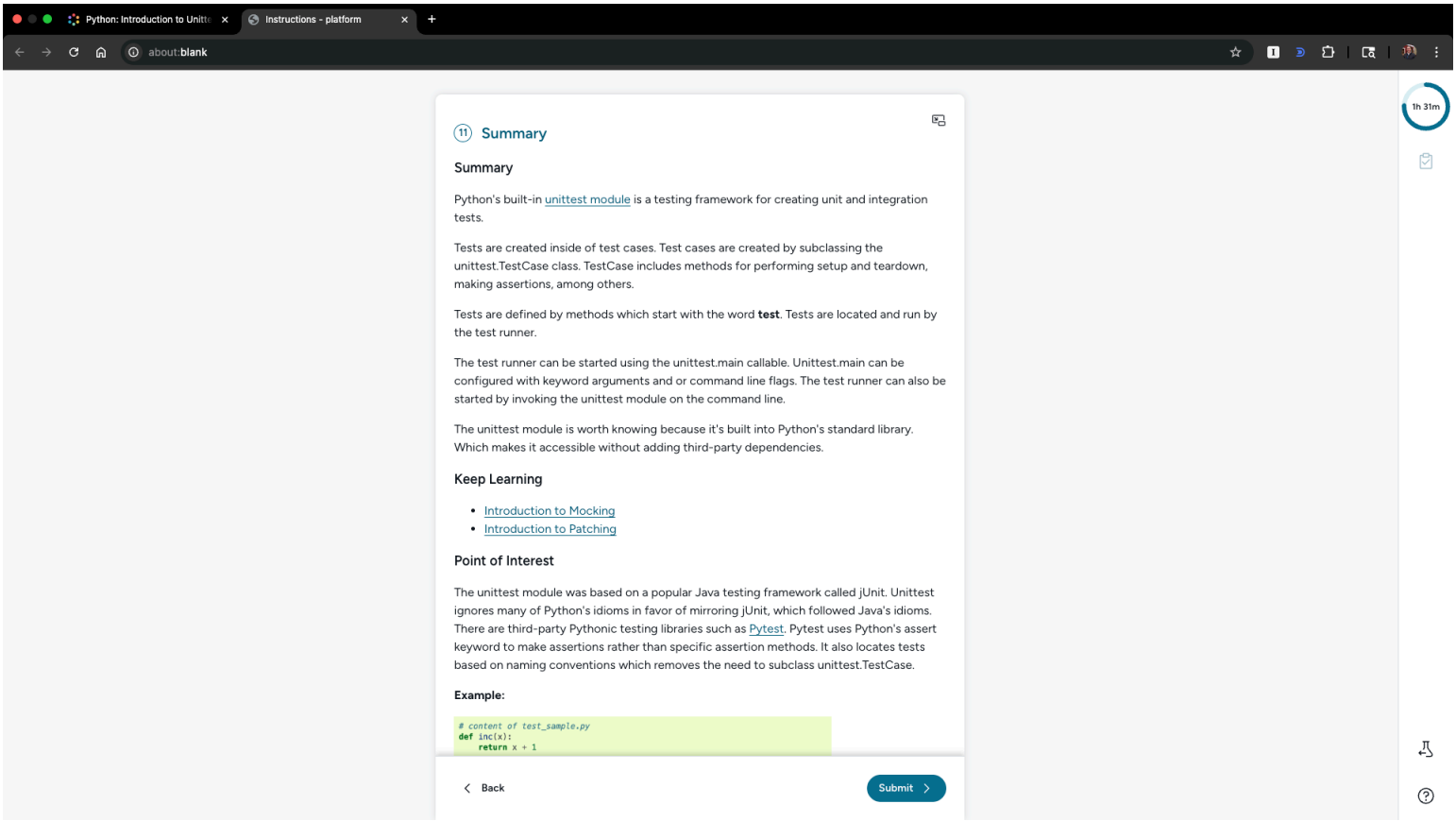
Python: Introduction to Unitte
Instructions - platform
about:blank
1h 31m
11 Summary
Summary
Summary
Python's built-in unittest module is a testing framework for creating unit and integration tests.
Tests are created inside of test cases. Test cases are created by subclassing the unittest.TestCase class. TestCase includes methods for performing setup and teardown, making assertions, among others.
Tests are defined by methods which start with the word test. Tests are located and run by the test runner.
The test runner can be started using the unittest.main callable. Unittest.main can be configured with keyword arguments and or command line flags. The test runner can also be started by invoking the unittest module on the command line.
The unittest module is worth knowing because it's built into Python's standard library. Which makes it accessible without adding third-party dependencies.
Keep Learning
Introduction to Mocking
Introduction to Patching
Point of Interest
The unittest module was based on a popular Java testing framework called jUnit. Unittest ignores many of Python's idioms in favor of mirroring jUnit, which followed Java's idioms. There are third-party Pythonic testing libraries such as Pytest. Pytest uses Python's assert keyword to make assertions rather than specific assertion methods. It also locates tests based on naming conventions which removes the need to subclass unittest.TestCase.
Example:
# content of test_sample.py
def inc(x):
return x + 1
Back
Submit

Python: Introduction to Unitte | Instructions - platform | about:blank

- Introduction to Mocking
- Introduction to Patching

Point of Interest

The unittest module was based on a popular Java testing framework called jUnit. Unittest ignores many of Python's idioms in favor of mirroring jUnit, which followed Java's idioms. There are third-party Pythonic testing libraries such as Pytest. Pytest uses Python's assert keyword to make assertions rather than specific assertion methods. It also locates tests based on naming conventions which removes the need to subclass unittest.TestCase.

Example:

```
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(3) == 5
```

To execute it:

```
$ pytest
============================= test session starts ==============================
platform linux -- Python 3.x.y, pytest-7.x.y, pluggy-1.x.y
rootdir: /home/sweet/project
collected 1 item

test_sample.py F                                                      [100%]

================================= FAILURES =================================
_____________________________ test_answer _____________________________

    def test_answer():
>       assert inc(3) == 5
E       assert 4 == 5
E        +  where 4 = inc(3)

test_sample.py:6: AssertionError
========================= short test summary info ==========================
FAILED test_sample.py::test_answer - assert 4 == 5
============================ 1 failed in 0.12s =============================
```

— docs.pytest.org

Report an issue

Back | Submit

1h 31m