|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение высшего образования* ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных технологий (МОСИТ)

|  |
| --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2** |
| **по дисциплине** |
| **«Тестирование и верификация программного обеспечения»** |

|  |  |
| --- | --- |
| Выполнил студент группы ИКБО-41-23 | Трофимов А.А. |
| Принял | Чернов Е.А. |

|  |  |  |
| --- | --- | --- |
| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_2025г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2025 г. | *(подпись руководителя)* |

Москва 2025

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc210687555)

[1 ПОСТАНОВКА ЗАДАЧИ 4](#_Toc210687556)

[2 ХОД РАБОТЫ 5](#_Toc210687557)

[2.1 Модульное тестирование 5](#_Toc210687558)

[2.2 Мутационное тестирование 11](#_Toc210687559)

[3 ЗАКЛЮЧЕНИЕ 13](#_Toc210687560)

# ВВЕДЕНИЕ

Модульное тестирование является одним из ключевых методов обеспечения качества программного обеспечения. Оно проверяет отдельные функции, методы или классы в изоляции от остальных компонентов, что позволяет выявлять ошибки на ранних стадиях разработки и повышать надёжность кода. В языке Python для написания и запуска модульных тестов широко используется библиотека pytest, обеспечивающая удобный синтаксис и автоматический поиск тестовых файлов.

Однако модульных тестов недостаточно для полной оценки качества покрытия. Для этого применяется мутационное тестирование: в код преднамеренно вносятся небольшие изменения (мутации), после чего тесты запускаются повторно. Если тесты обнаружили изменение, мутант считается «убитым», иначе — «выжившим». Этот метод позволяет выявить слабые места в тестах и повысить надёжность программного продукта.

# 1 ПОСТАНОВКА ЗАДАЧИ

Цель работы: ознакомиться с процессом модульного и мутационного тестирования, включая разработку, проведение тестов, исправление ошибок, анализ тестового покрытия, а также оценку эффективности тестов путём применения методов мутационного тестирования.

Для достижения поставленной цели работы студентам необходимо выполнить ряд задач:

− изучить основы модульного тестирования и его основные принципы;

− освоить использование инструментов для модульного тестирования (pytest для Python, JUnit для Java и др.);

− разработать модульные тесты для программного продукта и проанализировать их покрытие кода;

− изучить основы мутационного тестирования и освоить инструменты для его выполнения (MutPy, PIT, Stryker);

− применить мутационное тестирование к программному продукту, оценить эффективность тестов;

− улучшить существующий набор тестов, ориентируясь на результаты мутационного тестирования;

− оформить итоговый отчёт с результатами проделанной работы.

# 2 ХОД РАБОТЫ

## 2.1 Модульное тестирование

В ходе работы был разработан программный модуль, который содержит пять функций для работы со списками, числами и строками. Все функции выполняют базовые вычисления и содержат проверки типов входных данных, чтобы предотвратить некорректное использование.

Ошибка была допущена специально в remove\_spaces ожидается что будут убраны пробелы, в реальности убираются все гласные

Листинг 1 – Код моего программного модуля converter.py

|  |
| --- |
| def is\_palindrome(s: str) -> bool:  """Проверка, является ли строка палиндромом"""  s = ''.join(ch.lower() for ch in s if ch.isalnum())  return s == s[::-1]  def count\_vowels(s: str) -> int:  """Подсчёт количества гласных букв"""  vowels = "aeiouаеёиоуыэюя"  return sum(1 for ch in s.lower() if ch in vowels)  def reverse\_string(s: str) -> str:  """Переворот строки"""  return s[::-1]  def to\_upper(s: str) -> str:  """Преобразование строки в верхний регистр"""  return s.upper()  def remove\_spaces(s: str) -> str:  """Функция с преднамеренной ошибкой:  должно удалять только пробелы, но удаляет все гласные"""  return ''.join(ch for ch in s if ch.lower() not in "aeiouаеёиоуыэюя") |

Остальные функции работают корректно и выполняют следующие задачи: script2 — гипотенуза; script3 — обработка строки; script4 — выбор для подсчёта двух разных переменных; script5 — генерация пароля.

Далее от другого участника команды был получен программные модули, в которых имеется 5 функций, для проведения их тестирования и выявления ошибки.

Листинг 2 – Код программного модуля badscript1

|  |
| --- |
| def main(key, x, y):  if key == 'key':  return x + y  else:  x = 10  return x \* y |

Листинг 3 – Код программного модуля script1

|  |
| --- |
| def main(key, x, y):  if key == 'key':  return x + y  else:  x = 10  return x \* y |

Листинг 4 – Код программного модуля script2

|  |
| --- |
| import math  def main(a, b):  if 0 in (a, b):  return 0  return math.sqrt((a\*\*2) \* (b\*\*2)) |

Листинг 5 – Код программного модуля script3

|  |
| --- |
| import re  def main(input\_string):  pattern = r'global\s+(\w+)\s+is\s\*#(-?\d+)'  matches = re.findall(pattern, input\_string)  result = {}  for key, value in matches:  result[key] = int(value)  return result |

Листинг 6 – Код программного модуля script4

|  |
| --- |
| def main(x, y, choice):  if choice == 0:  return x + y  if choice == 1:  return x - y  if choice == 2:  return x \* y  return 0 |

Листинг 7 – Код программного модуля script5

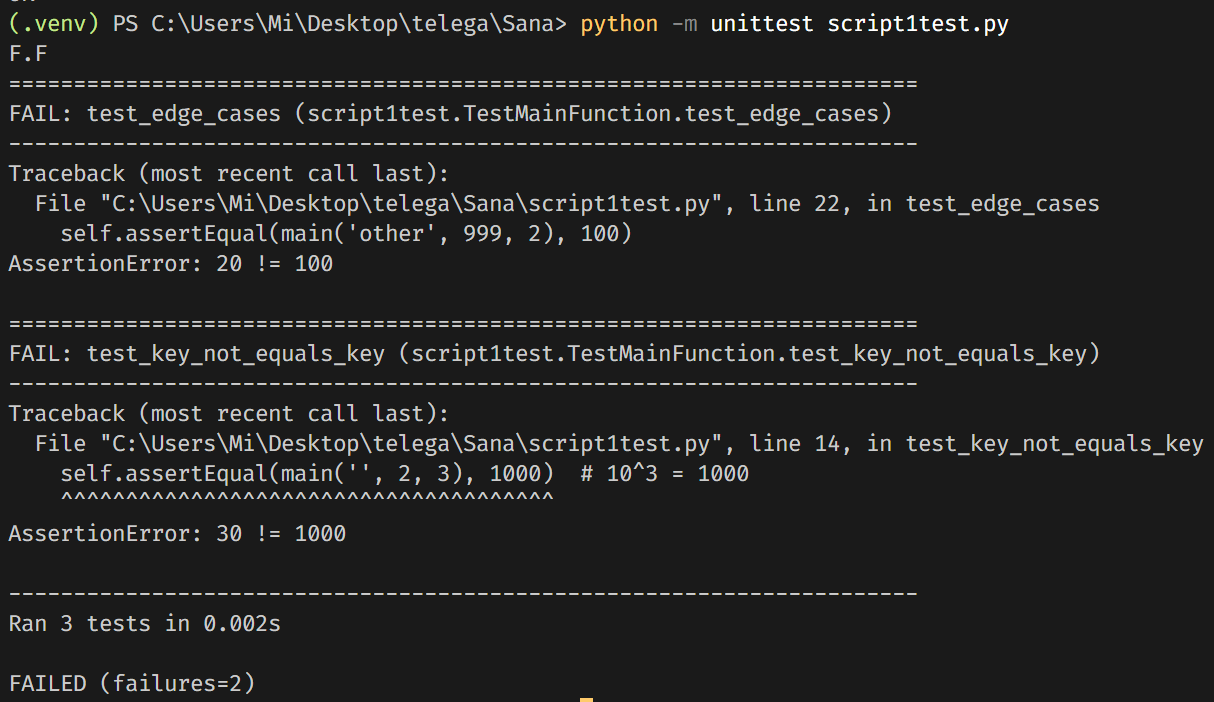
|  |
| --- |
| import secrets  import string  def main(length):  alphabet = string.ascii\_letters  if length < 0:  raise ValueError  return ''.join(secrets.choice(alphabet) for i in range(length)) |

Далее были написаны тесты для программного модуля другого участника команды с целью выявления ошибки в коде. Тесты представлены на листинге 8.

Листинг 8 - Тесты для чужого модуля

|  |
| --- |
| import unittest  from script4 import main  class TestMainFunction(unittest.TestCase):  def test\_addition\_choice\_0(self):  self.assertEqual(main(5, 3, 0), 8)  self.assertEqual(main(10, 20, 0), 30)  self.assertEqual(main(-5, -3, 0), -8)  self.assertEqual(main(-10, 5, 0), -5)  self.assertEqual(main(0, 0, 0), 0)  self.assertEqual(main(5, 0, 0), 5)  self.assertEqual(main(0, 5, 0), 5)  self.assertAlmostEqual(main(2.5, 3.7, 0), 6.2)  self.assertAlmostEqual(main(-1.5, 2.5, 0), 1.0)  def test\_subtraction\_choice\_1(self):  self.assertEqual(main(10, 3, 1), 7)  self.assertEqual(main(5, 5, 1), 0)  self.assertEqual(main(-5, -3, 1), -2)  self.assertEqual(main(-10, 5, 1), -15)  self.assertEqual(main(0, 0, 1), 0)  self.assertEqual(main(5, 0, 1), 5)  self.assertEqual(main(0, 5, 1), -5)  self.assertAlmostEqual(main(5.5, 2.2, 1), 3.3)  self.assertAlmostEqual(main(1.5, 2.5, 1), -1.0)  def test\_multiplication\_choice\_2(self):  self.assertEqual(main(5, 3, 2), 15)  self.assertEqual(main(10, 10, 2), 100)  self.assertEqual(main(-5, 3, 2), -15)  self.assertEqual(main(-5, -3, 2), 15)  self.assertEqual(main(0, 5, 2), 0)  self.assertEqual(main(5, 0, 2), 0)  self.assertEqual(main(0, 0, 2), 0)  self.assertAlmostEqual(main(2.5, 4, 2), 10.0)  self.assertAlmostEqual(main(1.5, 2.5, 2), 3.75)  def test\_default\_case(self):  self.assertEqual(main(5, 3, 3), 0)  self.assertEqual(main(10, 20, -1), 0)  self.assertEqual(main(1, 1, 100), 0)  self.assertEqual(main(1, 1, None), 0)  self.assertEqual(main(100, 200, 5), 0)  self.assertEqual(main(-5, -10, 2.5), 0) |

В результате тестирования была выявлена ошибка в функции celsius\_to\_fahrenheit из-за неправильного порядка переменных в формуле. Ошибка представлена на рисунке 1.



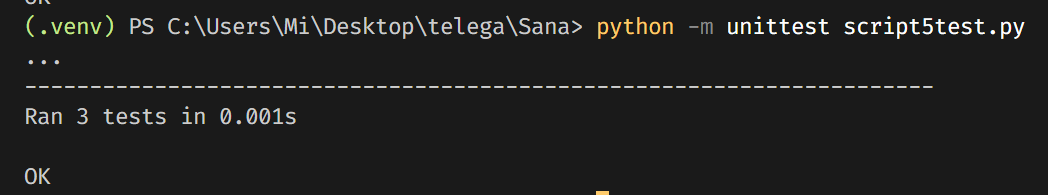
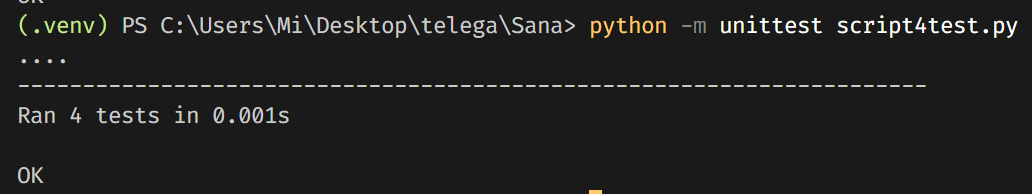
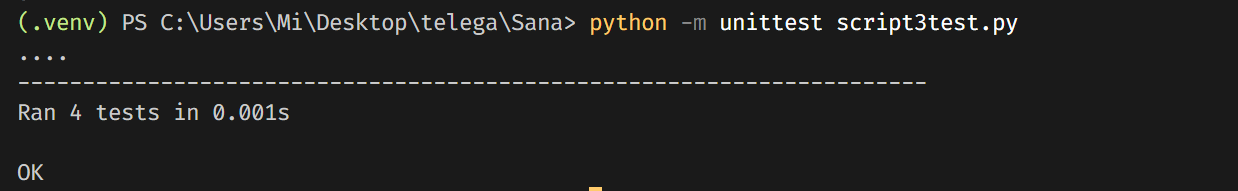
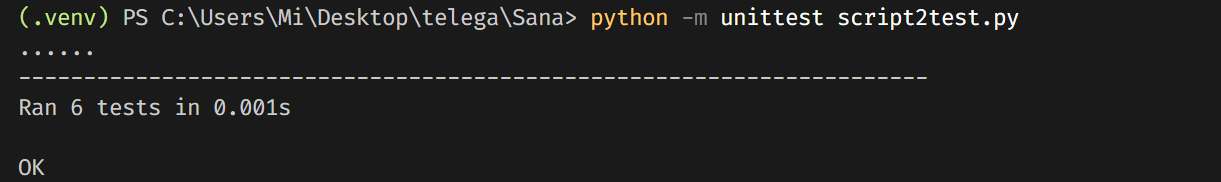
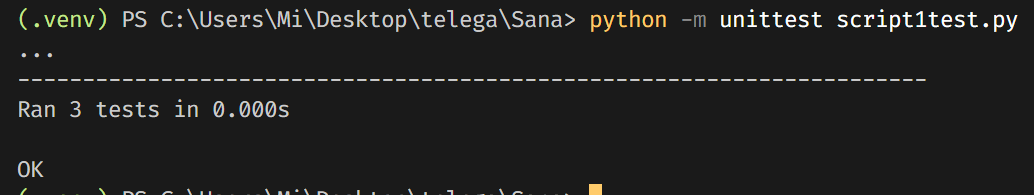
**Рисунок 1 – Найденная ошибка**

Рисунок 2 – остальные тесты

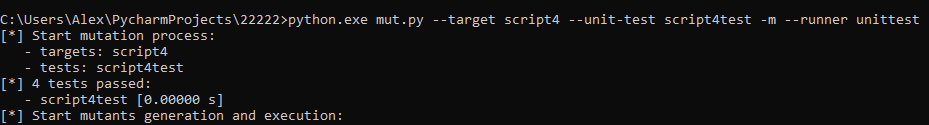
После данного этапа программа была возвращена на доработку, затем снова были проведены тесты, которые в этот раз прошли успешно. Результат приведен на рисунке 2.



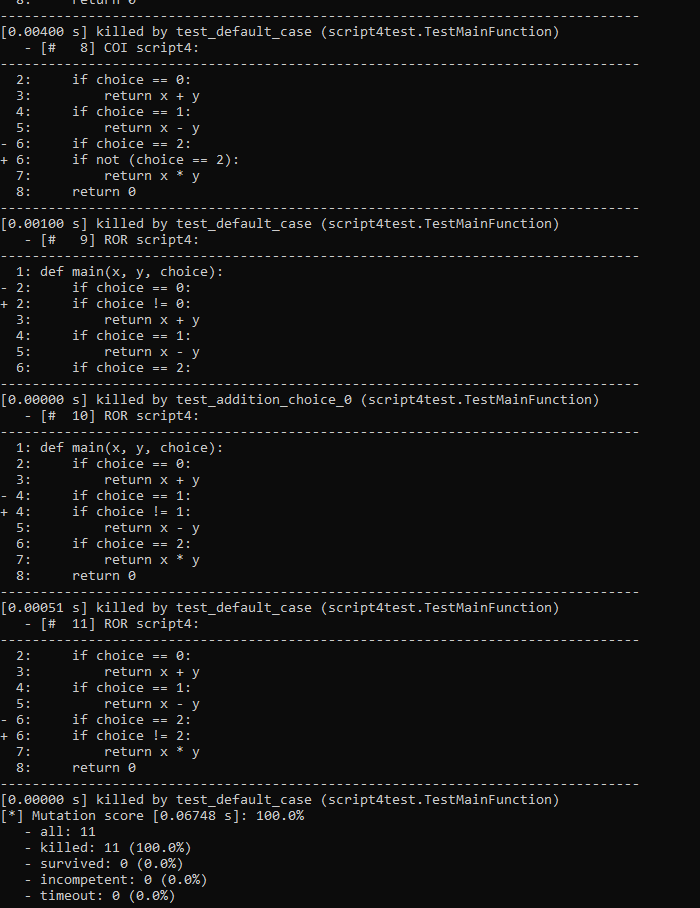
**Рисунок 2 – Успешно пройденные тесты**

## 2.2 Мутационное тестирование

На втором этапе практической работы было применено мутационное тестирование для оценки качества тестов, разработанных на предыдущем этапе. Данный этап помогает проверить, насколько эффективно тесты обнаруживают ошибки в программном модуле.



**Рисунок 3 – Запуск мутационных тестов**



**Рисунок 4 – Успешный мутационный тест**

# 3 ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы была проведена проверка качества тестирования разработанного программного модуля с применением модульного и мутационного подходов.

Модульные тесты подтвердили корректность работы функций и выявили преднамеренно внесённую ошибку. Мутационное тестирование показало, что исходные тесты не обеспечивали полного покрытия, так как часть мутантов «выжила». Это позволило определить слабые места тестов и внести необходимые исправления.

После доработки тестов количество выживших мутантов значительно сократилось, что говорит о повышении строгости проверок и эффективности тестового покрытия.

Результаты работы подтвердили, что сочетание модульного и мутационного тестирования является надёжным инструментом оценки качества программного продукта, повышающим его устойчивость и надёжность.