**Теоретическая часть**

1. Корректность

***Что такое корректность программы?***

- Определение: Программа считается корректной, если она выполняет поставленную задачу в соответствии с её спецификацией для всех допустимых входных данных. Иными словами, при выполнении программы её поведение соответствует заданным требованиям.

***Чем отличается корректность от надежности?***

- Корректность — это соответствие поведения программы спецификации.

- Надежность — это способность программы стабильно работать в течение времени и при различных условиях, минимизируя сбои и ошибки.

***Пример:***

- Надежная, но некорректная программа — программа, которая стабильно выполняет неправильную задачу (например, всегда возвращает неправильный результат, но делает это стабильно).

- Корректная, но ненадежная — программа, которая правильно решает задачу, но часто падает или сбоит при определенных условиях.

2. Эталоны (спецификации) корректности

***Что такое спецификация программы?***

- Определение: Документ или описание, в котором формулируются требования к поведению программы, её функциям, ограничениям и условиям выполнения. Она служит эталоном для оценки корректности.

***Почему она необходима?***

- Для определения, что именно считается правильным поведением программы.

- Для проверки соответствия реализации заявленным требованиям.

***Виды спецификаций:***

- Функциональные требования: описание функций, которые должна выполнять программа.

- Предусловия: условия, истинные перед вызовом функции.

- Постусловия: гарантии, которые программа должна обеспечить после выполнения функции при выполнении предусловий.

- Инварианты: условия, сохраняющиеся неизменными во время выполнения программы или внутри циклов.

***Для функции calculate\_positive\_average(numbers):***

|  |  |
| --- | --- |
| Тип спецификации | Формулировка |
| Предусловие | Numbers – непустой список чисел |
| Постусловие | Возвращаемое число – среднее арифметическое только положительных чисел из numbers, при этом если положительных чисел нет, возвращается или исключение |

3. Методы проверки корректности

***Тестирование***

- Суть: запуск программы с набором входных данных (тест-кейсами) и проверка соответствия результатов ожидаемым.

- Сильные стороны: легко реализуемо, помогает обнаружить ошибки.

- Слабые стороны: не гарантирует обнаружение всех ошибок, зависит от качества тестовых сценариев.

- Тест-кейс: конкретный набор входных данных и ожидаемый результат.

***Статический анализ***

- Суть: автоматическая проверка исходного кода без его выполнения.

- Ошибки, которые выявляет: потенциальные ошибки, уязвимости, нарушения стиля, возможные логические ошибки.

- Пример инструмента: ESLint, SonarQube.

- Пример ошибки: использование неинициализированной переменной.

***Формальные методы (доказательство корректности)***

- Идея: использование математических методов для доказательства, что программа соответствует спецификации.

- Инвариант цикла: условие, которое остается истинным на протяжении всего выполнения цикла. Он нужен для доказательства, что цикл завершится и выполнит свою задачу.

***Сравнение методов***

- Для небольшой функции наиболее подходит \*\*тестирование\*\* — быстро и просто.

- Для крупной системы — \*\*формальные методы\*\* или \*\*статический анализ\*\*, так как они позволяют обеспечить более высокий уровень уверенности в корректности.

**Практическая часть**

***Анализ кода функции calculate\_positive\_average***

1. Спецификация

***Предусловия***

- Что должно быть истинно перед вызовом функции?

- numbers — это последовательность (список, кортеж или другой итерируемый объект), содержащий элементы, которые можно складывать и делить.

- В идеале, элементы numbers — это числа (целые или с плавающей точкой). Однако, в текущем виде, функция не проверяет тип элементов.

***Постусловие***

- Что функция гарантирует вернуть при выполнении предусловия?

- Если в списке есть положительные числа, функция возвращает их среднее арифметическое.

- Если положительных чисел нет, функция возвращает 0.

***Инвариант цикла***

- Может ли быть сформулирован?

- Да. На каждой итерации:

- total — сумма всех положительных элементов, обработанных до текущей итерации.

- count — количество положительных элементов, обработанных до текущей итерации.

- Инвариант: после i-й итерации:

- total — сумма всех положительных элементов среди первых `i` элементов списка.

- count — число положительных элементов среди первых `i` элементов.

2. Ручное тестирование

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Ожидаемый результат | Фактический результат |
| 1 | [1, -2, 3, 4] | (1 + 3 + 4) / 3 = 2.666… | 2.666… |
| 2 | [] | 0 | 0 |
| 3 | [‘a’, 2, 3] | Ошибка или некорректное поведение | В текущем виде – ошибка выполнения, так как сложение с нечисловым типом невозможно |

3. Анализ корректности и ошибок

Почему здесь гарантированно не произойдет деления на ноль?

- Потому что деление происходит только при условии if count > 0. Если count равно 0, то переменная average устанавливается в 0, и деление не происходит.

- Это условие обеспечивает, что деление на ноль невозможно, так как оно выполняется только при count > 0.

***Обработка пустого списка***

- Да, функция корректно обрабатывает пустой список (`numbers = []`):

- count будет равно 0.

- В ветке else устанавливается average = 0.

— Это соответствует постусловию, так как при отсутствии положительных элементов возвращается 0.

***Возможная ошибка при нарушении предусловия***

- Если numbers не является списком или итерируемым объектом, например строкой, то:

- В цикле for num in numbers произойдет ошибка типа (TypeError), так как строка итерируется по символам, а не по числам.

- Для повышения надежности можно:

- Проверять тип входных данных и выбрасывать исключение с понятным сообщением, например:

python

if not isinstance(numbers, (list, tuple)):

raise TypeError("Параметр numbers должен быть списком или кортежем."

- Или проверять тип элементов внутри цикла и выбрасывать исключение при несоответствии.

4. Выводы

***Корректность функции***

- Да, функция в текущем виде корректна относительно своей очевидной спецификации — она вычисляет среднее положительных элементов и возвращает 0, если их нет.

- Обоснование: тесты показывают, что при корректных входных данных функция работает правильно, а при пустом списке возвращает 0, что соответствует постусловию.

***Метод проверки, если бы в строке A не было проверки count > 0***

- Наиболее эффективен был бы статический анализ или формальная верификация.

- Почему? — без проверки деления на ноль, возможна ошибка выполнения (ZeroDivisionError), которая легко пропустится при тестировании, если не предусмотрены такие случаи. Статический анализ способен обнаружить потенциальное деление на ноль, анализируя условие if count > 0.