# Типичные примеры тестирования кода с помощью модуля Pytest

## Установка Pytest

pip install pytest

## Структура проекта

project/

├── main.py ← основной код

├── test\_main.py ← тесты

## main.py

def add(a, b):

return a + b

def divide(a, b):

if b == 0:

raise ValueError("Division by zero")

return a / b

class User:

def \_\_init\_\_(self, username, active=True):

self.username = username

self.active = active

def deactivate(self):

self.active = False

### Пример 1: базовые тесты функций

# test\_main.py

from main import add, divide

def test\_add():

assert add(2, 3) == 5

assert add(-1, 1) == 0

def test\_divide():

assert divide(10, 2) == 5

### Пример 2: проверка исключений

import pytest

def test\_divide\_by\_zero():

with pytest.raises(ValueError, match="Division by zero"):

divide(5, 0)

### Пример 3: тестирование классов

from main import User

def test\_user\_deactivation():

user = User("alice")

assert user.active is True

user.deactivate()

assert user.active is False

### Пример 4: параметризация тестов

import pytest

from main import add

@pytest.mark.parametrize("a, b, expected", [

(1, 2, 3),

(0, 0, 0),

(-1, -1, -2),

])

def test\_add\_multiple(a, b, expected):

assert add(a, b) == expected

### Пример 5: использование фикстур

python

КопироватьРедактировать

import pytest

from main import User

@pytest.fixture

def active\_user():

return User("testuser")

def test\_user\_starts\_active(active\_user):

assert active\_user.active is True

## Запуск тестов

bash

КопироватьРедактировать

pytest test\_main.py

По умолчанию Pytest ищет файлы test\_\*.py и функции test\_\*

## Вывод результата

================ test session starts =================

collected 5 items

test\_main.py ..... [100%]

================== 5 passed in 0.12s ==================

| **Что тестировать** | **Как это делается** |
| --- | --- |
| Исключения | with pytest.raises(...) |
| Несколько кейсов | @pytest.mark.parametrize |
| Состояние до/после | @pytest.fixture |
| Мокинг внешних зависимостей | через unittest.mock или pytest-mock |
| Производительность / время | pytest-benchmark, pytest-timeout |
| Валидация API / JSON / схем | JSONSchema + assert |

# Посчитать покрытие кода

Покрытие кода — это метрика, которая показывает, какая часть вашего кода **фактически выполняется во время тестов**. Полезно, чтобы понимать, что вы действительно протестировали, а что — нет.

## 1: Установка

pip install pytest coverage

## 2: Пример проекта

project/

├── main.py

├── test\_main.py

**📄 main.py**

def add(a, b):

return a + b

def subtract(a, b):

return a - b

def multiply(a, b):

return a \* b

def divide(a, b):

if b == 0:

raise ValueError("Division by zero")

return a / b

**📄 test\_main.py**

from main import add, divide

def test\_add():

assert add(2, 3) == 5

def test\_divide():

assert divide(10, 2) == 5

## 3: Подсчёт покрытия

coverage run -m pytest

Это запустит pytest и будет отслеживать, какие строки кода были выполнены.

## 4: Посмотреть отчёт

coverage report -m

Пример вывода:

Name Stmts Miss Cover Missing

----------------------------------------

main.py 12 5 58% 4-5, 8-9

test\_main.py 5 0 100%

* Stmts: количество строк кода
* Miss: сколько строк не покрыто
* Cover: % покрытия
* Missing: номера строк, которые не были вызваны

## 5: HTML-отчёт (наглядно)

coverage html

Откройте файл:

htmlcov/index.html

Там будет показано:

* какие строки покрыты (зелёные),
* какие — пропущены (красные),
* и общий % покрытия.

## Часто используемые команды:

coverage erase # очистить старые данные

coverage run -m pytest # запустить тесты с отслеживанием

coverage report -m # текстовый отчёт

coverage html # сгенерировать HTML-отчёт