**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

Факультет **Инфокоммуникационных технологий**

Образовательная программа **Мобильные и сетевые технологии**

Направление подготовки(специальность) **09.03.03 Прикладная информатика**

**ЛАБОРАТОРНАЯ РАБОТА №1**

**По дисциплине «Программирование»**

**Тема: «Декораторы и асинхронное программирование»**

**Выполнил** Борисюк Е. Г. КЗ140.

**Проверил** Терещенко В.

**Дата** 31.05.2025

**Санкт-Петербург 2025**

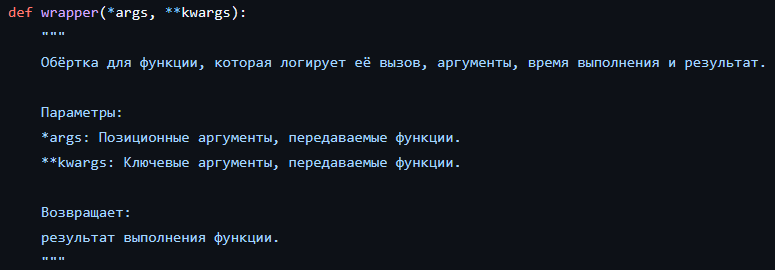
**Цель работы**

Изучить принцип работы декораторов и потренироваться в написании собственных декораторов

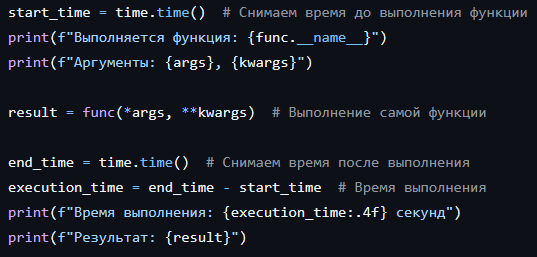
**Ход работы**

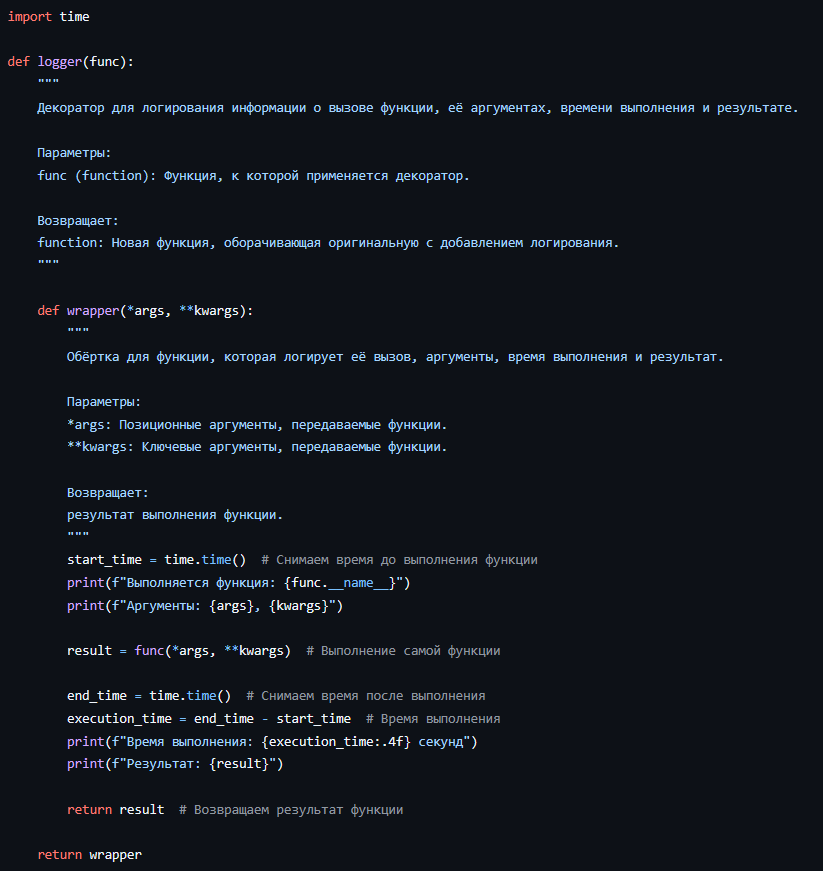
**Задача №1 декоратор без параметра**

Напишем функцию декоратора для логирования аргументов. Для этого нам понадобится обёртка функции, которая и будет логировать её.



Внутри функции реализуем вывод имени функции, аргументы функции, результат и время выполнения, как указано в условии. Для подсчёта времени работы функции будет использовать модуль time



После чего вернём результат функции, а потом уже и нашу обёртку. Целостный код выглядит следующим образом:  


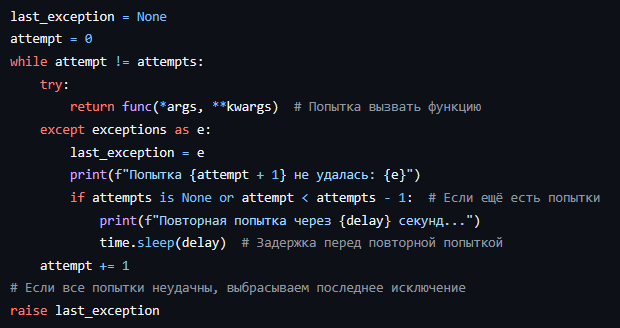
**Задача №2 декоратор с параметром**

Нам предстоит реализовать декоратор, который будет повторять выполнение функции какое-то количество раз, переданное нам в параметре декоратора.

Для начала заполним стандартные значения параметров, в случае, они не будут указаны при вызове декоратора 

В случае, если не был передан параметр attempts, то считаем количество попыток неограниченным, параметр delay по умолчанию одна секунда. Если же не был передан список исключений, то будем использовать список всех исключений python

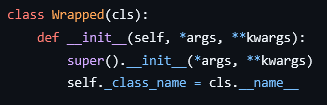


Далее напишем обёртку. Будем придерживаться следующей логики: заведем переменную, отвечающую за то, сколько попыток мы совершили. Будем выполнять цикл while, пока количество совершенных попыток не станет равно количеству данных функции. В случае, если количество попыток неограниченно, то количество совершённых попыток никогда не будет равно None, значит будет выполняться, пока не будет достигнут необходимый результат. Затем будем с помощью try-catch исполнять функцию. В случае успешного выполнения просто вернём результат наверх. Если же выполнение не было успешным, будем ловить ошибку из предложенного нам списка и выдавать её пользователю. Дополнительно сообщим номер попытки. Если же исключение не находится в списке, то программа завершит выполнение. Если мы достигнем лимита попыток, то вернём пользователю последнее исключение. В конце вернём обёртку  


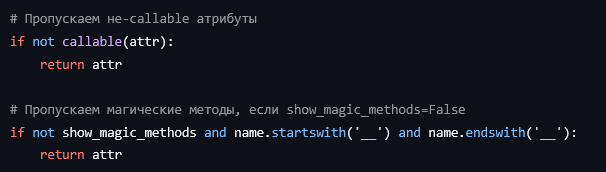
**Задача №3 декоратор класса без параметра**

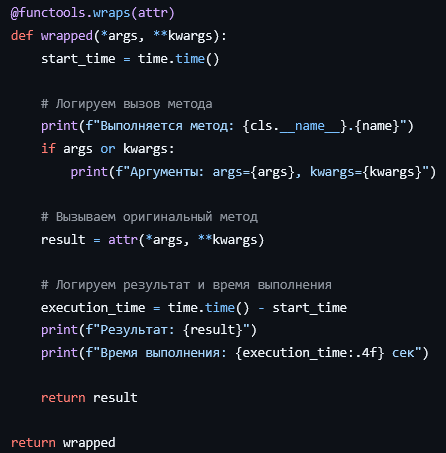
В данной задаче нам необходимо сделать декоратор для класса, который будет выполнять примерно ту же функцию, что и декоратор функции из задания №1, но с методами данного класса. Дополнительно в случае запроса пользователя нужно будет логировать магические методы.

Начнём с того, что создадим класс обёртки для нашего класса и проинициализируем его



Теперь переопределим метод \_\_getattribute\_\_, который в случае вызова метода будет обращаться к родительскому классу, доставать метод из него и логировать. Если метод невызываемый, то логировать его не будем. Если метод магический и логировать его не надо, то тоже просто вернём его.

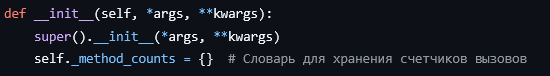


После этого напишем идентичную 1 заданию обёртку 

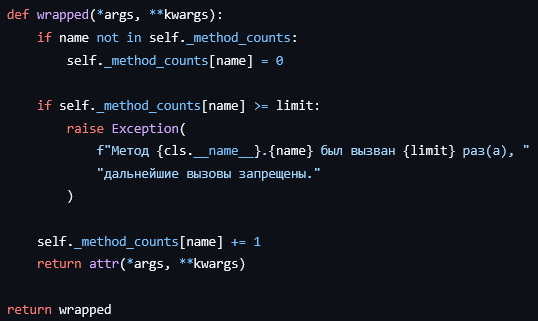
В конце вернём наш обёрнутый класс.

**Задача №4 декоратор класса с параметром**

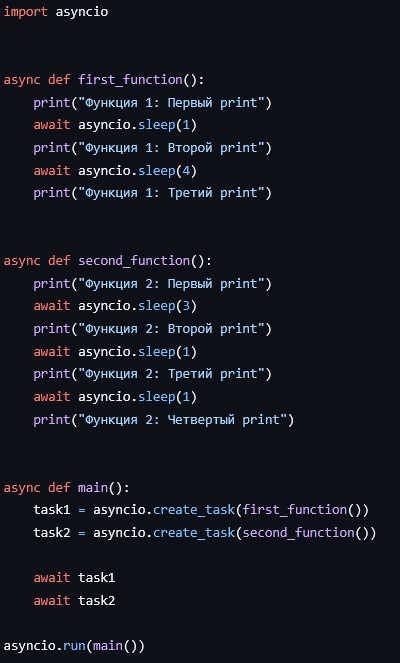
Теперь для каждого метода нужно будет ограничивать количество вызовов. Для этого воспользуемся кодом из задачи №3. При инициализации добавим нашему классу обёртки словарь, который по имени метода будет хранить количество обращений к методу



В обёртке будем сравнивать количество обращений с лимитом. Если мы его превышаем, то сообщим об этом пользователю, иначе выполним метод



**Задача №5** **введение в асинхронность**

Выполним буквально условие задачи 

При запуске программы можно заметить, что методы и правда выполняются параллельно, несмотря на задержку друг друга, и завершаются одновременно. Это наглядно показывает нам, что потоки выполняются полностью изолированно и параллельно.

**Вывод**

В результате выполнения данной лабораторной работы я научился писать собственные декораторы для функций, классов и методов, с параметрами и без. Это полезный опыт, так как он поможет мне отлаживать свой код эффективнее, чем добавление промежуточных выводов. Вдобавок задача №5 наглядно показала, как работают функции в асинхронном потоке и что разделение задач на потоки позволяет эффективнее решать одни и те же задачи.