is active, showing the command prompt 'PS G:\workspace\_script\study\python>' and the execution of 'python math2.py'. The output of the script is displayed as follows:

```
PS G:\workspace_script\study\python> python math2.py
3.141592
12.566368
7.5415920000000005
7
PS G:\workspace_script\study\python>
```