Тестирование в Python

Гейне М.А.

09.10.2024

Тестирование кода — это **процесс анализа и проверки исходного кода программного обеспечения с целью обнаружения и исправления ошибок, улучшения структуры кода и повышения его эффективности**. Это важный этап в разработке ПО, который помогает обеспечить стабильность, безопасность и производительность продукта.

Тестирование бывает:

- Ручное
- Автоматическое

Ручное тестирование

Ручное тестирование кода — это процесс анализа кода человеком без использования автоматизированных инструментов. Для ручного тестирования программа запускается и исполняется с различными данными для проверки работоспособности программы, корректности исполнения и т.д.

Автоматическое тестирование

Автоматическое тестирование кода — это использование автоматизированных инструментов и тестовых сценариев для выполнения тестов на коде. Автоматическое тестирование может значительно ускорить процесс тестирования и улучшить его точность.

Автоматическое тестирование позволяет проверять программу без непосредственного участия человека, с различными данными, в разных средах и т.д.

Типы автоматического тестирования

1. **Unit test** - тестирование отдельных элементов. В рамках unit тестирования проверяется модуль, класс, функция или другой программный элемент на соответствие ожидаемым характеристикам и поведению
2. **Integration test** - интеграционное тестирование программы, как объединения различных элементов. Проверяется, что использование отдельно взятых и *корректных* модулей вместе не вызывает ошибок и приводит к желаемому результату

3. **Функциональные тесты**: Удостоверяют, что программное обеспечение работает в соответствии с заданными требованиями, часто тестируя функцию или функцию в целом.
4. **Тесты "end-to-end" (E2E)**: Моделирование реальных пользовательских сценариев для проверки работы приложения от начала до конца.
5. **Регрессионные тесты**: Убедитесь, что новые изменения в коде не оказывают негативного влияния на существующие функциональные возможности.
6. **Тесты производительности**: Оценивают скорость, отзывчивость и стабильность работы при определенной рабочей нагрузке.
7. **Нагрузочные тесты**: Измеряют поведение системы под ожидаемой нагрузкой, чтобы убедиться, что она может справиться с высоким трафиком.
8. **Стресс-тесты**: Выведите систему за пределы ее возможностей, чтобы определить точку разрыва и посмотреть, как она восстанавливается.
9. **Тесты безопасности**: Проверьте уязвимости и убедитесь, что требования безопасности соблюдены.
10. **Тесты пользовательского интерфейса (UI)**: Проверяют, что элементы пользовательского интерфейса функционируют так, как ожидается.
11. и многие другие

Assert

```
>>> assert sum([1, 2, 3]) == 6, "Should be 6"
```

```
>>> assert sum([1, 1, 1]) == 6, "Should be 6"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError: Should be 6
```

```
def test_sum():
    assert sum([1, 2, 3]) == 6, "Should be 6"

if __name__ == "__main__":
    test_sum()
    print("Everything passed")
```

Pytest

Pytest - это популярный фреймворк для тестирования на Python, который широко используется для написания и выполнения тестов. Он разработан как простой и масштабируемый, поддерживающий как небольшие модульные тесты, так и сложное функциональное тестирование.

1. **Простой синтаксис**: Pytest использует минималистичный подход к написанию тестов. Тесты определяются как функции с использованием простых утверждений (`assert`), что делает их простыми для написания и чтения.
2. **Автообнаружение**: Pytest автоматически обнаруживает тестовые файлы и функции, основываясь на соглашениях об именовании (файлы `test_*.py` и функции с префиксом `test_`). Это позволяет легко организовывать и выполнять тесты без ручной настройки.
3. **Фикстуры**: Фикстуры Pytest позволяют создавать и разрушать тестовые среды. Фикстуры могут быть общими для всех тестов, использоваться условно и поддерживать инъекцию зависимостей, что позволяет легко управлять процессом создания и разрушения сложных тестовых

5. **Параметризация**: Pytest позволяет запускать тесты с разными наборами входных данных с помощью параметризации. Это полезно для тестирования нескольких случаев с одной и той же тестовой логикой, сокращая дублирование кода.
6. **Плагины и расширения**: Pytest имеет богатую экосистему плагинов, расширяющих его возможности (например, pytest-django, pytest-cov для покрытия). Пользователи также могут создавать собственные плагины для конкретных нужд.
7. **Интеграция с CI/CD**: Pytest хорошо интегрируется с инструментами непрерывной интеграции и непрерывной доставки (CI/CD), что делает его отличным выбором для автоматизированного тестирования в конвейерах разработки.

9. **Подробные отчеты**: Pytest предоставляет подробные и понятные результаты, включая трассировку стека, выделение неудачных утверждений и показ только самой важной информации для отладки.
10. **Поддержка различных типов тестов**: Pytest универсален и может использоваться для модульных тестов, интеграционных тестов, функциональных тестов и многого другого. Он достаточно гибок, чтобы адаптироваться к различным потребностям тестирования в проекте.

Пример

```python
# test_example.py

def add(x, y):
    return x + y

def test_add():
    assert add(2, 3) == 5
    assert add(-1, 1) == 0
    assert add(0, 0) == 0
```

```
pytest
```

```
============================================= test session starts ==============================================
platform linux -- Python 3.10.12, pytest-8.2.0, pluggy-1.5.0
rootdir: /home/mikegeine/assignments/text_analysis
configfile: pytest.ini
plugins: anyio-3.6.2
collected 1 item

test_example.py .                                                                                      [100%]

============================================== 1 passed in 0.02s ===============================================
```

Fixtures

`pytest` fixtures - это способ предоставления данных, тестовых дублей или установки состояния для ваших тестов. Фикстуры - это функции, которые могут возвращать широкий диапазон значений. Каждый тест, зависящий от фикстуры, должен явно принимать эту фикстуру в качестве аргумента.

```
# test_format_data.py

def test_format_data_for_display():
    people = [
        {
            "given_name": "Alfonsa",
            "family_name": "Ruiz",
            "title": "Senior Software Engineer",
        },
        {
            "given_name": "Sayid",
            "family_name": "Khan",
            "title": "Project Manager",
        },
    ]

    assert format_data_for_display(people) == [
        "Alfonsa Ruiz: Senior Software Engineer",
        "Sayid Khan: Project Manager",
    ]
```

```
# test_format_data.py

import pytest

@pytest.fixture
def example_people_data():
    return [
        {
            "given_name": "Alfonsa",
            "family_name": "Ruiz",
            "title": "Senior Software Engineer",
        },
        {
            "given_name": "Sayid",
            "family_name": "Khan",
            "title": "Project Manager",
        },
    ]

def test_format_data_for_display(example_people_data):
    assert format_data_for_display(example_people_data) == [
        "Alfonsa Ruiz: Senior Software Engineer",
        "Sayid Khan: Project Manager",
    ]

def test_format_data_for_excel(example_people_data):
    assert format_data_for_excel(example_people_data) == """given,family,title
Alfonsa,Ruiz,Senior Software Engineer
Sayid,Khan,Project Manager
"""
# ...
```

- Фикстуры обладают свойствами модульности, т.е. их можно импортировать и они могут зависеть и импортировать другие фикстуры. Это позволяет строить новые уровни абстракции
- Фикстуры можно использовать в файле `conftest.py`, сделав их доступными для всех тестов
- Фикстуры позволяют "мокать" отдельные элементы программы с использованием `monkeypatch`, например обращения к внешним ресурсам

```python
# conftest.py

import pytest
import requests

@pytest.fixture(autouse=True)
def disable_network_calls(monkeypatch):
    def stunted_get():
        raise RuntimeError("Network access not allowed during testing!")
    monkeypatch.setattr(requests, "get", lambda *args, **kwargs: stunted_get())
```

Маркеры

- Маркеры позволяют добавить категории вашим тестам. К примеру, можно пометить все тесты базы данных маркером `database_connection`
- Наличие маркеров позволяет запускать только необходимые категории тестов, а не все сразу
- `@pytest.mark.database_access`
- `pytest -m database_access` или `pytest -m "not database_access"`
- Имеются встроенные маркеры, такие как `skip`, `skipif`, `xfail`

Параметризация

- Параметризация позволяет написать семейство тестов с общей логикой, но разными параметрами

```python
def test_is_palindrome_empty_string():
    assert is_palindrome("")

def test_is_palindrome_single_character():
    assert is_palindrome("a")

def test_is_palindrome_mixed_casing():
    assert is_palindrome("Bob")

def test_is_palindrome_with_spaces():
    assert is_palindrome("Never odd or even")

def test_is_palindrome_with_punctuation():
    assert is_palindrome("Do geese see God?")

def test_is_palindrome_not_palindrome():
    assert not is_palindrome("abc")

def test_is_palindrome_not_quite():
    assert not is_palindrome("abab")
```

```
@pytest.mark.parametrize("palindrome", [
    "",
    "a",
    "Bob",
    "Never odd or even",
    "Do geese see God?",
])
def test_is_palindrome(palindrome):
    assert is_palindrome(palindrome)

@pytest.mark.parametrize("non_palindrome", [
    "abc",
    "abab",
])
def test_is_palindrome_not_palindrome(non_palindrome):
    assert not is_palindrome(non_palindrome)
```

```
@pytest.mark.parametrize("maybe_palindrome, expected_result", [
    ("", True),
    ("a", True),
    ("Bob", True),
    ("Never odd or even", True),
    ("Do geese see God?", True),
    ("abc", False),
    ("abab", False),
])
def test_is_palindrome(maybe_palindrome, expected_result):
    assert is_palindrome(maybe_palindrome) == expected_result
```

Ещё немного полезных фич

- `pytest --durations=n` позволяет замерить время исполнения и вывести n наиболее медленных тестов
- `pytest-randomly` - плагин, позволяющий исполнять тесты в случайном порядке, что может помочь обнаружить зависимые тесты
- `pytest-cov` - плагин, позволяющий оценить покрытие кода тестами
- `pytest-bdd` - плагин для Behavior Driven Development (BDD), добавляющий описания поведения на естественном языке

Спасибо за внимание! :)