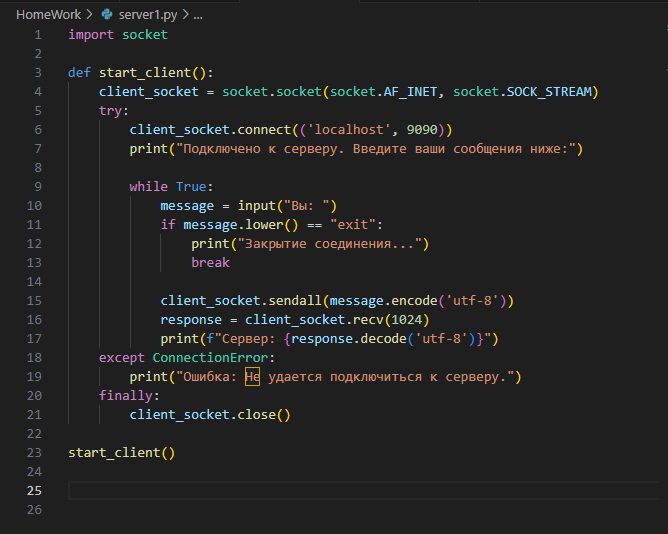
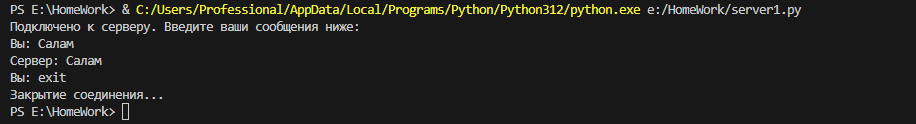
Сергушов Павел ПМ22-4

Лабораторная работа 7

**Упражнение 1. Основы работы с сокетами в Python**







1. **Чтение сообщений от пользователя в консоли и их отправка на сервер:**
   * Реализован цикл для считывания сообщений с консоли с помощью **input()**.
   * Каждое сообщение кодируется в байты с помощью **.encode('utf-8')** и отправляется на сервер через **client\_socket.sendall()**.
2. **Завершение работы клиента по команде "exit":**
   * Внутри цикла проверяется, если пользователь вводит "exit", программа выводит сообщение о закрытии соединения и прерывает цикл.
   * Соединение с сервером корректно закрывается с помощью **client\_socket.close()**.
3. **Обработка исключений на сервере при неожиданном разрыве соединения клиентом:**
   * Внутри цикла приема данных от клиента используется блок **try-except**.
   * Обрабатывается исключение **ConnectionResetError**, возникающее при неожиданном разрыве соединения клиентом. В этом случае соединение с клиентом закрывается, и сервер продолжает ожидать новых подключений.
4. **Обработка исключений в клиентском приложении при сетевых ошибках:**
   * Используется блок **try-except** для обработки исключений **ConnectionError**, которые могут возникнуть при попытке подключения к серверу.
   * В случае ошибки выводится сообщение, и соединение корректно закрывается.
5. **Расширение функционала сервера:**
   * + 1. **Логирование подключений и сообщений:**
     + Настроено логирование с использованием модуля **logging**, лог-файлы сохраняются в **server.log**.
     + Логируются время подключения, адрес клиента и полученные от клиента сообщения.
       1. **Обработка команды shutdown для завершения работы сервера:**
     + Добавлена обработка команды "shutdown". Если сервер получает эту команду от клиента, он отправляет клиенту сообщение о завершении работы, логирует событие и инициирует завершение работы сервера, вызывая исключение **KeyboardInterrupt**.
     + Перед завершением работы сервер корректно закрывает все активные соединения.

**Упражнение 2. Многопоточное обслуживание клиентов**



GIL — это механизм в интерпретаторе CPython, который ограничивает выполнение Python-кода одним потоком одновременно. Это значит, что даже на системах с несколькими процессорами многопоточные программы на Python не могут использовать все ядра процессора параллельно для выполнения Python-кода.

**Влияние GIL на многопоточность**

* **Ограниченная параллельность:** Из-за GIL в любой момент времени только один поток может выполнять Python-код, что ограничивает параллельное выполнение задач, требующих интенсивного использования процессора (CPU-bound задач).
* **Потоки ввода-вывода (I/O-bound):** GIL освобождается во время операций ввода-вывода (например, чтение из сети или файла), что позволяет другим потокам продолжать работу. Это делает многопоточность полезной для задач, связанных с вводом-выводом.

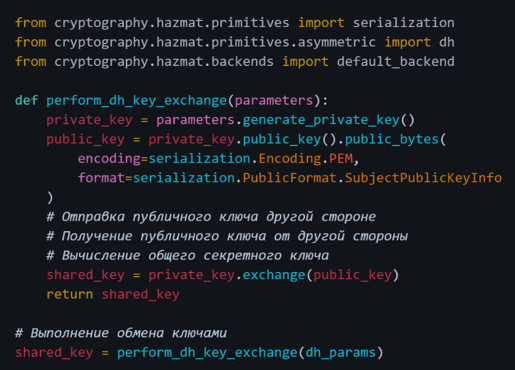
**Почему многопоточность полезна в сетевом программировании**

* **Обработка множества подключений:** В сетевом программировании задачи часто связаны с вводом-выводом, такие как чтение и запись в сеть. Многопоточность позволяет серверу обрабатывать несколько подключений одновременно, так как GIL освобождается во время операций ввода-вывода.
* **Простота реализации:** Многопоточность упрощает создание серверов, обрабатывающих множество клиентов, благодаря изоляции кода для каждого подключения в отдельном потоке.
* **Сложность переключения контекста:** Хотя GIL ограничивает параллелизм, переключение контекста между потоками при выполнении операций ввода-вывода менее затратное по сравнению с переключением между процессами.

**Упражнение 3. Реализация протокола Диффи-Хеллмана и асимметричное шифрование**

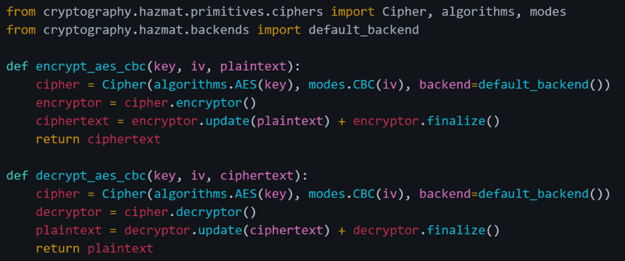


**Реализация обмена ключами и проверки успешности обмена**:



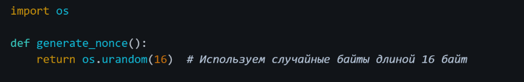
**Изучение и реализация различных режимов работы AES**

Для изучения и реализации различных режимов работы AES можно использовать библиотеку **cryptography**. Рассмотрим пример для режима CBC:



**Механизм защиты от атак с повторным воспроизведением сообщений**:

Для защиты от атак с повторным воспроизведением сообщений можно использовать nonce-значения. Например:



**Создание набора автоматических тестов:**

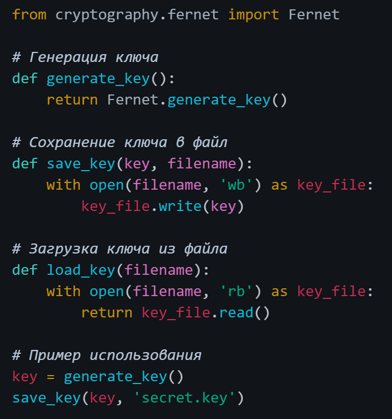
Для написания автоматических тестов можно использовать библиотеку **unittest**.

**Интеграция в существующее приложение:**

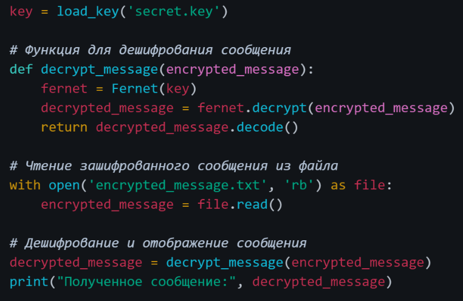
Чтобы интегрировать протокол Диффи-Хеллмана и шифрование AES в ваше приложение для обмена сообщениями, реализуйте необходимые функции и классы, опираясь на приведенные примеры. Это позволит безопасно передавать сообщения между клиентом и сервером с использованием данных протоколов и методов шифрования.

**Упражнение 4. Симметричное шифрование сообщений**

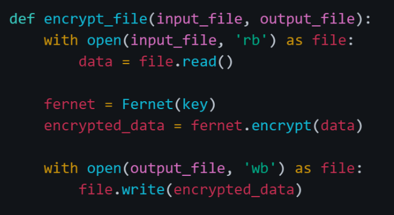
Генерация и сохранение секретного ключа и отправителя сообщений:



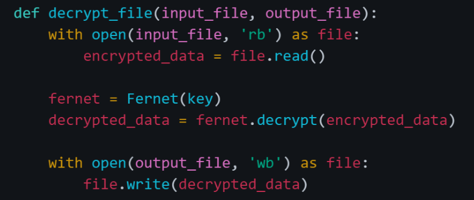
Скрипт получателя сообщений:



Функция для шифрования файла:



Функция для дешифрования файла:



**Упражнение 5. Работа с UDP-сокетами**

Модификация UDP-клиента для поддержки интерактивной отправки сообщений:



Модификация UDP-сервера для обработки специальной команды и корректного завершения:



Теперь UDP-клиент и сервер поддерживают интерактивный режим работы, позволяющий отправлять сообщения многократно и завершать сеанс по команде "exit".



Теперь сервер обрабатывает команду shutdown, обрабатывает исключения при разрыве соединения с клиентом и предоставляет логирование для отслеживания событий.