**xfail mark and pytest.raises**

It is also possible to specify a raises argument to [pytest.mark.xfail](https://docs.pytest.org/en/stable/reference/reference.html" \l "pytest-mark-xfail-ref), which checks that the test is failing in a more specific way than just having any exception raised:

**def** f**():**

**raise** **IndexError()**

**@pytest.**mark**.**xfail**(**raises**=IndexError)**

**def** test\_f**():**

f**()**

This will only “xfail” if the test fails by raising IndexError or subclasses.

* Using [pytest.mark.xfail](https://docs.pytest.org/en/stable/reference/reference.html" \l "pytest-mark-xfail-ref) with the raises parameter is probably better for something like documenting unfixed bugs (where the test describes what “should” happen) or bugs in dependencies.
* Using [pytest.raises()](https://docs.pytest.org/en/stable/reference/reference.html" \l "pytest.raises" \o "pytest.raises) is likely to be better for cases where you are testing exceptions your own code is deliberately raising, which is the majority of cases.

In pytest , we can create our own assertions using :  
*# content of conftest.py*

**from** test\_foocompare **import** Foo

**def** pytest\_assertrepr\_compare**(**op**,** left**,** right**):**

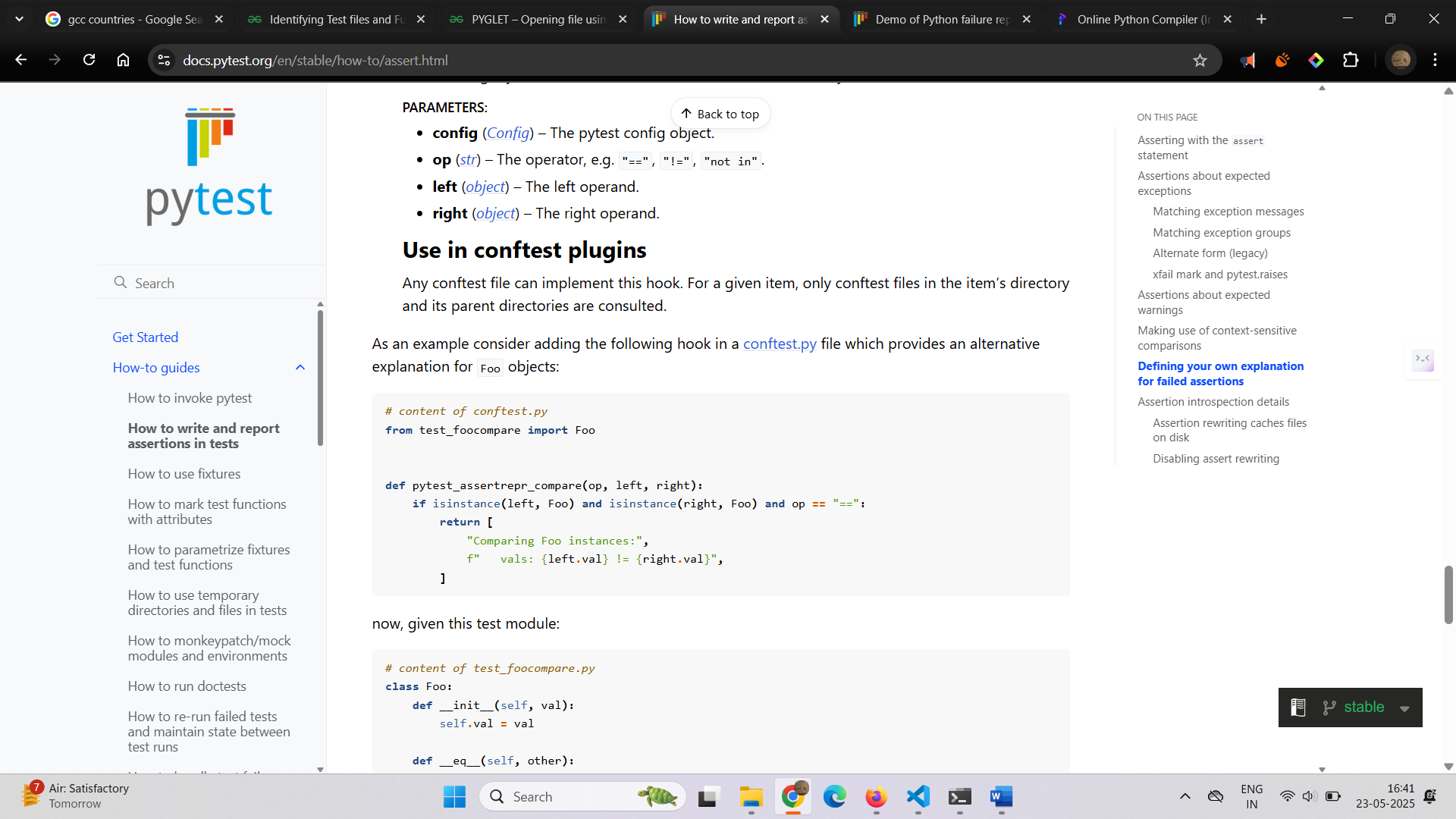
**if** isinstance**(**left**,** Foo**)** **and** isinstance**(**right**,** Foo**)** **and** op **==** "=="**:**

**return** **[**

"Comparing Foo instances:"**,**

f" vals: {left**.**val} != {right**.**val}"**,**

**]**



Fixtures:

*def.n :*

In testing, a [fixture](https://en.wikipedia.org/wiki/Test_fixture#Software) provides a defined, reliable and consistent context for the tests. This could include environment (for example a database configured with known parameters) or content (such as a dataset).

Fixtures define the steps and data that constitute the *arrange* phase of a test (see [Anatomy of a test](https://docs.pytest.org/en/stable/explanation/anatomy.html#test-anatomy)). In pytest, they are functions you define that serve this purpose. They can also be used to define a test’s *act* phase; this is a powerful technique for designing more complex tests

# Fixtures:

Test functions requests fixtures by declaring them as arguments .

When running ,pytest look sofr the fixtures passed as args to test case function.

Look at : C:\Users\mutay\OneDrive\Documents\courses\PyATB\WebAutomation\GeM-project\pages\test\_example.py

# Use case of fixtures: Sharing test data[¶](https://docs.pytest.org/en/stable/explanation/fixtures.html#sharing-test-data)

If you want to make test data from files available to your tests, a good way to do this is by loading these data in a fixture for use by your tests. This makes use of the automatic caching mechanisms of pytest.

Another good approach is by adding the data files in the tests folder. There are also community plugins available to help to manage this aspect of testing, e.g. [pytest-datadir](https://pypi.org/project/pytest-datadir) and [pytest-datafiles](https://pypi.org/project/pytest-datafiles)

A fixture can request other fixture.

A test or a fixture can request more than one fixture at a time.

Q, : Fixtures can request other fixtures

can this priciple b eused to navigate between the pages while testing in web automation uisng Selenium or playwright