**Отчет по практической работе №**5

**по дисциплине МДК 01.02 “Инструментальные средства разработки программного обеспечения”.**

Выполнил: студент

группы 319

Муртазин Руслан Шамилевич

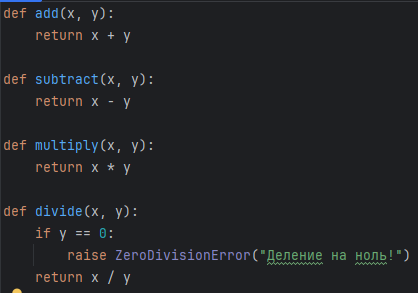
Дата 27.11.2024

**Цель работы:** изучить процесс отладки отдельных модулей программного проекта, выявить типичные ошибки и научиться эффективно использовать инструменты отладки для их устранения.

**Основная структура задания:**

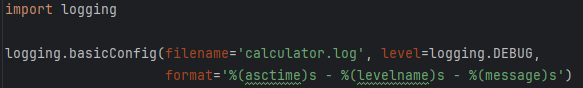
**Подготовка среды для отладки**

Проект был создан в PyCharm, указав директорию для сохранения. Затем был создан файл calculator.py, в котором был реализован базовый калькулятор с функциями сложения (add), вычитания (subtract), умножения (multiply) и деления (divide). Функция divide содержала проверку на деление на ноль, генерируя исключение ZeroDivisionError в случае попытки деления на ноль.

****

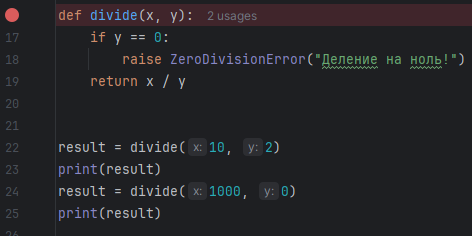
PyCharm использует встроенный отладчик для Python, поэтому дополнительная настройка не потребовалась.

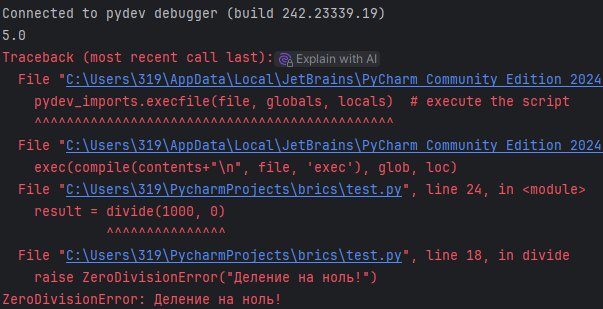
Импортируем модуль logging и настроим его. Это создаст лог-файл calculator.log с отладкой.

****

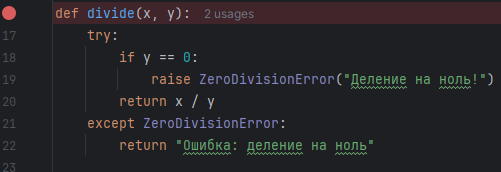
**Отладка логики модуля**

Возьмём функцию divide из калькулятора. Установим точку останова внутри функции. Запустим отладку и вызовем divide с разными аргументами:

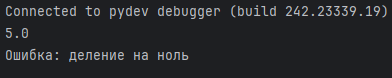




В данном случае, функция divide работает корректно, обрабатывает исключение ZeroDivisionError. Однако можно улучшить её, добавив обработку исключения, чтобы программа не падала, а возвращала какое-то значение, например None или специальное сообщение:

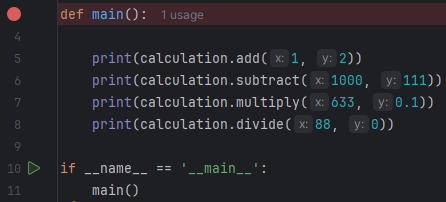


Теперь функция более устойчива к ошибкам.

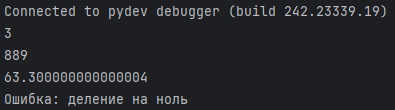


**Интеграционная отладка**

Запустим main.py. Проверим, что сложение, вычитание и умножение работают корректно, а деление на ноль обрабатывается без падения программы.



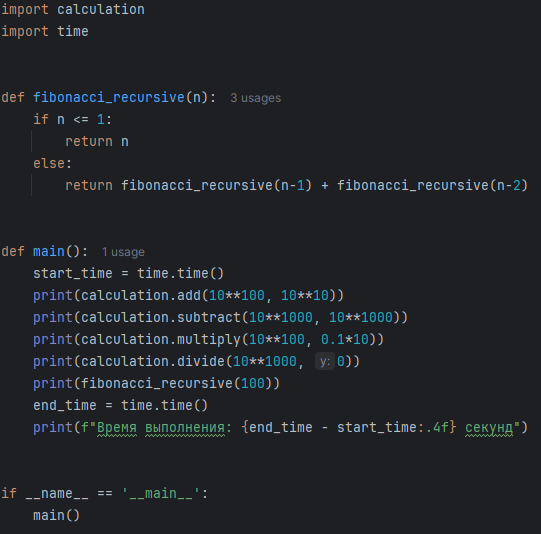
Запустим main.py в режиме отладки в PyCharm. Пошагово пройдёмся по коду, наблюдая, как данные передаются между функциями и как обрабатываются результаты. Убедимся, что нет неожиданных взаимодействий между функциями.



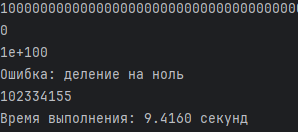
Если бы во время пошаговой отладки обнаружены ошибки (например, неправильная обработка ошибок, несоответствие типов данных), мы внесли бы необходимые исправления в код, обновив документацию к модулю calculator.py, отразив изменения в поведении функций и обработке исключений. Этот процесс гарантирует, что модули работают корректно в контексте всего проекта и не конфликтуют с другими частями системы.

**Отладка производительности и использования памяти**

Реализуем новую функцию и дополним ей код

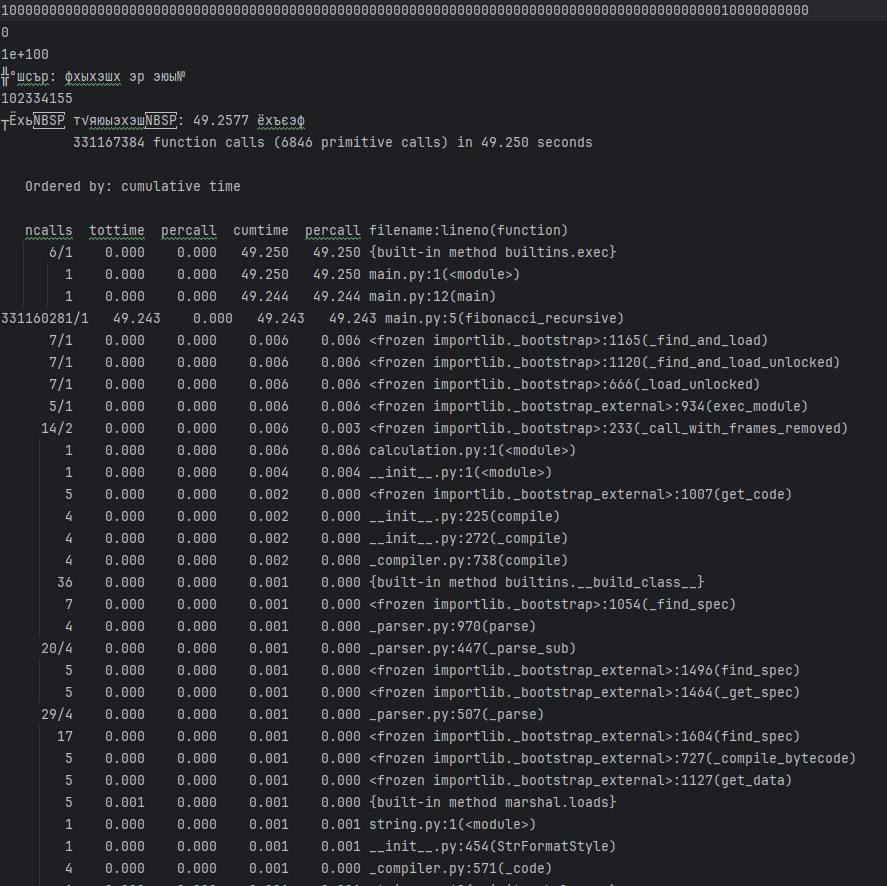


Видим, что код выполняется значительное количество времени.



Запустим скрипт из командной строки с помощью cProfile.

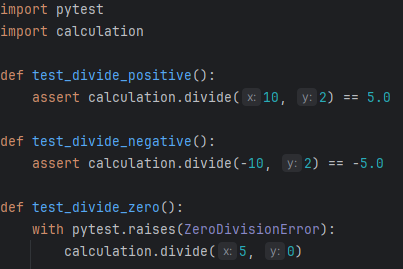




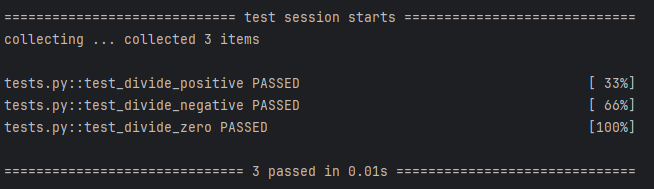
В случае с простым калькулятором, профилирование может не показать значительных результатов. Эти инструменты более полезны для анализа больших и сложных программ. Если необходимо оценить общее время выполнения кода, модуля time будет достаточно.

**Автоматизированное тестирование и отладка**

Сначала убедимся, что pytest установлен (pip install pytest). Затем создадим файл test\_calculator.py рядом с calculator.py.

****

Эти тесты проверяют функцию divide с разными входными данными.

****

Если какие-либо тесты падают, нужно искать и исправлять ошибки в коде. Этот цикл “исправление-тестирование” повторяется до тех пор, пока все тесты не будут пройдены успешно.

Этот подход гарантирует, что изменения в коде не нарушают существующий функционал, делая разработку более надёжной. Автоматизированное тестирование особенно важно при частых обновлениях и изменениях кода.

**Вывод работы:** в ходе практической работы я изучил процесс отладки отдельных модулей программного проекта, выявил типичные ошибки и научился эффективно использовать инструменты отладки для их устранения.