`xfail` and `skip`
What to do with tests you know will fail

Paul Ganssle

@pganssle

This talk on Github: pganssle-talks/pytexas-2022-xfail

@pganssle

ganssle.io

`skipif` vs `xfail`

`skipif` — Use for tests that *are supposed to fail*.

```
@pytest.mark.skipif(sys.version_info < (3, 8),
                    reason="Module-level __getattr__ introduced in Python 3.8")
def test_module_getattr_works():
    with clear_dateutil_imports():
        import dateutil
        assert dateutil.tz is not None  # Lazy-loaded
```

`xfail` — Use for tests that *currently fail, but shouldn't*.

```
@pytest.mark.xfail(reason="Fractional hours and minutes not implemented yet!")
def test_fractional_hour():
    # ISO 8601 allows fractional hours and minutes
    assert (dateutil.parser.isoparse("2021-03-26T14.5") ==
            datetime.datetime(2021, 3, 26, 14, 30))
```

jbrockmendel commented on Oct 30, 2017 — Contributor

This is a class of cases that I intend to fix, but its not high on the priority list. As **@pganssle** mentioned, its a re-write as opposed to a bug-fix.

One way you could help keep this from drifting down the todo list would be to submit a PR creating a test for this case marked as `xfail`.

pganssle commented on Oct 31, 2017 — Member

I'm going to be honest, I don't love `xfail` tests, but I am willing to be persuaded about that.

From `dateutil` issue #487

@pganssle
ganssle.io

Paul Ganssle

Archives for Paul Ganssle

MON 13 DECEMBER 2021

`xfail` and code coverage

MON 22 NOVEMBER 2021

A pseudo-TDD workflow using expected failures

TUE 09 NOVEMBER 2021

How and why I use pytest's `xfail`

Atom Feed

RSS Feed

OTHER PAGES

Archive

About

SOCIAL

Homepage

@pganssle

ganssle.io

Example: Perfect square function

```python
import math

def is_perfect_square(n: int) -> bool:
    """Determine if any int i exists such that i × i = n."""
    s = math.sqrt(n)
    return s == int(s)
```

With tests:

```python
import square_mod

import pytest


@pytest.mark.parametrize("n", [0, 1, 2, 4, 9, 16, 25, 36])
def test_squares(n):
    assert square_mod.is_perfect_square(n)


@pytest.mark.parametrize("n", [3, 5, 6, 7, 8, 27, 32])
def test_non_squares(n):
    assert not square_mod.is_perfect_square(n)
```

```
============================================= test session starts ==============================================
platform linux -- Python 3.10.0, pytest-7.1.1, pluggy-1.0.0
collected 14 items

test_square_mod.py ..............                                                                        [100%]

============================================== 14 passed in 0.02s ==============================================
```

@pganssle
ganssle.io

Found a bug: Add a test

```python
def test_negative():
    assert not square_mod.is_perfect_square(-4)
```

```
============================================= test session starts ==============================================
platform linux -- Python 3.10.0, pytest-7.1.1, pluggy-1.0.0
collected 15 items

test_square_mod.py ..............F                                                                      [100%]

=================================================== FAILURES ===================================================
_____________________________________________ test_negative ______________________________________________

    def test_negative():
>       assert not square_mod.is_perfect_square(-4)

test_square_mod.py:17:
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

n = -4

    def is_perfect_square(n: int) -> bool:
        """Determine if any int i exists such that i × i = n."""
>       s = math.sqrt(n)
E       ValueError: math domain error

square_mod.py:5: ValueError
=========================================== short test summary info ============================================
FAILED test_square_mod.py::test_negative - ValueError: math domain error
========================================== 1 failed, 14 passed in 0.08s ========================================
```

@pganssle
ganssle.io

xfail: Tests that are *expected* to fail

```python
@pytest.mark.xfail(reason="Bug #11493: Negative values not supported!")
@pytest.mark.parametrize("n", [-1, -3, -4])
def test_negative(n):
    # When called with a negative value for n, this test raises ValueError!
    assert not square_mod.is_perfect_square(m)
```

Failure is expected, so the tests pass:

```
============================================= test session starts ==============================================
platform linux -- Python 3.10.2, pytest-7.1.1, pluggy-1.0.0
plugins: subtests-0.7.0, runtime-xfail-1.0.3, cov-3.0.0, hypothesis-6.39.4
collected 17 items

test_square_mod.py ..............xxx                                                                    [100%]

========================================== 14 passed, 3 xfailed in 0.03s ===========================================
```

@pganssle
ganssle.io

XFAIL becomes XPASS

```python
def is_perfect_square(n: int) -> bool:
    """Determine if any real integer i exists such that i × i = n."""

    # Negative numbers are not squares according to the definition of the function
    if n < 0:
        return False

    s = math.sqrt(n)
    return s == int(s)
```

```
============================================= test session starts ==============================================
platform linux -- Python 3.10.0, pytest-7.1.1, pluggy-1.0.0
collected 15 items

test_square_mod.py ..............X                                                                  [100%]

========================================== 14 passed, 1 xpassed in 0.02s ===========================================
```

@pganssle
ganssle.io

Treating failure to fail as a failure

```python
@pytest.mark.xfail(strict=True,
                   raises=ValueError,
                   reason="Bug #11493: Negative values not supported")
@pytest.mark.parametrize("n", [-1, -3, -4])
def test_negative(n):
    assert not square_mod.is_perfect_square(n)
```

```
============================================= test session starts ==============================================
platform linux -- Python 3.10.0, pytest-7.1.1, pluggy-1.0.0
collected 17 items

test_square_mod.py ..............FFF                                                                  [100%]

=================================================== FAILURES ===================================================
_____________________________________________ test_negative[-1] _____________________________________________
[XPASS(strict)] Bug #11493: Negative values not supported
_____________________________________________ test_negative[-3] _____________________________________________
[XPASS(strict)] Bug #11493: Negative values not supported
_____________________________________________ test_negative[-4] _____________________________________________
[XPASS(strict)] Bug #11493: Negative values not supported
=========================================== short test summary info ============================================
FAILED test_square_mod.py::test_negative[-1]
FAILED test_square_mod.py::test_negative[-3]
FAILED test_square_mod.py::test_negative[-4]
========================================== 3 failed, 14 passed in 0.03s ==========================================
```

@pganssle
ganssle.io

Make `strict=True` the default:

In `tox.ini` / `pytest.ini`:

```
[pytest]
xfail_strict = True
```

In `setup.cfg`:

```
[tool:pytest]
xfail_strict = True
```

In `pyproject.toml`:

```
[tool.pytest.ini_options]
xfail_struct = True
```

See the pytest documentation for the latest options for global configuration, or the documentation on the `xfail` option.

Being specific about *why* the test is failing

```python
@pytest.mark.xfail(reason="Bug #11493: Negative values not supported!")
@pytest.mark.parametrize("n", [-1, -3, -4])
def test_negative(n):
    # When called with a negative value for n, this test raises ValueError!
    assert not square_mod.is_perfect_square(m)
```

```
============================================= test session starts ==============================================
platform linux -- Python 3.10.2, pytest-7.1.1, pluggy-1.0.0
plugins: subtests-0.7.0, runtime-xfail-1.0.3, cov-3.0.0, hypothesis-6.39.4
collected 17 items

test_square_mod.py ..............xxx                                                                  [100%]

========================================== 14 passed, 3 xfailed in 0.03s ===========================================
```

@pganssle
ganssle.io

Being specific about *why* the test is failing

```python
@pytest.mark.xfail(reason="Bug #11493: Negative values not supported!")
@pytest.mark.parametrize("n", [-1, -3, -4])
def test_negative(n):
    # When called with a negative value for n, this test raises ValueError!
    assert not square_mod.is_perfect_square(m)
```

```
============================================= test session starts ==============================================
platform linux -- Python 3.10.2, pytest-7.1.1, pluggy-1.0.0
plugins: subtests-0.7.0, runtime-xfail-1.0.3, cov-3.0.0, hypothesis-6.39.4
collected 17 items

test_square_mod.py ..............xxx                                                                  [100%]

========================================== 14 passed, 3 xfailed in 0.03s ===========================================
```

Actually failing because of `NameError`

Being specific about *why* the test is failing

```python
@pytest.mark.xfail(
    raises=ValueError,
    reason="Bug #11493: Negative values not supported!"
    )
@pytest.mark.parametrize("n", [-1, -3, -4])
def test_negative(n):
    # When called with a negative value for n, this test raises ValueError!
    assert not square_mod.is_perfect_square(m)
```

```
============================================= test session starts ==============================================
platform linux -- Python 3.10.0, pytest-7.1.1, pluggy-1.0.0
collected 17 items

test_square_mod.py .............FFF                                                                    [100%]

================================================== FAILURES ==================================================
_____________________________________________ test_negative[-1] _____________________________________________
test_square_mod.py:21: in test_negative
    assert not square_mod.is_perfect_square(m)
E   NameError: name 'm' is not defined
_____________________________________________ test_negative[-3] _____________________________________________
test_square_mod.py:21: in test_negative
    assert not square_mod.is_perfect_square(m)
E   NameError: name 'm' is not defined
_____________________________________________ test_negative[-4] _____________________________________________
test_square_mod.py:21: in test_negative
    assert not square_mod.is_perfect_square(m)
E   NameError: name 'm' is not defined
=========================================== short test summary info ============================================
FAILED test_square_mod.py::test_negative[-1] - NameError: name 'm' is not defined
FAILED test_square_mod.py::test_negative[-3] - NameError: name 'm' is not defined
FAILED test_square_mod.py::test_negative[-4] - NameError: name 'm' is not defined
========================================= 3 failed, 14 passed in 0.09s =========================================
```

@pganssle
ganssle.io

Why should I care?

Document your acceptance criteria

Adding a failing test to the test suite documents the conditions necessary to fix the bug.

Why should I care?

Document your acceptance criteria

Adding a failing test to the test suite documents the conditions necessary to fix the bug.

Test your tests!

Start by writing the xfailing test, if it doesn't fail, your test won't catch the regression.

Why should I care?

Document your acceptance criteria

Adding a failing test to the test suite documents the conditions necessary to fix the bug.

Test your tests!

Start by writing the xfailing test, if it doesn't fail, your test won't catch the regression.

Impose regression tests early!

Your test suite will fail if you accidentally fix a bug - remove the `xfail` and you'll prevent regressions from the mo
merged!

Good bug reports contain an example that is:

Complete

As much code as you need to reproduce the issue.

Minimal

As *little* code as you can.

Verifiable

Anyone can use the example to reproduce the failure.

@pganssle
ganssle.io

Good bug reports contain an example that is:

Complete

As much code as you need to reproduce the issue.

Minimal

As *little* code as you can.

Verifiable

Anyone can use the example to reproduce the failure.

These are also the properties of a good test!

@pganssle
ganssle.io

Using `xfail` and `skipif`: Markers

Bare decorator (with or without `reason`)

```python
@pytest.mark.xfail
def test_my_failing_function():
    assert my_function(-3) == 2


@pytest.mark.xfail(reason="NaN handling is not working properly yet.")
def test_float_function():
    float_function(float("nan"))


@pytest.mark.skip(reason="Probably shouldn't do unconditional skipping")
def test_this_is_pointless():
    some_random_function()
```

Using `xfail` and `skipif`: Markers

Boolean condition

```python
@pytest.mark.xfail(sys.version_info > (3, 10),
                   reason="AST handling changed in Python 3.10")
def test_some_ast_tomfoolery():
    my_ast_tomfoolery_function("path/to/python_file.py")


@pytest.mark.skipif(sys.version_info < (3, 9),
                    reason="zoneinfo introduced in Python 3.9")
def test_zoneinfo():
    import zoneinfo
    tz = zoneinfo.ZoneInfo("America/Chicago")

    ...
```

Condition string

```python
@pytest.mark.skipif("not hasattr(os, 'fspath')")
def path_normalization():
    assert os.fspath(MyPathClass("a/b/c")) == "a/b/c"
```

Condition strings are discouraged

@pganssle
ganssle.io

Using `xfail` and `skipif`: Markers

The `run` parameter

```python
@pytest.mark.xfail(
    run=False
    reason="Test will segfault if run!"
)
def test_surrogate_characters():
    parsed = datetime.fromisoformat("2022-03-26\ud800)13:15:04")
    expected = datetime(2022, 3, 26, 13, 15, 4)
    assert parsed == expected
```

```python
@pytest.mark.xfail(
    sys.version_info < (3, 9),
    run=False,
    reason="Test segfaults earlier than Python 3.9"
)
def test_conditional_segfault():
    function_that_segfaults_on_38()
```

@pganssle
ganssle.io

Using `xfail` and `skipif`: Parameterized tests

```python
@pytest.mark.parametrize("n", [
    1,
    pytest.param(0,
        marks=pytest.mark.xfail(
            raises=ZeroDivisionError,
            reason="Zero not handled correctly"
        )
    ),
    pytest.param(-1,
        marks=pytest.mark.xfail(
        reason="Not working for negative numbers."
        )
    ),
    5,
])
def test_is_euler_number(n):
    assert is_euler_number(n)
```

```
============================================= test session starts ==============================================
platform linux -- Python 3.10.2, pytest-7.1.1, pluggy-1.0.0
collected 4 items

test_euler_numbers.py .xx.                                                                             [100%]

========================================== 2 passed, 2 xfailed in 0.03s ==========================================
```

@pganssle
ganssle.io

Using `xfail` and `skip`

```
@pytest.mark.parametrize("payload", list(
    pathlib.Path.glob("test_data/test_data_*")
))
def test_process_data_files(payload):
    with open(payload, "rb") as f:
        test_data = load_data(f)

    if test_data.num_elements > 255:
        # Don't actually use this! See next slide!
        pytest.xfail(reason="bug #2445: Processing of data files with > 255 fails.")
```

Can embed these into functions, context managers and fixtures

```
@contextlib.contextmanager
def in_local_timezone(tz: str) -> Iterator[None]:
    if sys.platform.startswith("win") or os.eviron.get("TZ_CHANGE_DISALLOWED") == "true":
        pytest.skip("Time zone change not allowed")

    old_tz = get_current_tz()
    set_tz(tz)
    time.tzset()
    yield
    set_tz(old_tz)
    time.tzset()


def test_in_new_york():
    # This test is skipped automatically in platforms and environments that
    # don't support changing the time zone.
    with in_local_timezone("America/New_York"):
        assert my_date_function("2021-01-01") > my_date_function("2020-01-01")
```

Papercut: `pytest.xfail()` is basically skip!

- `pytest.xfail()` stops execution immediately — there is no way to get XPASS
- `xfail_strict` has no effect
- This is deliberate

Solution: `pytest-runtime-xfail`

```
@pytest.mark.parametrize("payload", list(
    pathlib.Path.glob("test_data/test_data_*")
))
def test_process_data_files(payload, runtime_xfail):
    with open(payload, "rb") as f:
        test_data = load_data(f)

    if test_data.num_elements > 255:
        # Don't actually use this! See next slide!
        runtime_xfail(reason="bug #2445: Processing of data files with > 255 fails.")
```

Papercut: `xfail` with `hypothesis`

```python
import hypothesis
from hypothesis import strategies as st

@hypothesis.given(n=st.integers())
def test_my_function(n: int) -> None:
    if n < 0:
        pytest.xfail("Zero and negative numbers not working!")

    assert my_function(n) > n
```

Problems:

- Reports a single top-level `XFAIL`:

```
============================================= test session starts ==============================================
platform linux -- Python 3.10.2, pytest-7.1.1, pluggy-1.0.0
plugins: hypothesis-6.39.4
collected 1 item

test_hypothesis.py x                                                                                    [100%]

============================================== 1 xfailed in 0.22s ==============================================
```

- Disguises real failures in non-`xfail` cases
- Doesn't work with `pytest-runtime-xfail`

@pganssle
ganssle.io

Using `unittest`

```python
import unittest

class MyTest(unittest.TestCase):
    def test_pass(self):
        self.assertTrue(True)

    @unittest.expectedFailure
    def test_xfail(self):
        self.assertTrue(False)

    @unittest.expectedFailure
    def test_xpass(self):
        self.assertTrue(True)
```

Run with `unittest`:

```
$ python -m unittest -v
test_pass (test_unittest.MyTest) ... ok
test_xfail (test_unittest.MyTest) ... expected failure
test_xpass (test_unittest.MyTest) ... unexpected success

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (expected failures=1, unexpected successes=1)
```

@pganssle
ganssle.io

Using `unittest`

```python
import unittest

class MyTest(unittest.TestCase):
    def test_pass(self):
        self.assertTrue(True)

    @unittest.expectedFailure
    def test_xfail(self):
        self.assertTrue(False)

    @unittest.expectedFailure
    def test_xpass(self):
        self.assertTrue(True)
```

Run with `pytest`:

```
$ pytest test_unittest.py
============================================= test session starts ==============================================
platform linux -- Python 3.10.2, pytest-7.1.1, pluggy-1.0.0
collected 3 items

test_unittest.py .xF                                                                                    [100%]

=================================================== FAILURES ===================================================
_____________________________________________ MyTest.test_xpass _____________________________________________
Unexpected success
=========================================== short test summary info ============================================
FAILED test_unittest.py::MyTest::test_xpass
======================================== 1 failed, 1 passed, 1 xfailed in 0.04s ========================================
```

@pganssle
ganssle.io

`unittest`: Missing features

- No `strict=False` or `run=False`
- No support for `raises`
- No way to specify a reason
- Conditional `xfail` not built-in:

```
def conditional_xfail(condition):
    if condition:
        return unittest.expectedFailure
    else:
        return lambda x: x
```

@pganssle
ganssle.io

Skipping with `unittest`

Decorators:

```
class MySkipTest(unittest.TestCase):
    @unittest.skip("Unconditional skipping")
    def skip_unconditional(self):
        pass

    @unittest.skipIf(sys.platform.startswith("win"), "Not supported on Windows")
    def test_nix_only(self):
        ...

    @unittest.skipUnless(sys.platform.startswith("win"), "Only supported on Windows")
    def test_windows_only(self):
        ...
```

At runtime:

```
    def test_three_day_weekend(self):
        if 4 <= datetime.now().weekday() <= 6:
            self.skipTest("This test has negotiated a 3-day weekend.")

        ...



    def test_api_integration(self):
        r = requests.get("https://url.to/some/resource")
        if r.status_code != 200:
            raise unittest.SkipTest("Resource not available")

        ...
```

@pganssle
ganssle.io

Summary

- Use `xfail` for tests that you know are failing, but shouldn't be.
- Always use `skip` conditionally — it's for tests that aren't *supposed* to pass.
- Set `xfail_strict=True`

Further reading

- "How and why I use pytest's `xfail`": https://blog.ganssle.io/articles/2021/11/pytest-xfail.html
- "A pseudo-TDD workflow using expected failures": https://blog.ganssle.io/articles/2021/11/pseudo-tdd-xfail
- "`xfail` and code coverage": https://blog.ganssle.io/articles/2021/12/xfail-coverage.html
- `pytest-runtime-xfail`: https://pypi.org/project/pytest-runtime-xfail/
- Also featured on Test & Code: https://testandcode.com/guests/paul-ganssle

@pganssle
ganssle.io