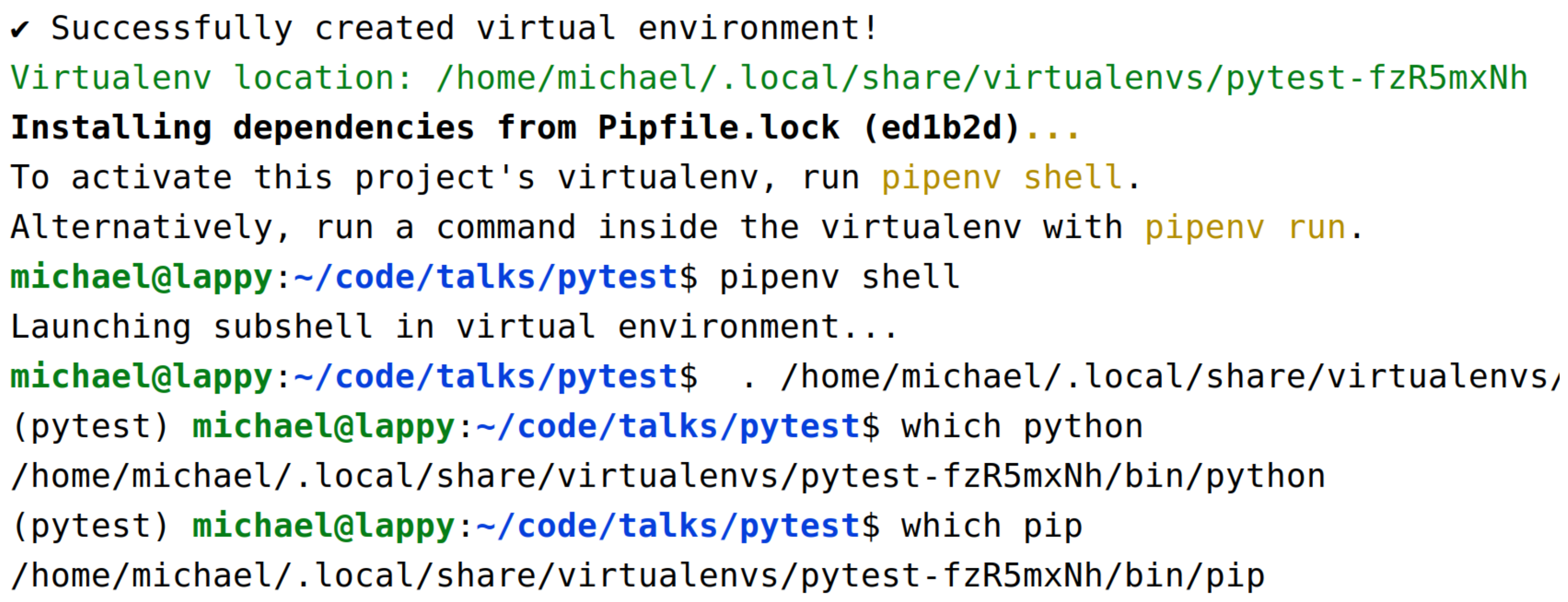
Introduction to pytest

pytest

- Defacto standard testing tool for Python
- unittest is in the standard library, pytest is better
- pytest is a command line tool for collecting and running tests
- Also a framework for writing tests
- Extendable by plugins (not very hard to write), for example pytest-asyncio for async testing
- Very widely used, lots of documentation and videos
- The purpose of testing is to ***verify behaviour***
- *https://pytest.org/*

Installing pytest

- Install pytest into a virtual environment
- pipenv is commonly used to manage environments and dependencies
- Pipfile and Pipfile.lock specify dependencies
- Install and create the virtual environment with "***pipenv install***"
- Activate the virtual environment with "*pipenv shell*"



```
✔ Successfully created virtual environment!
Virtualenv location: /home/michael/.local/share/virtualenvs/pytest-fzR5mxNh
Installing dependencies from Pipfile.lock (ed1b2d)...
To activate this project's virtualenv, run pipenv shell.
Alternatively, run a command inside the virtualenv with pipenv run.
michael@lappy:~/code/talks/pytest$ pipenv shell
Launching subshell in virtual environment...
michael@lappy:~/code/talks/pytest$  . /home/michael/.local/share/virtualenvs/
(pytest) michael@lappy:~/code/talks/pytest$ which python
/home/michael/.local/share/virtualenvs/pytest-fzR5mxNh/bin/python
(pytest) michael@lappy:~/code/talks/pytest$ which pip
/home/michael/.local/share/virtualenvs/pytest-fzR5mxNh/bin/pip
```

Creating a Test Suite

- Test collection is done with a naming convention:
- Write tests as functions* in files called "*`test_something.py`*" (etc)
- They probably live in a project directory called "*tests*"
- Run the tests with pytest
- A "*test suite*" is a collection of tests found from test files

(*) Tests can be collected in classes or generated. Test functions are most common though.

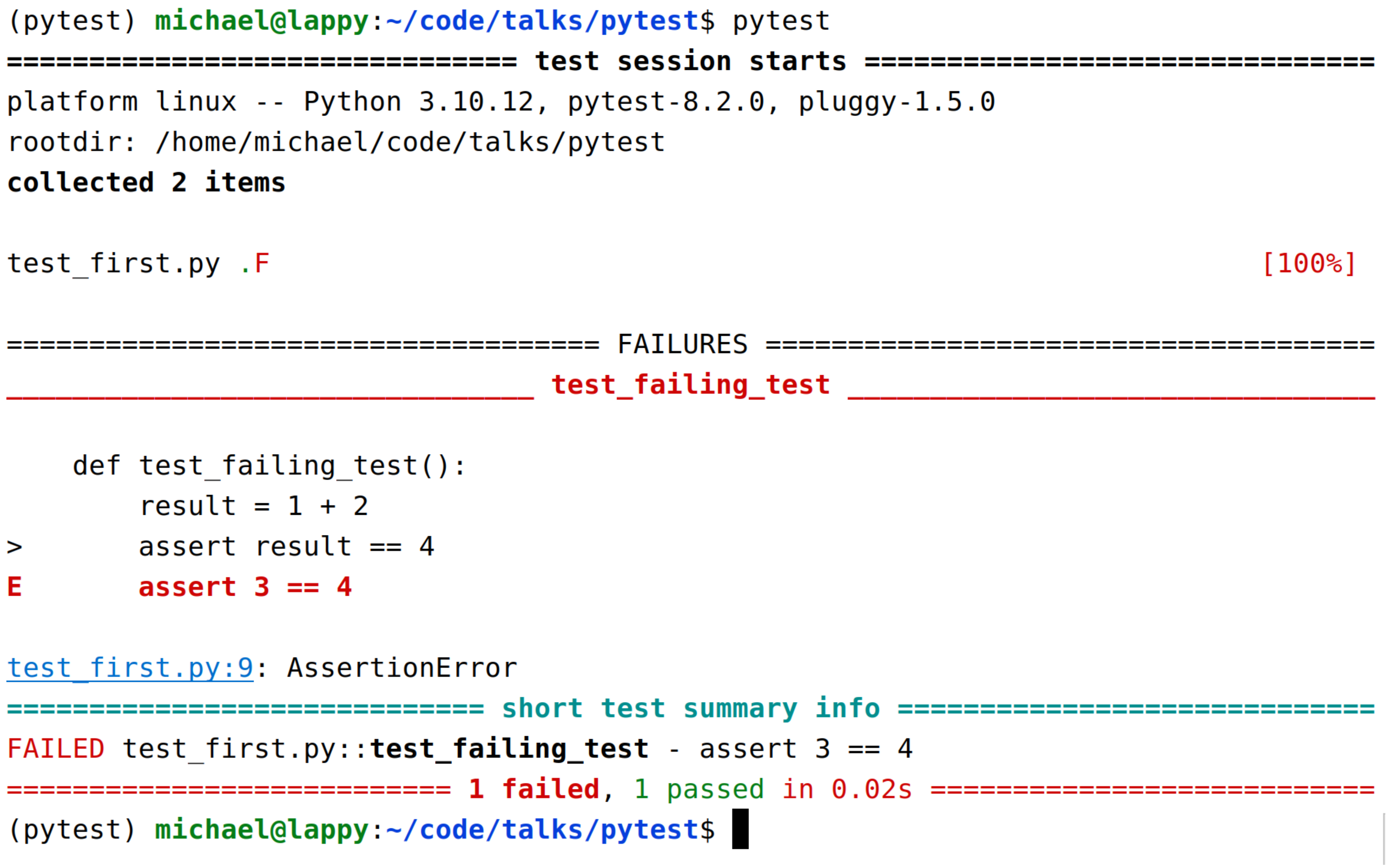
Test Functions

- Functions should be named "*test_something*" as well
- Use the assert statement to verify something
- The test fails with a useful error message if an assert fails or something goes wrong

```python
def test_function():
    result = 1 + 2
    assert result == 3


def test_failing_test():
    result = 1 + 2
    assert result == 4
```

Test Run



```
(pytest) michael@lappy:~/code/talks/pytest$ pytest
================================ test session starts ================================
platform linux -- Python 3.10.12, pytest-8.2.0, pluggy-1.5.0
rootdir: /home/michael/code/talks/pytest
collected 2 items

test_first.py .F                                                              [100%]

===================================== FAILURES ======================================
_________________________________ test_failing_test _________________________________

    def test_failing_test():
        result = 1 + 2
>       assert result == 4
E       assert 3 == 4

test_first.py:9: AssertionError
============================== short test summary info ==============================
FAILED test_first.py::test_failing_test - assert 3 == 4
============================ 1 failed, 1 passed in 0.02s ============================
(pytest) michael@lappy:~/code/talks/pytest$ 
```

Setting up the System Under Test

- The code you're testing is the "*System under test*"
- It usually needs setting up before you can test it
  - You might need to run a server
  - You might need to provide or populate test data
  - You might need to mock out some external services for the tests to work
- We can setup the system under test using pytest **fixtures**

Fixtures

- Test functions specify test fixtures as parameters
- Fixtures are made available once they've been imported
- When a test is run the fixture is called by pytest and passed into the function for you
- A common place to put them is *conftest.py* which pytest always checks