



МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«МИРЭА – РОССИЙСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»
РТУ МИРЭА

Институт Информационных технологий

Кафедра Математического обеспечения и стандартизации информационных
технологий

Отчет по практической работе

по дисциплине «Тестирование и верификация ПО»

Выполнили:

Студенты группы **ИКБО-11-22**

Андрусенко Л.Д.

Герасимов Л.Д.

Голованев Н.А.

Гришин А.В.

Проверил:

Мельников Д.А.

2024 г.

Оглавление

СОДЕРЖАНИЕ	Ошибка! Закладка не определена.
ЭТАП №1. СОЗДАНИЕ ПРОСТОГО МОДУЛЯ ПРОГРАММЫ.....	3
ЭТАП №2. РАЗРАБОТКА ДОКУМЕНТАЦИИ МОДУЛЯ ПРОГРАММЫ.....	7
1. Консольная игра «Крестики-Нолики»	7
1.1. Описание функционала.....	7
1.2. Описание функций.....	8
1.3. Логика работы программы.....	10
2. Приложение на Python для работы с массивами (списками).....	11
2.1. Описание функционала.....	11
2.2. Описание функций.....	11
3. Консольный калькулятор.....	14
3.1. Описание функционала.....	14
3.2. Описание функций.....	14
4. Приложение на Python для работы со строками.....	16
4.1. Описание функционала.....	16
4.2. Описание функций.....	16
ЭТАП №3. ТЕСТИРОВАНИЕ ПО.	20
ЭТАП №4. ИСПРАВЛЕНИЕ ОШИБКИ.....	23
Ошибка в игре «Крестики-нолики»	23
Ошибка в приложении для работы с массивами.....	24
Ошибка в консольном калькуляторе.....	25
Ошибка в приложении для работы со строками.....	26
ЭТАП №5. ИТОГОВОЕ ТЕСТИРОВАНИЕ.	28
Заключение:.....	29

ЭТАП №1. СОЗДАНИЕ ПРОСТОГО МОДУЛЯ ПРОГРАММЫ.

Каждый из участников команды создаёт свой простой модуль программы (с минимальным количеством функций равной 5 и при желании более) на любом языке программирования, с которым знакомы все члены команды. В одну из функций закладывается ошибка.

```
def draw_board(board): 3 usage
    """Функция рисует игровое поле."""
    for row in range(3):
        print(" | ".join(board[row]))
        if row < 2:
            print("-" * 5)

def check_winner(board, player): 8 usage
    """Функция проверяет наличие победителя."""
    win_conditions = [
        # Горизонтальные линии
        [(0, 0), (0, 1), (0, 2)],
        [(1, 0), (1, 1), (1, 2)],
        [(2, 0), (2, 1), (2, 2)],
        # Вертикальные линии
        [(0, 0), (1, 0), (2, 0)],
        [(0, 1), (1, 1), (2, 1)],
        [(0, 2), (1, 2), (2, 2)],
        # Диагонали
        [(0, 0), (1, 1), (2, 2)],
        [(0, 2), (1, 1), (2, 0)],
    ]

    for condition in win_conditions:
        if all(board[r][c] == player for r, c in condition):
            return True
    return False

def check_draw(board): 3 usage
    """Функция проверяет, заполнено ли всё поле (ничья)."""
    return all(cell in ['X', 'O'] for row in board for cell in row)

def get_player_input(board, player): 3 usage
    """Функция получает ввод игрока и проверяет корректность хода."""
    while True:
        try:
            move = int(input(f"Игрок {player}, выберите ячейку (1-9): ")) - 1
            if move < 0 or move >= 9:
                print("Некорректный ввод. Введите число от 1 до 9.")
                continue

            row, col = divmod(move, 3)
            if board[row][col] in ['X', 'O']:
                print("Эта ячейка уже занята. Попробуйте другую.")
            else:
                return row, col
        except ValueError:
            print("Некорректный ввод. Введите число.")
```

```
def play_game(): 1 usage
    """Функция запускает игру."""
    board = [['1', '2', '3'], ['4', '5', '6'], ['7', '8', '9']]
    current_player = 'X'

    while True:
        draw_board(board)
        row, col = get_player_input(board, current_player)
        board[row][col] = current_player

        if check_winner(board, current_player):
            draw_board(board)
            print(f"Игрок {current_player} победил!")
            break
        elif check_draw(board):
            draw_board(board)
            print("Ничья!")
            break

        current_player = 'O' if current_player == 'X' else 'X'

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

Рисунок 1 – Консольная игра «Крестики-нолики»

```
def invert_array(arr): 6 usages
# Инвертируем массив, итерируя его в обратном порядке
inverted = []
for i in range(len(arr) - 1, -1, -1):
    inverted.append(arr[i])
return inverted

def bubble_sort(arr): 7 usages
# Сортировка массива методом пузырька
n = len(arr)
sorted_arr = arr[:]
for i in range(n):
    for j in range(0, n - i - 1):
        if sorted_arr[j] > sorted_arr[j + 1]:
            sorted_arr[j], sorted_arr[j + 1] = sorted_arr[j + 1], sorted_arr[j]
    return sorted_arr

def find_min(arr): 5 usages
# Поиск минимального элемента массива
min_value = arr[0]
for i in range(1, len(arr)):
    if arr[i] < min_value:
        min_value = arr[i]
return min_value

def find_max(arr): 5 usages
# Поиск максимального элемента массива
max_value = arr[0]
for i in range(1, len(arr)):
    if arr[i] > max_value:
        max_value = arr[i]
return max_value

def sum_array(arr): 7 usages
# Сумма всех элементов массива
total = 0
for num in arr:
    total += num
return total

def average_array(arr): 6 usages
# Среднее арифметическое элементов массива
total = sum_array(arr)
avg = total / len(arr)
return avg

def count_occurrences(arr, value): 6 usages
# Подсчет количества вхождений элемента в массив
count = 0
for num in arr:
    if num == value:
        count += 1
return count
```

```
def main(): 1 usage
while True:
    print("\nВыберите операцию:")
    print("1. Инвертировать массив")
    print("2. Сортировка массива (пузырьковая сортировка)")
    print("3. Найти минимальный элемент массива")
    print("4. Найти максимальный элемент массива")
    print("5. Сумма всех элементов массива")
    print("6. Среднее арифметическое элементов массива")
    print("7. Подсчитать количество вхождений элемента в массиве")
    print("0. Выйти")

    choice = input("Ваш выбор: ")

    if choice == '0':
        break

    arr = list(map(int, input("Введите массив чисел через пробел: ").split()))

    if choice == '1':
        print("Инвертированный массив:", invert_array(arr))
    elif choice == '2':
        print("Отсортированный массив:", bubble_sort(arr))
    elif choice == '3':
        print("Минимальный элемент массива:", find_min(arr))
    elif choice == '4':
        print("Максимальный элемент массива:", find_max(arr))
    elif choice == '5':
        print("Сумма всех элементов массива:", sum_array(arr))
    elif choice == '6':
        print("Среднее арифметическое элементов массива:", average_array(arr))
    elif choice == '7':
        value = int(input("Введите элемент для подсчета: "))
        print(f"Элемент {value} встречается {count_occurrences(arr, value)} раз(а)")
    else:
        print("Неверный выбор. Попробуйте снова.")

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

Рисунок 2 – Приложение для работы с массивами

```
import operator

# Функция для вычисления выражений, введенных в одну строку
def calculate_expression(expr): 9 usages
    try:
        result = eval(expr)
        return result
    except Exception as e:
        return f"Error in expression: {str(e)}"

# Мappings операторов для пошаговых операций
operations = {
    '+': operator.add,
    '-': operator.sub,
    '*': operator.mul,
    '/': operator.truediv
}
```

```
def step_by_step_calculator(): 2 usages
    current_result = None
    current_operator = None

    while True:
        if current_result is None:
            try:
                current_result = float(input("Введите число: "))
            except ValueError:
                print("Ошибка: введите корректное число.")
                continue
        else:
            input_value = input("Введите знак операции (+, -, *, /) или число: ").strip()

            # Если это оператор
            if input_value in operations:
                current_operator = input_value
            else:
                try:
                    num = float(input_value)
                    if current_operator is None:
                        print("Ошибка: введите сначала операцию.")
                    else:
                        # Применяем последнюю операцию
                        try:
                            current_result = operations[current_operator](current_result, num)
                            print(f"Текущий результат: {current_result}")
                        except ZeroDivisionError:
                            print("Ошибка: деление на ноль!")
                            current_result = None
                except ValueError:
                    print("Ошибка: введите корректное число или оператор.")

if __name__ == "__main__":
    print("Добро пожаловать в калькулятор!")

    mode = input("Выберите режим работы (1 - строковый ввод, 2 - пошаговый ввод): ").strip()

    if mode == "1":
        expression = input("Введите выражение (например, '10 + 10 / 2'): ")
        result = calculate_expression(expression)
        print(f"Результат: {result}")
    elif mode == "2":
        step_by_step_calculator()
    else:
        print("Неверный выбор режима.")
```

Рисунок 3 – Консольный калькулятор

```

def to_uppercase(s): 7 usages
    result = ""
    for char in s:
        # Если символ является строчной буквой (a-z)
        if 'a' <= char <= 'z':
            # Преобразуем его в заглавную, изменив ASCII-код
            result += chr(ord(char) - 32)
        else:
            result += char
    return result

def to_lowercase(s): 7 usages
    result = ""
    for char in s:
        # Если символ является заглавной буквой (A-Z)
        if 'A' <= char <= 'Z':
            # Преобразуем его в строчную, изменив ASCII-код
            result += chr(ord(char) + 32)
        else:
            result += char
    return result

def invert_string(s): 6 usages
    result = ""
    # Итерируем строку в обратном порядке
    for i in range(len(s) - 1, -1, -1):
        result += s[i]
    return result

def count_vowels(s): 7 usages
    vowels = "aeiouAEIOU"
    count = 0
    for char in s:
        # Проверим является ли символ гласной
        if char in vowels:
            count += 1
    return count

def count_consonants(s): 7 usages
    consonants = "bcdfghijklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ"
    count = 0
    for char in s:
        # Проверим является ли символ согласной
        if char in consonants:
            count += 1
    return count

```

```

def count_consonants(s): 7 usages
    consonants = "bcdfghijklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ"
    count = 0
    for char in s:
        # Проверим является ли символ согласной
        if char in consonants:
            count += 1
    return count

def remove_spaces(s): 6 usages
    result = ""
    for char in s:
        # Добавим символ только если он не пробел
        if char != ' ':
            result += char
    return result

def main(): 1 usage
    while True:
        print("\nВыберите операцию:")
        print("1. Инвертировать строку")
        print("2. Перевести все буквы в заглавные")
        print("3. Перевести все буквы в строчные")
        print("4. Посчитать количество гласных")
        print("5. Посчитать количество согласных")
        print("6. Удалить пробелы")
        print("7. Сделать первую букву заглавной")
        print("0. Выйти")

        choice = input("Ваш выбор: ")

        if choice == '0':
            break

        string = input("Введите строку: ")

        if choice == '1':
            print("Инвертированная строка:", invert_string(string))
        elif choice == '2':
            print("Заглавные буквы:", to_uppercase(string))
        elif choice == '3':
            print("Строчные буквы:", to_lowercase(string))
        elif choice == '4':
            print("Количество гласных:", count_vowels(string))
        elif choice == '5':
            print("Количество согласных:", count_consonants(string))
        elif choice == '6':
            print("Строка без пробелов:", remove_spaces(string))
        elif choice == '7':
            print("Первая буква заглавная:", capitalize_first_letter(string))
        else:
            print("Неверный выбор. Попробуйте снова.")

if __name__ == "__main__":
    main()

```

Рисунок 4 – Приложение для работы со строками

ЭТАП №2. РАЗРАБОТКА ДОКУМЕНТАЦИИ МОДУЛЯ ПРОГРАММЫ.

Каждый участник пишет документацию по своему выбранному программному продукту и передает ее следующему участнику. В документации должно быть полное описание функционала приведенного кода, описание работы, выбранного для тестирования программного продукта или отдельно взятого кода.

Наименование проекта:

Андрусенко Л.Д.: Консольная игра “Крестики-Нолики”.

Гришин А.В.: Приложение на Python для работы с массивами (списками).

Герасимов Л.Д.: Консольный калькулятор.

Голованев Н.А.: Приложение на Python для работы со строками.

Основание для разработки:

Разработка учебного проекта в рамках курса по программированию на языке Python и тестированию модулей с использованием библиотеки unittest.

1. Консольная игра «Крестики-Нолики»

1.1. Описание функционала

Программа представляет собой консольную игру “Крестики-нолики”. Игра предназначена для двух игроков, которые ходят по очереди. Игроки могут выбрать ячейку на 3x3 поле, и их цель — разместить свои символы (“X” или “O”) так, чтобы выстроить три одинаковых символа подряд по горизонтали, вертикали или диагонали. Программа включает в себя следующие функции:

- **Рисование игрового поля.**
- **Получение хода игрока с проверкой на корректность ввода.**
- **Проверка победителя** после каждого хода.
- **Проверка на ничью**, когда все ячейки заполнены, но победителя нет.

- **Основной цикл игры**, где игроки поочередно делают ходы.

1.2. Описание функций

draw_board(board)

Функция отображает текущее состояние игрового поля в консоль. Поле представлено в виде сетки 3x3, где каждая клетка либо содержит номер свободной ячейки (если она не занята), либо символ “X” или “O”, если ячейка уже занята игроком.

- **Параметры:**

- board (список списков) — текущее состояние игрового поля, представленного в виде 3 строк и 3 столбцов.

- **Пример вывода:**

- X | 2 | O

- ----

- 4 | X | 6

- ----

- 7 | 8 | O

check_winner(board, player)

Функция проверяет, есть ли победитель на текущем этапе игры, определяя, выстроил ли игрок три одинаковых символа по горизонтали, вертикали или диагонали.

- **Параметры:**

- `board` (список списков) — текущее состояние игрового поля.
- `player` (строка) — символ игрока (“X” или “O”).

- **Возвращает:** `True`, если игрок выиграл, или `False`, если выигрышной комбинации нет.

- **Пример вызова:**

```
check_winner(board, 'X')
```

```
check_draw(board)
```

Функция проверяет, заполнено ли все игровое поле и не осталось ли пустых ячеек. Это нужно для определения ничьей.

- **Параметры:**

- `board` (список списков) — текущее состояние игрового поля.

- **Возвращает:** `True`, если все клетки заняты, и нет победителя, или `False`, если ещё остались свободные клетки.

- **Пример вызова:**

```
check_draw(board)
```

```
get_player_input(board, player)
```

Функция отвечает за ввод хода игрока. Пользователь должен ввести число от 1 до 9, соответствующее номеру ячейки, в которую он хочет поставить свой символ. Функция проверяет, корректен ли ввод, и не занята ли выбранная ячейка.

- **Параметры:**

- `board` (список списков) — текущее состояние игрового поля.
- `player` (строка) — символ текущего игрока (“X” или “O”).

- **Возвращает:** Кортеж (`row`, `col`) — координаты выбранной ячейки на игровом поле.

- **Пример вызова:**

```
get_player_input(board, 'O')
```

`play_game()`

Основная функция программы, которая запускает игровой процесс. В ней реализован основной игровой цикл, в котором игроки поочередно делают ходы до тех пор, пока не будет выявлен победитель или не наступит ничья. Каждый ход обновляет состояние игрового поля и проверяет, завершилась ли игра.

- **Параметры:** Нет.

- **Пример вызова:**

```
if __name__ == "__main__":
```

```
    play_game()
```

1.3. Логика работы программы

Инициализация игрового поля. Поле представляет собой список списков (3 строки и 3 столбца), где каждый элемент соответствует номеру ячейки от 1 до 9. Это позволяет игрокам выбирать ячейки, вводя соответствующие номера.

Основной игровой цикл:

В начале каждого хода программа выводит текущее состояние игрового поля с помощью функции `draw_board()`.

Затем текущий игрок вводит номер ячейки, куда хочет сделать ход, через функцию `get_player_input()`. Если ход некорректен (например, ячейка уже занята), программа повторно запрашивает ввод.

После каждого хода программа проверяет, есть ли победитель с помощью функции `check_winner()`. Если игрок выиграл, игра завершается и выводится соответствующее сообщение.

Если на поле больше нет свободных ячеек и победителя нет, игра завершается ничьей (функция `check_draw()`).

После каждого хода право хода передается следующему игроку (поочередно “X” и “O”).

Завершение игры. Игра продолжается до тех пор, пока не будет выявлен победитель или ничья.

2. Приложение на Python для работы с массивами (списками).

2.1. Описание функционала

Приложение предоставляет ряд функций для работы с массивами (списками) чисел.

Программа имеет консольный интерфейс, через который пользователь может выбирать требуемую операцию, вводить массив чисел, а также получать результат выбранной операции.

2.2. Описание функций

1. `invert_array(arr)`

Инвертирует (переворачивает) порядок элементов в массиве.

- **Параметры:** `arr` — список чисел, который нужно инвертировать.
- **Возвращает:** Новый список с элементами в обратном порядке.
- **Пример работы:**
- `invert_array([1, 2, 3, 4])` # Вернет `[4, 3, 2, 1]`

2. `bubble_sort(arr)`

Выполняет сортировку массива методом пузырька. Метод работает за счет многократного прохода по массиву и обмена соседних элементов местами, если они находятся в неправильном порядке.

- **Параметры:** `arr` — список чисел, который нужно отсортировать.
- **Возвращает:** Отсортированный по возрастанию список.
- **Пример работы:**
- `bubble_sort([3, 1, 4, 2])` # Вернет `[1, 2, 3, 4]`

3. `find_min(arr)`

Находит минимальный элемент в массиве.

- **Параметры:** `arr` — список чисел, в котором нужно найти минимальное значение.
- **Возвращает:** Минимальный элемент списка.
- **Пример работы:**
- `find_min([3, 1, 4, 2])` # Вернет `1`

4. `find_max(arr)`

Находит максимальный элемент в массиве.

- **Параметры:** `arr` — список чисел, в котором нужно найти максимальное значение.
- **Возвращает:** Максимальный элемент списка.

- **Пример работы:**
- `find_max([3, 1, 4, 2])` # Вернет 4

5. `sum_array(arr)`

Вычисляет сумму всех элементов массива.

- **Параметры:** `arr` — список чисел, сумму которых нужно вычислить.
- **Возвращает:** Сумму всех элементов списка.
- **Пример работы:**
- `sum_array([3, 1, 4, 2])` # Вернет 10

6. `average_array(arr)`

Вычисляет среднее арифметическое всех элементов массива.

- **Параметры:** `arr` — список чисел, для которых нужно вычислить среднее арифметическое.
- **Возвращает:** Среднее арифметическое всех элементов списка.
- **Пример работы:**
- `average_array([3, 1, 4, 2])` # Вернет 2.5

7. `count_occurrences(arr, value)`

Подсчитывает количество вхождений заданного элемента в массиве.

- **Параметры:**
 - `arr` — список чисел, в котором нужно найти количество вхождений.
 - `value` — элемент, количество вхождений которого нужно подсчитать.
- **Возвращает:** Число вхождений элемента в список.
- **Пример работы:**

- `count_occurrences([3, 1, 4, 2, 3], 3)` # Вернет 2

Основная программа (`main()`)

Главная функция программы предоставляет консольный интерфейс для пользователя, позволяя выбрать одну из предложенных операций.

3. Консольный калькулятор

3.1. Описание функционала

Данный код реализует калькулятор с двумя режимами работы:

1. **Строковый ввод:** пользователь вводит математическое выражение одной строкой, и программа вычисляет результат с помощью функции `eval()`.
2. **Пошаговый ввод:** пользователь вводит числа и операторы поочередно, и программа выполняет пошаговые арифметические операции.

Программа поддерживает следующие операции:

- Сложение (+)
- Вычитание (-)
- Умножение (*)
- Деление (/)

3.2. Описание функций

1. Функция `calculate_expression(expr)`:

- Эта функция принимает строковое математическое выражение, переданное пользователем, и вычисляет его результат с помощью функции `eval()`.
- Если при выполнении возникает ошибка (например, синтаксическая ошибка в выражении), она обрабатывается и возвращается сообщение об ошибке.

Пример работы функции:

`calculate_expression('10 + 10 / 2')` # Вернет 15.0

2. Маппинг операторов operations:

- Это словарь, в котором каждому арифметическому оператору (+, -, *, /) сопоставлена соответствующая функция из модуля `operator`, позволяющая выполнять соответствующую операцию.

3. Функция `step_by_step_calculator()`:

- Это функция, реализующая пошаговый ввод. Пользователь вводит сначала число, затем оператор, затем следующее число.
- После каждого ввода программа отображает промежуточный результат. Если была допущена ошибка (например, попытка деления на ноль или неверный ввод), она выводит сообщение об ошибке.
- Процесс продолжается до тех пор, пока пользователь не завершит выполнение программы вручную (прерывание через `Ctrl+C`).

Алгоритм работы функции:

- Сначала программа запрашивает у пользователя первое число.
- Затем ожидается ввод оператора или числа.
- Если вводится оператор, программа запрашивает следующее число и применяет к предыдущему результату соответствующую операцию.

- В случае попытки деления на ноль программа выводит сообщение об ошибке.

4. Основной блок программы:

- При запуске программы пользователь выбирает режим работы калькулятора:
 - **Режим 1** — строковый ввод: пользователь вводит выражение, и программа его вычисляет.
 - **Режим 2** — пошаговый ввод: пользователь вводит числа и операции пошагово, и программа вычисляет промежуточные результаты.
- Если пользователь вводит неправильный режим, программа выводит сообщение о неверном выборе.

4. Приложение на Python для работы со строками

4.1. Описание функционала

Приложение предоставляет несколько функций для работы со строками.

Программа также имеет текстовое меню, с помощью которого пользователь может выбирать требуемую операцию.

4.2. Описание функций

to_uppercase(s)

Преобразует все строчные буквы в строке в заглавные. Символы, которые не являются буквами (например, цифры и знаки препинания), остаются неизменными.

• Аргументы:

- *s (str)*: исходная строка.

- **Возвращаемое значение:** строка, в которой все строчные буквы заменены заглавными.

Пример:

```
to_uppercase("Hello") # Возвращает "HELLO"
```

to_lowercase(s)

Преобразует все заглавные буквы в строке в строчные. Не буквенные символы не изменяются.

- **Аргументы:**
 - s (str): исходная строка.
- **Возвращаемое значение:** строка, в которой все заглавные буквы заменены строчными.

Пример:

```
to_lowercase("HELLO") # Возвращает "hello"
```

invert_string(s)

Инвертирует строку, то есть меняет порядок символов в строке на обратный.

- **Аргументы:**
 - s (str): исходная строка.
- **Возвращаемое значение:** строка, символы которой расположены в обратном порядке.

Пример:

```
invert_string("Hello") # Возвращает "olleH"
```

count_vowels(s)

Подсчитывает количество гласных (как строчных, так и заглавных) в строке.

- **Аргументы:**

- s (str): исходная строка.

- **Возвращаемое значение:** количество гласных в строке.

Пример:

```
count_vowels("Hello") # Возвращает 2
```

count_consonants(s)

Подсчитывает количество согласных (как строчных, так и заглавных) в строке.

- **Аргументы:**

- s (str): исходная строка.

- **Возвращаемое значение:** количество согласных в строке.

Пример:

```
count_consonants("Hello") # Возвращает 3
```

remove_spaces(s)

Удаляет все пробелы из строки.

- **Аргументы:**

- s (str): исходная строка.

- **Возвращаемое значение:** строка без пробелов.

Пример:

```
remove_spaces("Hello World") # Возвращает "HelloWorld"
```

capitalize_first_letter(s)

Преобразует первую строчную букву строки в заглавную. Если первая буква уже заглавная или отсутствует, строка не изменяется.

- **Аргументы:**
 - `s (str)`: исходная строка.
- **Возвращаемое значение:** строка, где первая строчная буква сделана заглавной.

Пример:

```
capitalize_first_letter("hello") # Возвращает "Hello"
```

main()

Функция является основной точкой входа программы. Она предоставляет меню, позволяющее пользователю выбрать одну из предложенных операций и применить её к введённой строке.

ЭТАП №3. ТЕСТИРОВАНИЕ ПО.

Участники внутри своей группы производят обмен программными продуктами по кругу. Для полученного ПО пишутся Unit-тесты для каждой из функций в программе. Цель тестирования найти заложенную ошибку.

```
1 import builtins
2 import unittest
3
4 from tictac import *
5
6 class TestTicTacToe(unittest.TestCase):
7
8     def test_check_winner_horizontal(self):
9         """Тест на победу по горизонтали."""
10         board = [
11             ['X', 'X', 'X'],
12             ['O', 'O', '6'],
13             ['7', '8', '9']
14         ]
15         self.assertTrue(check_winner(board, player='X'))
16         self.assertFalse(check_winner(board, player='O'))
17
18     def test_check_winner_vertical(self):
19         """Тест на победу по вертикали."""
20         board = [
21             ['X', 'O', 'X'],
22             ['X', 'O', '6'],
23             ['X', '8', 'O']
24         ]
25         self.assertTrue(check_winner(board, player='X'))
26         self.assertFalse(check_winner(board, player='O'))
27
28     def test_check_winner_diagonal(self):
29         """Тест на победу по диагонали."""
30         board = [
31             ['X', 'O', 'X'],
32             ['O', 'X', '6'],
33             ['7', '8', 'X']
34         ]
35         self.assertTrue(check_winner(board, player='X'))
36         self.assertFalse(check_winner(board, player='O'))
37
38         board = [
39             ['O', 'X', 'X'],
40             ['4', 'O', '6'],
41             ['7', '8', 'O']
42         ]
43         self.assertTrue(check_winner(board, player='O'))
44
45     def test_check_draw(self):
46         """Тест на ничью."""
47         # Пое полностью заполнено, и нет победителя
48         board = [
49             ['X', 'O', 'X'],
50             ['X', 'O', 'O'],
51             ['O', 'X', 'X']
52         ]
53         self.assertTrue(check_draw(board))
54
55         # Пое не полностью заполнено
56         board = [
57             ['X', 'O', 'X'],
58             ['X', 'O', '6'],
59             ['O', 'X', 'X']
60         ]
61         self.assertFalse(check_draw(board))
62
63     def test_get_player_input(self):
64         """Тест на ввод игрока."""
65         board = [
66             ['X', 'O', '3'],
67             ['4', 'X', '6'],
68             ['7', '8', '9']
69         ]
70
71         # Проверяем правильный ход
72         input_values = ['3']
73
74         def mock_input(s):
75             return input_values.pop(0)
76
77         builtins.input = mock_input
78
79         row, col = get_player_input(board, player='O')
80         self.assertEqual(first=(row, col), second=(0, 2)) # Проверяем, что ячейка (0, 2) выбрана
81
82         # Проверяем занятую ячейку
83         board[0][2] = 'X'
84         input_values = ['2', '9'] # Вводим занятую ячейку и затем правильную
85         builtins.input = mock_input
86
87         row, col = get_player_input(board, player='O')
88         self.assertEqual(first=(row, col), second=(2, 2)) # Ожидаем (2, 2), так как '9' - корректный выбор
89
90
91 if __name__ == "__main__":
92     unittest.main()
93
```

Рисунок 5 - Unit-тесты для «Крестики-нолики»

```

1  import unittest
2
3  # Подключаем функции, которые будем тестировать
4  from arrays import invert_array, bubble_sort, find_min, find_max, sum_array, average_array, count_occurrences
5
6  class TestArrayFunctions(unittest.TestCase):
7
8      def test_invert_array(self):
9          self.assertEqual(invert_array([1, 2, 3, 4]), [4, 3, 2, 1])
10         self.assertEqual(invert_array([7, 8, 9]), [9, 8, 7])
11         self.assertEqual(invert_array([]), [])
12         self.assertEqual(invert_array([5]), [5])
13
14     def test_bubble_sort(self):
15         self.assertEqual(bubble_sort([4, 2, 7, 1]), [1, 2, 4, 7])
16         self.assertEqual(bubble_sort([1, 2, 3]), [1, 2, 3])
17         self.assertEqual(bubble_sort([5]), [5])
18         self.assertEqual(bubble_sort([]), [])
19         self.assertEqual(bubble_sort([9, 9, 9]), [9, 9, 9])
20
21     def test_find_min(self):
22         self.assertEqual(find_min([4, 2, 7, 1]), 1)
23         self.assertEqual(find_min([10, -5, 0, 12]), -5)
24         self.assertEqual(find_min([100]), 100)
25
26     def test_find_max(self):
27         self.assertEqual(find_max([4, 2, 7, 1]), 7)
28         self.assertEqual(find_max([10, -5, 0, 12]), 12)
29         self.assertEqual(find_max([100]), 100)
30
31     def test_sum_array(self):
32         self.assertEqual(sum_array([1, 2, 3, 4]), 10)
33         self.assertEqual(sum_array([0, 0, 0]), 0)
34         self.assertEqual(sum_array([-1, 1]), 0)
35         self.assertEqual(sum_array([]), 0)
36
37     def test_average_array(self):
38         self.assertEqual(average_array([1, 2, 3, 4]), 2.5)
39         self.assertEqual(average_array([10, 20, 30]), 20)
40         self.assertAlmostEqual(average_array([0, 0, 0]), 0)
41         self.assertAlmostEqual(average_array([-1, 1]), 0)
42
43     def test_count_occurrences(self):
44         self.assertEqual(count_occurrences([1, 2, 3, 4, 2], 2), 2)
45         self.assertEqual(count_occurrences([1, 1, 1], 1), 3)
46         self.assertEqual(count_occurrences([1, 2, 3], 4), 0)
47         self.assertEqual(count_occurrences([], 1), 0)
48
49     if __name__ == '__main__':
50         unittest.main()

```

Рисунок 6 - Unit-тесты для программы для работы с массивами

```

1  import unittest
2  from calc import calculate_expression, step_by_step_calculator
3
4
5  class TestCalculator(unittest.TestCase):
6      def test_calculate_expression_valid(self):
7          self.assertEqual(calculate_expression("10 + 5"), 15)
8          self.assertEqual(calculate_expression("10 / 2"), 5)
9          self.assertEqual(calculate_expression("2 * 3 + 1"), 7)
10         self.assertEqual(calculate_expression("10 - 3 * 2"), 4)
11
12     def test_calculate_expression_invalid(self):
13         self.assertTrue("Error in expression" in calculate_expression("10 / 0"))
14         self.assertTrue("Error in expression" in calculate_expression("10 +"))
15         self.assertTrue("Error in expression" in calculate_expression("abc"))
16
17
18     if __name__ == '__main__':
19         unittest.main()

```

Рисунок 7 - Unit-тесты для консольного калькулятора

```

1 import unittest
2
3 from strings import to_uppercase, to_lowercase, invert_string, count_vowels, count_consonants, remove_spaces
4
5
6 class TestStringFunctions(unittest.TestCase):
7
8     def test_to_uppercase(self):
9         self.assertEqual(to_uppercase('hello'), 'HELLO')
10        self.assertEqual(to_uppercase('Hello'), 'HELLO')
11        self.assertEqual(to_uppercase('HELLO'), 'HELLO')
12        self.assertEqual(to_uppercase('123abc!'), '123ABC!')
13        self.assertEqual(to_uppercase(''), '')
14
15     def test_to_lowercase(self):
16        self.assertEqual(to_lowercase('HELLO'), 'hello')
17        self.assertEqual(to_lowercase('Hello'), 'hello')
18        self.assertEqual(to_lowercase('hello'), 'hello')
19        self.assertEqual(to_lowercase('123ABC!'), '123abc!')
20        self.assertEqual(to_lowercase(''), '')
21
22     def test_invert_string(self):
23        self.assertEqual(invert_string('hello'), 'olleh')
24        self.assertEqual(invert_string('Hello'), 'olleH')
25        self.assertEqual(invert_string(''), '')
26        self.assertEqual(invert_string('A'), 'A')
27
28     def test_count_vowels(self):
29        self.assertEqual(count_vowels('hello'), 2)
30        self.assertEqual(count_vowels('HELLO'), 2)
31        self.assertEqual(count_vowels('bcdfg'), 0)
32        self.assertEqual(count_vowels('aeiouAEIOU'), 10)
33        self.assertEqual(count_vowels(''), 0)
34
35     def test_count_consonants(self):
36        self.assertEqual(count_consonants('hello'), 3)
37        self.assertEqual(count_consonants('HELLO'), 3)
38        self.assertEqual(count_consonants('aeiouAEIOU'), 0)
39        self.assertEqual(count_consonants('bcdfg'), 5)
40        self.assertEqual(count_consonants(''), 0)
41
42     def test_remove_spaces(self):
43        self.assertEqual(remove_spaces('hello world'), 'helloworld')
44        self.assertEqual(remove_spaces(' hello world '), 'helloworld')
45        self.assertEqual(remove_spaces(''), '')
46        self.assertEqual(remove_spaces('no_spaces_here'), 'no_spaces_here')
47
48 if __name__ == '__main__':
49     unittest.main()

```

Рисунок 8 - Unit-тесты для программы для работы со строками

ЭТАП №4. ИСПРАВЛЕНИЕ ОШИБКИ.

Ошибка в игре «Крестики-нолики»

Краткое описание ошибки: «неправильно определялся победитель».

Статус ошибки: открыта («Open»).

Категория ошибки: серьезная («Major»).

Тестовый случай: «Проверка алгоритма функционирования программы».

Описание ошибки:

1. Загрузить программу.
2. В поле ввода ввести строку «1».
3. Нажать кнопку «Пуск».
4. В поле ввода ввести строку «4».
5. Нажать кнопку «Пуск».
6. В поле ввода ввести строку «2».
7. Нажать кнопку «Пуск».
8. В поле ввода ввести строку «5».
9. Нажать кнопку «Пуск».
10. В поле ввода ввести строку «3».
11. Нажать кнопку «Пуск».
12. Полученный результат: undefined

Ожидаемый результат: X

```
n_conditions = [  
    # Горизонтальные линии  
    [(0, 0), (1, 1), (0, 2)],  
    [(1, 0), (1, 1), (1, 2)],  
]
```

```

min_conditions = [
    # Горизонтальные линии
    [(0, 0), (0, 1), (0, 2)],
    [(1, 0), (1, 1), (1, 2)],

```

Рисунок 9 - исправление ошибок

Ошибка в приложении для работы с массивами

Краткое описание ошибки: «не работал поиск мин. значения в массиве».

Статус ошибки: открыта («Open»).

Категория ошибки: серьезная («Major»).

Тестовый случай: «Проверка алгоритма функционирования программы».

Описание ошибки:

1. Загрузить программу.
2. В поле ввода ввести строку «4 2 7 1».
3. Нажать кнопку «Пуск».
4. Полученный результат: 7

Ожидаемый результат: 1

```

for i in range(1, len(arr)):
    if arr[i] < min_value:
        min_value = arr[i]

```



```
for i in range(1, len(arr)):  
    if arr[i] < min_value:  
        min_value = arr[i]  
return min_value
```

Рисунок 10 - исправление ошибок

Ошибка в консольном калькуляторе

Краткое описание ошибки: «вместо значения выражения возвращался аргумент».

Статус ошибки: открыта («Open»).

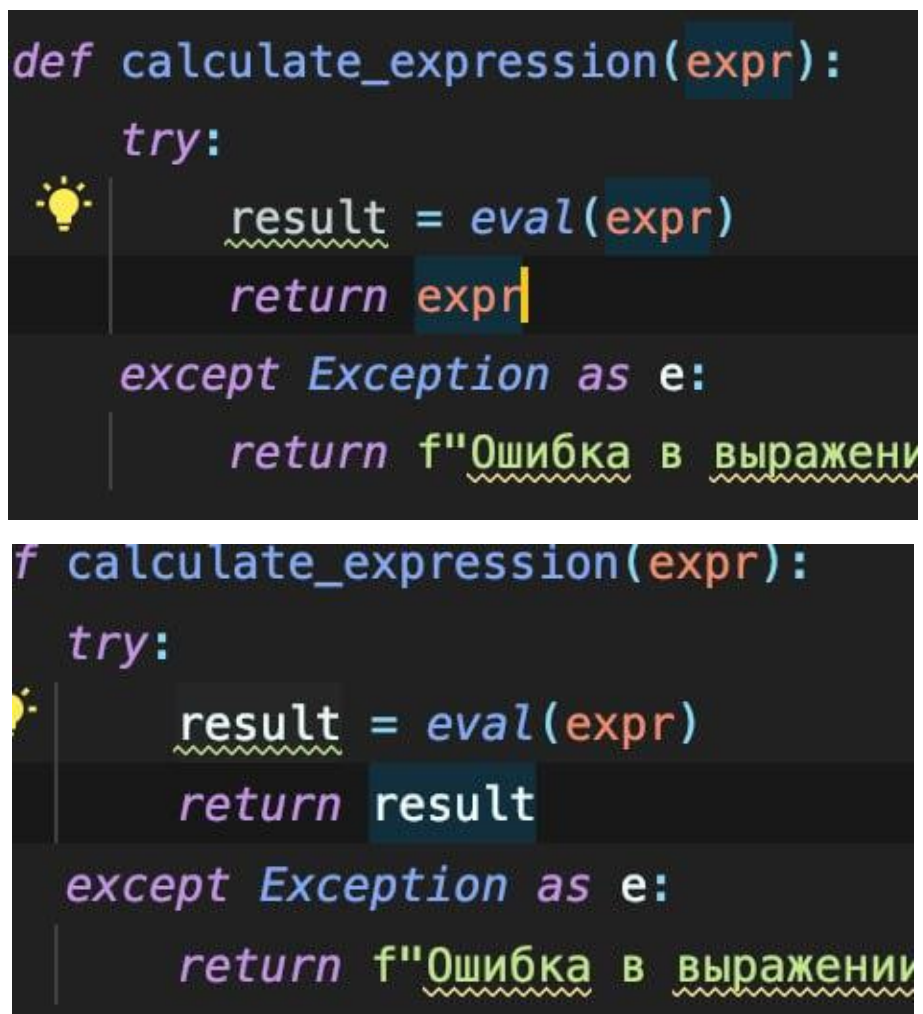
Категория ошибки: серьезная («Major»).

Тестовый случай: «Проверка алгоритма функционирования программы».

Описание ошибки:

1. Загрузить программу.
2. В поле ввода ввести строку «1».
3. Нажать кнопку «Пуск».
4. В поле ввода ввести строку «10 + 5».
5. Нажать кнопку «Пуск».
6. Полученный результат: 10 + 5

Ожидаемый результат: 15



```
def calculate_expression(expr):
    try:
        result = eval(expr)
        return expr
    except Exception as e:
        return f"Ошибка в выражении"

def calculate_expression(expr):
    try:
        result = eval(expr)
        return result
    except Exception as e:
        return f"Ошибка в выражении"
```

Рисунок 11 - исправление ошибок

Ошибка в приложении для работы со строками

Краткое описание ошибки: «не работало преобразование в заглавные буквы».

Статус ошибки: открыта («Open»).

Категория ошибки: серьезная («Major»).

Тестовый случай: «Проверка алгоритма функционирования программы».

Описание ошибки:

1. Загрузить программу.
2. В поле ввода ввести строку «2».
3. Нажать кнопку «Пуск».
4. В поле ввода ввести строку «abc».
5. Нажать кнопку «Пуск».

6. Полученный результат: undefined

Ожидаемый результат: ABC

```
if 'a' <= char <= 'z':  
    # Преобразуем его в заглавную, изменив ASCII  
    result += chr(ord(char) + 32)  
else:  
    result += char  
  
# Если символ является строчной буквой (a-z)  
if 'a' <= char <= 'z':  
    # Преобразуем его в заглавную, изменив ASCII-ко  
    result += chr(ord(char) - 32)  
else:  
    result += char
```

Рисунок 12 - исправление ошибок

ЭТАП №5. ИТОГОВОЕ ТЕСТИРОВАНИЕ.

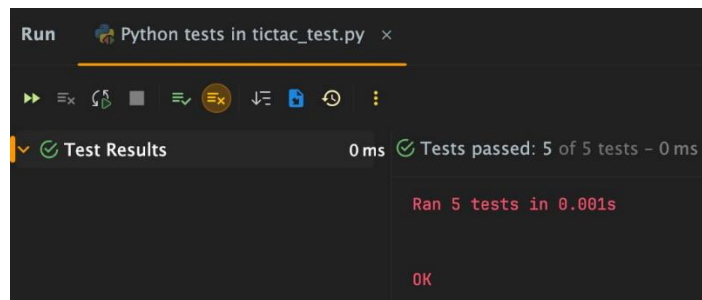


Рисунок 13 - Запуск Unit-тестов («Крестики-нолики»)

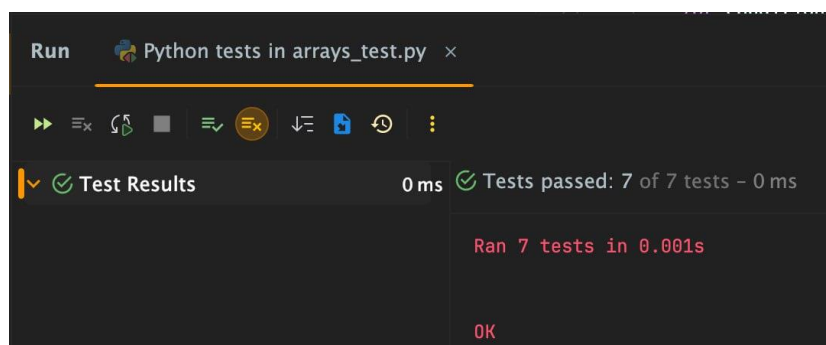


Рисунок 14 - Запуск Unit-тестов (приложение для работы с массивами)

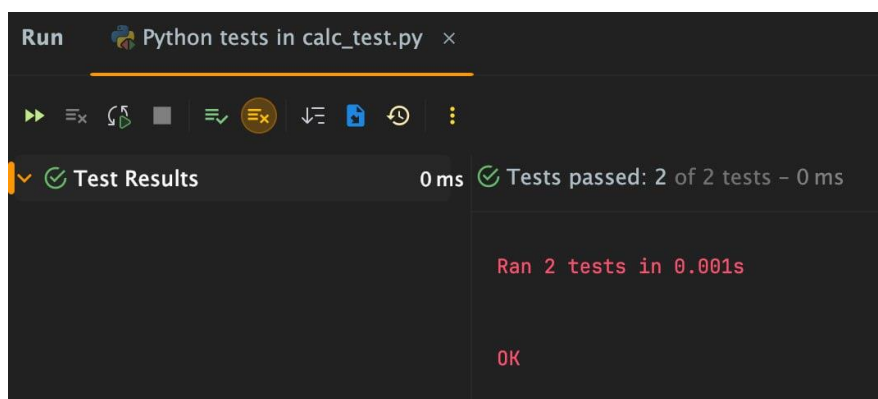


Рисунок 15 - Запуск Unit-тестов (консольный калькулятор)

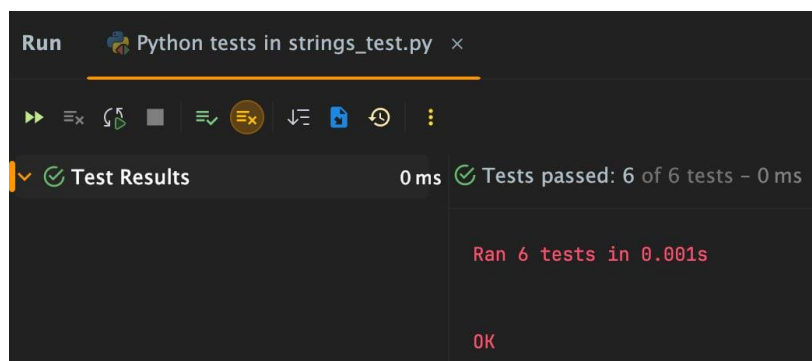


Рисунок 16 - Запуск Unit-тестов (приложение для работы со строками)

Заключение:

В ходе работы были освоены навыки написания модульных тестов, обнаружения и исправления ошибок в программном коде. Модульное тестирование показало свою эффективность в выявлении ошибок на ранних этапах разработки.