Giới thiệu về lập trình với Python của CS50

OpenCourseWare

Quyên tặng (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan) (https://www.threads.net/@davidjmalan)
(https://twitter.com/davidjmalan)

Bài giảng 5

Kiểm tra đơn vị

- Cho đến bây giờ, bạn có thể đã kiểm tra mã của riêng mình bằng cách sử dụng `print` các câu lệnh.
- Ngoài ra, bạn có thể đã dựa vào CS50 để kiểm tra mã cho mình!
- Việc viết mã để kiểm tra chương trình của riêng bạn là điều phổ biến nhất trong ngành.
- Trong cửa sổ bảng điều khiển của bạn, nhập `code calculator.py`. Lưu ý rằng trước đây bạn có thể đã mã hóa tệp này trong bài giảng trước. Trong trình soạn thảo văn bản, hãy đảm bảo rằng mã của bạn xuất hiện như sau:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))
```

```
def square(n):
    return n * n


if __name__ == "__main__":
    main()
```

Lưu ý rằng bạn có thể tự mình kiểm tra đoạn mã trên một cách hợp lý bằng cách sử dụng một số con số rõ ràng như `2`. Tuy nhiên, hãy xem xét lý do tại sao bạn có thể muốn tạo một thử nghiệm để đảm bảo rằng đoạn mã trên hoạt động phù hợp.

- Theo quy ước, hãy tạo một chương trình thử nghiệm mới bằng cách nhập `code test_calculator.py` và sửa đổi mã của bạn trong trình soạn thảo văn bản như sau:

```
from calculator import square


def main():
    test_square()


def test_square():
    if square(2) != 4:
        print("2 squared was not 4")
    if square(3) != 9:
        print("3 squared was not 9")


if __name__ == "__main__":
    main()
```

Lưu ý rằng chúng ta đang nhập `square` hàm từ `square.py` dòng mã đầu tiên. Theo quy ước, chúng ta tạo một hàm có tên là `test_square`. Bên trong hàm đó, chúng ta xác định một số điều kiện để kiểm tra.

- Trong cửa sổ bảng điều khiển, gõ `python test_calculator.py`. Bạn sẽ nhận thấy rằng không có gì được xuất ra. Có thể là mọi thứ đều ổn! Ngoài ra, có thể chức năng kiểm tra của chúng tôi đã không phát hiện ra một trong những "trường hợp góc" có thể gây ra lỗi.
- Hiện tại, mã của chúng tôi đang kiểm tra hai điều kiện. Nếu chúng tôi muốn kiểm tra nhiều điều kiện hơn, mã kiểm tra của chúng tôi có thể dễ dàng bị cồng kềnh. Làm cách nào chúng tôi có thể mở rộng khả năng thử nghiệm của mình mà không cần mở rộng mã thử nghiệm?

`assert`

- Lệnh của Python `assert` cho phép chúng ta nói với trình biên dịch rằng điều gì đó, một khẳng định nào đó, là đúng. Chúng ta có thể áp dụng điều này cho mã thử nghiệm của mình như sau:

```
from calculator import square


def main():
    test_square()


def test_square():
    assert square(2) == 4
    assert square(3) == 9


if __name__ == "__main__":
    main()
```

Lưu ý rằng chúng ta đang khẳng định chắc chắn cái gì `square(2)` và `square(3)` nên bằng nhau. Mã của chúng tôi giảm từ bốn dòng thử nghiệm xuống còn hai.

- Chúng ta có thể cố ý phá vỡ mã máy tính của mình bằng cách sửa đổi nó như sau:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n + n


if __name__ == "__main__":
    main()
```

Lưu ý rằng chúng ta đã thay đổi `*` toán tử thành a `+` trong hàm bình phương.

- Bây giờ đang chạy `python test_square.py` trong cửa sổ giao diện điều khiển, bạn sẽ nhận thấy rằng trình `AssertionError` biên dịch được đưa ra. Về cơ bản, đây là trình biên dịch cho chúng ta biết rằng một trong những điều kiện của chúng ta không được đáp ứng.
- Một trong những thách thức mà chúng tôi hiện đang phải đối mặt là mã của chúng tôi có thể trở nên nặng nề hơn nếu chúng tôi muốn cung cấp nhiều kết quả lỗi mô tả hơn cho người dùng của mình. Rất hợp lý, chúng ta có thể viết mã như sau:

```
from calculator import square


def main():
    test_square()


def test_square():
    try:
        assert square(2) == 4
    except AssertionError:
```

```
        print("2 squared is not 4")
    try:
        assert square(3) == 9
    except AssertionError:
        print("3 squared is not 9")
    try:
        assert square(-2) == 4
    except AssertionError:
        print("-2 squared is not 4")
    try:
        assert square(-3) == 9
    except AssertionError:
        print("-3 squared is not 9")
    try:
        assert square(0) == 0
    except AssertionError:
        print("0 squared is not 0")


if __name__ == "__main__":
    main()
```

Lưu ý rằng việc chạy mã này sẽ tạo ra nhiều lỗi. Tuy nhiên, nó không tạo ra tất cả các lỗi trên. Đây là một minh họa rõ ràng rằng đáng để thử nghiệm nhiều trường hợp để bạn có thể nắm bắt được các tình huống có lỗi mã hóa.

- Đoạn mã trên minh họa một thách thức lớn: Làm cách nào chúng tôi có thể giúp việc kiểm tra mã của bạn trở nên dễ dàng hơn mà không cần hàng tá dòng mã như trên?

Bạn có thể tìm hiểu thêm trong tài liệu của Python về `assert` (https://docs.python.org/3/reference/simple_stmts.html#assert).

`pytest`

- `pytest` là thư viện của bên thứ ba cho phép bạn kiểm tra đơn vị chương trình của mình. Nghĩa là, bạn có thể kiểm tra các chức năng trong chương trình của mình.
- Để sử dụng `pytest` xin vui lòng gõ `pip install pytest` vào cửa sổ giao diện điều khiển của bạn.
- Trước khi áp dụng `pytest` vào chương trình của chúng tôi, hãy sửa đổi `test_calculator` chức năng của bạn như sau:

```
from calculator import square


def test_assert():
    assert square(2) == 4
    assert square(3) == 9
    assert square(-2) == 4
```

```
    assert square(-3) == 9
    assert square(0) == 0
```

Lưu ý cách đoạn mã trên xác nhận tất cả các điều kiện mà chúng ta muốn kiểm tra.

- `pytest` cho phép chúng tôi chạy chương trình của mình trực tiếp thông qua nó, để chúng tôi có thể dễ dàng xem kết quả của các điều kiện thử nghiệm hơn.
- Trong cửa sổ terminal, gõ `pytest test_calculator.py`. Bạn sẽ nhận thấy ngay rằng đầu ra sẽ được cung cấp. Hãy chú ý đến màu đỏ `F` ở gần đầu kết quả đầu ra, cho biết có điều gì đó trong mã của bạn bị lỗi. Hơn nữa, hãy lưu ý rằng màu đỏ `E` cung cấp một số gợi ý về lỗi trong `calculator.py` chương trình của bạn. Dựa trên kết quả đầu ra, bạn có thể tưởng tượng một kịch bản `3 * 3` được xuất ra `6` thay vì `9`. Dựa trên kết quả của thử nghiệm này, chúng tôi có thể sửa `calculator.py` mã của mình như sau:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n * n


if __name__ == "__main__":
    main()
```

Lưu ý rằng chúng ta đã thay đổi `+` toán tử thành hàm a `*` trong hàm bình phương, đưa nó về trạng thái hoạt động.

- Chạy lại `pytest test_calculator.py`, chú ý không có lỗi nào được tạo ra. Chúc mừng!
- Hiện tại, việc `pytest` dừng chạy sau lần thử nghiệm thất bại đầu tiên là không lý tưởng. Một lần nữa, hãy đưa `calculator.py` mã của chúng ta trở lại trạng thái bị hỏng:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n + n


if __name__ == "__main__":
    main()
```

Lưu ý rằng chúng ta đã thay đổi `*` toán tử thành a `+` trong hàm bình phương, đưa nó về trạng thái hỏng.

- Để cải thiện mã kiểm tra của chúng tôi, hãy sửa đổi `test_calculator.py` để chia mã thành các nhóm kiểm tra khác nhau:

```
from calculator import square


def test_positive():
    assert square(2) == 4
    assert square(3) == 9


def test_negative():
    assert square(-2) == 4
    assert square(-3) == 9


def test_zero():
    assert square(0) == 0
```

Lưu ý rằng chúng tôi đã chia năm bài kiểm tra giống nhau thành ba chức năng khác nhau. Các khung kiểm tra như `pytest` sẽ chạy từng chức năng, ngay cả khi có lỗi ở một trong số chúng. Chạy lại `pytest test_calculator.py`, bạn sẽ nhận thấy có nhiều lỗi hơn đang được hiển thị. Nhiều lỗi xuất ra hơn cho phép bạn khám phá thêm điều gì có thể gây ra sự cố trong mã của bạn.

- Sau khi cải thiện mã kiểm tra của chúng tôi, hãy đưa `calculator.py` mã của bạn trở lại trạng thái hoạt động hoàn toàn:

```
def main():
    x = int(input("What's x? "))
    print("x squared is", square(x))


def square(n):
    return n * n


if __name__ == "__main__":
    main()
```

Lưu ý rằng chúng ta đã thay đổi `+` toán tử thành hàm a `*` trong hàm bình phương, đưa nó về trạng thái hoạt động.

- Chạy lại `pytest test_calculator.py` bạn sẽ nhận thấy không có lỗi nào được tìm thấy.
- Tóm lại, với tư cách là một lập trình viên, bạn có thể xác định bao nhiêu điều kiện kiểm tra mà bạn thấy phù hợp!

Bạn có thể tìm hiểu thêm trong tài liệu của Pytest về `pytest` (https://docs.pytest.org/en/7.1.x/getting-started.html).

Chuỗi thử nghiệm

- Quay ngược thời gian, hãy xem xét đoạn mã sau `hello.py`:

```
def main():
    name = input("What's your name? ")
    hello(name)


def hello(to="world"):
    print("hello,", to)


if __name__ == "__main__":
    main()
```

  Lưu ý rằng chúng ta có thể muốn kiểm tra kết quả của hàm `hello`.

- Hãy xem xét đoạn mã sau cho `test_hello.py`:

```
from hello import hello


def test_hello():
    assert hello("David") == "hello, David"
    assert hello() == "hello, world"
```

  Nhìn vào mã này, bạn có nghĩ rằng phương pháp thử nghiệm này sẽ hoạt động tốt không? Tại sao bài kiểm tra này có thể không hoạt động tốt? Lưu ý rằng `hello` hàm `hello.py` in ra một cái gì đó: Tức là nó không trả về một giá trị nào!

- Chúng ta có thể thay đổi `hello` chức năng của mình bên trong `hello.py` như sau:

```
def main():
    name = input("What's your name? ")
    print(hello(name))


def hello(to="world"):
    return f"hello, {to}"


if __name__ == "__main__":
    main()
```

  Lưu ý rằng chúng tôi đã thay đổi `hello` hàm của mình để trả về một chuỗi. Điều này thực sự có nghĩa là bây giờ chúng ta có thể sử dụng `pytest` để kiểm tra `hello` chức năng.

- Đang chạy `pytest test_hello.py`, mã của chúng tôi sẽ vượt qua tất cả các bài kiểm tra!

- Giống như trường hợp thử nghiệm trước trong bài học này, chúng ta có thể chia nhỏ các thử nghiệm của mình ra một cách riêng biệt:

```
from hello import hello
```

```
def test_default():
    assert hello() == "hello, world"


def test_argument():
    assert hello("David") == "hello, David"
```

Lưu ý rằng đoạn mã trên tách thử nghiệm của chúng tôi thành nhiều hàm sao cho tất cả chúng sẽ chạy, ngay cả khi có lỗi xảy ra.

Tổ chức các bài kiểm tra vào các thư mục

- Mã thử nghiệm đơn vị sử dụng nhiều thử nghiệm phổ biến đến mức bạn có khả năng chạy toàn bộ thư mục thử nghiệm chỉ bằng một lệnh.
- Đầu tiên, trong cửa sổ terminal, thực thi `mkdir test` để tạo một thư mục có tên `test`.
- Sau đó, để tạo bài kiểm tra trong thư mục đó, hãy nhập vào cửa sổ terminal `code test/test_hello.py`. Lưu ý rằng `test/` hướng dẫn thiết bị đầu cuối tạo `test_hello.py` trong thư mục có tên `test`.
- Trong cửa sổ soạn thảo văn bản, sửa đổi tệp để bao gồm mã sau:

```
from hello import hello


def test_default():
    assert hello() == "hello, world"


def test_argument():
    assert hello("David") == "hello, David"
```

Lưu ý rằng chúng tôi đang tạo một bài kiểm tra giống như chúng tôi đã làm trước đây.

- `pytest` sẽ không cho phép chúng tôi chạy thử nghiệm dưới dạng một thư mục chỉ với tệp này (hoặc toàn bộ tập hợp tệp) mà không có `__init__` tệp đặc biệt. Trong cửa sổ terminal của bạn, tạo tệp này bằng cách nhập `code test/__init__.py`. Lưu ý `test/` như trước, cũng như dấu gạch dưới kép ở hai bên của `init`. `__init__.py` Ngay cả khi để trống tệp này , `pytest` người ta vẫn thông báo rằng toàn bộ thư mục chứa `__init__.py` các bài kiểm tra có thể chạy được.
- Bây giờ, gõ `pytest test` vào terminal, bạn có thể chạy toàn bộ `test` thư mục mã.

Bạn có thể tìm hiểu thêm trong tài liệu về cơ chế nhập khẩu (https://docs.pytest.org/en/7.1.x/explanation/pythonpath.html?highlight=folder#pytest-import-mechanisms-and-sys-path-pythonpath) của Pytest .

Tổng hợp

Kiểm tra mã của bạn là một phần tự nhiên của quá trình lập trình. Kiểm tra đơn vị cho phép bạn kiểm tra các khía cạnh cụ thể của mã của mình. Bạn có thể tạo các chương trình của riêng mình để kiểm tra mã của bạn. Ngoài ra, bạn có thể sử dụng các khung như `pytest` để chạy thử nghiệm đơn vị cho mình. Trong bài giảng này, bạn đã tìm hiểu về...

- Kiểm tra đơn vị
- `assert`
- `pytest`