Лабораторная работа №7

Выполнила Чикомазова Алиса 504497 P-3122

Архитектура проекта:

logger.py: Содержит параметризованный декоратор @logger, его настройку для работы с файлами и модулем logging, а также реализацию и демонстрацию функций get_currencies и solve_quadratic с логированием.

tests_.py: Содержит unittest для проверки как чистой бизнес-логики (get_currencies), так и функциональности самого декоратора @logger.

currencies.log: Автоматически генерируемый файл, используемый для логирования вызовов get_currencies.

currencies: Содержит реализацию и демонстрацию функции get_currencies без логирования

1. Параметризуемый декоратор logger

Декоратор @logger разработан как универсальное средство для отслеживания хода выполнения любой функции.

Цели логирования: Декоратор является параметризованным, принимая аргумент handle. Реализация позволяет направлять логи в:

Файловые объекты (TextIOWrapper, io.StringIO).

Объекты logging.Logger (для интеграции со стандартным модулем логирования Python).

Логирование событий: Декоратор фиксирует три ключевых этапа:

Старт функции с указанием всех переданных аргументов.

Успешное завершение с фиксацией возвращаемого результата.

Ошибка (Exception) с указанием типа и сообщения об исключении.

Обработка исключений: При возникновении ошибки декоратор сначала записывает сообщение об ERROR, а затем повторно выбрасывает исключение, сохраняя тем самым логику работы вызывающего кода.

Сохранение метаданных: Использование @functools.wraps(func) обеспечивает сохранение оригинального имени и документации декорируемой функции.

2. Функция получения курсов валют

Функция get_currencies предназначена для запроса актуальных курсов валют с API Центробанка РФ.

Применение декоратора: Функция декорирована с использованием предварительно настроенного файлового логгера: @logger(handle=file_logger). Это обеспечивает запись всех вызовов, результатов и ошибок в файл currencies.log.

Надежность и обработка ошибок:

Сетевые ошибки: Перехватываются requests.ConnectionError, requests.Timeout и requests.HTTPError с генерацией ConnectionError.

Формат данных: Выполняется проверка на корректность ответа JSON (json.JSONDecodeError) и наличие ключевого слова "Valute" (KeyError).

Валидация валют: Проверяется существование запрошенных кодов валют, а также корректность типа данных самого курса (должен быть int или float).

3. Функция solve_quadratic служит демонстрацией логирования в консоль с помощью модуля logging.

Настройка логгера: Используется отдельный логгер quad_logger, настроенный на вывод сообщений на уровне INFO в консоль.

Применение декоратора: Функция декорирована как @logger(handle=quad_logger).

Специальная логика: Функция включает дополнительные проверки и логирование:

Обработка случаев a=0 (линейное уравнение) или a=0, b=0 (некорректное уравнение).

В случае отрицательного дискриминанта функция логирует WARNING (используя встроенный метод quad_logger.warning) и возвращает None.

4. Тестирование (tests_.py)

Тестирование включает проверку как бизнес-логики, так и функциональности декоратора.

А. Тесты бизнес-логики get_currencies

Тесты полностью изолируют функцию от внешних зависимостей (API ЦБ) с помощью мокирования (unittest.mock.patch):

Проверяется успешный возврат ожидаемых курсов.

Проверяется корректный выброс исключений: ConnectionError (при ошибках сети/HTTP), ValueError (некорректный JSON), KeyError (отсутствие ключа Valute или запрошенной валюты), и TypeError (курс не является числом).

Б. Тесты декоратора logger

Тестирование с io.StringIO: Используется для имитации файлового вывода. Проверяется корректность формата логов при успехе (INFO: Started, INFO: Finished) и при ошибке (ERROR: Failed).

Проброс исключений: Тесты подтверждают, что после записи лога ERROR исключение гарантированно пробрасывается для дальнейшей обработки.

Тестирование с logging.Logger: Используется MagicMock для проверки, что декоратор вызывает правильные методы (.info() или .error()) переданного ему объекта logging.Logger.