Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

О Т Ч Е Т
О практической работе по дисциплине «Тестирование программного обеспечения для разработчиков»

Тема: Лабораторная работа 1. Unit tests.

Обучающийся: Фурсаев Роман Владимирович, К3321

Санкт-Петербург,
2025

Цель работы

Выберите открытый проект на GitHub (можно свой пет-проект).

Необходимо определить, какие части проекта критически важны и должны быть покрыты тестами:
Основные функции и методы.

Критические участки (например, расчёты, валидации, бизнес-логика).

Ключевые сценарии использования (use cases).

Написание тестов

Создайте не менее 5 юнит-тестов, охватывающих разные части функционала.

Обязательно включите тесты на корректные и граничные случаи.

Используйте подход AAA (Arrange–Act–Assert).

Соблюдайте принципы FIRST (Fast, Isolated, Repeatable, Self-validating, Timely).

Ход работы

1. **Создание пэт-проекта для дальнейших тестов.**

Так как у меня нет “Нормальных” проектов, где я могу проверить логику или функции (я фронт разработчик).
Было принято решение реализовать простейший конвертер валюты.

simple-currency-converter > probapp.py > ...

```python
#Курс актуален на 06.10.2025 21:40

EXCHANGE_RATES = {
    "USD": 1.0,
    "EUR": 0.85,
    "RUB": 83.0,
    "CNY": 7.12
}

def convert_currency(amount: float, from_currency: str, to_currency: str) -> float:
    if from_currency not in EXCHANGE_RATES:
        raise ValueError(f"Неизвестная валюта: {from_currency}")
    if to_currency not in EXCHANGE_RATES:
        raise ValueError(f"Неизвестная валюта: {to_currency}")

    amount_in_usd = amount / EXCHANGE_RATES[from_currency]
    converted_amount = amount_in_usd * EXCHANGE_RATES[to_currency]
    return round(converted_amount, 2)

def main():
    print("Доступные валюты:", ", ".join(EXCHANGE_RATES.keys()))

    try:
        amount = float(input("Введите сумму для конвертации: "))
        from_currency = input("Из какой валюты (код): ").upper()
        to_currency = input("В какую валюту (код): ").upper()

        result = convert_currency(amount, from_currency, to_currency)
        print(f"{amount} {from_currency} = {result} {to_currency}")

    except ValueError as e:
        print("Ошибка:", e)

if __name__ == "__main__":
    main()
```

Рис 1. Код пэт-проекта

```
Доступные валюты: USD, EUR, RUB, CNY
Введите сумму для конвертации: 123
Из какой валюты (код): EUR
В какую валюту (код): RUB
123.0 EUR = 12010.59 RUB
```

Рис 2. Пример выполнения проекта

Код максимально простой и рабочий. После этого я приступил к реализации тестов.

2. Реализация unit-tests

Я создал 5 разных тестов, каждый из которых охватывает свою часть функционала. Реализация через pytest

```python
import pytest
from probapp import convert_currency, EXCHANGE_RATES, main


def test_usd_to_eur():
    assert convert_currency(100, "USD", "EUR") == 85.0

def test_rub_to_usd():
    assert convert_currency(8300, "RUB", "USD") == 100.0

def test_same_currency():
    assert convert_currency(500, "CNY", "CNY") == 500

def test_invalid_from_currency():
    with pytest.raises(ValueError):
        convert_currency(100, "ABC", "USD")

def test_invalid_to_currency():
    with pytest.raises(ValueError):
        convert_currency(100, "USD", "XYZ")
```

Рис 3. Реализация тестов

У меня 5 разных тестов, сейчас расскажу про каждый.

```python
def test_usd_to_eur():
    assert convert_currency(100, "USD", "EUR") == 85.0

def test_rub_to_usd():
    assert convert_currency(8300, "RUB", "USD") == 100.0

def test_same_currency():
    assert convert_currency(500, "CNY", "CNY") == 500

def test_invalid_from_currency():
    with pytest.raises(ValueError):
        convert_currency(100, "ABC", "USD")

def test_invalid_to_currency():
    with pytest.raises(ValueError):
        convert_currency(100, "USD", "XYZ")
```

Рис 4. Тесты

Тест 1. test_usd_to_eur

Тут я проверяю, как конвертируееется 100 долларов в евро. Банально сравниваю результат функции convert_currency с ожидаемым результатом.

Цель: проверить правильность расчёта для реальной конверсии между разными валютами. Важна для основных сценариев использования: пользователь вводит сумму и валюты, ожидает точный результат.

Тест 2. test_rub_to_usd

Точно такой же тест, но в обратную сторону

Цель: Проверить универсальность формулы, что она работает не только в направлении USD → другая валюта.

Тест 3. test_same_currency

Проверяю попытку конвертации валюты самой в себя.

Цель: проверить граничный случай, где результат должен быть равен исходной сумме.

Тест 4. test_invalid_from_currency

Я проверяю тут попытку ковертации из неизвестной валюты.

Цель: убедиться, что функция корректно обрабатывает некорректный ввод исходной валюты. Важно для предотвращения некорректных расчётов и аварийного поведения.

Тест 5. test_invalid_to_currency

Проверяю попытку конвертации в неизвестную валюту.

Цель: убедиться, что функция корректно обрабатывает некорректный ввод целевой валюты.

3. Результат теста

Запуск тестов через pytest в виртуальном окружении:

```
PS D:\git\simple-currency-converter> python -m venv venv
PS D:\git\simple-currency-converter> .\venv\Scripts\activate.ps1
(venv) PS D:\git\simple-currency-converter> pip install pytest
```

Рис 5. Установка pytest

```
(venv) PS D:\git\simple-currency-converter> python -m pytest test_propapp.py -v
============================================= test session starts =============================================
platform win32 -- Python 3.12.6, pytest-8.4.2, pluggy-1.6.0 -- D:\git\simple-currency-converter\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: D:\git\simple-currency-converter
collected 5 items

test_propapp.py::TestCurrencyConverter::test_invalid_from_currency PASSED                            [ 20%]
test_propapp.py::TestCurrencyConverter::test_invalid_to_currency PASSED                              [ 40%]
test_propapp.py::TestCurrencyConverter::test_rub_to_usd PASSED                                       [ 60%]
test_propapp.py::TestCurrencyConverter::test_same_currency PASSED                                    [ 80%]
test_propapp.py::TestCurrencyConverter::test_usd_to_eur PASSED                                       [100%]

============================================== 5 passed in 0.02s ==============================================
(venv) PS D:\git\simple-currency-converter>
```

Рис 6. Результат работы тестов

Как можем заметить, все тесты мой проект успешно прошёл.

Теперь проверю покрытие кода моими тестами

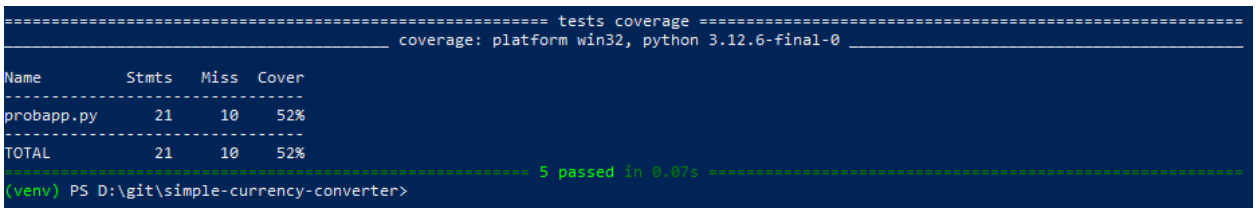

Рис 7. Покрытие с помощью pytest-cov.

У меня довольно низкий процент покрытия, ведь я тестировал только основные функции, игнорируя main(), где пользователь вводит данные.

Так что я решил добавить ещё тесты, чтобы поднять свой процент покрытия.

Я добавил monkeypatch, что иммитировать ввод пользователя, чтобы тесты были автоматическими и воспроизводимыми.

```python
def test_main_valid(monkeypatch, capsys):
    inputs = iter(['100', 'USD', 'EUR'])
    monkeypatch.setattr('builtins.input', lambda _: next(inputs))
    main()
    captured = capsys.readouterr()
    assert '100.0 USD = 85.0 EUR' in captured.out

def test_main_invalid_from(monkeypatch, capsys):
    inputs = iter(['100', 'ABC', 'EUR'])
    monkeypatch.setattr('builtins.input', lambda _: next(inputs))
    main()
    captured = capsys.readouterr()
    assert 'Неизвестная валюта' in captured.out

def test_main_invalid_to(monkeypatch, capsys):
    inputs = iter(['100', 'USD', 'XYZ'])
    monkeypatch.setattr('builtins.input', lambda _: next(inputs))
    main()
    captured = capsys.readouterr()
    assert 'Неизвестная валюта' in captured.out
```

Рис 8. Дополнительные тесты

Я создал итеератор с заранее заданными значениями, которые "введёт пользователь".

Тест 2.1. test_main_valid
Проверяю, есть ли ожидаемая строка в выводе.

Цель: проверить, что функция правильно конвертирует и выводит результат.

Тест 2.2. test_main_invalid_from

Проверяю ввод некорректной исходной валюты

Цель: проверить, что функция ловит ошибку и выводит сообщение «Неизвестная валюта».

Тест 2.3. test_main_invalid_to
Проверяю ввод некорректной целевой валюты

Цель: проверить, что функция ловит ошибку и выводит сообщение «Неизвестная валюта».

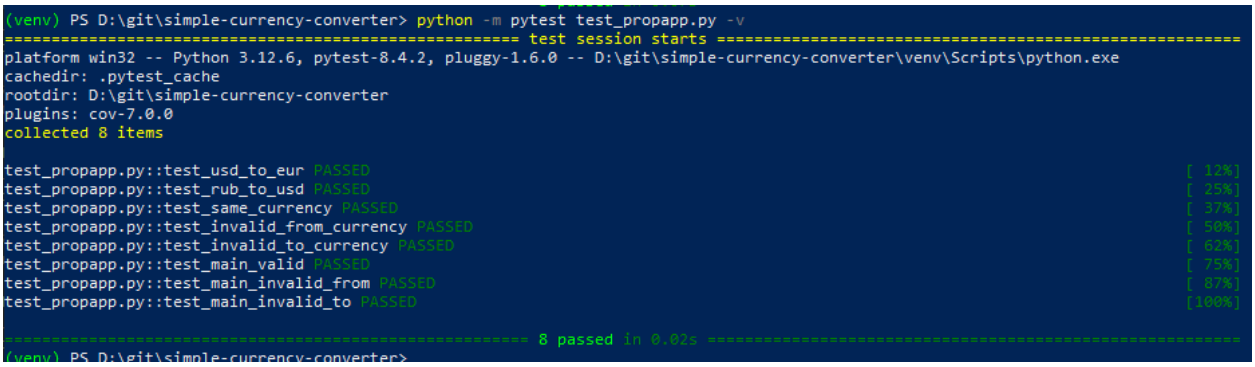

Рис 9. Результат работы всех тестов

Теперь проверю покрытие с помощью pytest-cov

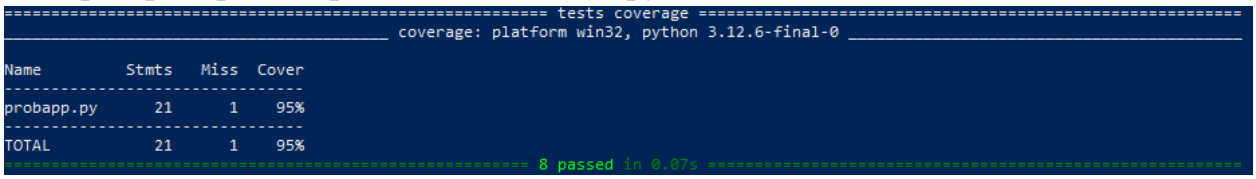

Рис 10. Второй тест покрытия

Как можем заметить, ситуация значительно улучшилась и теперь у меня покрытиее тестами 95%, а не 52%.

Выводы о качестве тестирования и обнаруженных проблемах.

1. Полнота покрытия кода

- Все критические функции (convert_currency и main) протестированы.
- Использование monkeypatch и capsys позволило покрыть строки с input(), print() и блоки try/except.

- В результате coverage достиг 95%, что свидетельствует о почти полном тестировании всех ветвей программы.

2. Проверка корректных сценариев
- Тесты проверяют стандартные конверсии между валютами (USD → EUR, RUB → USD, одинаковые валюты).
- Это гарантирует, что основной функционал работает правильно.

3. Проверка граничных и ошибочных случаев
- Тесты на неизвестные валюты (from и to) подтверждают, что программа корректно выбрасывает исключения и информирует пользователя.
- Таким образом обработка ошибок полностью контролируется, программа не падает при некорректном вводе.

4. Автоматизация и повторяемость
- Тесты не требуют ручного ввода, их можно запускать многократно в CI/CD.
- Все тесты быстрые и изолированные — соответствуют принципам FIRST.

5. Проверка пользовательского интерфейса
- Тестирование main() через monkeypatch и capsys гарантирует, что пользователь видит корректный вывод.

Обнаруженные проблемы до тестирования

Низкий coverage до добавления тестов main()
- Без тестов на main() строки с input(), print() и блоки обработки ошибок не выполнялись.
- Coverage был ~52%, хотя основная логика конверсии работала.

Итог

1. Тесты полностью покрывают функциональность конвертера, включая:

- Основной расчёт валют
- Обработку ошибок
- Поток ввода/вывода пользователя

2. Качество тестирования высокое:
- Все ветви кода проверены
- Нет «слепых зон» в логике
- Программа защищена от некорректного ввода

3. Обнаруженные проблемы устранены после добавления тестов на main():
- Coverage увеличен до 95%
- Все возможные пользовательские ошибки обрабатываются корректно