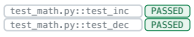
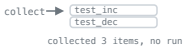
# pytest

A quick reference for pytest, from plain assert tests to fixtures, parametrization, and markers that keep a suite fast and readable.



## Write & Run Your First Test

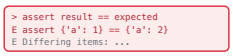
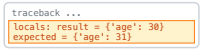
A test is a function that asserts; pytest finds and runs it.

| PURPOSE | COMMAND | RESULT |
|---|---|---|
| Write a test (plain assert). | `def test_inc():`<br>`  assert inc(3) == 4` |  |
| Run the whole suite. | `$ pytest` |  |
| Run quietly (one line). | `$ pytest -q` |  |
| Run one file, then one test. | `$ pytest test_math.py::test_inc` |  |
| Verbose: show each node id. | `$ pytest -v` |  |
| See what would run, run nothing. | `$ pytest --collect-only` |  |

pytest discovers every test_*.py file and test_* function; narrow with a path or node id, -q for quiet, -v for node ids

## Read the Failure

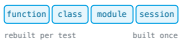
Assert rewriting shows you exactly what differed.

| PURPOSE | COMMAND | RESULT |
|---|---|---|
| A test fails (F in the strip). | `$ pytest` |  |
| Read the rewritten assert. | `assert result == expected` |  |
| Show more of the diff. | `$ pytest -vv` |  |
| Drop into a debugger on failure. | `$ pytest --pdb` |  |
| Show local variables. | `$ pytest -l` |  |
| Stop after the first failure. | `$ pytest --maxfail=1 (-x)` |  |

a failed assert is rewritten to show both sides and a focused diff; -vv for the full diff, -l for locals, --pdb to debug, -x to stop early

## Fixtures & the Dependency Graph

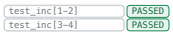
Fixtures provide setup; tests request them by name.

| PURPOSE | COMMAND | RESULT |
|---|---|---|
| Define a fixture. | `@pytest.fixture`<br>`def numbers(): return [1, 2, 3]` |   |
| Request it by parameter name. | `def test_total(numbers): ...` |   |
| Fixtures depend on fixtures. | `@pytest.fixture def total(numbers): ...` |   |
| Setup then teardown with yield. | `conn = connect();yield conn; conn.close()` |   |
| Control reuse with scope. | `@pytest.fixture(scope="module")` |   |
| Share fixtures across files. | `# put it inconftest.py` |   |

a test requests a fixture by naming it as a parameter; yield adds teardown, scope controls reuse, conftest.py auto-shares

## Parametrize Many Inputs

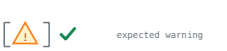
One test body, many cases; each case is its own node.

| PURPOSE | COMMAND | RESULT |
|---|---|---|
| Run over a table of cases. | `@pytest.mark.parametrize("x, exp",`<br>`[(1,2),(2,3),(3,4)])` |   |
| See generated case ids. | `$ pytest -v` |   |
| Give a case a readable id. | `pytest.param(5, 6, id="five")` |   |
| Mark a single case xfail. | `pytest.param(0, 1,`<br>`marks=pytest.mark.xfail)` |   |
| Stack params for a grid. | `@parametrize("a", [1, 2])`<br>`@parametrize("b", [10, 20])` |   |
| Parametrize a fixture instead. | `@pytest.fixture(params=[1, 2, 3])` |   |

each parametrize row is its own node with its own id; stack decorators for the Cartesian product, or use fixture params

## Test Raises & Warns

Assert that code raises an exception or emits a warning.

| PURPOSE | COMMAND | RESULT |
|---|---|---|
| Assert an exception is raised. | `with pytest.raises(ZeroDivisionError): 1/0` |   |
| Match the exception message. | `pytest.raises(ValueError, match=r"invalid")` |   |
| Inspect the caught exception. | `pytest.raises(ValueError)as excinfo: ...` |   |
| Assert a warning is emitted. | `with pytest.warns(UserWarning): ...` |   |
| Float comparisons that wobble. | `assert 0.1 + 0.2 ==pytest.approx(0.3)` |   |
| Capture warnings to inspect. | `def test_w(recwarn):`<br>`  assert len(recwarn) == 1` |   |

wrap failing code in with pytest.raises(...); pin the message with match=, capture via as excinfo; warns/approx for warnings/floats

## Markers, Selection, Skip / xfail

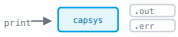
Tag tests, then run the subset you want.

| PURPOSE | COMMAND | RESULT |
|---|---|---|
| Tag a test with a marker. | `@pytest.mark.slow def test_big(): ...` |   |
| Run only tagged tests. | `$ pytest -m slow` |   |
| Select by name substring. | `$ pytest -k "raises or warns"` |   |
| Skip unconditionally. | `@pytest.mark.skip(reason="not ready")` |   |
| Skip on a condition. | `@pytest.mark.skipif(`<br>`  sys.platform=="win32", ...)` |   |
| Expect a known failure. | `@pytest.mark.xfail(reason="bug #123")` |   |

mark tests with @pytest.mark.<name>, select with -m or -k; skip / skipif drop a test, xfail flags a known failure

## Isolate Side Effects

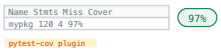
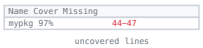
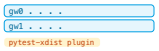
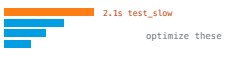
Built-in fixtures hand you a clean temp dir, env, and captured output.

| PURPOSE | COMMAND | RESULT |
|---|---|---|
| Get a fresh temp directory. | `def test_io(tmp_path):`<br>`  (tmp_path/"f").write_text(...)` |  |
| Set an env var safely. | `monkeypatch.setenv("API_KEY", "x")` |  |
| Replace an attribute / function. | `monkeypatch.setattr("mod.now", lambda: fixed)` |  |
| Capture stdout / stderr. | `print("hi");capsys.readouterr().out` |  |
| Assert on log records. | `def test_log(caplog):`<br>`  "boom" in caplog.text` |  |
| Mock with a fixture (plugin). | `def test_call(mocker):`<br>`  mocker.patch("mod.send")` |  |

built-in fixtures isolate each test: tmp_path (clean dir), monkeypatch (env/attrs, auto-reverted), capsys/caplog (captured output)

## Measure & Speed Up

See coverage, fail fast, parallelize, re-run only failures.

| PURPOSE | COMMAND | RESULT |
|---|---|---|
| Measure line coverage. | `$ pytest --cov=mypkg` |  |
| Show which lines are missed. | `--cov=mypkg --cov-report=term-missing` |  |
| Stop on the first failure. | `$ pytest -x` |  |
| Re-run only what failed. | `$ pytest --lf` |  |
| Run tests in parallel. | `$ pytest -n auto` |  |
| Find the slowest tests. | `$ pytest --durations=10` |  |

--cov (pytest-cov) reports coverage; -x stops early, --lf reruns failures, -n auto parallelizes (pytest-xdist), --durations finds slow tests