**Задача 1: Декоратор для логирования**

1. Создайте декоратор log\_decorator, который будет записывать в файл log.txt информацию о вызове функции:
   * Время вызова функции.
   * Имя функции.
   * Аргументы, с которыми функция была вызвана.
   * Время завершения работы функции.
   * Время выполнения функции (в секундах).
2. Примените этот декоратор к функции calculate, которая принимает два числа и оператор (+, -, \*, /), и возвращает результат выполнения операции.
3. После вызова функции calculate, в файле log.txt должна появиться запись:

[2025-03-20 12:00:00] Функция 'calculate' вызвана с аргументами: (10, 5, '+')

[2025-03-20 12:00:01] Функция 'calculate' завершена. Время выполнения: 1.0 сек.

@log\_decorator

def calculate(a, b, operation):

if operation == '+':

return a + b

elif operation == '-':

return a - b

elif operation == '\*':

return a \* b

elif operation == '/':

return a / b

else:

raise ValueError("Неподдерживаемая операция")

calculate(10, 5, '+')

**Задача 2: Декоратор для ограничения вызовов**

1. Создайте декоратор rate\_limit, который будет ограничивать количество вызовов функции в течение определенного времени (например, не более 3 вызовов в минуту).
2. Если лимит превышен, декоратор должен выводить сообщение: "Превышен лимит вызовов. Попробуйте позже."

Пример использования:

@rate\_limit(max\_calls=3, period=60)

def send\_message(message):

print(f"Сообщение отправлено: {message}")

for \_ in range(5):

send\_message("Привет!")

1. После выполнения кода, первые три вызова функции send\_message должны сработать нормально, а на четвертый и пятый вызовы должно быть выведено сообщение о превышении лимита.

**Задача 3: Декоратор для кэширования**

1. Создайте декоратор cache\_decorator, который будет кэшировать (сохранять в словарь параметры и результат функции) результаты выполнения функции.
2. Если функция вызывается с теми же аргументами, что и ранее, декоратор должен возвращать результат из кэша, не вызывая функцию повторно.

Пример использования:

@cache\_decorator

def fibonacci(n):

if n <= 1:

return n

return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(10))

1. После первого вызова fibonacci(10) результат должен быть закэширован, и при повторном вызове с тем же аргументом функция не должна вычисляться заново.