Тестирование PYTHON приложений

План

1. Источник

2. Тестирование для чего нужно

3. Основные виды тестирования

4. Интеграционное тестирование

5. Модульное (юнит) тестирование

6. Требование к тестам

Покрытие тестами

1. Источник

1. [https://habr.com/ru/post/269759/#:~:text=2)%20%40pytest.,%D1%82%D0%B5%D1%81%D1%82%20%D0%B1%D1%83%D0%B4%D0%B5%D1%82%20%D0%BF%D0%BE%D0%BC%D0%B5%D1%87%D0%B0%D1%82%D1%8C%D1%81%D1%8F%20%D0%B4%D0%B0%D0%BD%D0%BD%D0%BE%D0%B9%20%D0%BC%D0%B5%D1%82%D0%BA%D0%BE%D0%B9](https://habr.com/ru/post/269759/" \l ":~:text=2) @pytest.,тест будет помечаться данной меткой).
2. https://devpractice.ru/unit-testing-in-python-part-1/
3. https://habr.com/ru/company/otus/blog/481806/
4. https://webdevblog.ru/vvedenie-v-testirovanie-v-python/

2.Тестирование для чего нужно

Прежде чем писать тесты, нужно сначала принять пару решений:

1. Что вы хотите проверить?
2. Вы пишете юнит-тест или интеграционный тест?

Структура теста должна следовать этому рабочему процессу:

1. Определитесь и создайте входные данные
2. Осуществите проверку
3. Сравните результат с ожидаемым результатом

Негласное правило - если на участке кода проявилась ошибка (Баг), то пишим тест на этот случай!

Тесты меняют при изменении кода!

Код не меняют при изменении тестов!

3. Основные виды тестирования

* Отладка с помощью print()
* Использование визуального отладчика Python
* Юнит тестирование с помощью pytest и mocks
* Интеграционное тестирование

4. Интеграционное тестирование

Тестирование нескольких компонентов называется **интеграционным тестированием (integration testing) или** функциональным тестированием.

**Интеграционный тест**, который проверяет, что компоненты в вашем приложении правильно работают друг с другом.

Основная проблема с интеграционным тестированием — это когда интеграционный тест не дает правильного результата.

Oчень трудно диагностировать проблему, не имея возможности определить, какая часть системы вышла из строя.

Интеграционно­е тестирование может потребовать от пользователя или пользователя приложения:

* Вызов HTTP REST API
* Вызов Python API
* Вызов веб сервисов
* Запуск командной строки
* Наличие рабочей тестовой бази данних

5. Модульное (юнит) тестирование

Модульный тест **(unit test)** — это как правило небольшой тест, который проверяет правильность работы отдельного компонента.

Модульный тест поможет вам выделить то, что сломано в вашем приложении, и быстро это исправить.

### 6. Требование к тестам

* тест должен проверять только отдельно-взятую функцию (метод) - Атомарность
* тесты должны быть как можно мельче
* Если внутри проверяемого функционала много условий(валидаторы), то каждое из условий проверяеться в отдельном тесте.
* Покрытие тестами должно быть как можно больше (количество тестов как можно больше)
* тесты должны охватывать как положительные так и отрицательные сценарии.
* Tестирование не должно влиять на состояние системы (Базу Данных)
* Изолированость

### 7. Побочные эффекты

Часто выполнение фрагмента кода во время тестирования изменяет другие вещи в программе, такие как атрибут класса, файл в файловой системе или значение в базе данных. Это называется **побочные эффекты (side effects)** и являются важной частью тестирования. Решите, проверяется ли побочный эффект, прежде чем включать его в свой список утверждений.

Если вы обнаружите, что блок кода, который вы хотите протестировать, имеет много побочных эффектов, возможно, вы нарушаете [принцип единой ответственности](https://en.wikipedia.org/wiki/Single_responsibility_principle). Нарушение принципа единой ответственности означает, что фрагмент кода выполняет слишком много задач, и было бы лучше, если бы он подвергся рефакторингу. Следование принципу единой ответственности — это отличный способ разработки кода, позволяющего легко создавать повторяемые и простые модульные тесты и, в конечном счете, надежные приложения.