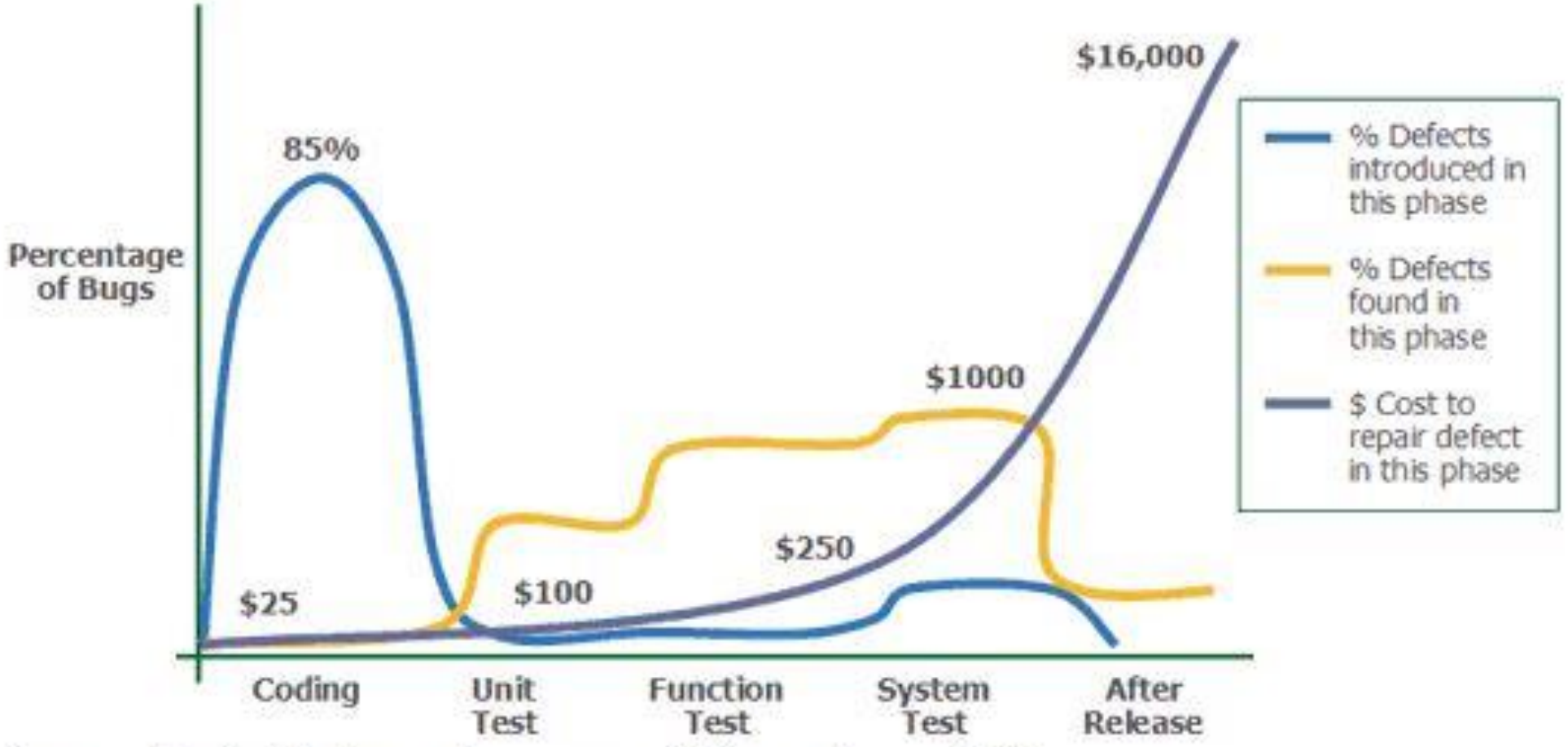
7. Test in Isolation by Stubbing the mock

給我 Test in Isolation?
幾個理由

1. 執行速度更快：

1. 單元測試通常執行速度更快，因為它們僅關注測試特定的代碼單元，而不是整個系統。這有助於在開發過程中獲得更快的反饋，並實現更高效的迭代和開發流程。

The cost of finding a bug – an evaluation
我們要測試左移



Source: *Applied Software Measurement*, Capers Jones, 1996

The COST and Difficulty of building test automation in practice

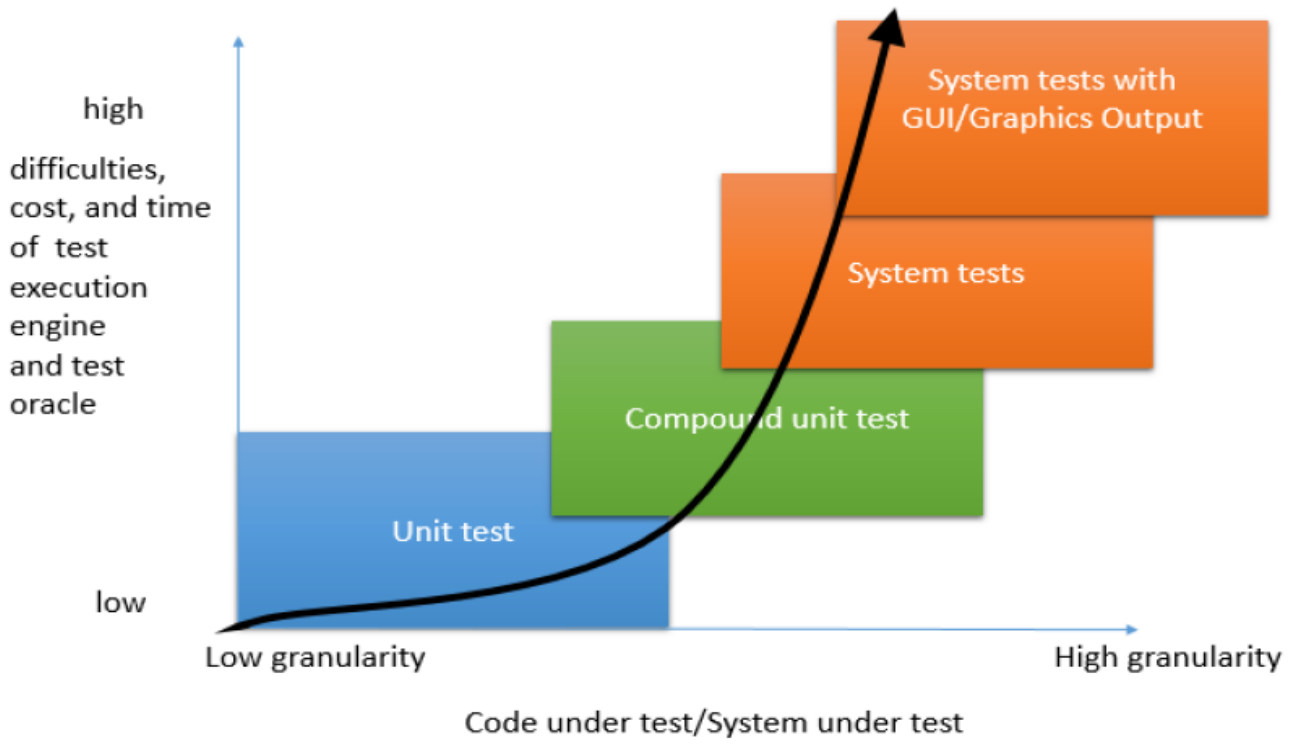


Fig. 1. The technical difficulties of test execution engine and test oracles as the granularity of code under test grows.

2. 精細的故障分析：

- 當測試失敗時，將測試隔離到特定的程式碼單元中有助於更容易識別失敗原因。通過消除必須處理整個系統的複雜性，可以更有效地找出並解決問題，從而減少找出問題的時間。

3. 減少相依性：

- 單元測試僅依賴於被測試單元的直接相依性。通過模擬或 **Stubbing** 化外部相依性（如數據庫、API或外部服務），
- 你可以避免與真實相依性進行交互時可能出現的複雜性和潛在不穩定性。這也確保測試在外部系統的可用性或狀態不變的情況下一致運行。

4. 更好的原始碼設計

- 單元測試通常鼓勵良好的原始碼設計原則，如模組化和耦合分離。編寫可測試的程式碼通常涉及創建較小、定義良好的單元，並具有清晰的邊界，從而使程式碼更易於維護、擴展和理解。

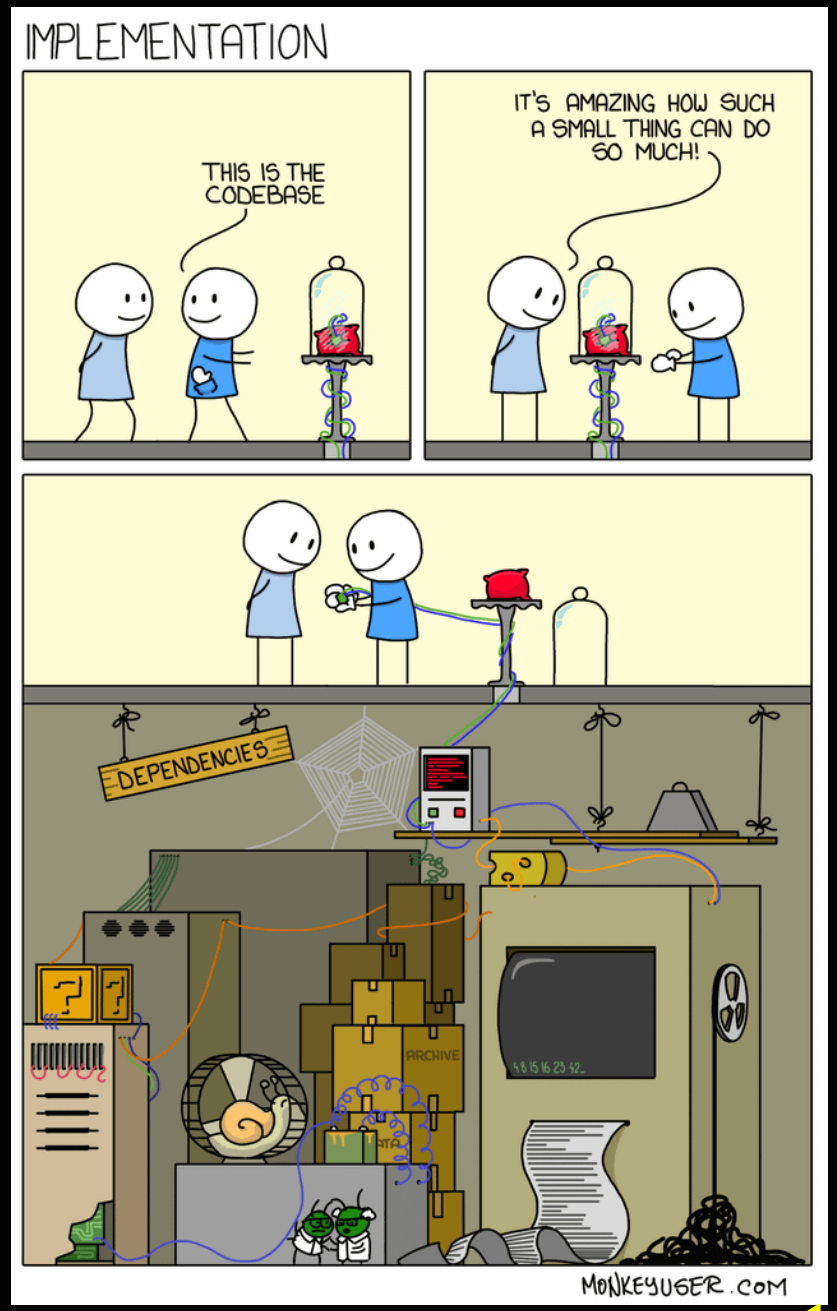
5. 預防回歸

- 單元測試為回歸測試提供了一個安全網。在進行更改或重構代碼時，運行單元測試可確保修改後的單元繼續正常運行，檢測任何意外的副作用。這有助於預防回歸並保持原始代碼庫的穩定性。
- 也就是說，團隊合作中你可以知道別人弄壞了你什麼
- 來講一下 assert (defensive programming) 的古
- **其實這是 DEVOPS 最關鍵的作為 – Quality Assurance 不開倒車**

6. 協作和擴展：

- 單元測試促進團隊成員之間的**協作**。每個開發人員可以獨立地處理自己分配的單元，以孤立方式運行測試，而不會干擾其他人的工作。
- 此外，隨著代碼庫的增長，孤立測試可實現更好的擴展性，允許在多個單元之間進行並行化(Concurrency) 和分佈式測試(Distributed Testing)。

External Dependences



3 個最常需要 mock 的 external dependencies

- mock API calls
- mock databases queries
- mock conditions difficult to generate in a test environment

Mocking/Fake Objects

Lab 7.0: 請執行 ratesign.py 並且觀察其行為

Python decorator @

Welcome | ratesign.py

ratesign.py > ...

```python
def extend_behavior(func):
    print('extended')
    return func

@extend_behavior
def some_func():
    print('hello')

print('hello world')
```

==

Welcome | ratesign.py

ratesign.py > ...

```python
def extend_behavior(func):
    print('extended')
    return func

def some_func():
    print('hello')
some_func = extend_behavior(some_func)

print('hello world')
```

```
PS D:\my-work-space\mirror-lab\lab training 2022\AUO teaching slide isolation by mock stubbing\Lab 7.0 @ Symbol> python .\ratesign.py
extended
hello world
```

多了一個輸出，詭異?

- Decorator (Design Pattern) 通常就是把自己丟給一個裝飾器，走過一些外加功能
- 然後再傳回嶄新的自己

Lab 7.0.5: 請執行 log.py 並且觀察其行為

EXPLORER

OPEN EDITORS
- Welcome
- log.py

LAB 7.0.5 @ PYCODE @ SYMBOL
- .log
- log.py

log.py > f

```python
def log(func):
    def wrapper():
        print("Started")
        func()
        print("Ended")
    return wrapper

@log
def f():
    print('hello')

f();
```

這一次讓他回傳
另外一個 function

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE powershell

```
程\7. @ Test in isolation by mock stubbing\Lab 7.0.5 @ PyCode @ Symbol>
fwd-i-search: _
PS D:\my-work-space\mirror-lab\lab training 2022\AUO teaching slides\2023 AUO 軟體測試實作課
程\7. @ Test in isolation by mock stubbing\Lab 7.0.5 @ PyCode @ Symbol> python ./log.py
Started
hello
Ended
PS D:\my-work-space\mirror-lab\lab training 2022\AUO teaching slides\2023 AUO 軟體測試實作課
程\7. @ Test in isolation by mock stubbing\Lab 7.0.5 @ PyCode @ Symbol>
```

OUTLINE
TIMELINE

Pytest fixtures 初體驗

Lab 7.1.5 : 請把 test_fixure.py 執行起來，並且觀察其行為

- Pytest 提供了 fixture 的 decorator
- Fixture 的意思就是固定好的裝飾
- 3-6 宣告了一個 input_value 的 fixture
- 所謂的 fixture 就是 test function 被執行起來的前後，都會被自動加上的動作
- 8, 11 在 test function 有參數，這個參數必須是個被**宣告過已存在**的 fixture (不是你自己圖爽亂加的參數)
- 未來當 test function 被執行起來的過程中，pytest 會先執行 test function 的參數(也就是pytest.fixture) ，然後再執行 test function 的 actual behavior

Welcome | test_fixture.py

test_fixture.py > test_divisible_by_6

```
1   import pytest
2
3   @pytest.fixture
4   def input_value():
5       input = 39
6       return input
7
8   def test_divisible_by_3(input_value):
9       assert input_value % 3 == 0
10
11  def test_divisible_by_6(input_value):
12      assert input_value % 6 == 0
```

你用 pytest,fixture
定義了一個 input_value fixture

是的你沒有看錯，本來 test function 都沒有參數的，現在有了。
有了會做什麼?

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE | powershell

```
input_value = 39

    def test_divisible_by_6(input_value):
>       assert input_value % 6 == 0
E       assert (39 % 6) == 0

test_fixture.py:12: AssertionError
========================== short test summary info ==========================
FAILED test_fixture.py::test_divisible_by_6 - assert (39 % 6) == 0
========================= 1 failed, 1 passed in 0.09s =========================
PS D:\my-work-space\mirror-lab\lab training 2022\AUO teaching slides\2023 AUO 軟體測試實作課程\8.
@Test in isolation with SpyOn\@ Lab 8.1 pytest fixture practice>
```

Fixture 是 function，通常是 test function 執行起來之前會被 pass 給 test function。可以用來餵 data 給 test function，例如資料庫連線資料，URLS....

在通曉了 Python decorator/pytest fixture 之後，我們來看看 pytest 怎麼運用 decorator 來做事情

Lab 7.1 請執行 dnd.py 以及 test_dnd.py 並且多執行幾次

python –m pytest

File Edit Selection View Go Run ···

Lab 7.1 @ PyCode Mocking a used function (not defined)

EXPLORER

OPEN EDITORS 1 unsaved

GROUP 1

Welcome

test_dnd.py

dnd.py

GROUP 2

dnd.py

LAB 7.1 @ PYCODE MOCKING A USED ...

__pycache__

.log

.pytest_cache

dnd.py

test_dnd.py

OUTLINE

TIMELINE

Welcome | test_dnd.py | dnd.py

test_dnd.py > ...

```python
from dnd import attack_damage

def test_attack_damage():
    assert attack_damage(1) == 6

```

dnd.py

dnd.py > attack_damage

```python
from random import randint

def attack_damage(modifiers):
    roll = randint(1,8)
    return modifiers + roll
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

powershell

```
n by mock stubbing\Lab 7.1 @ PyCode Mocking a used function (not defined)>
PS D:\my-work-space\mirror-lab\lab training 2022\AUO teaching slides\2023 AUO 軟體測試實作課程\7. @ Test in isolatio
n by mock stubbing\Lab 7.1 @ PyCode Mocking a used function (not defined)> python -m pytest
========================================== test session starts ==========================================
platform win32 -- Python 3.11.4, pytest-7.4.0, pluggy-1.2.0
rootdir: D:\my-work-space\mirror-lab\lab training 2022\AUO teaching slides\2023 AUO 軟體測試實作課程\7. @ Test in isolation by moc
k stubbing\Lab 7.1 @ PyCode Mocking a used function (not defined)
plugins: cov-4.1.0, mock-3.11.1
collected 1 item

test_dnd.py F                                                                                    [100%]

================================================= FAILURES =================================================
_____________________________________________ test_attack_damage _____________________________________________

    def test_attack_damage():
>       assert attack_damage(1) == 6
E       assert 2 == 6
E        +  where 2 = attack_damage(1)

test_dnd.py:4: AssertionError
========================================== short test summary info ==========================================
FAILED test_dnd.py::test_attack_damage - assert 2 == 6
```

來安裝一下 pytest-mock

dnd.py

dnd.py > ...

```python
from random import randint

def attack_damage(modifiers):
    roll = randint(1,8)
    return modifiers + roll
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE powershell

```
.2 PyCode Learning mock.patch> pip install pytest-mock
Requirement already satisfied: pytest-mock in c:\users\ypcheng\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2
cache\local-packages\python311\site-packages (3.11.1)
Requirement already satisfied: pytest>=5.0 in c:\users\ypcheng\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2
cache\local-packages\python311\site-packages (from pytest-mock) (7.4.0)
Requirement already satisfied: iniconfig in c:\users\ypcheng\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kf
che\local-packages\python311\site-packages (from pytest>=5.0->pytest-mock) (2.0.0)
Requirement already satisfied: packaging in c:\users\ypcheng\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kf
che\local-packages\python311\site-packages (from pytest>=5.0->pytest-mock) (23.1)
Requirement already satisfied: pluggy<2.0,>=0.12 in c:\users\ypcheng\appdata\local\packages\pythonsoftwarefoundation.python.3.11_
\localcache\local-packages\python311\site-packages (from pytest>=5.0->pytest-mock) (1.2.0)
Requirement already satisfied: colorama in c:\users\ypcheng\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfr
he\local-packages\python311\site-packages (from pytest>=5.0->pytest-mock) (0.4.6)

[notice] A new release of pip is available: 23.1.2 -> 23.2.1
[notice] To update, run: C:\Users\ypcheng\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0
 pip install --upgrade pip
PS D:\my-work-space\mirror-lab\lab training 2022\AUO teaching slides\2023 AUO 軟體測試實作課程\7. @ Test in isolation by mock st
```

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.11.4 64-bit (micros

pytest-mock

This pytest plugin provides a `mocker` fixture which is a thin-wrapper around the patching API provided by the mock package:

```
import os

class UnixFS:

    @staticmethod
    def rm(filename):
        os.remove(filename)

def test_unix_fs(mocker):
    mocker.patch('os.remove')
    UnixFS.rm('file')
    os.remove.assert_called_once_with('file')
```

Besides undoing the mocking automatically after the end of the test, it also provides other nice utilities such as `spy` and `stub`, and uses pytest introspection when comparing calls.

Install

Install using pip:

```
$ pip install pytest-mock
```

CONTENTS:

Lab 7.2 請將 test_dnd.py 改成下圖，並且執行 pytest

test_dnd.py ● dnd.py

test_dnd.py > ...

```python
from dnd import attack_damage
from unittest import mock

@mock.patch("dnd.randint", return_value=5, autospec=True)
def test_attack_damage(mock_randint):
    assert attack_damage(1) == 6
```

dnd.py

dnd.py > attack_damage

```python
from random import randint

def attack_damage(modifiers):
    roll = randint(1,8)
    return modifiers + roll
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE powershell

```
PS D:\my-work-space\mirror-lab\lab training 2022\AUO teaching slides\2023 AUO 軟體測試實作課程\7. @ Test in isolation by mock stubbing\Lab 7.2 @
 PyCode Learning mock.patch> python -m pytest
================================= test session starts =================================
platform win32 -- Python 3.11.4, pytest-7.4.0, pluggy-1.2.0
rootdir: D:\my-work-space\mirror-lab\lab training 2022\AUO teaching slides\2023 AUO 軟體測試實作課程\7. @ Test in isolation by mock stubbing\Lab
 7.2 @ PyCode Learning mock.patch
plugins: cov-4.1.0, mock-3.11.1
collected 1 item

test_dnd.py .                                                            [100%]

================================== 1 passed in 0.02s ==================================
PS D:\my-work-space\mirror-lab\lab training 2022\AUO teaching slides\2023 AUO 軟體測試實作課程\7. @ Test in isolation by mock stubbing\Lab 7.2 @
 PyCode Learning mock.patch>
```

@mock.patch 幹了什麼事?

- mock.path 是 pytest 提供的 decorator
- **它會把下面的 test_attack_damange 吃進去攪和**
- **你可以想像，它做的事情就是程式碼替代**
- mock .path 利用 decorator 的技巧把 attack_damage 裡面用到的
- randint function 都直接改成回傳 5
- randint 叫做被 stubbed 的 function
- randint 的 wrapper 叫做 test stubs (你也可以叫 mock，現在很難分清楚)

```python
from dnd import attack_damage
from unittest import mock

@mock.patch("dnd.randint", return_value=5, autospec=True)
def test_attack_damage(mock_randint):
    assert attack_damage(1) == 6

```

mock 利用 decorator 的技巧把 randint 的 wrapper 回傳回來
讓你未來能夠運用

Welcome | ratesign.py

ratesign.py > ...

```python
def extend_behavior(func):
    print('extended')
    return func

@extend_behavior
def some_func():
    print('hello')

print('hello world')
```

想像中的 stubbed by MagicMock

dnd.py > attack_damage

```
1   from random import randint
2
3   def attack_damage(modifiers):
4       roll = MagicMock()
5       return modifiers + roll
```

常見的誤區. 很容易混淆. 觀念的釐清

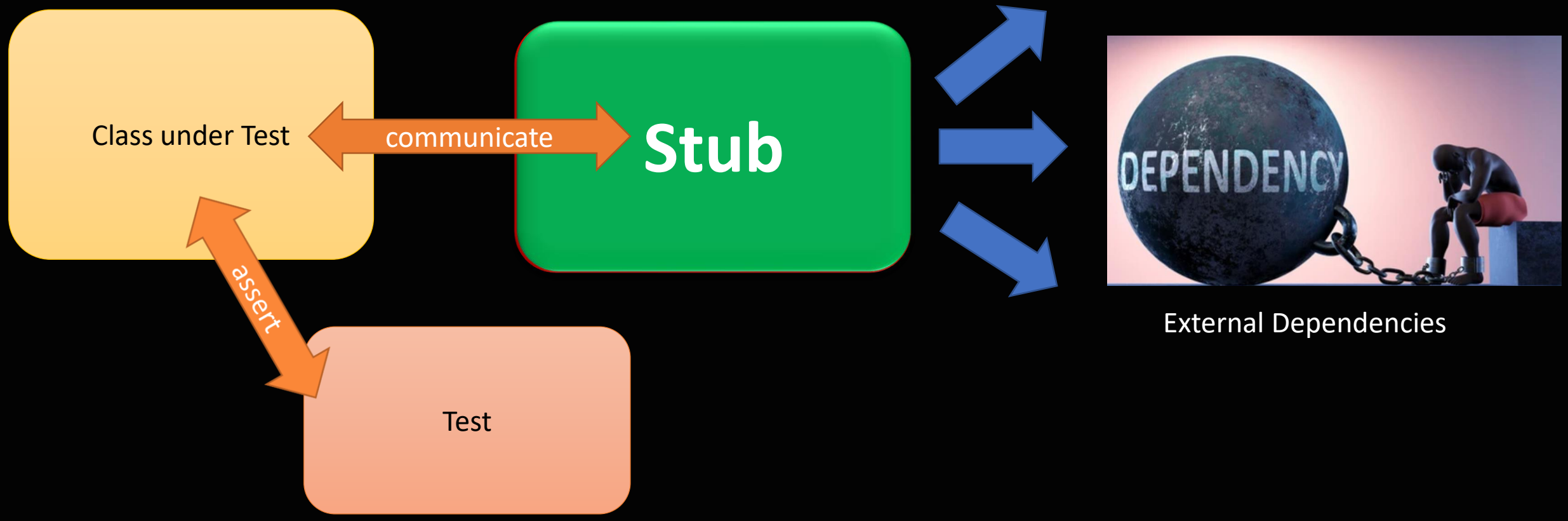
錯 !!! 不能這麼用
Mock 的 decorator 的 wrapper 要包裝的不是 stubbed function 被定義的地方，而是被叫用的地方。還記得 decorator 裝飾的是 test_attack_damage

"Mock where the object is used"

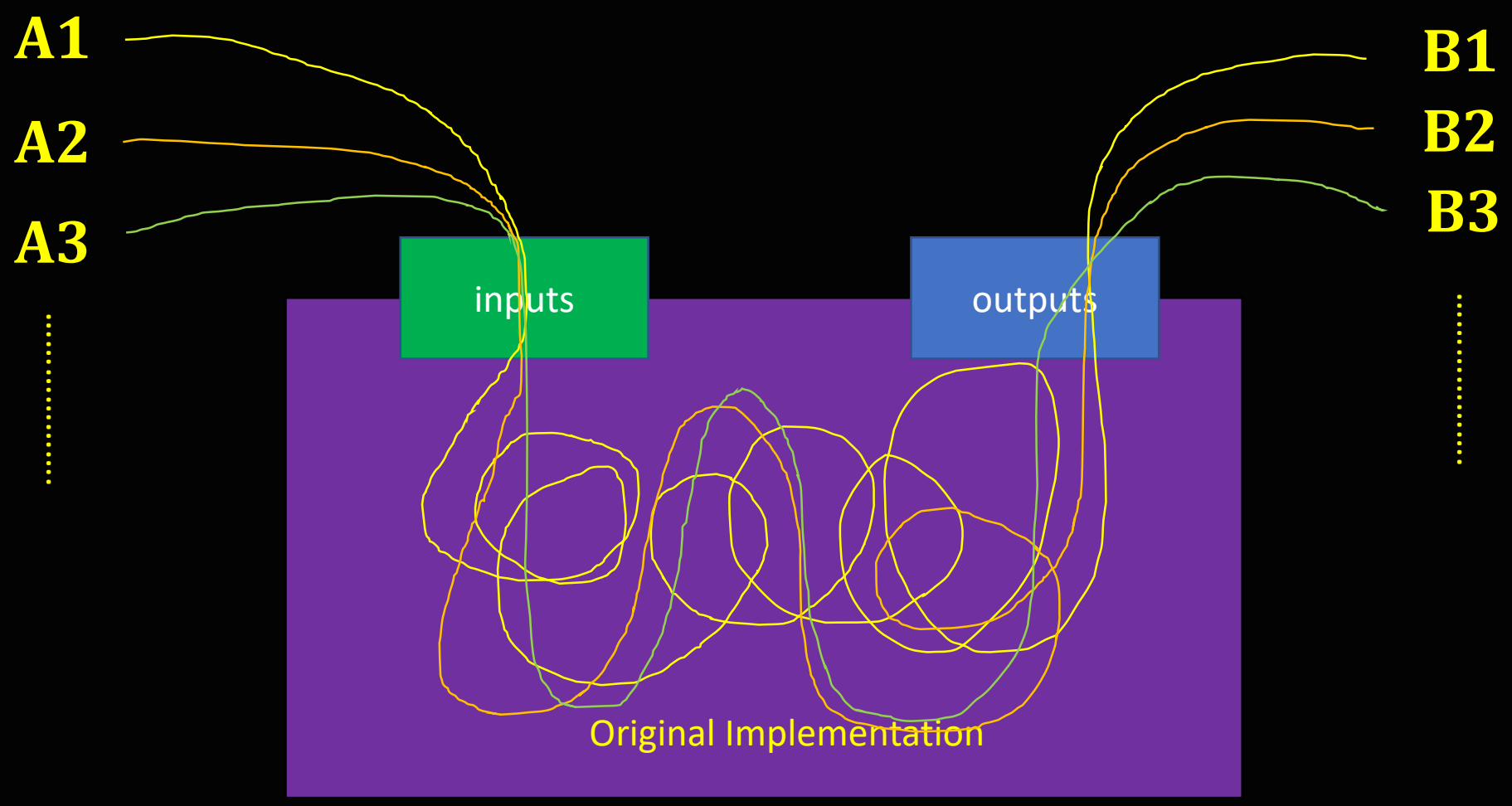
```
1  from dnd import attack_damage
2  from unittest import mock
3
4  @mock.patch("random.randint", return_value=5, autospec=True)
5  def test_attack_damage(mock_randint):
6      assert attack_damage(1) == 6
7
8
```

來講一下Stubbing 的原理

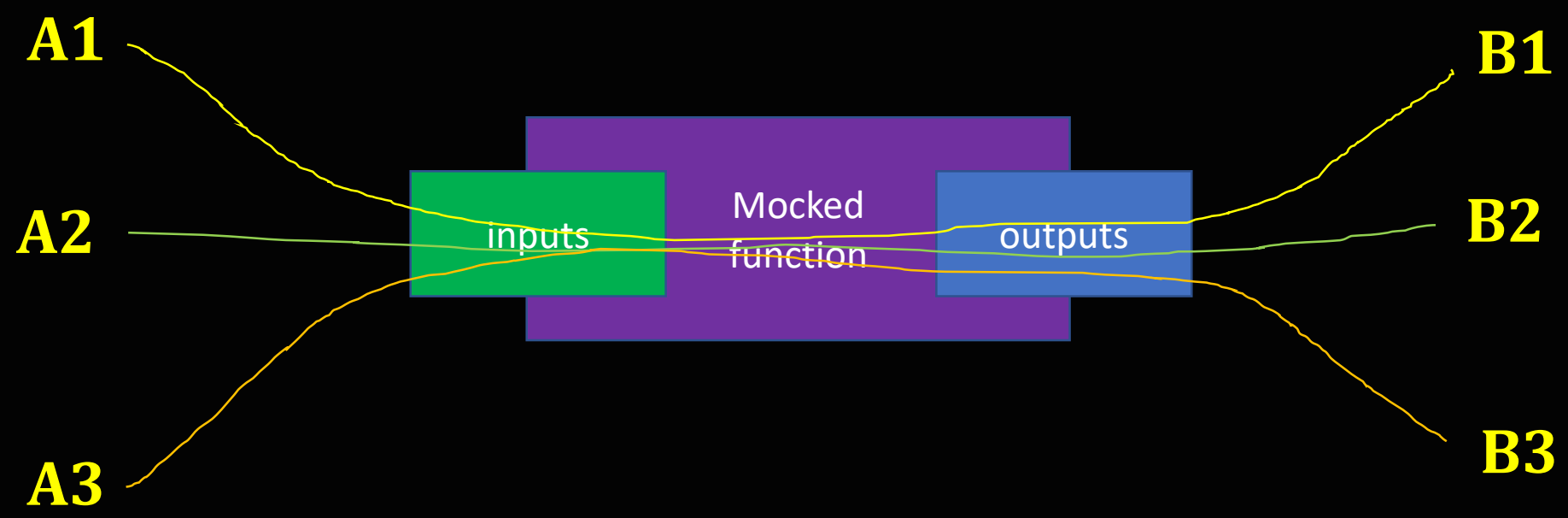
Introducing a Stub



A stub can never fail the test


A1
A2
A3
inputs
outputs
B1
B2
B3
Original Implementation

Replace Original Implementation by a fake/stubbed/mocked function



我們來好好探索一下

Lab 7.3: 給你下一頁的例子。data_fetcher.py & data_processor.py

File Edit Selection View Go Run ··· Lab 7.3 @ PyCode data_processors

EXPLORER

OPEN EDITORS
- GROUP 1
  - Welcome
  - data_fetcher.py
  - data_processor.py
  - test_data_processor.py
- GROUP 2
  - data_fetcher.py
  - data_processor.py

LAB 7.3 @ PYCODE DATA_PROCESSORS
- __pycache__
- .log
- .pytest_cache
- data_fetcher.py
- data_processor.py
- test_data_processor.py

OUTLINE
TIMELINE

Welcome | data_fetcher.py | data_processor.py

data_fetcher.py > ...

```python
# data_fetcher.py

import requests

def fetch_data_from_api(url):
    response = requests.get(url)
    if response.status_code == 200:
        return response.json()
    return None
```

這個 http request
是個 external dependence
呼叫要錢!!

如果這個 http request 成功
他會回傳類似 [1, 2, 3]

如果這個 http request 不成功
它會回傳 None

data_fetcher.py | data_processor.py

data_processor.py

```python
# data_processor.py

from data_fetcher import fetch_data_from_api

def process_data(data):
    # Some data processing logic, for example, s
    return sum(data)

def get_and_process_data(url):
    data = fetch_data_from_api(url)
    if data:
        return process_data(data)
    return None
```

真實案例的呼叫會叫用
https://example.com/api/data

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE powershell

如果你想解決編譯問題
請安裝 pip install requests

```
ubbing\Lab 7.3 @ PyCode data_processors> pip install requests
Requirement already satisfied: requests in c:\users\ypcheng\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\ypcheng\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (from requests) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\ypcheng\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (from requests) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\ypcheng\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (from requests) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\ypcheng\appdata\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages (from requests) (2023.7.22)

[notice] A new release of pip is available: 23.1.2 -> 23.2.1
[notice] To update, run: C:\Users\ypcheng\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\python.exe -m pip install --upgrade pip
PS D:\my-work-space\mirror-lab\lab training 2022\AUO teaching slides\2023 AUO 軟體測試實作課程\7. @ Test in isolation by mock st
ubbing\Lab 7.3 @ PyCode data_processors>
```

Lab 7.3: TASK TO DO
請你寫兩個 test methods

1. 使用 mock.patch 把 fetch_data_from_api 包裝起來
2. test method 中要叫用 get_and_process_data("https://example.com/api/data")
3. 請預設這個 mocked function 會回傳 [1,2,3]
4. 其中一個 test method 請 assert 結果為 6
5. 其中一個 test method 請 assert 結果為 None 的情境

請上傳 test_data_processor.py 助教指示的方式

另一種使用 context manager 的寫法

- 這種狀況我們就可以改用 with 的寫法：

```
# 以 with 開啟檔案
with open(filename) as f:
  # ...
```

- 這裡在使用 with 開啟檔案時，會將開啟的檔案一樣放在 f 變數中，但是這個 f 只有在這個 with 的範圍內可以使用，而離開這個範圍時 f 就會自動被關閉，回收相關的資源。

```
# 以 with 開檔並寫入檔案
with open('file.txt', 'w') as f:
  f.write('Hello, world!')
```

- 使用 with 的話，檔案使用完之後就會自動關閉，方便很多。

Comparisons

- Use decorator

```
@mock.path('xxx.yyy', return_value=???)
def test_iamagoodstudent(mock_yyy);
```

```
6   @mock.patch('data_processor.fetch_data_from_api', return_value=[1, 2, 3])
7   def test_get_and_process_data(mock_fetch):
```

- Use context manager

```
def test_iamagoodstudent():
        with mock.patch('xxx.yyy', return_value=???) as mock_yyy:
```

```
7   def test_get_and_process_data():
8       # Mocking the fetch_data_from_api function with mock data [1, 2, 3]
9       with mock.patch('data_processor.fetch_data_from_api', return_value=[1, 2, 3]) as mock_fetch:
```

Mock a class

Homework/Lab 7.5: 給你下一頁的例子。
data_access.py & data_processor.py

Let's mocking a class !!

File Edit Selection View Go Run ··· problem

EXPLORER

OPEN EDITORS

GROUP 1

data_processor.py

GROUP 2

data_processor.py

data_access.py

PROBLEM

.log

data_access.py

data_processor.py

data_processor.py > ...

```python
# data_processor.py

from data_access import Database

def process_and_store_data(value):
    db = Database()
    db.insert_data(value)
    all_data = db.get_all_data()
    return sum(all_data)

```

data_access.py > ...

```python
# data_access.py

class Database:
    def __init__(self):
        # Simulate a database connection or setup
        self.data = []

    def insert_data(self, value):
        # Simulate inserting data into the database
        self.data.append(value)

    def get_all_data(self):
        # Simulate fetching all data from the database
        return self.data

```

Code Pattern 範例來使用 class patch mocker

XXX 是 class 被初始化的地方

你想要 mock 的 Class 名稱

test_data_processor.py 2

D: > my-work-space > mirror-lab > lab training 2022 > AUO teaching slides > 2023 AUO 軟體測試實作課程

```python
# test_data_processor.py

import pytest
from unittest.mock import patch
from data_processor import process_and_store_data

@patch('data_processor.Database')
def test_process_and_store_data(MockDatabase):
    # Create a mock instance of the Database class
    mock_db_instance = MockDatabase.return_value

    # Define the return values for the mock methods
    mock_db_instance.get_all_data.return_value = [1, 2, 3]
```

Pytest-mock 回傳一個 mock 給你

正常你建立一個物件是用 xxx()
呼叫一個 method 也是用 yyy()
但是 mock 的叫用就是回傳 return_value

HOMEWORK/Lab 7.5 : TASK TO DO

1. 在這個例子中，真實的 DB讀寫是external dependency
2. 請撰寫一個 test function 練習 mock Database
   hint: @mock.patch('data_processor.Database')

3. mock get_all_data 讓它在測試中回傳 [1,2,3]
4. 呼叫 process_and_store_data 並且 assert 結果是 6
5. 請想一下 db.insert_data(value) 需要理會它嗎? (我們下回繼續)

請依照助教指示上傳

1. 你的 test_data_processor.py