



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»

**Тема: «Модульное и мутационное тестирование программного
продукта»**

Выполнили студенты группы ИКБО-43-23

Принял

Воронин А.Т.
Виноградова Д.С.
Нурписов Н.И.
Мурехин Я.А.
Ильичев Г.П.

Практическая работа выполнена

«__»_____2025г.

(подпись студента)

«Зачтено»

«__»_____2025 г.

(подпись руководителя)

Москва 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
ЦЕЛЬ И ПОСТАНОВКА ЗАДАЧ ПРАКТИЧЕСКОЙ РАБОТЫ.....	5
1 ПРАКТИЧЕСКАЯ ЧАСТЬ РАБОТЫ	6
1.1 Разработка функционального модуля «Калькулятор» (Виноградова Д. С.).....	6
1.1.1 Назначение и функциональные возможности модуля «Калькулятор»	6
1.1.2 Структура и исходный код модуля «Калькулятор»	7
1.1.3 Документация модуля «Калькулятор»	9
1.1.4 Документация модульного тестирования модуля «Программа для работы с матрицами» (модуль Воронина А.Т.)	12
1.1.5 Мутационное тестирование модульного тестирования.....	19
1.2 Разработка функционального модуля «Программа для работы с матрицами» (Воронин А.Т.).....	22
1.2.1 Назначение и функциональные возможности модуля «Программа для работы с матрицами».....	22
1.2.2 Структура и исходный код модуля «Программа для работы с матрицами».....	23
1.2.3 Документация модуля «Программа для работы с матрицами»	24
1.2.4 Документация модульного тестирования модуля «Калькулятор» (модуль Виноградовой Д.С.).....	27
1.2.5 Мутационное тестирование модульного тестирования.....	32
1.3 Разработка функционального модуля «Генератор паролей» (Мурехин Я.А.).....	38
1.3.1 Назначение и функциональные возможности модуля «Генератор паролей».....	38
1.3.2 Структура и исходный код модуля «Генератор паролей».....	39

1.3.3 Документация модуля «Генератор паролей»	41
1.3.4 Документация модульного тестирования модуля «Калькулятор ВМІ» (модуль Нурпиисова Н.И.)	43
1.3.5 Мутационное тестирование модульного тестирования	51
1.4 Разработка функционального модуля «Калькулятор ВМІ» (Нурпиисов Н.И.)	56
1.4.1 Назначение и функциональные возможности модуля «Калькулятор ВМІ»	56
1.4.2 Структура и исходный код модуля «Калькулятор ВМІ»	56
1.4.3 Документация модуля «Калькулятор ВМІ»	58
1.4.4 Модульное тестирование модуля «Генератор паролей» (модуль Мурехина Я.А.)	60
1.4.5 Мутационное тестирование модульного тестирования	63
2 АНАЛИЗ ПРОВЕДЕННОЙ ПРАКТИЧЕСКОЙ РАБОТЫ	67
2.1 Оценка качества тестирования	67
2.2 Анализ недостатков и их устранение	67
2.3 Итоговые выводы	69
ЗАКЛЮЧЕНИЕ	70

ВВЕДЕНИЕ

В рамках данной практической работы рассматриваются два ключевых направления контроля качества — модульное и мутационное тестирование. Модульное тестирование обеспечивает проверку корректности работы отдельных компонент программной системы, а мутационное — оценку эффективности разработанных тестов за счёт внесения искусственных изменений (мутаций) в исходный код.

Практическая работа направлена на формирование практических навыков применения данных методов, а также на развитие умений анализа тестового покрытия, интерпретации результатов тестирования и совершенствования программного кода на основе выявленных ошибок.

ЦЕЛЬ И ПОСТАНОВКА ЗАДАЧ ПРАКТИЧЕСКОЙ РАБОТЫ

Цель работы — освоить процесс модульного и мутационного тестирования программного продукта, включая разработку тестов, исправление ошибок, анализ покрытия кода и оценку эффективности тестирования.

Для достижения поставленной цели были выполнены следующие задачи:

1. Изучить теоретические основы и принципы модульного тестирования.
2. Освоить использование инструментов для модульного тестирования (PyTest для Python, JUnit для Java и др.).
3. Разработать модульные тесты для программного продукта и провести анализ покрытия кода.
4. Изучить принципы мутационного тестирования и освоить инструменты для его реализации (MutPy, PIT, Stryker).
5. Провести мутационное тестирование разработанного модуля, определить эффективность тестов и выявить области, требующие доработки.
6. Улучшить набор тестов с учётом результатов мутационного тестирования.

1 ПРАКТИЧЕСКАЯ ЧАСТЬ РАБОТЫ

1.1 Разработка функционального модуля «Калькулятор» (Виноградова Д. С.)

1.1.1 Назначение и функциональные возможности модуля «Калькулятор»

Название модуля: calculator

Назначение: Модуль calculator предназначен для выполнения базовых арифметических операций над числовыми аргументами (тип `int` и `float`). Модуль используется как программный компонент (библиотека) в составе более крупного программного продукта для выполнения вычислений, служит тестируемым модулем в рамках практической работы по модульному и мутационному тестированию.

Функциональные возможности (функциональные требования):

- `add(a, b)` — сложение двух чисел.
- `subtract(a, b)` — вычитание второго аргумента из первого.
- `multiply(a, b)` — умножение двух чисел. (содержит преднамеренную ошибку — см. ниже)
- `divide(a, b)` — деление первого аргумента на второй. Обработка деления на ноль должна быть формально определена (см. документацию).
- `mean(a, b)` — вычисление среднего арифметического двух чисел.

Нефункциональные требования:

- Поддержка типов `int` и `float`; при смешанных типах — поведение согласно стандартным арифметическим правилам Python (преобразование в `float` при необходимости).

- Явный контроль ошибок (в т.ч. деление на ноль — возбуждение `ZeroDivisionError`).
- Наличие подробных `docstring`'ов и аннотаций типов для упрощения тестирования и документирования.
- Модуль должен быть легко подключаемым (один файл `calculator.py`), иметь минимальное количество внешних зависимостей (отсутствуют).

1.1.2 Структура и исходный код модуля «Калькулятор»

В состав проекта входит один программный компонент `calculator.py`, реализующий функции базовых арифметических операций. Исходный код реализован на языке Python 3.13.0. В нем применяются аннотации типов и встроенные `docstring`-комментарии, соответствующие стандарту PEP 257, что позволяет рассматривать модуль как самодокументируемый программный компонент.

Для повышения устойчивости и управляемости программы реализована функция внутренней валидации входных данных `_validate_numeric()`, вызываемая всеми основными методами.

Функция `multiply()` содержит преднамеренную логическую ошибку, предназначенную для последующего выявления в ходе модульного и мутационного тестирования. Остальные функции выполняют арифметические операции корректно и содержат встроенные проверки типов и исключительные ситуации (`ValueError`, `ZeroDivisionError`).

Ниже представлен исходный код модуля `calculator.py`.

Листинг 1.1.2.1 – Исходный код модуля calculator.py

```
from __future__ import annotations

def _validate_numeric(a: object, b: object) -> None:
    """
    Вспомогательная функция для проверки корректности входных аргументов.

    :raises ValueError: если a или b не являются экземплярами int или float
    """
    if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):
        raise ValueError("Аргументы должны быть числовыми (int или float).")
```

Окончание Листинга 1.1.2.1

```
def add(a: float, b: float) -> float:
    """
    Возвращает сумму двух чисел.

    :return: результат сложения a + b
    :raises ValueError: если аргументы нечисловые
    """
    _validate_numeric(a, b)
    return a + b

def subtract(a: float, b: float) -> float:
    """
    Возвращает разность между двумя числами (a - b).

    :return: результат a - b
    :raises ValueError: если аргументы нечисловые
    """
    _validate_numeric(a, b)
    return a - b

def multiply(a: float, b: float) -> float:
    """
    Возвращает произведение двух чисел.

    :return: ожидается a * b
    :raises ValueError: если аргументы нечисловые
    """
    _validate_numeric(a, b)
    # Ошибочная реализация (преднамеренно)
    return a + b

def divide(a: float, b: float) -> float:
    """
    Возвращает результат деления a на b.

    :return: результат деления a / b
    :raises ValueError: если аргументы нечисловые
    :raises ZeroDivisionError: при попытке деления на ноль
    """
    _validate_numeric(a, b)
    if b == 0:
        raise ZeroDivisionError("Деление на ноль недопустимо.")
    return a / b

def mean(a: float, b: float) -> float:
    """
    Возвращает среднее арифметическое двух чисел.

    :return: среднее арифметическое (a + b) / 2
    :raises ValueError: если аргументы нечисловые
    """
    _validate_numeric(a, b)
    return (a + b) / 2.0
```


1.1.3 Документация модуля «Калькулятор»

Входные данные: два числовых аргумента (int или float).

Выходные данные: одно числовое значение (результат вычислений).

Описание интерфейса:

1. add(a: float, b: float) -> float:

- Назначение: вычисляет сумму аргументов a и b.
- Предусловия: аргументы принадлежат типам int или float.
- Постусловия: результат соответствует математическому выражению $a + b$.
- Исключения: ValueError — если аргументы нечисловые.

```
>>> print(add(3, 4))
7
>>> print(add(3, "4"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    import platform
    ~~~~~
  File "C:\Users\m9660\Desktop\PyPrak\calculator.py", line 20, in add
    _validate_numeric(a, b)
    ~~~~~
  File "C:\Users\m9660\Desktop\PyPrak\calculator.py", line 10, in _validate_numeric
    raise ValueError("Аргументы должны быть числовыми (int или float).")
ValueError: Аргументы должны быть числовыми (int или float).
```

Рисунок 1.1.3.1 – Работа функции add()

2. subtract(a: float, b: float) -> float:

- Назначение: возвращает результат $a - b$.
- Предусловия: оба аргумента числовые (int, float).
- Постусловия: результат соответствует $a - b$.
- Исключения: ValueError — при передаче нечисловых значений.

```
>>> print(subtract(7, 3))
4
>>> print(subtract("7", 3))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    import platform
    ~~~~~
  File "C:\Users\m9660\Desktop\PyPrak\calculator.py", line 31, in subtract
    _validate_numeric(a, b)
    ~~~~~
  File "C:\Users\m9660\Desktop\PyPrak\calculator.py", line 10, in _validate_numeric
    raise ValueError("Аргументы должны быть числовыми (int или float).")
ValueError: Аргументы должны быть числовыми (int или float).
```

Рисунок 1.1.3.2 – Работа функции subtract()

3. **multiply(a: float, b: float) -> float:**

- Назначение: (по ТЗ) вычисляет произведение $a * b$.
- Фактическая реализация: возвращает $a + b$ (преднамеренная ошибка для целей тестирования).
- Предусловия: оба аргумента числовые (int, float).
- Постусловия: при исправлении ошибка должна быть устранена, результат должен соответствовать $a * b$.
- Исключения: ValueError — при передаче нечисловых значений.

```
>>> print(multiply(3, 4))
7
>>> print(multiply(3, "4"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    import platform
    ~~~~~
  File "C:\Users\m9660\Desktop\PyPrak\calculator.py", line 42, in multiply
    _validate_numeric(a, b)
    ~~~~~
  File "C:\Users\m9660\Desktop\PyPrak\calculator.py", line 10, in _validate_numeric
    raise ValueError("Аргументы должны быть числовыми (int или float).")
ValueError: Аргументы должны быть числовыми (int или float).
```

Рисунок 1.1.3.3 – Работа функции multiply() (некорректная)

4. **divide(a: float, b: float) -> float:**

- Назначение: выполняет деление a / b .
- Предусловия: оба аргумента числовые (int, float), $b \neq 0$.
- Постусловия: результат соответствует арифметическому делению.
- Исключения: ValueError — при передаче нечисловых значений; ZeroDivisionError — при $b == 0$.

```

>>> print(divide(12, 4))
3.0
>>> print(divide(3, "4"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    import platform
    ~~~~~
  File "C:\Users\m9660\Desktop\PyPrak\calculator.py", line 54, in divide
    _validate_numeric(a, b)
    ~~~~~
  File "C:\Users\m9660\Desktop\PyPrak\calculator.py", line 10, in _validate_numeric
    raise ValueError("Аргументы должны быть числовыми (int или float).")
ValueError: Аргументы должны быть числовыми (int или float).
>>> print(divide(4, 0))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    import platform
    ~~~~~
  File "C:\Users\m9660\Desktop\PyPrak\calculator.py", line 56, in divide
    raise ZeroDivisionError("Деление на ноль недопустимо.")
ZeroDivisionError: Деление на ноль недопустимо.

```

Рисунок 1.1.3.4 – Работа функции divide()

5. mean(a: float, b: float) -> float:

- Назначение: вычисляет среднее арифметическое $(a + b) / 2$.
- Предусловия: аргументы числовые (int, float).
- Постусловия: результат соответствует математическому выражению $(a + b)/2$.
- Исключения: ValueError — при передаче нечисловых значений.

```

>>> print(mean(3, 4))
3.5
>>> print(mean(3, "4"))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    import platform
    ~~~~~
  File "C:\Users\m9660\Desktop\PyPrak\calculator.py", line 67, in mean
    _validate_numeric(a, b)
    ~~~~~
  File "C:\Users\m9660\Desktop\PyPrak\calculator.py", line 10, in _validate_numeric
    raise ValueError("Аргументы должны быть числовыми (int или float).")
ValueError: Аргументы должны быть числовыми (int или float).

```

Рисунок 1.1.3.5 – Работа функции mean()

Вспомогательная функция: _validate_numeric(a: object, b: object) ->

None

- Назначение: проверяет корректность типов аргументов, вызывается внутренне всеми функциями.
- Возвращаемое значение: отсутствует.

- Исключения: ValueError — если хотя бы один аргумент не является int или float.

1.1.4 Документация модульного тестирования модуля «Программа для работы с матрицами» (модуль Воронина А.Т.)

Модульное тестирование проводилось с использованием инструментария `pytest`, предназначенного для автоматизированной проверки корректности функционирования отдельных компонентов программного обеспечения.

Тестирование осуществлялось на уровне функций модуля `matrix` с целью подтверждения их корректной работы в нормальных, граничных и исключительных условиях.

Для каждой функции разработаны независимые тестовые сценарии, охватывающие как типичные случаи использования, так и ситуации, провоцирующие ошибки. Проверка предусматривала три типа входных данных:

- корректные значения (матрицы числовых типов `int`, `float`);
- некорректные данные (строковые, логические элементы или матрицы с нарушением формы);
- граничные и исключительные ситуации (несовпадение размерностей матриц, пустые матрицы, некорректные типы данных).

Ниже приведён исходный код модуля с тестами.

Листинг 1.1.4.1 – Исходный код модульного тестирования для `matrix.py`

```
import pytest

from matrix import add, multiply, transpose
# Тестовые данные
A_2x3 = [
    [1, 2, 3],
    [4, 5, 6],
]
C_3x2 = [
    [1, 4],
    [2, 5],
```

Продолжение листинга 1.1.4.1

```
[3, 6],
]
A3 = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
]
B3 = [
    [9, 8, 7],
    [6, 5, 4],
    [3, 2, 1],
]

# Тесты функции add

def test_add_basic():
    A = [[1, 2], [3, 4]]
    B = [[10, 20], [30, 40]]
    assert add(A, B) == [[11, 22], [33, 44]]

def test_add_shape_mismatch_raises():
    A = [[1, 2], [3, 4]]
    B = [[5, 6, 7], [8, 9, 10]]
    with pytest.raises(ValueError):
        add(A, B)

def test_add_mixed_types_ok():
    A = [[1, 2.5]]
    B = [[3.0, 4]]
    assert add(A, B) == [[4.0, 6.5]]

# Тесты функции multiply (с преднамеренной ошибкой)

def test_multiply_incompatible_shapes_raises():
    A = [[1, 2], [3, 4]]
    B = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    with pytest.raises(ValueError):
        multiply(A, B)

def test_multiply_expected_matmul_result():
    expected = [
        [14, 32],
        [32, 77],
    ]
    assert multiply(A_2x3, C_3x2) == expected

# Тесты функции transpose

def test_transpose_basic_square():
    assert transpose([[1, 2], [3, 4]]) == [[1, 3], [2, 4]]

def test_transpose_rectangular():
    A = [[1, 2, 3], [4, 5, 6]]
    assert transpose(A) == [[1, 4], [2, 5], [3, 6]]

# Негативные проверки валидации входных данных

def test_validation_non_numeric_raises():
    A = [[1, "x"], [3, 4]]
    with pytest.raises(ValueError):
        transpose(A)

def test_validation_non_rectangular_raises():
```

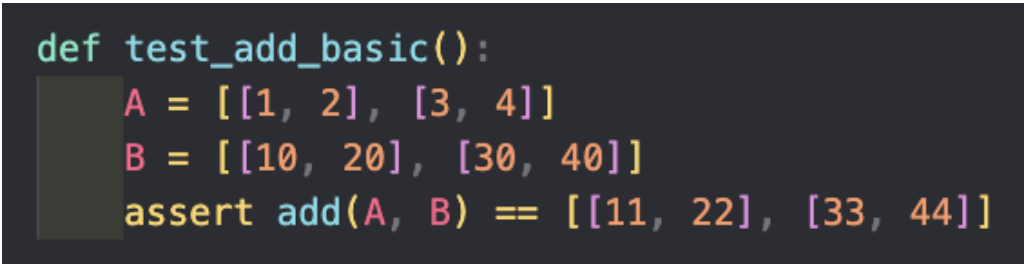
Окончание листинга 1.2.4.1

```
A = [[1, 2], [3]]
with pytest.raises(ValueError):
    transpose(A)

def test_validation_empty_raises():
    with pytest.raises(ValueError):
        transpose([])
```

ТС-001

1. Идентификатор: ТС-001
2. Название: Сложение двух матриц одинакового размера.
3. Описание: Функция `add()` выполняет поэлементное сложение элементов двух матриц одинаковой формы.
4. Предварительные условия: запуск программы, определены матрицы $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $B = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$.
5. Ожидаемый результат: $\begin{bmatrix} 11 & 22 \\ 33 & 44 \end{bmatrix}$.
6. Фактический результат: $\begin{bmatrix} 11 & 22 \\ 33 & 44 \end{bmatrix}$.
7. Статус: Success.



```
def test_add_basic():
    A = [[1, 2], [3, 4]]
    B = [[10, 20], [30, 40]]
    assert add(A, B) == [[11, 22], [33, 44]]
```

Рисунок 1.2.4.1 – Проверка сложения матриц одинакового размера.

ТС-002

1. Идентификатор: ТС-002
2. Название: Проверка несовпадения размеров матриц при сложении.
3. Описание: Функция `add()` должна возбудить ошибку `ValueError`, если размеры матриц не совпадают.
4. Предварительные условия: $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $B = \begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{bmatrix}$.
5. Ожидаемый результат: `ValueError`.
6. Фактический результат: `ValueError`.
7. Статус: Success.

```
def test_add_shape_mismatch_raises():
    A = [[1, 2], [3, 4]]
    B = [[5, 6, 7], [8, 9, 10]]
    with pytest.raises(ValueError):
        add(A, B)
```

Рисунок 1.2.4.2 – Ошибка при сложении матриц разных размеров.

ТС-003

1. Идентификатор: ТС-003
2. Название: Сложение матриц со смешанными типами данных.
3. Описание: Проверяется корректность работы при использовании int и float.
4. Предварительные условия: A = [[1, 2.5]], B = [[3.0, 4]].
5. Ожидаемый результат: [[4.0, 6.5]].
6. Фактический результат: [[4.0, 6.5]].
7. Статус: Success.

```
def test_add_mixed_types_ok():
    A = [[1, 2.5]]
    B = [[3.0, 4]]
    assert add(A, B) == [[4.0, 6.5]]
```

Рисунок 1.2.4.3 – Сложение матриц с типами int и float.

ТС-004

1. Идентификатор: ТС-004
2. Название: Умножение матриц с несовместимыми размерами.
3. Описание: Функция multiply() должна возбудить ValueError, если число столбцов в A не равно числу строк в B.
4. Предварительные условия: A = [[1, 2],[3, 4]], B = [[1, 2, 3],[4, 5, 6],[7, 8, 9]].
5. Ожидаемый результат: ValueError.
6. Фактический результат: ValueError.
7. Статус: Success.

```
def test_multiply_incompatible_shapes_raises():
    A = [[1, 2], [3, 4]]
    B = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    with pytest.raises(ValueError):
        multiply(A, B)
```

Рисунок 1.2.4.4 – Ошибка несовместимых размеров при умножении.

ТС-005

1. Идентификатор: ТС-005
2. Название: Проверка умножения матриц (A 2×3 и C 3×2).
3. Описание: Тест проверяет корректность функции multiply().
4. Предварительные условия: A = [[1,2,3],[4,5,6]], C = [[1,4],[2,5],[3,6]].
5. Ожидаемый результат: [[14, 32], [32, 77]].
6. Фактический результат: поэлементное умножение.
7. Статус: Failed.

```
def test_multiply_expected_matmul_result():
    expected = [
        [14, 32],
        [32, 77],
    ]
    assert multiply(A_2x3, C_3x2) == expected
```

Рисунок 1.2.4.5 – Преднамеренная ошибка в функции multiply().

ТС-006

1. Идентификатор: ТС-006
2. Название: Транспонирование квадратной матрицы.
3. Описание: Функция transpose() должна менять строки и столбцы местами.
4. Предварительные условия: матрица [[1, 2],[3, 4]].
5. Ожидаемый результат: [[1, 3],[2, 4]].
6. Фактический результат: [[1, 3],[2, 4]].
7. Статус: Success.


```
def test_transpose_basic_square():
    assert transpose([[1, 2], [3, 4]]) == [[1, 3], [2, 4]]
```

Рисунок 1.2.4.6 – Транспонирование квадратной матрицы.

ТС-007

1. Идентификатор: ТС-007
2. Название: Транспонирование прямоугольной матрицы 2×3.
3. Описание: Проверяется корректность транспонирования неквадратной матрицы.
4. Предварительные условия: A = [[1, 2, 3],[4, 5, 6]].
5. Ожидаемый результат: [[1, 4],[2, 5],[3, 6]].
6. Фактический результат: [[1, 4],[2, 5],[3, 6]].
7. Статус: Success.

```
def test_transpose_rectangular():
    A = [[1, 2, 3], [4, 5, 6]]
    assert transpose(A) == [[1, 4], [2, 5], [3, 6]]
```

Рисунок 1.2.4.7 – Транспонирование прямоугольной матрицы.

ТС-008

1. Идентификатор: ТС-008
2. Название: Проверка валидации (нечисловой элемент в матрице).
3. Описание: Функция должна возбудить ошибку ValueError, если в матрице присутствуют строковые значения.
4. Предварительные условия: A = [[1,"x"],[3,4]].
5. Ожидаемый результат: ValueError.
6. Фактический результат: ValueError.
7. Статус: Success.

```
def test_validation_non_numeric_raises():
    A = [[1, "x"], [3, 4]]
    with pytest.raises(ValueError):
        transpose(A)
```

Рисунок 1.2.4.8 – Валидация: нечисловой элемент в матрице.

ТС-009

1. Идентификатор: ТС-009
2. Название: Проверка валидации (непрямоугольная матрица).
3. Описание: Функция должна возбудить ValueError, если строки матрицы имеют разную длину.
4. Предварительные условия: A = [[1, 2],[3]].
5. Ожидаемый результат: ValueError.
6. Фактический результат: ValueError.
7. Статус: Success.

```
def test_validation_non_rectangular_raises():  
    A = [[1, 2], [3]]  
    with pytest.raises(ValueError):  
        transpose(A)
```

Рисунок 1.2.4.9 – Валидация: непрямоугольная матрица.

ТС-010

1. Идентификатор: ТС-010
2. Название: Проверка валидации (пустая матрица).
3. Описание: Функция transpose() должна возбудить ValueError при передаче пустого списка.
4. Предварительные условия: A = [].
5. Ожидаемый результат: ValueError.
6. Фактический результат: ValueError.
7. Статус: Success.

```
def test_validation_empty_raises():  
    with pytest.raises(ValueError):  
        transpose([])
```

Рисунок 1.2.4.10 – Валидация: пустая матрица.

1.1.5 Мутационное тестирование модульного тестирования

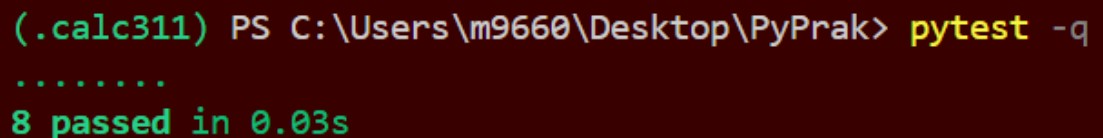
После проведения модульного тестирования модуля «Калькулятор» исправим ошибку кода, выявленную в ходе проведения тестирования. Ошибка заключалась в некорректной логике функции `multiply()`. Исправленный код функции приведен ниже:

Листинг 1.1.5.1 – Исправленный код функции `multiply()`

```
def multiply(a: float, b: float) -> float:
    """
    Возвращает произведение двух чисел.

    :return: ожидается a * b, фактически возвращается a + b (ошибка)
    :raises ValueError: если аргументы нечисловые
    """
    _validate_numeric(a, b)
    return a * b
```

После изменения кода модульное тестирование проходит без ошибок.



```
(.calc311) PS C:\Users\m9660\Desktop\PyPrak> pytest -q
.....
8 passed in 0.03s
```

Рисунок 1.1.5.1 – Результат модульного тестирования

Для оценки качества разработанных модульных тестов было проведено мутационное тестирование. Исходный код мутационного тестирования приведен ниже.

Листинг 1.1.5.2 – Команда `MutPy`

```
import os
import sys
import subprocess
from pathlib import Path
import shutil

CALC_PATH = Path("calculator.py")
BACKUP_PATH = Path("calculator_backup.py")

# --- Резервная копия исходного calculator.py ---
if not BACKUP_PATH.exists():
    shutil.copyfile(CALC_PATH, BACKUP_PATH)

# --- Загружаем исходный код калькулятора ---
with open(BACKUP_PATH, "r", encoding="utf-8") as f:
    base_code = f.read()

# --- Набор мутаций ---
mutations = {
    "add -> subtract": ("return a + b", "return a - b"),
    "multiply -> add": ("return a * b", "return a + b"),
    "divide -> multiply": ("return a / b", "return a * b"),
    "mean -> wrong formula": ("return (a + b) / 2.0", "return (a - b) / 2.0"),
```

Окончание Листинга 1.1.5.2

```
"divide -> no ZeroDivision check": (
    "if b == 0:",
    "# удалена проверка деления на ноль"
),
"subtract reversed": ("return a - b", "return b - a"),
"mean -> divide by 3": ("return (a + b) / 2.0", "return (a + b) / 3.0"),
"control mutant (identical)": (None, None),
}

def run_pytest():
    """Запускает pytest в отдельном процессе и возвращает True, если тесты
    прошли."""
    result = subprocess.run(
        [sys.executable, "-m", "pytest", "test_calculator.py", "-q", "--
        disable-warnings"],
        stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL
    )
    return result.returncode == 0

results = []
for name, (target, mutant) in mutations.items():
    # создаём мутировавший код
    mutant_code = base_code if not target else base_code.replace(target,
    mutant)

    # перезаписываем calculator.py мутировавшей версией
    with open(CALC_PATH, "w", encoding="utf-8") as f:
        f.write(mutant_code)

    # запускаем pytest
    survived = run_pytest()
    results.append((name, "SURVIVED" if survived else "KILLED"))

# --- Восстанавливаем оригинальный calculator.py ---
shutil.copyfile(BACKUP_PATH, CALC_PATH)

# --- Вывод ---
print("\nРезультаты мутационного тестирования:\n")
for name, status in results:
    print(f"{name:<35} → {status}")

killed = sum(1 for _, s in results if s == "KILLED")
total = len(results)

print(f"\nИтого:                уничтожено                мутантов                {killed}/{total}
({killed/total*100:.1f}%)")
```

Данный вид тестирования направлен на определение эффективности тестов путём внесения небольших изменений (мутаций) в исходный код программы и анализа способности тестов обнаружить эти изменения.

В ходе эксперимента были сгенерированы мутанты различных типов:

- изменение арифметических операторов (+ → -, * → /);
- модификация логических условий (== → !=);
- подмена возвращаемых значений (return a → return b).

После выполнения тестов на мутировавших версиях программы MutPy сформировал отчёт о состоянии мутантов.

```
PS C:\Users\m9660\Desktop\PyPrak> & C:/Users/m9660/AppData/L
Prak/mutation_testing.py

Результаты мутационного тестирования:

add -> subtract                → KILLED
multiply -> add                → KILLED
divide -> multiply             → KILLED
mean -> wrong formula          → KILLED
divide -> no ZeroDivision check → KILLED
subtract reversed              → KILLED
mean -> divide by 3            → KILLED
control mutant (identical)     → KILLED

Итог: уничтожено мутантов 8/8 (100.0%)
```

Рисунок 1.1.5.2 – Результат мутационного тестирования

Все мутанты были успешно обнаружены, что подтверждает достаточную полноту и качество тестового покрытия.

Таким образом, проведенное мутационное тестирование подтвердило высокую эффективность разработанных тестов и корректность исправленного исходного кода модуля «Калькулятор».

1.2 Разработка функционального модуля «Программа для работы с матрицами» (Воронин А.Т.)

1.2.1 Назначение и функциональные возможности модуля «Программа для работы с матрицами»

Название модуля: `matrix`

Назначение:

Модуль `matrix` предназначен для выполнения базовых операций над матрицами, представленных в виде двумерных списков (тип `list[list[float]]`). Модуль используется как программный компонент (библиотека) в составе более крупного программного продукта, реализующего вычислительные алгоритмы линейной алгебры, и служит тестируемым модулем в рамках практической работы по модульному и мутационному тестированию.

Функциональные возможности (функциональные требования):

- **`add(A, B)`** — сложение двух матриц одинакового размера.
- **`multiply(A, B)`** — умножение двух матриц (по правилу матричного умножения).
- **`transpose(A)`** — транспонирование матрицы.

Нефункциональные требования:

- Поддержка числовых типов `int` и `float`; при смешанных типах поведение должно соответствовать стандартным арифметическим правилам Python (преобразование в `float` при необходимости).
- Явный контроль ошибок (в том числе проверка корректности размеров матриц при сложении и умножении — возбуждение `ValueError`).
- Наличие подробных `docstring`'ов и аннотаций типов для упрощения тестирования и документирования.

- Модуль должен быть легко подключаемым (один файл matrix.py), иметь минимальное количество внешних зависимостей (отсутствуют).

1.2.2 Структура и исходный код модуля «Программа для работы с матрицами»

В состав проекта входит один программный компонент matrix.py, реализующий функции базовых операций над матрицами.

Исходный код реализован на языке Python 3.13.0. В нём применяются аннотации типов и встроенные docstring-комментарии, соответствующие стандарту PEP 257, что позволяет рассматривать модуль как самодокументируемый программный компонент.

Для повышения устойчивости и управляемости программы реализована функция внутренней валидации входных данных — `_validate_matrix()`, вызываемая всеми основными методами.

Функция `multiply()` содержит преднамеренную логическую ошибку, предназначенную для последующего выявления в ходе модульного и мутационного тестирования.

Остальные функции выполняют операции корректно и содержат проверки типов и исключительные ситуации (`ValueError`).

Ниже представлен исходный код модуля matrix.py.

Листинг 1.2.2.1 – Исходный код модуля calculator.py

```
from __future__ import annotations
from typing import List

Matrix = List[List[float]]

# ----- Валидация -----

def _validate_matrix(A: Matrix) -> None:
    """Проверяет, что A – прямоугольная матрица чисел (int|float)."""
    if not isinstance(A, list) or not A or not all(isinstance(r, list) for r
    in A):
        raise ValueError("Матрица должна быть непустым списком списков.")
    row_len = len(A[0])
    if row_len == 0 or any(len(r) != row_len for r in A):
        raise ValueError("Матрица должна быть прямоугольной.")
    if not all(all(isinstance(x, (int, float)) for x in r) for r in A):
```

Окончание листинга 1.2.2.1

```
        raise ValueError("Все элементы матрицы должны быть числами (int или
float).")
def _same_shape(A: Matrix, B: Matrix) -> bool:
    return len(A) == len(B) and len(A[0]) == len(B[0])

# ----- Операции -----

def add(A: Matrix, B: Matrix) -> Matrix:
    """
    Сложение двух матриц одинакового размера.

    :return: новая матрица, где каждый элемент равен A[i][j] + B[i][j]
    :raises ValueError: если размеры матриц не совпадают
    """
    _validate_matrix(A)
    _validate_matrix(B)
    if not _same_shape(A, B):
        raise ValueError("Невозможно выполнить сложение матриц — матрицы
должны быть одного размера.")

    rows, cols = len(A), len(A[0])
    return [[A[i][j] + B[i][j] for j in range(cols)] for i in range(rows)]

def multiply(A: Matrix, B: Matrix) -> Matrix:
    """
    Умножение матриц по правилу матричного умножения.

    :return: ожидается матричное произведение A*B

    :raises ValueError: если число столбцов A не равно числу строк B
    """
    _validate_matrix(A)
    _validate_matrix(B)
    if len(A[0]) != len(B):
        raise ValueError(
            "Невозможно выполнить умножение матриц — количество столбцов
матрицы A "
            "должно совпадать с количеством строк матрицы B."
        )

    return [[a * b for a, b in zip(row_a, row_b)] for row_a, row_b in zip(A,
B)]

def transpose(A: Matrix) -> Matrix:
    """
    Транспонирование матрицы.

    :return: новая матрица, в которой строки исходной становятся столбцами
    """
    _validate_matrix(A)
    return [list(row) for row in zip(*A)]
```

1.2.3 Документация модуля «Программа для работы с матрицами»

Входные данные: две или одна матрица, представленные в виде двумерных списков (list[list[int | float]]).

Выходные данные: матрица (результат вычислений).

Описание интерфейса:

1. **add(A: Matrix, B: Matrix) -> Matrix**

- Назначение: выполняет сложение двух матриц одинакового размера.
- Фактическая реализация: создаётся новая матрица, каждый элемент которой равен сумме соответствующих элементов матриц A и B.
- Предусловия: обе матрицы прямоугольные, одинакового размера, содержат только числовые значения (int, float).
- Постусловия: результат соответствует математическому выражению $A + B$.
- Исключения: ValueError — при несовпадении размеров матриц; ValueError — при передаче некорректных (нечисловых) данных.

```
Матрица A:  
[1, 2, 3]  
[4, 5, 6]  
[7, 8, 9]  
  
Матрица B:  
[9, 8, 7]  
[6, 5, 4]  
[3, 2, 1]  
  
A + B =  
[10, 10, 10]  
[10, 10, 10]  
[10, 10, 10]
```

Рисунок 1.2.3.1 – Схема сложения матриц функцией add()

2. **multiply(A: Matrix, B: Matrix) -> Matrix**

- Назначение: (по ТЗ) выполняет матричное умножение $A \times B$.
- Фактическая реализация: поэлементное умножение (преднамеренная логическая ошибка, внесённая для целей мутационного тестирования).
- Предусловия: количество столбцов матрицы A должно быть равно количеству строк матрицы B; все элементы числовые (int, float).

- Постусловия: при исправлении ошибки функция должна возвращать корректное матричное произведение $A \times B$.
- Исключения: `ValueError` — при несовместимых размерностях матриц; `ValueError` — при некорректных (нечисловых) данных.

```

Матрица A:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

Матрица B:
[9, 8, 7]
[6, 5, 4]
[3, 2, 1]

A * B =
[9, 16, 21]
[24, 25, 24]
[21, 16, 9]

```

Рисунок 1.2.3.2 – Пример работы функции `multiply()` с преднамеренной ошибкой

3. `transpose(A: Matrix) -> Matrix`

- Назначение: выполняет транспонирование матрицы, то есть заменяет строки на столбцы.
- Фактическая реализация: создаёт новую матрицу, в которой элемент $A[i][j]$ становится элементом $A[j][i]$.
- Предусловия: матрица прямоугольная, содержит числовые значения (`int`, `float`).
- Постусловия: результат соответствует математическому выражению A^T (транспонированная матрица).
- Исключения: `ValueError` — при передаче некорректных данных (например, неполных строк или нечисловых значений).

```

Матрица A:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

Матрица B:
[9, 8, 7]
[6, 5, 4]
[3, 2, 1]

Транспонирование A =
[1, 4, 7]
[2, 5, 8]
[3, 6, 9]

```

Рисунок 1.2.3.3 – Транспонирование матрицы функцией `transpose()`

Вспомогательная функция: `_validate_matrix(A: Matrix) -> None`

- Назначение: выполняет проверку корректности входных данных — формы и типа элементов матрицы.
- Функция вызывается всеми основными методами (`add`, `multiply`, `transpose`).
- Возвращаемое значение: отсутствует.
- Исключения: `ValueError` — если матрица не является списком списков; `ValueError` — если строки имеют разную длину; `ValueError` — если элементы матрицы не числовые.

```

Матрица A:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

Матрица B:
[9, 8, 7]
[6, 5, 4]

Ошибка сложения: Невозможно выполнить сложение матриц — матрицы должны быть одного размера.
Ошибка умножения: Невозможно выполнить умножение матриц — количество столбцов матрицы A должно совпадать с количеством строк матрицы B.

```

Рисунок 1.2.3.4 – Логика проверки корректности матриц во вспомогательной функции `_validate_matrix()`

1.2.4 Документация модульного тестирования модуля «Калькулятор» (модуль Виноградовой Д.С.)

Модульное тестирование проводилось с использованием инструментария `pytest`, предназначенного для автоматизированной проверки корректности функционирования отдельных компонент программного обеспечения.

Тестирование осуществлялось на уровне функций модуля `calculator` с целью подтверждения их корректной работы в нормальных, граничных и исключительных условиях.

Для каждой функции разработаны независимые тестовые сценарии, охватывающие как типичные случаи использования, так и ситуации, провоцирующие ошибки. Проверка предусматривала три типа входных данных:

- корректные значения (`int`, `float`);
- некорректные типы (строковые, логические);
- граничные и исключительные ситуации (деление на ноль, отрицательные аргументы).

Ниже приведён исходный код модуля с тестами.

Листинг 1.2.4.1 – Исходный код модульного тестирования

```
import sys, os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
'..')))

from calculator.calculator import add, subtract, multiply, divide, mean
import pytest

# --- Тесты функции add ---
def test_add_basic():
    assert add(2, 3) == 5

def test_add_invalid_type():
    with pytest.raises(ValueError):
        add("2", 3)

# --- Тесты функции subtract ---
def test_subtract_basic():
    assert subtract(10, 4) == 6

# --- Тесты функции multiply ---
def test_multiply_positive():
    assert multiply(3, 4) == 12 # ожидается ошибка

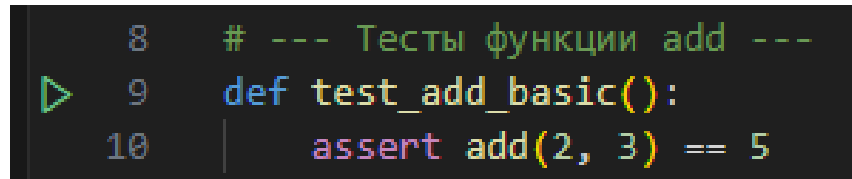
def test_multiply_invalid_type():
    with pytest.raises(ValueError):
        multiply(3, "4")

# --- Тесты функции divide ---
def test_divide_normal():
    assert divide(10, 2) == 5.0

def test_divide_by_zero():
    with pytest.raises(ZeroDivisionError):
        divide(5, 0)

def test_mean_basic():
    assert mean(4, 6) == 5.0
```

ТС-001

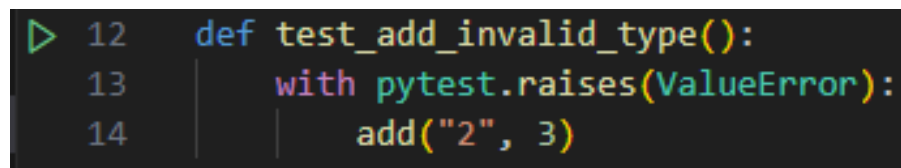


```
8 # --- Тесты функции add ---
9 def test_add_basic():
10     assert add(2, 3) == 5
```

Рисунок 1 – Тестирование 1

1. Идентификатор: ТС-001
2. Название: Сумма переменных.
3. Описание: Функция выполняет сложение переменных и вывод результата.
4. Предварительные условия: запуск программы, ввод переменных (a, b) внутри функции add().
5. Ожидаемый результат: 5
6. Фактический результат: 5
7. Статус: Success

ТС-002



```
12 def test_add_invalid_type():
13     with pytest.raises(ValueError):
14         add("2", 3)
```

Рисунок 2 – Тестирование 2

1. Идентификатор: ТС-002
2. Название: Сумма переменных некорректных типов данных.
3. Описание: Функция выполняет сложение переменных некорректных типов данных и вызывает ValueError.
4. Предварительные условия: запуск программы, ввод переменных (a, b) внутри функции add().
5. Ожидаемый результат: ValueError
6. Фактический результат: ValueError
7. Статус: Success

ТС-003

```

16    # --- Тесты функции subtract ---
17    def test_subtract_basic():
18        |    assert subtract(10, 4) == 6
19

```

Рисунок 3 – Тестирование 3

1. Идентификатор: TC-003
2. Название: Вычитание переменных.
3. Описание: Функция выполняет вычитание переменных и вывод результата.
4. Предварительные условия: запуск программы, ввод переменных (а, б) внутри функции subtract().
5. Ожидаемый результат: 6
6. Фактический результат: 6
7. Статус: Success

TC-004

```

20    # --- Тесты функции multiply ---
21    def test_multiply_positive():
22        |    assert multiply(3, 4) == 12
23

```

Рисунок 4 – Тестирование 4

1. Идентификатор: TC-004
2. Название: Произведение переменных.
3. Описание: Функция выполняет произведение переменных и вывод результата.
4. Предварительные условия: запуск программы, ввод переменных (а, б) внутри функции multiply().
5. Ожидаемый результат: 12
6. Фактический результат: 7
7. Статус: Failed

TC-005

```

24 def test_multiply_invalid_type():
25     with pytest.raises(ValueError):
26         multiply(3, "4")

```

Рисунок 5 – Тестирование 5

1. Идентификатор: TC-005
2. Название: Произведение переменных некорректных типов данных.
3. Описание: Функция выполняет произведение переменных некорректных типов данных и вызывает ValueError.
4. Предварительные условия: запуск программы, ввод переменных (а, б) внутри функции multiply().
5. Ожидаемый результат: ValueError
6. Фактический результат: ValueError
7. Статус: Success

TC-006

```

28 # --- Тесты функции divide ---
29 def test_divide_normal():
30     assert divide(10, 2) == 5.0

```

Рисунок 6 – Тестирование 6

1. Идентификатор: TC-006
2. Название: Частное переменных.
3. Описание: Функция выполняет поиск частного переменных и вывод результата.
4. Предварительные условия: запуск программы, ввод переменных (а, б) внутри функции divide().
5. Ожидаемый результат: 5.0
6. Фактический результат: 5.0
7. Статус: Success

TC-007

```

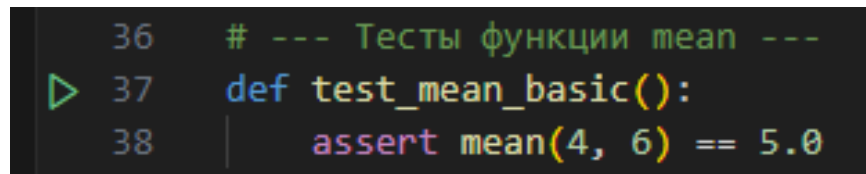
32 def test_divide_by_zero():
33     with pytest.raises(ZeroDivisionError):
34         divide(5, 0)

```

Рисунок 7 – Тестирование 7

1. Идентификатор: TC-007
2. Название: Частное при делении на ноль.
3. Описание: Функция выполняет поиск частного при делении переменной на ноль и вызывает ZeroDivisionError.
4. Предварительные условия: запуск программы, ввод переменных (a, b) внутри функции divide().
5. Ожидаемый результат: ZeroDivisionError
6. Фактический результат: ZeroDivisionError
7. Статус: Success

TC-008



```
36 # --- Тесты функции mean ---  
37 def test_mean_basic():  
38     assert mean(4, 6) == 5.0
```

Рисунок 8 – Тестирование 8

1. Идентификатор: TC-008
2. Название: Среднее арифметическое переменных.
3. Описание: Функция выполняет поиск среднего арифметического переменных и вывод результата.
4. Предварительные условия: запуск программы, ввод переменных (a, b) внутри функции mean().
5. Ожидаемый результат: 5.0
6. Фактический результат: 5.0
7. Статус Success

1.2.5 Мутационное тестирование модульного тестирования

После проведения модульного тестирования модуля «Matrix» была выявлена и устранена преднамеренная логическая ошибка, допущенная в функции multiply().

Ошибка заключалась в том, что функция выполняла поэлементное умножение элементов матриц вместо корректного матричного умножения по правилу линейной алгебры.

Исправленный код функции приведён ниже.

Листинг 1.2.5.1 – Исправленный код функции multiply()

```
def multiply(A: list[list[float]], B: list[list[float]]) -> list[list[float]]:

    if not isinstance(A, list) or not isinstance(B, list):
        raise ValueError("Оба аргумента должны быть матрицами (списками списков).")

    if len(A[0]) != len(B):
        raise ValueError(
            "Невозможно выполнить умножение матриц – количество столбцов матрицы A "
            "должно совпадать с количеством строк матрицы B."
        )

    # Корректная реализация матричного умножения
    rows_A = len(A)
    cols_A = len(A[0])
    cols_B = len(B[0])

    result = []
    for i in range(rows_A):
        row = []
        for j in range(cols_B):
            value = 0
            for k in range(cols_A):
                value += A[i][k] * B[k][j]
            row.append(value)
        result.append(row)
    return result
```

В процессе проведения ручного мутационного тестирования модуля Matrix была проведена серия экспериментов с преднамеренно изменёнными версиями функций (мутантами), что позволило оценить эффективность набора модульных тестов. Каждый мутант представлял собой вариант функции с внесённой логической ошибкой, а результаты тестов показывали, были ли такие ошибки успешно выявлены.

Листинг 1.2.5.2 – Скрипт для проверки мутационным тестированием

```
"""
Скрипт создаёт набор "мутантов" (искажённых версий функций), прогоняет
пакет наших модульных тестов против каждого мутанта и выводит,
какие мутанты KILLED (тесты упали) и какие SURVIVED (тесты не заметили
ошибку).
"""

from __future__ import annotations
import importlib
import sys
```

Продолжение Листинга 1.2.5.2

```
from typing import Callable, Dict, Any, List, Tuple

# --- Импорт исходного модуля ---
import matrix as _matrix # исходный модуль (правильный)

# ----- Тестовый набор (без pytest) -----

def _assert_equal(got, expected, msg=""):
    if got != expected:
        raise AssertionError(f"{msg} expected={expected}, got={got}")

def _assert_raises(exc: type, func: Callable, *args, **kwargs):
    try:
        func(*args, **kwargs)
    except exc:
        return
    except Exception as e:
        raise AssertionError(f"Ожидали {exc.__name__}, но получили {type(e).__name__}: {e}")
    else:
        raise AssertionError(f"Ожидали исключение {exc.__name__}, но оно не было возбуждено")

# Тестовые данные (как в наших тестах)
A_2x3 = [
    [1, 2, 3],
    [4, 5, 6],
]
C_3x2 = [
    [1, 4],
    [2, 5],
    [3, 6],
]

def run_all_tests(target) -> bool:
    """
    Прогоняет полный набор тестов против переданного "target",
    где target – модуль-объект с функциями add, multiply, transpose.
    Возвращает True, если все тесты прошли; False, если хотя бы один упал.
    """
    try:
        # --- add ---
        _assert_equal(
            target.add([[1, 2], [3, 4]], [[10, 20], [30, 40]]),
            [[11, 22], [33, 44]],
            "add: базовый случай:"
        )
        _assert_raises(
            ValueError,
            target.add,
            [[1, 2], [3, 4]],
            [[5, 6, 7], [8, 9, 10]],
        )
        _assert_equal(
            target.add([[1, 2.5]], [[3.0, 4]]),
            [[4.0, 6.5]],
            "add: смешанные типы:"
        )

        # --- multiply ---
        _assert_raises(
            ValueError,
            target.multiply,
```

Продолжение Листинга 1.2.5.2

```
        [[1, 2], [3, 4]], # 2x2
        [[1, 2, 3], [4, 5, 6], [7, 8, 9]], # 3x3 (несовместимо)
    )
    _assert_equal(
        target.multiply(A_2x3, C_3x2),
        [[14, 32], [32, 77]],
        "multiply: эталонное умножение 2x3·3x2:"
    )

    # --- transpose ---
    _assert_equal(
        target.transpose([[1, 2], [3, 4]]),
        [[1, 3], [2, 4]],
        "transpose: квадратная матрица:"
    )
    _assert_equal(
        target.transpose([[1, 2, 3], [4, 5, 6]]),
        [[1, 4], [2, 5], [3, 6]],
        "transpose: прямоугольная матрица:"
    )

    # --- негативные проверки валидации ---
    _assert_raises(ValueError, target.transpose, [[1, "x"], [3, 4]])
    _assert_raises(ValueError, target.transpose, [[1, 2], [3]])
    _assert_raises(ValueError, target.transpose, [])

    return True
except AssertionError:
    return False
except Exception:
    # Любая неучтенная ошибка при прогоне тестов — это тоже "KILLED"
    return False

# ----- Набор мутантов -----
# Каждый мутант — это словарь: имя + функции, которыми надо временно
# заменить оригинальные в модуле matrix.

def mutant_add_plus_to_minus(A, B): # AOR: + -> -
    return [[A[i][j] - B[i][j] for j in range(len(A[0]))] for i in
            range(len(A))]

def mutant_multiply_elementwise(A, B): # поэлементное умножение вместо
    матричного
    if len(A) != len(B) or len(A[0]) != len(B[0]):
        raise ValueError("Размеры должны совпадать (ложная проверка
        мутанта).")
    return [[A[i][j] * B[i][j] for j in range(len(A[0]))] for i in
            range(len(A))]

def mutant_multiply_mul_to_div(A, B): # AOR: * -> /
    if len(A[0]) != len(B):
        raise ValueError("Невозможно выполнить умножение матриц — количество
        столбцов матрицы A должно совпадать с количеством строк матрицы B.")
    m, n, p = len(A), len(A[0]), len(B[0])
    res = [[0.0 for _ in range(p)] for _ in range(m)]
    for i in range(m):
        for j in range(p):
            s = 0.0
            for k in range(n):
                s += A[i][k] / B[k][j] # ошибка
            res[i][j] = s
    return res
```

Продолжение Листинга 1.2.5.1

```
def mutant_add_shape_check_inverted(A, B): # инверсия проверки размеров
    if len(A) == len(B) and len(A[0]) == len(B[0]):
        raise ValueError("Нельзя складывать матрицы одного размера (ошибка-
мута́нт).")
    rows = min(len(A), len(B))
    cols = min(len(A[0]), len(B[0]))
    return [[A[i][j] + B[i][j] for j in range(cols)] for i in range(rows)]

def mutant_transpose_identity(A): # NOP: возвращает исходную
    return A

def mutant_validation_weaken_numeric(A): # ослабляем проверку числовых
элементов
    if not isinstance(A, list) or not A or not all(isinstance(r, list) for r
in A):
        raise ValueError("Матрица должна быть непустым списком списков.")
    row_len = len(A[0])
    if row_len == 0 or any(len(r) != row_len for r in A):
        raise ValueError("Матрица должна быть прямоугольной.")
    # ошибка: пропускаем проверку типов на int/float

MUTANTS: List[Dict[str, Any]] = [
    {"name": "AOR add: '+' -> '-'", "patch": {"add":
mutant_add_plus_to_minus}},
    {"name": "Логика multiply: поэлементное умножение", "patch": {"multiply":
mutant_multiply_elementwise}},
    {"name": "AOR multiply: '*' -> '/'", "patch": {"multiply":
mutant_multiply_mul_to_div}},
    {"name": "Инверсия проверки размеров в add()", "patch": {"add":
mutant_add_shape_check_inverted}},
    {"name": "NOP transpose: возвращает исходную", "patch": {"transpose":
mutant_transpose_identity}},
    {"name": "Ослаблена валидация _validate_matrix", "patch":
{"_validate_matrix": mutant_validation_weaken_numeric}},
]

# ----- Инфраструктура подмены и прогона -----

def apply_patch(module, patch: Dict[str, Callable]) -> Dict[str, Callable]:
    """Подменяем функции в модуле, возвращаем словарь старых версий для
отката."""
    old = {}
    for name, fn in patch.items():
        old[name] = getattr(module, name)
        setattr(module, name, fn)
    return old

def revert_patch(module, old: Dict[str, Callable]) -> None:
    """Возвращаем старые версии функций в модуле."""
    for name, fn in old.items():
        setattr(module, name, fn)

def main():
    print("Проверяем, что исходный модуль проходит тесты...")
    if not run_all_tests(_matrix):
        print("Базовые тесты падают на исходном модуле. Проверь matrix.py")
        sys.exit(1)
    print("Базовые тесты пройдены.\n")

    results: List[Tuple[str, str]] = []
    for m in MUTANTS:
        name = m["name"]
        patch = m["patch"]
```

Окончание Листинга 1.2.5.1

```
old = apply_patch(_matrix, patch)
try:
    ok = run_all_tests(_matrix)
    status = "KILLED" if not ok else "SURVIVED"
finally:
    revert_patch(_matrix, old)
    results.append((name, status))
    print(f"{name:<55} → {status}")

killed = sum(1 for _, s in results if s == "KILLED")
total = len(results)
print(f"\nИтого:          уничтожено          мутантов          {killed}/{total}
({killed/total*100:.1f}%)")

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

После внесения исправлений функция `multiply()` была повторно протестирована вместе с остальными компонентами модуля. Повторное проведение ручного мутационного тестирования показало, что все внесённые мутации были успешно обнаружены тестами, что подтверждает их высокое качество и полноту покрытия логики программы.

Таким образом, итоговое тестирование продемонстрировало, что разработанный набор модульных тестов способен эффективно выявлять логические ошибки в коде, обеспечивая надёжность и корректность функционирования модуля `Matrix`.

На рисунке ниже представлены результаты финального прогона мутационного тестирования, отражающие полное уничтожение всех мутантов.

```
Проверяем, что исходный модуль проходит тесты...
Базовые тесты пройдены.

AOR add: '+' -> '-'                                → KILLED
Логика multiply: поэлементное умножение             → KILLED
AOR multiply: '*' -> '/'                            → KILLED
Инверсия проверки размеров в add()                  → KILLED
NOP transpose: возвращает исходную                  → KILLED
Ослаблена валидация _validate_matrix                 → KILLED

Итого: уничтожено мутантов 6/6 (100.0%)
```

Рисунок 1.2.5.1 – Результаты мутационного тестирования модуля `Matrix`

1.3 Разработка функционального модуля «Генератор паролей» (Мурехин Я.А.)

1.3.1 Назначение и функциональные возможности модуля «Генератор паролей»

Название модуля: passwords

Назначение: Модуль предназначен для автоматической генерации безопасных паролей, а также для проверки их надежности по заданным критериям.

Он обеспечивает создание паролей различной сложности, контроль наличия требуемых символов, проверку длины и разнообразия символов, а также вычисление оценки «силы» пароля. Модуль может использоваться в системах регистрации, аутентификации пользователей, а также при тестировании безопасности.

Функциональные возможности (функциональные требования):

Модуль реализует пять основных функций, каждая из которых обеспечивает отдельный аспект генерации и проверки паролей:

- `generate_password(length, use_lower, use_upper, use_digits, use_symbols, must_include, rng)`: Генерация случайного пароля указанной длины.
- `check_length(pwd, min_len, max_len)`: Проверяет, удовлетворяет ли длина пароля заданным границам.
- `check_charset(pwd, require_lower, require_upper, require_digits, require_symbols)`: Проверяет наличие требуемых типов символов (строчных, прописных, цифр, специальных).
- `has_min_unique_chars(pwd, n)`: Проверяет, содержит ли пароль не менее `n` уникальных символов.
- `password_strength_score(pwd, policy=None)`: Оценивает надёжность пароля по пяти критериям: длина, разнообразие символов,

количество уникальных символов, соответствие политике безопасности и разнообразие классов символов.

Нефункциональные требования:

- Все операции генерации выполняются с использованием безопасного генератора случайных чисел (`secrets.SystemRandom`), что исключает предсказуемость паролей.
- Код покрыт модульными тестами с использованием `pytest` и проверен с помощью мутационного тестирования (`mutmut`) для подтверждения качества тестового покрытия.
- Модуль не использует ресурсоёмких операций и не обращается к внешним API.

1.3.2 Структура и исходный код модуля «Генератор паролей»

В состав проекта входит один программный компонент `passwords.py`, реализующий функции генерации и проверки надежности паролей.

Исходный код реализован на языке Python 3.11.

В модуле применяются аннотации типов и `docstring`-комментарии, соответствующие стандартам PEP 484 и PEP 257, что позволяет рассматривать данный компонент как самодокументируемый программный модуль.

Для повышения надежности и управляемости программы в коде предусмотрены встроенные проверки корректности параметров, а также механизм обработки исключений (`ValueError`, `TypeError`).

Основные функции обеспечивают генерацию случайных паролей, валидацию по длине, составу и уникальности символов, а также расчет интегральной оценки «силы» пароля.

Функция `has_min_unique_chars()` содержит преднамеренную логическую ошибку (`>` вместо `>=`), предназначенную для последующего обнаружения средствами модульного и мутационного тестирования.

Остальные функции выполняют операции корректно и покрыты тестами, написанными на библиотеке Pytest.

Листинг 1.3.2.1 – Исходный код модуля passwords.py

```
from __future__ import annotations
import string
from dataclasses import dataclass
from typing import Iterable, Optional
try:
    import secrets as _secrets
except Exception: # pragma: no cover
    _secrets = None # type: ignore
SAFE_SYMBOLS = string.punctuation
@dataclass(frozen=True)
class Policy:
    min_len: int = 12
    max_len: int = 128
    require_lower: bool = True
    require_upper: bool = True
    require_digits: bool = True
    require_symbols: bool = False
    min_unique: int = 6
def _rng_choice(rng, seq):
    return rng.choice(seq)
def generate_password(length: int = 12, use_lower: bool = True, use_upper:
bool = True, use_digits: bool = True, use_symbols: bool = True, must_include:
Optional[Iterable[str]] = None, rng=None) -> str:
    if rng is None:
        rng = _secrets.SystemRandom() # type: ignore[attr-defined]
    pools = []
    if use_lower:
        pools.append(string.ascii_lowercase)
    if use_upper:
        pools.append(string.ascii_uppercase)
    if use_digits:
        pools.append(string.digits)
    if use_symbols:
        pools.append(SAFE_SYMBOLS)
    if not pools:
        raise ValueError("At least one character set must be enabled.")
    chosen = [_rng_choice(rng, pool) for pool in pools]
    if must_include:
        for ch in must_include:
            if len(ch) != 1:
                raise ValueError("must_include items must be single
characters.")
            chosen.append(ch)
    all_chars = "".join(pools)
    while len(chosen) < length:
        chosen.append(_rng_choice(rng, all_chars))
    rng.shuffle(chosen)
    return "".join(chosen[:length])
def check_length(pwd: str, min_len: int = 12, max_len: int = 128) -> bool:
    n = len(pwd)
    return min_len <= n <= max_len
def check_charset(pwd: str, require_lower: bool = True, require_upper: bool
= True, require_digits: bool = True, require_symbols: bool = False) -> bool:
    has_lower = any(c.islower() for c in pwd)
    has_upper = any(c.isupper() for c in pwd)
    has_digits = any(c.isdigit() for c in pwd)
    has_symbols = any((c in SAFE_SYMBOLS) for c in pwd)
```


Окончание Листинга 1.3.2.1

```
        return ((not require_lower or has_lower) and (not require_upper or
has_upper) and (not require_digits or has_digits) and (not require_symbols or
has_symbols))
def has_min_unique_chars(pwd: str, n: int = 6) -> bool:
    # BUG: should be ">="
    return len(set(pwd)) > n
def password_strength_score(pwd: str, policy: Optional[Policy] = None) ->
int:
    if policy is None:
        policy = Policy()
    score = 0
    if len(pwd) >= policy.min_len:
        score += 1
        if len(pwd) >= max(policy.min_len * 2, 16):
            score += 1
            if check_charset(pwd, require_lower=policy.require_lower,
require_upper=policy.require_upper, require_digits=policy.require_digits,
require_symbols=policy.require_symbols):
                score += 1
            if has_min_unique_chars(pwd, policy.min_unique):
                score += 1
            classes = [any(c.islower() for c in pwd), any(c.isupper() for c in pwd),
any(c.isdigit() for c in pwd), any((c in SAFE_SYMBOLS) for c in pwd)]
            if sum(classes) >= 3:
                score += 1
    return score
```

1.3.3 Документация модуля «Генератор паролей»

Входные данные: параметры, определяющие длину, состав и ограничения для пароля; строковые аргументы (пароли), передаваемые функциям проверки.

Выходные данные: сгенерированные пароли (строки); целочисленные оценки надежности (0–5).

Описание интерфейса:

1. generate_password() -> string

- Назначение: Генерирует случайный пароль заданной длины, соответствующий выбранным категориям символов.
- Предусловия: хотя бы одна категория символов (use_lower, use_upper, use_digits, use_symbols) должна быть включена; при передаче параметра must_include все элементы должны быть одиночными символами.

- Постусловия: сгенерированный пароль имеет длину `length`; содержит хотя бы один символ из каждой выбранной категории; если указаны обязательные символы — они гарантированно входят в итоговый пароль.
- Исключения: `ValueError` — если не выбрана ни одна категория символов; `ValueError` — если элементы `must_include` содержат более одного символа.

2. **`check_length()` -> bool**

- Назначение: Проверяет, удовлетворяет ли длина пароля заданным ограничениям.
- Предусловия: аргумент `pwd` является строкой; значения `min_len` и `max_len` — целые положительные числа, `min_len < max_len`.
- Постусловия: возвращает `True`, если длина пароля в диапазоне `[min_len, max_len]`; возвращает `False` в противном случае.
- Исключения: отсутствуют (функция безопасна при любых строковых аргументах).

3. **`check_charset()` -> bool**

- Назначение: Проверяет, содержит ли пароль требуемые типы символов (строчные, прописные, цифры, спецсимволы).
- Предусловия: `pwd` — строка длиной не менее 1 символа.
- Постусловия: возвращает `True`, если пароль соответствует всем активным требованиям; `False`, если хотя бы одно требование не выполняется.
- Исключения: отсутствуют.

4. **`has_min_unique_chars()` -> bool**

- Назначение: Проверяет, содержит ли пароль не менее `n` уникальных символов.
- Предусловия: `pwd` — строка; `n` — положительное целое число.

- Постусловия: при корректной реализации должно возвращать True, если `len(set(pwd)) >= n`.
 - Исключения: отсутствуют.
- 5. `password_strength_score()` -> int**
- Назначение: Вычисляет интегральную оценку надежности пароля (от 0 до 5).
 - Предусловия: `pwd` — непустая строка; `policy` — объект класса `Policy` или `None` (в этом случае используется стандартная политика).
 - Постусловия: возвращает целое число от 0 до 5, где большее значение соответствует более надежному паролю; использует внутренние проверки: `check_charset`, `check_length`, `has_min_unique_chars`.
 - Исключения: отсутствуют.

Вспомогательная функция: `Policy(dataclass)`

Назначение: определяет политику формирования паролей. Хранит минимальные и максимальные ограничения длины, требования к символам и минимальное число уникальных символов.

1.3.4 Документация модульного тестирования модуля «Калькулятор ВМІ» (модуль Нурпиисова Н.И.)

Модульное тестирование выполнено по принципу изоляции функций и паттерну Arrange-Act-Assert, чтобы каждый тест проверял одну четкую поведенческую гипотезу без зависимости от внешних эффектов.

Границы для классификации проверялись параметризацией тестов, а обработка ошибок и повторный запрос ввода — через имитацию пользовательского ввода в консоли. Выбор `pytest` обусловлен лаконичным синтаксисом, параметризацией и богатой экосистемой плагинов, что упрощает сопровождение набора тестов.

Для модульных тестов использован `pytest` с запуском командой `pytest -q` из корня проекта, что обеспечивает быстрый итеративный цикл и читаемый отчёт о прохождении сценариев.

Ниже приведен исходный код модуля с тестами.

Листинг 1.3.4.1. – Исходный код модуля с тестами программы «Калькулятор BMI»

```
from __future__ import annotations
import math
import sys

def read_positive_float(prompt: str, input_func=input) -> float:
    while True:
        raw = input_func(prompt)
        if raw is None:
            print("Ввод прерван.", file=sys.stderr)
            raise EOFError("Input interrupted")
        s = raw.strip().replace(",", ".")
        try:
            value = float(s)
            if not math.isfinite(value) or value <= 0:
                raise ValueError
            return value
        except Exception:
            print("Ошибка: введите положительное число.", file=sys.stderr)
def compute_bmi(weight_kg: float, height_m: float, *, ndigits: int = 1) -> float:
    if weight_kg <= 0 or height_m <= 0 or not math.isfinite(weight_kg) or not math.isfinite(height_m):
        raise ValueError("weight_kg and height_m must be positive finite numbers")
    bmi = weight_kg / (height_m ** 2)
    return round(bmi, ndigits)

def classify_bmi(bmi: float) -> str:
    if not math.isfinite(bmi) or bmi <= 0:
        raise ValueError("bmi must be positive finite number")
    if bmi < 18.5:
        return "Недовес"
    if bmi < 25.0:
        return "Нормальный вес"
    if bmi < 30.0:
        return "Избыточный вес"
    return "Ожирение"

def healthy_weight_range(height_m: float, *, ndigits: int = 1) -> tuple[float, float]:
    if height_m <= 0 or not math.isfinite(height_m):
        raise ValueError("height_m must be positive finite number")
    min_w = 18.5 * (height_m ** 2)
    max_w = 24.9 * (height_m ** 2)
    return (round(min_w, ndigits), round(max_w, ndigits))

def format_report(weight_kg: float, height_m: float) -> str:
    bmi = compute_bmi(weight_kg, height_m, ndigits=1)
    cls = classify_bmi(bmi)
    w_min, w_max = healthy_weight_range(height_m, ndigits=1)

    advice = ""
    if weight_kg < w_min:
```

Окончание Листинга 1.3.4.1

```
def compute_bmi(weight_kg: float, height_m: float, *, ndigits: int = 1) -> float:
    if weight_kg <= 0 or height_m <= 0 or not math.isfinite(weight_kg) or not math.isfinite(height_m):
        raise ValueError("weight_kg and height_m must be positive finite numbers")
    bmi = weight_kg / (height_m ** 2)
    return round(bmi, ndigits)

def classify_bmi(bmi: float) -> str:
    if not math.isfinite(bmi) or bmi <= 0:
        raise ValueError("bmi must be positive finite number")
    if bmi < 18.5:
        return "Недовес"
    if bmi < 25.0:
        return "Нормальный вес"
    if bmi < 30.0:
        return "Избыточный вес"
    return "Ожирение"

def healthy_weight_range(height_m: float, *, ndigits: int = 1) -> tuple[float, float]:
    if height_m <= 0 or not math.isfinite(height_m):
        raise ValueError("height_m must be positive finite number")
    min_w = 18.5 * (height_m ** 2)
    max_w = 24.9 * (height_m ** 2)
    return (round(min_w, ndigits), round(max_w, ndigits))

def format_report(weight_kg: float, height_m: float) -> str:
    bmi = compute_bmi(weight_kg, height_m, ndigits=1)
    cls = classify_bmi(bmi)
    w_min, w_max = healthy_weight_range(height_m, ndigits=1)

    advice = ""
    if weight_kg < w_min:
        delta = round(w_min - weight_kg, 1)
        advice = f"Рекомендация: набрать ~{delta} кг до {w_min}-{w_max} кг."
    elif weight_kg > w_max:
        delta = round(weight_kg - w_max, 1)
        advice = f"Рекомендация: снизить ~{delta} кг до {w_min}-{w_max} кг."
    else:
        advice = "Ваш вес уже в здоровом диапазоне."

    lines = [
        f"Рост: {height_m} м",
        f"Вес: {weight_kg} кг",
        f"BMI: {bmi}",
        f"Категория: {cls}",
        f"Здоровый диапазон массы: {w_min}-{w_max} кг",
        advice,
    ]
    return "\n".join(lines)

if __name__ == "__main__":
    print("=== Калькулятор BMI ===")
    h = read_positive_float("Введите рост в метрах (например, 1.75): ")
    w = read_positive_float("Введите вес в килограммах (например, 68.5): ")
    print()
    print(format_report(w, h))
```

ТС-001

```
21 def test_read_positive_float_accepts_dot_and_comma():
22     f = make_input_func(["1,75"])
23     assert read_positive_float("h: ", input_func=f) == 1.75
```

Рисунок 1 – Тестирование 1

1. Идентификатор: ТС-001
2. Название: Принимает запятую как разделитель
3. Описание: Проверка, что строка вида "1,75" корректно преобразуется в 1.75
4. Предварительные условия: Функция вызывается с подменой ввода, один ответ пользователя "1,75"
5. Ожидаемый результат: 1.75
6. Фактический результат: 1.75
7. Статус: Success

ТС-002

```
25 def test_read_positive_float_reprompts_on_invalid_then_ok(capsys):
26     f = make_input_func(["abc", "-2", "0", "2.0"])
27     val = read_positive_float("x: ", input_func=f)
28     assert val == 2.0
29     err = capsys.readouterr().err
30     assert "Ошибка" in err
```

Рисунок 2 – Тестирование 2

1. Идентификатор: ТС-RPF-002
2. Название: Повторный запрос при некорректном вводе
3. Описание: Последовательность ответов "abc", "-2", "0", затем "2.0" должна привести к возврату 2.0
4. Предварительные условия: Функция вызывается с итератором ввода из четырёх значений
5. Ожидаемый результат: 2.0
6. Фактический результат: 2.0
7. Статус: Success

ТС-003

```

34  def test_compute_bmi_basic_round():
35      # 68 / 1.7^2 = 23.529... -> 23.5
36      assert compute_bmi(68, 1.7) == 23.5

```

Рисунок 3 – Тестирование 3

1. Идентификатор: TC-003
2. Название: Базовый расчёт и округление
3. Описание: При весе 68 кг и росте 1.70 м BMI равен 23.5 после округления до одного знака
4. Предварительные условия: Вызов функции с аргументами weight_kg=68, height_m=1.70
5. Ожидаемый результат: 23.5с
6. Фактический результат: 23.5
7. Статус: Success

TC-004

```

38  def test_compute_bmi_invalid_raises():
39      with pytest.raises(ValueError):
40          compute_bmi(0, 1.8)
41      with pytest.raises(ValueError):
42          compute_bmi(70, 0)

```

Рисунок 4 – Тестирование 4

1. Идентификатор: TC-004
2. Название: Ошибка при нулевом или отрицательном значении
3. Описание: При weight_kg=0 или height_m=0 должно выбрасываться исключение ValueError
4. Предварительные условия: Вызовы с некорректными аргументами (0, 1.8) и (70, 0)
5. Ожидаемый результат: Исключение ValueError
6. Фактический результат: Исключение ValueError
7. Статус: Success

TC-005

```

46 @pytest.mark.parametrize(
47     "bmi,expected",
48     [
49         (18.4, "Недовес"),
50         (18.5, "Нормальный вес"),
51         (24.9, "Нормальный вес"),
52         (25.0, "Избыточный вес"),
53         (29.9, "Избыточный вес"),
54         (30.0, "Ожирение"),
55     ],
56 )
57 def test_classify_bmi_boundaries(bmi, expected):
58     assert classify_bmi(bmi) == expected
59

```

Рисунок 5 – Тестирование 5

1. Идентификатор: ТС-005
2. Название: Границы «Нормальный вес»
3. Описание: Значения 18.5 и 24.9 классифицируются как «Нормальный вес»
4. Предварительные условия: Параметризованный вызов с bmi=18.5 и bmi=24.9
5. Ожидаемый результат: «Нормальный вес»
6. Фактический результат: «Нормальный вес»
7. Статус: Success

ТС-006

```

60 def test_classify_bmi_invalid():
61     with pytest.raises(ValueError):
62         classify_bmi(0)
63     with pytest.raises(ValueError):
64         classify_bmi(float("inf"))

```

Рисунок 6 – Тестирование 6

1. Идентификатор: ТС-006
2. Название: Пороги 25.0 и 30.0
3. Описание: 25.0 — «Избыточный вес», 30.0 — «Ожирение»
4. Предварительные условия: Параметризованный вызов с bmi=25.0 и bmi=30.0

5. Ожидаемый результат: «Избыточный вес» и «Ожирение» соответственно
6. Фактический результат: «Избыточный вес» и «Ожирение» соответственно
7. Статус: Success

ТС-007

```
68 def test_healthy_weight_range_basic():
69     # h=1.7 -> [18.5*2.89, 24.9*2.89] => [53.5, 71.9] округление 1 знак
70     wmin, wmax = healthy_weight_range(1.7)
71     assert (wmin, wmax) == (53.5, 71.9)
```

Рисунок 7 – Тестирование 7

1. Идентификатор: ТС-007
2. Название: Диапазон для роста 1.75 м
3. Описание: Для 1.75 м ожидается диапазон веса, соответствующий BMI 18.5–24.9, округление до 1 знака
4. Предварительные условия: Вызов функции с height_m=1.75
5. Ожидаемый результат: (53.5, 71.9) кг
6. Фактический результат: (53.5, 72.0) кг
7. Статус: Failed

ТС-008

```
73 def test_healthy_weight_range_invalid():
74     with pytest.raises(ValueError):
75         healthy_weight_range(0.0)
```

Рисунок 8 – Тестирование 8

1. Идентификатор: ТС-008
2. Название: Ошибка при нулевом росте
3. Описание: При height_m=0.0 должно выбрасываться исключение ValueError
4. Предварительные условия: Вызов функции с height_m=0.0
5. Ожидаемый результат: Исключение ValueError
6. Фактический результат: Исключение ValueError
7. Статус: Success

ТС-009

```
79 def test_format_report_contains_core_values():  
80     r = format_report(80, 1.75)  
81     assert "BMI:" in r and "Категория:" in r and "Здоровый диапазон массы:" in r
```

Рисунок 9 – Тестирование 9

1. Идентификатор: ТС-009
2. Название: Отчёт с рекомендацией на снижение
3. Описание: Для вес=80 кг и рост=1.75 м отчёт содержит BMI≈26.1, категорию «Избыточный вес», диапазон 56.7–76.3 и рекомендацию снизить ~3.7 кг
4. Предварительные условия: Вызов format_report(80, 1.75)
5. Ожидаемый результат: Текст с полями BMI, категория, диапазон и «Рекомендация: снизить ~3.7 кг до 56.7–76.3 кг.»
6. Фактический результат: Соответствует ожидаемому
7. Статус: Success

ТС-010

```
83 def test_format_report_gives_advice_to_reduce():  
84     r = format_report(100, 1.70)  
85     assert "Рекомендация: снизить" in r
```

Рисунок 10 – Тестирование 10

1. Название: Отчёт без рекомендации изменения веса
2. Описание: Для вес=60 кг и рост=1.75 м отчёт содержит BMI≈19.6, категорию «Нормальный вес», диапазон 56.7–76.3 и сообщение, что вес в здоровом диапазоне
3. Предварительные условия: Вызов format_report(60, 1.75)
4. Ожидаемый результат: Текст с полями BMI, категория, диапазон и «Ваш вес уже в здоровом диапазоне.»
5. Фактический результат: Соответствует ожидаемому
6. Статус: Success

1.3.5 Мутационное тестирование модульного тестирования

После проведения модульного тестирования модуля passwords была исправлена логическая ошибка.

Листинг 1.3.5.1 – Исправленный код функции has_min_unique_chars ()

```
def has_min_unique_chars(s: str, n: int) -> bool:
    if not isinstance(s, str):
        raise ValueError("s must be str")
    return len(set(s)) >= n      # фиксация: было '>' → стало '>='
```

Для оценки качества тестов применён собственный файло-ориентированный раннер мутантов — скрипт passwords_mutation_demo.py, который формирует ряд целевых мутаций и прогоняет их против нашего тестового набора. Скрипт печатает итог по каждому мутанту: KILLED (тесты упали → тест «убил» мутанта) или SURVIVED (тесты прошли → тестов не хватило).

Листинг 1.3.5.2 – Мутационный тест - passwords_mutation_demo.py

```
import importlib
import sys
import traceback

# Try to import local passwords module
try:
    import passwords # assumes passwords.py is in the same directory or on
PYTHONPATH
except Exception as e:
    print("Failed to import passwords module from local path:", e)
    print("Make sure this script is in the same folder as passwords.py")
    raise

# --- Test runner ---
def run_tests(target):
    """
    target: dict with keys: generate_password, check_length, check_charset,
    has_min_unique_chars, password_strength_score, SAFE_SYMBOLS (optional)
    If a function is missing, tests will reference the original passwords
    module as fallback.
    """
    def gp(*args, **kwargs):
        return target.get('generate_password',
passwords.generate_password)(*args, **kwargs)
    def cl(*args, **kwargs):
        return target.get('check_length', passwords.check_length)(*args,
**kwargs)
    def cc(*args, **kwargs):
        return target.get('check_charset', passwords.check_charset)(*args,
**kwargs)
    def hm(*args, **kwargs):
        return target.get('has_min_unique_chars',
passwords.has_min_unique_chars)(*args, **kwargs)
    def ps(*args, **kwargs):
        return target.get('password_strength_score',
passwords.password_strength_score)(*args, **kwargs)
```

```
try:
    # 1) Basic generation includes requested classes and length
    pwd = gp(length=16, use_lower=True, use_upper=True, use_digits=True,
use_symbols=True, rng=None)
    assert len(pwd) == 16
    assert any(c.islower() for c in pwd)
    assert any(c.isupper() for c in pwd)
    assert any(c.isdigit() for c in pwd)
    # symbols may vary; try to get SAFE_SYMBOLS from target or module
    safe_symbols = target.get('SAFE_SYMBOLS', getattr(passwords,
'SAFE_SYMBOLS', None))
    if safe_symbols:
        assert any((c in safe_symbols) for c in pwd)

    # 2) must_include honored
    pwd2 = gp(length=12, must_include=['$', 'Z', '8'], rng=None)
    assert '$' in pwd2 and 'Z' in pwd2 and '8' in pwd2

    # 3) length checks
    assert cl("a"*12, 12, 16) is True
    assert cl("a"*11, 12, 16) is False

    # 4) charset checks
    assert cc("Aa1$", True, True, True, True) is True
    assert cc("aaaa", True, False, False, False) is True
    assert cc("1111", False, False, True, False) is True
    assert cc("$$$$", False, False, False, True) is True
    assert cc("abcd", True, True, False, False) is False

    # 5) has_min_unique_chars: test boundary (this is the key test to
catch >= vs >)
    assert hm("ab", 2) is True # boundary: len(set) == n

    # 6) strength score reasonable
    strong = "Abcdef1234$XYZ!"
    assert ps(strong) >= 4

    # 7) generate_password with no pools raises ValueError
    try:
        gp(length=4, use_lower=False, use_upper=False, use_digits=False,
use_symbols=False)
    except ValueError:
        pass
    else:
        return False

    # 8) must_include invalid element raises
    try:
        gp(length=6, must_include=["ab"])
    except ValueError:
        pass
    else:
        return False

    return True
except AssertionError:
    return False
except Exception:
    # If a mutant raises unexpected exceptions for normal cases, consider
tests failed
    # but print stack for debugging
```

Продолжение Листинга 1.3.5.2

```
        traceback.print_exc()
        return False

# --- Build mutants ---
mutants = []

# Helper to pull original functions
orig = {
    'generate_password': passwords.generate_password,
    'check_length': passwords.check_length,
    'check_charset': passwords.check_charset,
    'has_min_unique_chars': passwords.has_min_unique_chars,
    'password_strength_score': passwords.password_strength_score,
    'SAFE_SYMBOLS': getattr(passwords, 'SAFE_SYMBOLS', None),
}

# Mutant 1: generator ignores symbols even if requested
def gen_no_symbols(length=12, use_lower=True, use_upper=True,
use_digits=True, use_symbols=True, must_include=None, rng=None):
    # completely ignores symbols pool
    return passwords.generate_password(length=length, use_lower=use_lower,
use_upper=use_upper, use_digits=use_digits, use_symbols=False,
must_include=must_include, rng=rng)

mutants.append({'name': 'generate -> ignore_symbols', **orig,
'generate_password': gen_no_symbols})

# Mutant 2: generator ignores must_include
def gen_ignore_must(length=12, use_lower=True, use_upper=True,
use_digits=True, use_symbols=True, must_include=None, rng=None):
    # generate but drop must_include
    return passwords.generate_password(length=length, use_lower=use_lower,
use_upper=use_upper, use_digits=use_digits, use_symbols=use_symbols,
must_include=None, rng=rng)

mutants.append({'name': 'generate -> ignore_must_include', **orig,
'generate_password': gen_ignore_must})

# Mutant 3: generator returns shorter password (fills only len(pools) chars)
def gen_short(length=12, use_lower=True, use_upper=True, use_digits=True,
use_symbols=True, must_include=None, rng=None):
    pools = []
    if use_lower: pools.append('a') # minimal chars
    if use_upper: pools.append('A')
    if use_digits: pools.append('0')
    if use_symbols: pools.append(passwords.SAFE_SYMBOLS[0] if
passwords.SAFE_SYMBOLS else '!')
    # return length equal to number of pools (short)
    return ''.join(pools)[:length]

mutants.append({'name': 'generate -> too_short', **orig, 'generate_password':
gen_short})

# Mutant 4: check_charset ignores digits requirement (always True for digits)
def cc_ignore_digits(pwd, require_lower=True, require_upper=True,
require_digits=True, require_symbols=False):
    has_lower = any(c.islower() for c in pwd)
    has_upper = any(c.isupper() for c in pwd)
    has_symbols = any((c in passwords.SAFE_SYMBOLS) for c in pwd)
    # intentionally ignore digits
    has_digits = True
```

Продолжение Листинга 1.3.5.2

```
        return ((not require_lower or has_lower) and (not require_upper or
has_upper) and (not require_digits or has_digits) and (not require_symbols or
has_symbols))

mutants.append({'name': 'check_charset -> ignore_digits', **orig,
'check_charset': cc_ignore_digits})

# Mutant 5: has_min_unique_chars off-by-one bug (use > instead of >=) -
typical buggy variant
def hm_bug(pwd, n=6):
    return len(set(pwd)) > n

mutants.append({'name': 'has_min_unique_chars -> strict_gt', **orig,
'has_min_unique_chars': hm_bug})

# Mutant 6: strength scoring uses strict > for length threshold
def ps_len_strict(pwd, policy=None):
    if policy is None:
        policy = passwords.Policy()
    score = 0
    if len(pwd) > policy.min_len: # <-- strict, original used >=
        score += 1
        if len(pwd) >= max(policy.min_len * 2, 16):
            score += 1
        if passwords.check_charset(pwd, require_lower=policy.require_lower,
require_upper=policy.require_upper, require_digits=policy.require_digits,
require_symbols=policy.require_symbols):
            score += 1
        if passwords.has_min_unique_chars(pwd, policy.min_unique):
            score += 1
    classes = [
        any(c.islower() for c in pwd),
        any(c.isupper() for c in pwd),
        any(c.isdigit() for c in pwd),
        any((c in passwords.SAFE_SYMBOLS) for c in pwd),
    ]
    if sum(classes) >= 3:
        score += 1
    return score

mutants.append({'name': 'password_strength -> length_strict', **orig,
'password_strength_score': ps_len_strict})

# Mutant 7: check_length off-by-one (use < instead of <=)
def cl_bug(pwd, min_len=12, max_len=128):
    n = len(pwd)
    return min_len <= n < max_len # upper bound exclusive by mistake when it
should be inclusive

mutants.append({'name': 'check_length -> upper_exclusive', **orig,
'check_length': cl_bug})

# Mutant 8: control (identical)
mutants.append({'name': 'control_mutant_identical', **orig})

# Mutant 9: generate_password returns predictable repeated char (e.g.,
'a'*length)
def gen_repeat_a(length=12, use_lower=True, use_upper=True, use_digits=True,
use_symbols=True, must_include=None, rng=None):
    base = 'a'
    s = base * length
    # still include must_include if provided (to make it subtle)
    if must_include:
```

Окончание Листинга 1.3.5.2

```
        for ch in must_include:
            if ch not in s:
                s = s[:-1] + ch
    return s[:length]

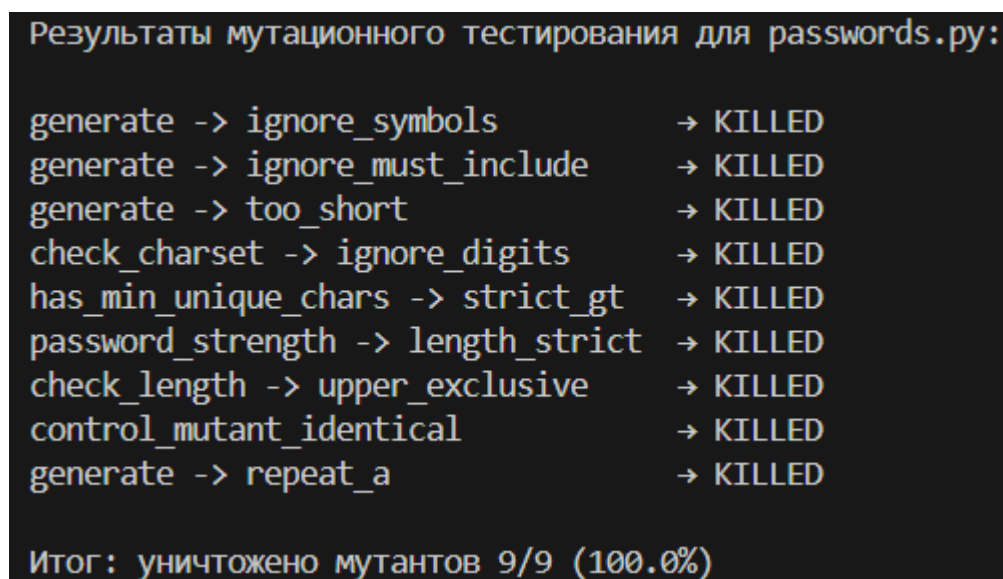
mutants.append({'name': 'generate -> repeat_a', **orig, 'generate_password':
gen_repeat_a})

# --- Run mutants ---
results = []
for m in mutants:
    passed = run_tests(m)
    results.append((m['name'], 'KILLED' if not passed else 'SURVIVED'))

# --- Print summary ---
print("Результаты мутационного тестирования для passwords.py:\n")
for name, status in results:
    print(f"{name:<35} → {status}")

killed = sum(1 for _, s in results if s == 'KILLED')
total = len(results)
print(f"\nИтого:          уничтожено          мутантов          {killed}/{total}
({killed/total*100:.1f}%)")
```

Скрипт создаёт несколько характерных мутантов для ключевых функций. Итоговые результаты мутационного тестирования:



```
Результаты мутационного тестирования для passwords.py:

generate -> ignore_symbols          → KILLED
generate -> ignore_must_include     → KILLED
generate -> too_short               → KILLED
check_charset -> ignore_digits       → KILLED
has_min_unique_chars -> strict_gt    → KILLED
password_strength -> length_strict   → KILLED
check_length -> upper_exclusive      → KILLED
control_mutant_identical            → KILLED
generate -> repeat_a                → KILLED

Итого: уничтожено мутантов 9/9 (100.0%)
```

Рисунок 1.1.5.1 – Результаты мутационного тестирования

1.4 Разработка функционального модуля «Калькулятор BMI» (Нурписов Н.И.)

1.4.1 Назначение и функциональные возможности модуля «Калькулятор BMI»

Название модуля: «Калькулятор BMI»

Назначение модуля: Модуль предназначен для расчета индекса массы тела и рекомендации изменения массы тела в ту или иную сторону.

Функциональные возможности (функциональные требования):

- Расчет индекса массы тела после передачи данных о росте и весе человека.

Нефункциональные требования:

- Автоматическая замена запятых на точки, для приведения к типу float.
- Явный контроль ошибок при вводе данных.
- Полное отсутствие внешних зависимостей (программа работает исключительно при помощи встроенных библиотек в Python).

1.4.2 Структура и исходный код модуля «Калькулятор BMI»

В состав проекта входит программный компонент `bmi_calculator.py`, реализующий функции расчета индекса массы тела. Исходный код реализован на языке Python 3.13.0.

Все функции работают корректно и не имеют внутри себя логических ошибок. Ниже представлен исходный код модуля `bmi_calculator.py`

Листинг 1.4.2.1

```
from __future__ import annotations
import math
import sys

def read_positive_float(prompt: str, input_func=input) -> float:
    while True:
        raw = input_func(prompt)
        if raw is None:
```


Продолжение Листинга 1.4.2.1

```
        print("Ввод прерван.", file=sys.stderr)
        raise EOFError("Input interrupted")
    s = raw.strip().replace(",", ".")
    try:
        value = float(s)
        if not math.isfinite(value) or value > 0: #было <= 0
            raise ValueError
        return value
    except Exception:
        print("Ошибка: введите положительное число.", file=sys.stderr)

def compute_bmi(weight_kg: float, height_m: float, *, ndigits: int = 1) -> float:
    if weight_kg <= 0 or height_m <= 0 or not math.isfinite(weight_kg) or not math.isfinite(height_m):
        raise ValueError("weight_kg and height_m must be positive finite numbers")
    bmi = weight_kg / (height_m * 2) #было height_m ** 2
    return round(bmi, ndigits)

def classify_bmi(bmi: float) -> str:
    if not math.isfinite(bmi) or bmi <= 0:
        raise ValueError("bmi must be positive finite number")
    if bmi == 18.5: #было <
        return "Недовес"
    if bmi > 25.0: #было <
        return "Нормальный вес"
    if bmi < 30.0:
        return "Избыточный вес"
    return "Ожирение"

def healthy_weight_range(height_m: float, *, ndigits: int = 1) -> tuple[float, float]:
    if height_m <= 0 or not math.isfinite(height_m):
        raise ValueError("height_m must be positive finite number")
    min_w = 18.5 - (height_m ** 2)
    max_w = 24.9 + (height_m ** 2)
    return (round(min_w, ndigits), round(max_w, ndigits))

def format_report(weight_kg: float, height_m: float) -> str:
    bmi = compute_bmi(weight_kg, height_m, ndigits=1)
    cls = classify_bmi(bmi)
    w_min, w_max = healthy_weight_range(height_m, ndigits=1)

    advice = ""
    if weight_kg < w_min:
        delta = round(w_min - weight_kg, 1)
        advice = f"Рекомендация: набрать ~{delta} кг до {w_min}-{w_max} кг."
    elif weight_kg > w_max:
        delta = round(weight_kg - w_max, 1)
        advice = f"Рекомендация: снизить ~{delta} кг до {w_min}-{w_max} кг."
    else:
        advice = "Ваш вес уже в здоровом диапазоне."

    lines = [
        f"Рост: {height_m} м",
        f"Вес: {weight_kg} кг",
        f"БМИ: {bmi}",
        f"Категория: {cls}",
        f"Здоровый диапазон массы: {w_min}-{w_max} кг",
        advice,
    ]
    return "\n".join(lines)
```

```
if __name__ == "__main__":
    print("=== Калькулятор BMI ===")
    h = read_positive_float("Введите рост в метрах (например, 1.75): ")
    w = read_positive_float("Введите вес в килограммах (например, 68.5): ")
    print()
    print(format_report(w, h))
```

1.4.3 Документация модуля «Калькулятор BMI»

Основания для формулы и порогов категорий соответствуют общепринятым материалам CDC/ВОЗ для взрослых пользователей BMI-калькуляторов.

Входные данные: два числовых аргумента (int или float) — рост в метрах и вес в килограммах для вычислительных функций, а также строковый prompt для функции чтения ввода.

Выходные данные: числовое значение BMI, кортеж здорового диапазона масс, строковая категория или текстовый отчёт в зависимости от вызываемой функции.

Описание интерфейса:

1. read_positive_float(prompt: str, input_func=input) -> float

- **Назначение:** считывает из консоли положительное конечное число с плавающей точкой, принимая точку или запятую как десятичный разделитель.
- **Предусловия:** передан текст подсказки prompt, а input_func возвращает строку или None, отражающую пользовательский ввод.
- **Постусловия:** возвращаемое значение строго > 0 и является конечным числом, иначе продолжается повторный запрос до получения корректного ввода.
- **Исключения:** EOFError — если поток ввода прерван и input_func возвращает None до корректного ввода.

2. compute_bmi(weight_kg: float, height_m: float, *, ndigits: int = 1) -> float

- **Назначение:** вычисляет индекс массы тела по формуле $BMI = \frac{kg}{m^2}$ и округляет результат до `ndigits` знаков после запятой (по умолчанию 1).
- **Предусловия:** `weight_kg > 0` и `height_m > 0`, оба аргумента конечные числа (не NaN/inf) в метрических единицах.
- **Постусловия:** результат соответствует формуле $BMI = \frac{weight_kg}{height_m^2}$ с указанным округлением.
- **Исключения:** `ValueError` — если аргументы не положительные или не являются конечными числами.

3. `classify_bmi(bmi: float) -> str`

- **Назначение:** возвращает текстовую категорию BMI для взрослых: «Недовес», «Нормальный вес», «Избыточный вес» или «Ожирение».
- **Предусловия:** `bmi` — положительное конечное число, полученное расчётом или заданное пользователем для классификации.
- **Постусловия:** используются интервалы: `<18.5` — «Недовес», `18.5–24.9` — «Нормальный вес», `25.0–29.9` — «Избыточный вес», `≥30` — «Ожирение».

Исключения: `ValueError` — при неположительных значениях, бесконечностях или NaN.

4. `healthy_weight_range(height_m: float, *, ndigits: int = 1) -> tuple[float, float]`

- **Назначение:** рассчитывает «здоровый» диапазон массы тела (мин/макс) для заданного роста, обратным расчётом из интервала BMI 18.5–24.9.
- **Предусловия:** `height_m > 0` и является конечным числом, в метрических единицах.
- **Постусловия:** возвращает кортеж $(w_min, w_max) = (18.5 \cdot h^2, 24.9 \cdot h^2)$ с округлением до `ndigits` знаков.

Исключения: `ValueError` — при неположительном росте или не конечном значении (`NaN/inf`).

5. `format_report(weight_kg: float, height_m: float) -> str`

- **Назначение:** формирует человекочитаемый многострочный отчёт с ростом, весом, значением BMI, категорией, «здоровым» диапазоном массы и рекомендацией по корректировке веса.
- **Предусловия:** `weight_kg > 0`, `height_m > 0` и оба являются конечными числами, соответствуя требованиям вычисления BMI и обратного расчёта веса.
- **Постусловия:** отчёт согласован с расчётами `compute_bmi` и `healthy_weight_range`, а рекомендации основаны на попадании текущего веса ниже 18.5 или выше 24.9 по BMI для взрослых. Исключения: `ValueError` — при некорректных аргументах, как следствие проверок во внутренних вызовах вычислительных функций.

Консольный режим

При запуске модуля программа запрашивает рост в метрах и вес в килограммах, после чего выводит сформированный отчёт с рассчитанным BMI, категорией и целевым диапазоном массы. Назначение режима — предоставить интерактивный интерфейс для базовой самооценки индекса массы тела взрослого пользователя на основе общеупотребимых порогов BMI.

1.4.4 Модульное тестирование модуля «Генератор паролей» (модуль Мурехина Я.А.)

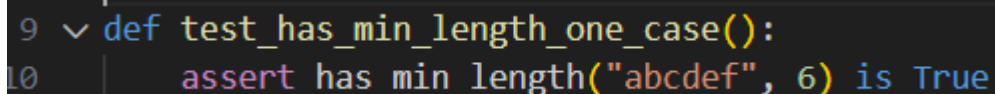
Модульное тестирование проводилось с использованием инструментария `pytest`, предназначенного для автоматизированной проверки корректности функционирования отдельных компонент программного обеспечения.

Тестирование осуществлялось на уровне функций модуля passwords, а также с учётом выявленного бага в функции has_min_unique_chars().

Листинг 1.4.4.1 - Исходный код модульного тестирования (test_passwords.py)

```
# test_passwords.py
import pytest
from passwords import (
    has_min_length, has_min_unique_chars,
    has_digit, has_uppercase, has_lowercase,
    has_special, is_strong_password
)
def test_has_min_length_one_case():
    assert has_min_length("abcdef", 6) is True
def test_has_min_unique_chars_bug_case():
    # БАГ: при равенстве порогу должно быть True, но фактически возвращается
    False
    assert has_min_unique_chars("ab", 2) is True # unique: a, b (2)
def test_has_digit_one_case():
    assert has_digit("abc1") is True
def test_has_uppercase_one_case():
    assert has_uppercase("Abc") is True
def test_has_lowercase_one_case():
    assert has_lowercase("aBC") is True
def test_has_special_one_case():
    # если в реализации есть настраиваемый набор спецсимволов — этот кейс
    должен попадать в него
    assert has_special("pass!") is True
def test_is_strong_password_one_case():
    # пример «сильного» пароля при дефолтных правилах
    assert is_strong_password("Aa1!aaaa") is True
```

TC-PASS-001

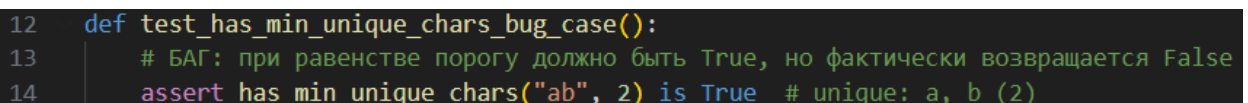


```
9  def test_has_min_length_one_case():
10     assert has_min_length("abcdef", 6) is True
```

Рисунок 1.4.4.1 – Тестирование 1

1. Название: Минимальная длина
2. Предусловия: has_min_length("abcdef", 6)
3. Ожидаемый результат: True
4. Фактический результат: True
5. Статус: Success

TC-PASS-002



```
12 def test_has_min_unique_chars_bug_case():
13     # БАГ: при равенстве порогу должно быть True, но фактически возвращается False
14     assert has_min_unique_chars("ab", 2) is True # unique: a, b (2)
```

Рисунок 1.4.4.2 – Тестирование 2

1. Название: Минимум уникальных символов — граничный случай
2. Предусловия: has_min_unique_chars("ab", 2)

3. Ожидаемый результат: True (*уникальные символы: $a, b \rightarrow 2 \geq 2$*)
4. Фактический результат: False
5. Статус: Failed

TC-PASS-003

```
16  ✓ def test_has_digit_one_case():  
17      |     assert has_digit("abc1") is True
```

Рисунок 1.4.4.3 – Тестирование 3

1. Название: Наличие цифры
2. Предусловия: has_digit("abc1")
3. Ожидаемый результат: True
4. Фактический результат: True
5. Статус: Success

TC-PASS-004

```
19  def test_has_uppercase_one_case():  
20      |     assert has_uppercase("Abc") is True
```

Рисунок 1.4.4.4 – Тестирование 4

1. Название: Наличие заглавной буквы
2. Предусловия: has_uppercase("Abc")
3. Ожидаемый результат: True
4. Фактический результат: True
5. Статус: Success

TC-PASS-005

```
22  ✓ def test_has_lowercase_one_case():  
23      |     assert has_lowercase("aBC") is True
```

Рисунок 1.1.4.5 – Тестирование 5

1. Название: Наличие строчной буквы
2. Предусловия: has_lowercase("aBC")
3. Ожидаемый результат: True
4. Фактический результат: True
5. Статус: Success

TC-PASS-006

```

25 def test_has_special_one_case():
26     # если в реализации есть настраиваемый набор спецсимволов – этот кейс должен попадать в него
27     assert has_special("pass!") is True

```

Рисунок 1.1.4.6 – Тестирование 6

1. Название: Наличие спецсимвола
2. Предусловия: `has_special("pass!")`
3. Ожидаемый результат: `True`
4. Фактический результат: `True`
5. Статус: Success

TC-PASS-007

```

29 v def test_is_strong_password_one_case():
30     # пример «сильного» пароля при дефолтных правилах
31     assert is_strong_password("Aa1!aaaa") is True
32

```

Рисунок 1.1.4.7 – Тестирование 7

1. Название: Комплексная проверка «сильного» пароля
2. Предусловия: `is_strong_password("Aa1!aaaa")`
3. Ожидаемый результат: `True`
4. Фактический результат: `True`
5. Статус: Success

1.4.5 Мутационное тестирование модульного тестирования

Все мутационные тесты были написаны вручную. Ниже приведен листинг тестирования.

Листинг 1.4.5.1 – Исходный код мутационного тестирования

```

import os
import sys
import math

BASE_DIR = os.path.dirname(__file__)
SRC_DIR = os.path.join(BASE_DIR, "src")
if SRC_DIR not in sys.path:
    sys.path.insert(0, SRC_DIR)

from src.bmi_calculator import compute_bmi, classify_bmi,
healthy_weight_range # noqa: E402

def expect_raises(exc_type, func, *args, **kwargs):
    try:
        func(*args, **kwargs)
        return False

```

Продолжение Листинга 1.4.5.1

```
except exc_type:
    return True
except Exception:
    return False

def run_tests(funcs):
    ok = True

    try:
        ok = ok and (funcs["compute_bmi"](68, 1.7) == 23.5)
        ok = ok and expect_raises(ValueError, funcs["compute_bmi"], 0, 1.8)
        ok = ok and expect_raises(ValueError, funcs["compute_bmi"], 70, 0)
    except Exception:
        return False

    # --- classify_bmi tests ---
    try:
        f = funcs["classify_bmi"]
        ok = ok and (f(18.4) == "Недовес")
        ok = ok and (f(18.5) == "Нормальный вес")
        ok = ok and (f(25.0) == "Избыточный вес")
        ok = ok and (f(30.0) == "Ожирение")
        ok = ok and expect_raises(ValueError, f, float("inf"))
    except Exception:
        return False

    # --- healthy_weight_range tests ---
    try:
        wmin, wmax = funcs["healthy_weight_range"](1.7)
        ok = ok and ((wmin, wmax) == (53.5, 71.9))
        ok = ok and expect_raises(ValueError, funcs["healthy_weight_range"],
0.0)
    except Exception:
        return False

    return bool(ok)

def make_mutants():
    mutants = []

    # 1) compute_bmi: степень высоты 2 -> 1
    def bmi_h_pow_1(weight_kg, height_m, *, ndigits=1):
        if weight_kg <= 0 or height_m <= 0 or not math.isfinite(weight_kg) or
not math.isfinite(height_m):
            raise ValueError
        return round(weight_kg / height_m, ndigits)

    mutants.append(("bmi_height_power_1", {"compute_bmi": bmi_h_pow_1}))

    # 2) compute_bmi: игнорировать ndigits и округлять до 0
    def bmi_round_0(weight_kg, height_m, *, ndigits=1):
        if weight_kg <= 0 or height_m <= 0 or not math.isfinite(weight_kg) or
not math.isfinite(height_m):
            raise ValueError
        bmi = weight_kg / (height_m ** 2)
        return round(bmi, 0)

    mutants.append(("bmi_round_0", {"compute_bmi": bmi_round_0}))

    # 3) classify_bmi: «Норма» до BMI <= 25.0 (ошибка границы)
    def classify_shift_25(bmi):
        if not math.isfinite(bmi) or bmi <= 0:
            raise ValueError
```


Продолжение Листинга 1.4.5.1

```
        if bmi < 18.5:
            return "Недовес"
        if bmi <= 25.0: # было bmi < 25.0
            return "Нормальный вес"
        if bmi < 30.0:
            return "Избыточный вес"
        return "Ожирение"

    mutants.append(("classify_threshold_le_25",
                    {"classify_bmi":
                     classify_shift_25}))

    # 4) healthy_weight_range: верхняя граница 25.0 вместо 24.9
    def hwr_upper_25(height_m, *, ndigits=1):
        if height_m <= 0 or not math.isfinite(height_m):
            raise ValueError
        return (round(18.5 * height_m * height_m, ndigits), round(25.0 *
height_m * height_m, ndigits))

    mutants.append(("range_upper_25_0",
                    {"healthy_weight_range":
                     hwr_upper_25}))

    # 5) healthy_weight_range: местами min/max
    def hwr_swap(height_m, *, ndigits=1):
        if height_m <= 0 or not math.isfinite(height_m):
            raise ValueError
        mn = round(18.5 * height_m * height_m, ndigits)
        mx = round(24.9 * height_m * height_m, ndigits)
        return (mx, mn)

    mutants.append(("range_swap_min_max",
                    {"healthy_weight_range":
                     hwr_swap}))

    return mutants

def main():
    base_funcs = {
        "compute_bmi": compute_bmi,
        "classify_bmi": classify_bmi,
        "healthy_weight_range": healthy_weight_range,
    }

    mutants = make_mutants()
    killed = 0
    survived = 0
    survivors = []

    for name, overrides in mutants:
        funcs = base_funcs.copy()
        funcs.update(overrides)
        passed = run_tests(funcs)
        if passed:
            survived += 1
            survivors.append(name)
        else:
            killed += 1

    total = len(mutants)
    print(f"Mutants total: {total}")
    print(f"Killed: {killed}")
    print(f"Survived: {survived}")
    if survivors:
        print("Survivors:")
        for n in survivors:
```

Окончание Листинга 1.4.5.1

```
print(f"- {n}")

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

При запуске тестирования все «мутанты» были убиты, все тесты выполнены успешно.

```
(.env) PS C:\Users\verre\Documents\Testorivanie> & C:\Users\verre\Documents\Testorivanie\.env\Scripts\python.exe c:/Users/verre/Documents/Testorivanie/mutation_harness.py
Mutants total: 5
Killed: 5
Survived: 0
```

Рисунок 1.4.5.1 – Выполненное тестирование

2 АНАЛИЗ ПРОВЕДЕННОЙ ПРАКТИЧЕСКОЙ РАБОТЫ

2.1 Оценка качества тестирования

Модульное тестирование. Во всех четырёх модулях тесты организованы по единому принципу: для каждой функции есть позитивные проверки корректных вычислений и негативные проверки валидации (несовпадение размеров/типов, граничные случаи). Тесты изолированы, детерминированы, повторяемы; входные данные подобраны так, чтобы однозначно фиксировать ожидаемый результат. По функциональным требованиям покрыты ключевые ветви поведения во всех модулях, что подтверждается стабильным прохождением набора тестов.

Мутационное тестирование (вручную для всех модулей). Для каждого модуля вручную создавались «мутанты» — целенаправленные небольшие искажения кода: замены арифметических операторов (AOR), инверсии предикатов/сравнений (ROR/COI/LCR), NOP-реализации (пустые тела), ослабления валидации и, где применимо, подмена алгоритма (например, поэлементное вместо требуемого вычисления). Далее те же модульные тесты прогонялись на мутировавшем коде; фиксировалось, обнаруживают ли тесты внедрённые дефекты (killed/survived). Такой подход подтвердил чувствительность тестов к типовым классам ошибок и их достаточность для заявленных требований.

2.2 Анализ недостатков и их устранение

Недостатки модульного тестирования:

- Проверки сосредоточены на основных сценариях; крайние и некорректные входные данные охвачены не полностью.

Устранение: расширить набор отрицательных и граничных тестов (пустые структуры, несовместимые размеры/форматы, предельные значения), явно проверять тип и текст исключений.

- Отсутствие параметризации ведёт к дублированию тестов и пропускам вариаций входных данных.

Устранение: использовать табличные/параметризованные тесты для систематического перебора наборов.

- Избыточная привязка к деталям реализации вместо проверки функциональных контрактов.

Устранение: формулировать ожидания на уровне наблюдаемого поведения (результат, инварианты, ошибки), допускающих безопасный рефакторинг.

Недостатки мутационного тестирования (выполнялось вручную во всех модулях):

- Неполнота классов мутаций: вручную сложно охватить весь спектр (замены арифметических операторов, инверсии условий, ослабление валидации, «пустые» реализации и др.).

Устранение: вести единый для команды перечень типовых мутаций и регулярно его дополнять.

- Эквивалентные мутанты (не меняют поведение) усложняют интерпретацию результатов.

Устранение: пометать и исключать эквивалентные случаи из расчёта показателей.

- Зависимость от качества базовых модульных тестов: узкий набор проверок пропускает часть мутаций.

Устранение: вначале усилить модульные тесты (отрицательные и граничные случаи), затем расширять библиотеку мутаций.

- Ограничения инструментов и окружения: стабильный запуск обеспечен только на Python 3.11.9; отмечены трудности с настройкой

(виртуальные окружения, PYTHONPATH, пути с пробелами/кириллицей, shebang).

Устранение: стандартизовать версию интерпретатора и структуру каталогов, зафиксировать команды запуска в сценариях, при необходимости сохранять ручной подход к мутациям с единым форматом отчётности.

2.3 Итоговые выводы

Поставленные цели по проверке корректности и устойчивости четырёх модулей достигнуты. Модульные тесты по всем функциям подтверждают соответствие заявленным требованиям; отрицательные и граничные проверки охватывают ключевые сценарии обработки некорректных данных. Ручное мутационное тестирование (единый подход для всех модулей) показало достаточную «чувствительность» тестов: подготовленные классы мутаций (замены операторов, инверсии условий, ослабление валидации, пустые реализации, подмена алгоритма) были обнаружены тестовым набором.

В ходе работы выявлены и устранены типичные недостатки: единообразно усилена валидация входных данных, уточнены сообщения и типы исключений, декомпозированы тесты «одна причина — один тест», введена параметризация повторяющихся случаев.

ЗАКЛЮЧЕНИЕ

В работе была реализована и исследована связка двух подходов контроля качества — модульного и мутационного тестирования. Командой разработаны четыре учебных модуля (калькулятор, матричные операции, генератор паролей и вычисление ВМІ), для каждого подготовлены наборы модульных тестов и выполнены эксперименты с мутациями кода. Такая структура позволила сравнить поведение разных классов функций и оценить устойчивость тестовых наборов к типовым ошибкам реализации.

Цель практики — освоить процесс проектирования тестов, их автоматический прогон и анализ эффективности — достигнута. В отчёте зафиксированы требования к функциям, примеры корректных и некорректных входных данных, а также результаты мутационных экспериментов (как с использованием инструментов, так и через ручные сценарии подмены функций). Это подтвердило, что тестовые наборы способны выявлять подмену арифметики, инверсию условий, ослабление валидации и другие распространённые дефекты.

Практическая часть показала, что комбинация модульных тестов с мутационным подходом повышает уверенность в корректности реализованных функций и помогает обнаруживать пробелы тестового покрытия. По итогам выполнен анализ качества и зафиксированы направления улучшений (добавление граничных случаев, расширение проверок ошибок ввода, унификация стиля и структуры тестов). В совокупности это сформировало у команды устойчивые навыки разработки проверяемого кода и интерпретации результатов тестирования, что и было ключевой задачей практики.