

# Pytest Tutorial

郭益華

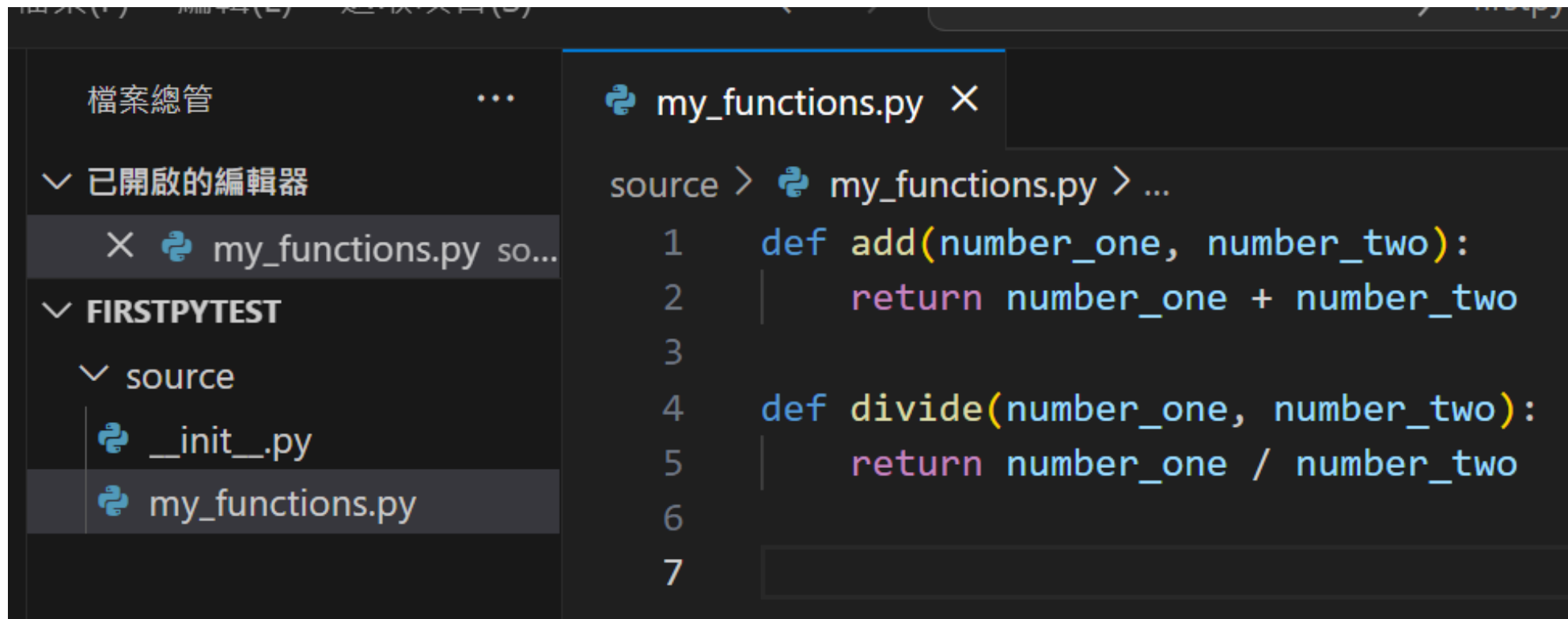
**GitHub**

# 目錄

1. [基礎測試\(function\)](#)
2. [Class-based Tests](#)
3. [Fixtures](#)
4. [Mark & Parametrize](#)
5. [Mocking](#)

# 1. 基礎測試 function()

# 撰寫基本的兩數相加及相除的函式



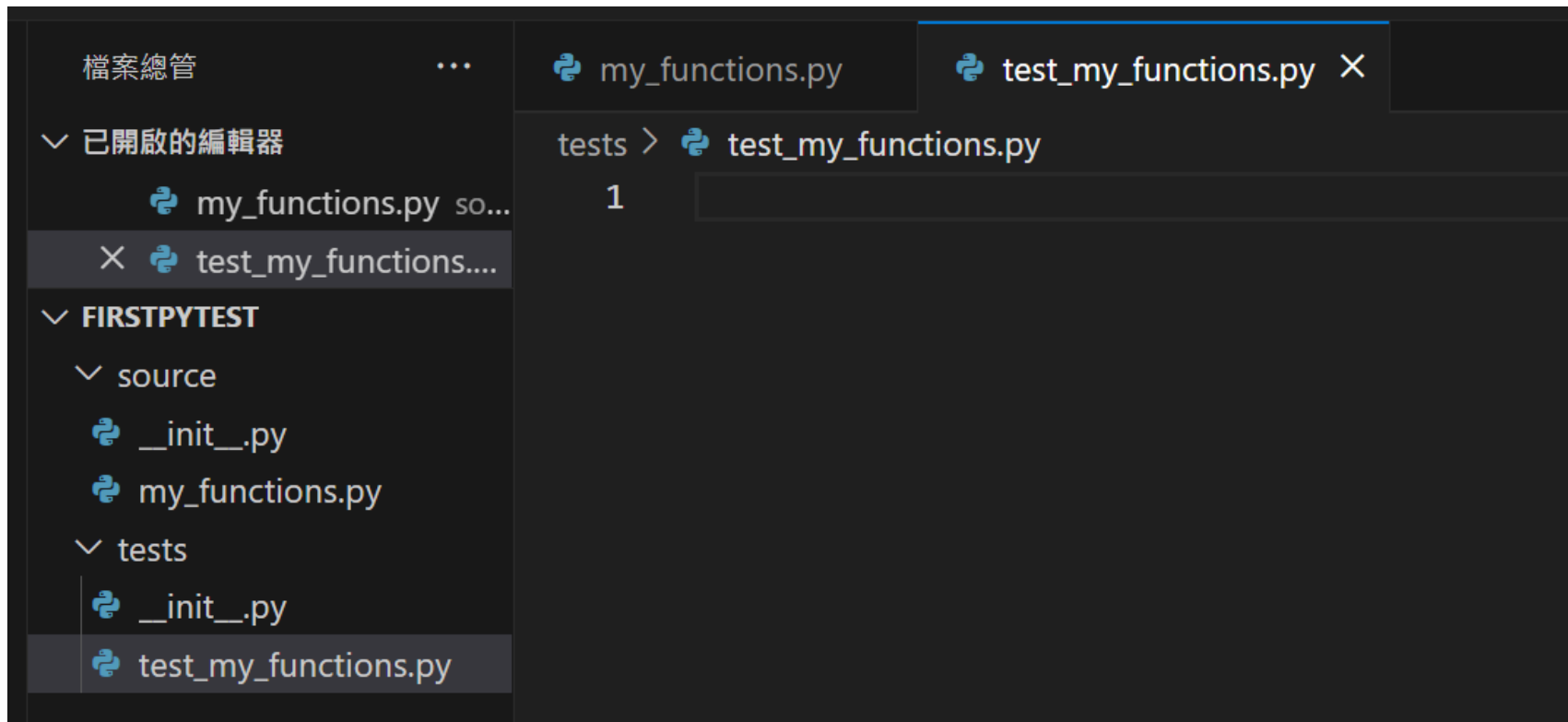
The screenshot shows a code editor with a sidebar on the left and a main editing area on the right. The sidebar contains a file explorer with the following structure:

- 檔案總管 (File Explorer)
  - 已開啟的編輯器 (Opened Editors)
    - my\_functions.py so...
  - FIRSTPYTEST
    - source
      - \_\_init\_\_.py
      - my\_functions.py (selected)

The main editing area shows the code for `my_functions.py`:

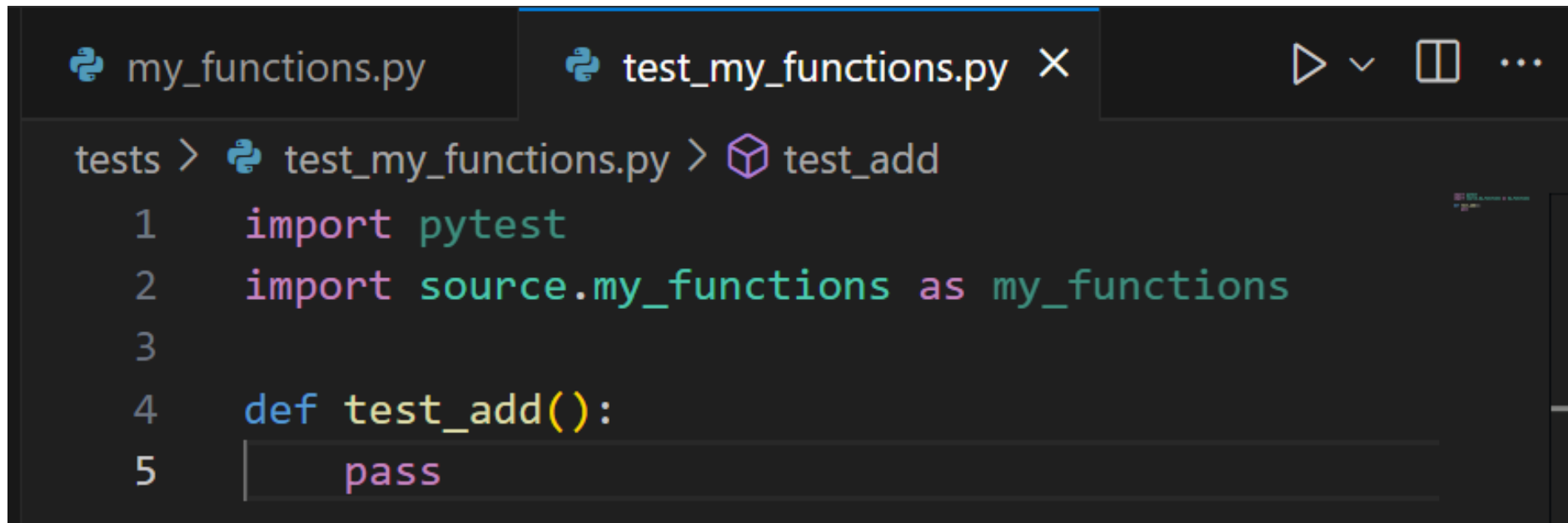
```
source > my_functions.py > ...  
1  def add(number_one, number_two):  
2      return number_one + number_two  
3  
4  def divide(number_one, number_two):  
5      return number_one / number_two  
6  
7
```

# 新增測試資料夾及程式碼



- 測試資料夾一定要命名為tests
- 測試程式碼開頭也必須命名test
- pytest會根據資料夾及程式碼名稱進行相對應的測試

# 測試 add()



The screenshot shows a code editor with two tabs: `my_functions.py` and `test_my_functions.py`. The `test_my_functions.py` tab is active. The breadcrumb navigation shows the path: `tests > test_my_functions.py > test_add`. The code in the editor is as follows:

```
1 import pytest
2 import source.my_functions as my_functions
3
4 def test_add():
5     pass
```

# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest> pytest tests/test_my_functions.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest
collected 1 item

tests\test_my_functions.py . [100%]

===== 1 passed in 0.02s =====
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest>
```

# 指定 assert

```
my_functions.py  test_my_functions.py X  ▶ ▾ □ ...

tests > test_my_functions.py > test_add
1  import pytest
2  import source.my_functions as my_functions
3
4  def test_add():
5      result = my_functions.add(1, 4)
6      assert result == 5
```

指定斷言(assert) ,  
1 + 4 = 5

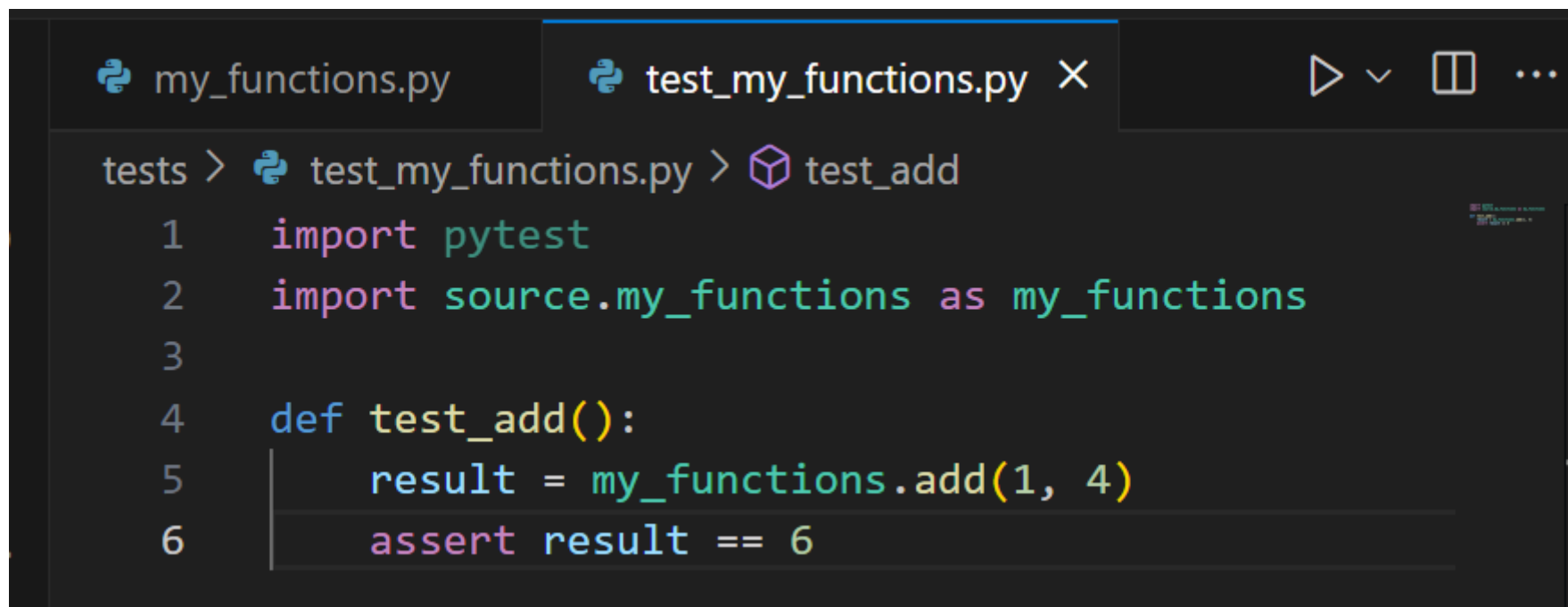
```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest> pytest tests/test_my_functions.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest
collected 1 item

tests\test_my_functions.py . [100%]

===== 1 passed in 0.05s =====
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest>
```



# 故意將 assert 設為 6



The screenshot shows a code editor with two tabs: `my_functions.py` and `test_my_functions.py`. The active tab is `test_my_functions.py`, which contains the following Python code:

```
tests > test_my_functions.py > test_add
1  import pytest
2  import source.my_functions as my_functions
3
4  def test_add():
5      result = my_functions.add(1, 4)
6      assert result == 6
```

The code is written in a dark-themed editor. The `assert` statement on line 6 is highlighted with a light blue background, indicating it is the current line of execution or focus. The breadcrumb navigation at the top of the editor shows the path: `tests > test_my_functions.py > test_add`.

# 會產生錯誤訊息，顯示 assert 有錯誤

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest> pytest tests/test_my_functions.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest
collected 1 item

tests\test_my_functions.py F [100%]

===== FAILURES =====
----- test_add -----

    def test_add():
        result = my_functions.add(1, 4)
>       assert result == 6
E       assert 5 == 6

tests\test_my_functions.py:6: AssertionError
===== short test summary info =====
FAILED tests/test_my_functions.py::test_add - assert 5 == 6
===== 1 failed in 0.21s =====
```

# 新增除法測試 10除以5

```

tests > test_my_functions.py > test_divide
1  import pytest
2  import source.my_functions as my_functions
3
4  def test_add():
5      result = my_functions.add(1, 4)
6      assert result == 5
7
8  def test_divide():
9      result = my_functions.divide(10, 5)
10     assert result == 2

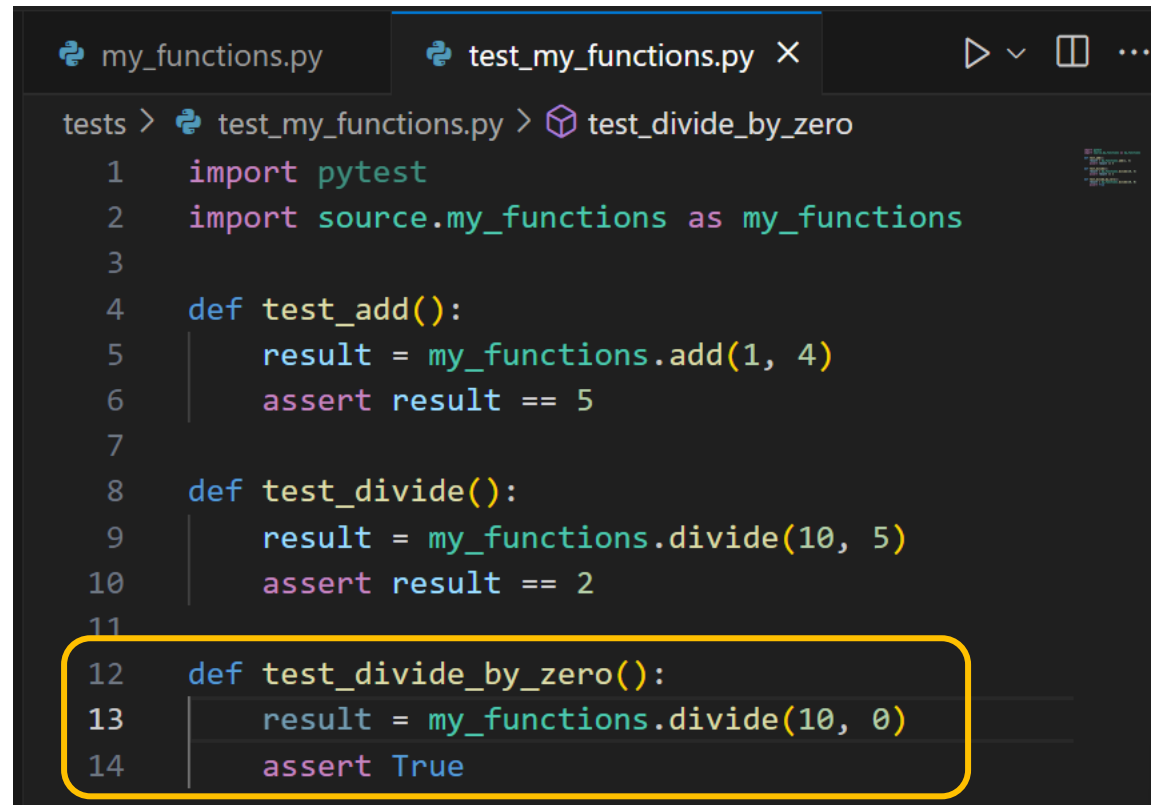
```

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest> pytest tests/test_my_functions.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest
collected 2 items

tests\test_my_functions.py .. [100%]

===== 2 passed in 0.02s =====
```

# 測試除以0



```
my_functions.py  test_my_functions.py X  ▶ ▢ ...
tests > test_my_functions.py > test_divide_by_zero
1  import pytest
2  import source.my_functions as my_functions
3
4  def test_add():
5      result = my_functions.add(1, 4)
6      assert result == 5
7
8  def test_divide():
9      result = my_functions.divide(10, 5)
10     assert result == 2
11
12     def test_divide_by_zero():
13         result = my_functions.divide(10, 0)
14         assert True
```

# 會產生 ZeroDivisionError

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest> pytest tests/test_my_functions.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest
collected 3 items

tests\test_my_functions.py ..F [100%]

===== FAILURES =====
----- test_divide_by_zero -----

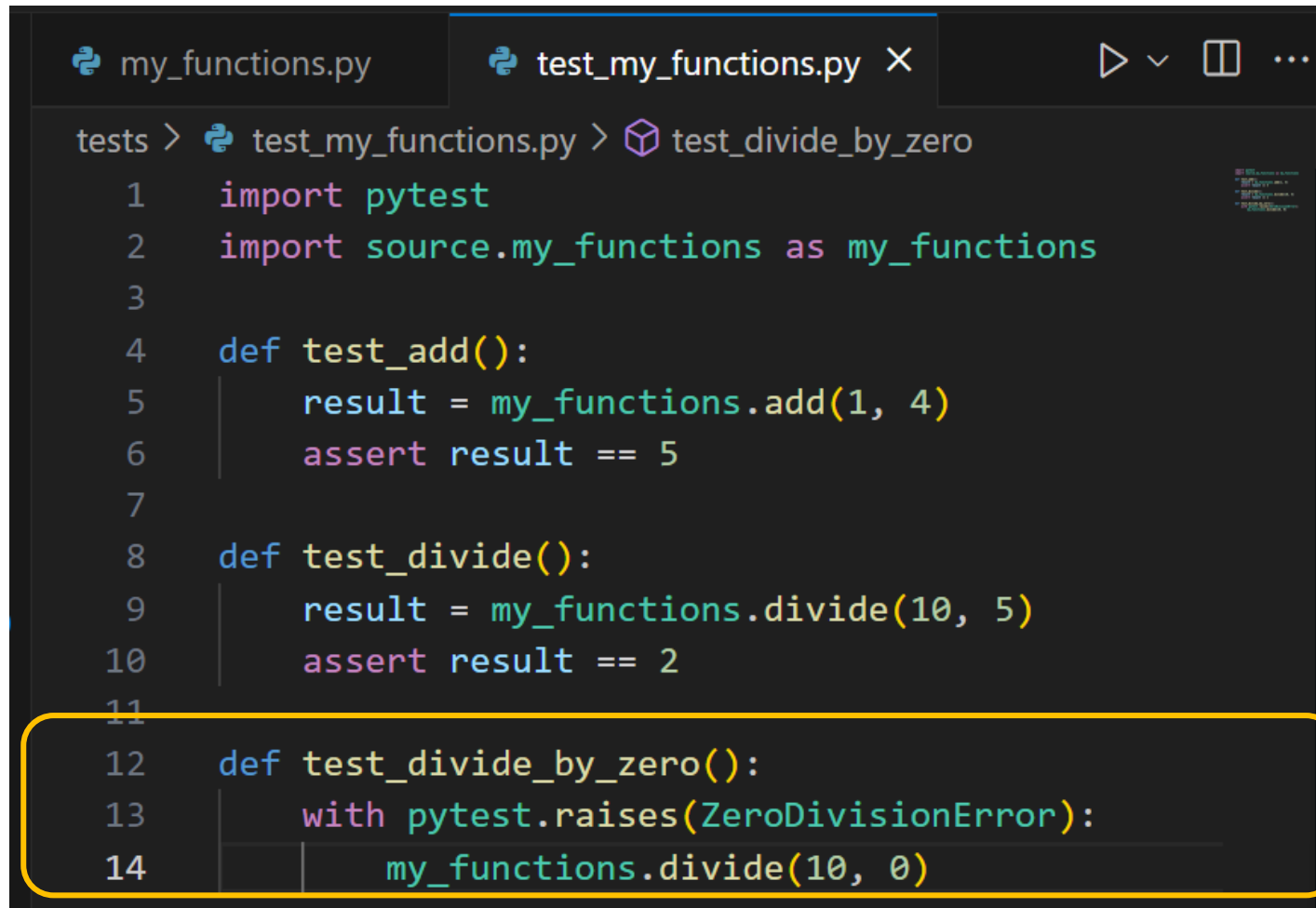
    def test_divide_by_zero():
>     result = my_functions.divide(10, 0)

tests\test_my_functions.py:13:
-----
number_one = 10, number_two = 0

    def divide(number_one, number_two):
>     return number_one / number_two
E     ZeroDivisionError: division by zero

source\my_functions.py:5: ZeroDivisionError
===== short test summary info =====
FAILED tests/test_my_functions.py::test_divide_by_zero - ZeroDivisionError: division by zero
===== 1 failed, 2 passed in 0.12s =====
```

# 將測試碼改為with ZeroDivisionError



The image shows a code editor with two tabs: `my_functions.py` and `test_my_functions.py`. The `test_my_functions.py` tab is active, showing the following code:

```
tests > test_my_functions.py > test_divide_by_zero
1  import pytest
2  import source.my_functions as my_functions
3
4  def test_add():
5      result = my_functions.add(1, 4)
6      assert result == 5
7
8  def test_divide():
9      result = my_functions.divide(10, 5)
10     assert result == 2
11
12     def test_divide_by_zero():
13         with pytest.raises(ZeroDivisionError):
14             my_functions.divide(10, 0)
```

The code for `test_divide_by_zero` (lines 12-14) is highlighted with a yellow rounded rectangle.

# 測試成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest> pytest tests/test_my_functions.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest
collected 3 items

tests\test_my_functions.py ... [100%]

===== 3 passed in 0.02s =====
```

# 修改主程式碼 divide()

```
my_functions.py × test_my_functions.py
source > my_functions.py > ...
1 def add(number_one, number_two):
2     return number_one + number_two
3
4 def divide(number_one, number_two):
5     if number_two == 0:
6         raise ValueError
7     return number_one / number_two
8
```

```
my_functions.py test_my_functions.py ×
tests > test_my_functions.py > test_divide_by_zero
1 import pytest
2 import source.my_functions as my_functions
3
4 def test_add():
5     result = my_functions.add(1, 4)
6     assert result == 5
7
8 def test_divide():
9     result = my_functions.divide(10, 5)
10    assert result == 2
11
12 def test_divide_by_zero():
13     # 當我調用這個函數時，預計會出現除法錯誤，
14     # 所以即使函數本身失敗或函數中存在錯誤，我們可以確認有這個錯誤存在
15     # ，而視為正常現象。
16     with pytest.raises(ZeroDivisionError):
17         my_functions.divide(10, 0)
```



# 產生錯誤 ValueError 的問題

```
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest
collected 3 items

tests\test_my_functions.py ..F [100%]

===== FAILURES =====
----- test_divide_by_zero -----

def test_divide_by_zero():
    # 當我調用這個函數時，預計會出現除法錯誤，
    # 所以即使函數本身失敗或函數中存在錯誤，我們可以確認有這個錯誤存在
    # ，而視為正常現象。
    with pytest.raises(ZeroDivisionError):
>         my_functions.divide(10, 0)

tests\test_my_functions.py:17:
-----
number_one = 10, number_two = 0

def divide(number_one, number_two):
    if number_two == 0:
>         raise ValueError
E         ValueError

source\my_functions.py:6: ValueError

===== short test summary info =====
FAILED tests/test_my_functions.py::test_divide_by_zero - ValueError
===== 1 failed, 2 passed in 0.12s =====
```

# 將測試碼的 with 改為 ValueError

```
def test_divide_by_zero():  
    # 當我調用這個函數時，預計會出現除法錯誤，  
    # 所以即使函數本身失敗或函數中存在錯誤，我們可以確認有這個錯誤存在  
    # ，而視為正常現象。  
    with pytest.raises(ValueError):  
        my_functions.divide(10, 0)
```

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest> pytest tests/test_my_functions.py  
===== test session starts =====  
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0  
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest  
collected 3 items  
  
tests\test_my_functions.py ... [100%]  
  
===== 3 passed in 0.02s =====
```

# 測試字串相加

```
my_functions.py  test_my_functions.py X
tests > test_my_functions.py > test_add_string
1  import pytest
2  import source.my_functions as my_functions
3
4  def test_add():
5      result = my_functions.add(1, 4)
6      assert result == 5
7
8  def test_add_string():
9      result = my_functions.add("I like", "burgers")
10     assert result == "I like burgers"
11
12 def test_divide():
13     result = my_functions.divide(10, 5)
14     assert result == 2
```

# 產生錯誤

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest> pytest tests/test_my_functions.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest
collected 4 items

tests\test_my_functions.py .F.. [100%]

===== FAILURES =====
----- test_add_string -----

  def test_add_string():
      result = my_functions.add("I like", "burgers")
>      assert result == "I like burgers"
E       AssertionError: assert 'I likeburgers' == 'I like burgers'
E       - I like burgers
E       ?      -
E       + I likeburgers

tests\test_my_functions.py:10: AssertionError
===== short test summary info =====
FAILED tests/test_my_functions.py::test_add_string - AssertionError: assert 'I likeburgers' == 'I like burgers'
===== 1 failed, 3 passed in 0.12s =====
```

可發現是字串空格的問題

# 修改 like

```
4 def test_add():
5     result = my_functions.add(1, 4)
6     assert result == 5
7
8 def test_add_string():
9     result = my_functions.add("I like ", "burgers")
10    assert result == "I like burgers"
11
12 def test_divide():
13     result = my_functions.divide(10, 5)
14     assert result == 2
```

like後方需空一格

# 測試成功

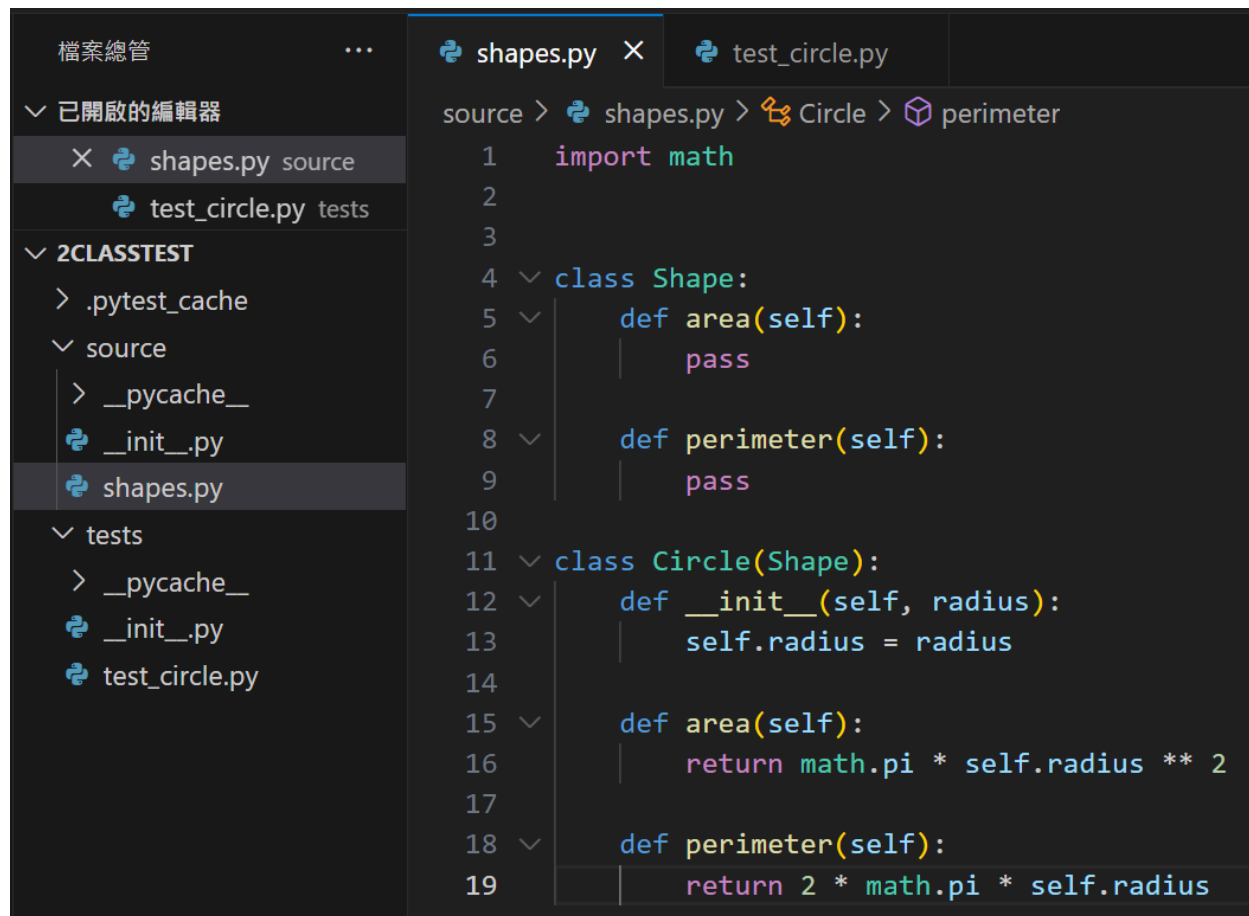
```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest> pytest tests/test_my_functions.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\firstpytest
collected 4 items

tests\test_my_functions.py .... [100%]

===== 4 passed in 0.03s =====
```

## 2. Class-based Tests

# 撰寫一個計算圓形面積周長的class

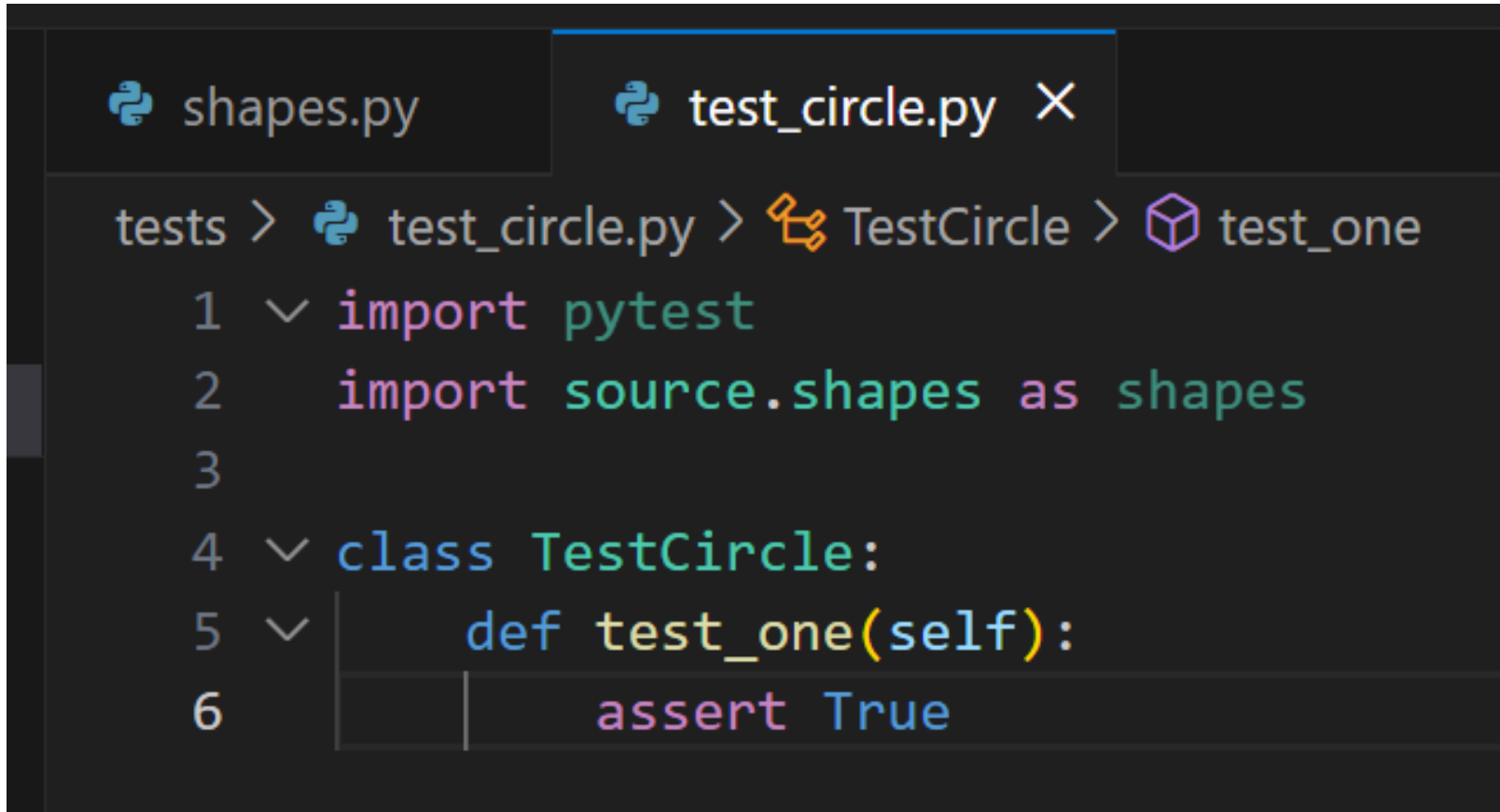


The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'source' folder containing 'shapes.py' and a 'tests' folder containing 'test\_circle.py'. The code editor shows the 'shapes.py' file with the following code:

```
source > shapes.py > Circle > perimeter
1  import math
2
3
4  class Shape:
5      def area(self):
6          pass
7
8      def perimeter(self):
9          pass
10
11 class Circle(Shape):
12     def __init__(self, radius):
13         self.radius = radius
14
15     def area(self):
16         return math.pi * self.radius ** 2
17
18     def perimeter(self):
19         return 2 * math.pi * self.radius
```



# 撰寫測試.py



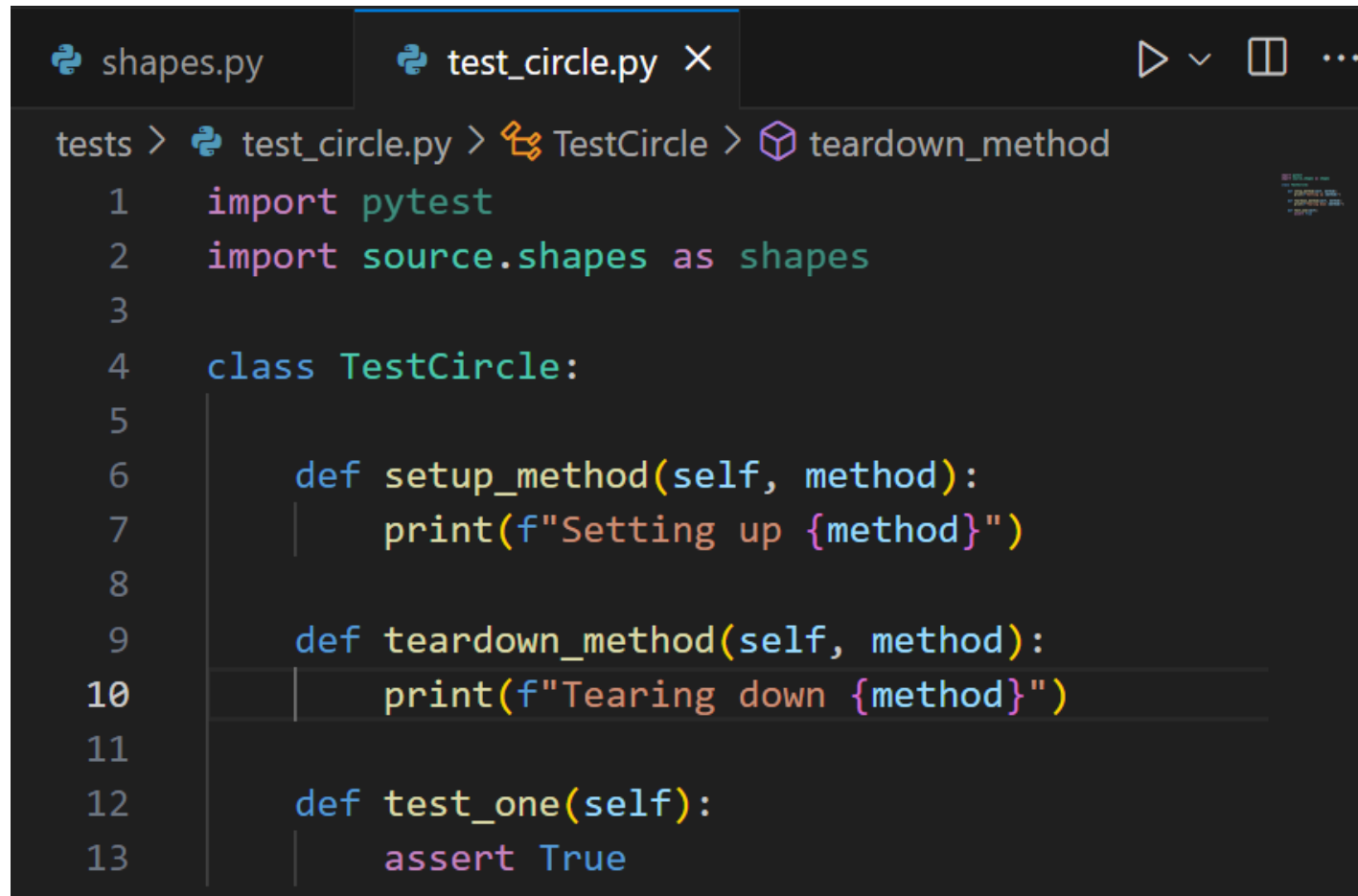
```
tests > test_circle.py > TestCircle > test_one
1  import pytest
2  import source.shapes as shapes
3
4  class TestCircle:
5      def test_one(self):
6          assert True
```

# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_circle.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 1 item

tests\test_circle.py . [100%]

===== 1 passed in 0.02s =====
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest>
```



```
shapes.py test_circle.py X
tests > test_circle.py > TestCircle > teardown_method
1 import pytest
2 import source.shapes as shapes
3
4 class TestCircle:
5
6     def setup_method(self, method):
7         print(f"Setting up {method}")
8
9     def teardown_method(self, method):
10        print(f"Tearing down {method}")
11
12    def test_one(self):
13        assert True
```

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_circle.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 1 item

tests\test_circle.py . [100%]

===== 1 passed in 0.01s =====
```

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest -s
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 1 item

tests\test_circle.py Setting up <bound method TestCircle.test_one of <tests.test_circle.TestCircle object at 0x0000021EFD4E1C60>>
.Tearing down <bound method TestCircle.test_one of <tests.test_circle.TestCircle object at 0x0000021EFD4E1C60>>

===== 1 passed in 0.02s =====
```

-s 是pytest的一個選項，它可以將測試過程中的標準輸出（stdout）和標準錯誤輸出（stderr）顯示在終端上

# 測試面積計算

```
shapes.py test_circle.py X
tests > test_circle.py > TestCircle > test_area
1  import pytest
2  import math
3  import source.shapes as shapes
4
5  class TestCircle:
6
7      def setup_method(self, method):
8          print(f"Setting up {method}")
9          self.circle = shapes.Circle(10)
10
11     def teardown_method(self, method):
12         print(f"Tearing down {method}")
13
14     # def test_one(self):
15     #     assert True
16
17     def test_area(self):
18         assert self.circle.area() == math.pi * self.circle.radius ** 2
```

# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_circle.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 1 item

tests\test_circle.py . [100%]

===== 1 passed in 0.01s =====
```

# 測試周長

```
def test_area(self):  
    assert self.circle.area() == math.pi * self.circle.radius ** 2  
  
def test_perimeter(self):  
    result = self.circle.perimeter()  
    expected = 2 * math.pi * self.circle.radius  
    assert result == expected
```

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_circle.py  
===== test session starts =====  
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0  
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest  
collected 2 items  
  
tests\test_circle.py .. [100%]  
  
===== 2 passed in 0.02s =====
```

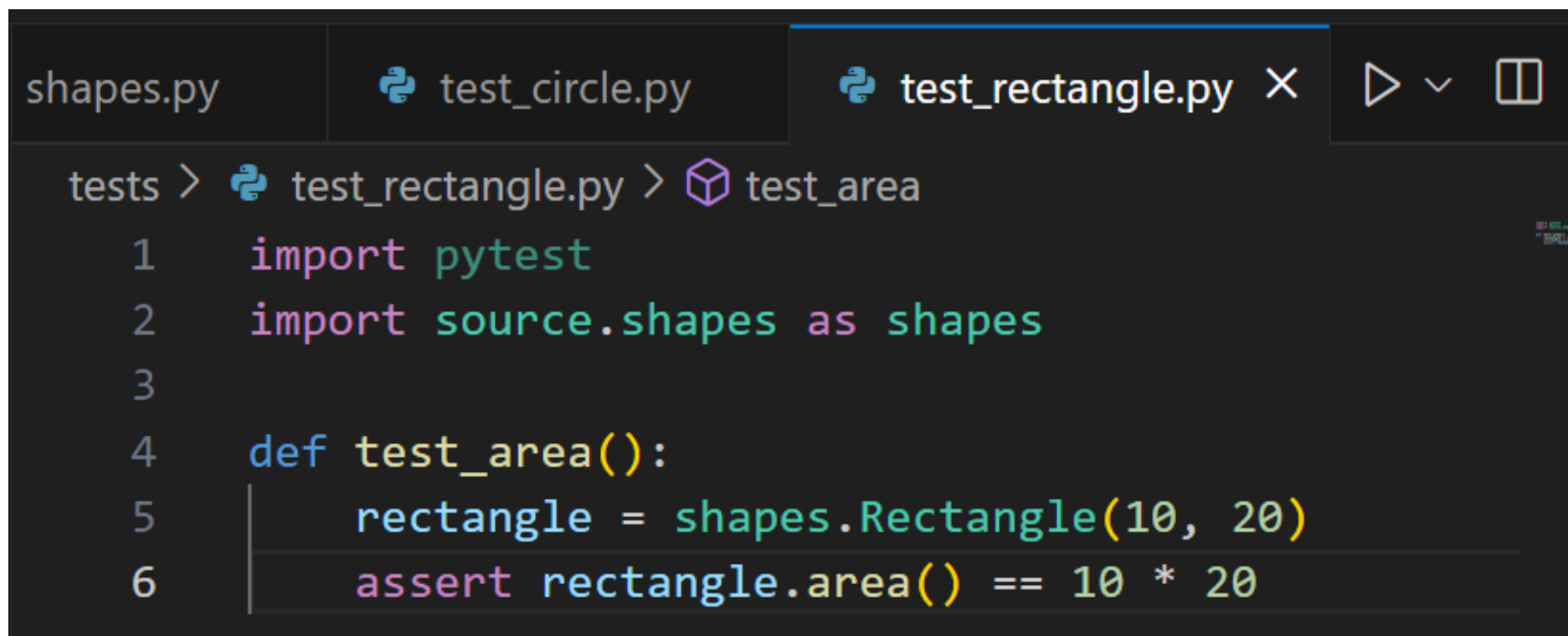
# 3. Fixtures



# 新撰寫一個計算長方形面積周長的class

```
class Rectangle(Shape):  
    def __init__(self, length, width):  
        self.length = length  
        self.width = width  
  
    def area(self):  
        return self.length * self.width  
  
    def perimeter(self):  
        return (self.length * 2) + (self.width * 2)
```

# 撰寫測試碼



The image shows a code editor with three tabs: shapes.py, test\_circle.py, and test\_rectangle.py. The test\_rectangle.py tab is active. The breadcrumb navigation shows 'tests > test\_rectangle.py > test\_area'. The code defines a test function 'test\_area' that creates a 'Rectangle' object with dimensions 10 and 20, and asserts that its area is equal to 10 \* 20.

```
shapes.py  test_circle.py  test_rectangle.py ×  ▶  ▾  □
```

tests > test\_rectangle.py > test\_area

```
1  import pytest
2  import source.shapes as shapes
3
4  def test_area():
5      rectangle = shapes.Rectangle(10, 20)
6      assert rectangle.area() == 10 * 20
```

# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_rectangle.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 1 item

tests\test_rectangle.py . [100%]

===== 1 passed in 0.02s =====
```

# 測試周長與面積

```
ts > test_rectangle.py > test_perimeter
1 import pytest
2 import source.shapes as shapes
3
4 def test_area():
5     rectangle = shapes.Rectangle(10, 20)
6     assert rectangle.area() == 10 * 20
7
8 def test_perimeter():
9     rectangle = shapes.Rectangle(10, 20)
10    assert rectangle.perimeter() == (10*2) + (20*2)
```

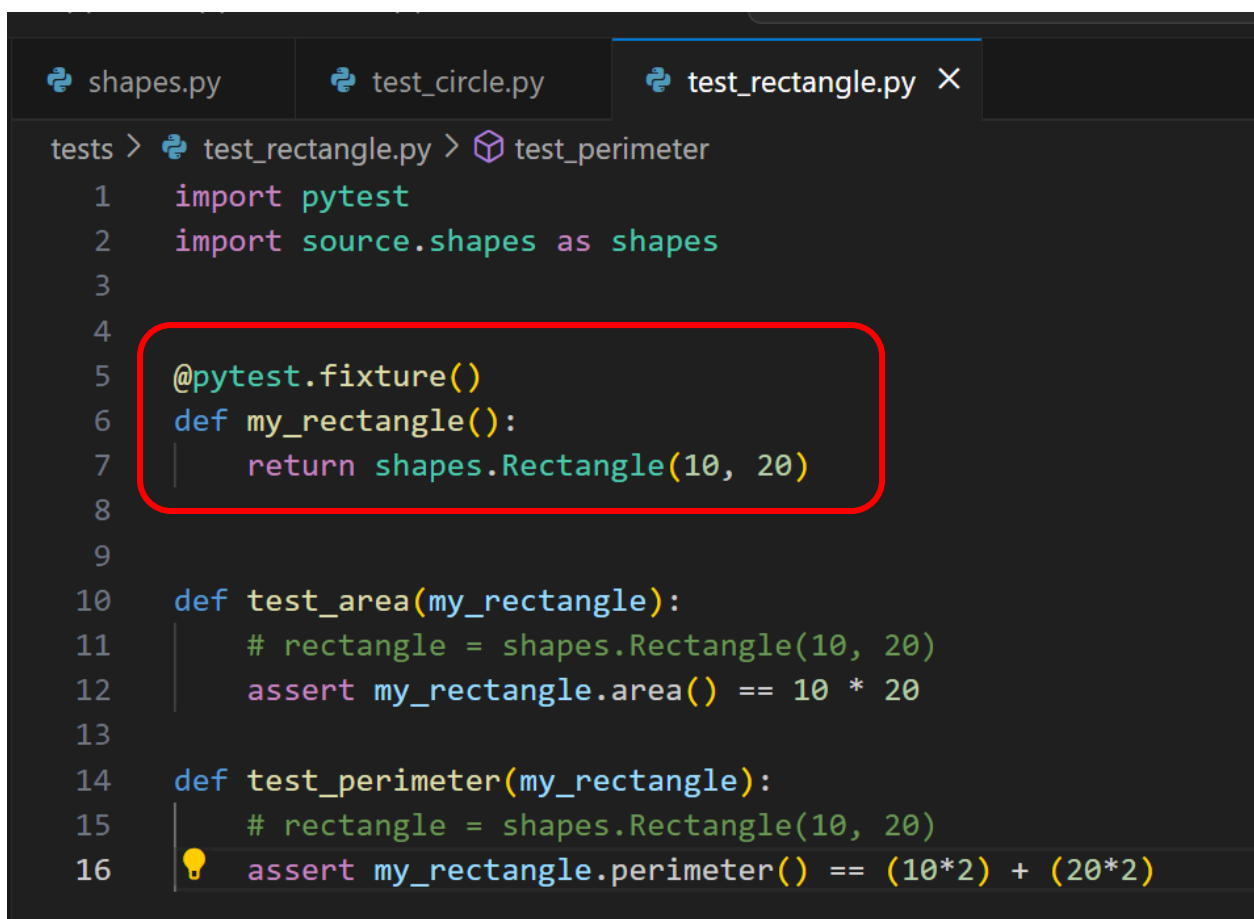
```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_rectangle.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 2 items

tests\test_rectangle.py .. [100%]

===== 2 passed in 0.02s =====
```

# 修改程式碼-Fixture

使用 fixture 先定義好測試參數，就可以重複呼叫，不需要每次都手動填寫參數



```
shapes.py  test_circle.py  test_rectangle.py X
tests > test_rectangle.py > test_perimeter
1  import pytest
2  import source.shapes as shapes
3
4
5  @pytest.fixture()
6  def my_rectangle():
7      return shapes.Rectangle(10, 20)
8
9
10 def test_area(my_rectangle):
11     # rectangle = shapes.Rectangle(10, 20)
12     assert my_rectangle.area() == 10 * 20
13
14 def test_perimeter(my_rectangle):
15     # rectangle = shapes.Rectangle(10, 20)
16     assert my_rectangle.perimeter() == (10*2) + (20*2)
```

# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_rectangle.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 2 items

tests\test_rectangle.py .. [100%]

===== 2 passed in 0.02s =====
```

# 在主程式碼新增一個 辨識是否為長方形的函式

```
shapes.py × test_circle.py test_rectangle.py ▶ ▾ □
source > shapes.py > ...
17
18     def perimeter(self):
19         return 2 * math.pi * self.radius
20
21     class Rectangle(Shape):
22         def __init__(self, length, width):
23             self.length = length
24             self.width = width
25
26         def __eq__(self, other):
27             if not isinstance(other, Rectangle):
28                 return False
29
```

# 撰寫測試碼

```
shapes.py test_circle.py test_rectangle.py X
tests > test_rectangle.py > test_not_equal
1 import pytest
2 import source.shapes as shapes
3
4
5 @pytest.fixture()
6 def my_rectangle():
7     return shapes.Rectangle(10, 20)
8
9 @pytest.fixture()
10 def weird_rectangle():
11     return shapes.Rectangle(5, 6)
12
13 def test_area(my_rectangle):
14     # rectangle = shapes.Rectangle(10, 20)
15     assert my_rectangle.area() == 10 * 20
16
17 def test_perimeter(my_rectangle):
18     # rectangle = shapes.Rectangle(10, 20)
19     assert my_rectangle.perimeter() == (10*2) + (20*2)
20
21 def test_not_equal(my_rectangle, weird_rectangle):
22     assert my_rectangle != weird_rectangle
```



# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_rectangle.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 3 items

tests\test_rectangle.py ... [100%]

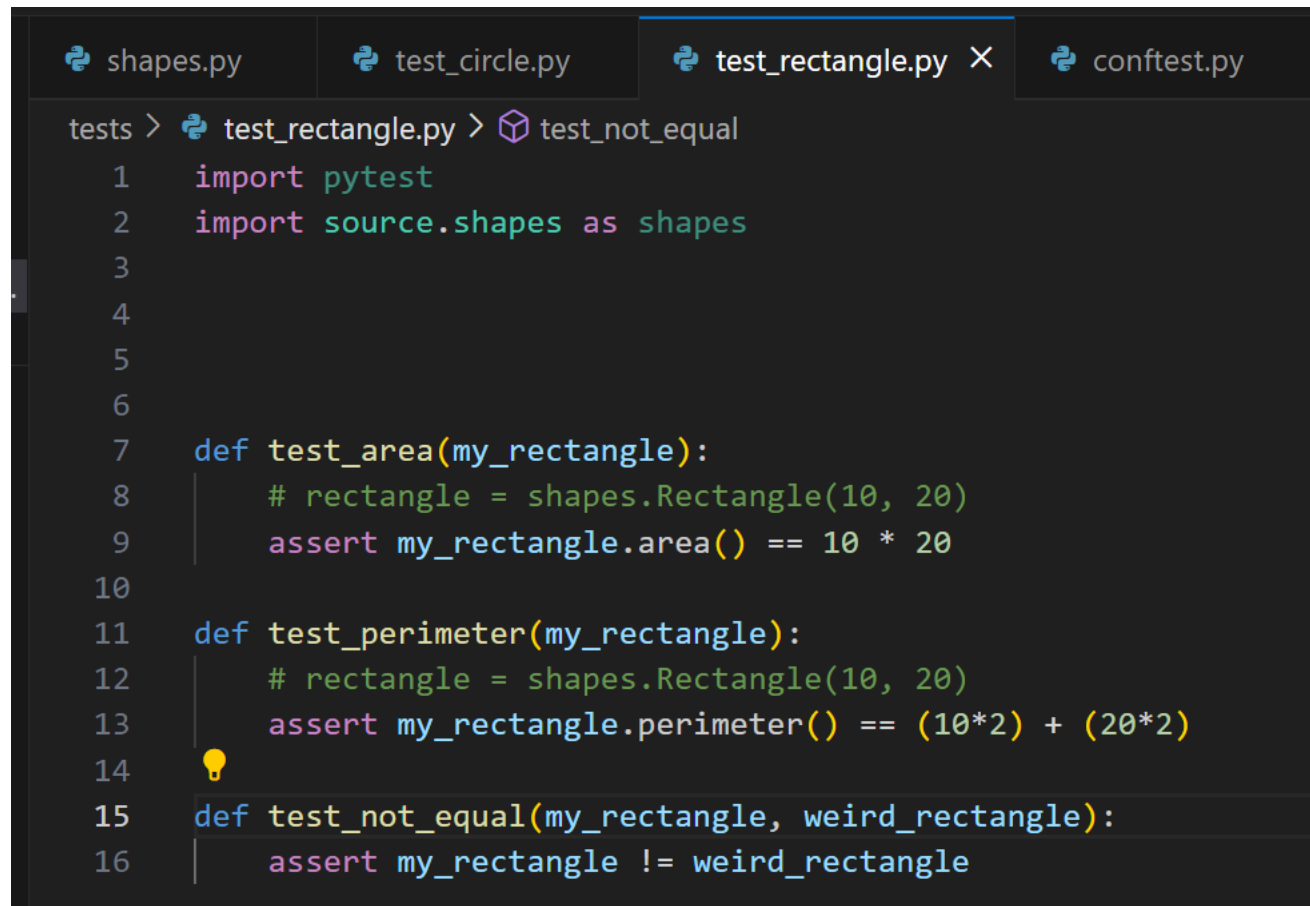
===== 3 passed in 0.02s =====
```

# 將Fixture獨立於一個檔案中


```
shapes.py  test_circle.py  test_rectangle.py  conftest.py X
tests > conftest.py > ...
1  import pytest
2  import source.shapes as shapes
3
4  @pytest.fixture()
5  def my_rectangle():
6      return shapes.Rectangle(10, 20)
7
8  @pytest.fixture()
9  def weird_rectangle():
10     return shapes.Rectangle(5, 6)
```

- 在前面Fixture與測試碼都混和在一起，太多可能導致程式碼混亂。
- 新增一個 名為 `conftest.py`
- 切記檔名必須為 `conftest`
- pytest會自動偵測檔名

# 測試碼保留原本的測試函式



The screenshot shows a code editor with four tabs: shapes.py, test\_circle.py, test\_rectangle.py (active), and conftest.py. The active tab displays the following Python code:

```
tests > test_rectangle.py > test_not_equal
1  import pytest
2  import source.shapes as shapes
3
4
5
6
7  def test_area(my_rectangle):
8      # rectangle = shapes.Rectangle(10, 20)
9      assert my_rectangle.area() == 10 * 20
10
11 def test_perimeter(my_rectangle):
12     # rectangle = shapes.Rectangle(10, 20)
13     assert my_rectangle.perimeter() == (10*2) + (20*2)
14     
15 def test_not_equal(my_rectangle, weird_rectangle):
16     assert my_rectangle != weird_rectangle
```

# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_rectangle.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 3 items

tests\test_rectangle.py ... [100%]

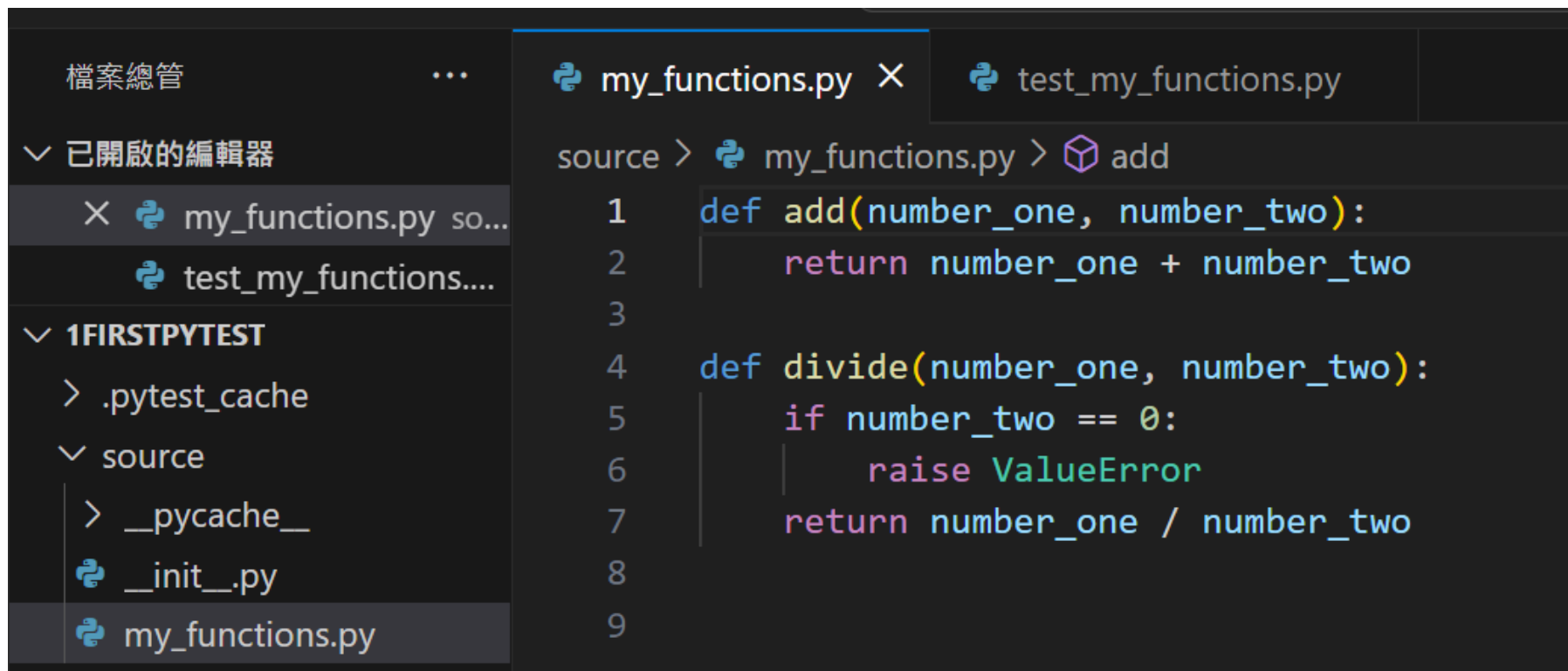
===== 3 passed in 0.02s =====
```

## 4. Mark & Parametrize

# 說明

- `usefixtures`：在測試函數或類上使用 fixtures。
- `filterwarnings`：過濾某些測試函數的警告。
- `skip`：總是跳過測試函數。
- `skipif`：如果滿足某些條件，則跳過測試函數。
- `xfail`：如果滿足某些條件，則產生“預期失敗”的結果。
- `parametrize`：對同一個測試函數執行多次調用。

# 回到 my\_functions.py



檔案總管 ...

已開啟的編輯器

- × my\_functions.py so...
- test\_my\_functions....

1FIRSTPYTEST

- > .pytest\_cache
- source
  - > \_\_pycache\_\_
  - \_\_init\_\_.py
  - my\_functions.py

source > my\_functions.py > add

```
1 def add(number_one, number_two):
2     return number_one + number_two
3
4 def divide(number_one, number_two):
5     if number_two == 0:
6         raise ValueError
7     return number_one / number_two
8
9
```

# 新增 time 作為後續範例使用

```
my_functions.py  test_my_functions.py X
tests > test_my_functions.py > test_very_slow
1  import pytest
2  import time
3  import source.my_functions as my_functions
4
23
24  def test_very_slow():
25      time.sleep(5)
26      result = my_functions.divide(10, 5)
27      assert result == 2
```



# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\1firstpytest> pytest .\tests\test_my_functions.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\1firstpytest
collected 5 items

tests\test_my_functions.py ..... [100%]

===== 5 passed in 5.05s =====
```

# 使用 mark

@pytest.mark.[name]

```
@pytest.mark.slow
def test_very_slow():
    time.sleep(5)
    result = my_functions.divide(10, 5)
    assert result == 2
```

# 成功

測試的時候即可直接在終端輸入指令:

**pytest -m slow**

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\1firstpytest> pytest .\tests\test_my_functions.py -m slow
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\1firstpytest
collected 5 items / 4 deselected / 1 selected

tests\test_my_functions.py . [100%]

===== warnings summary =====
tests\test_my_functions.py:23
  C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\1firstpytest\tests\test_my_functions.py:23: PytestUnknownMarkWarning: Unknown pytest.mark.slow - is this a typo? You can register custom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-to/mark.html
    @pytest.mark.slow

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 1 passed, 4 deselected, 1 warning in 5.04s =====
```

# 使用skip

```
@pytest.mark.skip(reason="This feature is currently broken")  
def test_add():  
    assert my_functions.add(1, 2) == 3
```

# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\1firstpytest> pytest .\tests\test_my_functions.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\1firstpytest
collected 5 items

tests\test_my_functions.py s.... [100%]

===== warnings summary =====
tests\test_my_functions.py:23
  C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\1firstpytest\tests\test_my_functions.py:23: PytestUnknownMarkWarning: Unknown pytest.mark.slow - is this a typo? You can register custom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-to/mark.html
    @pytest.mark.slow

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 4 passed, 1 skipped, 1 warning in 5.03s =====
```

# 使用 xfail

```
3  @pytest.mark.xfail(reason="We know we cannot divide by zero")
4  def test_divide_zero_broken():
5      my_functions.divide(4, 0)
```

# 成功

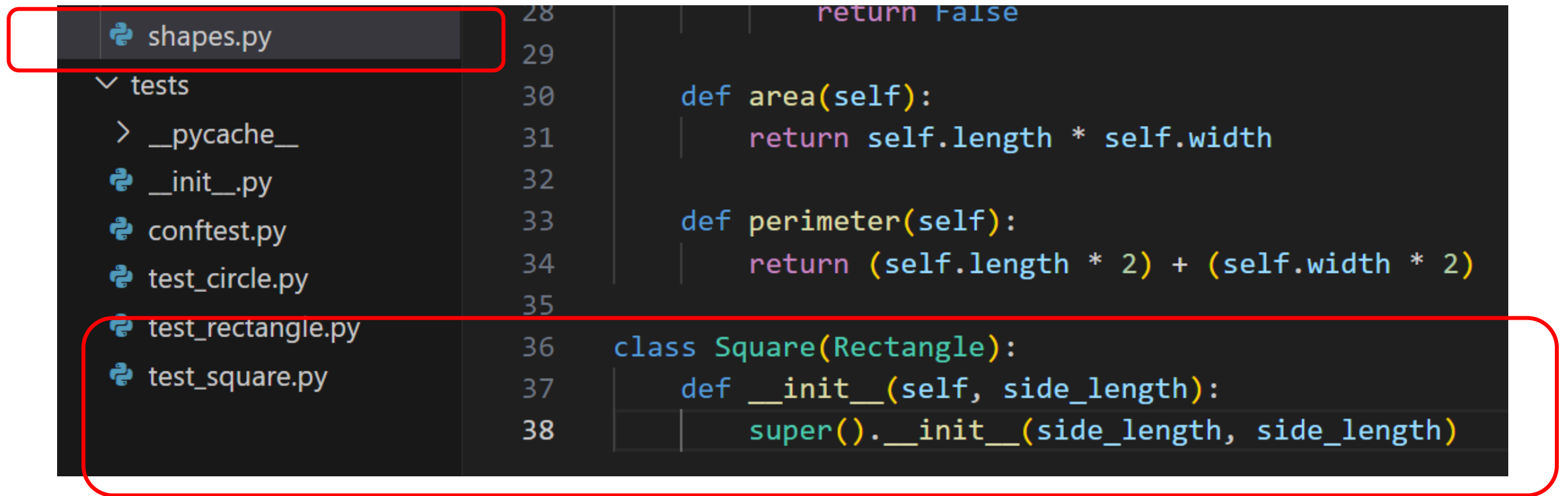
```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\1firstpytest> pytest .\tests\test_my_functions.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\1firstpytest
collected 6 items

tests\test_my_functions.py s...x [100%]

===== warnings summary =====
tests\test_my_functions.py:23
  C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\1firstpytest\tests\test_my_functions.py:23: PytestUnknownMarkWarning: Unknown pytest.mark.slow - is this a typo? You can register custom marks to avoid this warning - for details, see https://docs.pytest.org/en/stable/how-to/mark.html
    @pytest.mark.slow

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 4 passed, 1 skipped, 1 xfailed, 1 warning in 5.08s =====
```

# 回到 shapes.py 撰寫 class Square



```
28         return False
29
30     def area(self):
31         return self.length * self.width
32
33     def perimeter(self):
34         return (self.length * 2) + (self.width * 2)
35
36     class Square(Rectangle):
37         def __init__(self, side_length):
38             super().__init__(side_length, side_length)
```



# 使用 parametrize 一次測試多種

```
shapes.py test_square.py × test_circle.py test_rectangle.py conftest.py
tests > test_square.py > test_multiple_square_areas
1 import pytest
2 import source.shapes as shapes
3
4
5 @pytest.mark.parametrize("side_length, expected_area", [(5, 25), (4, 16), (9, 81)])
6 def test_multiple_square_areas(side_length, expected_area):
7     assert shapes.Square(side_length).area() == expected_area
```

一次測試三種

# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_square.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 3 items

tests\test_square.py ... [100%]

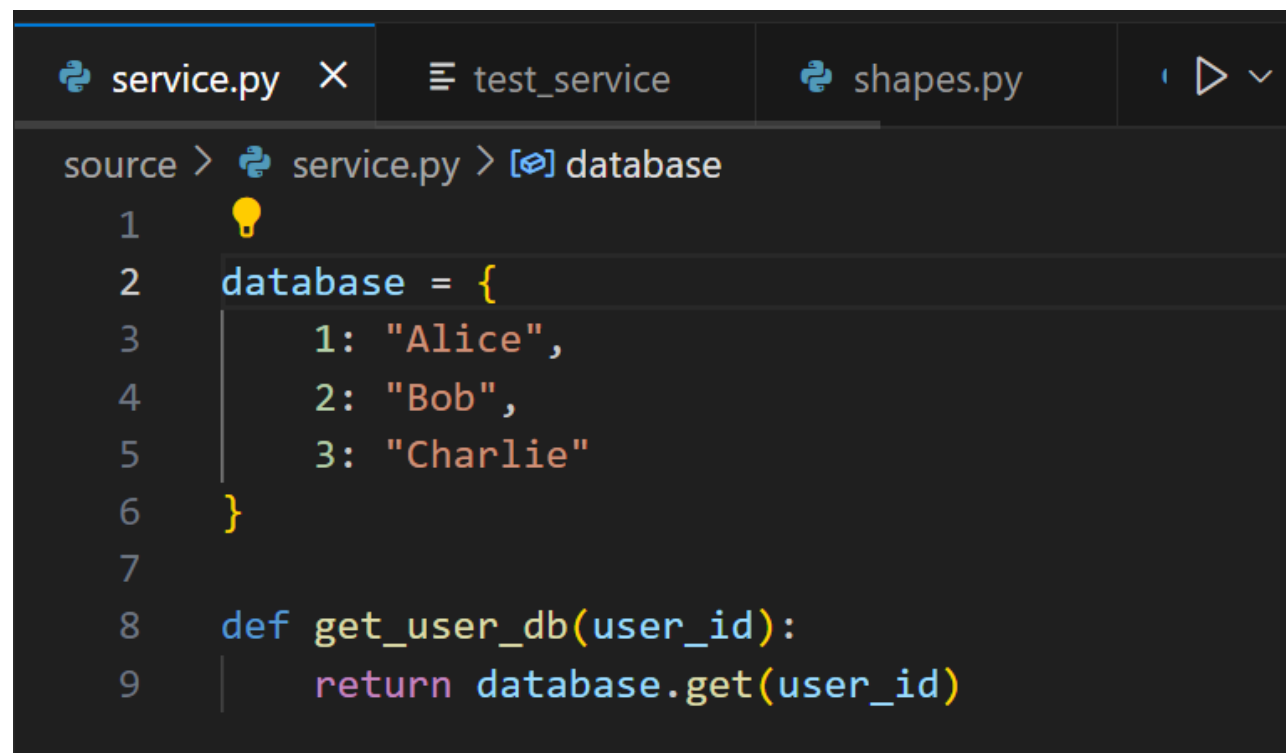
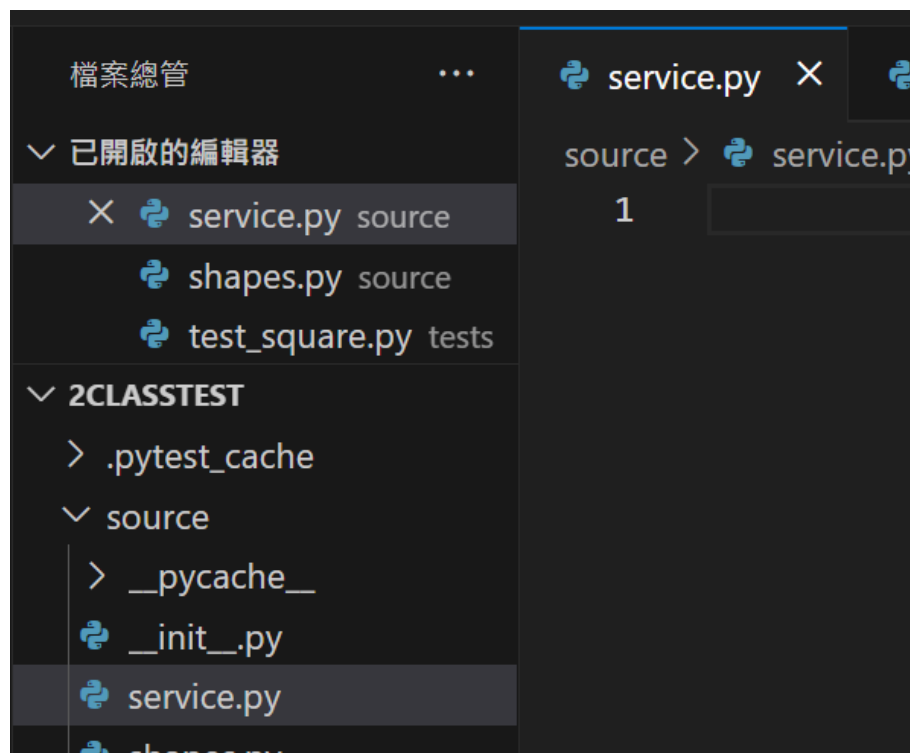
===== 3 passed in 0.01s =====
```

# 5. Mocking

# 說明

- 模擬API測試，使用真正的API可能會需要付費或是遇到一些不可抗力的因素(ex: API網站故障、維護等等)導致API無法順利使用。
- 利用模擬API測試的方式則可以避免這樣的情形，增加測試效率。

# 在source資料夾中先建立一個 service.py



# 撰寫 test\_service.py

```
service.py  test_service.py ×  shapes.py  test_square.py

tests > test_service.py > test_get_user_from_db
1  import pytest
2  import source.service as service
3  import unittest.mock as mock
4
5  @mock.patch("source.service.get_user_from_db")
6  def test_get_user_from_db(mock_get_user_from_db):
7      mock_get_user_from_db.return_value = "Mocked Alice"
8      user_name = service.get_user_from_db(1)
9
10     assert user_name == "Mocked Alice"
```

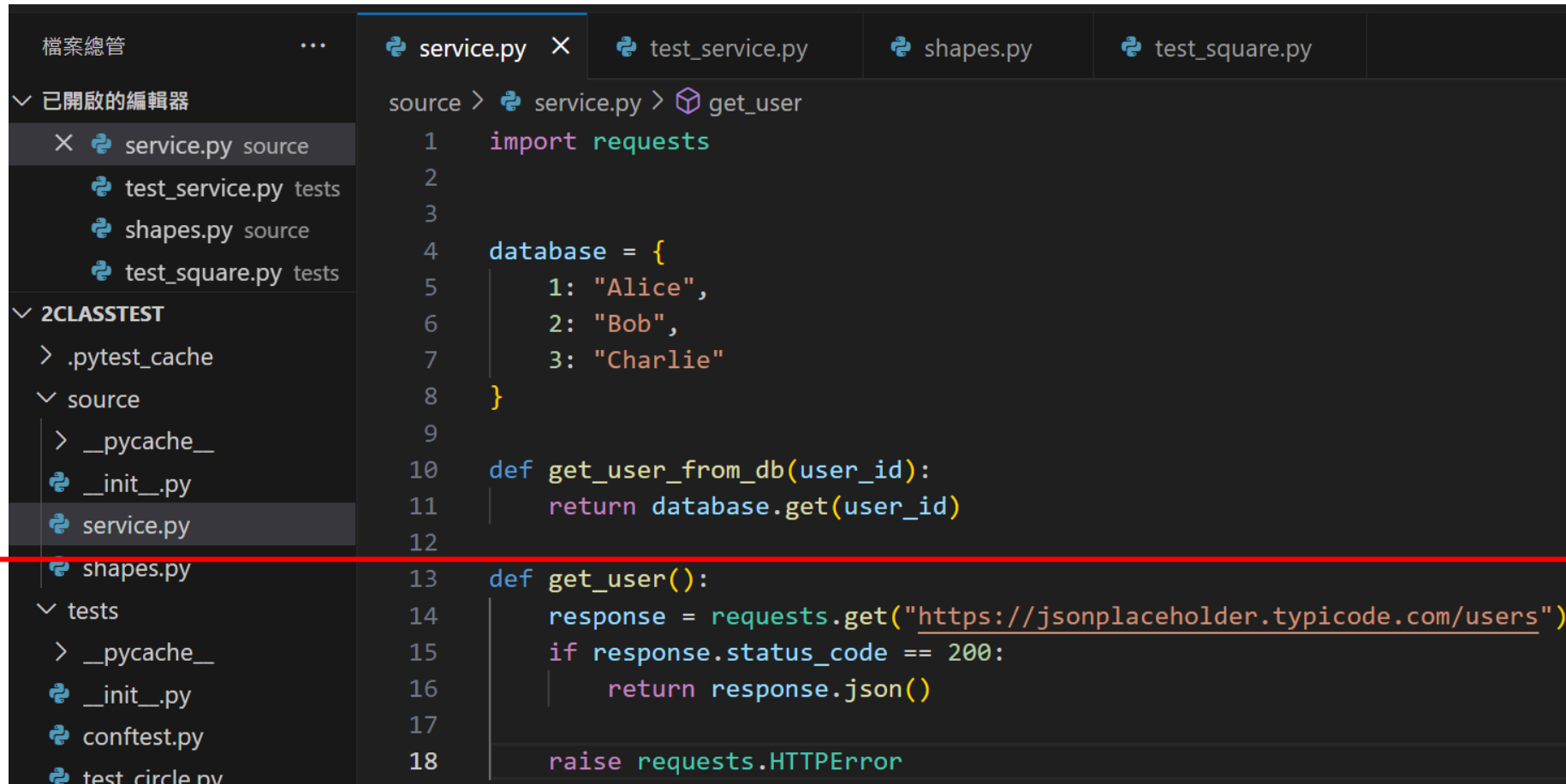
# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_service.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 1 item

tests\test_service.py . [100%]

===== 1 passed in 0.05s =====
```

# 使用 jsonplaceholder 網站API實際測試

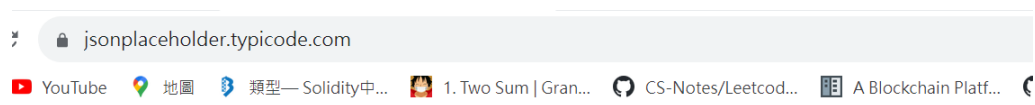


The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'source' directory containing 'service.py', 'shapes.py', and 'test\_square.py'. The code editor shows the content of 'service.py' with the following code:

```
source > service.py > get_user
1  import requests
2
3
4  database = {
5      1: "Alice",
6      2: "Bob",
7      3: "Charlie"
8  }
9
10 def get_user_from_db(user_id):
11     return database.get(user_id)
12
13 def get_user():
14     response = requests.get("https://jsonplaceholder.typicode.com/users")
15     if response.status_code == 200:
16         return response.json()
17
18     raise requests.HTTPError
```



# API產生出的範例使用者資料



## {JSON} Placeholder

Free fake API for testing and prototyping.

Powered by [JSON Server](#) + [LowDB](#). Tested with [XV](#).

Serving ~2 billion requests each month.

← → ↺ jsonplaceholder.typicode.com/users

Gmail YouTube 地圖 類型—Solidity中... 1. Two Sur

```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
  },
]
```

# 撰寫 模擬API回傳資料的測試

```
service.py  test_service.py X  shapes.py  test_square.py
tests > test_service.py > test_get_users
1  import pytest
2  import source.service as service
3  import unittest.mock as mock
4
5  @mock.patch("source.service.get_user_from_db")
6  def test_get_user_from_db(mock_get_user_from_db):
7      mock_get_user_from_db.return_value = "Mocked Alice"
8      user_name = service.get_user_from_db(1)
9
10     assert user_name == "Mocked Alice"
11
12 @mock.patch("requests.get")
13 def test_get_users(mock_get):
14     mock_response = mock.Mock()
15     mock_response.status_code = 200
16     mock_response.json.return_value = {
17         "id": 1,
18         "name": "Leanne Graham",
19         "username": "Bret",
20         "email": "Sincere@april.biz",
21         "address": {
22             "street": "Kulas Light",
23             "suite": "Apt. 556",
24             "city": "Gwenborough",
25             "zipcode": "92998-3874",
```

```
39     mock_get.return_value = mock_response
40     data = service.get_users()
41     assert data == {
42         "id": 1,
43         "name": "Leanne Graham",
44         "username": "Bret",
45         "email": "Sincere@april.biz",
46         "address": {
47             "street": "Kulas Light",
48             "suite": "Apt. 556",
49             "city": "Gwenborough",
50             "zipcode": "92998-3874",
51             "geo": {
52                 "lat": "-37.3159",
```

# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_service.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 2 items

tests\test_service.py .. [100%]

===== 2 passed in 0.31s =====
```

# 測試 Error

```
@mock.patch("requests.get")
def test_get_users_error(mock_get):
    mock_response = mock.Mock()
    mock_response.status_code = 400
    mock_get.return_value = mock_response
    with pytest.raises(requests.HTTPError):
        service.get_users()
```

# 成功

```
PS C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest> pytest .\tests\test_service.py
===== test session starts =====
platform win32 -- Python 3.10.1, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\jerry\Desktop\mastercourse\dataEngineer\pytestCourse\2classtest
collected 3 items

tests\test_service.py ... [100%]

===== 3 passed in 0.11s =====
```

**End**