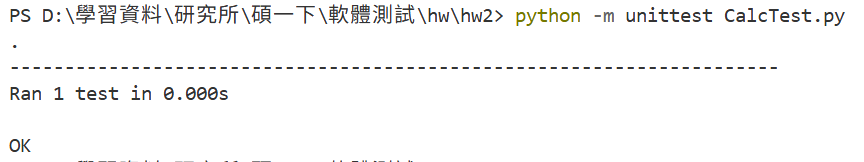
**Homework 2: Practice TDD by adding functions to Calc**

313552054 網工所碩一 陳妍嘉

**1.**

**Calc currently implements one function: it adds two integers. Use test-driven design to add additional functionality to subtract two integers, multiply two integers, and divide two integers. First, create a failing test for one of the new functionalities,** **modify the class until the test passes, and** **perform any necessary refactoring. Repeat until all required functionality has been added to your latest version of Calc and all tests pass.   
  
Remember that in TDD, the tests determine the requirements. Before modifying the software, you must encode decisions such as whether the division method returns an integer or a floating point number in automated tests. Submit printouts of all tests, your** **final version of Calc, and a screenshot showing that all tests pass. Most importantly, include a narrative describing each TDD test created, the changes needed to make it pass, and any necessary refactoring. You can use any programming language to develop Calc.  We have attached Java, JS, and Python Calc implementations.   
  
Please find the attached file with the full specifications. Optional exercises are also in the attached CalcTDD.pdf.**

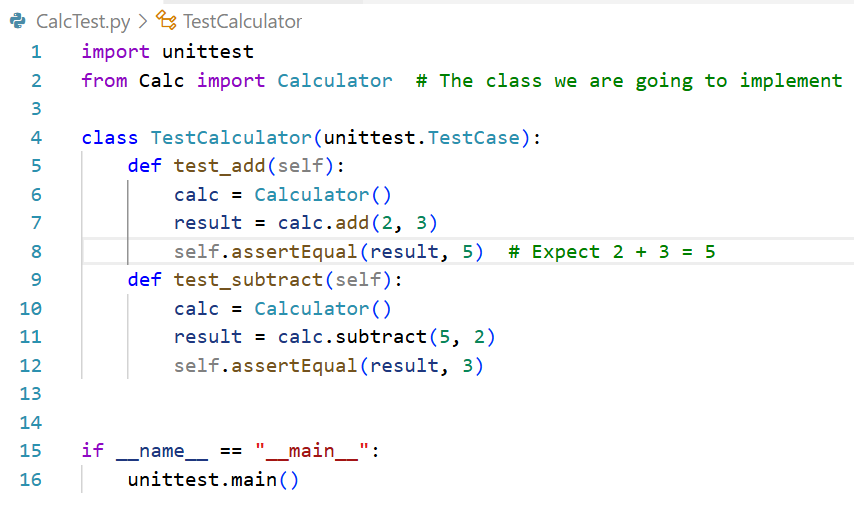
**初始跑測試:**



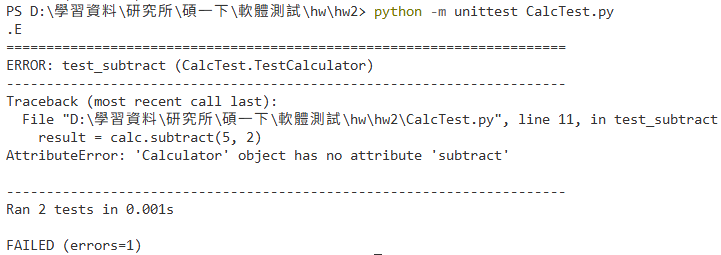
* 實作減法

步驟一：create a failing test for one of the new functionalities:

我首先撰寫測試 test\_subtract()，期望 5 - 2 = 3。

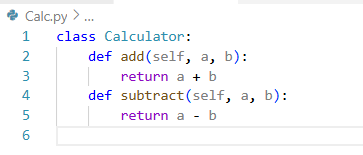


執行後失敗(結果如下圖)，因為 Calculator 尚未有此方法。

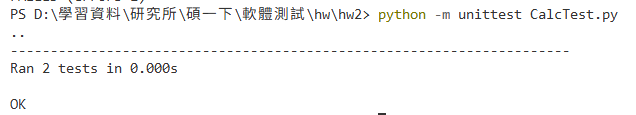


步驟二：modify the class until the test passes

在 Calc.py 中新增 subtract(self, a, b) 並回傳 a - b



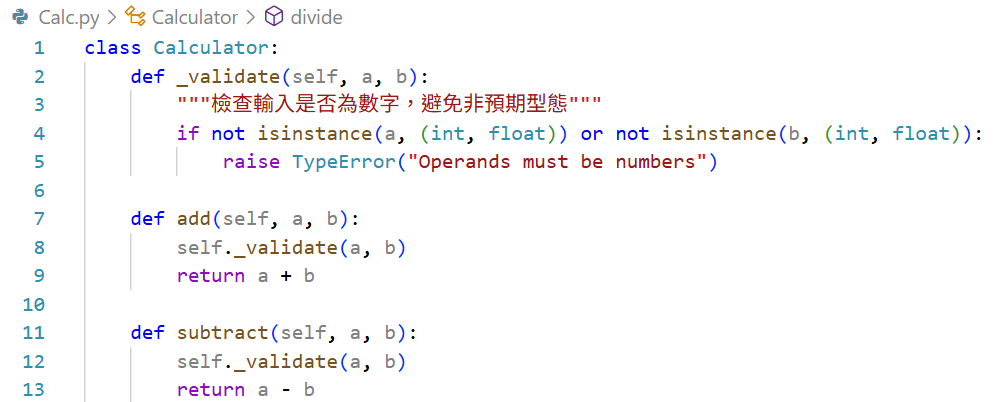
執行測試後通過:



步驟三：perform any necessary refactoring：

在確認減法功能正確後，我觀察到「加法」與「減法」結構完全相同，僅差在運算符號。

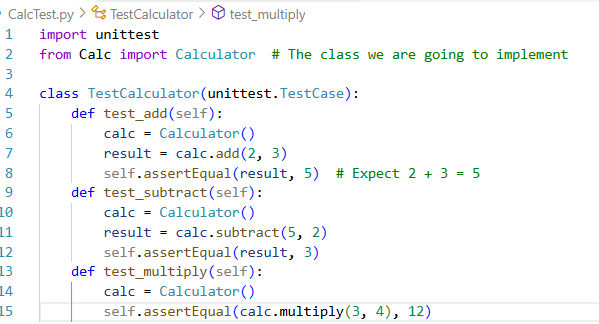
因此，我進行第一次重構（Refactor），加入共用驗證函式 \_validate()，確保輸入皆為數值類型(整數或浮點數皆可)，並統一檢查邏輯，避免重複程式碼，並確認測試通過。



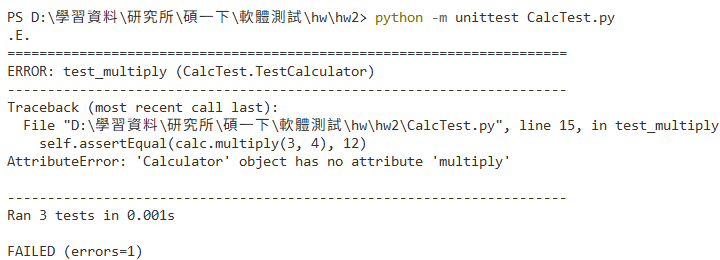
* 實作乘法

步驟一：create a failing test for one of the new functionalities:

同樣先新增測試 test\_multiply() 驗證 3 × 4 = 12

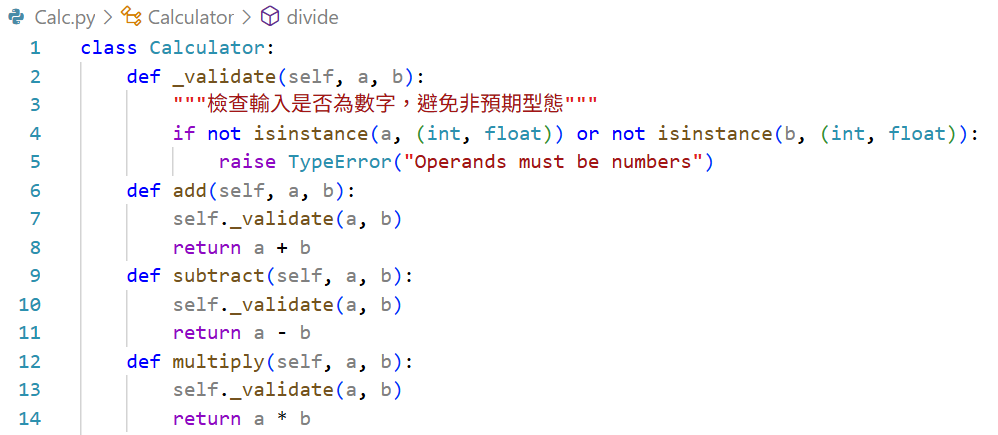


執行測試後失敗:

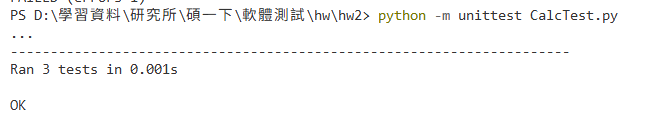


步驟二：modify the class until the test passes

測試失敗後補上 multiply() 實作



執行測試通過:



步驟三：perform any necessary refactoring：

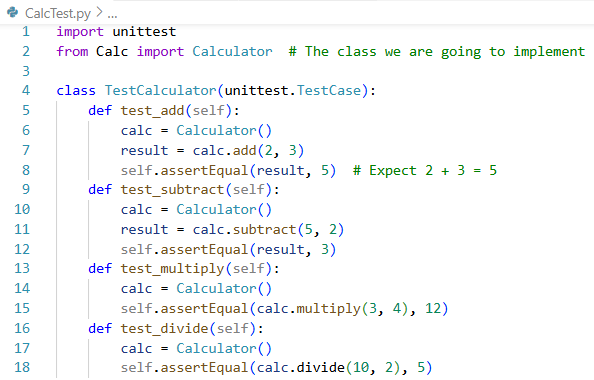
程式在步驟二就保持上一個實作時重構的架構，已經加入共用驗證函式 \_validate()，確保輸入皆為數值類型。

因此經過檢查後，發現無須重構。

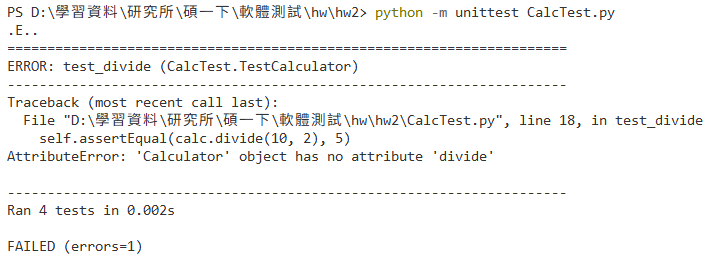
* 實作除法

步驟一：create a failing test for one of the new functionalities:

撰寫 test\_divide() 驗證 10 ÷ 2 = 5。

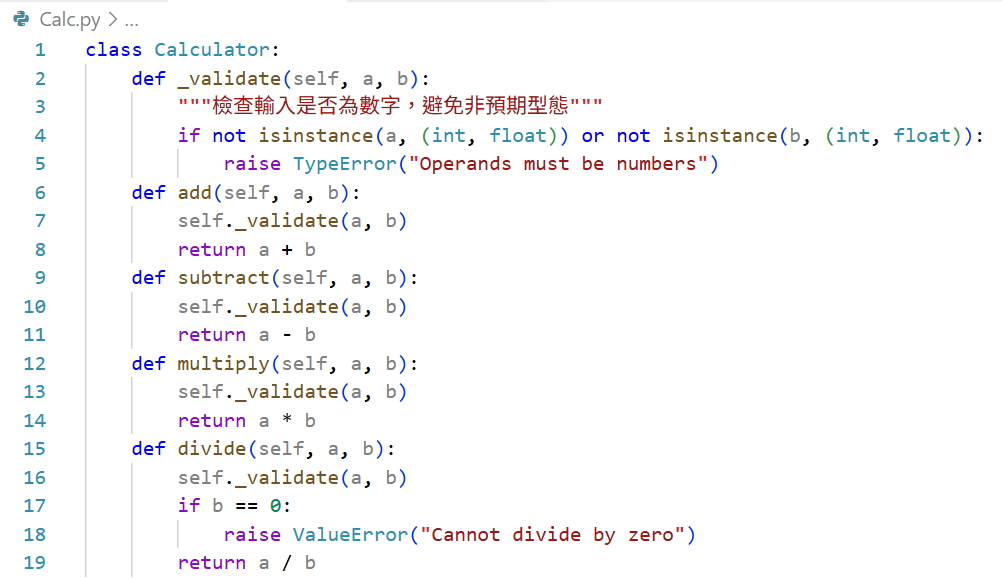


執行測試:

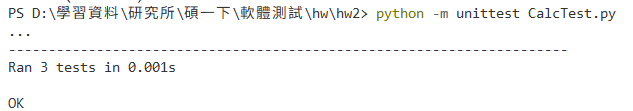


步驟二：modify the class until the test passes

新增 divide() 函式，我選擇讓除法回傳浮點數並在除數為零時拋出例外。



完成後所有測試皆通過:



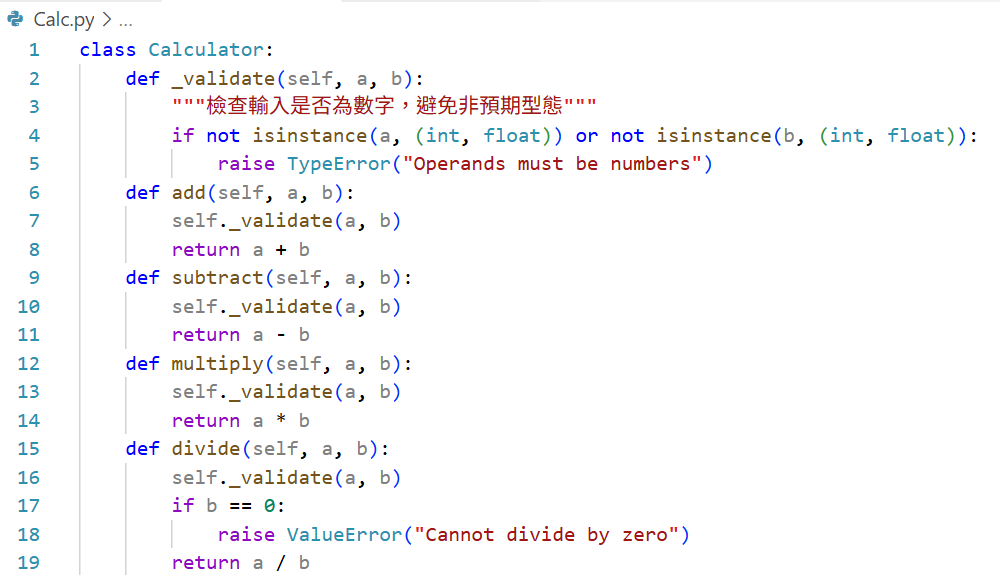
步驟三：perform any necessary refactoring：

程式在步驟二時就保持在減法實作時重構的架構，已經加入共用驗證函式 \_validate()，確保輸入皆為數值類型。

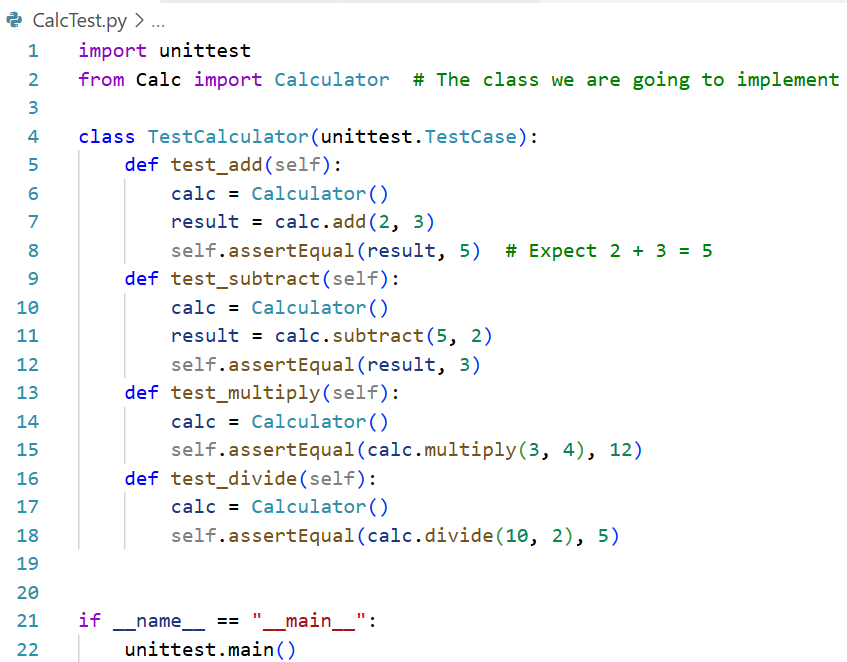
因此經過檢查後，發現無須重構。

最終成果

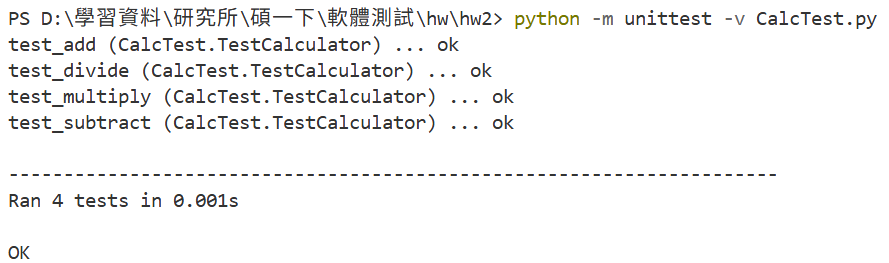
最終版本的Calc.py程式如下(final version of Calc)：



測試程式碼CalcTest.py如下：



執行測試python -m unittest -v CalcTest.py，結果如下：

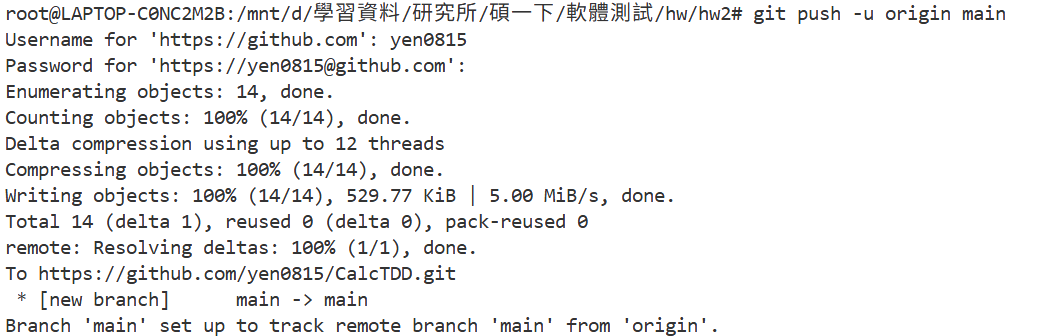


所有測試皆通過

2.

**(Optional) You can use a continuous integration setup in GitHub. Include version control for both source code and tests and populate both with a simple example. Experiment with “breaking the build” by either introducing a fault into the source code or adding a failing test case. Restore the build.**

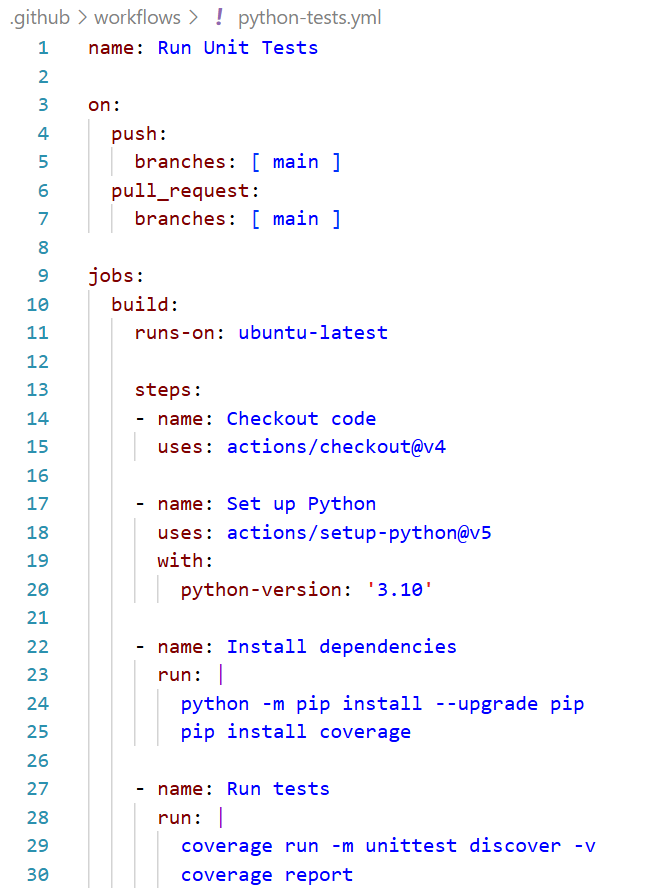
建立 GitHub Repo，連結為<https://github.com/yen0815/CalcTDD.git>：



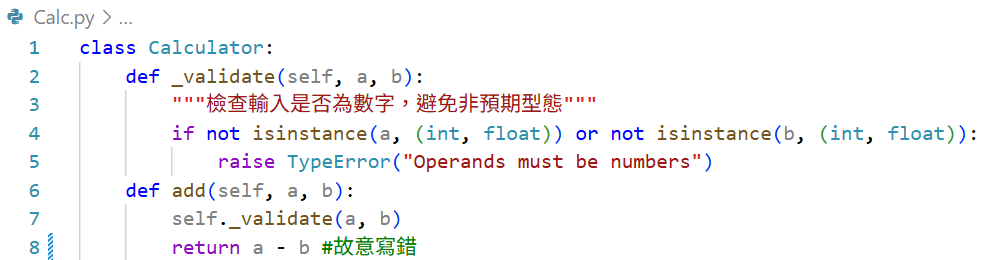
新增 CI Workflow：

在專案中建立資料夾： .github/workflows/

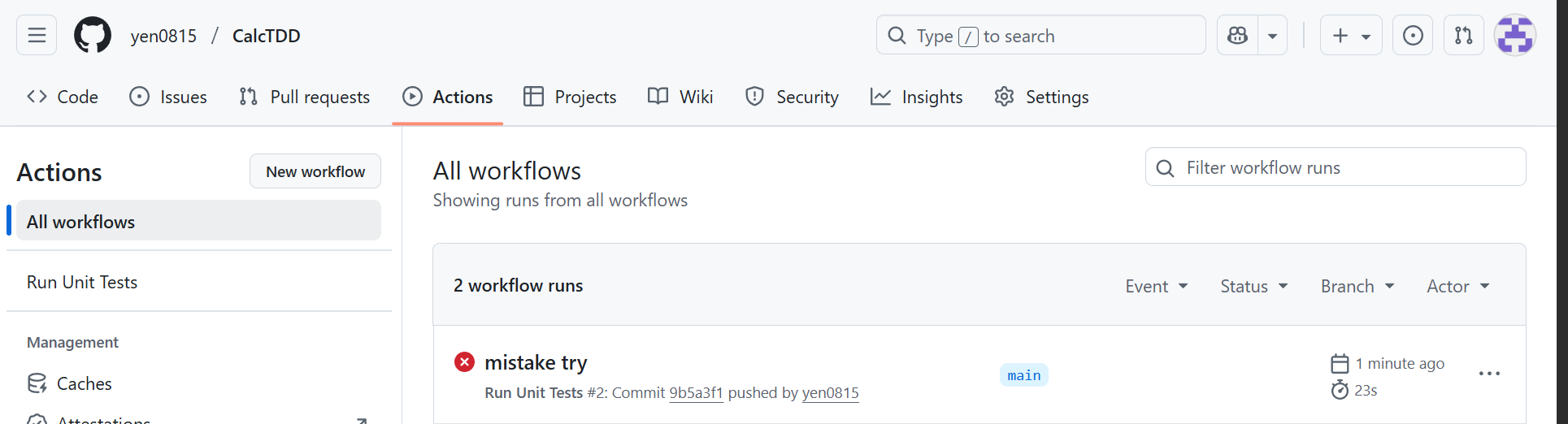
新增一個檔案：python-tests.yml，並推上GitHub



故意在 Calc.py 中引入錯誤，並推上GitHub：



會看到 GitHub Actions 的測試失敗：



修回原本的正確程式，再次 push → 測試恢復成功

