Unit 6: Pytest

Part One

This task includes code and tests for a wallet. We were instructed to copy the code and run the tests.

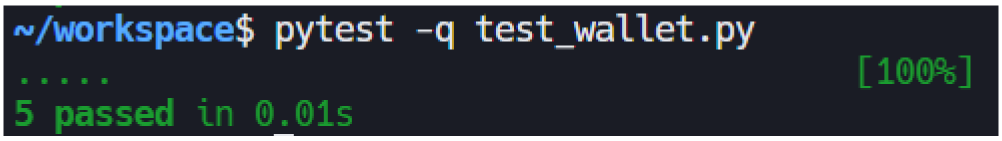
wallet.py

```python
# code source: https://semaphoreci.com/community/tutorials/testing-python-applications-
with-pytest
# wallet.py
class InsufficientAmount(Exception):
    pass

class Wallet(object):
    def __init__(self, initial_amount=0):
        self.balance = initial_amount

    def spend_cash(self, amount):
        if self.balance < amount:
            raise InsufficientAmount('Not enough available to spend
{}'.format(amount))
        self.balance += amount

    def add_cash(self, amount):
        self.balance += amount
```

test_wallet.py

```python
# code source: https://semaphoreci.com/community/tutorials/testing-python-applications-
with-pytest
# test_wallet.py
import pytest
from wallet import Wallet, InsufficientAmount

def test_default_initial_amount():
    wallet = Wallet()
    assert wallet.balance == 0

def test_setting_initial_amount():
    wallet = Wallet(100)
    assert wallet.balance == 100

def test_wallet_add_cash():
    wallet = Wallet(10)
    wallet.add_cash(90)
    assert wallet.balance == 100

def test_wallet_spend_cash():
    wallet = Wallet(20)
    wallet.spend_cash(10)
    assert wallet.balance == 10

def test_wallet_spend_cash_raises_exception_on_insufficient_amount():
    wallet = Wallet()
    with pytest.raises(InsufficientAmount):
        wallet.spend_cash(100)
```

Output



```
~/workspace$ pytest -q test_wallet.py
.....                                                    [100%]
5 passed in 0.01s
```

Part Two

We were then asked to change the code in wallet to make the test fail.

Code changes

The first test is to test the default value of the wallet. This can be broken by changing the initial amount from 0 to 50.

```
def __init__(self, initial_amount=50):
    self.balance = initial_amount
```

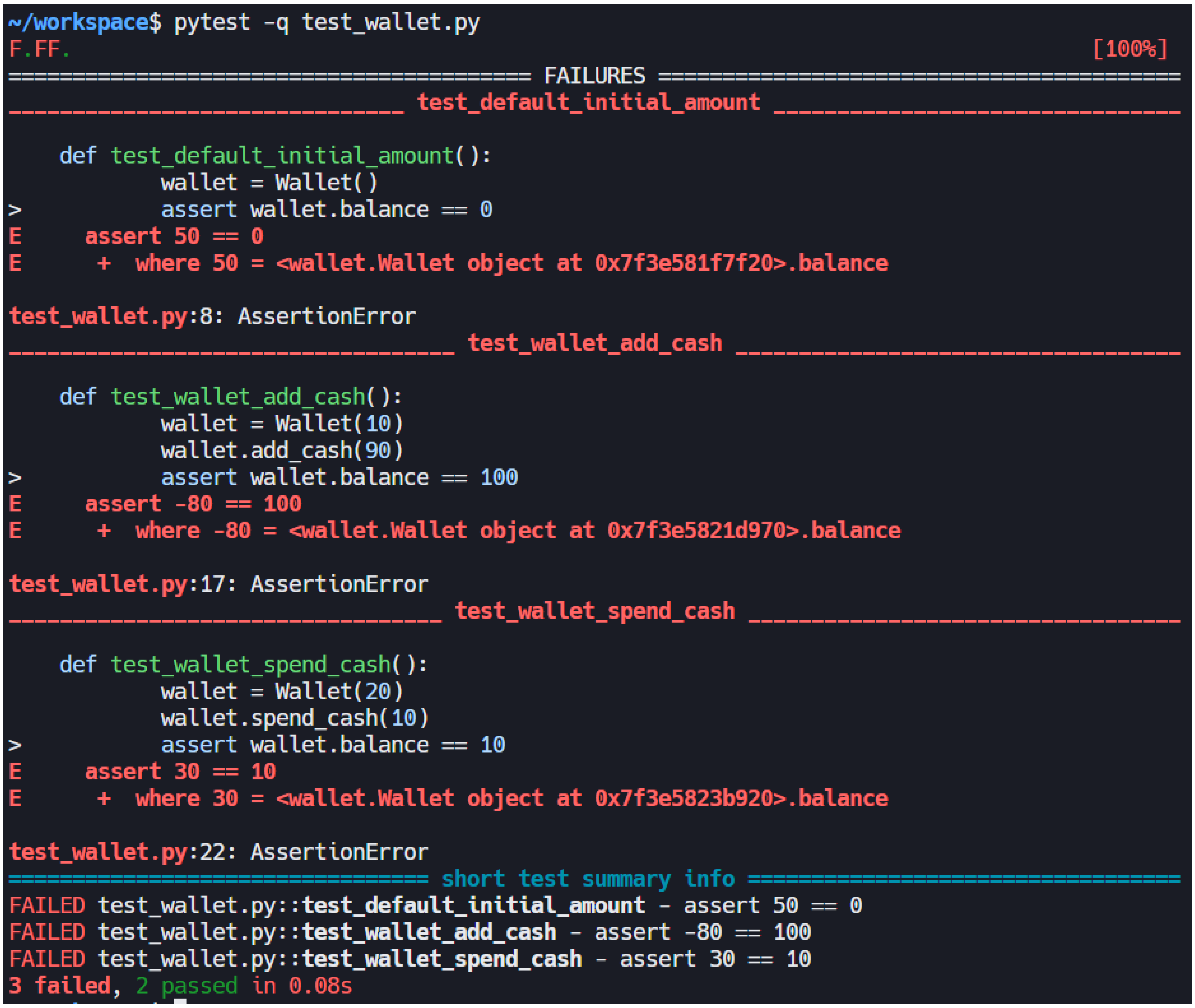
The test to add cash can be broken by changing the + to a -.

```
def add_cash(self, amount):
    self.balance -= amount
```

The test for spending cash can again be broken by changing the maths.

```
def spend_cash(self, amount):
    if self.balance < amount:
        raise InsufficientAmount('Not enough available to spend {}'.format(amount))
    self.balance += amount
```

Output



```
~/workspace$ pytest -q test_wallet.py
F.FF.                                                                    [100%]
=================================== FAILURES ===================================
__________________________ test_default_initial_amount _________________________

    def test_default_initial_amount():
            wallet = Wallet()
>           assert wallet.balance == 0
E     assert 50 == 0
E      +  where 50 = <wallet.Wallet object at 0x7f3e581f7f20>.balance

test_wallet.py:8: AssertionError
______________________________ test_wallet_add_cash ____________________________

    def test_wallet_add_cash():
            wallet = Wallet(10)
            wallet.add_cash(90)
>           assert wallet.balance == 100
E     assert -80 == 100
E      +  where -80 = <wallet.Wallet object at 0x7f3e5821d970>.balance

test_wallet.py:17: AssertionError
_____________________________ test_wallet_spend_cash ___________________________

    def test_wallet_spend_cash():
            wallet = Wallet(20)
            wallet.spend_cash(10)
>           assert wallet.balance == 10
E     assert 30 == 10
E      +  where 30 = <wallet.Wallet object at 0x7f3e5823b920>.balance

test_wallet.py:22: AssertionError
=========================== short test summary info ============================
FAILED test_wallet.py::test_default_initial_amount - assert 50 == 0
FAILED test_wallet.py::test_wallet_add_cash - assert -80 == 100
FAILED test_wallet.py::test_wallet_spend_cash - assert 30 == 10
3 failed, 2 passed in 0.08s
```