# Testing Your Code

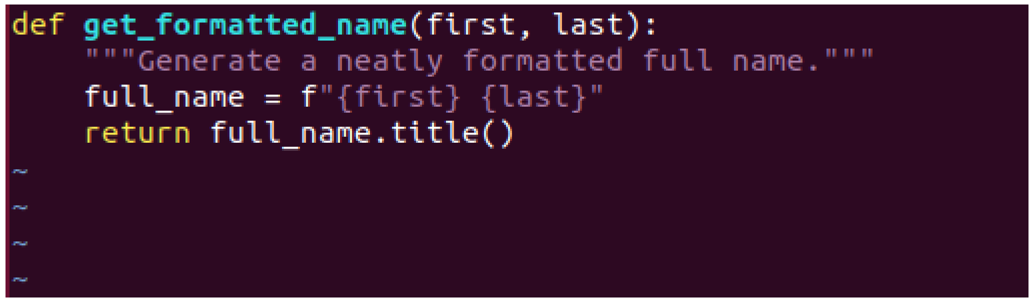
## Testing a Function

# Unit Tests and Test Cases

- The module `unittest` from the Python standard library provides tools for testing your code.
- A *unit test* verifies that one specific aspect of a function's behavior is correct.
- A *test case* is a collection of unit tests that together prove that a function behaves as it's supposed to, within the full range of situations you expect it to handle.
- A test case with *full coverage* includes a full range of unit tests covering all the possible ways you can use a function.
- Achieving full coverage on a large project can be daunting.
- It's often good enough to write tests for your code's critical behaviors and then aim for full coverage only if the project starts to see widespread use.

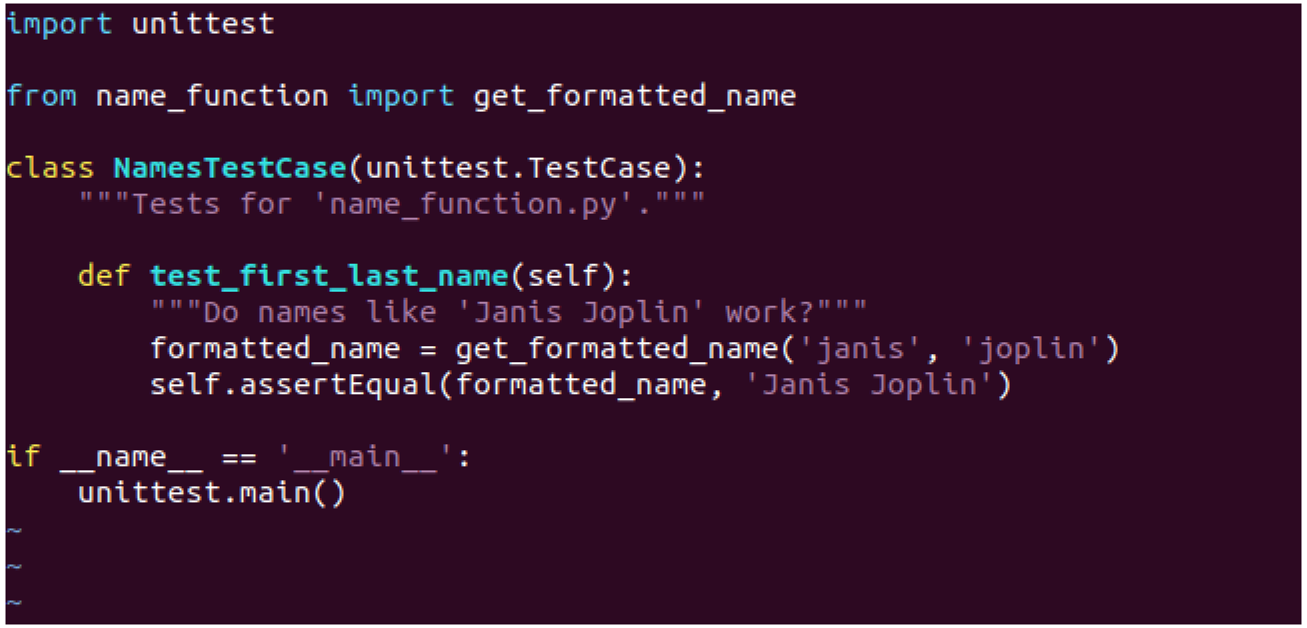
# A Passing Test

- `name_function.py`

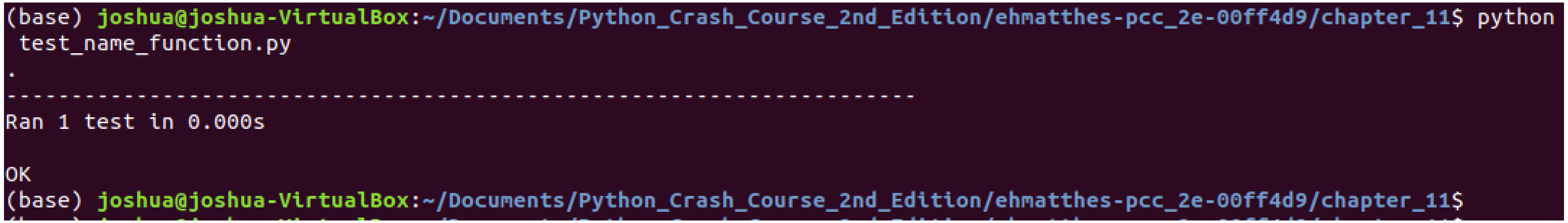


```
def get_formatted_name(first, last):
    """Generate a neatly formatted full name."""
    full_name = f"{first} {last}"
    return full_name.title()
~
~
~
~
```

- test_name_function.py



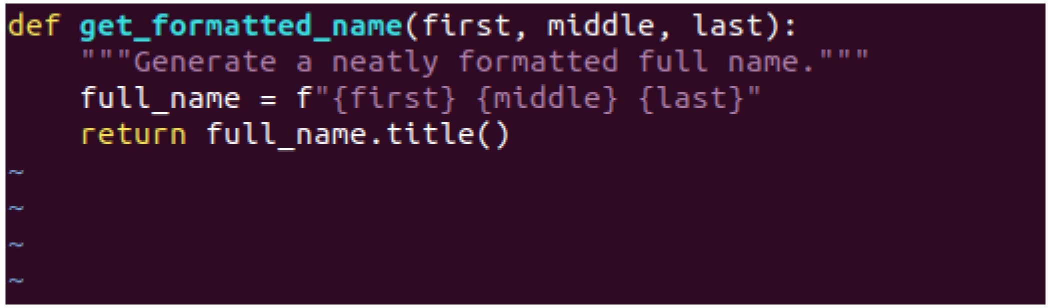
```
import unittest

from name_function import get_formatted_name

class NamesTestCase(unittest.TestCase):
    """Tests for 'name_function.py'."""

    def test_first_last_name(self):
        """Do names like 'Janis Joplin' work?"""
        formatted_name = get_formatted_name('janis', 'joplin')
        self.assertEqual(formatted_name, 'Janis Joplin')

if __name__ == '__main__':
    unittest.main()
~
~
~
```



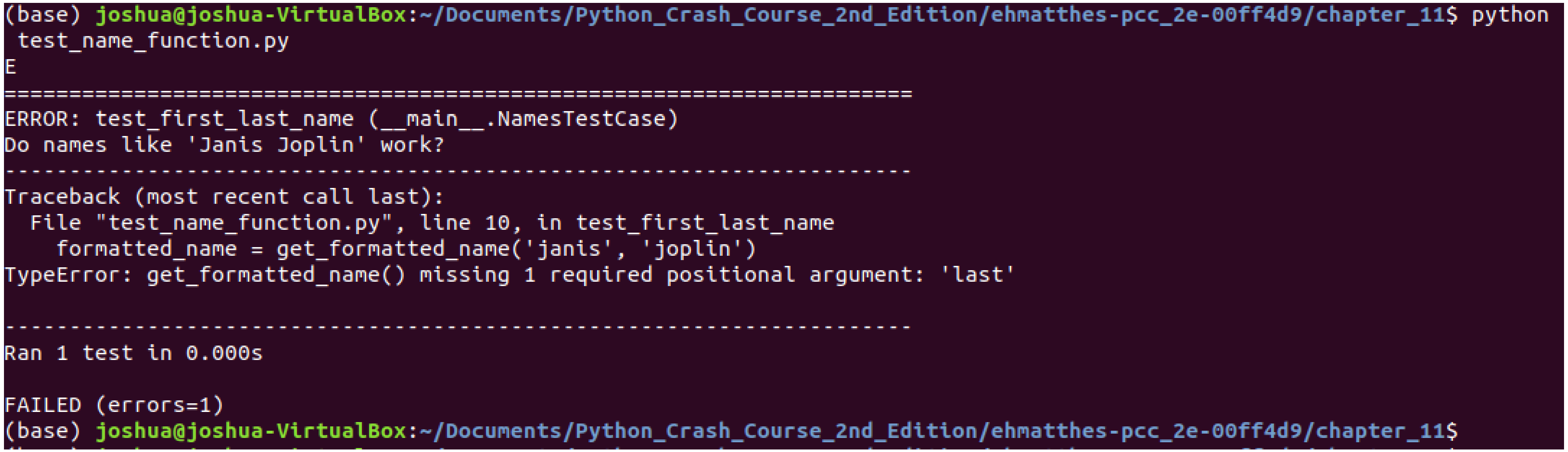
```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$ python
 test_name_function.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$
```

## A Failing Test

- Modified `name_function.py`



```
def get_formatted_name(first, middle, last):
    """Generate a neatly formatted full name."""
    full_name = f"{first} {middle} {last}"
    return full_name.title()
~
~
~
~
```

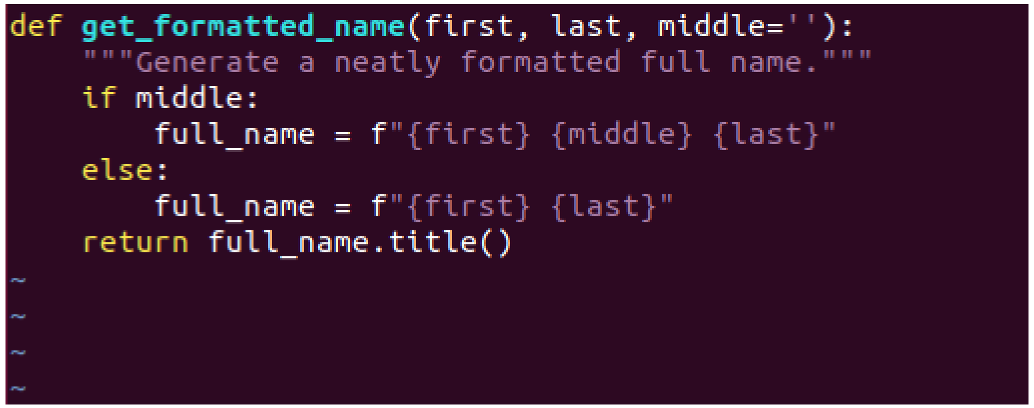


```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$ python
 test_name_function.py
E
======================================================================
ERROR: test_first_last_name (__main__.NamesTestCase)
Do names like 'Janis Joplin' work?
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_name_function.py", line 10, in test_first_last_name
    formatted_name = get_formatted_name('janis', 'joplin')
TypeError: get_formatted_name() missing 1 required positional argument: 'last'

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (errors=1)
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$
```

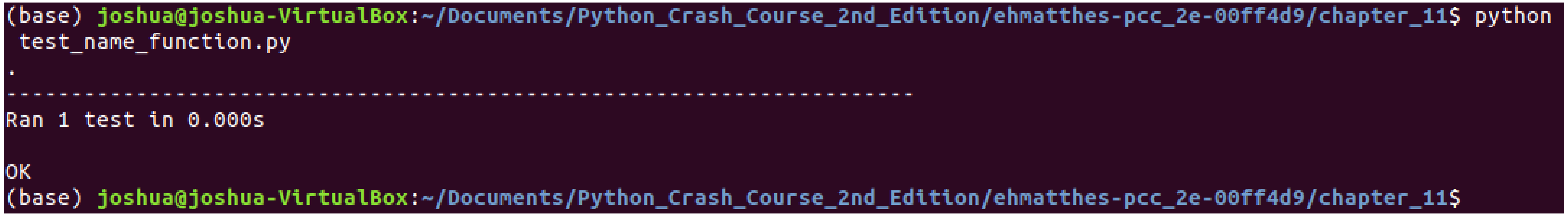
## Responding to a Failed Test

- What do you do when a test fails?
- Assuming you’re checking the right conditions, a passing test means the function is behaving correctly and a failing test means there’s an error in the new code you wrote.
- So when a test fails, don’t change the test.
- Instead, fix the code that caused the test to fail.

- Modified `name_function.py`



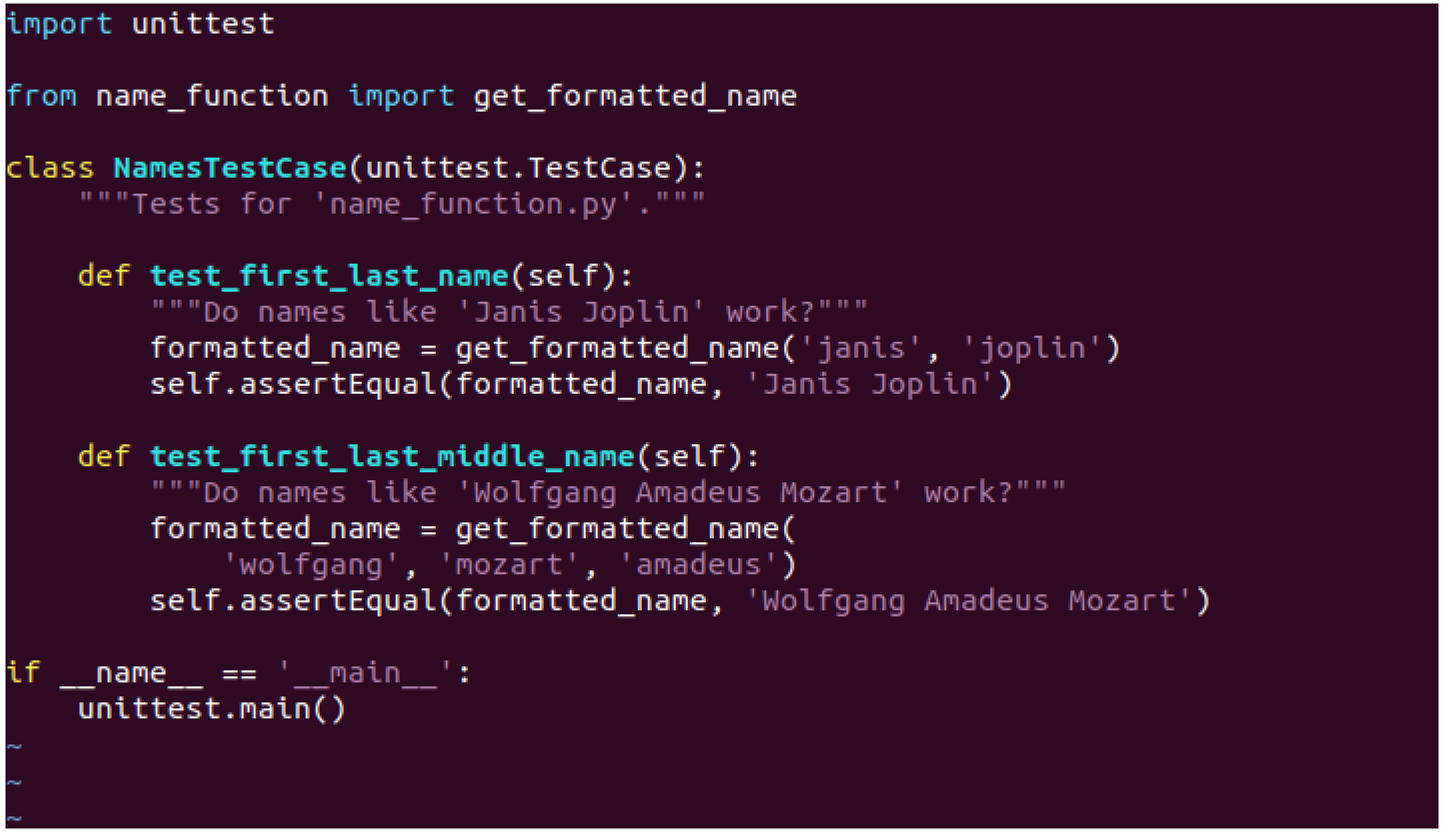
```
def get_formatted_name(first, last, middle=''):
    """Generate a neatly formatted full name."""
    if middle:
        full_name = f"{first} {middle} {last}"
    else:
        full_name = f"{first} {last}"
    return full_name.title()
~
~
~
~
```



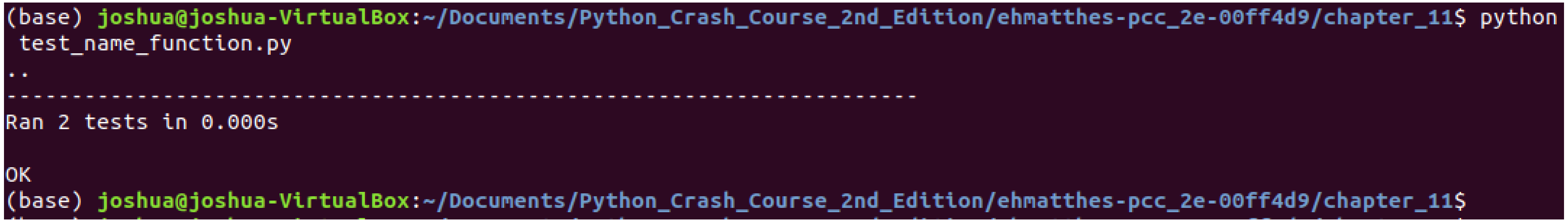
```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$ python
 test_name_function.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$
```

## Adding New Tests

- **Modified** `test_name_function.py`



```
import unittest

from name_function import get_formatted_name

class NamesTestCase(unittest.TestCase):
    """Tests for 'name_function.py'."""

    def test_first_last_name(self):
        """Do names like 'Janis Joplin' work?"""
        formatted_name = get_formatted_name('janis', 'joplin')
        self.assertEqual(formatted_name, 'Janis Joplin')

    def test_first_last_middle_name(self):
        """Do names like 'Wolfgang Amadeus Mozart' work?"""
        formatted_name = get_formatted_name(
            'wolfgang', 'mozart', 'amadeus')
        self.assertEqual(formatted_name, 'Wolfgang Amadeus Mozart')

if __name__ == '__main__':
    unittest.main()
~
~
~
```

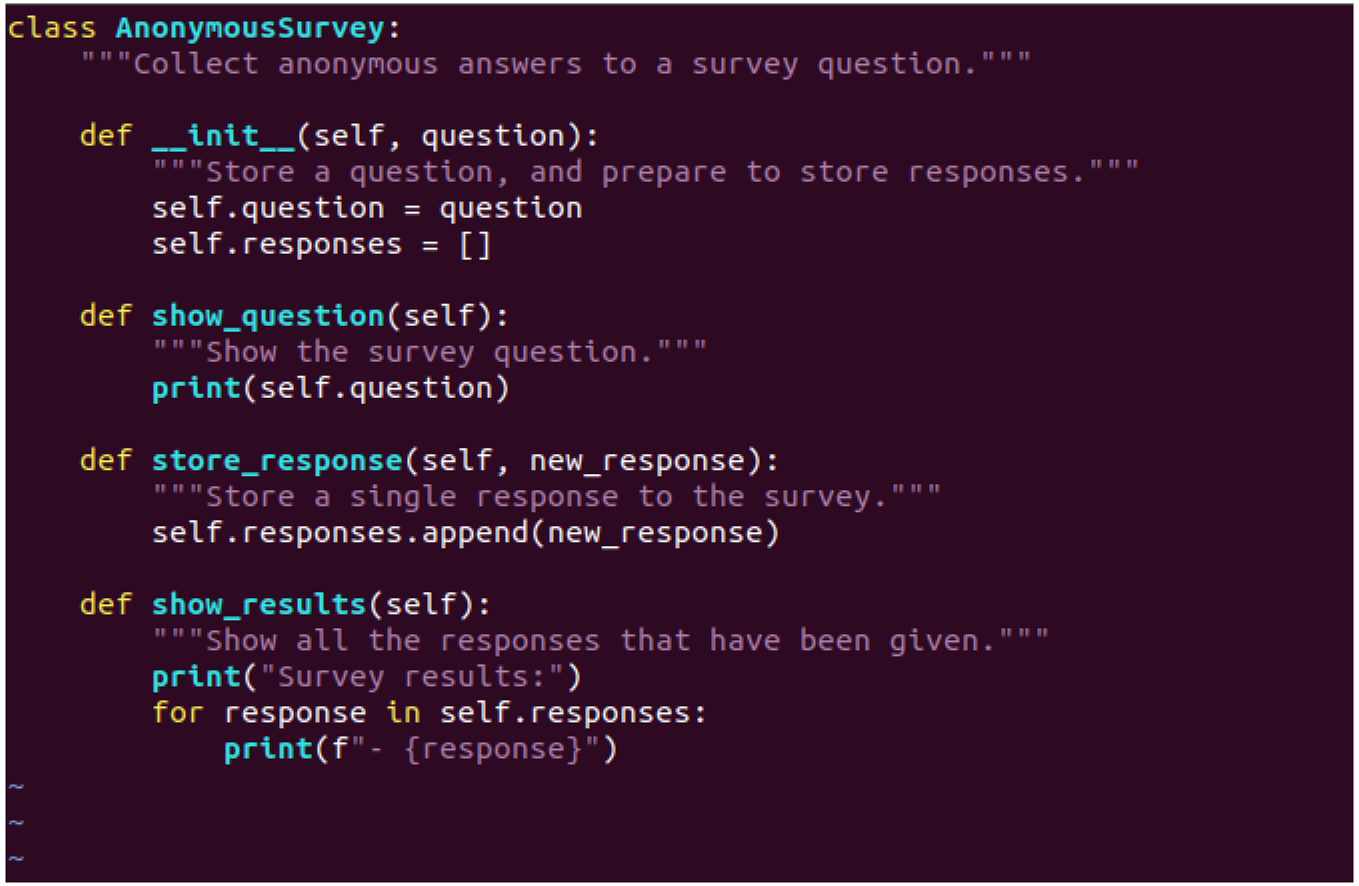


```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$ python
 test_name_function.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$
```

## Testing a Class

## A Class to Test

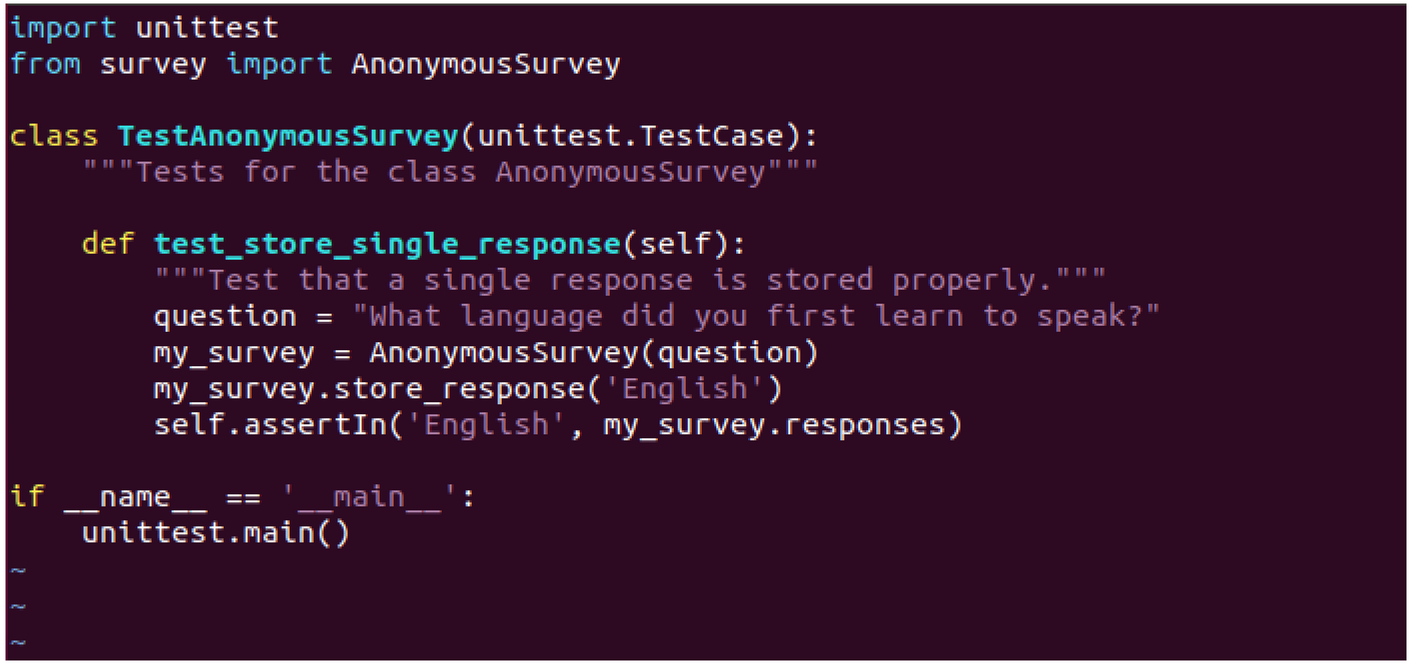
- `survey.py`



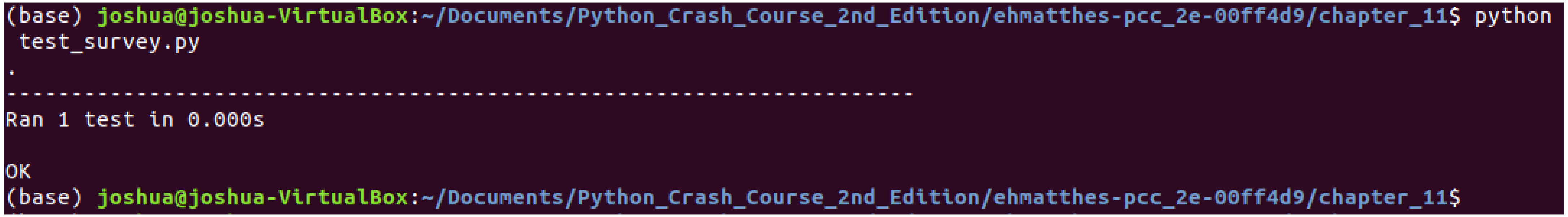
```
class AnonymousSurvey:
    """Collect anonymous answers to a survey question."""

    def __init__(self, question):
        """Store a question, and prepare to store responses."""
        self.question = question
        self.responses = []

    def show_question(self):
        """Show the survey question."""
        print(self.question)

    def store_response(self, new_response):
        """Store a single response to the survey."""
        self.responses.append(new_response)

    def show_results(self):
        """Show all the responses that have been given."""
        print("Survey results:")
        for response in self.responses:
            print(f"- {response}")
~
~
~
```

## Testing the `AnonymousSurvey` Class

- `test_survey.py`



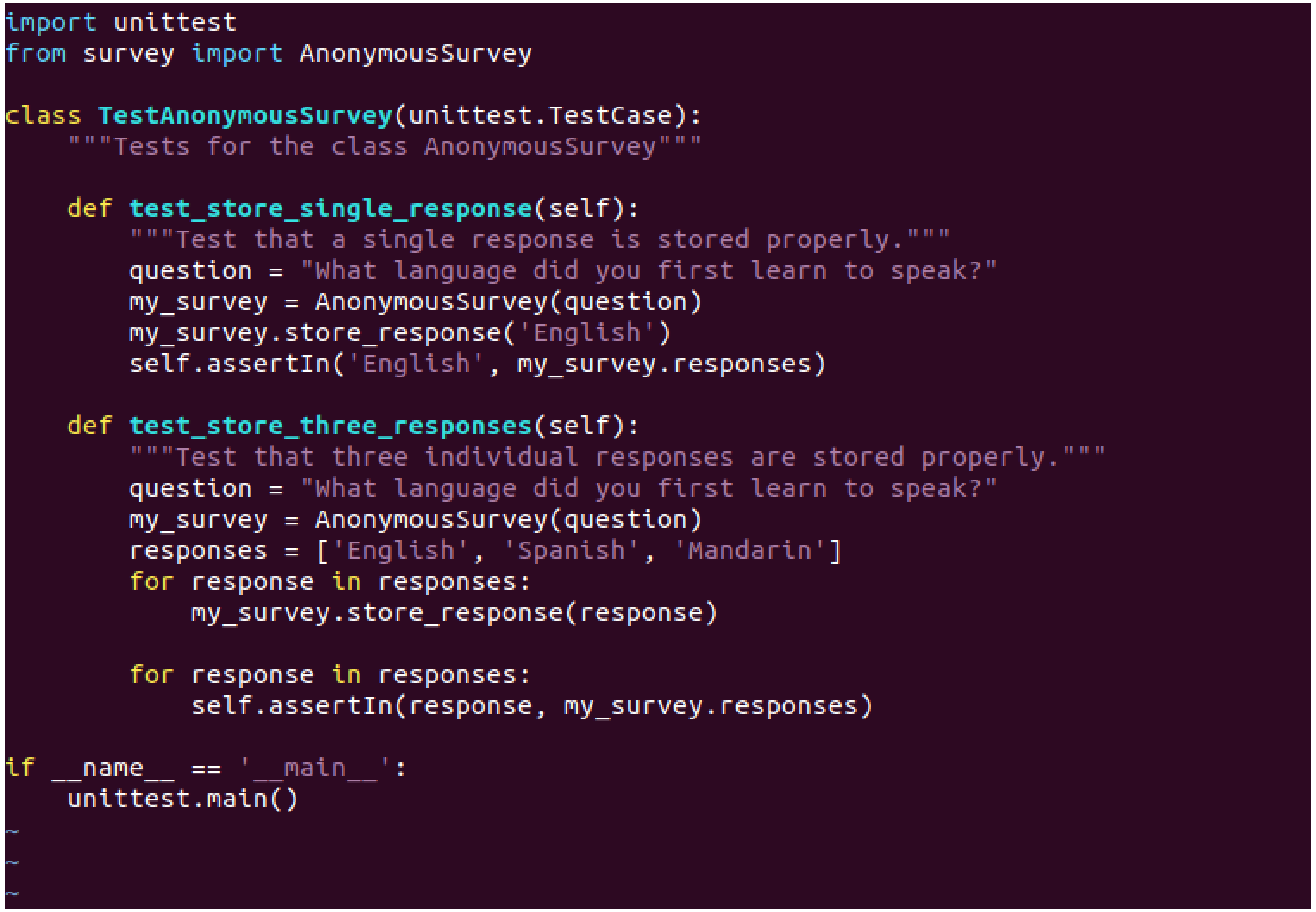
```
import unittest
from survey import AnonymousSurvey

class TestAnonymousSurvey(unittest.TestCase):
    """Tests for the class AnonymousSurvey"""

    def test_store_single_response(self):
        """Test that a single response is stored properly."""
        question = "What language did you first learn to speak?"
        my_survey = AnonymousSurvey(question)
        my_survey.store_response('English')
        self.assertIn('English', my_survey.responses)

if __name__ == '__main__':
    unittest.main()
~
~
~
```



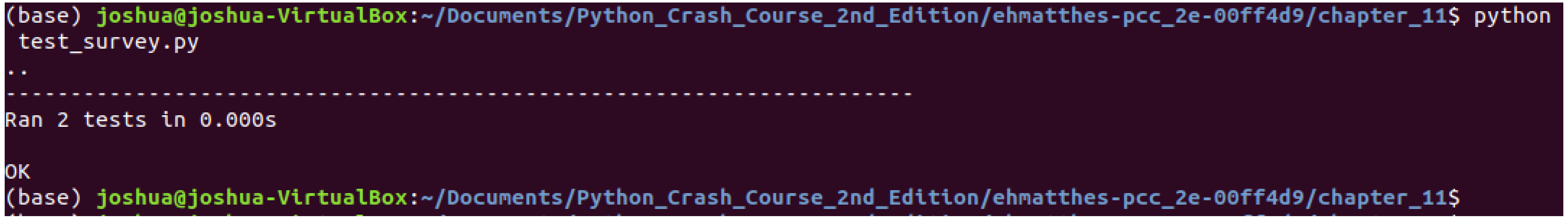
```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$ python
 test_survey.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$
```

- This is good, but a survey is useful only if it generates more than one response.
- Let’s verify that three responses can be stored correctly.

- Modified `test_survey.py`



```python
import unittest
from survey import AnonymousSurvey

class TestAnonymousSurvey(unittest.TestCase):
    """Tests for the class AnonymousSurvey"""

    def test_store_single_response(self):
        """Test that a single response is stored properly."""
        question = "What language did you first learn to speak?"
        my_survey = AnonymousSurvey(question)
        my_survey.store_response('English')
        self.assertIn('English', my_survey.responses)

    def test_store_three_responses(self):
        """Test that three individual responses are stored properly."""
        question = "What language did you first learn to speak?"
        my_survey = AnonymousSurvey(question)
        responses = ['English', 'Spanish', 'Mandarin']
        for response in responses:
            my_survey.store_response(response)

        for response in responses:
            self.assertIn(response, my_survey.responses)

if __name__ == '__main__':
    unittest.main()
~
~
~
```

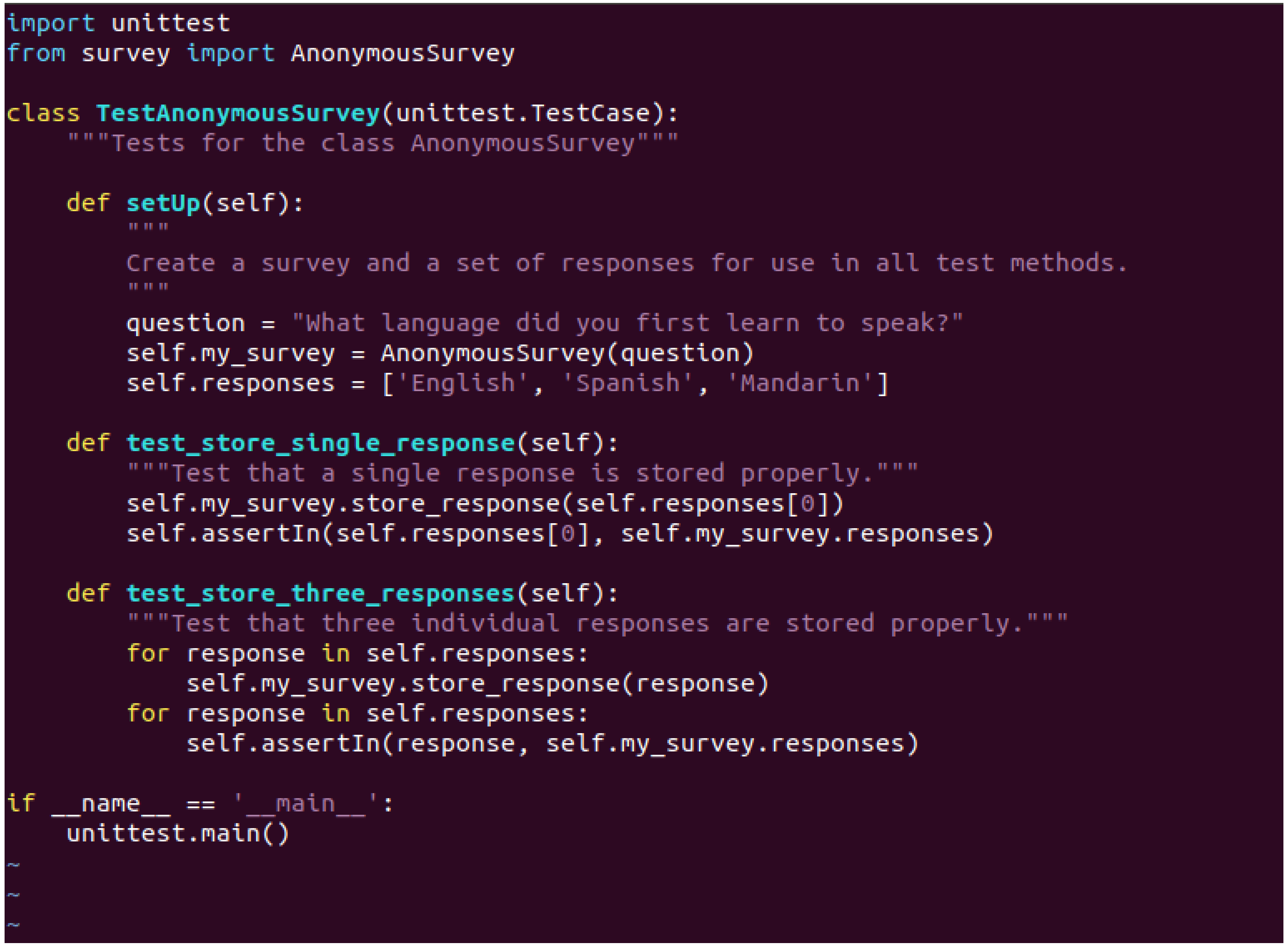


```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$ python
 test_survey.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$
```

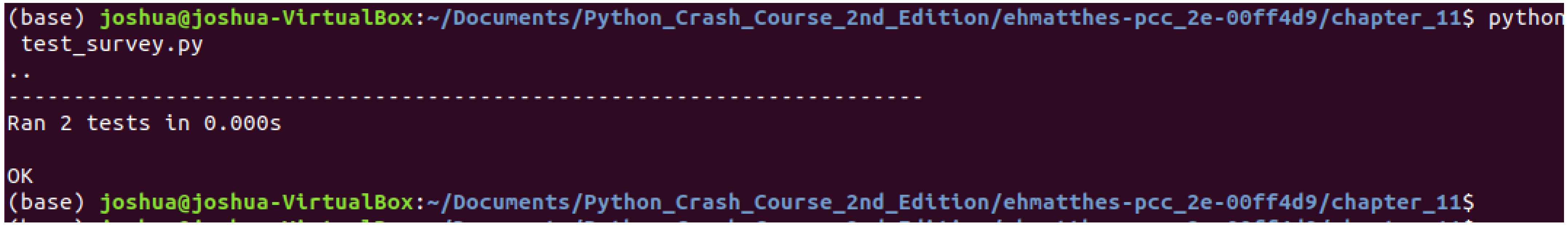
## The `setUp()` Method

- In `test_survey.py` we created a new instance of `AnonymousSurvey` in each test method, and we created new responses in each method.
- The `unittest.TestCase` class has a `setUp()` method that allows you to create these objects once and then use them in each of your test methods.
- When you include a `setUp()` method in a `TestCase` class, Python runs the `setUp()` method before running each method starting with *`test_`*.
- Any objects created in the `setUp()` method are then available in each test method you write.

- Modified `test_survey.py`



```
import unittest
from survey import AnonymousSurvey

class TestAnonymousSurvey(unittest.TestCase):
    """Tests for the class AnonymousSurvey"""

    def setUp(self):
        """
        Create a survey and a set of responses for use in all test methods.
        """
        question = "What language did you first learn to speak?"
        self.my_survey = AnonymousSurvey(question)
        self.responses = ['English', 'Spanish', 'Mandarin']

    def test_store_single_response(self):
        """Test that a single response is stored properly."""
        self.my_survey.store_response(self.responses[0])
        self.assertIn(self.responses[0], self.my_survey.responses)

    def test_store_three_responses(self):
        """Test that three individual responses are stored properly."""
        for response in self.responses:
            self.my_survey.store_response(response)
        for response in self.responses:
            self.assertIn(response, self.my_survey.responses)

if __name__ == '__main__':
    unittest.main()
~
~
~
```



```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$ python
 test_survey.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_11$
```