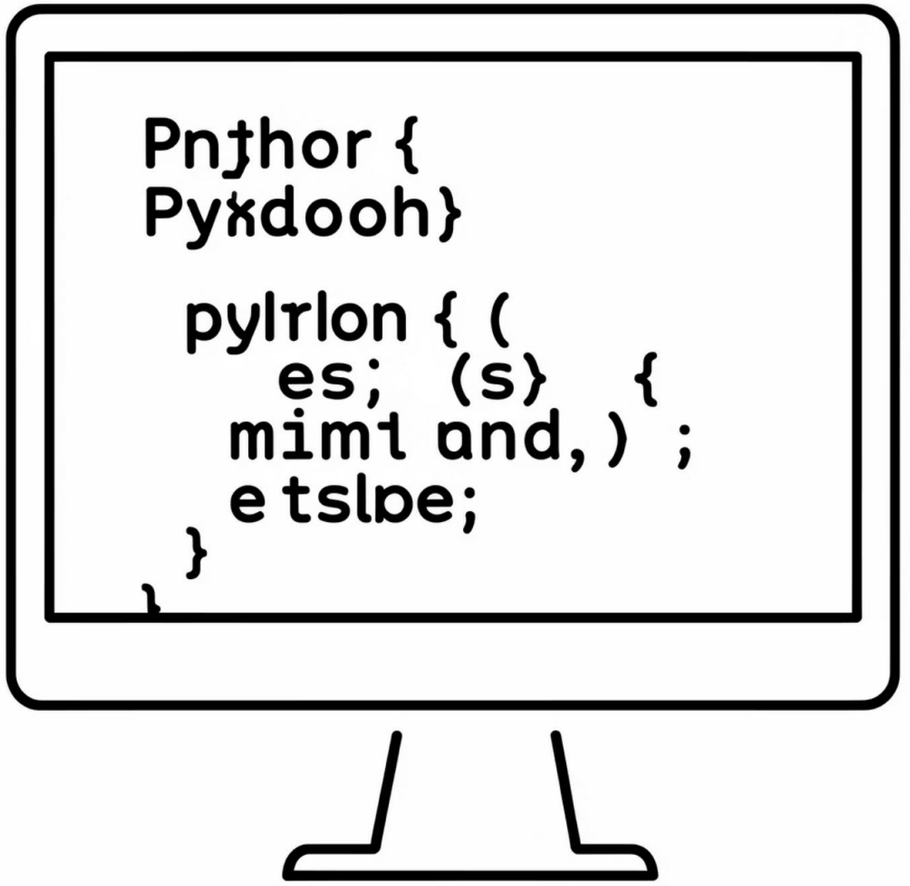
Pytest: Фреймворк для тестирования Python

Мощный инструмент для создания качественного, надежного кода

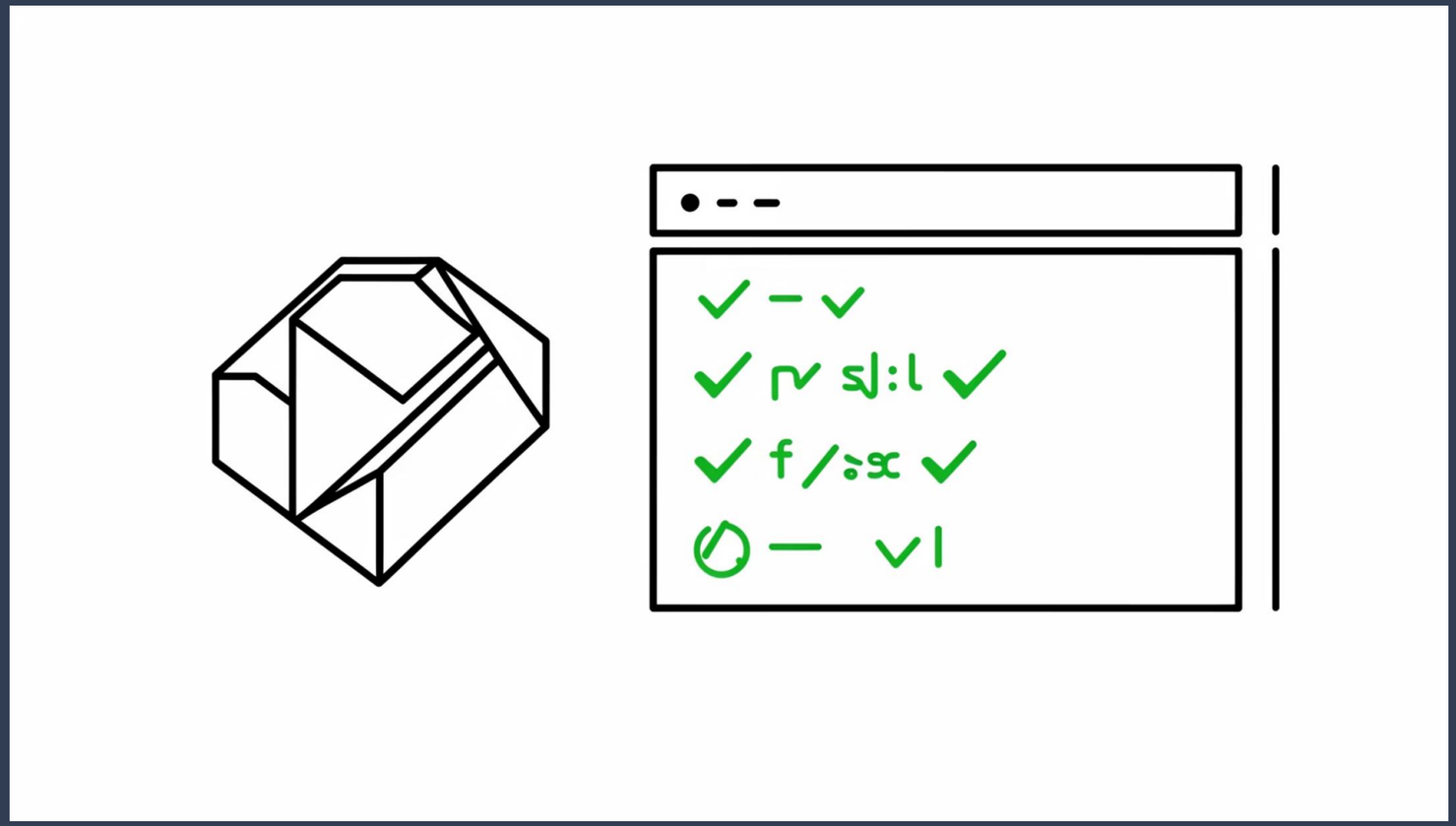


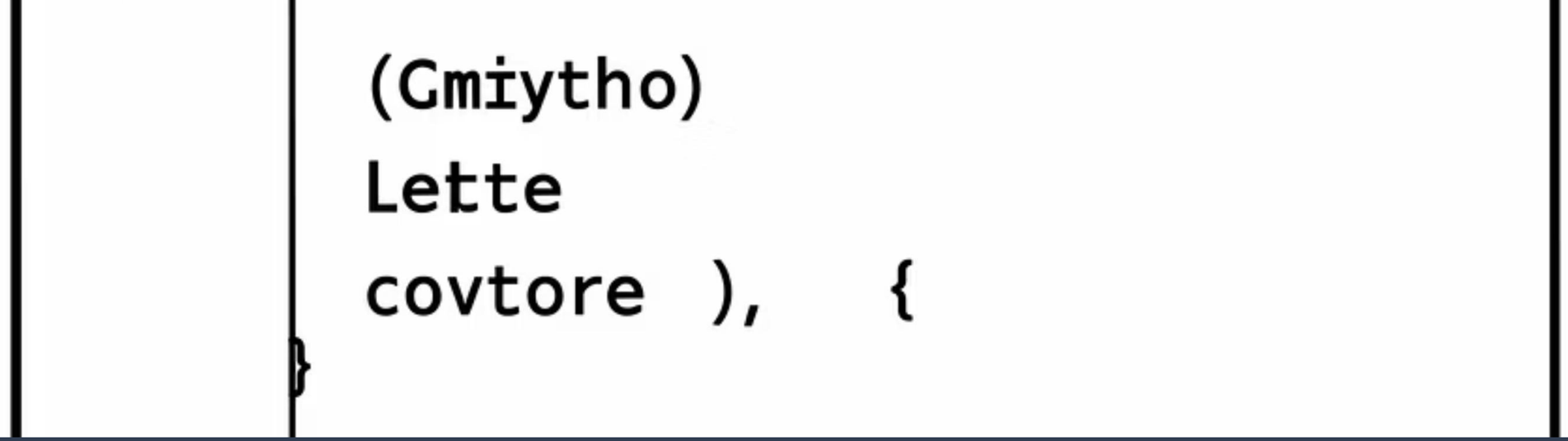
Что такое Pytest?

Pytest — это современный фреймворк для тестирования Python-кода. Он прост в использовании, но при этом обладает мощными возможностями для создания сложных тестовых сценариев.

Ключевые особенности:

- Простой и интуитивный синтаксис
- Автоматическое обнаружение тестов
- Гибкая система фикстур
- Минимальная конфигурация





Установка и начало работы

01

Установка

pip install pytest

02

Создание файла

test_sample.py с тестами

03

Запуск тестов

pytest в терминале

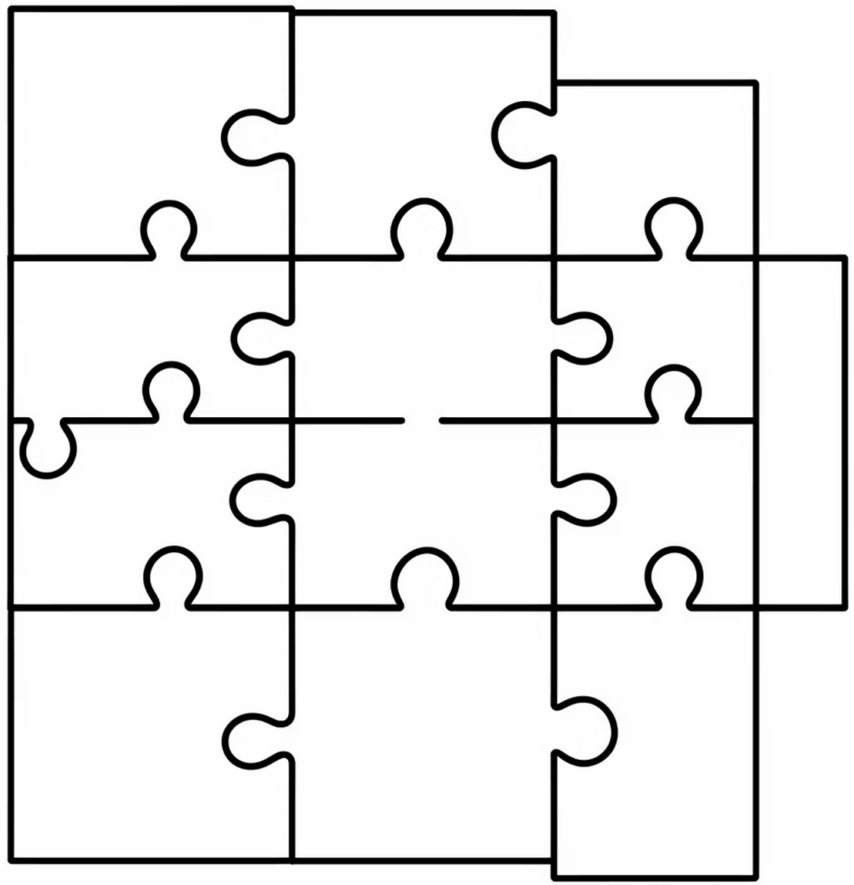
Простые тесты: Первые шаги

Создание базового теста не требует специального синтаксиса или классов. Просто напишите функцию с префиксом test_:

```
def test_addition():
    assert 2 + 2 == 4

def test_string():
    name = "Python"
    assert len(name) > 0
```

Pytest автоматически обнаружит и выполнит эти функции при запуске.



Мощь фикстур

Фикстуры — ключ к эффективному тестированию

Фикстуры позволяют создавать подготовленные данные и ресурсы для тестов. Они устраняют дублирование кода и делают тесты более читаемыми.

Настройка данных

Создание тестовых данных перед каждым тестом

Очистка ресурсов

Автоматическое освобождение памяти и закрытие соединений

Переиспользование

Одна фикстура может использоваться в множестве тестов

Фикстуры в действии

Пример простой фикстуры:

```
import pytest

@pytest.fixture
def sample_data():
    return [1, 2, 3, 4, 5]

def test_sum(sample_data):
    assert sum(sample_data) == 15
```

Результат: Тест успешно проходит

Правильная структура:

```
@pytest.fixture
def database_connection():
    # Установка соединения
    conn = connect()
    yield conn
    # Закрытие соединения
    conn.close()
```

Результат: Ресурс автоматически освобождается

Тестирование исключений

Проверка ошибок

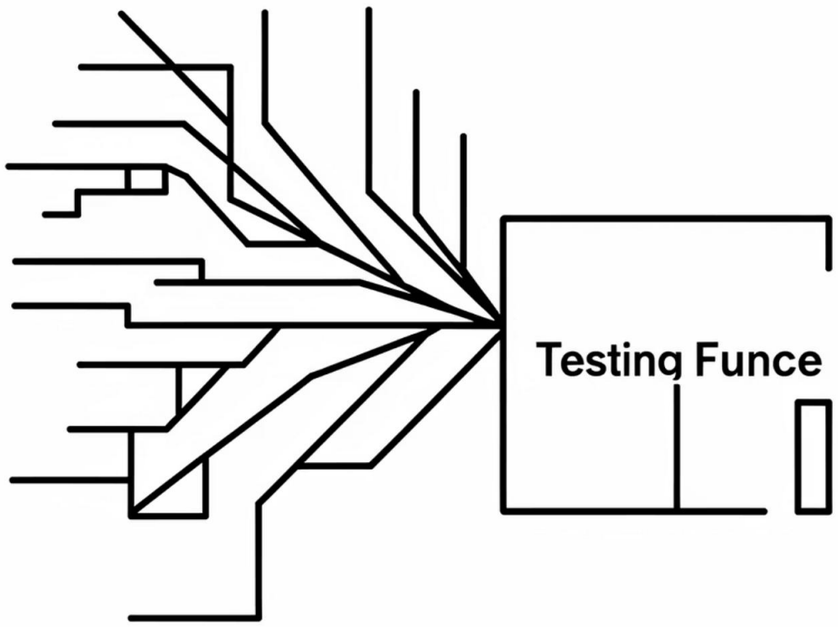
Убедитесь, что ваш код правильно обрабатывает исключительные ситуации

```
with pytest.raises(ValueError):
    int("не число")
```

Информативные сообщения

Проверяйте не только тип исключения, но и сообщение об ошибке

```
with pytest.raises(
 ValueError,
 match="Invalid input"
):
 process_data("")
```

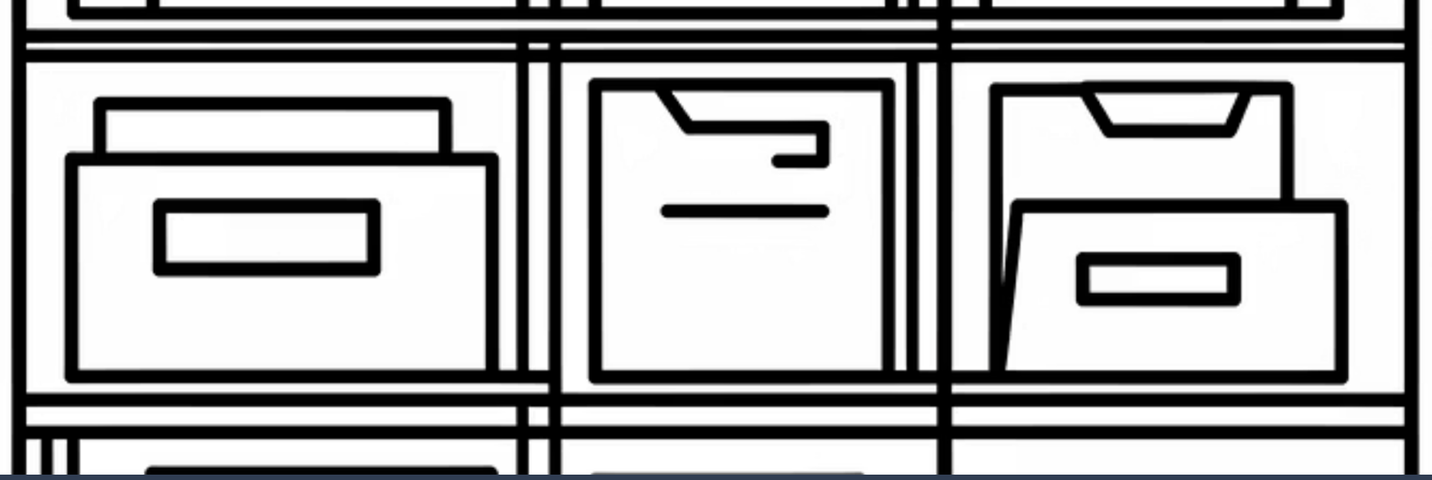


Тестирование с параметрами

Декоратор @pytest.mark.parametrize позволяет запускать один тест с разными входными данными:

```
@pytest.mark.parametrize(
    "input_val, expected",
    [
        (2, 4),
        (3, 9),
        (4, 16),
        (0, 0)
    ]
)
def test_square(input_val, expected):
    assert input_val ** 2 == expected
```

Совет: Экономит время и убирает дублирование тестов для разных входных значений.



Организация тестов

test_unit.py

Модульные тесты отдельных функций

test_integration.py

Проверка взаимодействия компонентов

test_end_to_end.py

Тесты полного рабочего процесса

Структурируйте тесты по принципу AAA: Arrange (подготовка), Act (действие), Assert (проверка).

Выводы и следующие шаги

Начните сегодня

Добавьте первый тест в свой проект

Пишите регулярно

Тестируйте каждое новое изменение

Интегрируйте CI/CD

Автоматический запуск тестов при каждом коммите

Pytest — ваш надежный помощник в создании качественного кода. Начните с простого и постепенно осваивайте более сложные возможности.