Unit testing & Code coverage

What is Unit Testing?

According to wikipedia, unit testing is a software testing method by which individual units of source code are tested to determine whether they are fit for use.

Why Unit Test?

- Tests Reduce Bugs in New Features and Existing Features
- Tests Are Good Documentation
- Tests Reduce the Cost of Change
- Faster Debugging
- Faster Development
- Better Design

Python Testing frameworks

unittest

- In the Python Standard Library

nose

- Not in the Standard Library. Simpler tests than unittest

pytest

- Not in the Python Standard Library.

PyTest

Installation

- pip install pytest

General Information

- pytest -h
- Use test as prefix in each function

Test all files

- py.test
- Automatically recognize all files starting with prefix test
- Use test as prefix in test filename

Test specific file

- pytest [options] [filename]
- pytest filename -v

Pytest flags

-k : matches with expression

- pytest -k “add or string”

-m : matches with marker

- Add decorator
- pytest -m [marker expression]

Test a function

- pytest [filename]::[function name]

Pytest flags

-x : stops after failed execution

- pyest -x

--maxfail==[value] : stops after maxfail number of execution

- pytest --maxfail=2

--tb=no : only failure or success

- pytest --tb=no

-s / –capture=no : enable print statement

Pytest - Xfail/Skip Tests

@pytest.mark.xfail

Pytest will execute the xfailed test, but it will not be considered as part failed or passed tests. Details of these tests will not be printed even if the test fails (remember pytest usually prints the failed test details).

@pytest.mark.skip

Skipping a test means that the test will not be executed.

@pytest.mark.skip(reason=”Do not test this for now”)

Show the reason why it is skipped

@pytest.mark.skipif(sys.version_info > (3, 3), reason='Do not run this now')

Skip when the condition is satisfied and show the reason

Pytest - Parameterizing Tests

@pytest.mark.parametrize

- to run the test against multiple sets of inputs.

Example:

```
@pytest.mark.parametrize("num, output",[(1,11),(2,22),(3,35),(4,44)])
def test_multiplication_11(num, output):
   assert 11*num == output
```

Code Coverage

- measure of exhaustiveness of a test suite.
- 100% code coverage means that a system is fully tested.
- Theoretically, the higher code coverage is, the fewer defects a system has. Of course, tests are not enough to catch all kinds of errors, but in this uneven battle, we need all the help we can get.

Commands

Installation

- pip install pytest-cov

runs pytest using coverage run

- coverage run -m pytest

generates the coverage report

- coverage report -m

Types of code coverage

Statement coverage

- measuring executed lines of code won't cut it
- `(number_of_code_lines_run_at_least_once_under_tests / total_number_of lines) * 100%`

Branch coverage

- when there is decision-making whether to do one or another thing, we call it branching.
- possible code paths are called branches
- `(number_of_branches_executed_at_least_once_under_tests / all_branches) * 100%`

Condition coverage

- Condition coverage assumes that in order to achieve 100% code coverage, the test suite needs to check situations in which every expression is True and False.
- `(number_of_executed_bool_states_of_operands / number_of_all_operands * 2) * 100%`

Resources

Pytest

https://www.tutorialspoint.com/pytest/pytest_parameterizing_tests.htm

https://docs.pytest.org/en/7.1.x/contents.html

Code Coverage:

https://medium.com/@sumanrbt1997/code-coverage-with-pytest-1f72653b0bf2

https://breadcrumbscollector.tech/how-to-use-code-coverage-in-python-with-pytest/